

Dead Language Processing

B.Eng.602 / B.DH.12 / M.EP.02b / M.EP.05d / M.DH.10 / M.DH.11

revision of March 14, 2024

Term:	Summer 2024	Instructor:	Dr P. S. Langeslag
Time:	Mondays 10:15–11:45	Office:	SEP 2.306
Room:	KWZ 2.602	Office hours:	By appointment (office/online)
Credits:	See module description	E-mail:	planges@uni-goettingen.de
Prerequisites:	See module description	Course website:	langeslag.uni-goettingen.de/dlp

This syllabus comprises an [Overview](#) (p. 1), a [Schedule](#) (p. 2), and an annotated [Bibliography](#) (p. 11).

Overview

Course Description

Natural language processing (NLP) is the interpretation and generation of human language by computational means. For most living languages, this includes spoken and written modes of communication, and large languages such as English and Mandarin have been so thoroughly modelled that we need only plug into a Google API to gain access to predictive text input, speech-to-text and text-to-speech engines, grammar checks, and translation. By contrast, our access to premodern languages is primarily text-based, and what modest annotation tools have been created for them have overwhelmingly favoured such larger languages as Latin. What few off-the-shelf libraries are available for the analysis of Old English, then, are limited and in dire need of improvement. The computational handling of these corpora accordingly takes us back to the basics, manipulating strings of text, but it also raises the question whether machine learning can be of help with so modest a corpus.

In this course, you will learn to deploy such Python libraries as NLTK and CLTK as well as the Python Standard Library and regular expressions to organize Modern English, but above all Old English text and generate statistics on the basis of plaintext corpora. You will try your hand at stemming, quantitative analysis, and word embedding, while picking up key principles from the realms of linear algebra and machine learning along the way so you may develop an understanding of what separates the abilities of humans and machines to process language, and how to go about training a machine to help out with your research or other text-based projects.

The course assumes no prior knowledge of Python, algebra, or machine learning, and students in modules outside the English Department in particular are not expected to have a prior knowledge of Old English.

Assessment

Students of [B.EP.11b](#), [M.EP.02b](#), and [M.EP.05d](#) will write a **term paper** (**due 26 August**; see module description for length requirement) either on the topic of NLP itself or relying on its methods for the study of a premodern corpus. Students of [M.EP.02b](#) also write the lecture exam; the other graduate modules do not have an exam.

Students registered for modules outside the English Department are, in most cases, expected simply to keep up with the homework and will not be separately assessed by way of an exam or term paper; but please check your module description and/or ask the instructor of the accompanying seminar for details.

Diversity

This course is run with the understanding that students bring a variety of backgrounds into the classroom in such domains as socioeconomics, appearance, culture, religion, ability, gender, age, home/family situation, and sexual identity. With different backgrounds come different needs and sensitivities. If you feel your needs or those of a fellow student require special attention or are being compromised, please feel free to make this known to me by whatever channel seems most appropriate. (For more serious concerns, the [Faculty](#) and the [University](#) each have their own points of contact as well.) I will treat all requests seriously and with confidentiality, and will seek to make accommodations within my abilities and reason. At the same time, you too owe it to your fellow students to treat them with respect regardless of their background and identity. Do not stand in the way of anyone's well-being.

Tutorials

If you wish to learn, or improve your reading knowledge of, Old English, look for tutorials and reading groups on [Stud.IP](#). If capacity allows, the tutorials for the Introduction to Medieval English Literature and Culture are open to you, as is the student-run Old English Reading Group.

Schedule

Please prepare the following readings and do the following work *ahead* of the corresponding class, and take notes for in-class discussion.

Your answers to study questions are *not* to be submitted in writing; instead, these questions help you prepare for in-class discussion, while also guiding your exam preparation if applicable. You won't be able to answer every last question, and some have no "correct" answers.

Week 1 (8 April): Concepts

Read:

- Eistenstein ch. 1: "Introduction" (10 pp.; [Stud.IP](#))

Takeaway: A concise introduction to the themes and challenges of NLP, with some reference to foundational publications.

Study questions:

1. "Much of today's natural language processing research can be thought of as applied machine learning" (2). Can you formulate *when* it makes sense to involve machine learning in NLP, and when it doesn't?
2. "Natural language processing raises some particularly salient issues around ethics, fairness, and accountability" (4). Can you think of any such issues we might encounter in the analysis of medieval corpora?
3. "From a practical standpoint, linguistic structure seems to be particularly important in scenarios where training data is limited" (6). What implications does this understanding have for the tackling of medieval corpora?

Further reading (optional):

- Eistenstein: "Preface" (3 pp.; [Stud.IP](#))

Takeaway: Identifies disciplines of relevance to NLP, and signposts the book's structure.

Reading notes:

1. This preface cites a good number of introductory texts on relevant adjacent disciplines. A notable title among the cited works is *Mathematics for Machine Learning*, which is available open access.
2. Errata: the second instance of "phenomena" on p. x should be singular "phenomenon"; "multi sentence" on the same page should read "multi-sentence."

Week 2 (15 April): NLTK

Read:

- Langeslag, "JupyterLab" (4 pp.)

Takeaway: Basic instructions on how to navigate the environment you'll rely on for your homework.

- Bird et al. ch. 1: "Language Processing and Python" (33 pp.)

Takeaway: Walks you through querying the textbook's "Examples" corpus using the Python interpreter, and teaches the basics of Python and some widespread domains of NLP along the way.

