Multiplex Lemmatization with the Classical Language Toolkit

Patrick J. Burns
University of Texas at Austin / Quantitative Criticism Lab
Classical Language Toolkit
First LiLa Workshop: Linguistic Resources & NLP Tools for Latin | 6.3.19

cltk.org
@diyclassics

Slides available at:
http://bit.ly/burns-lila2019-slides

Background

What is the Classical Language Toolkit?

The Classical Language Toolkit (CLTK) is a free and open-source Python package that offers natural language processing (NLP) support for the languages of Ancient, Classical, and Medieval Eurasia.

Language-specific tokenizers, lemmatizers, POS-taggers, morphological parsers, etc. are available, under development, or in the feature-request list. Latin and Greek functionality are currently most complete.

What is the Classical Language Toolkit?

- Open-source community collaborating at https://github.com/cltk
- Founded by Kyle P. Johnson, Classics PhD from NYU and NLP Research Scientist at Accenture
- Academic Advisors: Gregory Crane (Leipzig/Tufts) , Neil Coffee (Buffalo), Peter Meineck (NYU), Leonard Muellner (Brandeis/CHS)

CLTK Goals

- *Low: Good analysis-friendly corpora/datasets for NLP of historical languages (Latin, Ancient/Classical Greek, Egyptian hieroglyphs, Hebrew, Sanskrit, Tibetan, Classical Chinese, etc.)*

CLTK Goals

- Low: Good analysis-friendly corpora/datasets for NLP of historical languages (Latin, Ancient/Classical Greek, Egyptian hieroglyphs, Hebrew, Sanskrit, Tibetan, Classical Chinese, etc.)
- *Medium: Collect & generate linguistic data for quantified classics*

CLTK Goals

- Low: Good analysis-friendly corpora/datasets for NLP of historical languages (Latin, Ancient/Classical Greek, Egyptian hieroglyphs, Hebrew, Sanskrit, Tibetan, Classical Chinese, etc.)
- Medium: Collect & generate linguistic data for quantified classics
- *High: Framework for an integrated study of the ancient world; next generation comparative philology*

