The Classical Language Toolkit: An NLP Framework for Pre-Modern Languages

Kyle P. Johnson
Accenture
kyle@kyle-p-johnson.com

Patrick J. Burns
Department of Classics
University of Texas at Austin
patrick.burns@austin.utexas.edu

John Stewart
Amplify
johnstewart@aya.yale.edu

Todd G. Cook
Appen
todd.g.cook@gmail.com

Clément Besnier
clem@clementbesnier.fr

William J. B. Mattingly
Data Science Lab
Smithsonian Institution
wma229@g.uky.edu

Abstract

This paper announces version `1.0` of the Classical Language Toolkit (CLTK), an NLP framework for pre-modern languages. The vast majority of NLP, its algorithms and software, is created with assumptions particular to living languages, thus neglecting certain important characteristics of largely non-spoken historical languages. Further, scholars of pre-modern languages often have different goals than those of living-language researchers. To fill this void, the CLTK adapts ideas from several leading NLP frameworks to create a novel software architecture that satisfies the unique needs of pre-modern languages and their researchers. Its centerpiece is a modular processing pipeline that balances the competing demands of algorithmic diversity with pre-configured defaults. The CLTK currently provides pipelines, including models, for almost 20 languages.

1 Introduction

Pre-modern (or historical) languages are linguistically no different than those with speakers living today. Differences, however, manifest in how pre-modern languages are preserved, to what extent they are preserved, how they may be analyzed, and the ends to which they are studied. NLP is comprised of "computational techniques for the purpose of learning, understanding, and producing human language content" (Hirschberg and Manning, 2015, 261). In principle, such techniques may be applied to pre-modern languages. But because NLP, its algorithms and software, presumes living languages, there remains a significant void for NLP for pre-modern languages.

The Classical Language Toolkit (CLTK) is a Python library that borrows ideas from state-of-the-art NLP software, in order to cater to the particular needs of pre-modern languages and their researchers.[1] Its centerpiece is a modular processing pipeline that balances the competing demands of algorithmic diversity with pre-configured defaults. The CLTK currently provides pipelines, including models, for almost 20 languages. This architecture allows for relatively easy customization of currently available pipelines to new languages.

1.1 NLP for Pre-modern Languages

The authors adopt the term *pre-modern* to encompass the ISO 639-3 definitions of *ancient* (whose speakers died over 1,000 years ago), *extinct* (speakers who died within the last 200–300 years), and *historic* (distinct antecedents to living languages) (SIL International). The CLTK aims to treat all such languages, as they survive in written texts, from the 33rd century B.C. (Sumerian) up until the start of the A.D. 19th century.[2]

Pre-modern languages have traits distinguishing them from living languages, including:

- **A finite corpus**: Since native speakers no longer generate new texts, corpora may be too small for some machine learning algorithms, thus requiring rules-based or hybrid

[1] `http://cltk.org`. Begun in 2014, v. `0.1` was a collection of user-submitted NLP algorithms, plus models, for about a dozen pre-modern languages. In this `1.0` release, the CLTK offers a standard API and pre-configured processing pipelines. Burns et al. (2019) contains some earlier history and concepts behind v. `0.1`. The MIT-licensed code is available in version control (`https://github.com/cltk/cltk`) and packaged on PyPI (with `pip install cltk`).

[2] This cutoff date need not be absolute, as the date of introduction of the printing press may be taken into consideration. The press, which spread asynchronously, normalizes orthography and reduces copyist errors (Eisenstein, 1979, 181–225), thus obviating need for some of the CLTK's tools. As orthography stabilizes, coming closer to contemporary usage, living-language NLP becomes increasingly tractable. The Chinese movable type press (A.D. 11th century) could be considered an exception, though modern metal typefaces, with attendant productivity gains, were not applied to Chinese texts until the mid-19th century (Wilkinson, 2000, 451–453). The Sumerian date comes from (Michalowski, 2004, 19).

Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations, pages 20–29, August 1st - August 6th, 2021.

approaches. In some cases, a language's corpus may be small enough that it can be fully annotated.[3]

- **Variation**: Corpora of pre-modern languages are likely to demonstrate greater variation than living languages. This may include non-standardized orthography, regional dialects, and temporal language change (over spans of hundreds and even thousands of years).[4]

- **Limited resources**: Interest in pre-modern languages is largely scholarly or religious, meaning less funding from government and industry for the creation of resources such as text corpora, treebanks, and lexica.

These three differences spur the need for NLP specific to pre-modern languages.

