Automatic Seminar Tagging

George Taylor

Semester 1 - 2016/17

Abstract

The aim of seminar tagging in this case is to take electronic seminar announcements and automatically identify features such as paragraphs, sentences and seminar details (speaker, location, start time and end time) and parse this into a tagged file. Pyhton 2.7 has been used for the tagger. The program `SeminarTagging.py` can be found at https://github.com/georgehtaylor1/NLP_Assignment.

Announcement Structure

The common structure of each announcement plays a huge role in their tagging. Every file consists of a header line enclosed in angled brackets, followed by a list of paramaters that are formatted as `paramater: value`, finishing with the abstract, or content of the announcement. This consistent structure allows the announcement to be split up into sections with ease.

Processing the Header

Using the fact that all announcements contain an "Abstract:" line, each line in the header can be processed by partitioning on the colon and storing the results in a dictionary. Additionally, by checking the parameter certian values can be determined:

- If the paramater is "Time" then the value can be processed to extract the start and end times.
- If the parameter is "Place" then the value can be tagged and stored as a location.
- If the parameter is "Who" then the speaker can be extracted from the speaker using the fact that the speaker is stored as the first part of the value, sometimes followed by a list of other details such as an address

This is done for every line that matches the format until the abstract is reached, if a line doesn't match the format then it is assumed to be part of the previous line so it is added to the previous entry in the dictionary.

Processing Times

Because times can be given in many different formats it is critical that the program be able to recognise a time, and also when two times are the same. To identify times the regular expression

```
\d{1,2}([:.,]\d{2})?(\s?([aApP]\.?[mM]\.?))?
```

is extremely successful. A similar expression

```
(\d{1,2})([:.,](\d{2}))?(\s?([aApP]\.?[mM]\.?))?
```

is used to extract the minutes and hours from a time given in any format that can be combined into a 4 digit 24 hour format. This means that two times can be compared for equality regardless of how they are formatted allowing the start and times stored from the header to be tagged anywhere in the abstract without any extra processing.

Processing the Abstract

When processing the abstract, the program will first train a sentence tokenizer provided by the nltk package Punkt (Bird et al., 2016). This is done by extractng sentences from the training data and passing them to the trainer which uses an unsupervised learning algorithm (Kiss and Strunk, 2006) to learn a tokenizer that can extract sentences. The program will first split the abstract into paragraphs by any occurence of '\n\n' representing a one line gap in the abstract, the paragraphs can then be enclosed by `<paragraph>` tags. For each of the paragraphs, the sentence tokenizer will then split the paragraph into constitiuent sentences that can be surrounded with `<sentence>` tags. For every identified sentence, any occurences of times matching the start or end times will be tagged accordingly and if a speaker wasn't identified in the header then the sentence will be part of speech tagged and chunked using the grammar:

```
SP: {<NNP><NNP><NNP>}
SP: {<NNP><NNP>}
```

in order to extract any names that it may contain. If a name is found then it is assumed to be a speaker and tagged accordingly.

Results

When run on untagged data, the program will produce a set of tagged files which from visual inspection are mostly accurate. The program will typically correctly tag all locations and times and most names provided they are not too obscure. With regards to sentence and pargraph tagging in the abstract, the program typically performs quite poorly. Because of inconsistencies and choices by the original tagger it is difficult to create a general solution to produce a tagging. For example, paragraphs are often closed before the last sentence is closed in the paragraph, which is difficult to predict programmatically. For this reason, when evaluating the success of the system, only tags for time, location and speaker are compared. If the program has correctly identified the times, location and speaker then it is considered a success, otherwise it is a failure.

When running the test script on the 184 tagged test files, there are 77 files (41.85%) that have been tagged completely correctly. The following table shows the success percentages when different tags are checked independently:

Tags	Success (%)
Start time	96.74
End time	92.39
Location	46.74
Speaker	26.63

From this table it is clear that while the times can be found with a very high accuracy, the task of identifying locations and speakers is severely detremental to the performance of the tagger.

Ontology Construction

The program `OntologyConstruction.py` can be found at https://github.com/georgehtaylor1/NLP_Assignment. The idea of ontology construction is to classify seminar announcements into groups, i.e. group announcements by subject. This is a particularly difficult task as a result there being no tagged data. Two attempts to solve this problem have been attempted:

K-Means

First, the program will attempt to split the announcements into k distinct groups using the k-means algorithm. The program first creates a full vocabulary of every noun in the data, this is then used to create vectors for every file where a 1 in the vector indicates that the word is present in the file. The k-means algorithm is then run on the set of vectors to try and train the mean vectors to classify the data. This was largely unsuccessful as it would tend to identify the whole set of data as a single cluster as a result of their being very little distinction between the clusters.

Hierarchical Clustering

A second attempt uses hierarchical clustering in an attempt to cluster the files. Again vectors are calculated for each file which are clustered using euclidian distance as a heuristic. Again, because of the relative lack of distinction between the clusters the hierarchical clustering would typically produce a tree that branched only in one direction.

References

Steven Bird, Ewan Klein, and Edward Loper. Willy and steven bird and edward loper and joel notham and arthur darcet, 2016. URL `http://www.nltk.org/_modules/nltk/tokenize/punkt.html`. Accessed: 02/12/2016.

Tibor Kiss and Jan Strunk. Willy and steven bird and edward loper and joel notham and arthur darcet, 2006. URL `http://www.mitpressjournals.org/doi/pdf/10.1162/coli.2006.32.4.485`. Accessed: 02/12/2016.