Information Extraction of Seminars – Report

Entity Tagging

The main class used for entity tagging is seminars.py. getFileToTag is responsible for reading in the corpus and tagging sentences and paragraphs. It returns a list of tokens, which is passed through to any other class that requires them via the getTokens() method. These tokens, are then part of speech tagged. My POS tagger is created in the posTagger.py class. Here, I train my tagger using the brown corpus, and then use a bigram tagger, with a unigram back off, to tag each token of the corpus. I save and load the pos tagger between the classes using pickle. Using the pos tagging, I identify noun phrases (NP), and check if the NP is a name using nameCheck(), allowing me to tag speakers. Similarly, I identify VB's and if it is equal to 'place', I tag locations, using the foundVB() method. The new data is all returned, mainly from the Named Entity Recognition class (ner.py) back into Seminars.py, where it is reassembled as a new document, and outputted to the file location "my_seminars_tagged/file.txt." The files are created and written to in getFileToTag.py.
The dataFromTraining.py class extracts speakers and locations from the training data using regex and places it under the res file in the appropriate text file. These files are used when tagging speakers and locations as they are previously seen data that we know to be correct.

Tagging Sentences and Paragraphs

This occurs in getFileToTag.py via the tagSentsAndParas() method. Using regex and the english.pickle sentence tokenizer, I split the corpus into two parts: header and abstract. This allows me to ensure sentences and paragraphs are only tagged in the body of the email. I also filter out any lines that include colons (e.g. Where: Baker Hall), as these are not tagged as sentences in the training data. I then split the text, now tagged with sentences, by "\n\n" to tag the paragraphs. I then re-join the text, and tokenize it to remove escape characters.

Tagging Times

In the seminars.py class, I use the regex: '([0-9]+(\s|:|pm|am){1}[0-9]*$)' in order to identify times . If a match is found the tagTime() method in the ner class is called so that the time can be correctly tagged as a start or end time. If the word before the passed in time value is a dash (-), we know that it much be an end time, otherwise it is a start time. Unfortunately, the method and regex I use to tag time does not take situations such as 16:09-17:05 into consideration, where there are no spaces between the respective times and the dash (e.g. 16:09 – 17:05)

Tagging Speakers

Speaker tagging occurs when a noun phrase is identified. In the namecheck() method, I check surrounding words, to see if they are also names. (Despite the fact some names will be classed as NP many will also have a classification of None due to their non-existence in the brown corpus). I have filtered out any people that are known hosts, days of the weeks and months. To check whether a given word could be name I am using: the three given gazetteers of names; titles (e.g. Dr and Mr); the text file (res/speakers.txt) that contains all speakers from the original training data; and Wikipedia. My wikification methods are in the wikification.py file. They work by searching for the word 'born' amongst the top Wikipedia entries word. If there is an entry that contains the word born, the word may be being used as a name. This also helps to filter out cities, countries and company names.

Tagging Locations

Tagging locations occur when the words 'place', 'location' or 'where' are found in the corpus. Tagging locations works in a similar way to tagging speakers, only the method used in ner is called tagLocation(). In tagLocation(), I check for: numerical values (as a lot of rooms have numbers e.g. 4069 WeH); the res/location.txt file which is made up of data from the training data; and the two keywords 'room' and 'hall.' I do not use wikification for locations as many of the locations in the data are not known by Wikipedia. I planned to implement tagging of locations in other places in the document, however did not get around to it. The method I would have taken to do this would have been as follows: Find all occurrences of the identified location from the place tag(etc.) and substitute it with the tagged version of the location.

Evaluation

All measures are given to 4 significant figures. Due to time constraints, I was not able to calculate accuracy for sentences and paragraphs.

301.txt – 484.txt (184)		301.txt – 392.txt (first 92)		393.txt – 484.txt (last 92)	
Speaker Accuracy:	11.59%	Speaker Accuracy:	9.196%	Speaker Accuracy:	13.98%
Location Accuracy:	46.87%	Location Accuracy:	57.01%	Location Accuracy:	36.72%
Start Time Accuracy:	15.55%	Start Time Accuracy:	14.76%	Start Time Accuracy:	16.33%
End Time Accuracy:	60.87%	End Time Accuracy:	45.65%	End Time Accuracy:	76.09%
Overall Accuracy:	13.13%	Overall Accuracy:	12.56%	Overall Accuracy:	13.70%
Overall Precision:	17.90%	Overall Precision:	17.07%	Overall Precision:	18.72%
Overall Recall:	30.22%	Overall Recall:	28.44%	Overall Recall:	32.00%
Overall F measure:	21.37%	Overall F measure:	20.47%	Overall F measure:	22.28%

These statistics are all the results of running the evaluate.py file. When calculating my values, I have ignored True Negatives (TN) as they are difficult to work out based on the provided data. The overall accuracy of my system is lower than I expected, however I have highlighted ways in which I feel this accuracy could be improved below.