1.2 Researchers of Pre-modern Languages

Researchers of pre-modern languages have concerns that are likely *philological*, *linguistic*, or *pedagogical*. Philology is an approach to pre-modern writing that focuses on the historical origins of texts; it is comparative as well as genealogical in nature (Turner, 2014, x). Historical linguists study diachronic change in a language itself, as opposed to philologists' focus upon written language.[5] Educators have unique concerns, too, including foremost that students generally do not learn by speaking and that they begin studying difficult, original texts within a year of study. In the classroom, a high premium is put upon sight translation, which is accomplished by the sub-tasks of identifying words' parts-of-speech, grammatical constructions, and lexical headwords.[6] These three objectives may find some representation among users of living-language NLP,[7] however they are not significant stimuli to industrial and governmental research.

1.3 Previous Work

Two software architectural patterns, the *framework* and the *pipeline*, are most relevant to the CLTK's design.

As NLP matured in the early 2000's, frameworks (or *toolkits*) emerged with the purpose of making the technology easier for non-specialists to use. To this end, these frameworks generally have documentation friendly for beginners, value diversity in algorithms, treat multiple languages, provide data sets, help with text preprocessing, and provide pre-trained models.[8] Of these characteristics, the CLTK especially values multilingual and multi-algorithmic NLP, the latter of which being necessary to accommodate the varying state of data sets of pre-modern languages. The CLTK shows some especial similarity to the quanteda library for the R language (Benoit et al., 2018), as it contains novel algorithms yet also "wraps" other NLP libraries.

Several NLP frameworks have popularized the pipeline processing architecture, in which default algorithms (tokenization, POS tagging, dependency parsing, etc.) are run in series upon input text. Algorithms may be added or removed from a default pipeline. Increasingly, frameworks use identical algorithms for every language, without special consideration for a language's nuances.

Aside from the CLTK, NLP tools for pre-modern languages have been uncommon,[9] despite a steady growth of language resources.[10] Pre-modern languages are often low-resource. Low-resource software applications, however, have tended toward transcription[11] and, in the case of en-

[3] As with Gothic, for which the only sizable evidence surviving is a 6th century manuscript containing a 4th century translation of the Bible (Miller, 2019, 1, 8–15), most of which the PROIEL project has annotated (Haug and Jøhndal, 2008).

[4] Sumerian, for example, survived 3,000 years (Michalowski, 2004, 19). Piotrowski (2012, 14–22) introduces the categories of difference (diachronic spelling variation), variance (synchronic spelling variation), and uncertainty (information loss during digital transcription).

[5] On linguists' focus on spoken language change: Hock (1991, 1–10) and Campbell (2013, 1–5); on contrast to philology: Hock (1991, 3–5) and Campbell (2013, 373, 391–392). Philology is fundamentally "intepretation of textual data" (Hock 1991, 5).

[6] See Adams (2016) on the origins of this pedagogy in the English-speaking world.

[7] E.g., for secondary language acquisition (Inniss et al., 2006)

[8] Prominent frameworks include the NLTK (Bird and Loper, 2004), OpenNLP (Apache Software Foundation, 2011), CoreNLP (Manning et al., 2014), spaCy (Honnibal and Johnson, 2015), and Stanza (Qi et al., 2020).

[9] For a previous discussion of NLP pipelines for the CLTK, see Burns (2019). There has been some noteworthy work on how generally pre-modern NLP should be done (Piotrowski, 2012; Köntges et al., 2019; McGillivray et al., 2019); also Zeldes and Schroeder (2016), a Python library for Coptic.

[10] Treebanks exist for twelve Indo-European languages according to the PROIEL annotation standards (Haug and Jøhndal, 2008; Eckhoff and Berdicevskis, 2015; Bech and Eide, 2014); texts also for Greek and Latin (Celano et al., 2014), Sanskrit (Hellwig et al., 2020), Cuneiform (Sumerian, Akkadian, etc.) (Englund, 2016), historical Arabic (Belinkov et al., 2016), and Classical Chinese (Lee and Kong, 2012; Yasuoka, 2019).

[11] E.g., Brugman et al. (2004); Ulinski et al. (2014).

dangered languages, language preservation.[12] An interesting exception may be UralicNLP (Hämäläinen, 2019), which provides algorithms intended for relatively small data sets in Finnish and related languages.

2 System Design

An NLP pipeline within a framework architecture standardizes I/O while preserving algorithmic diversity. The CLTK should provide:

- **Modular processing pipelines**: Each language should come with a pre-configured pipeline set to defaults expected by most users. A user should be able to modify, replace, and add processes to a pipeline. Pipelines may be adjusted for new languages.
- **Diversity of algorithms**: When there are several popular ways researchers perform a particular process (e.g., tagging entities with a word list or a neural network), the CLTK should support them both. Due to limited language resources, such as digitized texts and treebanks, machine learning at times may not be tractable (and if so, then only certain algorithms).[13] While rules-based approaches often do not adapt to the dynamism of living languages, they can perform well in restricted tasks within narrow domains.[14]
- **Standard I/O**: To optimize user productivity and facilitate scholarly communication, an API should accept standard input for all human languages. Likewise, when linguistically justified, outputs should be expressed using data structures and representations that are shared across languages.
- **Model management**: The project must provide models for every pipeline.

