DIGITAL CLASSICS ONLINE

Constructing Stoplists for Historical Languages[1]

Patrick J. Burns

Abstract: Stoplists are lists of words that have been filtered from documents prior to text analysis tasks, usually words that are either high frequency or that have low semantic value. This paper describes the development of a generalizable method for building stoplists in the Classical Language Toolkit (CLTK), an open-source Python platform for natural language processing research on historical languages. Stoplists are not readily available for many historical languages, and those that are available often offer little documentation about their sources or method of construction. The development of a generalizable method for building historical-language stoplists offers the following benefits: 1. better support for well-documented, data-driven, and replicable results in the use of CLTK resources; 2. reduction of arbitrary decision-making in building stoplists; 3. increased consistency in how stopwords are extracted from documents across multiple languages; and 4. clearer guidelines and standards for CLTK developers and contributors, a helpful step forward in managing the complexity of a multi-language open-source project.

1. Introduction

Stoplists are lists of words that have been filtered from documents prior to text analysis tasks, usually words that are either high frequency or that have low semantic value.[2] Such words have been described as lexical "noise" which prevents "signal," that is semantically or thematically significant content, from being accurately discriminated.[3] For this reason, stoplists have been called "negative dictionaries," that is lexica of unwanted terms.[4] Stopwords tend to be articles, particles, prepositions, conjunctions, pronouns, and other, often indeclinable, function words, but depending on the disposition of the corpus used as the basis for the stoplist, they can include other parts of speech. For example, forms of the verb "to be" and similar high frequency verbs are not uncommon. Through the removal of such words, text analysis tasks, such as text classification, text summarization, and information retrieval, benefit in areas like noise reduction, feature reduction, or speed optimization.

---

1 An early version of this paper was presented at the Global Philology: Big Textual Data workshop at Universität Leipzig in the summer of 2017. The author would like to thank the workshop's organizer, Thomas Koentges, and the workshop's participants for feedback and suggestions. The paper has also benefited from the feedback of the Classical Language Toolkit's community of open-source contributors.

2 Manning et al. (2008) 26, Rasmussen (2009). On the origins of the terms "stop word/stopword" and "stop list/stoplist," see Flood (1999).

3 Luhn (1957), Rijsbergen (1975) 14–17, Salton/McGill (1983) 71–72.

4 Salton/McGill (1983) 71.

Stopwords are extracted from a document or document collection on the basis of some formalization of the relative importance of individual words. Christopher Manning, for example, offers the following strategy for building stoplists:

> Sort the terms by collection frequency (the total number of times each term appears in the document collection), and then…take the most frequent terms, often hand-filtered for their semantic content relative to the domain of the documents being indexed, as a stop list, the members of which are then discarded during indexing.[5]

Yet "collection frequency" is not the only basis used for constructing stoplists, and research has shown that it is also not necessarily the best basis. Accordingly, several other methodologies have been proposed. Stoplists have been developed, for example, based on inverse document frequency and entropy measures in addition to or in combination with collection frequency. Furthermore, research in this area has moved decidedly in the direction of limiting, or in some way formalizing, the arbitrary decision-making–that is, the "hand-filtered" stage of Manning's method–that often went into earlier stoplist construction.[6]

In addition to the basis by which stopwords are extracted from a document collection, there are additional considerations that go into constructing these lists. Stoplists can be generic or domain-specific, that is based on a broad cross-section of the language or based on a restricted, well-defined collection of texts, respectively.[7] Different domains have different definitions of signal and noise, and building domain-specific lists decreases the likelihood that important terms will be lost in stopword removal. Considerations of preprocessing are also involved in stoplist construction. Accordingly, decisions such as whether to remove numbers from documents or retain case need to be made when constructing stoplists. Lastly, a decision needs to be made about the size of the list, a consideration for which there is no consensus and which varies widely by application.[8]

The Classical Language Toolkit (CLTK) is an open-source Python platform for natural language processing (NLP) research on historical languages.[9] The project offers NLP support for the languages of Ancient, Classical, and Medieval Eurasia with the goals of compiling analysis-friendly corpora, collecting and generating linguistic data, and acting as a free and open platform for generating scientific research for fields such as historical linguistics and comparative philology among others. The CLTK offers a panoply of NLP resources and tools in service of these goals. Stoplists are among these offerings and the following languages are represented in some capacity at the time of writing: Arabic, Classical Greek, Classical Hindi, Biblical Hebrew, Latin, Marathi, Middle English, Old English, Old French, Old Norse, Punjabi, and Sanskrit. That said, the aim of the CLTK is to offer as complete a set of tools as possible for each language in its scope, as well as a set of tools and resources that is, within reason, consistent across languages. In this respect, there is room for both increased coverage and improved quality.

---

5 Manning et al. (2008) 27.

6 Fox (1989) 19, concerned about "arbitrary decisions," was already noting the importance of consulting "empirical studies of word frequencies."

7 Zaman et al. (2011) 133 uses the terms "arbitrary" and "tailored."

8 Fox (1989) considers 421 words to be "maximally efficient" for general English application, but the reasons for this are not entirely clear. Saini/Rakholia (2016) report an average length of 200 words for stoplists across 42 languages in both Latin and non-Latin scripts.

9 Johnson et al. (2018).

At present, and excluding CLTK offerings, NLP resources are not always readily available for historical languages, stoplists included.[10] Generic stoplists have long been available online for Classical Greek and Latin, for example via the Perseus Project and stopwords-json.[11] Recently, Aurélien Berra has published a repository of Greek and Latin stoplists for use with Voyant Tools, which represents a big step forward in both the presentation and documentation of historical-language stoplists.[12] There has also been recent interest in building stoplists for Sanskrit.[13] But for the most part, historical languages lack this kind of support completely and, when it is available, the absence of documentation, provenance, and related issues of reproducibility and citation hamper their use.

