Computer-Assisted Language Comparison in Practice

Tutorials on Computational Approaches to the History and Diversity of Languages

Converting Streitberg's Gothic Dictionary to a CLDF Wordlist on a Windows System

Viktor Martinović
Finno-Ugrian Department
University of Vienna

I recently converted the Gothic dictionary written by Wilhelm Streitberg to a CLDF wordlist. Since I was using Windows, I had some difficulties during the conversion progress, which Unix system users may not have to deal with. I thought it would be useful to share my experience here and point out that users of Windows operating systems should be aware of certain aspects when converting data to CLDF.

Overview

I created the CLDF wordlist (Forkel et al. 2018) using the CLDFBench package (Forkel and List 2020) and the workflow of Lexibank (List et al. 2021). A good overview of the advantages CLDF offers can be found in List (2021a). Furthermore, there are three tutorials that helped me, and I will refer to them as "first", "second", and "third tutorial" in this post. The first and second tutorials are part of the official documentation of the CLDF project on GitHub and proved to be a helpful introduction to the topic. You will learn about the general ideas and terminologies behind CLDF there, as well as the usage of CLDFBench. The third tutorial (List 2021b) focuses on PyLexibank (Forkel et al. 2021), a plugin for CLDFBench. You are also invited to explore the GitHub repository of this article, particularly issue #2 in the issues section, to see the problems I have encountered. There is an additional tutorial on YouTube: https://www.youtube.com/watch?v=zgLrDJ0zMPQ

Preliminary Steps

Create an account on GitHub, download Python (tick "Add Python to Windows Path" when installing), and download Git and GitHub Desktop, since using Git through the command line gave me this error-message when I tried to push:

```
! [remote rejected] master -> master (refusing to allow
an OAuth App to create or update workflow
`.github/workflows/python-package.yml` without `workflow`
```

Computer-Assisted Language Comparison in Practice
Volume 5, Number 2
https://calc.hypotheses.org/3318

Published on 20/02/2022

```
scope) error: failed to push some refs to
'https://github.com/martino-vic/streitberggothic.git'
```

Create a batch-file and place it into the root directory by opening a text file, typing below text, and saving it as "streitberggothic--dev.bat":

```
cldfbench lexibank.makecldf lexibank_streitberggothic.py --
concepticon=C:\Users\PATH\TO\YOURFOLDER\concepticon\concepticon-data --
glottolog=C:\Users\PATH\TO\YOURFOLDER\glottolog --
clts=C:\Users\PATH\TO\YOURFOLDER\clts --concepticon-version=v2.5.0 --glottolog-
version=v4.5 --clts-version=v2.2.0 --dev

pause
```

It is important to place this file in the correct directory. The error message Invalid dataset spec: <lexibank.dataset> lexibank_streitberggothic.py means you are running your terminal from the wrong directory, more about this in a bit. I have also placed a second batch-file in the same folder with the same command but without the --dev flag. Let's break down this script:

- `cldfbench`: As mentioned earlier, PyLexibank is a plug-in to CLDFBench, so once we install the former, we will gain access to commands of the latter.
- `lexibank.makecldf`: This is the main command that will convert the raw data to CLDF.
- `lexibank_streitberggothic.py`: This is the Python script that does the conversion and that we will create later in this tutorial. In the current form, our batch-file would need to be in the same directory as this script. If the batch-file was one directory above, we would need to specify the path: cldfbench lexibank.makecldf streitberggothic/lexibank_streitberggothic.py So the path is specified relative to the location of the batch-file (i.e. our cTimes New Romanurrent working directory) with forward slashes before lexibank_streitberggothic.py. An alternative way would be to skip the path (i.e. keep the command like first shown) but to cd into the correct directory before running it.
- `--concepticon=C:\Users\PATH\TO\YOURFOLDER\concepticon\concepticon\concepticon-data --glottolog=C:\Users\PATH\TO\YOURFOLDER\glottolog --clts=C:\Users\PATH\TO\YOURFOLDER\clts --concepticon-version=v2.5.0 --glottolog-version=v4.5 -clts-version=v2.2.0`. These define the location and version of the three repositories you must clone first, as described in the fifth point of the third tutorial: „[M]ake sure to have downloaded actual versions of Glottolog,

Concepticon, and CLTS, ideally placing them in a single folder to make it easier to remember where they are on the system" – Note that these libraries together are more than 1GB in size, so ensure you have sufficient disk space and a good Internet connection.
- `--dev`: The „development" flag. If you don't use it, the algorithm may spend a few minutes looking up details about your language in Glottolog.
- `pause`: This ensures that the terminal does not automatically close after the command is executed so that you can check the output that has been printed to the console.

