Report: Information Extraction of Seminars

Andreea Bîlă

December 10, 2018

TAGGING

For the tagging, I used primarily a Regular Expressions approach. I explored the possibility of using various NER and POS taggers, but I found that these were much more time exhausting with only little more benefits provided. I attempted to use the Stanford NER classifier to identify the names of speakers, but the execution time was, in my opinion, too long.

- **`readContents()`:** In this method, I read all the content from the files and store it in multiple dictionary trees and I populate the `speakers` and `locations` sets.
  - `mapfiles` for every file it maps its name to all the content of the file.
  - `mapHeaders` for every file it maps its name to the header of the file.
  - `mapContent` for every file it maps its name to the content excluding the header of the file.
  - `speakers` holds all the speakers found in the headers of the emails using the `findHeaderSpeaker()` method.
  - `locations` holds all the locations found in the headers of the emails using the `findHeaderSpeaker()` method.
- **`extractTrainingData()`:** This method reads all the training files from the `training/` folder, finds all the tagged data and stores the speakers in the speakers set and the locations in the locations set. This method is not used in the final program, as it proved non-beneficial to the accuracy, precision and recall evaluation.
- **`tag()`:** This method takes some text and it replaces it with its tag wrapped version in the email content. The parameters are: the index where the text starts, the text itself, the name of the file and the name of the tag. The procedure is: in `mapFiles[fileName]`, replace the content with everything up to the index of the initial text, followed by the newly tagged text, followed by everything after the index of the initial text plus the length of the initial text.
- **`hasVerb()`:** This method takes some texts as parameter and return `True` if the text contains a verb and `False` if not.
- **`isSentence()`:** This method takes some text as parameter and checks if it is a valid sentence by checking if it has a verb using the `hasVerb()` method and if it starts with particular characters or words that signify it should not be counted as a sentence.
- **`tagSentences()`:** This method takes some text, splits it into sentences using `nltk sent_tokenize()` and checks if every one of those sentences is a sentence using the `isSentence()` method. If the sentence passes the verification, it gets tagged using the `tag` method.
- **`tagParagraphs()`:** This method takes a file name as parameter and splits the content in smaller pieces at the occurrence of two consecutive newline characters. These pieces are then checked with the `isSentence()` method, since the unit measure of a paragraph is a sentence. If the check is passed, the text piece is tagged as a paragraph using the `tag` method and the `tagSentences()` method is called to tag the sentences within this paragraph
- **`findHeaderLocation()`:** This method searches for the keyword *'Place'* in the header of the file and returns the remainder of the line.
- **`tagLocation()`:** This method looks for occurrences of locations from the `locations` set in the file and tags them if found.
- **`findHeaderSpeaker()`:** This method searches for the keyword *'Who'* in the header of the file and returns the name after the keyword.
- **`tagSpeaker()`:** This method looks for occurrences of speakers from the `speakers` set in the file and tags them if found.

- **`tagTime():`** This method searches for the keyword *'Time'* in the header of the file and stores the time values following the keyword. In the content, it looks for expressions matching the regex for time. If these expressions coincide with the time from the header, they are tagged.
- **`printTaggedFiles():`** This method writes new files with the tagged content in the `output/` folder
- **`emptyOutputFolder()`** This method (found in the `utilities.py` file) empties the contents of the folder given as parameter.
- **`main:`** In the main method, first we empty the `output/` folder so new files can be written. The method `readContent()` is called to initialise the data fields. We then iterate over the files and tag all the components within. Finally the `printTaggedFiles()` method is called to create the new tagged files.

EVALUATOR

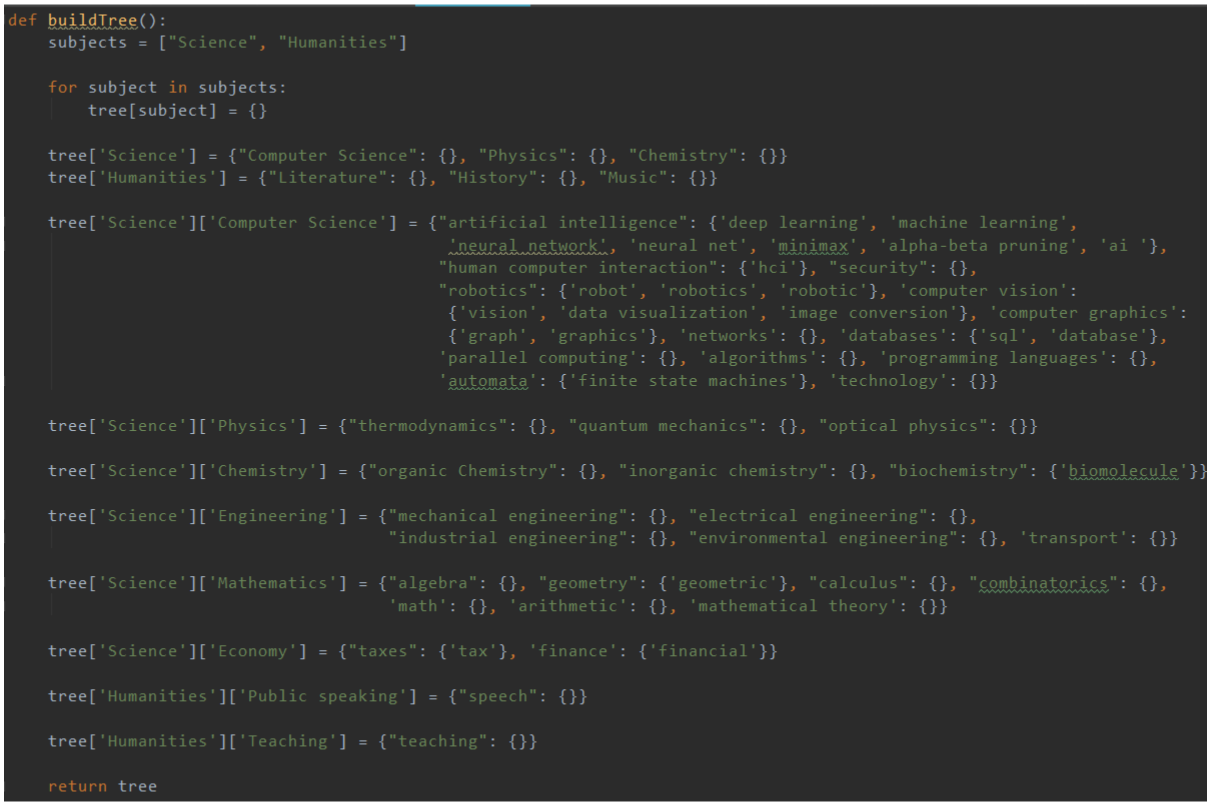
The evaluator is located in the `eval.py` file.
THE PROCEDURE: it identifies all tags in all the output files and creates lists of their occurrences. Using those lists, it counts the number of true positives, true negatives, false positives, and false negatives. For every tag content in the output file that coincided to that tag in the correctly tagged files, the true positives count was increased by one. For this evaluation, the true negatives were not taken into account (TN = 0 always). For every tag content that appeared in the output file but not the correctly tagged file, the false negatives count was increased by one. For every tag content that appeared in the correctly tagged file but not in the output file, the false positives count was increased by one. The total true positive, false positive and false negative counts are computed by adding together the counts for each type of tag (i.e. speaker, location) for each file. The accuracy, precision, recall and F1 measure for each tag and overall are computed using the formulas from the lecture slides. Finally, they are printed to the command line.

