Computational semantics
TP Word Sense Disambiguation

Timothée Bernard — timothee.bernard@u-paris.fr

April 22nd 2024

- The main goal of this TP is for you to implement and test the Simplified Lesk algorithm and a couple of WSD baselines.
- The `TWA-sensetagged` directory contains the files of a small dataset called 'TWA' (for 'Two-Way Ambiguities'). This dataset contains, for each of six ambiguous word forms (ex: *plant*, *crane*, *palm*), examples of use of this word form for two different senses (ex: *plant* in the factory sense, *plant* in the vegetal sense). There is one file for each of the ambiguous word forms, plus one file that is the concatenation of all of these (`all.test`).

1 First steps

- For this TP, we will use WordNet through the NLTK library. In Python REPL (the online Python interface in the terminal), execute '`import nltk`', then 'nltk.download('wordnet')' and '`nltk.download('omw-1.4')`'. You might need to install NLTK before doing this, with (in a terminal) '`pip3 install ntlk`'.
- Before doing anything else, make sure you know how to execute `wsd.py` with the path of one of the files of the dataset (in the `TWA-sensetagged` directory) as argument, and without getting any NLTK-related error.
- Open at least one of the corpus `.test` file, to see how it is structured.
- Study the code in the three Python files.
- What are the transformations applied as preprocessing to the texts when the WSD instances are loaded?
- Print the sense distribution in the whole corpus (modify `wsd.py`).

2 Baselines

- Evaluate the random baseline on the whole corpus.
- Implement the most frequent sense baseline and then evaluate it using different splits of the corpus into training and test sets.

3 Simplified Lesk

- Implement a version of Simplified Lesk that uses all tokens of the texts (instead of a window) and no IDF values. For the signature of a sense, use
  - the definition of each of the corresponding WordNet synsets (see `WN_CORRESPONDANCES`),
  - all of the corresponding examples in WordNet,
  - and the corresponding training instances.

  Once '`from nltk.corpus import wordnet`' has been executed, you can access a synset from its name using `wordnet.synset`. Example: '`wordnet.synset('plant.n.01')`'. Two useful *methods* of the Synset class are `definition` and `examples`. You can get more information about this class on the Internet or by executing '**help**(nltk.corpus.reader.wordnet.Synset)' (after having executed '`import nltk`'). Do not hesitate to try things on the REPL before or while you are implementing the algorithm in `wsd.py`.
- Add an integer `window_size` parameter to the constructor or the `train` method of `SimplifiedLesk`. When this parameter is different from `-1`, it should specify the size of the window of tokens taken into account by the algorithm.
- Add a boolean `use_idf` parameter to the constructor or the `train` method of `SimplifiedLesk`. When this parameter is set to `True`, IDF values should be used in addition to filtering out stop words.

4 Cross-validation

- When very little annotated data is available (as here) but one still needs to split it into training and test sets (resulting in a very small and likely biased test set), one usually uses *cross-validation*: The annotated data is randomly split a large number of times (say, 100) into training and test sets, and the performance is averaged over all splits.
- Modify the code so that all algorithms are evaluated by cross-validation.
- Study the impact of the number of training examples on the performance of the various algorithms.

5 Naive Bayes classifier

- Using a library or after having reimplemented the algorithm, test a WSD Naive Bayes classifier such that each feature is the presence of a word form (there is thus as many features as forms in the vocabulary and these features are binary ones).[1]

This is an assignment (only) for students repeating the year

- If you are repeating the year, you have until Sunday, April 28$^{\text{th}}$, to send me your code. You can work alone or in group.
- Code in standard Python files, or possibly in a Colab notebook, but do not use any other notebook system.

[1] Make sure you understand what a Naive Bayes classifier is before writing any line of code.

- Your grade will depend in part on your code and in part on a short oral examination. To schedule an appointment, contact me at my `@u-paris.fr` email address.
- The quality of your code will be taken into account. Your (scientific) methodology too.
- Double-check every piece of code that you write.
- Remember that it is usually a bad idea to try to implement advanced features before the making sure that basic ones work properly.