[12]E.g., Katinskaia et al. (2017); Buszard-Welcher (2018).

[13]For example, surviving literary Ancient Greek texts, from c. 800 B.C. to A.D. 1453, amount to only 65M words (Berkowitz and Squitier, 1990). By contrast, the original English-language BERT was trained on 3,300M tokens (Devlin et al., 2019, 5). (Nevertheless, a BERT model has been made for the Latin language with 643M tokens (Bamman and Burns, 2020, 2).) On small historical corpora, Hamilton et al. (2016) demonstrates benefits of SVD word embeddings over word2-vec.

[14]For example, the CLTK's meter scanners for Latin poetry (`cltk/prosody/lat/verse.py`).

```
>>> from cltk import NLP
>>> cltk_nlp = NLP(language="lat")
⚡ CLTK version '1.0.14'.
Pipeline for language 'Latin' (ISO:
↪   'lat'): `LatinNormalizeProcess`,
↪   `LatinStanzaProcess`,
↪   `LatinEmbeddingsProcess`,
↪   `StopsProcess`,
↪   `LatinNERProcess`,
↪   `LatinLexiconProcess`.
>>> text = "Marcus Cato, ortus
↪   municipio Tusculo adulescentulus,
↪   priusquam honoribus operam daret,
↪   versatus est in Sabinis, quod
↪   ibi heredium a patre relict um
↪   habebat."
>>> cltk_doc =
↪   cltk_nlp.analyze(text=text)
>>> print(cltk_doc.tokens[:12])
['Marcus', 'Cato', ',', 'ortus',
↪   'municipio', 'Tusculo',
↪   'adulescentulus', ',',
↪   'priusquam', 'honoribus',
↪   'operam', 'daret']
>>> print(cltk_doc.pos[:12])
['PROPN', 'PROPN', 'PUNCT', 'NOUN',
↪   'NOUN', 'NOUN', 'ADJ', 'PUNCT',
↪   'ADV', 'NOUN', 'NOUN', 'VERB']
>>> print(cltk_doc.words[11].string)
daret
>>> print(cltk_doc.words[11].pos)
POS.verb
>>> print(cltk_
↪   doc.words[11].features)
{Aspect: [imperfective], Mood:
↪   [subjunctive], Number:
↪   [singular], Person: [third],
↪   Tense: [imperfect], VerbForm:
↪   [finite], Voice: [active]}
```

Code Block 1: Example of `NLP()` (3.1) processing the first sentence of Cornelius Nepos' *M. Porcius Cato*.

3 Architecture and Usage

The CLTK has one primary interface, `NLP()`, and five custom data types: When a user calls `NLP.analyze()`, it outputs a `Doc`, which contains all processed information. At `Doc.words` is a list of `Word` objects, each of which contains token-level information added by each `Process`. A `Pipeline` contains a list of `Process` objects for a given language.

3.1 NLP()

The CLTK's `NLP()` class offers a common interface for all languages, for which a pipeline of NLP algorithms is called. Calling `analyze()`, the class's only public method, triggers each `Process` in succession. The CLTK executes the algorithms and returns a `Doc` object. Code Block 1

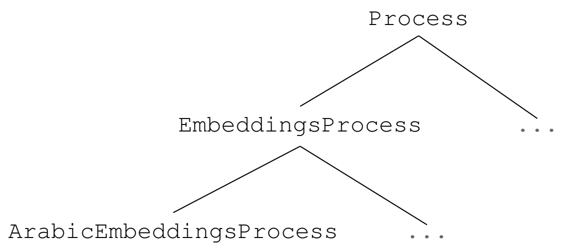

Figure 1: Illustration of the inheritance of `Process` (3.2) objects.

illustrates its use.[15]

3.2 `Process`

An algorithm in the CLTK may be called directly or wrapped in a `Process` that is incorporated into in a `Pipeline`. Each of the following classes, which inherit from `Process`, keep the project's algorithms organized according the kind of NLP they contain (Figure 1).[16]