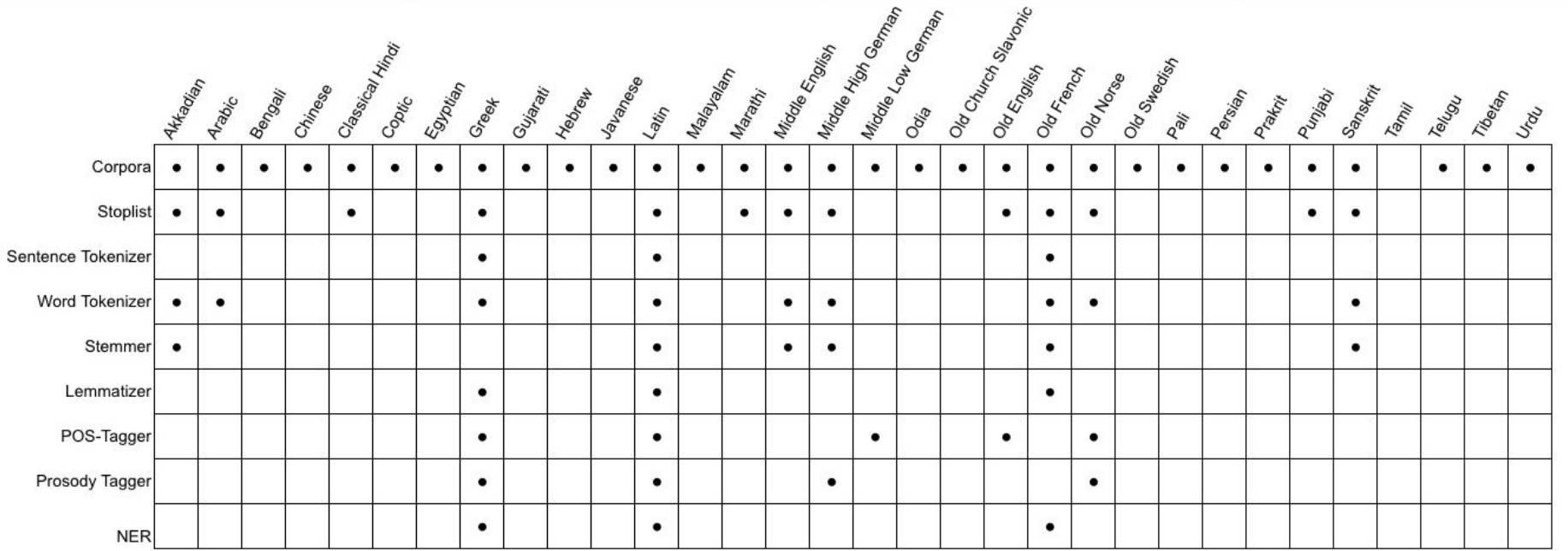
CLTK Stats

- Began 2014
- 2,671 commits at https://github.com/cltk/cltk
- 82 contributors
- 71 watchers, 519 stars, 279 forks
- 67 people, 20 teams
- 107 releases (with Zenodo DOI for every release)
    - Plans for v.1 release
- 89% code coverage
- 2016-2018 Google Summer of Code participating organization

Google Summer of Code 2016

- CLTK work on Backoff Latin Lemmatizer
- Modeled after NLTK Backoff POS Tagger
- Series of trained and rules-based lemmatizers run in sequence
- Can be “tuned” for specific languages

CLTK's BLARK in Progress



Toward a Historical Language BLARK

- CLTK as basic language resource kit (Krauwer 2003)
  - "minimum general text corpus required to be able to do any precompetitive research for the language at all"
  - "collection of basic tools to manipulate and analyse the corpora"
  - "collection of skills that constitute the minimal starting point for the development of a competitive NL/Speech technology industry."

Krauwer, S. 2003. “The Basic Language Resource Kit (BLARK) as the First Milestone for the Language Resources Roadmap.” Proceedings of the 2003 International Workshop on Speech and Computer (SPECOM 2003) : 8-15; cf. also Passarotti in SALTMIL 2010: 29.

CLTK Research

Today's talk draws on some forthcoming papers/work...

- Burns, P.J. "Backoff Tagging as a Philological Method"
- Burns, P.J. "Object-oriented Philology"
- Burns, P.J. 2019. "Building a Text Analysis Pipeline for Classical Languages," In Berti, M. ed. *Digital Classical Philology*. DeGruyter.
- (the code itself; i.e. github.com/cltk/lemmatize/ etc.)

Backoff Lemmatization

NLTK Backoff Tagging

"[Backoff tagging] allows you to chain taggers together so that if one tagger doesn't know how to tag a word, it can **pass the word on to the next backoff tagger**...and so on until there are no backoff taggers left to check"

