Patrick J. Burns

Building a Text Analysis Pipeline for Classical Languages

Abstract: With large text collections for Ancient Greek and Latin now widely available, classicists are increasingly interested in extracting information systematically from these texts. The fields of information retrieval and natural language processing offer tools and methods to address this, but classical-language support can be limited and researchers must often cobble together separate, sometimes incompatible tools to accomplish basic text analysis tasks. In this chapter, I review the tools currently available for digital philological work on Ancient Greek and Latin and introduce the Classical Language Toolkit, an open-source Python framework that addresses the desideratum of a complete text analysis pipeline for historical languages.

1 Introduction

With large text collections for Ancient Greek and Latin now widely available, classicists are increasingly interested in extracting information systematically from these texts and constructing derivative datasets. Digital philologists have been able to turn to the fields of information retrieval and natural language processing (NLP) for tools and methods to accomplish these goals, but classical-language support can be limited and, as a result, researchers must often cobble together separate, sometimes incompatible tools to accomplish basic text analysis tasks and approximate the kinds of integrated solutions available for work in modern languages.

In the first part of this chapter, I review examples of text analysis frameworks that are available for work in modern languages (such as Stanford CoreNLP and the Natural Language Toolkit), highlighting in particular one of the defining features of these frameworks – the pipeline, a sequential workflow of transformation and annotation. Each of the frameworks listed above offer a complete pipeline of text analysis tasks, including tokenization, lemmatization, part-of-speech and morphological tagging, and named entity extraction, among other tasks. Pipelines, considered the "standard approach to realize text

Patrick J. Burns, Quantitative Criticism Lab, University of Texas at Austin

https://doi.org/10.1515/9783110599572-010

analysis processes,"[1] are an orderly and efficient way to proceed through a series of analysis tasks, especially in cases where it is useful or necessary for the results of certain tasks to be used as the starting point for processing subsequent tasks. In the second part of the chapter, I review examples of solutions to each task along the Greek and Latin text analysis pipeline and discuss briefly how they could be patched together into a makeshift pipeline if necessary.

By way of conclusion, I introduce the Classical Language Toolkit (CLTK), an open-source Python framework dedicated to natural language processing support for historical languages. CLTK has made progress in the past three years in collecting corpora for a wide variety of historical languages covering ancient, classical, and medieval Eurasia and building out the basic language resources to support these languages across the text analysis pipeline. CLTK shows promise of addressing the desideratum of a complete text analysis pipeline for Greek and Latin, as well as a large number of other less-resourced historical languages.[2]

2 Text analysis pipelines

In text analysis, transformation and annotation tasks are often processed in such a way that new annotations build on previous transformations and annotations of a given text. This sequence is commonly referred to as a pipeline, as in this definition from Henning Wachsmuth:

> "Text mining deals with tasks that often entail complex text analysis processes, consisting of several interdependent steps that aim to infer sophisticated information types from collections and streams of natural language input texts. [. . .] Because of the interdependencies between analyses, the standard way to realize a text analysis process is in the form of a text analysis pipeline, which sequentially applies each employed text analysis algorithm to its input."[3]

1 (Wachsmuth 2015, 37).

2 This chapter limits its scope to Ancient Greek and Latin, but it is important to point out that the development of NLP pipelines is an area of ongoing work for several historical languages. See, for example, Chiarcos et al. (2018) for Sumerian or Zeldes and Schroeder (2016) for Coptic.

3 (Wachsmuth 2015, 4). For a formal definition of text analysis pipelines, see Wachsmuth (2015, 37). For a clear explanation of the terminology involved in describing pipelines, specifically the use of the terms "tool" and "component," see de Castilho and Gurevych (2014, 2). In this chapter, I use "tool" to refer to a piece of software, a web application, or a web service that performs a text analysis task; I use "component" to refer to a tool that is included as a discrete step in a text analysis pipeline.

So, for example, the input sentence

Quo usque tandem abutere, Catilina, patientia nostra?

may be first transformed into a list of words (and punctuation marks) by a tokenizer to yield

['Quo', 'usque', 'tandem', 'abutere', ',', 'Catilina', ',', 'patientia', 'nostra', '?']

which in turn may be annotated by a part-of-speech (POS) tagger into a parallel list of POS tags to yield

['ADV', 'ADV', 'ADV', 'VERB', 'PUNCT', 'NOUN', 'PUNCT', 'NOUN', 'ADJ'][4]

and so on. In this configuration, the POS tagger depends not directly on the plaintext that was originally fed into the pipeline, but rather uses as its input the output of the preceding tokenizer. These kinds of relations between the constituent parts, or components, of a pipeline are illustrated in Figure 1.