The Perseus Project stoplists can serve as an example of the difficulties present when working with existing lists. There is no documentation to explain how these lists were constructed; that is, there is no information about the corpus used, no information about the basis by which it was constructed, and no explanation for the length of the list, among other considerations. To this can be added that, by the standards of current code-driven research, the code used to construct the lists is not available.[14] This is compounded by the fact that the Perseus lists are not published as self-contained datasets: they do not provide version control or persistent identifiers for proper citation. All of these factors should concern a historical-language researcher faced with the need to use a stoplist for a text analysis task.

The goal of this paper is to present a generalizable method in the CLTK for the construction of stoplists and to describe the implementation of this method in the CLTK Stop module. By abstracting stoplist construction and adding language-agnostic stopword extraction tools to the project, the CLTK is able to offer consistency and standardization in historical-language tool development. These tools can then be customized to support language-specific requirements. In addition, by providing a generalizable stoplist construction module to CLTK contributors, the project is better able to manage the complexity of a multi-language open-source project.

The remainder of this paper is organized as follows: Section 2 offers a literature review of computational work on building stoplists, with special attention to languages other than English. Section 3 describes the architecture of the CLTK and where the Stop module fits, as well as the logic used in writing classes and methods for stoplist construction. Section 4 presents the results of this method for a sample language, namely Latin. By way of conclusion, Section 5 presents the benefits of the Stop module and proposes future directions for development.

---

10 Interestingly, Weinberg (2004) 129 traces the origin of the stoplist to a 16th-century biblical Hebrew concordance that included a one-page list of "function" words to be excluded. On the limited availability of resources, this could in fact be broadened to include lower-resourced modern languages as well; for example, see Makrehchi/Kamel (2008) 224, who identify the lack of non-English resources for text-retrieval tasks as a primary motivation for developing stopword extraction algorithms.

11 These lists can be found at https://www.perseus.tufts.edu/hopper/stopwords and https://github.com/6/stopwords-json/blob/master/dist/la.json, respectively. The Tesserae project (https://tesserae.caset.buffalo.edu/) uses a default stoplist based on corpus frequency. Note too that there is a desideratum for Greek and Latin stoplists, to say nothing of other historical languages, in other prominent distributions, for example in the Python package stop-words, available at http://pypi.python.org/pypi/stopwords.

12 Berra (2018). For Voyant Tools, see Rockwell et al. (2012).

13 Raulji/Saini (2016), Raulji/Saini (2017).

14 See, for example, Peng (2011).

2. Related Research

Applications for stopword removal include improving information retrieval, document classification, sentiment analysis, and readability assessment, among others.[15] Much of the early research in this area was done on the English language, beginning in particular with H. P. Luhn's theorization of information retrieval.[16] But more recently, stopword studies have been published for a variety of languages, including Ahmaric, Arabic, Chinese, Greek, Gujarati, Hebrew, Malay, Mongolian, Punjabi, Sanskrit, Thai and Yoruba, among others.[17] Some studies have produced generic stoplists for the languages in question. In English, for example, the Brown stoplist and the Van stoplist have been used widely and have been referred to as "classical" stoplists.[18] In the language-specific studies listed above, stoplists have been derived both from simple word frequency as well as from other weighted document measures (for example, inverse document frequency, mean probability, variance probability, entropy measures, and combinations thereof). Other methods have also been proposed, including "term-based random sampling," the use of "combinatorial values," and the use of "term adjacency."[19] Another study employed a rules-based approach so as to avoid entirely a statistical approach to stopword extraction for the Gujarati language.[20]

Some issues that appear often in stoplist research include the issue of generic versus domain-specific list construction, preprocessing, and stoplist size. Early stoplist work focused on the construction of generic reference lists.[21] But recent trends in stoplist research favor the development of generalizable methods that can construct lists on-the-fly for any given domain.[22] There is no consensus as to preprocessing decisions for stoplist construction, and, at any rate, these decisions are prone to be highly language-specific. That said, papers such as Savoy (1999) and Zaman et al. (2011) offer examples of how such early decisions can be formalized before further processing. Lastly, there is the issue of stoplist size. This appears to be art as often as science. The DIALOG search interface, for example, took a minimalist

15 See, among many possible examples, Fox (1989), Silva/Ribeiro (2003), Lo et al. (2005), Daowadung/Chen (2012), Saif et al. (2014).

16 Luhn (1957), Luhn (1958), Luhn (1960).

17 Ahmaric: Miretie/Khedkar (2018); Arabic: El-Khair (2006); Chinese: Zou et al. (2006); Yao/Ze-wen (2011); French: Savoy (1999); Greek: Lazarinis (2007); Gujarati: Rakholia/Saini (2017); Hebrew: HaCohen-Kerner/Shmuel (2010); Malay: Chekima/Alfred (2016); Mongolian: Zheng/Gaowa (2010); Persian: Sadeghi/Vegas (2014); Punjabi: Puri et al. (2013), Kaur/Saini (2016); Sanskrit: Raulji/Saini (2016), Raulji/Saini (2017); Thai: Daowadung/Chen (2012); Yoruba: Tijani et al. (2017).

18 See, for example, Lo et al. (2005) 17. The Perseus Project stoplists have attained a kind of "classical" status within the Greek and Latin research community, though their limitations are noted above.

19 Term-based random sampling: Lo et al. (2005); combinatorial values: Choy (2012); term adjacency: Rose et al. (2010).

20 Rakholia/Saini (2017).

21 See, for example, Fox (1989).

22 For example, Lo et al. (2005) 17 note in their stoplist study: "Each collection of documents is unique. It is therefore sensible to automatically fashion a different stopword list for different collections in order to maximise the performance of an IR system."

approach, capping their stoplist at just nine words: "an and by for from of the to with."[23] Berra pushes in the other direction, including 4,001 stopwords for Latin and 6,696 for Greek in the most recent versions of the Voyant Tools stoplists.[24]

3. Description

This study describes the development and implementation of a generalizable method for building stoplists in the CLTK, namely the Stop module. The code for the Stop module, like all code in the CLTK, is written in Python (version 3.7 at the time of writing).[25] The Stop module is available beginning with version 0.1.84 of the CLTK and is available as part of the free, open-source platform under an MIT license.