- **`NormalizeProcess`**: Reads `Doc.raw`, then does Unicode normalization and other text transformation as required per language; outputs to `Doc.normalized_text`.
- **`TokenizationProcess`**: Normally the first `Process` run, splits input string into word tokens; sets string value at `Word.string`.
- **`SentenceProcess`**: Determines sentence boundaries and sets integer at `Word.index_sentence`.
- **`StopsProcess`**: Checks whether a token is contained within a stopword list; adds Boolean value at `Word.stop`.
- **`LemmatizationProcess`**: Reads `Word.string`, and perhaps other contextual information, then sets value at `Word.lemma`.[17]
- **`MorphologyProcess`**: Determines morphology and writes word class (noun, verb, etc.) and features (case, tense, etc.).[18] Values output by morphological taggers, before being set at `Word.pos` and `Word.features`, are normalized to custom CLTK data types that model the annotations of the Universal Dependencies project (see 3.4.3).
- **`DependencyProcess`**: Outputs results of a dependency grammar parser at `Word.dependency_relation` and `Word.governor`.[19]
- **`NERProcess`**: Determines whether a token is a named entity and, if so, what kind; sets string value at `Word.named_entity`.
- **`EmbeddingsProcess`**: Fetches word embedding from a language model; sets array at `Word.embedding`.[20]
- **`PhonologyProcess`**: Ascertains phonological properties of a word (specifically with the inheriting `PhonologicalTranscriptionProcess`) and then reconstructs a phonetic representation in IPA; sets output at `Word.phonetic_transcription`.[21]
- **`ProsodyProcess`**: Scans input strings and outputs scans of their poetic meter.[22]
- **`StemmingProcess`**: Writes a token's stem to `Word.stem`.[23]
- **`WordNetProcess`**: Queries WordNet and writes a word's synset to `Word.synsets`.[24]
- **`LexiconProcess`**: Matches `Word.lemma` to a dictionary's headword and writes to `Word.definition`.
- **`StanzaProcess`**: A `Process` has been created for Stanza because of its usefulness

[15] Text and translation from Rolfe (1984, 282–283): "Marcus Cato, born in the town of Tusculum, in his early youth, before entering on an official career, lived in the land of the Sabines, since he had there an hereditary property, left him by his father."

[16] See Appendix for how the actual code is organized.

[17] Previous work on CLTK lemmatization documented at Burns (2020).

[18] The CLTK relies on Stanza for morphological parsing for Chinese, Coptic, Gothic, Greek, Latin, Old Church Slavonic, and Old French. See also `StanzaProcess`. Other software, however, may be used, as in the case of Akkadian (`cltk/morphology/akk.py`).

[19] At time of publication, the CLTK uses the Stanza project's pretrained models with `StanzaProcess`. In the future, custom-trained models (e.g., with spaCy or Stanza) will be wrapped by `DependencyProcess`. See also section 3.4.4 for post-processing the flat `Doc.words` into a tree.

[20] Using fastText embeddings for Arabic, Aramaic, Gothic, Latin, Old English, Pali, and Sanskrit (Bojanowski et al., 2016); using NLPL for Ancient Greek and Old Church Slavonic (`http://vectors.nlpl.eu`).

[21] Subclassed `SyllabifierProcess` is also available for dividing words into a list of syllable strings; sets output at `Word.syllables`.

[22] Currently available for Greek, Latin, Middle High German, and Old Norse. Prose analysis of Latin clausulae also available (Keeline and Kirby, 2019).

[23] Akkadian, Latin, Middle English, Middle High German, and Old French.

[24] See Short for Latin WordNet API; Ancient Greek and Sanskrit WordNets are under development.

```python
from dataclasses import dataclass,
↪ field
from typing import List, Type
from cltk.core.data_types import
↪ Language, Pipeline, Process
from cltk.languages.utils import
↪ get_lang

@dataclass
class LatinPipeline(Pipeline):
    """Default ``Pipeline`` for
    ↪ Latin."""
    description: str = "Pipeline for
    ↪ the Latin language"
    language: Language =
    ↪ get_lang("lat")
    processes: List[Type[Process]] =
    ↪ field(
        default_factory=lambda: [
            LatinNormalizeProcess,
            LatinStanzaProcess,
            LatinEmbeddingsProcess,
            StopsProcess,
            LatinNERProcess,
            LatinLexiconProcess,
        ]
    )
```

Code Block 2: Example of `LatinPipeline` (3.3) and the processes declared within it; defined at `cltk/languages/pipelines.py`.

for seven languages (see ft. 18).