Reading notes:

1. Some chapters (but not all) in the online edition of this textbook omit the chapter reference in headings; thus "\$1.1.3" in these notes appears as [section "1.3 Searching Text"](#) in the HTML.
2. The exercises at the end of each chapter are optional; we won't discuss them in class. I do, however, urge you to try out all the book's code examples ("listings") given over the course of each chapter's main content as you do your weekly readings.

3. In your interpreter, NLTK's downloader may not default to the graphical interface printed in the textbook. Either way, the easiest way to download the textbook materials is simply to enter `nltk.download('book')` instead of `nltk.download()`, then skip to importing.
4. Please note that `nltk.FreqDist` really just reproduces `collections.Counter`, so the two may be used interchangeably. Just make sure to call it under the name by which you've imported it.

Study questions:

1. §1.1.4: Keeping in mind that corpora may be in any modern or premodern language, what language features does the `lexical_diversity()` function fail to account for?
2. §1.5.1: For each of the examples of word disambiguation here given, can you formulate if . . . then-disambiguation rules? Use shorthand prose or pseudo-code, not actual Python at this stage.

Week 3 (22 April): Handling Plaintext Corpora

Read:

- Bird et al. ch. 2: “Accessing Text Corpora and Lexical Resources” (33 pp.)
Takeaway: Introduces a range of English-language corpora and demonstrates a few more analytical queries.

Do:

1. Your local copy of the git repository includes a corpus of Old English preaching texts called ECHOE in the folder `echoe/`. Return to Bird et al. §2.1.9 (“Loading Your Own Corpus”), define `corpus_root` as `'echoe'`, load `*` into `PlaintextCorpusReader` as described in the instructions there, and inspect the `fileids()` so you know what documents are available to call.
2. Now return to Bird et al. §1.1.4 (“Counting Vocabulary”) and determine for a selection of ECHOE documents how their lexical diversity compares to some of the documents included with `nltk.book(text1, text2, etc.)`, running the necessary operations on the `words` property of your object, e.g. `len(wordlists.words('394.11.txt'))`; or, better, define variables like `document = wordlists.words('394.11.txt')` and run your `lexical_diversity` function on them. (`wordlists` is an odd choice of variable name for what is really a text corpus; you may want to choose something more appropriate, such as `echoe`, instead.) What strikes you about the lexical diversity of these Old English preaching texts compared to those shipped with the textbook? How do you explain your findings?
3. Now head to Bird et al. §1.3.1 (“Frequency Distributions”), add `from nltk import FreqDist` to your imports, and plot a non-cumulative graph for the 50 most frequent word forms in three different texts. Save your notebook for in-class discussion. What is the most frequent type? What could be done to clean up the results? Now compare your graph with the cumulative plot from the textbook. Why do you suppose the textbook said to generate a *cumulative* graph?

Tip: From this point on, we'll work with ECHOE almost every week. If you want to try our methods on a different corpus, nothing stops you from loading your own corpus into JupyterLab, or into a local installation of Python. Just make sure to format and normalize your corpus thoroughly (cf. Lane et al. §2.2.5), which for ECHOE I have done ahead of time to make your homework less cumbersome.

Week 4 (29 April): Raw Text Processing

Read:

- Bird et al. §§3.1–3.2, 3.4–3.5 (25 pp., from ch. 3: “Processing Raw Text”)
Takeaway: Explains how to import text data and use stock functions and regular expressions to manipulate strings.

Reading notes:

1. §3.1 “Electronic Books”: Though Project Gutenberg is thankfully accessible from Germany again, the files' front and back matter has changed somewhat since the current revision of the book was made available. Thus to locate the start of the back matter you will want to run `raw.rfind("*** END")` rather than `raw.rfind("End of Project Gutenberg's Crime")`.

2. Under §3.1 “**Reading Local Files**,” if you are following along with the examples, the easiest way to create a file in JupyterLab is to select the Text File icon in the launcher category “Other.” You can use the “right-click” context menu to rename `untitled.txt` into something more memorable after saving.
 3. Also under §3.1 “**Reading Local Files**,” the `U` flag on Python’s stock function `open()`, for universal newline mode, has been deprecated and superseded by an option `newline=None`, which is set by default, as is `'r'`; so just use `f = open('document.txt')`. Note that you do have to repeat the `open()` command after running the `.read()` method, as the **garbage collector** closes it at this point! As you get more Python under your belt, you’ll learn about ways of retaining information that won’t require you to reopen files.
 4. §3.3 is a detailed treatment of text encoding solutions in Python. As long as we ensure we only work on UTF8 systems and with UTF8 files, we don’t need to worry about this, but do refer back to this section if you run into issues with non-ASCII characters.
 5. If you are running a local installation of Python, you may have to install `bs4` and `feedparser` e.g. by running `pip install bs4 feedparser`.
- “**The Old English Scrabble Project**” (3100 words)

Do:

1. As the author of the blog post referenced above admits, the corpus for their letter distribution is almost wholly poetic. This shouldn’t matter, as the phonology of verse is no different than that of prose, but ECHOE with its c. 540,000 words is nevertheless a far more robust corpus than the 47,725-word corpus the author used. If you load ECHOE into a variable `echoe` following the instructions for week 3, you can determine the absolute ranked frequency of all characters in the corpus as follows:


```
from collections import Counter
Counter(echoe.raw())
```
2. Ignoring the space character and letters with diacritics, can you find a way to translate these figures into percentages?
3. What Scrabble letter point values would you assign on the basis of your analysis, and what formula have you used to arrive at them? Which become the letters with the highest Scrabble point values, and do you know the historical-linguistic background of why that is?