—Perkins, NLTK 3 Cookbook

Backoff Chain

UnigramTagger with dictionary

↓

UnigramTagger with training data

↓

RegexpTagger

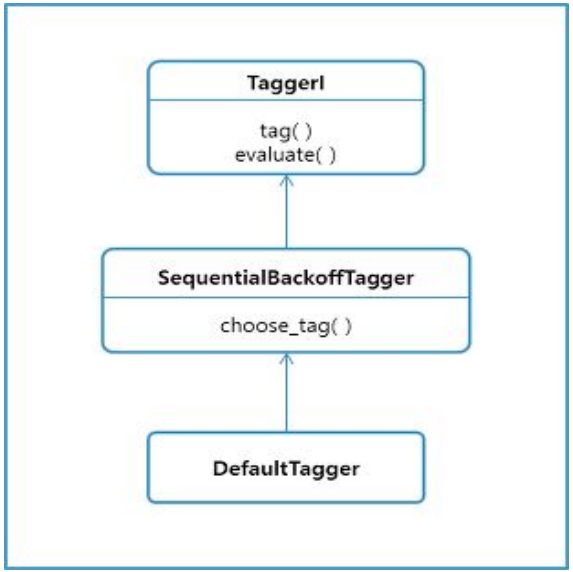
↓

DefaultTagger

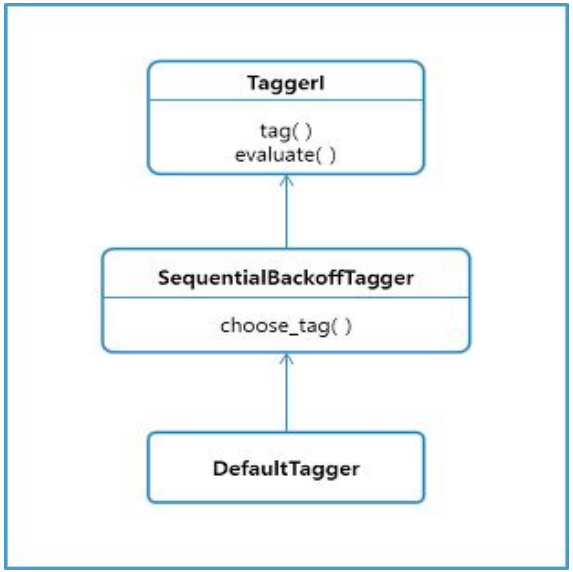
NLTK backoff POS taggers

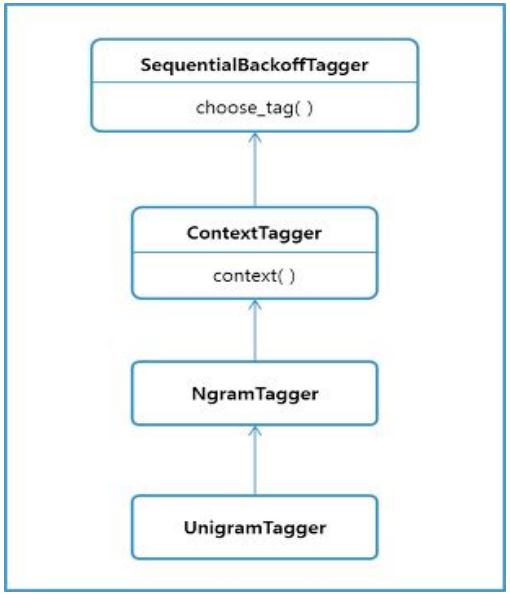
- Default: assigns all tokens the same tag
- Context: assigns tag based on ngrams from training data
- Model: assigns tag based on dictionary of values
- Regex: assigns tag based on pattern matching, esp. endings
- Affix: assigns tag based on prefixes and suffixes