3.3 `Pipeline`

A language has one `Pipeline` defining a list of `Process` objects, as illustrated in Code Block 2. The objects within `Pipeline.processes` are looped over when called by `NLP.analyze()`. Each time, a `Doc` is sent into the `Process` and a new `Doc`, now with an updated `Doc.words`, is produced. These algorithms are invoked by default, though a user may override them by declaring his own `Pipeline` and passing it to `NLP()`. At time of publication, 19 languages have pre-configured pipelines.[25]

3.4 `Doc`

The `NLP.analyze()` method returns a `Doc` object that contains all information generated by the `Pipeline` (example at Code Block 1). Most of this information is stored within a list of `Word` objects at `Doc.words`, which may be accessed directly or by helper methods, such as `Doc.tokens` (returning a list of token strings) and `Doc.embeddings` (a list of arrays). When these access methods are not enough, a user may post-process the `Doc` and add attributes to it or the `Word` objects within.

```python
>>> print(cltk_doc.words[11])
Word(index_char_start=None,
↪ index_char_stop=None,
↪ index_token=11, index_sentence=0,
↪ string='daret', pos=verb,
↪ lemma='do', stem=None,
↪ scansion=None,
↪ xpos='J3|modB|tem2|gen6',
↪ upos='VERB',
↪ dependency_relation='root',
↪ governor=-1, features={Aspect:
↪ [imperfective], Mood:
↪ [subjunctive], Number:
↪ [singular], Person: [third],
↪ Tense: [imperfect], VerbForm:
↪ [finite], Voice: [active]},
↪ category={F: [neg], N: [neg], V:
↪ [pos]}, stop=False,
↪ named_entity=False,
↪ syllables=None,
↪ phonetic_transcription=None,
↪ embedding=array([-1.2459e-01,
↪ ...], dtype=float32),
↪ definition="dō\n\n (old subj.
↪ duis, duit, duint, etc.), dedī,
↪ datus, are \n1 DA-, \nto hand
↪ over, deliver, give up, render,
↪ furnish, pay, surrender")
```

Code Block 3: Example of processed information contained within a `Word` (3.4.1) object. Continues from Code Block 1.

3.4.1 `Word`

`Word` stores all token information. Code Block 3 shows some of what a `Word` object may contain.

3.4.2 `Language`

The module `cltk/languages/glottolog.py` contains 219 `Language` objects, each of which contains information about a pre-modern language that is, or should be, covered by the CLTK.[26] Code Block 4 shows how to retrieve a `Language` with a three-letter ISO code. Each

[25] Akkadian ("akk"), Arabic ("arb"), Aramaic ("arc"), Classical Chinese ("lzh"), Coptic ("cop"), Gothic ("grc"), Hindi ("hin"), Latin ("lat"), Middle High German ("gmh"), Old English ("ang"), Middle English ("enm"), Old French ("frm"), Old Church Slavonic ("chu"), Old Norse ("non"), Pali ("pli"), Panjabi ("pan"), and Sanskrit ("san").

[26] Language definitions and data provided by Glottolog, a database of the world's languages (Hammarström et al., 2021). These 219 languages are those falling within the definition of pre-modern (discussed at 1.1), plus some with significant continuity between pre-modern and contemporary written forms: Standard Arabic, nine South Asian languages (Bengali, Hindi, etc.), Western Farsi, and Coptic.

```
>>> from cltk.languages.utils import
↪   find_iso_name
>>> print(find_iso_name("Latin"))
['lat']
>>> from cltk.languages.utils import
↪   get_lang
>>> print(get_lang("lat"))
Language(name='Latin',
↪   glottolog_id='lati1261',
↪   latitude=41.9026,
↪   longitude=12.4502, dates=[],
↪   family_id='indo1319',
↪   parent_id='impe1234',
↪   level='language',
↪   iso_639_3_code='lat', type='a')
```

Code Block 4: Example of a `Language` (3.4.2) object for Latin (ISO code `"lat"`).

`Pipeline` references these classes (see Code Block 2).

3.4.3 `MorphosyntacticFeature` and `MorphosyntacticFeatureBundle`

Beyond the categorical information at `Word.pos`, a language's `Pipeline` adds complete morphology at the `Word.features` accessor (see Code Block 5). The sometimes arbitrary output strings of morphological taggers ("indicative," "Indic.," etc.) are mapped to these specific CLTK classes (inheriting from `MorphosyntacticFeature`) that represent all features defined by version 2 of the Universal Dependencies project.[27] Hence, different taggers resolve to a common annotation schema.

3.4.4 `DependencyTree`

The CLTK uses the "built-in" `xml` library to make trees for modeling dependency parses. A `Word` is mapped into a `Form`, then `ElementTree` is used to organize these into a `DependencyTree` (see Code Block 6).

3.5 `FetchCorpus`

Git repositories host models developed by CLTK contributors.[28] When the software cannot find a required model, `FetchCorpus` is invoked to download the required dependency and put it within the appropriate directory at `~/cltk_data/`.[29]

[27]Annotation guidelines at Universal Dependencies (2016) and CLTK objects at `cltk/morphology/universal_dependencies_annotations.py`.

[28]All CLTK models are stored on GitHub at: `https://github.com/cltk/?q=model`.