Further reading (optional):

- Lewis, “**Rethinking the Value of Scrabble Tiles**” (800 words)
Takeaway: Lewis’s CoffeeScript module offers an objective way of determining Scrabble letter values based on their frequency and distribution in a user-supplied corpus.
Reading note:
 1. Note that Lewis’s script doesn’t output a recommendation for how many tiles of each letter should be included. Moreover, it doesn’t output a stable sum total of letter values (e.g. 74 for PDE, 108 for OE with the maximum value set to 10), so the recommended number of tiles for each letter can’t be a function of the letter value in absolute numbers.
- Norvig, “**English Letter Frequency Counts**” (3650 words)
Takeaway: An accessible read on letter distribution in Present-Day English.

Week 5 (6 May): Normalization, Tokenization, Stemming

Read:

- Bird et al. §§3.6–3.10 (15 pp., from ch. 3: “**Processing Raw Text**”)
Takeaway: Teaches the first stages in the tackling of any text corpus with the help of regular expressions.

Do:

1. Study an **overview of Old English inflection** alongside the rudimentary stemming syntax from the **Concepts slides**. Now come up with a dozen or so of the most effective stemming rules for Old English, turn them into a Python script along the lines demonstrated in the slides, and apply them to a few 20-token slices of ECHOE documents. Come to class prepared to discuss your findings.

Further reading (optional):

- Bird et al. ch. 4: “Writing Structured Programs” (44 pp.)
Takeaway: This chapter is not on NLP, but offers a more solid understanding of Python basics and is thus recommended reading for anyone with no background in Python.
- Bird et al. ch. 5: “Categorizing and Tagging Words” (34 pp.)
Takeaway: Introduces part-of-speech tagging, a common NLP application we won't be taking on in a major way this term.

Week 6 (13 May): Handling Tagged Corpora

Read:

- Langeslag, “NLTK's YCOE Module” (5 pp.)
Takeaway: A demonstration of NLTK's YCOECorpusReader module.

Do:

- Ælfric produced two “series” of *Catholic Homilies*: a volume of forty texts between c. 987 and early 991, and a second such set between 991 and early 992. He then produced a volume of saints' lives between c. 993 and c. 998. All three collections are contained in YCOE. Scholars generally agree that Ælfric tried out a new prose style, known as rhythmical prose, towards the end of the drafting stage for his second series of homilies, which had come into its own by the time he began drafting his *Lives of Saints*. Is there some way you could quantify this development? And where do Ælfric's *Supplementary Homilies* stand on this spectrum? *Tip: if you find YCOE's index cumbersome, you can list just Ælfric's works using the following line: [(k,v) for k,v in documents.items() if 'Ælfric' in v] (you can copy and paste from the index.*
- Using what you have learned so far, compare the lexical diversity of the book of Genesis in the Authorized Version (King James Bible) shipped with `nltk.book` and the Old English translation contained in YCOE. This time, also compare them for length and come up with a way of correcting for that difference. Come prepared to discuss your findings in class.
- The Old English prose *Genesis* is believed to be the product of two translators. Ælfric claims to have produced a text “up to Isaac,” suggesting that the rest is by someone else, though scholars have found traces of Ælfric's style in isolated sections of the later part as well. Can you think of a computational way to determine where Ælfric ended his main stint?

Further reading (optional):

- Bird et al. ch. 8: “Analyzing Sentence Structure” (30 pp.)
Takeaway: Explains how NLP may be used to parse English syntactic structures.

Week 7 (20 May): NO CLASS (English Department reading week)

Week 8 (27 May): CLTK

Read:

- Kyle P. Johnson, “CLTK Demonstration” (1585 sloc)
Takeaway: A brief technical demonstration of how CLTK may be used off the shelf.

Reading notes:

1. This introduction was written as a Jupyter Notebook, so you may want to clone it directly into your course project folder; or you can simply read along on [GitHub](#), as the author has embedded his outputs in the file. If you do wish to run the code yourself, you'll have to uncomment the relevant lines in the first and third code fields before running them.
2. You may skip the section on Greek; the Latin section suffices for demonstration purposes.

- Langeslag, “CLTK” (5 pp., plus 3 pp. of appendices)

Takeaway: An attempt to make CLTK documentation both more accessible and more complete.

Do:

1. Now that you are in a position to lemmatize documents, investigate what happens to a document’s lexical diversity if we run it through the lemmatizer first. Quantify the difference, then reflect on the implications of your findings. *In choosing your corpus, just remember that YCOE includes punctuation whereas ECHOE doesn’t, so a comparison between the two would not be on equal terms unless you do a little more work to even them out.*

Week 9 (3 June): TF-IDF

Read:

- Lane et al. ch. 3: “Math With Words (TF-IDF Vectors)” (27 pp.)

Takeaway: An introduction to the mathematical magic behind the computational analysis of relevance.