POS Backoff Tagging in NLTK



POS Backoff Tagging in NLTK





Backoff Chain

UnigramLemmatizer with dictionary

↓

UnigramLemmatizer with training data

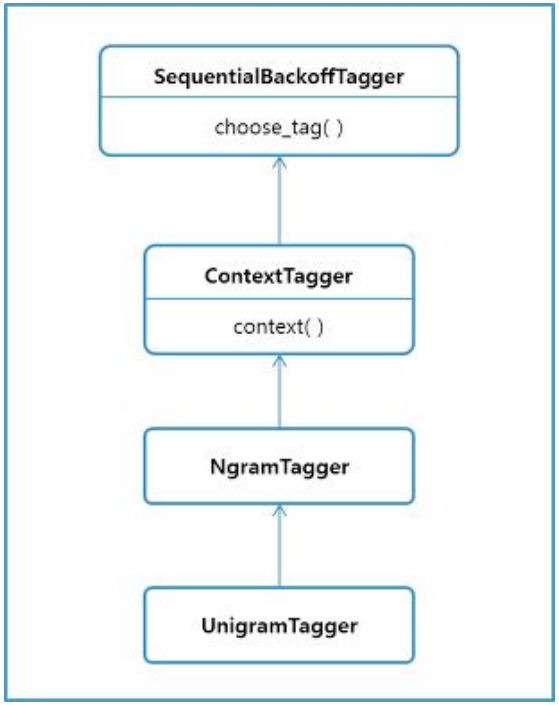
↓

RegexpLemmatizer

↓

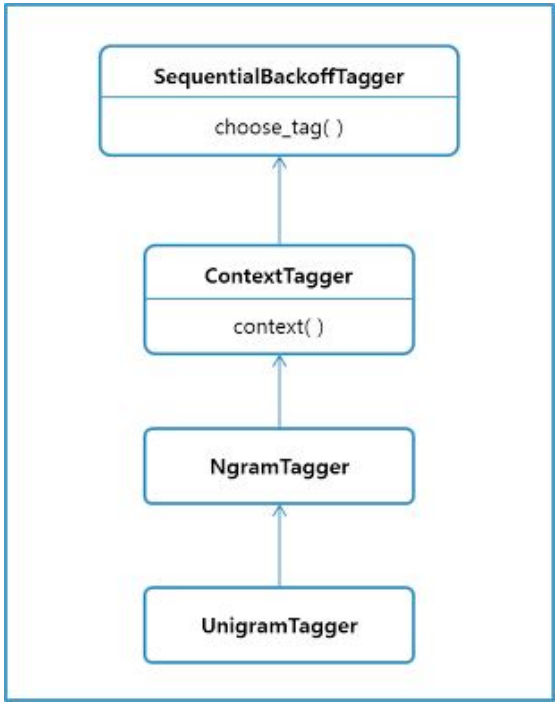
DefaultLemmatizer

Unigram Context Tagger



amor > *amor* (34 times)

Unigram Context Tagger

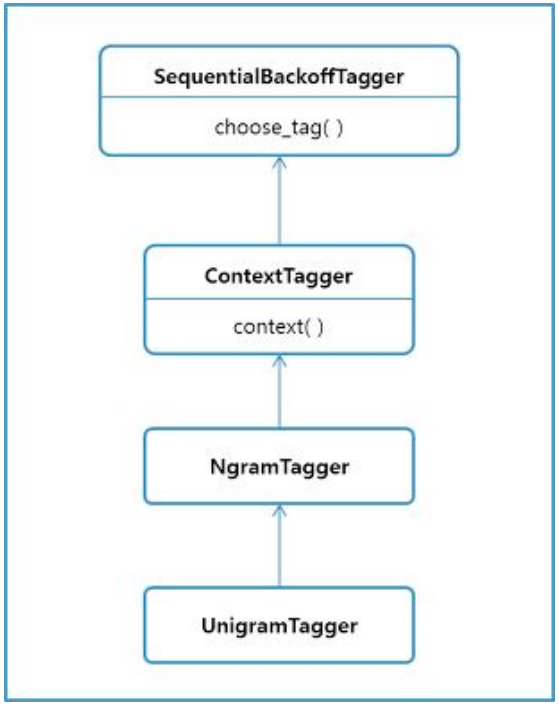


amor > *amor* (34 times)

vs.

amor > *amo* (0 times)

