Tagging

Outlining the Task

The task was, given few hundreds emails, make a program that finds and tags some particular information from seminar announcement emails, such as paragraphs, sentences, speaker, location, stime, etime.

Paragraphs

To find and tag all the paragraphs, I've used regular expression.

```
Regex = r’(?<=\n\n)(?:(?:\s*\b.+\b:(?:.|\s)+?)|(\s{0,4}[A-Za-z0-9](?:.|\n)+?\s*))(?=\n\n)’
```

Using this pattern, I am searching for any bunch of text which is surrounded by 2 new lines before and after it. This pattern also checks if we don't have any space characters at all before the first word or we have exactly 4 space characters before the first word. If this bunch of text starts with any other amount of space characters, then we don't tag it as a paragraph.

Results after evaluation: Precision: 0.7249, Recall: 0.6796, F Measure: 0.6651

Sentences

To find and tag the sentences, I am assuming that my paragraph tagger works correctly, so I am only checking the text, which is surrounded by paragraph tags. I am using two regular expressions to do it:

```
Regex_1 = r'(?<=<paragraph>)([\d\D]*?)(?=\<\/paragraph>)'
```

The regex above searches for any text, which is surrounded by paragraph tags. After we found all these paragraphs, we iterate through each of them and apply the following regular expression for each paragraph that we found:

```
Regex_2 = r'(([A-Z][^\.!?]*)(?=[\.!?]))'
```

This regular expression search for some text, which starts with capital alphabetical letter and ends with dot, exclamation mark or question mark. After that I also apply if statement that checks if this sentence actually is a normal sentence, not for example initials that the person can have (ex: A.Lebedjko) and which also start with a capital letter and end with dot.

Results after evaluation: Precision: 0.754, Recall: 0.6748, F Measure: 0.6978

Etime

To find the time at what each seminar will finish, I also have used regular expression.

Regex = '([ ]{0,1})([-]|(?:-|(?<=until)))([ ]{0,1})[0-9]{1,2}?\:[0-9][0-9][ ]{0,1}[am|AM|pm|PM |a.m|A.M|p.m|P.M|\n|\s]{0,3}

This regular expressions searches for the symbol like "-" or for the word "until" and checks what goes after that. If after that we see something in a time format, we tag it as an etime.

Results after evaluation: Precision: 0.8775, Recall: 0.8824, F Measure: 0.876

Stime

To find at what time each seminar will start, we also use regular expression.

Regex=r'((?<!<etime>)(\b([0-9]{1,2}(?::[0-9]{2}\s?(?:AM|PM|am|pm|a\.m|p\.m)|:[0-9]{2}|\s?(?:AM|PM|am|pm|a\.m|p\.m)))\b)(?!\<\/etime>))'

This regular expressions tags everything what is in a time format and what is not already tagged as etime.

Results after evaluation: Precision: 0.7642, Recall: 0.9787, F Measure: 0.8337

Speakers

In a majority of emails we have a field "Who:" from which we can extract the speakers. There is another regular expression below:

Regex = r'(?<=who:)(.*)(?=\n)'

This regular expression searches for a field "Who:". If it found it, it takes the whole line that goes after that field. Sometimes, in this line with "Who:" field we can also have a lot of extra information apart from a speaker name. To check this information and to be sure that we tag only a speaker, not some irrelevant words, we do the following:
If after "Who:" we have only two words, that it's probably a name and a surname, so we can tag it as a speaker. But in case if there are more than 2 words, we should do some further checks. We check if there are some titles like Dr. or Professor. If there are, we check if the next word after that title is a real name. To do it, we check if this word is in first names database (names-female.txt and names-male.txt). After that there can be a case that this person will have middle name, for example: Dr. David L. House. In this case we check if a next name after first name has length of one character and has a dot in the end. If it is a case we tag title, first name, middle name and last word which is supposed to be a surname as a speaker. If, this case doesn't happen and there is no middle name, we just assume, that instead a middle name, this word is a surname, so we get Dr. or Professor plus first name, plus surname. If there is no title such as Dr. or Professor, we just apply the same logic described above, but without titles.

Unfortunately, not al of the emails have a field “Who:”. If it is a case, my program searches for any titles such as Dr. or Professor and extracts a full name after that using regular expression.