Reading notes:

1. Focus on understanding the concepts and how the code works; no need to reproduce the code if that only distracts from that focus. However, the authors have made all the code available in Jupyter Notebook format [here](#); or open a terminal window and run
`git clone https://github.com/totalgood/nlpia.git` in a terminal window from your home directory to download all the textbook’s examples along with the source code of the `nlpia` package. Make sure to install the necessary packages (`pip install nlpia scipy sklearn`) before running the code examples.
2. Just to make sure you get this right: Zipf’s Law states that a word’s frequency approximates one over its frequency rank ($\frac{1}{rank}$), multiplied by the number of occurrences of the most frequent word. So given that *the* ranks first and *of* second in a typical Modern English corpus, a document counting 100 instances of the former should have about $100 \cdot \frac{1}{2} = 50$ of the latter.
3. Errata:
 - (a) On p. 74, the authors assert that “normalized term frequency. . . [is] the word count tempered by how long the document is.” This is the accepted definition of normalized TF. By their own method, however, the TF is in fact placed against the length of the list of *types*, not the list of tokens or document length! To see the difference, compare the output of `len(bag_of_words)` with that of `len(tokens)`. In their exercise on p. 87 the authors correctly measure against the list of tokens. Then on p. 90 the mathematical expression $count(d)$ is imprecise, because you are not counting the document, but rather the number of tokens it contains.
 - (b) On p. 77 for “This collections” read “This collection.”
 - (c) On p. 85, for “2” in “You’ll learn about part-of-speech tagging in chapter 2,” read “11.”
 - (d) On p. 87, ignore the object names `intro_doc` and `history_doc` given in the text body, as they don’t recur.
 - (e) On p. 90, to access `log()` you’ll have to import it from the `math` or `numpy` library.
 - (f) On p. 91 `query_vec` is defined twice where once would have sufficed.

Do:

1. Now that you have seen how to conduct TF-IDF analyses both from scratch (Lane et al. §§3.4.1–3.4.3) and through prepared packages (§3.4.3), take on a set of three ECHOE documents of your choice (e.g. three versions of the Assumption of Mary: 032.11, 048.54, and 382.13; or of *Sermo Lupi*: 068.04, 049B.40, and 331.27; or of Bethurum 17: 144.05, 331.26, and 331.30; or three randomly chosen documents. See instructions for week 3 on how to access ECHOE; then to line up your texts, define a list as under Lane §3.2 but along the lines of
`docs = [echoe.raw('032.11.txt'), echoe.raw('048.54.txt'), echoe.raw('382.13.txt')]`. Then create a vector matrix following the instructions in Lane §3.4.3. Finally, determine which two of your three documents are most alike as follows:

```
>>> from sklearn.metrics.pairwise import cosine_similarity
>>> cosine_similarity(model)
```

If your chosen documents are versions of the same text, inspect their respective length and determine whether differences in length correlate with cosine similarity. You may also be able to inspect the files side by side at echoe.uni-goettingen.de/testing/ to get an eyeball handle on similarity.

Further watching (optional):

- Vsauce, “The Zipf Mystery” (21m)

Takeaway: This video makes Zipf’s Law far more transparent, and offers a good deal of insight into some of its likeliest explanations.

Week 10 (10 June): Classification

Read:

- Bird et al. §§6.1.1–6.1.5, 6.2.1, 6.3 (16 pp., from ch. 6: “Learning to Classify Text”)

Takeaway: A hands-on lesson in classification tasks using machine learning.

Reading notes:

1. Remember to `import nltk` before reproducing the chapter’s code.
2. Take the “Your Turn” exercise of §6.1.1 seriously: this is where you acquire the skills to adapt a classifier to your own needs!
3. Tip: if you `import string` before starting the for loop in example 6.1.2, the line `for letter in 'abcdefghijklmnopqrstuvwxyz':` may be more comfortably expressed `for letter in string.ascii_lowercase:`

Do:

1. Return to the gendered names exercise of Bird et al. §§6.1.1–6.1.2 and repeat it for names appearing in documents in England up to the time the Domesday survey was completed (1086). To this end, load `pase/female.txt` and `pase/male.txt` instead of the files supplied with the textbook corpus. (Load and tokenize them using e.g. `male = open('pase/male.txt').read().splitlines();` you can then use syntax like `for name in male:`) What features prove to be the most informative for these (modernized!) names? How accurate is your classifier of Old English names relative to the textbook’s classifier of present-day names? What do you know about the way Old English names were chosen, how do you suppose these lists were obtained and normalized, and how do these facts complicate our ability to analyze them for gender in quite the same way as the modern lists? Can you think of other machine-readable features that might be more revealing?
2. The files `pase/female-dupes.txt` and `pase/male-dupes.txt` are like the files mentioned under (1) above, but without the elimination of duplicates; in other words, these add up to a full index (give or take) of named individuals whose names have come down to us from or via pre-Conquest England, but stripped down to just their given names, one per line. What can you learn from these data about relative frequency? (Hint: you will want to research the `Counter` module, which you can import from `collections`.) And can you confirm whether Zipf’s law applies to the frequency of recorded Old English names?
3. Bird et al. §6.1.4 shows how NLTK tools may be used to identify informative word suffixes. Can we apply this to ECHOE to improve on your week 5 stemming rules for Old English? (Return to week 3 for instructions on how to load ECHOE into NLTK’s `PlaintextCorpusReader`; don’t descend into the “feature extractor function,” as ECHOE is not a tagged corpus.)

Further reading (optional):

- Bird et al. §§6.4–6.7 (14 pp.)

Takeaway: The remainder of the chapter offers further detail on methods used in classification. Well worth a read if you intend to undertake any such work yourself!