Unigram Context Tagger



amor > *amor* wins

Regex Tagger

492 lines (452 sloc) | 20.4 KB

Raw | Blame | History

```
### Regexps for RomanNumeralLemmatizer

rn_patterns = [(r'(?=^[MDCLXVUI]+$)(?=^M{0,4}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|IU|V?I{0,3}|U?I{0,3})$)', 'NUM'),
               (r'(?=^[mdclxvui]+$)(?=^m{0,4}(cm|cd|d?c{0,3})(xc|xl|l?x{0,3})(ix|iv|iu|v?i{0,3}|u?i{0,3})$)', 'NUM')]

### Misc regexps for Latin; default patterns for use with RegexpLemmatizer
### Based on A&G 277ff. and trial/error
### Needs to be tested more thoroughly for false positives
###
### Ambiguous patterns have been commented out; in progress

latin_misc_patterns = [
    (r'(\w*)abimus', 'o'),

    (r'(\w*)toris', 'tor'),
    (r'(\w*)tori', 'tor'),
    (r'(\w*)torem', 'tor'),
    (r'(\w*)tore', 'tor'),
    (r'(\w*)tores', 'tor'),
    # (r'(\w*)torum', 'tor'),
    (r'(\w*)toribus', 'tor'),

    (r'(\w*)soris', 'sor'),
    (r'(\w*)sori', 'sor'),
    (r'(\w*)sorem', 'sor'),
    (r'(\w*)sore', 'sor'),
```

Ensemble Lemmatization

Disadvantages of backoff method

- Binary decision making
- Backoff chain broken by first selection so...
- Order of backoff lemmatizers is important

Backoff Chain

amor

assuming *amor* present in dictionary; present in training data; and *-or* present in multiple regex patterns

UnigramTagger with dictionary

↓

UnigramTagger with training data

↓

RegexpTagger

Ensemble Chain

amor

assuming *amor* present in dictionary; present in training data; and *-or* present in multiple regex patterns

UnigramTagger with dictionary

50% *amor*, 50% *amo*

↓

UnigramTagger with training data

100% *amor*, 0% *amo*

↓

RegexpTagger

70% *amor* (n.), 20% *amo*, 10% *amor* (v.)

Ensemble Chain

amor

assuming *amor* present in dictionary; present in training data; and *-or* present in multiple regex patterns

amor (n.) 73.33%
amo (v.) 23.33%
amor (v.) 3.33%

Advantages of ensemble method

- Avoid binary decision making (and chain breaking)
- Make use of all available information
  - Unambiguous results from individual taggers
  - Results of all taggers
- Restrict importance of backoff order

Backoff/Ensemble Lemmatization...

...as a Philological Method

Why a “philological” method?

Because of its multipass combination of probabilistic tagging based on existing Latin text, Latin lexical data, and a ruleset based on Latin morphology, the Backoff Lemmatizer can be described as following a philological method.

By this, I mean that the process reflects the **reading, decoding, and disambiguating strategies** of the trained philologist.

An example


Papa Franciscus
@Pontifex_In
Following
Salutem ex corde dicimus pedilusoribus et omnibus aures mentesque praebentibus attentas Certaminibus Mundialibus Sphaeromachiae, quae hodie in Russia incipiunt. Hoc athletarum certamen exoptamus, ut occasio sit concursus et fraternitatis.
Translate Tweet
4:17 AM - 15 Jun 2018
100 Retweets 279 Likes
8
100
279

An example


Papa Franciscus
@Pontifex_In
Following
Salutem ex corde dicimus pedilusoribus et omnibus aures mentesque praebentibus attentas Certaminibus Mundialibus Sphaeromachiae, quae hodie in Russia incipiunt. Hoc athletarum certamen exoptamus, ut occasio sit concursus et fraternitatis.
Translate Tweet
4:17 AM - 15 Jun 2018
100 Retweets 279 Likes
8
100
279

“Philological” backoff chain

lexicon

↓

“endings”

↓

context

“Philological” backoff chain

lexicon ≈ lexicon lemmatizer

↓

“endings” ≈ regex lemmatizer

↓

context ≈ model lemmatizer

Another example: Latin Jabberwocky

JABBERWOCKY

Lewis Carroll (Charles Lutwidge Dodgson)

Gabrobocchia [author unknown]

Est brilgum: tovi slimici
In vabo tererotitant
Brogovi sunt macresculi
Momi rasti strugitant.