Regex='Professor\s\w+\s\w+\.\s\w+|Professor\s\w+\s\w+|Dr\.\s\w+\s\w+\.\s\w+|Dr\.\s\w+\s\w+|Professor\sAssistant\s\w+\s\w+'

It is very similar logic to what we do if there is a field “Who:”, but now instead of using some if statements, we are using a regular expression to experiment a bit.

Results after evaluation: Precision: 0.4535, Recall: 0.4079, F Measure: 0.4213

Locations

To find and tag the locations, I am again using regular expressions. The first one is:

Regex_1 = r'(?<=place:)(.*)(?=\n)', flags=re.IGNORECASE

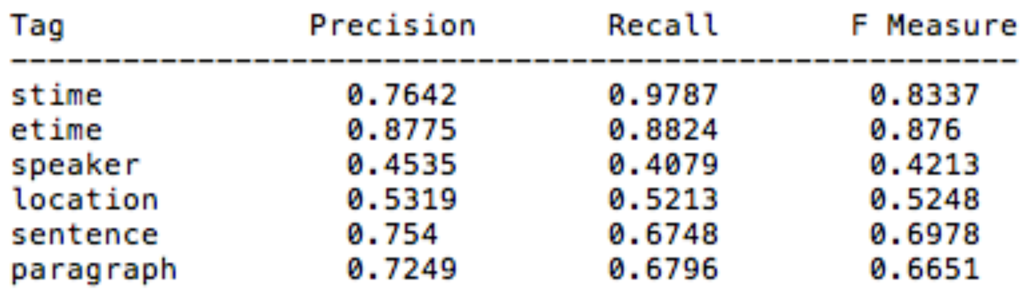
This regular expression searches for a field “Place: “ and tag everything what is in that line after that. If there is no field “Place: ” in an email, we apply the second regular expression. The second regular expression is exactly the same, with the only difference that it searches for a field “Where: “ instead of field “Place: ”.

Regex_2 = r'(?<=where:)(.*)(?=\n)', flags=re.IGNORECASE

If there is a case, when there are no fields “Place: “ and “Where: “, I have also tried to experiment and search for some phrases like “event will happen in the…” and after that apply POS tagger to extract all the nouns that are supposed to be a location, but this approach was giving very inaccurate results, so I’ve decided to not include this technique in my system.

Results after evaluation: Precision: 0.5319, Recall: 0.5213, F Measure: 0.5248

Summary table of all the results for each tagger



```
Tag               Precision        Recall        F Measure
-----------------------------------------------------------
stime               0.7642         0.9787         0.8337
etime               0.8775         0.8824         0.876
speaker             0.4535         0.4079         0.4213
location            0.5319         0.5213         0.5248
sentence            0.754          0.6748         0.6978
paragraph           0.7249         0.6796         0.6651
```

Ontology Construction

Outlining the Task

The task was to automatically the seminar announcements emails according to an ontology.

My approach

To do the ontology construction, I started with making an ontology tree by my hands. It has the following structure:

```
Ontology_tree ->
                Science & Enginnering School ->
                                Computer Science
                                Biology
                                Chemistry
                                Electronics
                                Physics
                Arts School ->
                                Politics
                                Languages
                                Performing Arts
                Business School ->
                                Finance
                Other
```

I filled all the leaves of the ontology tree (except Other) with some particular words that belong to each department. To classify the emails and to output to which department at university each email belongs, I made by program to use the following logic:
First of all, my program reads the email which was tagged using my tagging system. Once we read the email, because we have some information tagged, we can use this tags to delete some text from an email, that doesn't carry any useful information in terms of ontology, for example stime, etime, and speakers. We delete all these things, after that we delete all the tags from an email as well. We tokenize all the words which are left. We remove stop words. Remove all the words that have only 1 or 2 characters. Now we lemmatize all the words. After we cleaned the email and lemmatized all the words, we return 25 most popular words from an email. For each of these popular words, we are using wordnet, to get the synsets(set of synonyms) of these words. We append all these synsets to a list with popular words. After that, we just are comparing all these words with the words from each ontology tree leaf. We calculate all the matches and in the end we return the department with greatest number of matches. If you want to see a results, you can go to ontology folder and see a file called ontology_results.txt. It predicted that the majority of the emails belong to the Computer Science department, which is definitely correct. The problem with this

approach which I have decided to use, is that wordnet will not give you a lot of good synsets for some particular technical words.