The CLTK increasingly aims at the following development strategy: build language-agnostic abstract classes for major NLP tasks, such as word tokenization or lemmatization, which can then be inherited as necessary by language-specific subclasses. The CLTK Stop module follows this "object-oriented philology" development pattern.

23 Harter (1986) 88. In digital classics, this can be compared to the minimalist approach taken by the Tesserae Project in their default search settings, where the stoplist, as given in exported files, is also composed of nine words: *qui, quis, sum, et, in, is, non, hic, ego,* and *ut*. With respect to minimalist stoplists, note also Schofield et al. (2017) on stopword effectiveness in topic modeling: "Except for the dozen or so most frequent words, removing stopwords has no substantial effect on model likelihood, topic coherence, or classification accuracy."

24 Berra's lists are greatly enlarged by the inclusion of inflectional variants; so, for example, all morphological possibilities of the verb *sum* ("to be") are included in the Latin list without consideration of the relative frequency of individual forms. It is worth noting that Nothman et al. (2018) suggest that the inclusion of inflectional variants "generally seems an advisable path towards improved consistency."

25 The complete code for the Stop module is available in the CLTK GitHub repository at https://github.com/cltk/cltk/blob/master/cltk/stop/.

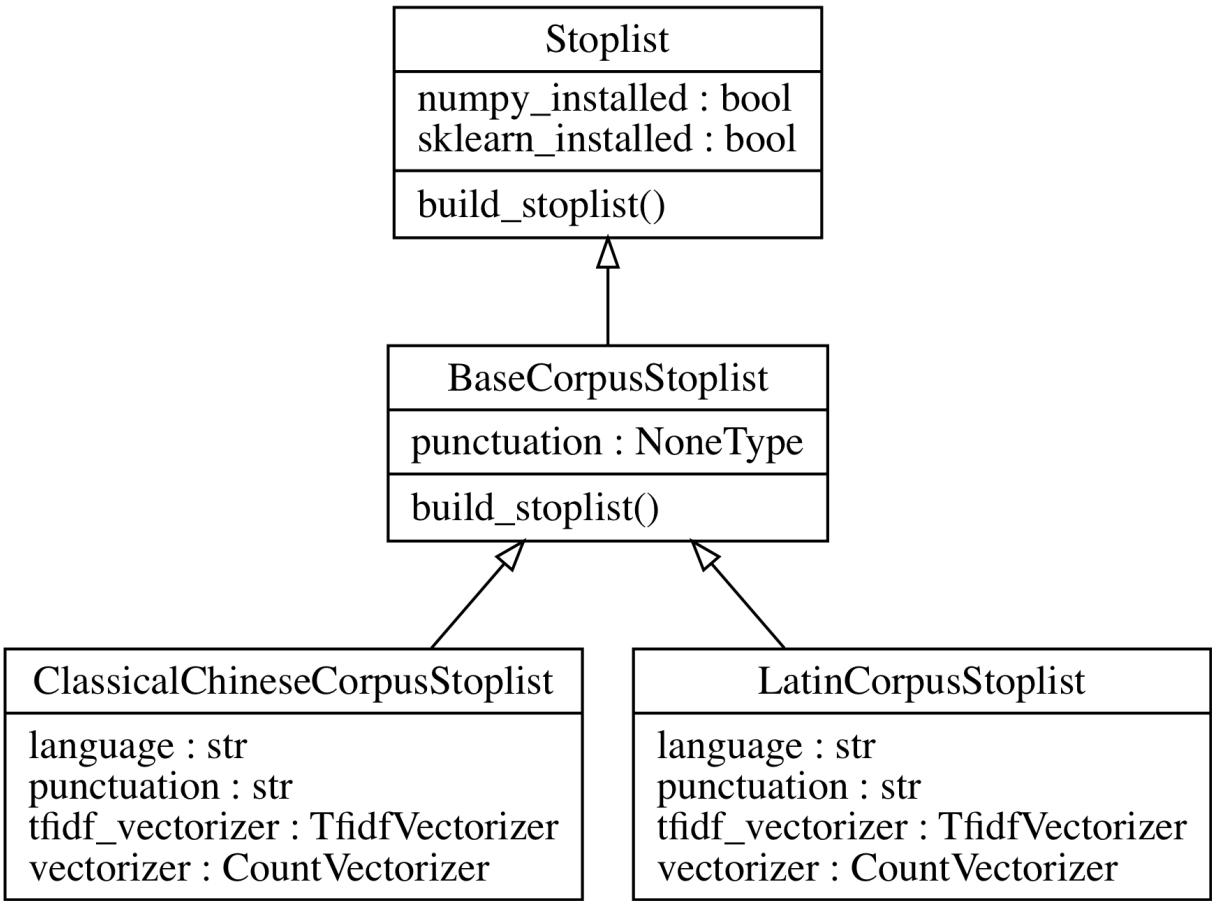

Fig 1: The inheritance tree for the CLTK Stop module. Note how language-specific classes, such as ClassicalChineseCorpusStoplist, inherit the build_stoplist method from BaseCorpusStoplist while overriding the language and punctuation attributes as necessary.

The module has the following structure, as shown in Figure 1:

1. There is a Stoplist abstract class that sits at the top of the inheritance hierarchy. Its main role is basic initialization: first, it allows a language to be set in case there are language-specific parameters that need to be set, and, secondly, it confirms that package dependencies for the class, namely Numpy and Scikit-learn, are installed.[26] Stoplist also contains an abstract method called "build_stoplist," which serves as a placeholder method to be overridden by subclasses of Stoplist. At the module level, that is in the file stop.py in the Stop module, the Stoplist abstract class has an additional subclass: BaseCorpusStoplist, which will be the focus of the remaining discussion in this section.