Even this system can grow quite complex as the number of components is increased. An advantage to working with pipelines is that this complexity is managed by the sequential workflow as well as the storage of annotations in parallel data structures. Another advantage of using well-defined pipelines in text analysis work is that this practice promotes shareability and reproducibility in research workflows.[5] Because of these advantages in managing and supporting task analysis processes, pipelines are considered the "standard approach" for this kind of work.[6]

4 There are several annotation schemes for POS tagging; this example uses the Universal POS tagset; cf. https://universaldependencies.org/u/pos/ (last access 2019.01.31).

5 (de Castilho and Gurevych 2014): "It is essential that [. . .] pipelines can easily be shared between researchers, to reproduce results, to evolve experiments, and to allow for a better understanding of the exact details of an experiment."

6 (Wachsmuth 2015, 37). There are disadvantages in working with pipelines as well. For example, they can be inefficient. Assigning each annotation task, for example, its own space in a pipeline adds certain processing overhead and can introduce redundancies where tasks are strongly correlated, such as (as we will see in greater detail below) lemmatization and POS tagging. In addition, pipelines can be subject to error propagation, since errors introduced early in the execution flow can have downstream consequences. See Marciniak and Strube (2005) and Clarke et al. (2012, 1–2) for potential areas of improvement in pipeline design.

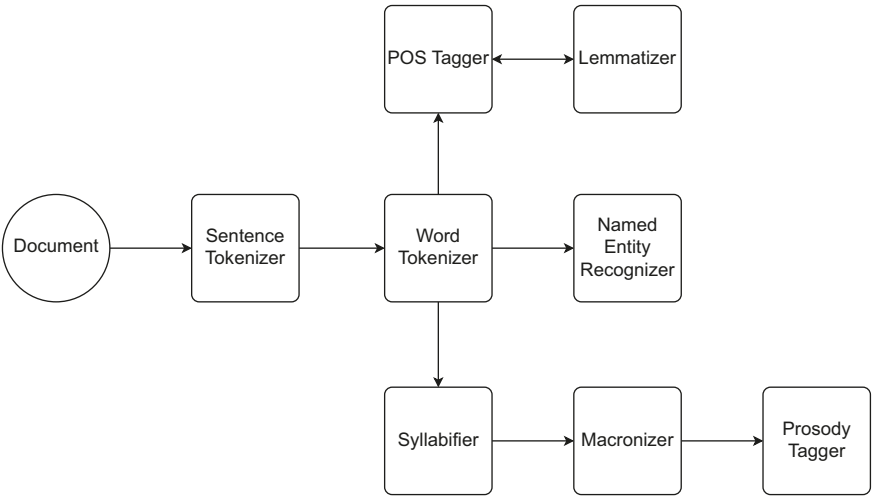

Figure 1: Here is a sample text analysis pipeline, proceeding left-to-right from a plaintext Latin document to a collection of derivative annotations.

3 Text analysis frameworks

The two most prominent frameworks for building pipelines have been GATE (General Architecture for Text Engineering) and UIMA (Unstructured Information Management Architecture), both of which are Java-based and use XML to define instructions for processing components.[7] Both GATE and UIMA offer robust systems for the sequential processing of unstructured text and allow for a great deal of flexibility and extensibility in design, either through rules-based annotations (for example, the JAPE annotation language for GATE) or through the development of Java annotation scripts.

These frameworks may prove useful for large, production-ready applications, but for many researchers in digital philology a more "batteries-included" framework is likely suitable enough. Options abound at present: OpenNLP, DKPro-Core,[8] ClearNLP, LingPipe, spaCy, Argo, Weblicht, to name

7 Gate: (Cunningham 2002); UIMA: (Ferrucci and Lally 2004).

8 Note that OpenNLP and DKPro are implementations of UIMA for which a collection of NLP components has been included by default.

just a few.[9] In the remainder of this section, I would like to concentrate on two frameworks with widespread adoption and active development that highlight, as I see it, two different philosophies toward the use of pipelines: Stanford CoreNLP and the Natural Language Toolkit.

Stanford CoreNLP is a self-described Java "annotation pipeline framework," with robust support for common tasks.[10] Pipelines in CoreNLP are conceived of as an "Annotation" object, that is a list of instructions of which components should be run in which order.[11] Text is added to the Annotation and then, as each component is run, either the transformed text or the annotated text is stored in the object. While specific components can be called at runtime, by default, the full pipeline is applied to a given text. The components are pre-defined such that a specific algorithm is used for each task in order to take advantage of state-of-the-art speed and accuracy. The presentation of a "core" pipeline with a set of "core" components is not an accident. As originally conceived, developers valued ease of use: "Most users benefit greatly from the provision of a set of stable, high quality linguistic analysis components, which can be easily invoked for common scenarios."[12] Accordingly, users are presented with a fully functional pipeline from the outset. It can be customized, but it does not need to be. Given no intervention from the user, a complete pipeline from tokenization to coreference resolution is ready to be run.[13]

9 OpenNLP: https://opennlp.apache.org/; DKPro-Core: https://dkpro.github.io/dkpro-core/; ClearNLP: https://github.com/clearnlp; LingPipe: http://alias-i.com/lingpipe/; spaCy: https://spacy.io/; Argo: http://argo.nactem.ac.uk/; Weblicht: https://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Main_Page (last access 2019.01.31). Language-specific options may prove useful depending on the research project; see, for example, FudanNLP for Chinese text (https://github.com/FudanNLP/fnlp) or IceNLP for Icelandic text (https://github.com/hrafnl/icenlp) (last access 2019.01.31). NLP, including its application to classical languages, is a quickly developing and ever evolving area. For digital philologists, one development worth watching is the integration of NLP datasets and models with web services, as for example with META-SHARE (Piperidis 2012) and Language Application Grid (LAPPS) (Verhagen et al. 2016).

