The Journal of Open Source Software

textNet: Directed, Multiplex, Multimodal Event Network Extraction from Textual Data

Elise Zufall[1] **and** [1]

1 UC Davis, USA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Introduction

A number of tools exist to generate networks based on co-occurrence of words within documents (such as the Nocodefunctions app (Levallois et al., 2012), the "textnets" package (Bail, 2024), InfraNodus (Paranyushkin, 2018), and many more). ## Statement of Need

Directed Graph Production

Multiplex Graph Output

Multimodal Graph Output

Existing packages such as the manynet package (Hollway, 2024) the default entity type tags for an NLP engine such as *spaCy* (Honnibal et al., 2021)),

Avoids Saturation

Installation

The stable version of this package can be installed from Github, using the *devtools* package (Wickham et al., 2022):

such as *spacyr* (Benoit et al., 2023), *pdftools* (Ooms, 2024), *igraph* (Csárdi et al., 2024), and *network* (Butts et al., 2023). To use the full functionality of *textNet*, such as pre-processing tools and post-processing analysis tools, we recommend installing these packages, which for *spacyr* requires integration with Python. However, the user may wish to preprocess and parse data using their own NLP engine, and skip directly to the textnet_extract() function, which does not depend on any of the aforementioned packages. The textnet_extract() function does, however, use functions from *pbapply* (Solymos et al., 2023), *data_table* (Barrett et al., 2024), *dplyr* (Wickham et al., 2023), and *tidyr* (Wickham et al., 2024).

Overview and Main Functions

- [OPTIONAL] Pre-processing: pdf_clean(), a wrapper for the pdftools::pdf_text() function which includes a custom header/footer text removal feature; and parse_text(), which is a wrapper for the *spacyr* package and uses the *spaCy* natural language processing engine (Honnibal et al., 2021) to parse text and perform part of speech tagging, dependency parsing, and named entity recognition (NER). Alternatively, as described below, the user can skip this step and load parsed text directly into the package.
- Network extraction: textnet_extract(), which generates a graph database from parsed text based upon tags and dependency relations
- Disambiguation: tools for cleaning, recoding, and aggregating node and edge attributes, such as the find_acronyms() function, which can be paired with the disambiguation() function to identify acronyms in the text and replace them with the full entity name.

- Exploration: the export_to_network() function for exporting the graph database to igraph and network objects, top_features() for viewing node and edge attributes, and combine_networks() for aggregating multiple document-based graphs based on common nodes.

Example

nst exogenous metadata that has been collected separately by the researcher regarding the different documents and their real-world context. The extracted networks, with their collections of verb attributes, node attributes, edge incidences, and edge attributes, can also be analyzed through a variety of tools, such as an Exponential Random Graph Model, to determine the probability of edge formation under certain conditions. A Temporal Exponential Random Graph Model could also shed light on the changes of a document over time, such as the multiple versions of the groundwater sustainability plan in this example.

Entity Network Extraction Algorithm

The directed network generated by *textNet* represents the collection of all identified entities in the document, joined by edges signifying the verbs that connect them. The user can specify which entity categories should be preserved. The output format is a list containing four data.tables: an edgelist, a nodelist, a verblist, and an appositive list.

The edgelist includes edge attributes such as verb tense, any auxiliary verbs in the verb phrase, whether an open clausal complement (Universal Dependencies code “xcomp”) is associated with the primary verb, whether any hedging words were detected in the sentence, and whether any negations were detected in the sentence.

The returned edgelist by default contains both complete and incomplete edges. A complete edge includes a source, verb, and target. An incomplete edge includes either a source or a target, but not both, along with its associated verb. Incomplete edges convey information about which entities are commonly associated with different verbs, even though they do not reveal information about which other entities they are linked to in the network. These incomplete edges can be filtered out when converting the output into a network object, such as through the *network* package or the *igraph* package. The nodelist returns all entities of the desired types found in the document, regardless of whether they were found in the edgelist. Thus, the nodelist allows the presence of isolates to be documented, as well as preserving node attributes. The verblist includes all of the verbs found in the document, along with verb attributes imported from *VerbNet* (Kipper-Schuler, 2006). This can be used to conduct analyses of certain verb classifications of interest. Finally, the appositive list is a table of entities that may be synonyms. This list is generated from entities whose universal dependency parsing labels as appositives, and whose head token points to another entity. These pairs are included in the table as potential synonyms. If this feature is used, cleaning and filtering by hand is recommended, as appositives can at times be misidentified by existing NLP tools. An automated alternative we recommend is our find_acronym tool, which scans the entire document for acronyms defined parenthetically in-text and compiles them in a table.