2. BaseCorpusStoplist inherits and overrides the "build_stoplist" method from Stoplist. Furthermore, it is customized by way of an array of parameters that need to be set for the task of stoplist construction. As Fox mentioned in his landmark stoplist publication for English: "Selecting a stop word list is more difficult than it appears."[27] Likewise, "selecting" the parameters that need to be considered when constructing a Python class and related functions for this task poses similar challenges. A primary goal in writing a generalizable method then is to eliminate, or at least formalize, the "various arbitrary

26 Numpy: Oliphant (2006); Scikit-learn: Pedregosa et al. (2011).

27 Fox (1989) 21.

decisions" that are often required.[28] Providing a class dedicated to producing domain-specific stoplists in a systematic manner reduces the number of arbitrary decisions that need to be made, and when decisions are made, it allows these decisions to be declared explicitly in parameter selection or otherwise documented directly in supporting code. Accordingly, stoplists built using the "build_stoplist" method meet two of the goals for stoplist construction stated in the introduction, that is they are well documented and replicable.

The "build_stoplist" method takes the following parameters: "texts", "size", "sort_words", "inc_values", "lower", "remove_punctuation", "remove_numbers", "include", "exclude", and "basis."

The parameter "texts" take a list of strings that serve as the source texts or document collection from which stopwords should be extracted. These texts are split into tokens and it is these tokens from which stopwords are drawn; no lemmatization is performed by the "build_stoplist" method.[29] The parameter "size" is the number of words included in the output list. The parameter "inc_values" is a Boolean variable that allows users to produce either a bare wordlist for output (inc_values = False) or include a value related to how words in the document collection are measured (the default setting, True). So, if words are measured by simple frequency, a word appearing 99 times in the "texts" would have a value of 99. The parameters "lower," "remove_punctuation," and "remove_numbers" are built-in preprocessing parameters for handling case (that is, whether to make all words in the collection lowercase), punctuation, and numbers; also Booleans, they are all set to True by default.

Since even in domain-specific applications, there may be an argument for including additional words or for "culling important terms,"[30] there are two parameters that allow users to formalize the "hand-filtered" aspect of stoplist construction, namely "include" and "exclude." Both parameters take a list of strings. Any words included in the former are added to the final output regardless of their value; words in the latter are removed. Both parameters are set to empty lists by default.

Lastly, the "build_stoplist" method allows users to set the basis on which words are measured. At present, the bases that are supported include: frequency, mean probability, variance probability, entropy, and a composite basis drawn from a 2006 paper on constructing stoplists by Zou et al.[31]

---

28 Savoy (1999) 949. For example, Rijsbergen (1975) 14, on the boundary decisions for cutting off both insignificant very high frequency and very low frequency words, writes: "A certain arbitrariness is involved in determining the cut-offs. There is no oracle which gives their values." That said, the goal here is to make explicit in code any decisions that might otherwise fall into the category of "certain arbitrariness."

29 It should also be noted that "build_stoplist" uses the default tokenization method of Scikit-learn's CountVectorizer class, namely whitespace tokenization. There is provisional support for character-based languages, such as Classical Chinese; in this case, a language-specific subclass of BaseCorpusStoplist is created with CountVectorizer's "analyzer" parameter set to analyze features at the character level. Since both CountVectorizer's "tokenizer" and "analyzer" parameters can take a callable, there is the potential for custom tokenizers and segmenters to be used in these language-specific subclasses. This is an area of current CLTK development.

30 Fox (1989) 19–20.

31 Zou et al. (2006).

3. BaseCorpusStoplist can be further subclassed by language, as for example with ClassicalChineseCorpusStoplist and LatinCorpusStoplist. These can be kept in language-specific files (for example, classical_chinese.py or latin.py), with certain parameters, such as the punctuation set for that language or whether to analyze texts by word or character, set in advance.

As noted above, the method that has been used as the default for classes in the CLTK Stop module is that of Zou et al. (2016). In their paper, Zou et al. present an "automatic aggregated methodology" which produces a stoplist through a combination of three measures, two with a document-statistical model (namely, mean probability and variance probability) and one with an informational model (namely, entropy). The document-statistical model measures mean probability to account for overall frequency in a document collection and variance probability to account for distribution across the documents. The information model uses entropy as a measure of how informative a word is within a given document. Lists of top terms are produced for each measure and the lists are aggregated using a Borda count to produce a composite list. As shown by Zou et al., we should expect stopwords to have a high mean probability and a low variance probability.[32] That is, they should both occur with great frequency in individual documents in the collection and their distribution across the documents should be stable. The CLTK Stop module, by using Zou et al.'s method as the default basis, formalizes this model, allowing users to generate statistically based composite stoplists for any document collection.

Zou et al. developed their method for use in Chinese, but their method has been used for languages such as Arabic, Persian, Punjabi, Thai and Yoruba.[33] Since it has already been shown to be effective for stoplist construction in several languages and across multiple scripts, this method seemed particularly well suited for general application in the CLTK Stop module.

4. Example: Constructing Latin Stoplists[34]

In this section, I provide an example of historical-language stoplists created using the CLTK Stop module for the Latin language using the default 'Zou' basis. This list is then compared to existing lists.

I have used the CLTK's Latin Library corpus (LL) to construct the example stoplist.[35] This is a corpus of plaintext files, scraped from the website "The Latin Library."[36] It consists of 2,152 files, representing Latin texts in a wide range of authors, works, and genres, and spanning chronologically from the earliest Latin authors to 20th-century Neo-Latin authors. For this

32 Zou et al. (2006) 1012: "Intuitively, the probability of a word to be a stop word is directly proportional to the mean of probability, but inversely to the variance of probability."