The only thing left to do now is to pip install pylexibank. It is recommended to do so in a virtual environment because there are many dependencies. If you want to learn about virtual environments, this tutorial has always helped me: https://www.youtube.com/watch?v=N5vscPTWKOk

Now that we have all the necessary tools in our hands, we need to create a repository. First, open your command prompt by typing "`cmd`" into the start menu search box. Change the working directory in the command prompt to the directory containing all GitHub repositories (`cd C:\Users\USERNAME\Documents\GitHub` by default). Then, follow closely the steps described in the second tutorial. This will give you the skeleton of the repository, which you can already publish on GitHub with GitHub Desktop. Publishing it will facilitate development since it enables you to open issues and discuss them with the team behind Lexibank. Now return to the third tutorial and follow all remaining instructions in "preliminary steps". These include a number of manual tasks, such as documenting a detailed bibliography, but they are simple and error-free tasks. Once you have completed this part, you are almost ready to start converting.

Step 1: Mapping Concepts to Concepticon

Unlike the third tutorial, this step is done with PySem (List 2021c), which connects the meanings in your table to Concepticon entries. From its new tutorial (List 2022), you can learn how to use it. I built it into my own script, which I called makeconcepts.py. This script takes Streitberg-1910-3645.tsv as input and gives concepts.tsv as output.

Step 2: Linking Languages to Glottolog

Follow the steps described in the third tutorial point four. If you are using the `--dev` flag in the beginning, the languages will not be combined with information from Glottolog.

Step 3: Creating the Lexibank Script

This is the most important part of the whole process since CLDF is not only about providing standardized data, but also documenting the production process. This means that all preprocessing steps are documented in a Python script, which we will create in this section. You can inspect my own script here. Let's break it down.

```
import pathlib
import re
import attr
from clldutils.misc
import slug
from pylexibank import Dataset as BaseDataset
from pylexibank import FormSpec, Concept
```

- `pathlib`: an inbuilt python library and the preferred way to handle file paths across operating systems.
- `re`: an inbuilt library to handle regular expressions.
- `attr`: belongs to the PyLexibank ecosystem and will be used to create our own custom language class.
- `slug`: will turn a string into a valid lowercase identifier with no spaces.
- `BaseDataset`: a class that our own class Dataset will later inherit.
- `FormSpec` is a class with which the cleaning of strings will be done.

```
REP = [(x, "") for x in "†*[]~?;+-"] + [(x, "a") for x in "áàā"] + [(x, "ɪ") for x in "ïîī"] + [("ē",
"e"), ("ō", "o"), ("ū", "u"), (" ", "_"), ("‚_", ", ")]
```

This is the list of tuples, with which the cleaning will be done. The first value in every tuple will be replaced by the second. REP stands for replacement.

```
@attr.s class
CustomConcept(Concept):
   POS = attr.ib(default=None)
```

Since our data frame will have one additional column, namely "POS", we need to create a custom language class.

```
def cln(word): return re.sub("[†\\d\\.\\*\\?\\~]", "", word)
```

This will apply the same cleaning procedure that we applied when mapping the concepts to Concepticon in step one.

```
class Dataset(BaseDataset):
    dir = pathlib.Path(__file__).parent
    id = "streitberggothic"
    form_spec = FormSpec(separators=",", first_form_only=True,
replacements=REP)
    concept_class = CustomConcept
```

As already announced: We're inheriting from BaseDataset, dir specifies the working directory, in this case, the parent directory of the current script. id is the name of our repository, we use the FormSpec class to plug in the replacement rules that we have defined earlier. In the last line, we are plugging in our custom language class with the additional "POS" column that we previously created.

```
def cmd_makecldf(self, args):
```

This function reads data that has been prepared and placed in the "raw" and "etc" folders. They are converted to CLDF, and written to the "cldf" folder. This function consists of four sections: "add bib", "add concept", "add language", and "add forms".

```
    # add bib
    args.writer.add_sources()
    args.log.info("added sources")
```