This network is directed such that the entities that form the subject of the sentence are denoted as the “source” nodes, and the remaining entities are denoted as the “target” nodes. To identify whether each entity is a “source” or a “target”, we use dependency parsing in the Universal Dependencies format, in which each token in a given sentence has an associated “syntactic head” token from which it is derived. Starting with each entity in the sentence, the chain of syntactic head tokens is traced back until either a subject or a verb is reached. If it reaches a subject first, the entity is considered a “source.” If it reaches a verb first, it is considered a “target.”

To identify the subject, we search for the presence of at least one of the following subject tags: “nsubj” (nominal subject), “nsubjpass” (nominal subject – passive), “csubj” (clausal subject), “csubjpass” (clausal subject – passive), “agent”, and “expl” (expletive). To identify the object, we search for the presence of at least one of the following: “pobj” (object of preposition), “iobj” (indirect object), “dative”, “attr” (attribute), “dobj” (direct object), “oprd” (object predicate), “ccomp” (clausal complement), “xcomp” (open clausal complement), “acomp” (adjectival complement), or “pcomp” (complement of preposition).

If a subject token is reached first (“nsubj,” “nsubjpass,” “csubj,” “csubjpass,” “agent,” or “expl”), this indicates that the original token is doing the verb action. That is, it serves some function related to the subject of the sentence. We designate this by tagging it “source,” since these types of relationships will be used to designate the “from” or “source” nodes in our directed network. If a verb token is reached first (“VERB” or “AUX”), this indicates that the verb action is occurring for or towards the original token, which we denote with the tag “target.” These tokens are potential “to” or “target” nodes in our directed network. Linking the two nodes is an edge representing the verb that connects them in the sentence.

Due to the presence of tables, lists, or other anomalies in the original document, it is possible that a supposed “sentence” has a head token trail that does not lead to a verb as is normatively the case. In these instances, the tokens whose trails terminate with a non-subject, non-verb token are assigned neither “source” nor “target” tags. Finally, an exception is made if an appositive token is reached first, since this indicates that the token in question is merely a synonym or restatement of an entity that is already described elsewhere in the sentence and, accordingly, should not be treated as a separate node. Tokens that lead to appositives are assigned neither “source” nor “target” tags, but are preserved as a separate appositive list.

If a verb phrase in the edgelist does not have any sources, the sources associated with the head token of the verb phrase’s main verb (that is, the verb phrase’s parent verb) are adopted as sources of that verb phrase. As of Version 1.0, *textNet* does not do this recursively, to preserve performance optimization.

The textNet::textnet_extract() function returns the full list of open clausal complement lemmas associated with the main verb as an edge attribute: “xcomp_verb”. The list of auxiliary verbs and their corresponding lemmas associated with the main verb, as well as the list of auxiliary verbs and corresponding lemmas associated with the open clausal complements linked to the main verb, are also included as edge attributes: “helper_token”, “helper_lemma”, “xcomp_helper_token”, and “xcomp_helper_lemma”, respectively.

The extraction function also detects hedging words and negations. The function textNet::textnet_extract() produces an edge attribute “has_hedge”, which is T if there is a hedging auxiliary verb (“may”,“might”,“can”,“could”) or main verb (“seem”,“appear”,“suggest”,“tend”,“assume”,“indicate”,“estimate”,“doubt”,“believe”) in the verb phrase.

Tense is also detected. The six tenses tagged by *spaCy* in textNet::parse_text() are preserved by textNet::textnet_extract() as an edge attribute “head_verb_tense”. This attribute can take on one of six values: “VB” (verb, base form), “VBD” (verb, past tense), “VBG” (verb, gerund or present participle), “VBN” (verb, past participle), “VBP” (verb, non-3rd person singular present), or “VBZ” (verb, 3rd person singular present). Additionally, an edge attribute “is_future” is generated by textNet::textnet_extract(), which is T if the verb phrase contains an xcomp, has the token “going” as a head verb, and a being verb token as an auxiliary verb (i.e. is of the form “going to ”) or contains one of the following auxiliary verbs: “shall”,“will”,“wo”, or “’ll” (i.e. is of the form “will ”).

Acknowledgements

The authors gratefully acknowledge the support of the Sustainable Agricultural Systems program, project award no. 2021-68012-35914, from the U.S. Department of Agriculture’s National Institute of Food and Agriculture and the National Science Foundation’s Dynamics of Integrated Socio-Environmental Systems program, grant no. 2205239.

Appendix

This appendix describes the pre-processing tools available through the *textNet* package, which enable the user to generate the data frame expected by the textnet_extract() function.