Week 11 (17 June): Semantic Analysis

Read:

- Lane et al. §§4.1–4.3.5 (26 pp., from ch. 4: “Finding Meaning in Word Counts (Semantic Analysis)”)

Takeaway: A challenging introduction to how meaning may be computed.

Reading notes:

1. Just like the last time we read a chapter from Lane et al., just try to understand the concepts; don't worry about reproducing the code. Given the complexity of the material, you'll want to read this excerpt more than once.
2. Erratum: in Figure 4.3 for "TE-IDF" read "TF-IDF."

Study questions:

1. §§4.1.1–4.1.2: Having read this section as well as Lane et al. ch. 3, can you draw up a list of advantages and disadvantages of relying on TF-IDF vectors as a semantic model?
2. §§4.1.4, 4.2: Can you sum up how latent semantic analysis works in under twenty words?

Do:

- Let's classify ECHOE documents by document topic vector. This can only be done effectively if we create a stopwords list that is considerably better than that included in CLTK, so let's generate it first and append it to CLTK's:

```
>>> from collections import Counter
>>> from nltk.corpus import PlaintextCorpusReader
>>> from cltk.stops.ang import STOPS
>>> root = '/usr/share/corpora/echoe'
>>> echoe = PlaintextCorpusReader(root, '.*')
>>> fdist = Counter(echoe.words())
>>> echoestops = [k for k,v in fdist.most_common(350) if k not in STOPS]
>>> stopwords = STOPS + echoestops
```

Now we are ready to create our model:

```
>>> import os
>>> from pprint import pprint
>>> from gensim import corpora
>>> from gensim.models import LsiModel
>>> from gensim.models import LdaModel
>>> import pyLDAvis.gensim_models
>>> from nltk.tokenize import word_tokenize
>>> os.chdir(root)
>>> corpus = []
>>> for i in os.listdir():
>>>     tokens = word_tokenize(open(i).read())
>>>     stopped = [i for i in tokens if not i in stopwords]
>>>     corpus.append(stopped)
>>> dictionary = corpora.Dictionary(corpus)
>>> dtmatrix = [dictionary.doc2bow(doc) for doc in corpus]
```

We'll use two models, Latent Semantic Analysis and Latent Dirichlet Allocation:

```
>>> lsa = LsiModel(dtmatrix, num_topics=5, id2word=dictionary)
>>> ldia = LdaModel(dtmatrix, num_topics=5, id2word=dictionary)
```

You can now inspect the topics and their composition as follows:

```
>>> pprint(lsa.print_topics())
>>> pprint(ldia.print_topics())
```

But you may find it easier to visualize LDIA data as follows:

```
>>> vis = pyLDAvis.gensim_models.prepare(ldia, dtmatrix, dictionary)
```

My intention was for an output like **this**, but that's not currently happening:

```
>>> vis
```

What strikes you about the terms that the respective models choose as the building blocks for their topics?

- The above code might have benefited from the use of a stemmer, but none is available for Old English, other than the one you wrote for week 5. And at any rate lemmatization might be more effective than stemming at improving topic modelling accuracy. Lemmatizing all of ECHOE for this task might be a tad resource-heavy, but do you know where in the pipeline you would perform it, and how you would go about it? And how would you implement your own stemmer in the above routine?

Further reading (optional):

- Lane et al. §§4.4–4.8.2 (30 pp., from ch. 4: “Finding Meaning in Word Counts (Semantic Analysis)”)
Takeaway: The remainder of the chapter offers further detail on latent semantic analysis, with emphases on principal component analysis and LDiA.

Reading note:

1. This excerpt is not supplied on Stud.IP in view of copyright restrictions.

Week 12 (24 June): Word Embeddings

Read:

- Jurafsky and Martin ch. 6: “Vector Semantics and Embeddings” (31 pp.)
Takeaway: An accessible, if occasionally maths-heavy, introduction to inferring meaning from the relationships between the words in a corpus.

Reading notes:

1. If you are struggling with the chapter, try reading Lynn, “An Introduction to Word Embeddings for Text Analysis” (3000 words) first.
2. §6.5 revisits TF-IDF, but it exclusively treats it as a method for weighting word vectors. Your understanding of the technique may benefit from the reading of two complementary angles!
3. Erratum: in formula 6.20, $p * j$ should read p_j .

Do:

- Let's train a word vector model on ECHOE. Because these models require tokenized sentences as input, we'll create a list of sentences (demarcated by linebreaks in ECHOE) prior to tokenization:

```
>>> import os
>>> root = '/usr/share/corpora/echoe'
>>> os.chdir(root)
>>> allsentences = []
>>> for i in os.listdir():
>>>     sentences = open(i).read().splitlines()
>>>     for sentence in sentences:
>>>         allsentences.append(sentence)
>>> from nltk.tokenize import word_tokenize
>>> corpus = [word_tokenize(sentence) for sentence in allsentences]
To improve accuracy, we may at this point want to scan for set phrases and treat them as units:
>>> from gensim.models.phrases import Phraser, Phrases
>>> from collections import Counter
>>> fdist = Counter([inner for outer in corpus for inner in outer])
>>> commonTerms = [k for k,v in fdist.most_common(250)]
>>> phrases = Phrases(corpus, common_terms=commonTerms)
>>> bigram = Phraser(phrases)
>>> corpus = list(bigram[corpus])
```