33 Alajmi et al. (2012), Daowadung/Chen (2012), Puri et al. (2013), Sadeghi/Vegas (2014), Tijani et al. (2017).

34 All of the code used to generate text statistics or to build and analyze the stoplists presented in this section of the paper can be found in the Jupyter notebook in the GitHub repository associated with this paper, available at https://github.com/diyclassics/stopwords-paper. The version of this repository at the time of publication has been archived with Zenodo at doi:10.5281/zenodo.1477208.

35 Available at https://github.com/cltk/latin_text_latin_library.

36 Available at http://www.thelatinlibrary.com/.

test, the texts have been minimally preprocessed; specifically, 1. HTML entities have been converted to the corresponding unicode character (that is, entities such as ">" have been converted to the ">" character), and 2. the texts have been truncated to start at the first word appearing 1,000 characters from the beginning of the text and to end at the word appearing 1,000 characters from the end in order to remove paratextual material included in "The Latin Library" texts, such as title and author information and text related to web navigation.[37] Case has been retained, as have numbers and punctuation. There are approximately 16.2 million tokens in this corpus.[38]

Using the "Zou" basis with the "size" parameter set to 100 in "build_stoplist," the Latin-specific instance of BaseCorpusStoplist produces the following stoplist for LL:

ab ac ad ante apud atque aut autem causa cui cuius cum de dei deus dum ea ego ei eius enim eo erat ergo esse esset est et etiam eum ex fuit haec hic his hoc iam id igitur illa ille in inter ipse ita me mihi modo nam ne nec neque nihil nisi nobis non nos nunc omnes omnia omnibus per post potest pro qua quae quam quem qui quia quibus quid quidem quis quo quod quoque res se secundum sed si sibi sic sicut sine sit sub sunt tamen te tibi tu tunc ubi uel uero uos ut.[39]

The LL stoplist, drawn as it is from the largest collection with widest variety of genres, can be taken as the closest thing to a generic stoplist for Latin and it compares favorably to existing lists.[40] It contains 55.4% of the words (41/92) from the Perseus stoplist and 77.5% of the words (31/40) from the stopwords-json stoplist. While on the surface, these numbers may appear low, ready answers appear upon review of the lists. The Perseus list, for example, consists primarily of lemmas (that is, dictionary headwords), while the LL list contains any token that meet the statistical criteria of the stopword extraction algorithm.[41] So, for example, the nominative forms of several personal pronouns, such as *ego* ("I"), *tu* ("you"), and *nos* ("we"), appear on both lists, but oblique forms, such as *mihi* ("to me"), *me* ("me"), *tibi* ("to you"), *te* ("you", accusative) *sibi* ("himself, herself, itself"), and *nobis* ("to us"), only appear on the LL list and so count against the 100-word limit set for this example. Another factor is that there are no nouns or adjectives in the Perseus list, while by strict statistical determination certain forms of the nouns *causa* ("cause, reason"), *deus* ("god"), and *res* ("thing") and certain forms of the adjective *omnis* ("all") appear on the LL list. Interestingly, the situation is reversed in the stopwords-json list where the largest point of difference when compared to the LL list is the

37 Because of this truncation, files consisting of less than 1000 characters have been removed from the fileset all together.

38 The WordPunctTokenizer from the Natural Language Toolkit has been used to tokenize the texts; cf. https://www.nltk.org/_modules/nltk/tokenize/regexp.html#WordPunctTokenizer.

39 Specific information about the measurements underlying the "Zou" basis, such as mean probability, variance probability, and entropy values, can be found in Appendix A of the Jupyter notebook at https://github.com/diyclassics/stopwords-paper/blob/master/notebooks/stopwords-paper.ipynb.

40 Complete results of comparison between the LL stoplist and the Perseus, stopwords-json, and Voyant Tools list can be found in Appendix B of the Jupyter notebook at https://github.com/diyclassics/stopwords-paper/blob/master/notebooks/stopwords-paper.ipynb.

41 There are some non-lemma forms such as *es* ("you are") and *est* ("he, she, it is") in the Perseus list, but these words are the exception.

inclusion of three oblique forms of res (*re*, *rem*, and *rebus*). It is unclear why these are included (and why other forms of *res* like *rei* and *rerum* are not).[42] As noted in Section 1, while stoplists tend to include mostly function words, this is only a tendency and perhaps a subjective one prone to the caveats against arbitrary decision-making noted by Manning. The direction taken in this paper tends towards a statistical definition of the stopword extraction process and these nouns meet the statistical criteria of the algorithm. Arguments for excluding these words can (and should) be made by other stoplist builders, but the burden will be on them to argue against the statistically derived description of a corpus of Latin text.

Conversely, because of the large size of Berra's Voyant Tools Latin stoplist, it makes more sense to reverse the direction of the comparison with this list. The Voyant Tools list contains 98% of the words in the LL stoplist. As seen in the previous comparisons, noun forms account for the missing forms: *deus* ("god") and its genitive form *dei* ("of god"). (Interesting to note that forms of *causa* and *res* have made Berra's cut.) Again, there are good arguments for exclusion of these terms from stoplists. But the comparison of the two lists demonstrates the importance of domain in creating (and by extension applying) stoplists. The Latin Library contains a large amount of post-Classical, Christian writing, in which "god" is a very high frequency token. Berra uses the Packard Humanities Institute Latin Texts version 5.3, which for the most part only covers the Latin language up to 200 CE, explaining the difference.[43] If a researcher planned to use a Latin stoplist with a set of Classical texts, the recommendation would be to build a custom list on a representative sample of Classical texts. As noted in the Introduction, this flexibility in stoplist construction is one of the main benefits of the CLTK Stop module.

42 Words like this can be compared to what Nothman et al. (2018) 9 call "controversial" words, that is words that appear in a stoplist though without clear statistical support.