Pre-Processing Step I: Process PDFs

This is a wrapper for pdftools, which has the option of using pdf_text or OCR. We have also added an optional header/footer removal tool. This optional tool is solely based on carriage returns in the first or last few lines of the document, so may inadvertently remove portions of paragraphs. However, not removing headers or footers can lead to improper inclusion of header and footer material in sentences, artificially inflating the presence of nodes whose entity names are included in the header and footer. Because of the risk of headers and footers to preferentially inflate the presence of a few nodes, the header/footer remover is included by default. It can be turned off if the user has a preferred header/footer removal tool to use instead, or if the input documents lack headers and footers.

```
library(textNet)
library(stringr)
URL <- "https://sgma.water.ca.gov/portal/service/gspdocument/download/2840"
download.file(URL, destfile = "old.pdf", method="curl")

URL <- "https://sgma.water.ca.gov/portal/service/gspdocument/download/9625"
download.file(URL, destfile = "new.pdf", method="curl")

pdfs <- c("old.pdf",
       "new.pdf")

old_new_text <- textNet::pdf_clean(pdfs, keep_pages=T, ocr=F, maxchar=10000,
                  export_paths=NULL, return_to_memory=T, suppressWarn = F, auto_headf
names(old_new_text) <- c("old","new")
```

Pre-Processing Step II: Parse Text

This is a wrapper for the pre-trained multipurpose NLP model *spaCy* (Honnibal et al., 2021), which we access through the R package *spacyr* (Benoit et al., 2023). It produces a table that can be fed into the textnet_extract function in the following step. To initialize the session, the user must define the "RETICULATE_PYTHON" path, abbreviated as "ret_path" in *textNet*, as demonstrated in the example below. The page contents processed in the Step 1 must now be specified in vector form in the "pages" argument. To determine which file each page belongs to, the user must specify the file_ids of each page. We have demonstrated how to do this below. The package by default does not preserve hyphenated terms, but rather treats them as separate tokens. This can be adjusted.

The user may also specify "phrases_to_concatenate", an argument representing a set of phrases for spaCy to keep together during its parsing. The example below demonstrates how to use this feature to supplement the NER capabilities of spaCy with a custom list of entities. This supplementation could be used to ensure that specific known entities are recognized; for instance, spaCy might not detect that a consulting firm such as "Schmidt and Associates" is one entity rather than two. Conversely, this capability could be leveraged to create a new category of entities to detect, that a pretrained model is not designed to specifically recognize. For instance, to create a public health network, one might include a known list of contaminants and diseases and designate custom entity type tags for them, such as "CONTAM" and "DISEASE"). In this example, we investigate the connections between the organizations, people, and geopolitical entities discussed in the plan with the flow of water in the basin. To assist with this, we have input a custom list of known water bodies in the region governed by our test document and have given it the entity designation "WATER". This is carried out by setting the variable "phrases_to_concatenate" to a character vector, including all of the custom entities. Then, the entity type can be set to the desired category. Note that this function is case-sensitive.

```
   library(findpython)
   ret_path <- find_python_cmd(required_modules = c('spacy', 'en_core_web_lg'))


   water_bodies <- c("surface water", "Surface water", "groundwater", "Groundwater", "Sa
Chowchilla canal", "lower aquifer", "upper aquifer", "upper and lower aquifers", "lower

   old_new_parsed <- textNet::parse_text(ret_path,
                          keep_hyph_together = F,
                          phrases_to_concatenate = water_bodies,
                          concatenator = "_",
                          text_list = old_new_text,
                                 parsed_filenames=c("old_parsed","new_parsed"),
                                 overwrite = T,
                          custom_entities = list(WATER = water_bodies))
```

Another NLP tool may be used instead of the built-in *textNet* function at this phase, as long as the output conforms to spaCy tagging standards: Universal Dependencies tags for the "pos" part-of-speech column (Nivre, 2017), and Penn Treebank tags for the "tags" column (Marcus et al., 1999). The textnet_extract function expects the parsed table to follow specific conventions. First, a row must be included for each token. The column names expected by textnet_extract are:

- doc_id, a unique ID for each page
- sentence_id, a unique ID for each sentence
- token_id, a unique ID for each token
- token, the token, generally a word, represented as a string
- lemma, the canonical or dictionary form of the token
- pos, a code referring to the token's part of speech, defined according to Universal Dependencies (Nivre, 2017).
- tag, a code referring to the token's part of speech, according to Penn Treebank (Marcus et al., 1999).
- head_token_id, a numeric ID referring to the token_id of the head token of the current row's token
- dep_rel, the dependency label according to ClearNLP Dependency labels (Choi, 2024)
- entity, the entity type category defined by OntoNotes 5.0 (Weischedel et al., 2012). This is represented as as string, ending in "_B" if it is the first token in the entity or "_I" otherwise).