We are now ready to configure and run our model:

```
>>> from gensim.models import Word2Vec
>>> model = Word2Vec(corpus, min_count=1, size=300, workers=2, window=5, iter=30)
This may take a minute or so. Once training is complete, you can test the model using queries like the following:
>>> model.wv.most_similar('deofol')
>>> model.wv.most_similar('niht')
>>> model.wv.most_similar('mæden')
>>> model.wv.most_similar('wop')
>>> model.wv.most_similar('blis')
>>> model.wv.distance('wer', 'cild')
>>> model.wv.distance('wif', 'cild')
>>> model.wv.distance('wer', 'wif')
```

```
>>> model.wv.distance('mæden', 'cniht')
>>> model.wv.distance('wop', 'blis')
```

What strikes you about the results you are getting? Where do you see room for improvement? Are you able to improve your model e.g. by tweaking the values of `min_count`, `size`, and `window`, or by subtly changing the final digit in the default setting `sample=1e-3`, or by extending the list of common terms? What other queries can you think of that may be of particular value in gauging model accuracy?

- As Lynn demonstrates, each attested term has its own vector, which can be called as e.g. `model['cyning']`, and can be used in arithmetic functions like `model.wv.similar_by_vector(model['cyning'] - model['mann'] + model['wif'])`. Is this an effective line of enquiry for our corpus? Explain. And what if we use DOEC instead of ECHOE as our corpus?

Further reading (optional):

- Lynn, “[An Introduction to Word Embeddings for Text Analysis](#)” (3000 words)
Takeaway: An accessible introduction to word vector modelling.
- Lynn, “[Word Embeddings in Python With Spacy and Gensim](#)” (2000 words)
Takeaway: A hands-on demonstration of word vector modelling.

Reading note:

1. Erratum? Under “Word Similarities / Synonyms,” the list of examples seems to have already been introduced in a passage that has not made it into the final text of the post.

Reading note:

1. The functions here based at `model.similarity()` and `model.most_similar()` have been deprecated in favour of `model.wv.similarity()` and `model.wv.most_similar()`.
- Lane et al. ch. 6: “[Reasoning With Word Vectors \(Word2vec\)](#)” (36 pp.)
Takeaway: A fuller introduction to Word2vec, GloVe, and the algebra underlying them.

Reading note:

1. This excerpt is not supplied on Stud.IP in view of copyright restrictions.

Week 13 (1 July): Neural Networks

Read:

- Lane et al. ch. 5: “[Baby Steps With Neural Networks \(Perceptrons and Backpropagation\)](#)” (25 pp.)

Reading notes:

1. Erratum: On p. 162, the instruction to `from random import random` is unnecessary since you end up using NumPy’s builtin `random` module.
2. If you run listing 5.4, import `sgdexperimental` instead of `SGD` from `keras.optimizers`; *the syntax is then*
Takeaway: A glimpse into the mechanics of machine learning.

Study questions:

1. What is bias for?
 2. What is the XOR logical operator?
 3. What problem does backpropagation solve?
 4. Explain local and global minima, batch and stochastic training strategies.
 5. What is overfitting?
- Korolev, “[Neural Network to Play a Snake Game](#)” (1300 words)
Takeaway: A playful walkthrough of the most basic principles of machine learning.

Reading note:

1. Many game demos may be found along similar lines: here’s [TicTacToe](#), [Tank](#), [Car](#), and [MarI/O](#).

Do:

- Spend some time in TensorFlow’s [Neural Network Playground](#). Pay attention to the loss graph as well as the output graph as you run each model. What happens if you use a linear model on XOR data? Can you find a configuration that solves spiral data to an acceptable loss rate within 500 epochs?

Further watching (optional):

- [Sanderson \(3Blue1Brown\)](#), “[Neural Networks](#)” (4 videos, 1 hour total)
Takeaway: This excellent calculus channel offers a great explainer of the same subject matter introduced in the chapter from Lane et al.

Week 14 (8 July): Drag-and-Drop Tools

Do:

- Try out the translation services at [glosbe.com/en/ang](#) and [oldenglishtranslator.co.uk](#). What can you infer about the mechanisms behind these web applications? What shortcomings do you notice, and how would you go about tackling them if you were the developer? Can you think of better approaches to translation?
- Try the tools at [lexos.wheatoncollege.edu](#). (If you don’t have any Old English texts to work with, paste a few items from [the Wikisource page for Ælfric of Eynsham](#).) Do their similarity query and vocabulary density functions yield results similar to those of your lexical diversity analysis for week 3 and your TF-IDF query for week 9? How might any differences in evaluation have arisen?

Bibliography

§1: Theory and Programming

Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. Sebastopol, CA: O’Reilly, 2009. <https://www.nltk.org/book/>.

The official book of the NLTK Python library. The HTML version is freely available CC-BY-NC-ND, and unlike the print and Kindle editions it has been updated to work with Python 3 and NLTK 3, so the code is more likely to work. However, even the online edition relies on some deprecated functions.

Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge: Cambridge University Press, 2020.

Eisenstein, Jacob. *Introduction to Natural Language Processing*. Cambridge, MA: MIT Press, 2019.

An excellent but challenging because purely mathematical-theoretical survey of NLP methodology.

Johnson, Kyle P. *CLTK Notebooks*. <https://github.com/cltk/cltk/tree/master/notebooks>.

Demonstrations for the current 1.0 release, not as extensive as the tutorials for the deprecated 0.1.x release.

———. *CLTK Tutorials*. <https://github.com/cltk/tutorials>.