[29]A language-specific Git repository is available for most languages, e.g., `"lat_models_cltk"` at the URI `https://github.com/cltk/lat_models_cltk.git`. Users may share private or non-official repositories by defining them at `~/cltk_data/distributed_corpora.yaml`.

```
>>> print(cltk_doc.words[11].featur
↪   es)
{Aspect: [imperfective], Mood:
↪   [subjunctive], Number:
↪   [singular], Person: [third],
↪   Tense: [imperfect], VerbForm:
↪   [finite], Voice: [active]}
>>> print(type(cltk_doc.wor
↪   ds[11].features))
<class 'cltk.morphology.mor
↪   phosyntax.MorphosyntacticFeatur
↪   eBundle'>
>>> print(cltk_doc.words[11].featur
↪   es["Aspect"][0])
Aspect.imperfective
>>> print(cltk_doc.words[11].featur
↪   es["Mood"][0])
Mood.subjunctive
```

Code Block 5: Example of `MorphosyntacticFeature` and `MorphosyntacticFeatureBundle` (3.4.3). Continues from Code Block 3.

```
>>> from cltk.dependency.tree import
↪   DependencyTree
>>> a_tree = DependencyTree.to_tree
↪   (cltk_doc.sentences[0])
>>> print(a_tree.get_dependencie
↪   s()[:5])
[nsubj(daret_11, Marcus_0),
↪   nsubj(daret_11, Cato_1),
↪   nsubj(daret_11, ortus_3),
↪   nsubj(daret_11, Marcus_0),
↪   nsubj(daret_11, Cato_1)]
```

Code Block 6: Example of `DependencyTree` (3.4.4). Continues from Code Block 1.

4 Conclusion and Future Work

The architecture of the CLTK v. `1.0` has an engineering rigor necessary to model the world's several hundred pre-modern languages. Currently, it serves the basic, and several more advanced, needs of researchers for 19 languages.

Software alone, however, is not sufficient. The CLTK lacks formal evaluations of its models' accuracies. At time of publication, most `Process` definitions wrap models trained by upstream projects (e.g., Stanza). While these projects report accuracies respective to their training sets (i.e., with cross-validation), they do not provide evaluations against outside benchmarks. Unfortunately, such benchmarks do not yet exist for pre-modern languages, with the exception of the recent Sprugnoli et al.

(2020) for Latin. To remedy this problem, the authors will focus upon the following areas:

- to create evaluation benchmarks for each NLP task, for each language;
- to make a `TrainingPipeline`, similar to the inference `Pipeline`, that would standardize the training of new models;
- to normalize duplicative treebanks;[30]
- and to develop Internet infrastructure for training and hosting models;

These efforts will improve scientific procedure for pre-modern NLP.

Another initiative involves experimentation with transfer learning, along the lines of Multilingual BERT (Pires et al., 2019), training on all surviving pre-modern texts. Because languages are related and because texts, even in different languages, often share entities, information sharing may prove felicitous.[31]

The pre-modern world, its languages and peoples, was deeply networked.[32] The CLTK is a comprehensive collection of NLP technologies to support the study of this history.

Acknowledgments

The authors owe a special dept of gratitude to all of the CLTK's contributors.[33] They also thank early readers of this manuscript: Neil Coffee, Greg Crane, Jonathan Everett, Luke Hollis, Thomas Keeline, Leonard Muellner, Nigel Nicholson, Monica Park, Michael Piotrowski, Marco Romani, and William M. Short. The project's name is a play on the Natural Language Toolkit (NLTK), on which v. `0.1` heavily relied. The CLTK logo of a Phoenician *aleph* (or *ʾālep*, 𐤀), being the first letter of the first alphabet, was created by Pierre-Marie Pédrot.[34]

[30] For example, Universal Dependencies hosts five different, and to various degrees incompatible, Latin treebanks. The largest is 450,000 tokens, though adding the other four would bring the count close to 1,000,000. Ancient Greek also has duplicative treebanks (each at about 200,000 tokens).

[31] Considerations include use of original orthography versus normalizing to orthographic or phonetic transliteration.

[32] Several studies on trans-cultural diffusion across Eurasia: Beckwith (2009); Frankopan (2015).

[33] `https://github.com/cltk/cltk/graphs/contributors`.

[34] `https://commons.wikimedia.org/wiki/File:PhoenicianA-01.svg`.

References

M. Adams. 2016. *Teaching Classics in English Schools, 1500–1840*. Cambridge Scholars Publishing, Newcastle upon Tyne.

Apache Software Foundation. Apache OpenNLP [online]. 2011.

David Bamman and Patrick J. Burns. 2020. Latin BERT: A contextual language model for classical philology.

Kristin Bech and Kristine Eide. The ISWOC corpus [online]. 2014.

