Pages 1–3
2018

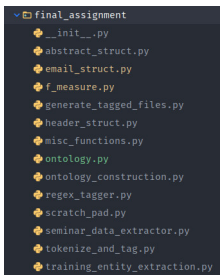
NLP: Data Extraction from Seminars

Sam Barrett (1803086)
MSci Computer Science w/ Industry Year
School of Computer Science
8/12/18

1 APPROACH

1.1 Project Structure

I took an Object oriented approach to solving the problem by first separating the header and abstract from each file and placing the untagged text as a variable within a relevant object structure.



1.2 Tagging Procedure (Section 1)

Having populated the Email Structures with the corresponding Header and Abstract structures populated with untagged text I call *email.tag_all()* on each. This procedure in turn analyses both the header and abstract structure. Analysis on the header starts first as it is a more structured and therefore, more reliable set of data. To analyse the header the text is scanned for certain 'tags' including lines which include "Time:" however, if these are not found, more generalised **regex** expressions are used producing data of various formats which are then parsed using hard coded logic (e.g start time has to be before end time).

In this section I have also used a ***Stanford tagger*** and ***Word tokeniser*** to extract speaker names as when passed with the result of word tokenising a string the Stanford tagger produces a list of word, type tuples which include tagging names fairly accurately based on a large corpus of names. In many of the tag extraction methods there is a form of hierarchy of methods, the most advanced being the default but progressively more simplistic / less accurate methods being used if these fail to produce any result.

I also extracted all locations and speakers from a secondary set of training data, which was used as a fall-back method for identifying speakers and locations in the text. This method scanned the header and abstract for instances of items found in the training set and ranked them on #occurences, the most common being selected as the most likely candidate for the respective tag.

Having extracted and parsed all the data found in the header, it is stored as attributes of the header structure (*self.location* etc.) for use when we go back and wrap occurrences with the appropriate HTML-style tag. Times and dates are also converted to a normalised form (%d-%m-%y and %H:%M) for storage which can be unpacked into the different possible notations via a separate function, this allows the tagger to correctly tag elements such as 1:00PM and 13:00 as the same attribute.

```
def __init__(self, untagged_header):
    self.speaker = None
    self.start_time = None
    self.end_time = None
    self.location = None
    self.date = None
    self.untagged_header = untagged_header
    self.tagged_header = None
    self.times = []
    self.dates = []
```

Once all available data has been extracted from the header and stored in attributes, the abstract is scanned for any attributes that are still empty (hold a *None* value). The same regex and Stanford tagger hierarchical object is used to scan the abstract for elements that are captured by the expression. Any data found during this process is stored in the relevant header attributes.

Now the data is stored in the proper formats & in the appropriate header attributes the tagging process starts. This process is called from the *email_struct* object as a sub-call of the *email.tag_all()* procedure. To tag the untagged plaint text header and abstract occurrences of the tag information, stored in the header, are searched for and wrapped appropriately. The normalised form of date and times stored are also unpacked into the different possible forms so all occurrences are matched. The tagged header and abstract are then appended and saved in a file of the same name as the original untagged version, only in a separate directory. There are many small sections of code that tailor the output of the tagging procedure to mimic that of the hand-tagged files, including the exclusion of lines containing strings such as "Location:" in the abstract. As without this they would be incorrectly tagged as sentences and paragraphs, reducing the average F_1 Score .

F_1 Score is then calculated by reading in these generated files along with the corresponding hand-tagged file and counting true positives in the generated and reference data as well as the total number of reference tags by comparing a list of all tags in these two documents. From these counts, Precision and Recall are calculated and F_1 Score from that. Average, modal, maximum and minimum F_1 Score is also calculated and all plotted on a scatter graph using the *matplotlib* python library.

