Lemmatization

P. S. Langeslag

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

Why Lemmatize?

- Linguistics: index word forms and frequency
- Corpus searchability
- NLP: dimension reduction

Principles

- Lemmatization is **classification**.
- Classification is traditionally a **supervised learning** task.
- Supervised learning requires a **high proportion of training data to labels**.
- Lemmatization is a **label-heavy** task.
- Therefore, many “naive” lemmatizers **bypass learning altogether**.

Naive Lemmatization

- Consult a list of form–label correspondences
- If a form matches multiple headwords, select the most frequent
- Do not consider token context

CLTK's Naive Old English Lemmatizer: Default Pipeline Output

```
>>> from nltk.corpus import PlaintextCorpusReader
>>> from cltk import NLP
>>> pipeline = NLP(language='ang', suppress_banner=True)
>>> echoe = PlaintextCorpusReader('echoe', '.*txt')
>>> vercelli9 = echoe.raw('394.11.txt')
>>> processed = pipeline.analyze(text=vercelli9)
>>> processed.tokens[:12]
['men', 'ða', 'leofestan', 'we', 'geleornodon', 'on', 'godcundum', 'gewritum',
'þæt', 'æghwylces', 'monnes', 'sawul']
>>> processed.lemmata[:12]
['mann', 'þa', 'leofestan', 'we', 'geleornodon', 'on', 'godcundum', 'gewrit',
'þæt', 'æghwylces', 'monnes', 'sawul']
```

CLTK's Naive Old English Lemmatizer: `best_guess=False`

```
[('Hit', ['hit']), ('sæigð', []), ('on', ['on', 'an']), ('halgen', ['halig']),
('bocan', []), ('þæt', ['þæt', 'se']), ('æfter', ['æfter']), ('gearan', []),
('ymbryne', []), ('swa', ['swa']), ('gewurðen', []), ('scule', []), ('þæt',
['þæt', 'se']), ('eall', ['eall', 'eal']), ('middeneard', []), ('mid', ['mid']),
('hæðenra', ['hæðen']), ('þeode', ['þeod']), ('geðrynge', []), ('by', []),
('and', ['and']), ('mid', ['mid']), ('heordan', []), ('hæftnysse', []), ('swa',
['swa']), ('swyðe', ['swiðe', 'swyðe']), ('gedrecced', []), ('and', ['and']),
('gedrefod', []), ('wurðeð', []), ('þæt', ['þæt', 'se']), ('hine', ['he']),
('uneaðe', []), ('ænig', ['ænig']), ('riht', ['riht']), ('gelefed', []), ('mann',
['mann']), ('mid', ['mid']), ('þan', ['se']), ('heofonlicen', []), ('kinges', []),
('tacne', []), ('gebletsigen', []), ('mote', ['motan']), ('oððe', ['oððe']),
('gesenigen', []), ('durre', ['durran']), ('Þas', ['þes']),
('geswæncennysse', []), ('we', ['we']), ('mægen', ['magan']), . . .
```

CLTK's Naive Old English Lemmatizer: `return_frequencies=True`

```
[[('to', -2.7736708137329047)], [('folc', -6.049241982277651)],
[('leof', -7.281385663570283)], [('mann', -6.829400539827225)],
[('habban', -6.742389162837596)], [('æfre', -6.924710719631551)], [],
[('geleafa', -8.534148632065651)], [('an', -5.02260319323463),
('on', -2.210686128731377)], [('an', -5.02260319323463)],
[('god', -5.116421948452285)], [('and', -2.8869365088978443)],
[('understandan', -9.227295812625597)], [('geornlice', -9.227295812625597)],
[('hu', -5.3771482109155375)], [('micel', -5.56373416649595)],
[('þearf', -7.030071235289377), ('þurfan', 0)], [('wesan', -7.435536343397541),
('seon', -7.435536343397541), ('is', -4.0343389617353855), ('þesan', 0)],
[('cristen', 0)], [('mann', -6.829400539827225)], [('þæt', -2.365584472144866),
('se', -2.9011463394704973)], [('hi', -3.120272924883342),
('heo', -4.271468755024335)], [('hi', -3.120272924883342),
('hira', -6.336924054729431)], [], [], . . .
```

CLTK's Naive Lemmatizer: Workings I

```
    def lemmatize_token(
        self, token: str, best_guess: bool = True, return_frequencies: bool = False
    ) -> Union[str, list[Union[str, tuple[str, float]]]]:

        lemmas = self.inverted_index.get(token.lower(), None)
        if not lemmas:
            mod_token = self._apply_regex(token.lower())
            lemmas = self.inverted_index.get(mod_token, None)

        if best_guess:
            if not lemmas:
                lemma = token
            elif len(lemmas) > 1:
                counts = [self.unigram_counts.get(word, 0) for word in lemmas]
                lemma = lemmas[argmax(counts)]
```

