The Journal of Open Source Software

FreeStylo: An easy-to-use stylistic device detection tool for stylometry

Felix Schneider [1¶] **and Joachim Denzler** [1]

[1] Computer Vision Group, Friedrich-Schiller-Universität Jena ¶ Corresponding author

DOI: 10.21105/joss.07596

Software

- Review
- Repository
- Archive

Editor: Tristan Miller

Reviewers:

- @sara-02
- @rlskoeser

Submitted: 27 October 2024
Published: 27 January 2026

Summary

Stylistic devices are deliberately chosen linguistic expressions that are used to convey a certain meaning or effect. They are often used in literature to create a certain atmosphere or to convey a certain message. Due to this matter, the detection of stylistic devices in text is an important task in stylometry, the study of linguistic style. Often, finding these stylistic devices is a tedious and costly process that involves close reading of the texts, ideally by multiple experts. This is extra costly especially if researchers aim to statistically analyze the usage of stylistic devices across various texts.

To improve this state, this package provides an easy-to-use command-line interface for detecting stylistic devices in text. The tool can be configured with a simple configuration file. It is designed to be usable by both experts and non-experts in programming. For those proficient in Python, this package also provides a library with a collection of classes to detect stylistic devices in text, together with customizable text preprocessing (tokenizing, POS tagging, etc.).

The command-line tool is to be used on plain text files. It preprocesses the text, detects the stylistic devices specified in the configuration file, and writes the annotations to a JSON file. The classes contained in the library can be used to work with the text in a more flexible way, e.g. by using different preprocessing methods or already preprocessed text. The resulting annotations can be saved in a JSON file or directly used as a data structure in Python. Additionally the library is easily extendable with your custom stylistic device detectors.

Statement of need

Freestylo is a package that provides a collection of approaches to detect stylistic devices in text. While there exists a great variety of NLP libraries like NLTK (Bird & Loper, 2004), spaCy (Honnibal et al., 2020), or CLTK (Johnson et al., 2021), and command-line tools like CWB (Evert & Hardie, 2011) or UCS (Evert, 2005), for the processing and low-level analysis of text, there is a lack of tools that are specifically designed for the detection of stylistic devices. Current options in this space for figurative language would be the online tool Figurative Checker (Ahmad, 2023) or the MMFLD framework (Lai et al., 2023), which is a Python framework. Other frameworks would be Kühn et al. (2024) for the detection of antithesis, Li et al. (2023) for the detection of metaphors, Schneider et al. (2021) for the detection of chiasmus, or Dubremetz & Nivre (2017) for antimetabole detection. Other tools are available – e.g. Coles (2019) and Marozick et al. (2021) for the detection of rhymes.

However, most of those frameworks are not available as a ready-to-use tool, but as frameworks or code implementations of papers. Also, many are only available for a specific language. A commercial tool that was designed as an aid for writers is the ProWritingAid (ProWritingAid Team, 2025), which seems to be able to find various features of texts. However, the whole extent of this tool is not visible from its informational material. Another commercial tool which is able to find stylistic devices such as animalification, similes, imagery, onomatopoeia, epizeuxis, and anadiplosis is the Literary Device Analyzer (Aussie AI Team, 2025).

Despite the name similarity, this package is not related to the R stylo package (Eder et al., 2016). The R stylo package is a package for high-level analysis of the writing style in a stylometric context, e.g. for authorship attribution. While the results of this package can also be used to compare the styles of different authors, the focus of this package is on the detection of stylistic devices in text. The resulting detections can be used for various purposes, e.g. for the comparison of different text genres or time periods, or also for direct stylometry tasks like authorship attribution.

Information about the usage of stylistic devices is important for many branches of stylometry, especially for the analysis of literary texts. This package aims to fill this gap by providing an easy-to-use tool and library for the detection of those stylistic devices. Due to its simple and easily configurable command-line interface, the tool itself is geared not only to people with programming knowledge, but also to literary scholars who use distant reading methods in their research. Additionally, it supports multiple languages by design and is easy to extend to other languages. The software contained in this package is designed to be used either as a library, in other Python programs, or as a stand-alone command-line tool.

Design and supported stylistic devices

The package contains a collection of approaches to detect stylistic devices in text. By default, the preprocessing is done by spaCy (Honnibal et al., 2020) or CLTK (Johnson et al., 2021). Currently, the supported devices all work on a word level. They rely either on supporting words, a word pair, or multiple consecutive related words. It would be possible to extend the package with other stylistic devices that follow similar principles. Additionally, the structure of the underlying framework is not restricted to these kinds of devices. The same annotation method could be used to – for example – mark scene boundaries and topic boundaries or changes in the tense of the text. The following stylistic devices are currently supported:

Chiasmus

This package includes the current state-of-the-art approach by Schneider et al. (2021) to detect chiasmi in text. A chiasmus is a rhetorical device that consists of two parallel phrases, where the second phrase is a semantically related inversion of the first phrase. For example, the phrase "Hard is the task, the samples are few" is a phrase that emphasizes the problem of missing examples with the oppositional posing of the words "hard" and "few".