43 See http://latin.packhum.org/about for a description of the PHI Latin Texts project. In Appendix C of the Jupyter notebook (https://github.com/diyclassics/stopwords-paper/blob/master/notebooks/stopwords-paper.ipynb), I have provided examples of three domain-specific stoplists constructed from subsets of the Latin Library corpus: one for the works of Cicero, one for the Biblia Sacra, or Latin text of the bible, and one for collections of Roman legal texts. It should be noted that *deus* is extracted by the 'Zou' algorithm from the Biblical texts, though it is not when applied to the Cicero texts or the legal texts.

5. Conclusion

The use of stoplists is not without its detractors. The origin of their use stems from physical and computational limitations, such as the avoidance of paper waste on the one hand and low processing power and memory on the other, that are increasingly obsolescent. Furthermore, their use is subject to issues of linguistic ambiguity that can be difficult to control for, like homonymity.[44] It has also been suggested that constructing domain-specific stoplists is time-consuming and subject to arbitrary interventions by the creator of the list, while at the same time not being particularly effective.[45] Nevertheless, there are strong arguments in favor of developing the CLTK Stop module.[46]

First, there are many existing NLP text processing tools, largely aimed at modern language applications, that make use of stoplists or include stoplist parameters. For example, the Natural Language Toolkit provides default stoplists for 21 languages.[47] Historical-language researchers learning NLP basics using the NLTK book find themselves at a loss when the example code asks them to import a list for an unsupported language.[48] Another example is the CountVectorizer class in Scikit-learn, which allows a stoplist to be designated in its parameter list.[49] This list is by default an English stoplist. Researchers wishing to apply machine-learning methods to historical-language text through Scikit-learn have the option of using a stoplist not appropriate for their research question or are otherwise obligated to design their own list or simply ignore the parameter. By using a list created with methods from the CLTK Stop module, users can be confident that this resource is appropriate for their language, not to mention the specific domain that they are working on. Moreover, by offering methods for reading document collections and systematically building stoplists, the module pushes back against criticisms about the "time-consuming" or "arbitrary" nature of the process.

---

44 Savoy (1999) 945.

45 Schofield et al. (2017).

46 In addition to the arguments presented below, there are other positive uses for statistically based word extraction methods similar to or identical to stopword extraction. These lists can be used, for example, as the basis for stylometric analysis and author-attribution studies; see Arun et al. (2009).

47 The NLTK Stopwords corpus includes Arabic, Azerbaijani, Danish, Dutch, English, Finnish, French, German, (modern) Greek, Hungarian, Indonesian, Italian, Kazakh, Nepali, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish, and Turkish; see https://www.nltk.org/nltk_data/.

48 Bird et al. (2015); see section 4.1 on "Wordlist Corpora." The development of the CLTK Stop module described in this paper has its origins in using the instructions for adding a new language to the spaCy NLP platform, which asks contributors to define a language-specific stoplist as their first task; see https://spacy.io/usage/adding-languages.

49 Pedregosa et al. (2011). For parameter definitions, see http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

The CLTK Stop module similarly pushes back against the criticisms of available historical-language stoplists mentioned in the Introduction, most especially the lack of documentation, provenance, sources, and construction methodology behind many existing resources.[50] Stoplists created using the CLTK Stop module have a clear advantage. The software itself is open source, documented, version-controlled, and citable. Moreover, any code written by a user that uses the CLTK Stop module to build a stoplist can be shared and cited, and as such the document collection used as the source will be explicitly defined in this code as will the parameters and any pre- or postprocessing applied to the lists. Accordingly, the CLTK Stop module offers better support for well-documented, data-driven, and replicable results.[51]

There are two additional benefits to the CLTK community in the offering of a generalizable method for stoplist construction. The first pertains to CLTK users, namely that they can be assured of consistency in how stopwords are extracted from documents across multiple languages. As I mentioned in the Introduction, the resources currently offered in the CLTK, specifically generic reference lists, have been cobbled together from a variety of sources with no underlying and unifying methodology for how these resources were developed. By offering an abstract class with language-agnostic methods that can be subclassed and then overridden and reparameterized as necessary, CLTK users can be confident that the stoplists that they create are based on research that has been shown to be applicable across a wide variety of languages but also take into account language-specific demands. So, for example, the Latin and Classical Chinese submodules both inherit from BaseCorpusStoplist but the former is subclassed in such a way that it tokenizes documents based on words and the latter on characters.

The second pertains to CLTK contributors. Stoplists are one element of the project's basic language resource kit for each language.[52] That said, they are currently available for only a limited number of languages in the project's scope. Before the development of the new Stops module, contributors who wished to add a stoplist for a given language did so with minimal direction. That is, there was no attempt at consistency in extraction methodology across languages, nor was there a "quality control" standard by which existing CLTK stoplists could be evaluated.[53] With the introduction of the Stop module, contributors now have a clear starting point for development, a helpful step forward in managing the complexity of a multi-language open-source project.

---

50 This problem is not limited to historical-language NLP stoplists as shown in Nothman et al (2018). For example, on the Scikit-learn default stoplists, Nothman et al. (2018) 7 write: "Despite its popular use, the current Scikit-learn maintainers cannot justify the use of this particular list, and are unaware of how it was constructed."

51 In this respect, it is compatible with the RAD (replicable, aggregable, and data-driven) paradigm described in Haswell (2005). Note also that Nothman et al. (2018) 11 propose the following strategies for stoplist provisioning in open-source software: better documentation, dynamic adaptation to specific NLP tasks, increased quality control in assessing the inclusion of controversial terms and evaluating completeness, and wider availability of tools for automatic list generation. CLTK Stop development aims at everything on this list, especially the last item.

52 Krauwer (2003).

53 See Nothman et al (2018) 11.

This paper describes the first version of a generalizable method for building stoplists for historical languages with the CLTK. That said, work on the module continues. Here are some concluding thoughts on future directions for the module.