10 (Manning et al. 2014); available online at https://stanfordnlp.github.io/CoreNLP/ (last access 2019.01.31). Note that since the writing of this chapter, a Python implementation of the Stanford tools has been released (StanfordNLP, available online at https://stanfordnlp.github.io/stanfordnlp/) with some out-of-the-box support for Greek and Latin. As it has just been released, it is too early to evaluate fully its impact of digital classical philology.

11 This process is described in detail at https://stanfordnlp.github.io/CoreNLP/pipelines.html (last access 2019.01.31).

12 (Manning et al. 2014, 56).

13 It is interesting to note, in the context of this chapter, that the developers of CoreNLP started the project from a desire to move away from what I have called makeshift pipelines; (Manning et al. 2014, 55): "Previously, when combining multiple natural language analysis

The Natural Language Toolkit, on the other hand, has a different philosophical orientation and as such a different approach to text analysis pipelines. NLTK is an open-source Python NLP framework with origins in a pedagogical approach to NLP.[14] From its inception, it promoted a set of "requirements," namely consistency, extensibility, documentation, simplicity, and modularity, alongside a set of "non-requirements," namely comprehensiveness, efficiency, and cleverness. The goal was to support a complete pipeline of text analysis tasks, while allowing users to "augment and replace existing components, learn structured programming by example, and manipulate models."[15] Considering its pedagogical focus, it is unsurprising that the framework has become so closely associated with what amounts to a user guide-as-textbook, *Natural Language Processing with Python*, or the "NLTK Book."[16] The structure of the book promotes a pipeline-centered take on text analysis as readers are guided from tokenization to tagging to other advanced tasks over the course of twelve chapters. Since the focus is on learning NLP basics and best practices, for each task, users are offered a number of options and are presented with the advantages and disadvantages of working with various interfaces, algorithms, and so on. For example, in chapter 5, users are introduced to several different POS taggers offered by NLTK (including default tagging, regular expression tagging, n-gram tagging, transformation-based tagging, and so on).[17] By working one's way through the NLTK book, it becomes possible to write basic Python scripts that function as pipelines.

These frameworks cover two different approaches to the problem of supporting end-to-end pipelines. CoreNLP, not unlike Gate or UIMA, works by specifying a set of instructions for defining the execution flow of different components. NLTK, on the other hand, at least in its original conception, has a pedagogical focus and so the creation of a pipeline, that is the decision about which components to use and how best to connect their inputs and outputs, is left to the user. Again, with respect to its pedagogical orientation, the focus is on the user understanding the function of each component and, just as importantly, understanding the relationship between each component, rather than simply setting a series of instructions in motion.

components, each with their own ad hoc APIs, we had tied them together with custom code glue. The initial version of the annotation pipeline was developed in 2006 in order to replace this jumble with something better."

14 (Loper and Bird 2002): "NLTK provides a simple, extensible, uniform framework for assignments, projects, and class demonstrations. [. . .] It was deliberately designed as courseware and gives pedagogical goals primary status." The project is available online at https://www.nltk.org (last access 2019.01.31).

15 (Loper and Bird 2002, 1).

16 (Bird et al. 2015).

17 (Bird et al. 2015, Ch. 5.4).

With their different philosophical orientations, CoreNLP and NLTK each offer pros and cons for how we should think about building pipelines for use with Greek and Latin texts. Before we can do this, however, it is necessary to look first at what is currently available with respect to "pipelines" for classical languages.

4 "Pipelines" for classical languages

4.1 Coverage of classical languages in text analysis frameworks

For all of the progress in text analysis frameworks for modern-language research, the fact remains that classical-language support still lags behind. This is particularly apparent with respect to the development of pipelines. Neither CoreNLP nor NLTK support Greek and Latin out of the box. As such, digital philologists working with these languages must forge an alternative path.

In the section that follows, I review the available "components" for classical languages, that is standalone tools that perform the kinds of transformations or yield the kinds of annotations we would expect in a fully implemented pipeline.