Christopher I. Beckwith. 2009. *Empires of the Silk Road: A History of Central Eurasia from the Bronze Age to the Present*. Princeton University Press, Princeton.

Yonatan Belinkov, Alexander Magidow, Maxim Romanov, Avi Shmidman, and Moshe Koppel. 2016. Shamela: A large-scale historical Arabic corpus. In *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*, pages 45–53, Osaka, Japan. The COLING 2016 Organizing Committee.

Kenneth Benoit, Kohei Watanabe, Haiyan Wang, Paul Nulty, Adam Obeng, Stefan Müller, and Akitaka Matsuo. 2018. quanteda: An R package for the quantitative analysis of textual data. *Journal of Open Source Software*, 3(30):774.

Luci Berkowitz and Karl A. Squitier. 1990. *Thesaurus Linguae Graecae: Canon of Greek Authors and Works*, 3rd edition. Thesaurus Linguae Graecae. Oxford University Press, New York.

Steven Bird and Edward Loper. 2004. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions*, page 31. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2016. Enriching word vectors with subword information. *CoRR*, abs/1607.04606.

Hennie Brugman, Albert Russel, and Xd Nijmegen. 2004. Annotating multi-media/multi-modal resources with ELAN. In *The Fourth International Conference on Language Resources and Evaluation*. European Language Resources Association (ELRA), Lisbon.

Patrick J. Burns. 2019. Building a text analysis pipeline for classical languages. In Monica Berti, editor, *Digital Classical Philology: Ancient Greek and Latin in the Digital Revolution*, number 10 in Age of Access? Grundfragen der Informationsgesellschaft, pages 159–176. de Gruyter, Berlin.

Patrick J. Burns. 2020. Ensemble lemmatization with the Classical Language Toolkit. *Studi e Saggi Linguistici*, 58(1):157–176.

Patrick J. Burns, Luke Hollis, and Kyle P. Johnson. 2019. The future of ancient literacy: Classical Language Toolkit and Google Summer of Code. *Classics@*, 17.

Laura Buszard-Welcher. 2018. New media for endangered languages. In Kenneth L. Rehg and Lyle Campbell, editors, *The Oxford Handbook of Endangered Languages*, Oxford Handbooks, chapter 22. Oxford University Press, Oxford.

Lyle Campbell. 2013. *Historical Linguistics*. Edinburgh University Press, Edinburgh.

Giuseppe G. A. Celano, Gregory Crane, and Bridget Almas. Ancient Greek and Latin dependency treebank v.2.1 [online]. 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Hanne Martine Eckhoff and Aleksandrs Berdicevskis. 2015. Linguistics vs. digital editions: The Tromsø Old Russian and OCS treebank. *Scripta & e-Scripta*, 14(15):9–25.

Elizabeth L. Eisenstein. 1979. *The Printing Press as an Agent of Change*. Cambridge University Press, Cambridge.

Robert K. Englund. 2016. The Cuneiform Digital Library Initiative: DL in DH. Microsoft PowerPoint.

Peter Frankopan. 2015. *The Silk Roads: A New History of the World*. Vintage Books, New York.

Mika Hämäläinen. 2019. UralicNLP: An NLP library for Uralic languages. *Journal of Open Source Software*, 4(37):1345.

William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic word embeddings reveal statistical laws of semantic change. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1489–1501, Berlin, Germany. Association for Computational Linguistics.

Harald Hammarström, Robert Forkel, Martin Haspelmath, and Sebastian Bank. Glottolog 4.4 [online]. 2021.

Dag T. T. Haug and Marius L. Jøhndal. 2008. Creating a parallel treebank of the old Indo-European Bible translations. In Caroline Sporleder and Kiril Ribarov, editors, *Proceedings of the Second Workshop on Language Technology for Cultural Heritage Data (LaTeCH 2008)*, pages 27–34. European Language Resources Association (ELRA), Marrakech.

Oliver Hellwig, Salvatore Scarlata, Elia Ackermann, and Paul Widmer. 2020. The treebank of Vedic Sanskrit. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5137–5146, Marseille, France. European Language Resources Association.

Julia Hirschberg and Christopher D. Manning. 2015. Advances in natural language processing. *Science*, 349(6245):261–266.

Hans Henrich Hock. 1991. *Principles of Historical Linguistics*, 2nd edition. Mouton de Gruyter, Berlin.

Matthew Honnibal and Mark Johnson. 2015. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal. Association for Computational Linguistics.

Tasha R. Inniss, John R. Lee, Marc Light, Michael A Grassi, George Thomas, and Andrew B. Williams. 2006. Towards applying text mining and natural language processing for biomedical ontology acquisition. In *Proceedings of the 1st International Workshop on Text Mining in Bioinformatics*, pages 7–14.