EVALUATOR				
Tag	Accuracy	Precision	Recall	F1 Measure
paragraph	22.33%	34.33%	38.98%	36.51%
sentence	51.94%	63.91%	73.5%	68.37%
stime	77.83%	78.23%	99.36%	87.54%
etime	77.84%	78.26%	99.31%	87.54%
speaker	46.82%	47.79%	95.86%	63.78%
location	67.52%	76.92%	84.68%	80.61%
TOTAL	61.23%	66.87%	87.9%	75.95%

Ontology

I used wikification and some other basic checks on the topics and contents of emails to classify them into fields from a manually built ontology tree.

- **`buildTree()`:** This method populates a tree with various subjects and corresponding keywords.
- **`getTopic()`:** This method searches for the keyword *'Topic'* in the header of the file and adds the remainder of the line after the keyword to the `topics` set.
- **`getType()`:** This method searches for the keyword *'Type'* in the header of the file and returns the remainder of the line after the keyword to the `types` set.
- **`ontology()`:** This method classifies the file given as parameter into a field from the ontology tree. Firstly, if the file is not already classified, I extract the topic of the email from the `mapTags` map and apply wikification to it, making sure to catch any timeout exception that the connection might produce. If any of the fields or keywords from the map appear in the result of the wikification, the email is classified accordingly to the appropriate field. If wikification fails to produce a classification for the file, I check whether any field or keyword appears in the topic itself. If that fails as well, I finally check the content of the email for any keywords and classify the email accordingly.
- **`printOntology()`:** This method writes the classification resulted from `ontology()` to the file `ontology-classification.txt`.
- **`main`:** In the main method, I iterate over the `mapFiles` map and call the `ontology()` method on each one. After ontology classification is completed, I call the `printOntology()` method to create the `ontology-classification.txt` file displaying the classification of the emails and their respective topics for clarity.



```python
def buildTree():
    subjects = ["Science", "Humanities"]

    for subject in subjects:
        tree[subject] = {}

    tree['Science'] = {"Computer Science": {}, "Physics": {}, "Chemistry": {}}
    tree['Humanities'] = {"Literature": {}, "History": {}, "Music": {}}

    tree['Science']['Computer Science'] = {"artificial intelligence": {'deep learning', 'machine learning',
                                            'neural network', 'neural net', 'minimax', 'alpha-beta pruning', 'ai '},
                                           "human computer interaction": {'hci'}, "security": {},
                                           "robotics": {'robot', 'robotics', 'robotic'}, 'computer vision':
                                            {'vision', 'data visualization', 'image conversion'}, 'computer graphics':
                                            {'graph', 'graphics'}, 'networks': {}, 'databases': {'sql', 'database'},
                                           'parallel computing': {}, 'algorithms': {}, 'programming languages': {},
                                           'automata': {'finite state machines'}, 'technology': {}}

    tree['Science']['Physics'] = {"thermodynamics": {}, "quantum mechanics": {}, "optical physics": {}}

    tree['Science']['Chemistry'] = {"organic Chemistry": {}, "inorganic chemistry": {}, "biochemistry": {'biomolecule'}}

    tree['Science']['Engineering'] = {"mechanical engineering": {}, "electrical engineering": {},
                                      "industrial engineering": {}, "environmental engineering": {}, 'transport': {}}

    tree['Science']['Mathematics'] = {"algebra": {}, "geometry": {'geometric'}, "calculus": {}, "combinatorics": {},
                                      'math': {}, 'arithmetic': {}, 'mathematical theory': {}}

    tree['Science']['Economy'] = {"taxes": {'tax'}, 'finance': {'financial'}}

    tree['Humanities']['Public speaking'] = {"speech": {}}

    tree['Humanities']['Teaching'] = {"teaching": {}}

    return tree
```

The Ontology Tree

**Decided to resubmit late as I came to realise I had deleted the call to the tagSentences() method and broken the evaluator with my final changes befoe the deadline.