The chiasmus detector contained in this package has been trained using the dataset published by Schneider et al. (2023). It works for English, German, and Middle High German. It has been trained with word vectors provided by the German `de_core_news_lg` model by spaCy (Honnibal et al., 2020) However, since the model does not use the word vectors directly, but only their cosine similarity, it can be used with any word vectors, as long as they provide a vector for each token in the text.

The chiasmus detector needs some special lists to function properly:

- denylist: a list of part-of-speech (POS) tags that are not allowed to be part of a chiasmus.
- allowlist: a list of POS tags that are allowed to be part of a chiasmus. Be careful: if such a list is given, no other POS tags are allowed to be part of a chiasmus.
- neglist: a list of negations in the target language.
- conjlist: a list of conjunctions in the target language.

For English, German, and Middle High German, defaults for the lists are provided in the package. However, you can provide your own lists if you want to use the chiasmus detector for a different language or if you want to use a different set of POS tags, for example.

Metaphor

The metaphor detection approach in this package has specifically been developed for the low-resource language Middle High German, but can also be applied to more common high-resource languages. Specifically, adjective–noun metaphors like “thirsty car” are detected using a machine learning–based rating model. The detector is based on the publication by Schneider et al. (2022).

Currently, the metaphor detector is available for English and Middle High German. The word vectors are expected to be generated by the spaCy model `en_core_web_lg` for English and by the provided word vector FastText model for Middle High German.

Alliteration and alliterative verse

The package further contains a detector for both alliteration and alliterative verse. Alliteration comprise phrases where the initial letters of words are the same. Since alliteration is a simple stylistic device, the detector is based on a simple rule-based approach that orders all alliterations in the given text by the number of words that are alliterated in the phrase. Additionally, the detector can also find alliterative verses, which can contain some additional words in between the words with the same initial letter. An example for alliterative verse would be “Pondering on the pending paper, I programmed the Python package.” The user can specify the maximum number of words that are allowed to be in between the alliterated words, as well as words and POS tags that do not count towards the non-alliterated words. For example, spaCy also tags punctuation and newlines, so the user can specify those to be excluded from the alliteration.

Epiphora

An epiphora is a rhetorical device that consists of multiple parallel phrases, where the last word of each phrase is the same. For example, “I thought of the paper, I wrote the paper, I published the paper” is an epiphora that emphasizes the importance of the paper. The way this detector works is by splitting the text into sentences, and then those sentences into phrases. The detector searches for adjacent phrases that end with the same word. Those phrase collections are then sorted by the number of phrases in the collection.

Polysyndeton

A polysyndeton is a rhetorical device that consists of multiple parallel phrases, where each phrase is connected by a conjunction. For example, “I thought of the paper, and then I started writing it, and then I published it, and then I received a lot of citations” is a polysyndeton that, in a broader context with a slower feel to it, emphasizes the the process of writing and publishing a paper. The detector works by getting a list of all conjunctions, or by getting the POS tag of conjunctions, or by getting both, and then splitting the text into sentences, and then counting the conjunctions in each sentence, and then sorting the sentences by number of conjunctions.

Usage

The package can be used both as a library and as a stand-alone command-line tool. Both from the library and from the command-line tool, the results can be saved in a JSON file. This JSON file will contain the complete tokenized text. When using the functions from the library, the result will be a Python container with a similar structure to the JSON file.

The stand-alone version can be configured using a simple JSON configuration file. The file should specify the language of the text and the stylistic devices to detect. The following is an example configuration file:

```json
{
    "language": "de",
    "annotations": {
        "chiasmus": {
            "window_size": 30,
            "allowlist": ["NOUN", "VERB", "ADJ", "ADV"],
            "denylist": [],
            "model": "/chiasmus_de.pkl"
        }
    }
}
```

You can then run the tool using the following command:

```
freestylo --config config.json --input input.txt --output output.json
```

This will read the text from the file `input.txt`, preprocess (tokenize, POS-tag, etc.) the text, detect the stylistic devices specified in the configuration file, and write the results to the file `output.json`.