$$Precision = \frac{\#TPClassified}{\#Classified} \qquad Recall = \frac{\#TPClassified}{\#TPInCorpus}$$

$$F_1 = 2\frac{PrecisionRecall}{Precision+Recall}$$

1.3 Ontology Construction (Section 2)

During the process of recognising and extracting tags from the header I also isolated the topic line (if present). Using this topic, a skeleton ontology containing keywords for each category and the ***word2vec*** python project I assessed the likelihood of a given topic falling into each predefined category. The file name was then appended to a list of files falling into the most likely category. The constructed ontology is saved after changes and this save file is loaded upon the instantiation of an Ontology object. My original ***word2vec*** model which assessed the similarity between topic keywords and words in the extracted topic was trained on the *Google news corpus* which is around 3GB in size, however ,in my experience, lacked technical vocabulary, resulting in many of the files being unclassified. In an attempt to improve this I trained a second

model on the *glove-twitter-200 corpus* of around 1GB in size which is compiled from 2B tweets, with a 1.2M token vocabulary.

2 RESULTS

2.1 F_1 Score of generated files

With all Tags:
F_1 Score : Mean = 0.60, Max = 0.88, Min = 0.14, Modal value = 0.67
Precision : Mean = 0.60, Max = 1.0, Min = 0.11, Modal Value = 0.5
Recall : Mean = 0.62, Max = 0.90, Min = 0.14, Modal Value = 0.5
(see Fig. 1.)

Without sentence & paragraph tags:
F_1 Score : Mean = 0.71, Max = 1.0, Min = 0.22, Modal value = 0.67
Precision : Mean = 0.82, Max = 1.0, Min = 0.33, Modal Value = 1.0
Recall : Mean = 0.64, Max = 1.0, Min = 0.17, Modal Value = 0.6
(see Fig. 2.)