4.2 Available "components" for classical languages

This section highlights tools that digital philologists have been able to avail themselves of in the absence of dedicated, well-resourced frameworks like CoreNLP or NLTK.[18]

4.2.1 Tokenization

In most text analysis pipelines, tokenization – whether the division of a text into paragraphs, sentences, words, or some other meaningful unit – is the first

18 This discussion of specific pipeline tasks in the following sections as well as the selection of tools and resources mentioned for supporting digital philological work on Greek and Latin is meant to be representative rather than comprehensive. I work here from the premise, "If I wanted to emulate a CoreNLP-style pipeline, what standalone tools could I use to get the job done."

step. Since the vast majority of Greek and Latin text collections are derived from modern editions in which sentences are punctuated and words are delimited by spaces, most tokenization tasks for these languages do not require a customized solution. Accordingly, there tend not to be standalone tools for tokenization, but rather this step tends to be built into the preprocessing stage of other components.

4.2.2 Lemmatization

Lemmatization (and the closely related areas of part-of-speech tagging and morphological tagging) has a long tradition of computational work in Greek and Latin and continues to be a particularly active area of research.[19] Unsurprisingly, then, it is perhaps the pipeline task best supported by standalone tools. There are both command line tools available for Greek and Latin lemmatization as well as web applications and services, allowing for great flexibility in how these tasks can be performed and how results can be obtained for use elsewhere in a makeshift pipeline.

Morpheus, developed for use in the Perseus Digital Library, is perhaps the best known lemmatizer for both languages.[20] A rules-based lemmatizer drawing on data from lexica available in Perseus, Morpheus returns lemmas alongside POS identifications and morphological parses on the site's Greek Word Study Tool and Latin Word Study Tool.[21] It can also be compiled locally and run from the command line. In either case, it is possible for users to extract annotations for use in a makeshift pipeline by either cutting-and-pasting Word Study Tool results or – the more direct and efficient method – by capturing the standard output from running the command-line scripts. This is more or less the pattern for another popular Latin lemmatizer, Whitaker's Words, which can also be run either through a web application or a command-line interface.[22] Another option for extracting lemmas from texts, and a good option for working with blocks of

19 See Bodson and Evrard (1966) for an example of early work in Latin lemmatization. Eger et al. (2015, 2016) offer two recent reviews and comparisons of lemmatizers.

20 (Crane 1991).

21 The Word Study Tools are available online at http://www.perseus.tufts.edu/hopper/morph (last access 2019.01.31). Morpheus is also available as web service through the Perseids Project; the documentation for this project is available online at https://github.com/perseids-project/perseids_docs/wiki/Morphology-Service-Setup (last access 2019.01.31).

22 (Whitaker 1993); available online at http://www.archives.nd.edu/cgi-bin/words.exe (last access 2019.01.31). The documentation for command-line operation of Words is available online at http://archives.nd.edu/whitaker/wordsdoc.htm (last access 2019.01.31).

text, comes from Biblissima through their Collatinus and Eulexis lemmatizers, for work in Latin and Greek respectively.[23] Lastly, at least with respect for Latin, tools have emerged to push lemmatization forward in terms of coverage, speed, and accuracy. Lemlat stands out for having widely expanded the lexical base for assisting lemmatization; already supporting a large lexicon drawn from Georges's *Handwörterbuch*, Gradenwitz's *Laterculi vocum Latinarum*, and the *Oxford Latin Dictionary*, Lemlat has also added a large amount of onomastic data to increase coverage significantly.[24] In addition to Lemlat, another lemmatizer that has more than held its own in a crowded field is LatMor, which compares favorably to the competition in coverage and accuracy, but with processing speeds that are up to 1200 times faster.[25]

Lemmatization is well supported by standalone tools, though perhaps somewhat better for Latin than for Greek. Digital philologists should have little trouble building lexical annotations of this sort for text analysis work. Disambiguation remains a concern (so, for example, correctly tagging the Latin preposition *cum* versus the conjunction *cum*), but advances in computational approaches to lemmatization combined with advances in related annotation tasks like POS tagging are helping to solve this problem.

4.2.3 Part-of-speech (and morphological) tagging

All of the lemmatization tools noted in the previous section also provide some manner of part-of-speech and morphological tagging. This make sense as there is a close relationship between these tasks. Accordingly, POS and morphological annotations from these tools can be captured alongside lexical annotations and used in a pipeline.

Nonetheless, one tool worth calling attention to is TreeTagger, a probabilistic POS tagger written by Helmut Schmid in the mid 1990s.[26] TreeTagger has extensive language support, including classical languages. Latin is supported by two parameter files, one based on selected data from PROIEL, Perseus, and the Index

23 (Ouvard 2010). Collatinus is available online at https://outils.biblissima.fr/fr/collatinus/; Eulexis at https://outils.biblissima.fr/fr/eulexis/ (last access 2019.01.31).

24 (Passarotti et al. 2017); (Budassi and Passarotti 2016); available online at http://www.ilc.cnr.it/lemlat/ (last access 2019.01.31).

25 (Springmann et al. 2016); available online at http://www.cis.uni-muenchen.de/~schmid/tools/LatMor/ (last access 2019.01.31).