Another example: Latin Jabberwocky

Est brilgum: tovi slimici
In vabo tererotitant
Brogovi sunt macresculi
Momi rasti strugitant.

- *est* ⟶ *sum*; *in* ⟶ *in* (100%; dict)
- *strugitant* ⟶ *strugito* (100%; model)
- *slimici* ⟶ *slimicus / slimex* (x% / y%; ensemble)
- *vabo* ⟶ *vabus / vabum* (x% / y%; ensemble w. context; i.e. not *vo, vare, vavi, vatum*)

CLTK Wrappers

Field of Latin Lemmatizers

- CLTK Backoff
- Collatinus
- LemLat
- Latmor
- Morpheus
- TreeTagger
- Whitaker's Words
- more...

Backoff wrappers

```
# Create CollatinusLemmatizer as subclass of NLTK's Sequential Backoff Tagger

from pycollatinus import Lemmatiseur

class CollatinusLemmatizer(DictLemmatizer):
    def __init__(self: object, backoff: object = None):
        """Setup for CollatinusLemmatizer()."""
        self.tagger = Lemmatiseur()
        self.collatinus_dict = {}

        # Use Collatinus to create dictionary for DictLemmatizer
        super().__init__(lemmas=self.collatinus_dict, backoff=backoff, source='Collatinus
output')

    def lemmatize(self: object, tokens: List[str], force_single_tag=False):
        text = " ".join([token.lower() for token in tokens]) # lower necessary?
        results = self.tagger.lemmatise_multiple(text)
        lemmas_ = [list(set([r['lemma'] for r in result])) for result in results]
        lemma_pairs = zip(tokens, lemmas_)
        self.collatinus_dict.update(lemma_pairs)
        return self.tag(tokens)

    def __repr__(self: object):
        if self.source:
            return f'<{type(self).__name__}: {self.source}>'
        else:
            return f'<{type(self).__name__}: {self.repr.repr(self.lemmas)}>'
```

Backoff Wrappers

- Leverage full range of development in digital philology
- Balance language-specific development
  - work with SFST or Ada within Python
- Promote the idea of NLP pipeline
  - i.e. standardize input/output
  - allow for component exchange depending on research question/task

Wrapper comparison

	tokens	treetagger	latmor	collatinus	cltk	combined
0	Nescius	[nescius]	[nescius]	[nescius]	[Nescius]	[nescius]
1	adsumptis	[adsumo]	None	[adsumo]	[adsumo]	[adsumo]
2	Priamus	None	None	[Priamus]	[Priamus]	[priamus]
3	pater	[pater]	[pater]	[pater]	[pater]	[pater]
4	Aesaco	None	None	[Aesacos]	[Aesaco]	[aesacos, aesaco]
5	ne	[ne]	[ne, nere]	[neo, ne, nes]	[ne]	[neo, ne, nes, nere]
6	alis	[ala, alum, alus]	[ala, alere, alum, alus]	[ala, alius, alo, alium, Alii, alum, Alus, Ale...	[ala]	[ala, alii, alius, alium, alo, alere, alum, al...
7	uiuere	[vivo]	None	[uiuo]	[uiuo]	[uiuo]
8	lugebat	[lugeo]	[lugere]	[lugeo]	[lugeo]	[lugeo, lugere]
9	tumulo	[tumulus]	[tumulum, tumulus, tumulare]	[tumulo, tumulus, tumulum]	[tumulus]	[tumulum, tumulus, tumulare, tumulo]

Latin Lemmatizers, Python support

- CLTK Backoff 
- Collatinus 
- LemLat
- Latmor
- Morpheus
- TreeTagger 
- Whitaker's Words
- more...