Creating your own detectors

The package is designed to be easily extendable with your own stylistic device detectors. The `src` folder contains example scripts that show how you can retrain the models for the existing chiasmus and metaphor detectors. You can also create your own stylistic device detectors by referring to the existing ones. The Alliteration Detector in particular provides a very simple example that can be used as a template for your own detectors. Please refer to the FreeStylo repository for more information on how to create your own detectors and contribute to the project. If you create and want to contribute your own detecors, pull requests are very welcome!

References

Ahmad, I. (2023). *Figurative checker online*. https://figurativechecker.com/

Aussie AI Team. (2025). *Literary device analyzer*. https://www.aussieai.com/editor/literary-device-analyzer

Bird, S., & Loper, E. (2004). NLTK: The Natural Language Toolkit. *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, 214–217. https://aclanthology.org/P04-3031

Coles, A. (2019). *Deep rhyme detection*. https://github.com/a-coles/deep-rhyme-detection

Dubremetz, M., & Nivre, J. (2017). Machine learning for rhetorical figure detection: More chiasmus with less annotation. In J. Tiedemann & N. Tahmasebi (Eds.), *Proceedings of the 21st Nordic Conference of Computational Linguistics (NoDaLiDa 2017)* (pp. 37–45). Linköping University Electronic Press. https://aclanthology.org/W17-0205/

Eder, M., Rybicki, J., & Kestemont, M. (2016). Stylometry with R: A package for computational text analysis. *The R Journal*, *8*(1), 107–121. https://doi.org/10.32614/RJ-2016-007

Evert, S. (2005). Empirical research on association measures: The UCS toolkit. *Phraseology 2005 Conference*. https://www.stephanie-evert.de/PUB/Evert2005phraseology.pdf

Evert, S., & Hardie, A. (2011). Twenty-first century Corpus Workbench: Updating a query architecture for the new millennium. *Proceedings of the Corpus Linguistics 2011 Conference*. http://www.birmingham.ac.uk/documents/college-artslaw/corpus/conference-archives/2011/Paper-153.pdf

Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A. (2020). *spaCy: Industrial-strength natural language processing in Python*. Zenodo. https://doi.org/10.5281/zenodo.1212303

Johnson, K. P., Burns, P. J., Stewart, J., Cook, T., Besnier, C., & Mattingly, W. J. B. (2021). The Classical Language Toolkit: An NLP framework for pre-modern languages. In H. Ji, J. C. Park, & R. Xia (Eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations* (pp. 20–29). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-demo.3

Kühn, R., Saadi, K., Mitrović, J., & Granitzer, M. (2024). Using pre-trained language models in an end-to-end pipeline for antithesis detection. In N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, & N. Xue (Eds.), *Proceedings of the 2024 Joint International Conference On Computational Linguistics, Language Resources And Evaluation* (pp. 17310–17320). https://aclanthology.org/2024.lrec-main.1502/

Lai, H., Toral, A., & Nissim, M. (2023). Multilingual multi-figurative language detection. In A. Rogers, J. Boyd-Graber, & N. Okazaki (Eds.), *Findings of the Association for Computational Linguistics: ACL 2023* (pp. 9254–9267). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.findings-acl.589

Li, Y., Wang, S., Lin, C., & Frank, G. (2023). Metaphor detection via explicit basic meanings modelling. *ArXiv*, *abs/2305.17268*. https://doi.org/10.48550/arXiv.2305.17268

Marozick, A., Elfandi, A., & Mayer, J. (2021). *RapAnalysis: Rhyme detection and analysis applied to user's Spotify data*. https://github.com/alexmarozick/RapAnalysis

ProWritingAid Team. (2025). *ProWritingAid: The storyteller's toolkit*. https://prowritingaid.com/

Schneider, F., Brandes, P., Barz, B., Marshall, S., & Denzler, J. (2021). Data-driven detection of general chiasmi using lexical and semantic features. In S. Degaetano-Ortlieb, A. Kazantseva, N. Reiter, & S. Szpakowicz (Eds.), *Proceedings of the 5th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature* (pp. 96–100). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.latechclfl-1.11

Schneider, F., Sickert, S., Brandes, P., Marshall, S., & Denzler, J. (2023). Hard is the task, the samples are few: A German chiasmus dataset. In Z. Vetulani & P. Paroubek (Eds.), *Human Language Technologies as a Challenge for Computer Science and Linguistics – 2023* (pp. 255–260).

Schneider, F., Sickert, S., Brandes, P., Marshall, S., & Denzler, J. (2022). Metaphor detection for low resource languages: From zero-shot to few-shot learning in Middle High German. In A. Bhatia, P. Cook, S. Taslimipoor, M. Garcia, & C. Ramisch (Eds.), *Proceedings of the 18th Workshop on Multiword Expressions @LREC2022* (pp. 75–80). European Language Resources Association. https://aclanthology.org/2022.mwe-1.11/