CLTK's Naive Lemmatizer: Workings II

```
        else:
            lemma = lemmas[0]

        if return_frequencies:
            lemma = (lemma, self._relative_frequency(lemma))
    else:
        lemma = [] if not lemmas else lemmas
        if return_frequencies:
            lemma = [(word, self._relative_frequency(word)) for word in lemma]

    return lemma
```

CLTK's Naive Lemmatizer: Workings III

```
def _relative_frequency(self, word: str) -> float:
    """Computes the log relative frequency for a word form"""

    count = self.unigram_counts.get(word, 0)
    return math.log(count / len(self.unigram_counts)) if count > 0 else 0
```

Why Prior Lemmatization Data Are More Valuable than Lemma Frequency

DOE A–H attested spelling results for “is”

1. hē, hēo, hit (ca. 200,000 occ.)
2. bēon (ca. 100,000 occ.)

Why Prior Lemmatization Data Are More Valuable than Lemma Frequency

DOE A–H attested spelling results for “is”

1. hē, hēo, hit (ca. 200,000 occ.)
2. bēon (ca. 100,000 occ.)

DOE A–H attested spelling results for “on”

1. and, ond (ca. 172,000 occ.)
2. ān (ca. 9000 occ.)
3. hēr-on (16 occ.)

Why Prior Lemmatization Data Are More Valuable than Lemma Frequency

DOE A–H attested spelling results for "is"

1. hē, hēo, hit (ca. 200,000 occ.)
2. bēon (ca. 100,000 occ.)

DOE A–H attested spelling results for "on"

1. and, ond (ca. 172,000 occ.)
2. ān (ca. 9000 occ.)
3. hēr-on (16 occ.)

DOE lemmatization results for "on"

1. on (85,009 assg.)
2. onǣlan (69 assg.)
3. ān (7 assg.)
4. headword #1013 (2 assg.)
5. onþēon (1 assg.)
6. onsēon (1 assg.)
7. onþwægennes (1 assg.)
8. unnan (1 assg.)
9. headword #21161 (1 assg.)
10. onþryccan (1 assg.)
11. on~eardian (1 assg.)
12. onþracian (1 assg.)

CLTK Shortcomings

1. Limited data (mostly “standard” spellings)
2. Confused data (multiple lemmas for the same word; p and diacritics)
3. Most likely headword calculated by dividing the frequency of the headword form (a questionable choice) by the size of the dictionary (not the sum of term occurrences)

Available Data Sets

Table 1: Potential data sources for the lemmatization of Old English

Source	Tokens	Terms	Lemmas	POS
DOE lemmatization (approx. M–S)	663,535	44,788	8476	no
ParCorOE	22,608		3114	yes
DOE att. sp. (A–H / A–Le)	n/a	87,875	15,533	yes
CLTK		10,171	6500	no
YCOE	1,639,127		n/a	yes
Bosworth–Toller				
Tichý PhD data set based B–T				
DOE provisional headword list				
Wiktionary				

The Value of High-Resolution Data

DOE Attested Spelling Data: *behygdiglīce* (adv.), "carefully, attentively"

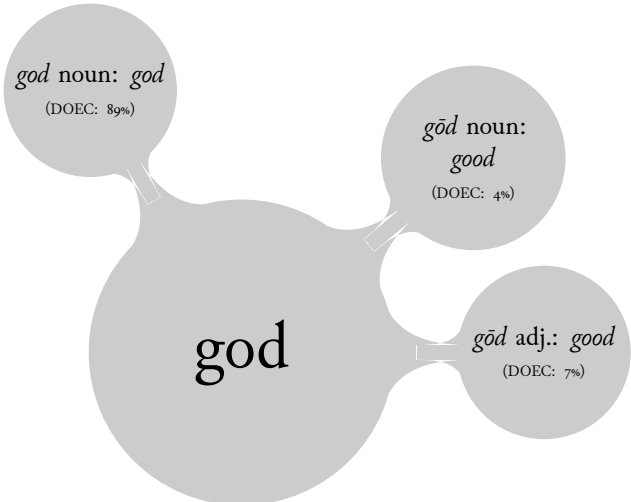
```
# be-hygdiglīce, be-hygdlīce
# 9 occ. (in multiple MSS, mainly in Bede)

behygdiglice
bihygdiglice
bighygdiglice
bihygdelice
bighygdelice
behydiglice
bihydiglice
bighydiglice
behydilice
bihydilice
bighydilice
behydelice
bighyldiglice
behigdilice
bihigdelice
bighigdelice
behidiglice
bighidiglice
bighidilice
behygdlice
bihygdlice
```

Proportion of Unambiguous Forms

Metric	*DOE* Att. Sp. A–H	DOEC Lemmatization M–S
Internally unambiguous terms	94%	96%
Internally unambiguous tokens		45%