Anisia Katinskaia, Javad Nouri, and Roman Yangarber. 2017. Revita: A system for language learning and supporting endangered languages. In *Proceedings of the joint workshop on NLP for Computer Assisted Language Learning and NLP for Language Acquisition*, pages 27–35, Gothenburg, Sweden. LiU Electronic Press.

Tom Keeline and Tyler Kirby. 2019. *Auceps syllabarum*: A digital analysis of Latin prose rhythm. *Journal of Roman Studies*, 109:161–204.

Thomas Köntges, Rhea Lesage, Bruce Robertson, Jeannie Sellick, and Lucie Wall Stylianopoulos. 2019. Open Greek and Latin: Digital humanities in an open collaboration with pedagogy. In *Libraries: Dialogue for Change*, Athens. IFLA WLIC.

John Lee and Yin Hei Kong. 2012. A dependency treebank of Classical Chinese poems. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 191–199, Montréal, Canada. Association for Computational Linguistics.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Barbara McGillivray, Jon Wilson, and Tobias Blanke. 2019. Towards a quantitative research framework for historical disciplines. In *CEUR Workshop Proceedings*, volume 2314.

Piotr Michalowski. 2004. Sumerian. In Roger D. Woodard, editor, *The Cambridge Encyclopedia of the World's Ancient Languages*, pages 19–59. Cambridge University Press, Cambridge.

D. Gary Miller. 2019. *The Oxford Gothic Grammar*. Oxford Linguistics. Oxford University Press, Oxford.

Michael Piotrowski. 2012. *Natural Language Processing for Historical Texts*. Number 17 in Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, San Rafael.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is Multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108, Online. Association for Computational Linguistics.

John C. Rolfe. 1984. *Cornelius Nepos*, volume 467 of *The Loeb Classical Library*. Harvard University Press, Cambridge, Mass.

William Michael Short. Latin WordNet [online].

SIL International. ISO 639-3 [online].

Rachele Sprugnoli, Marco Passarotti, Flavio Massimiliano Cecchini, and Matteo Pellegrini. 2020. Overview of the EvaLatin 2020 evaluation campaign. In *Proceedings of LT4HALA 2020 - 1st Workshop on Language Technologies for Historical and Ancient Languages*, pages 105–110, Marseille, France. European Language Resources Association (ELRA).

James Turner. 2014. *Philology: The Forgotten Origins of the Modern Humanities*. Princeton University Press, Princeton.

Morgan Ulinski, Anusha Balakrishnan, Daniel Bauer, Bob Coyne, Julia Hirschberg, and Owen Rambow. 2014. Documenting endangered languages with the WordsEye linguistics tool. In *Proceedings of the 2014 Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 6–14, Baltimore, Maryland, USA. Association for Computational Linguistics.

Universal Dependencies. UD Guidelines V2 [online]. 2016.

Endymion Wilkinson. 2000. *Chinese History: A Manual*. Number 52 in Harvard-Yenching Institute Monograph Series. Harvard University Press, Cambridge, Mass.

Koichi Yasuoka. 2019. Universal dependencies treebank of the Four Books in Classical Chinese. In *DADH2019: 10th International Conference of Digital Archives and Digital Humanities*, pages 20–28. Digital Archives and Digital Humanities.

Amir Zeldes and Caroline T. Schroeder. 2016. An NLP pipeline for Coptic. In *Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 146–155, Berlin, Germany. Association for Computational Linguistics.

A Appendix

The following top-level directories are found at `src/cltk`, within the project's repository.

- **`nlp`**: The main module, contains class `NLP()`
- **`alphabet`**: Manipulate characters of a language's orthographic system
- **`core`**: Custom data types, error handling
- **`corpora`**: Metadata for and preprocessing of specific data sets
- **`data`**: Download CLTK-hosted data sets
- **`dependency`**: Dependency parsing
- **`embeddings`**: Making and loading word embeddings
- **`languages`**: Definition of all pre-modern languages, text snippets for demonstration
- **`lemmatize`**: Find lemma for an inflected form
- **`lexicon`**: Find a lemma's definition in a dictionary
- **`morphology`**: Model morphology and syntax with data types from Universal Dependencies
- **`ner`**: Tag named entities (i.e., proper nouns)
- **`phonology`**: Syllabifying and tagging phonemes
- **`prosody`**: Scanning poetic meter
- **`sentence`**: Splitting sentences
- **`stem`**: Create unique stem from inflected form
- **`stops`**: Identify if a token is a stopword
- **`tag`**: Part-of-speech tagging
- **`text`**: Language-specific, extensible text preprocessing

- **`tokenizers`**: Create tokens from an input string
- **`utils`**: Helpers for feature extraction and disk I/O
- **`wordnet`**: Lookup of lemma on available online WordNets