Computer-Assisted Language Comparison in Practice

Tutorials on Computational Approaches to the History and Diversity of Languages

How to Map Concepts with the PySem Library

Johann-Mattis List
Department of Linguistic and Cultural Evolution
Max Planck Institute for Evolutionary Anthropology

Mapping concepts to common concept identifiers across resources has become an important task for the aggregation of lexical data from different sources. With the Concepticon, this task has been facilitated due to a specific mapping algorithm by which a concept list can be automatically mapped to the concept sets in the Concepticon reference catalogue in order to be later manually refined. PySem offers an additional possibility to map concepts to the Concepticon, but in contrast to the algorithm used in the Concepticon workflow, the PySem approach can be accessed from within Python applications.

Introduction

When aggregating lexical data from different sources, it is important to find out which words denote the same concepts across different datasets. Since the typical datasets which we want to aggregate are provided in the form of wordlists in which a list of concepts is translated into a list of languages, this task may seem trivial, but it has turned out to be quite more challenging than one might think at first, and we can find numerous inconsistencies in large aggregated data collections that result from taking this task not serious enough.

One way to carry out the identification of similar concepts across multiple datasts in a more consisten way is to link the individual concepts represented by their elicitation glosses to a common catalog of concepts. With the Concepticon (https://concepticon.clld.org, List et al. 2021a), such a reference catalog has been available for quite some time now (the first version was published in 2016, see List et al. 2016), and we have profited from the resource in establishing larger data collections such as the CLICS database of cross-linguistic colexifications (Rzymski et al. 2020) and the Lexibank repository (List et al. 2021b).

Computer-Assisted Language Comparison in Practice
Volume 5, Number 1
https://calc.hypotheses.org/3193/

Published on 10/01/2022

Concept Mapping in PyConcepticon

The typical workflow of concept mapping with the help of the Python code accompanying the Concepticon package (PyConcepticon, see Forkel et al. 2021, https://pypi.org/project/pyconcepticon), the mapping is done via the commandline (as outlined in Tjuka 2020). The underlying algorithm comes in two flavors, one just searches for direct matches among the elicitation gloss in the source list and the glosses that have been linked already to the Concepticon, the other method preprocesses the source gloss by trying to identify its basic characteristics (part of speech, if it contains stop characters, etc.) and carrying out some trivial stemming, to map it then against all entries which have been previously mapped to the Concepticon. In both cases, teh mapping is not only done against the "official" Concepticon glosses, but against all elicitation glosses which link to the Concepticon Concept Sets. The advantage of this procedure is that our mapper "learns" from new data and can at times provide quite surprising results, e.g., when searching for "eggplant", you will get a link to AUBERGINE.

```
$ concepticon lookup eggplant
GLOSS CONCEPTICON_ID CONCEPTICON_GLOSS SIMILARITY
-------- ---------- ---------- ----------
eggplant 1146 AUBERGINE 2
```

The disadvantage of the approach is that it is typically limited to the command line (unless one wants to hack into the PyConcepticon code to find out how to use the mapper class), and that one needs to have downloaded and point to the Concepticon data folder, which needs to be accessible to PyConcepticon.

Concept Mapping with PySem

In order to have a lightweight tool for the quick mapping of datasets to Concepticon which works from within Python scripts, I added a specific function to the PySem package (List 2021) along with a workflow by which each new release of the Concepticon data will lead to a new release of PySem in which a dump of the most recent Concepticon version is stored in a Zip file that can be directly accessed from within Python and does not require too much disk space. Thus, PySem offers a variant of the mapping algorithm underlying PyConcepticon, but uses a dump from Concepticon which allows users to use the package without having to download the Concepticon data package.

You can install PySem via the Python package index PIP:

```
$ pip install pysem
```

The mapping algorithm has been modified, and unit tests have been added. As a result, the mapping method distinguishes 20 forms of concept gloss similarity, but since it is an explicit method, there is no guarantee that the numbers from 0 to 20 that describe the similarity make actual sense numerically.

The highest similarity is achieved, if the text, the original string, is identical, and part-of-speech information is provided. A similarity of 10 indicates that the main part of the gloss is similar and the part-of-speech as well, and 9 indicates that part-of-speech information is missing or different.

When mentioning the “main part” of the gloss, this refers to the preprocessing procedure, by which PySem models elicitation glosses internally.

```
>>> from pysem.glosses import parse_gloss
>>> glosses = ["to kill", "kill", "kill (v.)", "kill (somebody)"]
>>> for gloss in glosses:
... parsed_gloss = parse_gloss(gloss)[0]
... print(parsed_gloss.main, parsed_gloss.pos)
kill verb
kill
kill verb
kill
```

As can be seen from this example, PySem extracts some basic parts of the elicitation glosses and also tries to infer additional information regarding the part of speech. The stemming procedure which PySem uses is trivial and simple, consisting of a set of prefixes, markers for part of speech which may recur inside elicitation glosses (such as a v. in brackets), and a list of bracket symbols which allow us to strip all those strings which occur inside brackets.

The whole workflow for PySem thus consists in (A) the preprocessing of the glosses, and (B) the comparison of the preprocessed form with the preprocessed glosses that were mapped in the past to the concept sets in the Concepticon project. All matches are ordered by their similarity score first, and then by the number of elicitation glosses which we can find in Concepticon. Since the mapping procedure was never formally evaluated, we have no way to say whether it is performing well or not, but from my own experience, it seems to perform good enough, and it also works reasonably fast.