Latin Lemmatizers, wrapper strategies

- Scrape stdout: LemLat, Latmor, Words
- Webservice: Morpheus

Arguments for Multiplex Approach

Arguments for Multiplex Approach

- Flexibility in construction

Sample Backoff Chain for Greek

```
# Backoff lemmatizing; importance of model definition

from cltk.lemmatize.greek.greek_model import GREEK_MODEL

lemmatizer_6 = DefaultLemmatizer('Unknown')
lemmatizer_5 = MorpheusWebserviceLemmatizer(backoff=lemmatizer_6)
lemmatizer_4 = RegexpLemmatizer(greek_sub_patterns, backoff=lemmatizer_5)
lemmatizer_3 = DictionaryLemmatizer(model={'μητιόεντος': 'μητιόεις', 'νάρθηκι': 'νάρθηξ'}, backoff
=lemmatizer_4)
lemmatizer_2 = TrainLemmatizer(train_sents, backoff=lemmatizer_3)
lemmatizer = DictionaryLemmatizer(model=GREEK_MODEL, backoff=lemmatizer_2)
```

Arguments for Multiplex Approach

- Flexibility in construction
- Familiarity of resources

Regex and “Paradigm” thinking

			SINGULAR.			
Nom.	λύων	λύουσα	λῦον	διδούς	διδοῦσα	διδόν
Gen.	λύοντος	λῡούσης	λύοντος	διδόντος	διδούσης	διδόντος
Dat.	λύοντι	λῡούσῃ	λύοντι	διδόντι	διδούσῃ	διδόντι
Acc.	λύοντα	λύουσαν	λῦον	διδόντα	διδοῦσαν	διδόν
Voc.	λύων	λύουσα	λῦον	διδούς	διδοῦσα	διδόν
			DUAL.			
N. A. V.	λύοντε	λῡούσᾱ	λύοντε	διδόντε	διδούσᾱ	διδόντε
G. D.	λῡόντοιν	λῡούσαιν	λῡόντοιν	διδόντοιν	διδούσαιν	διδόντοιν
			PLURAL.			
N. V.	λύοντες	λύουσαι	λύοντα	διδόντες	διδοῦσαι	διδόντα
Gen.	λῡόντων	λῡουσῶν	λῡόντων	διδόντων	διδουσῶν	διδόντων
Dat.	λύουσι	λῡούσαις	λύουσι	διδοῦσι	διδούσαις	διδοῦσι
Acc.	λύοντας	λῡούσᾱς	λύοντα	διδόντας	διδούσᾱς	διδόντα

Arguments for Multiplex Approach

- Flexibility in construction
- Familiarity of resources
- Avoidance (minimization?) of “black box”

Arguments for Multiplex Approach

- Flexibility in construction
- Familiarity of resources
- Avoidance (minimization?) of “black box”
- Reuse of related methods (NLP)

Arguments for Multiplex Approach

- Flexibility in construction
- Familiarity of resources
- Avoidance (minimization?) of “black box”
- Reuse of related methods (NLP)
- Reuse of existing tools (computational/philological)

Object-oriented Philology

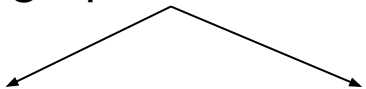
Object-oriented philological method

cltk.lemmatize

↓

backoff.py

DictLemmatizer() RegexpLemmatizer() TrainLemmatizer()



cltk.lemmatize.latin

backoff.py

DictLemmatizer() RegexpLemmatizer() TrainLemmatizer()

cltk.lemmatize.greek

backoff.py

DictLemmatizer() RegexpLemmatizer() TrainLemmatizer()

Future Directions

Future directions

- Standardization of outputs; LLOD
- Extension of LLOD and LLOD-based tools to other historical languages
- Spec. to CLTK—better integration of UDL standards to project

Questions

- How to recognize productive aspects of historical languages in LLOD framework (i.e. the Jabberwocky problem)?
- How to abstract lemmatization tools to languages where the process maybe radically different? (cf. Knowles/Don 2004)

Review article/notebook (in progress):
http://bit.ly/burns-latin-lemma-review

Select Bibliography

Almas, B. 2013. Morpheus-Wrapper. https://github.com/PerseusDL/morpheus-wrapper.