My most accurate tagging was End Time tagging, however the variation between the first half and second half raises a few concerns to me. Through my analysis of some of the text files in the second half I found that many of them do not actually have any end time tags, which explains why accuracy would be so high.

Speaker tagging is quite low, however when comparing the two respective files, I found that the major reason for this is that a lot of my tagging was incomplete. For example, I would tag Don Fullerton but the full speaker is Professor Don Fullerton. This could be solved in two ways: taking partial correctness into consideration or by improving the tagging of speakers.

The overall precision of my system is higher than accuracy. The precision of the system highlights that there are few false positives in the tagged data. In order to reduce FP's I would filter speakers and locations against more data. To do this I would make more use of wikification and I would tag elements based on the frequency of their occurrences. The recall of my system is quite high in comparison to other measures, highlighting the fact that I have quite a good ratio of true positives against true positives and true negatives.

Ontology Construction

The ontology part of this assignment is in the ontology.py file. I started off by creating a tree that contains the various subjects I had extracted by looking through the text files. For each subject, I assigned a list of key words to them to help with categorising the emails. I included a miscellaneous category as I knew that there would be some emails that my system would fail to categorise. The ontology works when the method main() is run from command line. I will briefly outline how the emails get categorised below.

The text file is opened using openTextFile, and anything following the 'Type:' and 'Topic' segments of the email using regex. I split the sentence that comes directly after topic into words, and categorise each word. To do this, I passed it into the check() method. Here the word is checked against the list of keywords for each branch respectively, returning the appropriate string. This is also done for type. These tags are all accumulated into a list of tuples of the following format: (category, word). This list is then passed through to the analyseTags() method.

Here, the number of occurrences for each category is counted and stored in a dictionary. If the highest category has a count greater than two, the email is classified with that category.
If the highest category has count of one, we pass it through to the wordNet() method.
If there are 0 occurrences for everything we simple return 'Misc' as we cannot effectively classify the email based on the given data set.

I use word net to calculate the distance between each word in the list and the category of choice. I have filtered out some words – in the avoid list – as they would not contribute positively to the outcome due to their neutrality. The similarity of each word to the category is added to the variable 'totalSum,' which I use to evaluate whether the email should be categorised or not. I chose my minimum sum value to be 0.5, as that means that for an average of a five-word description each one will have a similarity of around 0.1.

Finally, I add the text file to the appropriate category according to the tree and once all text files have been classified, I print a list of each text file in each category.

Ontology Evaluation

I analysed a few of the categorised text files manually and these were my findings:
450.txt was classified as AI and this was a correct classification. There were however many other text files that should have been classified under this category (an example being 308.txt which has been categorised under robotics instead).
Most of emails were classified as miscellaneous, indicating that the system was not good enough to effectively classify them.
The second biggest category was robotics. Many of these are actually robotics related, however a fair bulk of them should have been in various others (e.g. 354.txt should be under Graphics)

Future Improvements

I have included various improvements I would make to each section of the system above, however will include a summary in this section for clarity.

- I found myself spending more time working on entity tagging then ontology construction. If I were to re-do this assignment I'd ensure I split my time more equally.
- The accuracy for many parts of entity tagger was poorer than I expected them to be. In the future, I would take a different approach to tagging speakers and start times. With speakers, I would only tag the person who has the greatest number of occurrences with in the corpus. I am unsure as to how I'd improve the tagging of start times.
- I would improve my wikification algorithm and make better use of it during the speaker and location tagging.
- I would tag locations outside of the 'where:' section of the email as explained above. This is important as many emails do not actually have a where/place/location tag.
- I would improve my ontology by using word2vec in addition to wordNet. I feel that using two distance measuring tools would help to classify data more accurately as well as allowing me to classify more emails as the dataset used by both tools are different. I would also try to use Naïve Bayes Classification for a similar purpose.
- If I had more time. I would have used more data than simply topic and type to help me classify the emails. This would not only improve accuracy but help to classify more emails.
- I would opt to automate my system. Due to the way I initially built this system it was very difficult to change it to become automated. – Currently you have to type each file to be tagged one by one.