Information Extraction of Seminars – Report

Entity Tagging

The main class used for entity tagging is ‘seminars.py’. In order to get the file that we want to tag, it calls the ‘getFileToTag.py’ class. getFileToTag is responsible for reading in the corpus of the file 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 email corpus. I save and load the pos tagger between the classes using pickle. Using the pos tagging, I identify noun phrases (NP), and check to see if they are names using nameCheck(), allowing me to tag speakers. Similarly, I identify VB’s and if it is equal to ‘place’, I tag locations. It also calls the foundVB() method when the words ‘location’ or ‘where’ are found. The new data is all returned back, 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 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: Martin Luther King Room), as these are not tagged as sentences in the training data. I then split the text, now tagged with sentences, by “\n\n” to identify and tag the paragraphs. I then re-join the text together, 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 in the tokens. 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.

Tagging Speakers

As mentioned above, 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). Because of this I am checking for None types too when considering speakers. 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 gazetteers of names given on canvas; titles (e.g. Dr and Mr); the text file (res/speakers.txt) that contains all speakers from the original training data; and Wikipedia through wikification. My wikification methods are in the wikification.py file. It works by identifying the top data hits for the word, and searches for the word ‘born’ via regex in this data set. If there is an entry that contains the word born, the word may be used as a name. This 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. I initially only tagged according to ‘place’ and so the method used is called foundVB() (as in POS place is classified as VB). 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.

Evaluation  
All measures are given to 4 significant figures.

All 184 Text Files

Speaker Accuracy: 11.59%

Location Accuracy: 46.87%

Start Time Accuracy: 15.55%

End Time Accuracy: 60.87%

Sentence Accuracy:

Paragraph Accuracy:

Overall Accuracy: 13.13%

Overall Precision: 17.90%

Overall Recall: 30.22%

Overall F measure: 0.0%

First 92 (301-392) Text Files

Speaker Accuracy: 9.196%

Location Accuracy: 57.01%

Start Time Accuracy: 14.76%

End Time Accuracy: 45.65%

Sentence Accuracy:

Paragraph Accuracy:

Overall Accuracy: 12.56%

Overall Precision: 17.07%

Overall Recall: 28.44%

Overall F measure: 0.0%

Last 92 (393-484) Text Files

Speaker Accuracy: 13.98%

Location Accuracy: 36.72%

Start Time Accuracy: 16.33%

End Time Accuracy: 76.09%

Sentence Accuracy:

Paragraph Accuracy:

Overall Accuracy: 13.70%

Overall Precision: 18.72%

Overall Recall: 32.00%

Overall F measure: 0.0%

Ontology Construction

Future Improvements

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 between the two tasks.

I would only tag speakers that have the greatest number of occurrences in the text

I would tag locations outside of the ‘where:’ section of the email.