The most extensive CLTK documentation available, though written for a deprecated 0.1.x release.

———. *The Classical Language Toolkit (CLTK): Legacy Docs*, 2019. <https://legacy.cltk.org/en/latest/>.

The somewhat sparse documentation for the deprecated 0.1.x release of CLTK.

Jurafsky, Dan, and James H. Martin. *Speech and Language Processing*. 3rd ed. draft. December 29, 2021. Accessed January 6, 2022. <http://web.stanford.edu/~jurafsky/slp3/>.

Online drafts of an outstanding textbook that does not teach programming but only the concepts and models used for NLP.

Lane, Hobson, Cole Howard, and Hannes Hapke. *Natural Language Processing in Action: Understanding, Analyzing, and Generating Text with Python*. With a foreword by Arwen Griffioen. Shelter Island, NY: Manning, 2019.

Dense and detailed, this textbook assumes a good deal of prior knowledge and isn't the best at explaining concepts. However, if you're willing to challenge yourself, it does cover a lot of ground and it drives home the knowledge using practical Python exercises, as well as explaining key concepts not covered in other textbooks.

Lynn, Shane. "An Introduction to Word Embeddings for Text Analysis." <https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/>.

A short and accessible introduction.

———. "Word Embeddings in Python With Spacy and Gensim." <https://www.shanelynn.ie/word-embeddings-in-python-with-spacy-and-gensim/>.

A hands-on demonstration.

Matthes, Eric. *Python Crash Course*. 2nd ed. San Francisco, CA: No Starch, 2019.

An affordable and well-received entry-level Python textbook.

Mitchell, Melanie. *Artificial Intelligence: A Guide for Thinking Humans*. New York: Farrar, Straus and Giroux, 2019. Reprinted by Pelican 2019.

An impressively well-written introduction to the methods of and concerns over AI.

Perkins, Jacob. *Python 3 Text Processing with NLTK 3 Cookbook*. 2nd ed. Birmingham: Packt, 2014.

Walks readers through specific tasks, such as POS tagging or custom corpus importing, using NLTK.

"Real Python." <https://realpython.com>.

An excellent collection of tutorials and video courses. You can read (most?) text-based tutorials free of charge, though you have to make a free account after a while, and at least the video tutorials rely on a pricey membership model.

Sanderson, Grant. "3Blue1Brown." <https://www.youtube.com/c/3blue1brown>.

A calculus YouTube channel with excellent explainers on the maths behind programming, including machine learning.

"The Old English Scrabble Project," September 21, 2010. <https://wickedday.wordpress.com/2010/09/21/the-old-english-scrabble-project/>.

An account of an attempt to determine Old English character distribution in the service of Scrabble.

Vasilev, Yuli. *Natural Language Processing Using Python and spaCy: A Practical Introduction*. San Francisco: No Starch Press, 2020.

Concise but practical and valuable; focused on spaCy rather than NLTK.

"W3 Schools: Python Tutorial." <https://www.w3schools.com/python/>.

Accessible online tutorial. Not nearly as comprehensive as Real Python.

§2: Software and Resources

Bird, Steven, and Liling Tan. *NLTK: Natural Language Toolkit*. <https://www.nltk.org/>.

A Python NLP library that remains in widespread use, though it is not as advanced as spaCy.

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, August 2021. <https://doi.org/10.18653/v1/2021.acl-demo.3>. <https://aclanthology.org/2021.acl-demo.3>.

The writeup of the conference presentation announcing the current version of CLTK, a Python library for the processing of dead languages.

Kleinman, S., M. D. LeBlanc, M. Drout, W. Feng, and C. Zhang. *Lexos*. V. 4.0, 2019. <http://lexos.wheatoncollege.edu>.

A set of online tools for NLP analysis.

Korolev, Slava. “Neural Network to Play a Snake Game.” <https://towardsdatascience.com/today-im-going-to-talk-about-a-small-practical-example-of-using-neural-networks-training-one-to-6b2cbd6efdb3>.

A playful walkthrough of the most basic principles of machine learning.

Lewis, Joshua. “Valett.” <https://github.com/jmlewis/valett>.

A CoffeeScript module for determining letter values in board games like Scrabble.

Python Software Foundation. *Python*. V. 3.10.0, October 4, 2020. <https://www.python.org/>.

An accessible interpreted computer language often used for NLP.

Roberts, Jane, and Christian Kay, eds. *A Thesaurus of Old English*, 2017. With the help of Lynne Grundy, <https://oldenglishthesaurus.arts.gla.ac.uk>.

A thematically organized index of Old English vocabulary.

Smilkov, Daniel, and Shan Carter. “A Neural Network Playground.” <https://playground.tensorflow.org/>.

A drag-and-drop neural net simulation.

spaCy. V. 3.2. Explosion. <https://spacy.io/>.

A powerful NLP library for Python which, however, has no support for premodern languages.

Stolk, Sander. *Evoke: Exploring Vocabularies and the Concepts Their Words Evoke*. <http://evoke.uliet.net>.

A frontend for “A Thesaurus of Old English” and other data sets following the same thesaurus structure.

§3: Corpora

Clunies Ross, Margaret, Kari Ellen Gade, Guðrún Nordal, Edith Marold, Diana Whaley, and Tarrin Wills, eds. *Skaldic Poetry of the Scandinavian Middle Ages*. Accessed May 9, 2022. <https://skaldic.abdn.ac.uk/>.