- `args.writer.add_sources()`: This will read the sources.bib file, which was prepared during the preliminary steps, placed in the "raw" folder, and written to the folder "cldf".
- `args.log.info("added sources")`: This will print information on the console to help to debug and know which parts of the script are currently being executed.

```
    # add concept
    concepts = {}
    for i, concept in enumerate(self.concepts):
        idx = str(i + 1) + "_" + slug(concept["sense"])
        args.writer.add_concept(
            ID=idx,
```

```
            Name=concept["sense"],
            POS=concept["pos"],
            Concepticon_ID=concept["Concepticon_ID"],
            Concepticon_Gloss=concept["Concepticon_Gloss"]
        )
        concepts[concept["sense"], concept["pos"]] = idx
    args.log.info("added concepts")
```

- `concepts = {}`: This dictionary will be filled during the following loop and called when creating the column “Parameter_ID” in parameters.csv.
- `for i, concept in enumerate(self.concepts)`:: self.concepts is a list of ordered dictionaries based on the table read from the concepts.tsv file we created in step one and placed in the “etc” folder. Every dictionary in this list corresponds to a row within the data frame and is of the form: `OrderedDict([('sense', '†Aai'), ('pos', '<none>'), ('Concepticon_ID', ''), ('Concepticon_Gloss', '')])`. The key is always the column name provided in concepts.tsv, and the value is the specific column content of the particular row. And now we are looping through this list of dictionaries in our script.
- `idx = str(i + 1) + "_" + slug(concept["sense"])`: Here, we define the ID that will connect the forms in forms.csv (one of the output files) to the concepts in parameters.csv (another output file).
- `args.writer.add_concept(ID=idx, Name=concept["sense"], POS=concept["pos"], Concepticon_ID=concept["Concepticon_ID"], Concepticon_Gloss=concept["Concepticon_Gloss"])`: Like args.writer.add_sources() wrote sources.bib, this line is going to write parameters.csv based on concepts.tsv. The columns “ID”, “Name”, “Concepticon_ID”, “Concepticon_Gloss” and “POS” will be the column names in parameters.csv. The columns “Concepticon_ID”, “Concepticon_Gloss” can also be skipped since linking to Concepticon is optional. ID is the unique identifier of each row, as explained earlier. Name is the meaning of the word and concept["sense"] we are accessing the content of the column “sense” of the row that our loop is currently in. The same idea applies to Concepticon_ID,Concepticon_Glossand ID.
- `concepts[concept["sense"], concept["pos"]] = idx`: This part fills the empty dictionary we have created earlier.
- `args.log.info("added concepts")`: Here, we are telling the program to print the current status to the console.

```
# add languages
args.writer.add_languages()
args.log.info("added languages")
```

- `args.writer.add_languages()`: Here, we are reading languages.tsv from the folder "etc" that we created in step two and writing langauges.csv to the folder "cldf".
- `args.log.info("added languages")`: Here, we are printing a status update to the console. The --dev flag in our batch-file prevents the program from potentially getting stuck at this part.

```
# add forms
for idx, row in enumerate(self.raw_dir.read_csv(
    "Streitberg-1910-3645.tsv", delimiter="\t", dicts=True)[1:]):
  args.writer.add_forms_from_value(
    Local_ID=idx,
    Language_ID="Gothic",
    Parameter_ID=concepts[cln(row["sense"]), cln(row["pos"])],
    Value=row["form"],
    Source="557564")
```

- `for idx, row in enumerate(self.raw_dir.read_csv("Streitberg-1910-3645.tsv", delimiter="\t", dicts=True)[1:])`: This reads our main tsv-file from the folder "raw" with self.raw_dir.read_csv(). Provide the filename and the separator – in our case a tab. [1:] means we are skipping the first row, since that is our header. Thanks to dicts=True, we are once more looping through a list of dictionaries.
- `args.writer.add_forms_from_value(Local_ID=idx, Language_ID="Gothic", Parameter_ID=concepts[cln(row["sense"]), cln(row["pos"])], Value=row["form"], Source="557564")`: This defines the columns of forms.csv. Local_ID consist of integers 0, 1, 2, 3, … that come from the enumerate() function , Language_ID is always "Gothic", since we only have one language in this data set – this has to be identical with the identifier we have provided in languages.tsv. Parameter_ID is the slugified identifier from the dictionary "concepts" that we have populated in the previous

loop. Note that with cln() we are applying the exact same cleaning procedure as in makeconcepts.py, in order to access the correct dictionary keys. Value is the clean version of form and we have created this column by employing form_spec and REP.