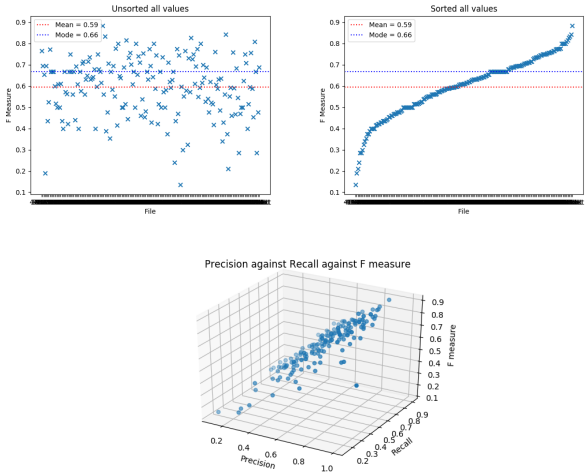


Fig. 1. F_1 Score over all tags

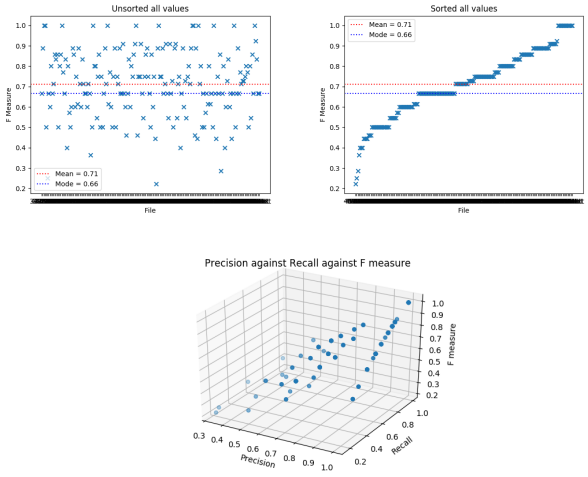


Fig. 2. F_1 Score without sentence & paragraph tags

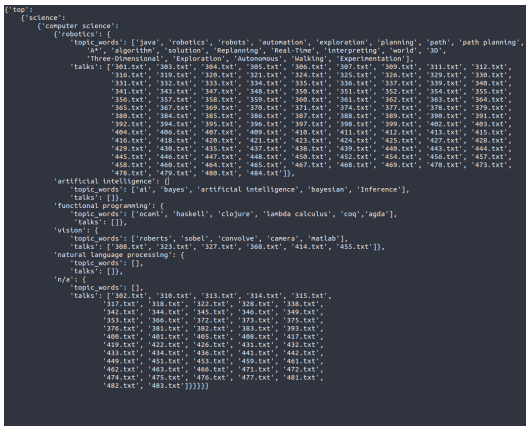


Fig. 3. Using *word2vec* model trained on **Google News corpus**

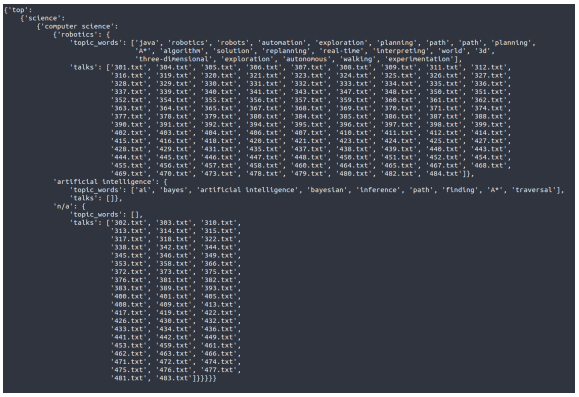


Fig. 4. Using *word2vec* model trained on **glove-twitter-200 corpus**

2.2 Produced Ontology

3 ANALYSIS

3.1 Tagging & F_1 Score

Clearly there is a broad range of values calculated for F_1 Score. This suggests that the consistency of my solution isn't as high as I would like. This could be due to many issues as when trying to capture an informal system within a formal system (human language within code) unless you hard code measures to account for a huge number of possible formats or exceptional circumstances there will be information your system misses or incorrectly classifies

However, I am happy with the mean and modal values for F_1 Score as they are between **0.6 and 0.7** I am particularly happy with the highest score achieved which was 0.88 i.e **0.88**, the file in question lacked any paragraph and sentence tags meaning a lot of my accuracy is lost over these categories.(See fig 2) where you can clearly see a much higher average F_1. score of **0.71** and a maximum score of **1.0**

Having analysed the results many of the incorrectly classified items were very close but due to formatting differences between the reference and generated set they are marked as incorrect. This would be a fairly simple fix but would require the extension or rewriting of the capture code at the first stage of tagging. Also, the implementation of the Stanford tagger for locations might aid the stability of the solution, especially over unseen data sets where locations are likely completely different to those seen in the training and test data sets.

3.2 Ontology Construction

I am less happy however, with the accuracy of my ontology construction. A large number of files are place into the *n/a* category meaning many of the files could not be correctly classified. I think this is in part due to the vocabulary *word2vec* is trained over (the Google news corpus or glove-twitter-200) as these will lack many technical words which are in the *topic_words* list of each category, meaning the *word2vec* struggles to classify many of the files leaving them as not classified or *n/a*.

Expanding both the vocabulary of the *word2vec* model and extending the *topic_words* of the ontology, including adding more commonly used words, would increase accuracy. Also a more diverse data set might allow for more extensive testing of the algorithm as currently, the vast majority of the exemplar seminar files should fall into the robotics category. Additionally, if there were seminars outside of the CS category the groups would be more distinct as currently the difference between, for example, Artificial Intelligence and Robotics is quite grey, making the process of differentiating between them less accurate.

4 CONCLUSION

To conclude, my solution to section 1 is far from perfect however, I feel a good attempt has been made and it would only take more hours and more analysis of the data-set to increase the F_1 Score of my tagger, or at least eliminate the lower quartile of by improving the capture rules and tagging procedure specifically surrounding sentence and paragraph tagging as these (as seen in Fig. 2.) are the categories most detrimental to the overall average score. The further use of taggers and tokenisers also might allow for my solution to be more robust as it would be more generalised allowing for more dissimilar data sets to be correctly tagged.

As for the ontology construction, there are many aspects that can be improved upon. As stated above, a more extensive technical vocabulary for the *word2vec* model along with a more populated selection of keywords in the skeleton ontology would result in more accurate classification. This alongside a more diverse data set to classify would likely result in a more accurate result due to more distinct boundaries between the different categories.