Zou et al.'s method has been used in stoplist research for several languages and, for this reason, is an appropriate starting point for developing generalizable methods. Still, as noted above, it is not the only method for stoplist construction. Future work will investigate how other methods for systematically building both generic and domain-specific stoplists, including term-based random sampling and term adjacency among others, could be effectively implemented in the CLTK. Further discussion of parameters available in "build_stoplist" and their applicability within the language-specific subclasses is also necessary.[54] Another area that will require additional research is the evaluation of historical-language stoplists. Just as these low-resource languages often lack stoplists all together, they also largely lack the tagged document collections necessary to serve as benchmarks in evaluation. Only with the development of more benchmark collections can we truly begin to evaluate the effectiveness of historical-language stoplists.[55] In the meantime, the CLTK Stop module offers a significant step forward in formalizing stoplist construction methodology and, in doing so, promotes consistency and maintains standards across languages for both users and developers, which in turn facilitates documentation and reproducibility of an NLP resource for historical-language research.

54 Discussions about best practices for stoplist construction in the CLTK are ongoing in the project's GitHub 'Issues'; see, for example, the following thread on "improvements to multilingual stop module," through which character-based tokenization for Classical Chinese was introduced to the Stop module: https://github.com/cltk/cltk/issues/743.

55 So, for example, supervised classifier-based stopword extraction methods, such as those used in Makrehchi/Kamel 2008, are not yet practical for most languages represented in the CLTK.

6. References

Alajmi et al. (2012): A. Alajmi, E. M. Saad and R. R. Darwish. "Toward an Arabic Stop-Words List Generation," International Journal of Computer Applications 46 (8), 8–13.

Arun et al. (2009): R. Arun, R. Saradha, R., V. Suresh, M. Narasimha Murty and C. E. Veni Madhavan, "Stopwords and Stylometry: A Latent Dirichlet Allocation Approach," in: NIPS Workshop on Applications for Topic Models: Text and Beyond, 1–4.

Berra (2018): Aurélien Berra, Ancient Greek and Latin Stopwords for Textual Analysis, version 2.1.0. https://github.com/aurelberra/stopwords. (accessed on 30 August 2018)

Bird et al. (2015). Steven Bird, Ewan Klein and Edward Loper, "Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit," https://www.nltk.org/book/. (accessed on 30 August 2018)

Chekima/Alfred (2016): Khalifa Chekima and Rayner Alfred, "An Automatic Construction of Malay Stop Words Based on Aggregation Method," in: Proceedings of Soft Computing in Data Science: Second International Conference, 180–189.

Choy (2012): Murphy Choy, "Effective Listings of Function Stop Words for Twitter," arXiv preprint, http://arxiv.org/abs/1205.6396.

Daowadung/Chen (2012): Patcharanut Daowadung and Yaw-Huei Chen, "Stop Word in Readability Assessment of Thai Text," in: 2012 IEEE 12th International Conference on Advanced Learning Technologies, 497–99.

El-Khair (2016): Ibrahim Abu El-Khair, "Effects of Stop Words Elimination for Arabic Information Retrieval: A Comparative Study," International Journal of Computing and Information Sciences 4 (3), 119–133.

Flood (1999): Flood, Barbara J., "Historical Note: The Start of a Stop List at Biological Abstracts," Journal of the American Society for Information Science 50 (12), 1066.

Fox (1989): Christopher Fox, "A Stop List for General Text." SIGIR Forum 24 (1–2), 19–21.
HaCohen-Kerner/Shmuel (2010): Yaakov HaCohen-Kerner and Yishai Blitz Shmuel, "Initial Experiments with Extraction of Stopwords in Hebrew," in: Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, 449–453.

Harter (1986): Stephen P. Harter, Online Information Retrieval: Concepts, Principles, and Techniques. San Diego.

Haswell (2005): Richard H. Haswell, "NCTE/CCCC's Recent War on Scholarship," Written Communication 22 (2): 198–223.

Johnson (2018): Kyle P. Johnson and the Classical Language Toolkit contributors, "The Classical Language Toolkit," http://cltk.org/. (accessed on 30 August 2018)

Kaur/Saini (2016): Jasleen Kaur and Jatinderkumar R. Saini, "Punjabi Stop Words: A Gurmukhi, Shahmukhi and Roman Scripted Chronicle," in: Proceedings of the ACM Symposium on Women in Research, 32–37.

Krauwer (2003): Steven Krauwer, "The Basic Language Resource Kit (BLARK) as the First Milestone for the Language Resources Roadmap." Proceedings of International Workshop Speech and Computer (SPECOM), 8–15.

Lazarinis (2007): Fotis Lazarinis, "Engineering and Utilizing a Stopword List in Greek Web Retrieval," Journal of the American Society for Information Science and Technology 58 (11), 1645–1652.

Lo et al. (2005): Rachel Tsz-Wai Lo, Ben He and Iadh Ounis, "Automatically Building a Stopword List for an Information Retrieval System," in: 5th Dutch-Belgium Information Retrieval Workshop, 17–24.

Luhn (1957): Hans Peter Luhn, "A Statistical Approach to Mechanized Encoding and Searching of Literary Information," IBM Journal of Research and Development 1 (4): 309–17.

Luhn (1958): Hans Peter Luhn, "The Automatic Creation of Literature Abstracts." IBM Journal of Research and Development 2 (2): 159–65.

Luhn (1960): Hans Peter Luhn, "Key Word-in-Context Index for Technical Literature (KWIC Index)." American Documentation 11 (4): 288–95.

Makrehchi/Kamel (2008): Masoud Makrehchi and Mohamed S. Kamel, "Automatic Extraction of Domain-Specific Stopwords from Labeled Documents," in: Advances in Information Retrieval, 222–33.