A current series of print editions of Old Norse poetry, as well as its digital home in the form of a collection of databases.

Healey, Antonette diPaolo, ed. *Dictionary of Old English Web Corpus*. Edited by John Price Wilkin and Xin Xiang. Toronto, 2009. <https://tapor.library.utoronto.ca/doecorpus/>.

A 3-million-word corpus covering approximately every text (but not every manuscript witness) of Old English.

McSparran, Frances, et al., eds. *Corpus of Middle English Prose and Verse*. Accessed May 9, 2022. <https://quod.lib.umich.edu/c/cme/>.

An XML text corpus of a wide range of print editions.

Orchard, Andy, ed. *A Consolidated Library of Anglo-Saxon Poetry*. Accessed May 3, 2022. <https://web.archive.org/web/20220320085850/https://clasp.ell.ox.ac.uk/>.

An XML corpus of Old English and early Anglo-Latin poetry; not yet released.

Pęzik, Piotr, Anna Cichosz, Jerzy Gaszewski, and Maciej Grabski, eds. *EnHiGLa*. Accessed May 2, 2022. <http://pelcra.pl/enhigla/>.

A cross-linguistic corpus of Old High German, Old English, and Latin prose and verse, syntactically annotated.

Pintzuk, Susan, Ann Taylor, Anthony Warner, Leendert Plug, and Frank Beths, eds. *York–Helsinki Parsed Corpus of Old English Poetry*. Accessed May 2, 2022. <https://www-users.york.ac.uk/~lang18/pcorpus.html>.

Subset of the Helsinki corpus parsed for syntax and parts of speech.

Rissanen, Matti, et al., eds. *Helsinki Corpus of English Texts*, 2011. <https://varieng.helsinki.fi/CoRD/corpora/HelsinkiCorpus/>.

A diachronic text corpus with minimal markup except for per-text information provided in the TEI header.

Rudolf, Winfried, et al. *Electronic Corpus of Anonymous Homilies in Old English*. Accessed May 2, 2022. <https://echoe.uni-goettingen.de>.

An Old English XML text corpus comprising all manuscript witnesses of the anonymous and Wulfstanian homilies as well as the prose saints' lives.

Taylor, Ann, Anthony Warner, Susan Pintzuk, and Frank Beths, eds. *The Toronto–Helsinki–Parsed Corpus of Old English Prose*, 2003. <https://www-users.york.ac.uk/~lang22/YCOE/YcoeHome.htm>.

A subset of DOEC, parsed for syntax and parts of speech.

§4: Scholarship

“ACL Anthology.” <https://aclanthology.org/>.

An index of journal articles on NLP and computational linguistics.

Davis, Matthew Evan, Tamsyn Mahoney-Steel, and Ece Turnator, eds. *Meeting the Medieval in a Digital World*. Leeds: ARC Humanities Press, 2018.

A showcase of recent digital approaches to medieval literature; the contribution by Smith and Butler involves NLP.

Horák, Aleš, and Adam Rambousek. “Lexicography and Natural Language Processing.” In *The Routledge Handbook of Lexicography*, edited by Pedro A. Fuertes-Olivera, 179–196. London: Routledge, 2017.

A brief overview of the value of each of these fields for the other.

Huang, Po-Sen, Huan Zhang, Ray Jiang, Robert Stanforth, Johannes Welbl, Jack W. Rae, Vishal Maini, Dani Yogatama, and Pushmeet Kohli. “Reducing Sentiment Bias in Language Models via Counterfactual Evaluation.” *Findings of the Association for Computational Linguistics*, 2020, 65–83.

Research into bias reduction in sentiment analysis.

Lewis, Joshua. “Rethinking the Value of Scrabble Tiles,” December 30, 2012. <https://web.archive.org/web/20130122133051/http://blog.useost.com/2012/12/30/valett/>.

The accompanying blog post to the release of Valett, providing a statistical grounding for Scrabble face value distribution.

Norvig, Peter. “English Letter Frequency Counts: Mayzner Revisited.” Or ETAOIN SRHLCU. Accessed April 17, 2022. <http://norvig.com/mayzner.html>.

An accessible read on letter distribution in Present-Day English.

Smith, William H., and Charles L. Butler. “Statistical Analysis and the Boundaries of the Genre of Old English Prayer.” In Davis, Mahoney-Steel, and Turnator, *Meeting the Medieval in a Digital World*, 11–26.

Applies relative term frequency to compare Old English prayers.

Stolk, Sander, and Thijs Porck, eds. "Exploring Early Medieval English Eloquence: A Digital Humanities Approach with 'A Thesaurus of Old English' and Evoke." *Amsterdamer Beiträge zur älteren Germanistik* 81.

A special issue on the Evoke application and its Old English data sets.

Torabi, Katayoun. "If (Not 'Quantize, Click, and Conclude'): {Digital Methods in Medieval Studies}." In Davis, Mahoney-Steel, and Turnator, *Meeting the Medieval in a Digital World*, 27–44.

A survey arguing that available tools do not replace close reading.

Younes, Nadja, and Ulf-Dietrich Reips. "Guideline for Improving the Reliability of Google Ngram Studies: Evidence from Religious Terms." *PLoS ONE* 14 (3 2019). <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0213554>.

Zhang, Sarah. "The Pitfalls of Using Google Ngram to Study Language." *Wired*, October 12, 2015. <https://www.wired.com/2015/10/pitfalls-of-studying-language-with-google-ngram/>.