Step 4: Final Check

Open your terminal, change directory into the root folder of your repository and run `pip install -e`. If you now double-click on the batch-file that we have created in the beginning, your console should look something like this after execution:

```
C:\Users\Viktor\Documents\GitHub>cldfbench lexibank.makecldf streitberggothic/lexibank_str
gothic.py --concepticon=C:\Users\Viktor\OneDrive\PhD\lexibank\concepticon\concepticon\conc
-data --glottolog=C:\Users\Viktor\OneDrive\PhD\lexibank\glottolog --clts=C:\Users\Viktor\C
\PhD\lexibank\clts --concepticon-version=v2.5.0 --glottolog-version=v4.5 --clts-version=v2
dev
INFO    running _cmd_makecldf on streitberggothic ...
INFO    added sources
INFO    added concepts
INFO    added languages
INFO    added forms
INFO    file written: C:/Users/Viktor/Documents/GitHub/streitberggothic/cldf/.transcriptio
t.json
INFO    Summary for dataset C:\Users\Viktor\Documents\GitHub\streitberggothic\cldf\cldf-me
json
- **Varieties:** 1
- **Concepts:** 3,386
- **Lexemes:** 3,644
- **Sources:** 1
- **Synonymy:** 1.08
- **Invalid lexemes:** 0
- **Tokens:** 24,899
- **Segments:** 50 (0 BIPA errors, 0 CTLS sound class errors, 50 CLTS modified)
- **Inventory size (avg):** 50.00
INFO    file written: C:/Users/Viktor/Documents/GitHub/streitberggothic/TRANSCRIPTION.md
INFO    file written: C:/Users/Viktor/Documents/GitHub/streitberggothic/cldf/lingpy-rcPara

INFO    ... done streitberggothic [12.8 secs]
```

Congratulations! You have successfully converted a csv file to CLDF! Now you can push the finished repository to GitHub using GitHub Desktop and request its integration into Lexibank!

Supplementary Material

The dataset is available online at GitHub (https://github.com/lexibank/streitberggothic) and Zenodo (https://doi.org/10.5281/zenodo.5899636). All code documented here is taken from the data on GitHub. It should therefore be straightforward to check the different steps described in this little tutorial.

References

Forkel, Robert and List, Johann-Mattis and Greenhill, Simon J. and Rzymski, Christoph and Bank, Sebastian and Cysouw, Michael and Hammarstrom, Harald and Haspelmath, Martin and Kaiping, Gereon A. and Gray, Russell D. (2018): Cross-Linguistic Data Formats, advancing data sharing and re-use in comparative linguistics. Scientific Data 5.180205. 1-10. https://doi.org/10.1038/sdata.2018.205

Forkel, Robert and List, Johann-Mattis (2020): CLDFBench. Give your Cross-Linguistic data a lift. In: Proceedings of the Twelfth International Conference on Language Resources and Evaluation. 6997-7004.

List, Johann-Mattis and Forkel, Robert and Greenhill, Simon J. and Rzymski, Christoph and Englisch, Johannes and Gray, Russell D. (2021): 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

Johann-Mattis List, "How to Share Data and Code when Submitting Papers to a Journal: Transparent Data (How to do X in Linguistics 8)," in Computer-Assisted Language Comparison in Practice, 02/08/2021a, https://calc.hypotheses.org/2877

Johann-Mattis List, "Converting the Vietic Dataset by Sidwell and Alwes from 2021 to CLDF," in Computer-Assisted Language Comparison in Practice, 08/09/2021b, https://calc.hypotheses.org/2954.

Forkel, Robert and Simon J Greenhill and Hans-Jorg Bibiko and Christoph Rzymski and Tiago Tresoldi and Johann-Mattis List (2021): PyLexibank. The python curation library for lexibank [Software Library, Version 2.8.2]. Geneva: Zenodo. https://github.com/lexibank/pylexibank

Hammarstrom, Harald and Haspelmath, Martin and Forkel, Robert and Bank, Sebastiaon (2021): Glottolog. Version 4.4. Leipzig:Max Planck Institute for Evolutionary Anthropology. https://glottolog.org

List, Johann-Mattis and Christoph Rzymski and Simon Greenhill and Nathanael Schweikhard and Kristina Pianykh and Annika Tjuka and Carolin Hundt and Robert Forkel (2021): 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 Anderson, Cormac and Tresoldi, Tiago and Forkel, Robert (2021): Cross-Linguistic Transcription Systems. Version 2.1.0. Jena: Max Planck Institute for the Science of Human History. https://clts.clld.org

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

Johann-Mattis List, "How to Map Concepts with the PySem Library," in Computer-Assisted Language Comparison in Practice, 10/01/2022, https://calc.hypotheses.org/3193.