Manning et al. (2012): Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge.

Miretie/Khedkar (2018): Sileshi Girmaw Miretie and Vijayshri Khedkar, "Automatic Generation of Stopwords in the Amharic Text," International Journal of Computer Applications 180 (10), 19–22.

Nothman et al. (2018): Joel Nothman, Hanmin Qin and Roman Yurchak, "Stop Word Lists in Free Open-Source Software Packages," in: Proceedings of Workshop for NLP Open Source Software (NLP-OSS), 7–12.

Oliphant (2006): Travis E. Oliphant, A Guide to NumPy, vol. 1, Spanish Fork, UT.

Pedregosa et al. (2011): Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss and Vincent Dubourg, "Scikit-Learn: Machine Learning in Python," Journal of Machine Learning Research 12: 2825–2830.

Peng (2011): Roger D. Peng, "Reproducible Research in Computational Science," Science 334 (6060): 1226–27.

Puri et al. (2013): Rajeev Puri, R. P. S. Bedi and Vishal Goyal, "Automated Stopwords Identification in Punjabi Documents," Research Cell: An International Journal of Engineering Sciences 8, 119–125.

Rakholia/Saini (2017): Rajnish M. Rakholia and Jatinderkumar R. Saini, "A Rule-Based Approach to Identify Stop Words for Gujarati Language," in: Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications, 797–806.

Rasmussen (2009): Edie Rasmussen, "Stoplists," in: Ling Liu and M. Tamer Özsu (eds.), Encyclopedia of Database Systems, Boston, 2794–96.

Raulji/Saini (2016): Jaideepsinh K. Raulji and Jatinderkumar R. Saini, "Stop-Word Removal Algorithm and Its Implementation for Sanskrit Language," International Journal of Computer Applications 150 (2), 15–17.

Raulji/Saini (2017): Jaideepsinh K. Raulji and Jatinderkumar R. Saini, "Generating Stopword List for Sanskrit Language," in: 2017 IEEE 7th International Advance Computing Conference (IACC), 799–802.

Rijsbergen (1975): C.J. van Rijsbergen, Information Retrieval, Newton, MA.

Rockwell et al. (2012): Geoffrey Rockwell, Stéfan Sinclair and the Voyant Tools team, "Voyant Tools," https://voyant-tools.org/. (accessed on 30 August 2018)

Rose et al. (2010): Stuart Rose, Dave Engel, Nick Cramer, Wendy Cowley, "Automatic Keyword Extraction from Individual Documents," in: Michael W. Berry and Jacob Kogan (eds.), Text Mining, Chichester, UK, 1–20.

Sadeghi/Vegas (2014): Mohammad Sadeghi and Jésus Vegas, "Automatic Identification of Light Stop Words for Persian Information Retrieval Systems," Journal of Information Science 40 (4), 476–487.

Saif et al. (2014): Hassan Saif, Miriam Fernández, Yulan He and Harith Alani, "On Stopwords, Filtering and Data Sparsity for Sentiment Analysis of Twitter," in: Proceedings of the 9th Language Resources and Evaluation Conference (LREC), 810–817.

Saini/Rakholia (2016): Jatinderkumar R. Saini and Rajnish M. Rakholia, "On Continent and Script-Wise Divisions-Based Statistical Measures for Stop-Words Lists of International Languages," in: Procedia Computer Science 89, 313–19.

Salton/McGill (1983): Gerard Salton and Michael J. McGill, Introduction to Modern Information Retrieval, New York City.

Savoy (1999): Jacques Savoy, "A Stemming Procedure and Stopword List for General French Corpora," Journal of the American Society for Information Science 50 (10), 944–52.

Schofield et al. (2017): Alexandra Schofield, Måns Magnusson and David Mimno, "Pulling Out the Stops: Rethinking Stopword Removal for Topic Models," in: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, 432–36.

Silva/Ribiero (2003): Catarina Silva and Bernardete Ribeiro, "The Importance of Stop Word Removal on Recall Values in Text Categorization," in: Proceedings of the International Joint Conference on Neural Networks, 1661–1666.

Tijani et al. (2017): Olatunde D. Tijani, A. T. Akinwale, Saidat A. Onashoga and E. O. Adeleke, "An Auto-Generated Approach Of Stop Words Using Aggregated Analysis," in: Proceedings of the 13th International Conference of the Nigeria Computer Society, 99–115.

Weinberg (2004): Bella Hass Weinberg, "Predecessors of Scientific Indexing Structures in the Domain of Religion," in: Second Conference on the History and Heritage of Scientific and Technical Information Systems, 126–134.

Yao/Ze-wen (2011): Zhou Yao and Cao Ze-wen, "Research on the Construction and Filter Method of Stop-Word List in Text Preprocessing," in: Fourth International Conference on Intelligent Computation Technology and Automation, 217–21.

Zaman et al. (2011): A. N. K. Zaman, Pascal Matsakis and Charles Brown, "Evaluation of Stop Word Lists in Text Retrieval Using Latent Semantic Indexing," in: 2011 Sixth International Conference on Digital Information Management, 133–36.

Zheng/Gaowa (2010): Gong Zheng and Guan Gaowa, "The Selection of Mongolian Stop Words," in: IEEE International Conference on Intelligent Computing and Intelligent Systems, vol. 2, 71–74.

Zou et al. (2006): Feng Zou, Fu Lee Wang, Xiaotie Deng, Song Han and Lu Sheng Wang, "Automatic Construction of Chinese Stop Word List," in: Proceedings of the 5th WSEAS International Conference on Applied Computer Science, 1010–1015.

7. Author[56]

Patrick J. Burns

New York University
Institute for the Study of the Ancient World

Email: patrick.j.burns@nyu.edu

56 The rights pertaining to content, text, graphics, and images, unless otherwise noted, are reserved by the author. This contribution is licensed under CC BY-SA 4.0 International.