Examples

In order to map your data to PySem, you need to load the function `to_concepticon` from within your Python script. Data passed to this function need to be provided in the form of a list of dictionaries, which have at least one key called gloss (but you can modify this by changing the keyword parameter `gloss_ref`). If you want to provide information on parts of speech, you can use the keyword parameter `pos_ref` to define the name of the key in your dictionary. The keyword `language` (using two-letter language codes) allows you to define the language that you want to use for the mapping procedure (PySem currently supports languages such as English, German, Russian, Portuguese, Chinese, French, and some more, but stemming works best for English, German, and French).

To get started, you can check the following code that illustrates how mapping can be done.

```
>>> from pysem.glosses import to_concepticon
>>> glosses = [{"gloss": "Hand", "pos": "noun"}, {"gloss": "Fuß / Bein", "pos": "noun"},
{"gloss": "Aubergine", "pos": "noun"}]
>>> for key, matches in to_concepticon(glosses, language="de", pos_ref="pos").items():
... print(key, matches)
Hand [['1277', 'HAND', 'noun', 20]]
Fuß / Bein [['1297', 'LEG', 'noun', 16]]
Aubergine [['1146', 'AUBERGINE', 'noun', 20]]
```

You can see, that the method yields a dictionary that contains a list of the potential matches (here limited to one exemplar, due to the keyword max_matches set to 1) that correspond to the original gloss, which is now the key. The first number in the list is the Concepticon ID, the second item is the Concepticon Gloss, the third is the part of speech, and the fourth is the similarity score. 20 in our case means that the string was identical and the part of speech was provided.

```
>>> glosses = [{"gloss": "Hand"}, {"gloss": "Fuß / Bein"}, {"gloss": "Aubergine"}]
>>> for key, matches in to_concepticon(glosses, language="de", pos_ref="pos",
max_matches=2).items():
... print(key, matches)
Hand [['1277', 'HAND', 'noun', 19], ['1277', 'HAND', 'noun', 17]]
Fuß / Bein [['1297', 'LEG', 'noun', 15], ['1301', 'FOOT', 'noun', 15]]
Aubergine [['1146', 'AUBERGINE', 'noun', 19], ['1146', 'AUBERGINE', 'noun', 17]]
```

From this example you can see that leaving glosses and extending the number of matches leads to more matches, with some being identical with respect to the Concepticon ID. These are cases in which the elicitation gloss in the Concepticon differs but links to the same ID. It means: there is one match, where Hand is completely identical (apart from the missing part of speech information) with the elicitation gloss in at least one concept list in Concepticon, and another one that is not completely identical.

Conclusion

If you want to use a lightweight package to map concepts to Concepticon or to compare concepts across resources with the help of the Concepticon from within a Python script, the PySem package should be a useful solution, since it offers a rather robust and reliable mapping algorithm that maps from different languages and can also be configured to some degree. The mapping itself may seem inconvenient, due to the nested output, but given that mapping is not a completely straightforward business, it seems useful to keep the output transparent for the time being, especially while we explore the algorithm and try to improve it further.

References

List, Johann-Mattis and Christoph Rzymski and Greenhill, Simon J. and Schweikhard, Nathanael E. and Kristina Pianykh and Annika Tjuka and Carolin Hundt and Robert Forkel (2021a): Concepticon. A resource for the linking of concept lists. Version 2.5.0. Jena:Max Planck Institute for the Science of Human History. https://concepticon.clld.org

List, Johann-Mattis and Forkel, Robert and Greenhill, Simon J. and Rzymski, Christoph and Englisch, Johannes and Gray, Russell D. (2021b): Lexibank: A public repository of standardized wordlists with computed phonological and lexical features [Preprint, Version 1]. Research Square 0.0. 1-31. [Preprint, under review, not peer-reviewed] https://doi.org/10.21203/rs.3.rs-870835/v1

List, Johann-Mattis and Cysouw, Michael and Forkel, Robert (2016): Concepticon. A resource for the linking of concept lists. In: Proceedings of the Tenth International Conference on Language Resources and Evaluation. 2393-2400. http://www.lrec-conf.org/proceedings/lrec2016/pdf/127_Paper.pdf

Forkel, Robert and Rzymski, Christoph and List, Johann-Mattis (2021): PyConcepticon [Python library, Version 2.8.0]. Geneva: Zenodo. https://github.com/concepticon/pyconcepticon

List, Johann-Mattis (2021): PySem: Python library for handling semantic data in linguistics. Leipzig:Max Planck Institute for Evolutionary Anthropology. https://pypi.org/project/pysem

Rzymski, Christoph and Tiago Tresoldi and Simon Greenhill and Mei-Shin Wu and Nathanael E. Schweikhard and Maria Koptjevskaja-Tamm and Volker Gast and Timotheus A. Bodt and Abbie Hantgan and Gereon A. Kaiping and Sophie Chang and Yunfan Lai and Natalia Morozova and Heini Arjava and Nataliia Hubler and Ezequiel Koile and Steve Pepper and Mariann Proos and Briana Van Epps and Ingrid Blanco and Carolin Hundt and Sergei Monakhov and Kristina Pianykh and Sallona Ramesh and Russell D. Gray and Robert Forkel and List, Johann-Mattis (2020): The Database of Cross-Linguistic Colexifications, reproducible analysis of cross- linguistic polysemies. Scientific Data 7.13. 1-12. https://clics.clld.org https://doi.org/10.1038/s41597-019-0341-x

Tjuka, Annika (2020): Adding concept lists to Concepticon: A guide for beginners. Computer-Assisted Language Comparison in Practice 3.1. https://calc.hypotheses.org/2225