26 (Schmid 1994); available online at http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/ (last access 2019.01.31).

Thomisticus and another much larger file based only on the Index Thomisticus; Ancient Greek with a parameter file based on PROIEL and Perseus data.[27] If one were designing a text analysis pipeline, one could easily capture its output on the command line, as with lemmatizers like Lemlat or LatMor. That said, wrappers (or programming interfaces that let you use code from one domain or language inside another) have been written so that TreeTagger can be used easily in Python, R, and JavaScript, among other languages, thus making it even easier to incorporate in custom-built pipelines.

4.2.4 Named Entity Recognition

Unlike lemmatization and POS tagging, named entity recognition (NER), or the systematic tagging of words in texts by category (so, *Roma* as a "location" or Σωκράτης as a "person") is not well-supported by standalone tools. With respect to Greek and Latin, a lack of annotated texts and robust language models underlies the problem.[28] All is not lost though as there is at least one (albeit longhand) way to retrieve annotations from Greek and Latin texts. Recogito is an online platform supporting the annotation of places, persons, and events through linked data.[29] While Recogito can provide automatic NER tagging (using Stanford CoreNLP), at present this feature is limited to English, French, German, and Spanish. That said, users can upload texts and annotate them by hand on the platform, and, with geographic entities in particular, the linked-data-enhanced advanced search does a good job with validating Greek and Latin annotations against online gazetteers.[30] These annotations can then be exported in a wide variety of data for integration into a makeshift pipeline.

27 Latin: (Brandolini n.d.; Passarotti n.d.); Ancient Greek: (Vatri and McGillivray n.d.). A complete list of parameter files for all supported languages can be found at http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/#parfiles (last access 2019.01.31).

28 Erdmann et al. (2016) review the challenges of named entity recognition on Latin texts and suggest directions forward.

29 (Simon et al. 2017); available online at https://recogito.pelagios.org/ (last access 2019.01.31).

30 For example, Ὀλύμπια does not match a location entity automatically in the Recogito annotation interface, but the advanced search feature yields matches from the following gazetteers: GeoNames, the Digital Atlas of the Roman Empire, and Pleiades. Since the writing of this chapter, Recogito has introduced beta support for Latin NER.

4.2.5 Miscellaneous pipeline components

Digital philological work on Greek and Latin text raises the need for components that are encountered rarely, if ever, in pipelines for modern languages. So, for example, macronization and prosody tagging may be useful tasks for analyzing Latin literature and should be considered in the construction of a pipeline for this domain.[31] Accordingly, sites like Pede Certo, which allows users to upload a block of Latin poetry and return a fully scanned version, or Macronizer, which will add macrons algorithmically to a Latin text, should also be considered as potential components depending on the research question at hand.[32]

5 Introducing the Classical Language Toolkit

Chaining together a number of incompatible tools may prove useful for some digital philological work, but it can hardly be considered a permanent, robust solution for Greek and Latin text analysis. Extensively modifying the source code and developing resources for one of the existing frameworks is also a possibility. That said, the degree of customization that would be necessary for these languages also favors a new solution. This is where the Classical Language Toolkit (CLTK) fits into the digital philological landscape, addressing the desideratum of a complete text analysis pipeline for less-resourced historical languages such as Greek and Latin.[33]

CLTK is an open-source Python framework founded in 2014 by Kyle P. Johnson dedicated to NLP support for historical languages.[34] CLTK has

31 See Kirby (2016, 21–25) for a recent overview of this work. Prosody tagging is also an area of interest in Greek text analysis; see Papakitsos (2010).

32 Pede Certo: (Colombi 2011); available online at http://www.pedecerto.eu (last access 2019.01.31). Macronizer: (Winge 2015); available online at http://alatius.com/macronizer/ (last access 2019.01.31). Winge (2015) deserves special attention here. In addition to its primary discussion of vowel length and macronization, his thesis is also noteworthy as a review of available text analysis components for Latin. In order to complete his thesis work, Winge had to, as I write in the introduction, "cobble together separate, sometimes incompatible tools." His methodology section reveals the substantial challenges he encountered, even if his results demonstrate the excellent work that can be done, once challenges are overcome, with this sort of makeshift pipeline.

33 On less-resourced historical languages, see Piotrowski (2012, 85–86).

34 (Johnson et al. 2019). I have been a contributor to the project, in particular the Latin tools, since 2015.

made progress in recent years collecting corpora for a wide variety of historical languages covering ancient, classical, and medieval Eurasia and building out the basic resources to support these languages across the entire text analysis pipeline. Current offerings include all of the components described in Section 4 with the significant advantage that the components are all available within the same suite of NLP tools. Accordingly, starting from a plaintext file, researchers can tokenize, lemmatize, perform part-of-speech tagging and related morphological analysis, and so on without having to resort to external tools, web applications, or web services. In this respect, CLTK supports Greek and Latin in ways similar to how CoreNLP and NLTK support modern languages.