Almas, B. 2017. “Perseids: Experimenting with Infrastructure for Creating and Sharing Research Data in the Digital Humanities.” Data Science Journal 16.

Bengfort, B., Bilbro, R. & Ojeda, T. 2018. Applied Text Analysis w/Python: Enabling Language-Aware Data Products with Machine Learning. Sebastopol, CA: O’Reilly.

Boschetti, F. & Grosso, A.M.D. 2014. “TeiCoPhiLib: A Library of Components for the Domain of Collaborative Philology.” Journal of the Text Encoding Initiative 8.

Bozzi, A., G. Cappelli, M. Passarotti, E. Pulcinelli, and P. Ruffolo. 1992. LemLat. http://www.ilc.cnr.it/lemlat/.

Celano, G. G. A., G. Crane, and B. Almas. 2017. The Ancient Greek and Latin Dependency Treebank. https://perseusdl.github.io/treebank_data/.

Crane, G. 2016. “Greco-Roman Studies in a Digital Age.” Daedalus 1452: 127–33.

Eger, S., T. vor der Brück, and A. Mehler. 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: 105–13.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley.

Johnson, K.P., Burns, P.J. et al. 2015-2018. “The Classical Language Toolkit.” (v. 0.1.99) http://cltk.org/.

Juršič, M., I. Mozetic, T. Erjavec, and N. Lavrac. 2010. LemmaGen: Multilingual Lemmatisation with Induced Ripple-Down Rules. Journal of Universal Computer Science: 1190–1214. https://doi.org/10.3217/jucs-016-09-1190.

Kestemont, M., and J. De Gussem. 2017. Integrated Sequence Tagging for Medieval Latin Using Deep Representation Learning. Journal of Data Mining & Digital Humanities, Special Issue on Computer-Aided Processing of Intertextuality in Ancient Languages. https://arxiv.org/abs/1603.01597v2.

Knowles, G. & Don, Z.M. 2004. “The Notion of a ‘Lemma’: Headwords, Roots and Lexical Sets.” IJCL 91: 69–81.

Loper, E., S. Bird, and T. Tresoldi. 2017. NLTK 3.2.5 Documentation: nltk.tag.sequential. http://www.nltk.org/_modules/nltk/tag/sequential.html.

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 ACL: System Demonstrations: 55–60.

McCaffrey, D. 2006. Reading Latin Efficiently and the Need for Cognitive Strategies, in When Dead Tongues Speak: Teaching Beginning Greek and Latin, ed. J. Gruber-Miller. New York: Oxford University Press.

Ouvard, Y., and P. Verkerk. 2014. Collatinus Web. http://outils.biblissima.fr/en/collatinus-web/index.php.

Perkins, J. 2014. Python 3 Text Processing with NLTK 3 Cookbook. Birmingham, U.K.: Packt.

Piotrowski, M. 2012. Natural Language Processing for Historical Texts. San Rafael, CA: Morgan & Claypool Publishers

Schmid, H. 1994. Probabilistic Part-of-Speech Tagging Using Decision Trees, In Proceedings of the Conference on New Methods in Language Processing, Manchester, UK.

Springmann, U., H. Schmid, and D. Najock. 2016. LatMor: A Latin Finite-State Morphology Encoding Vowel Quantity. Open Linguistics 21. https://doi.org/10.1515/opli-2016-0019.

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

Wachsmuth, H. 2015. Text Analysis Pipelines: Towards Ad-Hoc Large-Scale Text Mining. New York: Springer.

Multiplex Lemmatization with the Classical Language Toolkit

Patrick J. Burns
University of Texas at Austin / Quantitative Criticism Lab
Classical Language Toolkit
First LiLa Workshop: Linguistic Resources & NLP Tools for Latin | 6.3.19