Homographs: *god*



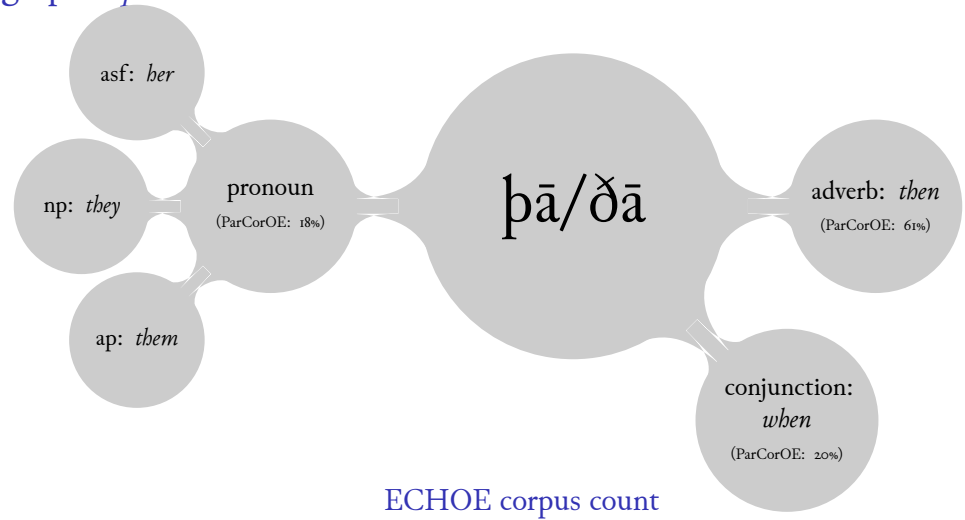
ECHOE corpus count

- 2691 occurrences of `godes`
- 1816 occurrences of `god`
- 1611 occurrences of `gode`
- 242 occurrences of `godum`
- 95 occurrences of `goda`

i.e. 6455 *standard* ambiguous forms

→ without disambiguation, c. 710 assignments would have to be hand-corrected (but all would have to be proofread)

Homographs: *þā*



ECHOE corpus count

- 14,132 occurrences

Word Sense Disambiguation

Table 3: Token context window

-2	-1	TARGET	+1	+2
criste	gelyfde	þa	cwæð	he

Word Sense Disambiguation

Table 3: Token context window

-2	-1	TARGET	+1	+2
criste	gelyfde	þa	cwæð	he

Table 4: POS context window

-2	-1	TARGET	+1	+2
PROPN	VERB	þa	VERB	PRON

Naive Part-of-Speech Assignment

Use counts from

- YCOE
- ParCorOE

Learning Features

```
features = {}
features['form'] = 'þa'
features['context1'] = 'criste'
features['context2'] = 'gelyfde'
features['context3'] = 'cwæð'
features['context4'] = 'he'
```

Multinomial Naive Bayes Classification

Prediction involves multiplying the label's **prior** with the likelihood of each feature occuring if that label is correct (the **posterior**). (This describes the nominator of the full theorem.)

e.g. the prior of "þa" being *þā (adverb)* is 0.6; the likelihood that *þā (adverb)* is followed by a verb is (say) 0.3; so the likelihood of both being true is 0.18. Since the likelihood of *þā (conjunction)* or any of the pronoun forms being followed by a verb is smaller, those combined likelihoods are more drastically decreased.

Multinomial Naive Bayes is for discrete features; Gaussian Naive Bayes is for continuous data, i.e. real numbers.

Bayes's Theorem in Full

$$P(H|E) = \frac{P(H)P(E|H)}{P(H)P(E|H)+P(\neg H)P(E|\neg H)}$$

so given that $P(adv) = 0.6$ and $P(verbfollows|adv) = 0.3$,
and let's say $P(verbfollows|\neg adv) = 0.8$:

$$P(adv|verbfollows) = \frac{0.6 \cdot 0.3}{0.6 \cdot 0.3 + 0.4 \cdot 0.8} = 0.36$$

Bayes's Theorem in Full

$$P(H|E) = \frac{P(H)P(E|H)}{P(H)P(E|H)+P(\neg H)P(E|\neg H)}$$

so given that $P(adv) = 0.6$ and $P(verbfollows|adv) = 0.3$,
and let's say $P(verbfollows|\neg adv) = 0.8$:

$$P(adv|verbfollows) = \frac{0.6 \cdot 0.3}{0.6 \cdot 0.3 + 0.4 \cdot 0.8} = 0.36$$

What's *naive* about Bayes's theorem is that we're treating each piece of evidence as independent of the others; in fact, if the word that follows is a verb, that of course has consequences for the likelihood of the next word over to be some other part of speech as well.

(See 3Blue1Brown.)

Word Sense Disambiguation Demo

See in repo: `disambiguation/`.

Bibliography I

Johnson, Kyle P., Patrick J. Burns, John Stewart, Todd Cook, Clément Besnier, and William J. B. Mattingly. "The Classical Language Toolkit: An NLP Framework for Pre-Modern Languages." In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, 20–29. Association for Computational Linguistics, 2021. https://doi.org/10.18653/v1/2021.acl-demo.3.

Sanderson, Grant. "3Blue1Brown," n.d. https://www.youtube.com/c/3blue1brown.