CLTK aims to meet the criteria of what Steven Krauwer calls the basic language toolkit, or BLARK.[35] The BLARK, according to Krauwer, consists of the "minimal set of language resources that is necessary to do any precompetitive research and education at all" in a given language. This includes but is not limited to 1. a collection of corpora, 2. lexical and grammatical resources, and 3. processing tools. CLTK offers all three:

1. CLTK has Ancient Greek corpora available based on the Perseus Digital Library, the First 1000 Years of Greek, and Lacus Curtius and Latin corpora available based on Perseus, The Latin Library, Lacus Curtius, and the Corpus Grammaticorum Latinorum among others, and has collected related resources such as treebanks from The Ancient Greek and Latin Dependency Treebank.[36]
2. CLTK has developed language models and lexical resources for probabilistic sentence tokenization, part-of-speech tagging, named entity recognition, and more for both languages.
3. As noted above, CLTK currently supports the following "processing tools": sentence tokenization, word tokenization, lemmatization, POS tagging, morphological tagging, basic named entity recognition, prosody tagging, and macronization. There are also modules available for other text analysis and NLP tasks such as syllabification, stemming, and phonological transcription, as well as experimental support for word embeddings using word2vec models trained on large collections of Greek and Latin text.[37]

35 (Krauwer 2003). See also, for Latin specifically, Passarotti (2010).

36 CLTK Corpora can be found in the main project GitHub repository at https://github.com/cltk (last access 2019.01.31). For The Ancient Greek and Latin Dependency Treebank, see Celano et al. (2014).

37 On word2vec and word embeddings, see Mikolov et al. (2013), and for their application to Latin-language text, see Bjerva and Praet (2015).

Development of the project is active; current offerings are continually being refined and new features are being added regularly.[38]

By establishing guidelines for a minimal toolkit, Krauwer hoped to "create better starting conditions for research, education and development in language [. . .] technology" and "facilitate porting of insights and expertise between languages, [. . .] ensuring interoperability and interconnectivity,"[39] all goals of CLTK.

One avenue of CLTK development currently under discussion with project administrators is the implementation of a data structure not unlike CoreNLP's "Annotation" object that would instantiate a complete text analysis pipeline for users upon initialization. With respect to the framework philosophies described in Section 3, CLTK has since its beginning more or less followed the NLTK's "pedagogical" approach; that is, a variety of potential components for each text processing task is made available to users and they learn which is best for their project. But the idea of getting users up and running quickly with a pre-defined pipeline of tried-and-true components, that is something closer to the CoreNLP approach, is certainly attractive, especially for lowering the barrier to entry for digital philological research and encouraging the adoption of CLTK as a general solution for text analysis on classical languages.

Another avenue of CLTK future development concerns the development, where possible, of wrappers for the tools mentioned in Section 4. Wrappers are a type of programming interface that allows you use code from one domain or language inside another without exposing the inner workings of the wrapped code. For example, I can write a Python wrapper for LatMor that uses Python commands to call this lemmatizer without users having to run LatMor themselves. The Python code sends inputs to LatMor (which is not written in Python), runs it as a background process, and stores the lemmatizer's output in a Python data structure for later use in a Python program. In this example, by including a CLTK wrapper for LatMor, we could enable its use as the lemmatization component in an otherwise CLTK-based pipeline. The advantage to users is clear. There is excellent work being done outside of CLTK in Greek and Latin

38 Krauwer also calls for BLARKs to include a "collection of skills" relating to effective use of the corpora and tools; CLTK provides extensive documentation and tutorials to support users in this way.

39 (Krauwer 2003, 1).

digital philology and users should be able to incorporate advances elsewhere in the field with as little friction as possible. As demonstrated in Section 4, a pipeline can always be assembled from disparate, incompatible components, but this is not the optimal situation. Wrappers can provide an intermediate solution through which effective pipelines can be constructed within the CLTK framework with a productive combination of both CLTK components and external components.

5.1 Access to the Classical Language Toolkit

CLTK is freely available and open source, published under the MIT license and hosted at https://github.com/cltk/cltk. More information about the project can be found at https://cltk.org/ and more information about the tools themselves, itemized by language, in the project's documentation at https://docs.cltk.org/en/latest/. Figure 2 shows the project's "pipeline" coverage at the time of writing.

6 Conclusion

Pipelines are an effective way to manage text analysis transformations and annotations and as such they are a defining feature of many NLP frameworks. Yet pipelines are only as good as the components (and the resources that components are based on, such as treebanks, lexica, grammars, and so on) that are available for a given language. At present, leading NLP frameworks like CoreNLP and NLTK support pipelines for a wide array of modern-language research, but options for classical-language research remain limited. This may change over time as Greek and Latin tools are incrementally developed for these frameworks. In the meantime, the Classical Language Toolkit fills the need for a comprehensive text analysis pipeline for these languages.