References

Bail, C. (2024). *Cbail/textnets* (Version 0.1.1). https://github.com/cbail/textnets

Barrett, T., Dowle, M., Srinivasan, A., Gorecki, J., Chirico, M., Hocking, T., Schwendinger, B., Stetsenko, P., Short, T., Lianoglou, S., Antonyan, E., Bonsch, M., Parsonage, H., Ritchie, S., Ren, K., Tan, X., Saporta, R., Seiskari, O., Dong, X., … Krylov, I. (2024). *Data.table: Extension of 'data.frame'* (Version 1.16.2). https://cran.r-project.org/web/packages/data.table/index.html

Benoit, K., Matsuo, A., Gruber, J., & Council (ERC-2011-StG 283794-QUANTESS), E. R. (2023). *Spacyr: Wrapper to the 'spaCy' 'NLP' library* (Version 1.3.0). https://cran.r-project.org/web/packages/spacyr/index.html

Butts, C. T., Hunter, D., Handcock, M., Bender-deMoll, S., Horner, J., Wang, L., Krivitsky, P.

N., Knapp, B., Bojanowski, M., & Klumb, C. (2023). *Network: Classes for relational data* (Version 1.18.2). https://cran.r-project.org/web/packages/network/index.html

Choi, J. (2024, August 7). *ClearNLP dependency labels. ClearNLP guidelines*. https://github.com/clir/clearnlp-guidelines/blob/master/md/specifications/dependency_labels.md

Csárdi, G., Nepusz, T., Traag, V., Horvát, S., Zanini, F., Noom, D., Müller, K., Salmon, M., Antonov, M., & details, C. Z. I. igraph author. (2024). *Igraph: Network analysis and visualization* (Version 2.1.1). https://cran.r-project.org/web/packages/igraph/index.html

Hollway, J. (2024). *Manynet: Many ways to make, modify, map, mark, and measure myriad networks* (Version 1.2.6). https://CRAN.R-project.org/package=manynet

Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A. (2021). *spaCy: Industrial-strength natural language processing in python* (Version 3.1.3). https://github.com/explosion/spaCy/tree/master

Kipper-Schuler, K. (2006). *VerbNet* (Version 3.3). University of Colorado Boulder. https://verbs.colorado.edu/verb-index/vn3.3/index.php

Levallois, C., Clithero, J. A., Wouters, P., Smidts, A., & Huettel, S. A. (2012). Translating upwards: Linking the neural and social sciences via neuroeconomics. *Nature Reviews Neuroscience*, *13*(11), 789–797. https://nocodefunctions.com/cowo/semantic_networks_tool.html

Marcus, M., Santorini, B., Marcinkiewicz, M. A., & Taylor, A. (1999). *Treebank-3 documentation*. https://doi.org/https://doi.org/10.35111/gq1x-j780

Nivre, J. (2017). *Universal POS tags* (Version 2). Universal Dependencies. https://universaldependencies.org/u/pos/

Ooms, J. (2024). *Pdftools: Text extraction, rendering and converting of PDF documents* (Version 3.4.1). https://cran.r-project.org/web/packages/pdftools/index.html

Paranyushkin, D. (2018). *InfraNodus*. Nodus Labs. https://infranodus.com/

Solymos, P., Zawadzki, Z., Bengtsson, H., & Team, R. C. (2023). *Pbapply: Adding progress bar to '*apply' functions* (Version 1.7-2). https://cran.rstudio.com/web/packages/pbapply/index.html

Weischedel, R., Pradhan, S., Ramshaw, L., Kaufman, J., Franchini, M., El-Bachouti, M., Xue, N., Palmer, M., Hwang, J. D., Bonial, C., Choi, J., Mansouri, A., Foster, M., Hawwary, A., Marcus, M., Taylor, A., Greenberg, C., Hovy, E., Belvin, R., & Houston, A. (2012). *OntoNotes release 5.0 with OntoNotes DB tool v0.999 beta*. BBN Technologies. https://catalog.ldc.upenn.edu/docs/LDC2013T19/OntoNotes-Release-5.0.pdf

Wickham, H., François, R., Henry, L., Müller, K., Vaughan, D., Software, P., & PBC. (2023). *Dplyr: A grammar of data manipulation* (Version 1.1.4). https://cran.r-project.org/web/packages/dplyr/index.html

Wickham, H., Hester, J., Chang, W., Bryan, J., & RStudio. (2022). *Devtools: Tools to make developing r packages easier* (Version 2.4.5). https://cran.r-project.org/web/packages/devtools/index.html

Wickham, H., Vaughan, D., Girlich, M., Ushey, K., Software, P., & PBC. (2024). *Tidyr: Tidy messy data* (Version 1.3.1). https://cran.r-project.org/web/packages/tidyr/index.html