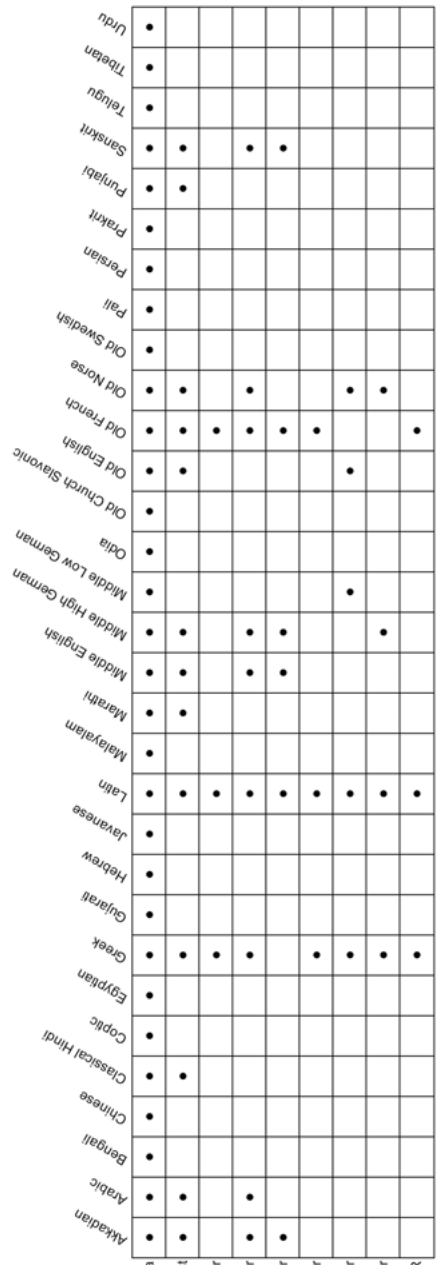

Figure 2: This grid shows the current coverage of the Classical Language Toolkit's basic language resource kit. Included here are all of the historical languages for which CLTK has corpora available. With respect to coverage for text analysis tasks, note that Greek and Latin are the best supported, though there is ongoing, active development across the full range of historical languages.

Bibliography

Bird, S.; Klein, E.; Loper, E. (2015): Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. https://www.nltk.org/book/ (last access 2019.01.31).

Bjerva, J.; Praet, R. (2015): "Word Embeddings Pointing the Way for Late Antiquity". In: Proceedings of the 9th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH). Association for Computational Linguistics, 53–57.

Bodson, A.; Evrard, É. (1966): "Le programme d'analyse automatique du latin". Revue Informatique et Statistique dans les Sciences Humaines 1966:2, 17–46.

Brandolini, G. (n.d.): Latin Parameter File (TreeTagger). http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/latin.par.gz (last access 2019.01.31).

Budassi, M.; Passarotti, M. (2016): "Nomen Omen. Enhancing the Latin Morphological Analyser Lemlat with an Onomasticon." In: Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH). Association for Computational Linguistics, 90–94.

Celano, G.G.A.; Crane, G.; Almas, B. (2014): The Ancient Greek and Latin Dependency Treebank. XML. https://perseusdl.github.io/treebank_data/ (last access 2019.01.31).

Chiarcos, C.; Khait, I.; Pagé-Perron, É.; Schenk, N.; Kandukuri, J.; Fäth, C.; Steuer, J.; McGrath, W.; Wang, J. (2018): "Annotating a Low-Resource Language with LLOD Technology: Sumerian Morphology and Syntax". Information 9:11, 1–16.

Clarke, J.; Srikumar, V.; Sammons, M.; Roth, D. (2012): "An NLP Curator (or: How I Learned to Stop Worrying and Love NLP Pipelines)". In: Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC). European Language Resources Association, 3276–3283.

Colombi, E. (2011): Pede Certo. http://www.pedecerto.eu (last access 2019.01.31).

Crane, G. (1991): "Generating and Parsing Classical Greek". Literary and Linguistic Computing 6:4, 243–245.

Cunningham, H. (2002): "GATE, A General Architecture for Text Engineering". Computers and the Humanities 36:2, 223–254.

de Castilho, R.E.; Gurevych, I. (2014): "A Broad-coverage Collection of Portable NLP Components for Building Shareable Analysis Pipelines". In: Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT. Association for Computational Linguistics and Dublin City University, 1–11.

Eger, S.; Gleim, R.; Mehler, A. (2016): "Lemmatization and Morphological Tagging in German and Latin: A Comparison and a Survey of the State-of-the-art". In: Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC). European Language Resources Association, 23–28.

Eger, S.; Vor der Brück, T.; Mehler, A. (2015): "Lexicon-assisted Tagging and Lemmatization in Latin: A Comparison of Six Taggers and Two Lemmatization Methods". In: Proceedings of the 9th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH). Association for Computational Linguistics, 105–113.

Erdmann, A.; Brown, C.; Joseph, B.; Janse, M.; Ajaka, P.; Elsner, M.; de Marneffe, M.-C. (2016): "Challenges and Solutions for Latin Named Entity Recognition." In: Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LTD4DH). The COLING 2016 Organizing Committee, 85–93.

Ferrucci, D.; Lally, A. (2004): "UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment". Natural Language Engineering 10, 327–348.

Johnson, K.P.; Hollis, L.; Burns, PJ.; CLTK Development Community (2019): CLTK: The Classical Language Toolkit. Python. https://cltk.org (last access 2019.01.31).

Kirby, T. (2016): A Computational Method for Comparative Greek and Latin Prosimetrics. Thesis. Sarasota, FL: New College of Florida.

Krauwer, S. (2003): "The Basic Language Resource Kit (BLARK) as the First Milestone for the Language Resources Roadmap". In Proceedings of International Conference on Speech and Computer (SPECOM 2003). Moscow State Linguistic University, 8–15.

Loper, E.; Bird, S. (2002): "NLTK: The Natural Language Toolkit". In: Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, Volume 1. Stroudsburg, PA: Association for Computational Linguistics, 63–70.

Manning, C.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, S.; McClosky, D. (2014): "The Stanford CoreNLP Natural Language Processing Toolkit". In: Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Association for Computational Linguistics, 55–60.

Marciniak, T.; Strube, M. (2005): "Beyond the Pipeline: Discrete Optimization in NLP". In: Proceedings of the Ninth Conference on Computational Natural Language Learning. Stroudsburg, PA: Association for Computational Linguistics, 136–143.

Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. (2013): "Efficient Estimation of Word Representations in Vector Space". arXiv preprint arXiv:1301.3781.

Ouvrard, Y. (2010): "Collatinus, lemmatiseur et analyseur morphologique de la langue latine." Études de linguistique appliquée 158:2, 223–230.

Papakitsos, E.C. (2010): "Computerized Scansion of Ancient Greek Hexameter". Literary and Linguistic Computing 26:1, 57–69.

Passarotti, M. (n.d.): Latin IT Parameter File (TreeTagger). http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/latinIT.par.gz (last access 2019.01.31).

Passarotti, M. (2010): "Leaving Behind the Less-Resourced Status. The Case of Latin through the Experience of the Index Thomisticus Treebank". In: 7th SaLTMiL Workshop on Creation and Use of Basic Lexical Resources for Less-Resourced Languages (LREC). European Language Resources Association, 27–32.

Passarotti, M.; Litta, E.; Budassi, M.; Ruffolo, P. (2017): "The Lemlat 3.0 Package for Morphological Analysis of Latin". In: Proceedings of the NoDaLiDa 2017 Workshop on Processing Historical Language. Linköping University Electronic Press, 24–31.

Piotrowski, M. (2012): Natural Language Processing for Historical Texts. San Rafael, CA: Morgan and Claypool.

Piperidis, S. (2012): "The META-SHARE Language Resources Sharing Infrastructure: Principles, Challenges, Solutions". In: Proceedings of the 8th International Conference on Language Resources and Evaluation. European Languages Resources Association (ELRA), 36–42.

Schmid, H. (1994): "Probabilistic Part-of-Speech Tagging Using Decision Trees". In: International Conference on New Methods in Language Processing. Stroudsburg, PA: Association for Computational Linguistics, 44–49.

Simon, R.; Barker, E.; Isaksen, L.; de Soto Cañamares, P. (2017): "Linked Data Annotation Without the Pointy Brackets: Introducing Recogito 2". Journal of Map & Geography Libraries 13:1, 111–132.

Springmann, U.; Schmid, H.; Najock, D. (2016): "LatMor: A Latin Finite-State Morphology Encoding Vowel Quantity". Open Linguistics 2:1.
Vatri, A.; McGillivray, B. (n.d.): Ancient Greek Parameter File (TreeTagger). http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/ancient-greek.par.gz (last access 2019.01.31).
Verhagen, M.; Suderman, K.; Wang, D.; Ide, N.; Shi, C.; Wright, J.; Pustejovsky, J. (2016): "The LAPPS Interchange Format". In: Revised Selected Papers of the Second International Workshop on Worldwide Language Service Infrastructure. Cham: Springer, 33–47.
Wachsmuth, H. (2015): Text Analysis Pipelines: Towards Ad-hoc Large-Scale Text Mining. New York: Springer.
Whitaker, W. (1993): Words. http://archives.nd.edu/whitaker/wordsdoc.htm (last access 2019.01.31).
Winge, J. (2015): Automatic Annotation of Latin Vowel Length. Bachelor's Thesis in Language Technology. Uppsala University: Department of Linguistics and Philology.
Zeldes, A.; Schroeder, C.T. (2016): "An NLP Pipeline for Coptic". In: Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities. Association for Computational Linguistics, 146–155.