Amsterdam University Press

COMPUTATIONAL COMMUNICATION RESEARCH 3.2 (2021) 180-194
HTTPS://DOI.ORG/10.5117/CCR2021.2.003.WELB

Extracting semantic relations using syntax

An R package for querying and reshaping dependency trees.

Kasper Welbers
VU University Amsterdam

Wouter van Atteveldt
VU University Amsterdam

Jan Kleinnijenhuis
VU University Amsterdam

Abstract

Most common methods for automatic text analysis in communication science ignore syntactic information, focusing on the occurrence and co-occurrence of individual words, and sometimes n-grams. This is remarkably effective for some purposes, but poses a limitation for fine-grained analyses into semantic relations such as *who does what to whom* and *according to what source*. One tested, effective method for moving beyond this bag-of-words assumption is to use a rule-based approach for labeling and extracting syntactic patterns in dependency trees. Although this method can be used for a variety of purposes, its application is hindered by the lack of dedicated and accessible tools. In this paper we introduce the rsyntax R package, which is designed to make working with dependency trees easier and more intuitive for R users, and provides a framework for combining multiple rules for reliably extracting useful semantic relations.

Introduction

Applications of automatic text analysis in social scientific research often rely on the *bag-of-words* assumption (Boumans & Trilling, 2016; Grimmer & Stewart, 2013). Texts are broken up into individual words, and hence represented only by their frequencies, regardless of the way in which these words are related to each other in syntax, or even word order and distance.

This approach culls a substantial part of the information contained in a text, but has been proven to be very useful for a wide variety of text analysis tasks. From supervised machine learning techniques for measuring document or sentence level sentiment (Barberá, Boydstun, Linn, McMahon, & Nagler, 2016), to unsupervised techniques for automatically classifying texts into topics (Roberts et al., 2014), the mere frequencies of individual words seem to contain sufficient information.

However, for certain types of analysis the syntactic relations between words are critical. If a text mentions two people and indicates a conflict, it would be rash to assume that the conflict involved these two people, let alone draw any conclusions regarding who initiated the conflict. Information pertaining to these semantic relations is encoded in syntax. This is also why state-of-the-art machine learning approaches for tasks such as opinion mining and machine translation rely on deep learning models that can take the relations between words into account (Goldberg, 2017; Nakov, Ritter, Rosenthal, Sebastiani, & Stoyanov, 2016).

In this paper we present the rsyntax R package, which facilitates a versatile and transparent method for using syntax in text analysis by making it easy to extract information from syntactic dependency trees. Dependency trees contain information about the syntax of a sentence in a graph representation, where every word is connected to another word with a certain syntactic relation. This formal representation of syntax makes it possible to perform a query search on syntactic patterns (Luotolahti, Kanerva, Pyysalo, & Ginter, 2015; Shlain, Taub-Tabib, Sadde, & Goldberg, 2020), enabling a rule-based analysis. This type of analysis can be used, for instance, to extract product attributions from reviews (Poria, Cambria, Ku, Gui, & Gelbukh, 2014), biological facts from scientific literature (Fundel, Küffner, & Zimmer, 2007), events data from news reports (Schrodt, 2015) or source-quote and subject-predicate clauses from conflict coverage (Van Atteveldt, 2008; Van Atteveldt, Sheafer, Shenhav, & Fogel-Dror, 2017).

In a way, this brings us full circle to early research in communication science that focused on extracting semantic relations (e.g. Lasswell & Leites, 1965; Osgood, Saporta, & Nunnally, 1956; Van Cuilenburg, Kleinnijenhuis, & De Ridder, 1986). It also connects with current work that uses 'core sentences ' or political claims as a unit of analysis rather than whole texts (Kleinnijenhuis, van Hoof, & van Atteveldt, 2019; Koopmans & Statham, 1999; Wueest, Clematide, Bünzli, Laupper, & Frey, 2011).

There are existing systems for querying graph data, but using theses systems for extracting information from dependency trees requires a fairly high-level understanding of graph theory and coding (Shlain et al., 2020).

rsyntax is specialized for working with dependency trees in a way that is accessible and intuitive for anyone with a basic understanding of R and data frames. By focusing on a common type of dependency tree where each token (i.e. word) only has a single parent, the complete syntactic information can be represented as columns in a data frame where each row is a word. In this format, querying syntactic patterns can be understood as a special form of selecting rows. Using visualization tools and a specialized system for querying and transforming dependency trees, users can extract useful semantic relations, and add this information to the data frame for further analysis.

Extracting semantic relations from texts

The general purpose of rsyntax is to provide a flexible system for labeling and extracting any type of syntactic pattern. But as a tool developed primarily for communication scientists, it is specialized for the extraction of semantic relations such as *who says what* and *who does what*. This application of rsyntax can be seen as a "specialized and simplified version of Semantic Role Labeling (SRL)" (Van Atteveldt et al., 2017). Labels are assigned to words to indicate their semantic role in a sentence, for example that John is the agent in the sentence "the window was broken by John" (Jurafsky & James, 2020). This enables us to refine frequency-based text analysis by focusing on specific components, such as things said or done by or to certain people or organizations.

The field of Semantic Role Labeling (SRL) is still in very active development (see e.g., He, Lee, Lewis, & Zettlemoyer, 2017; Zhou & Xu, 2015). Even though performance is rapidly increasing, accuracy remains limited, especially outside of the English language. This is partly due to the ambitious scope of most state-of-the-art semantic role parsers, which are developed to distinguish many (ideally all) possible semantic roles. For applications in communication science, this level of detail is often not needed. An effective alternative is therefore to extract only the semantic relations that a researcher is interested in by labeling and transforming syntactic dependency trees (Van Atteveldt, 2008; Van Atteveldt, Sheafer, Shenhav, & Fogel-Dror, 2017).

State-of-the-art syntactic dependency parsers are widely available, and some can also be used directly from within R. With rsyntax the researcher can develop custom rules to give any label to any syntactic pattern, which makes the process very flexible and transparent. The limitation of this approach is that semantic relations can often be expressed with many different syntactic patterns, and writing rules for all variations in a corpus by hand is not always feasible. Yet, prior studies show that only a few rules

are often required to capture the most dominant types of expression (see e.g., Author, 2017; Fundel et al., 2007; Poria et al., 2014).

Working with syntactic dependency trees

The method and tool discussed in this paper require that texts have first been preprocessed with a natural language processing (NLP) pipeline that includes a *dependency parser*. There are several packages in R that support dependency parsing, such as spacyr (Benoit & Matsuo, 2017; Honnibal & Johnson, 2015), and udpipe (Straka & Straková, 2017; Wijffels, 2019). Working with rsyntax requires a basic understanding of the data produced by these pipelines, and of dependency-based grammar in particular (for an excellent introduction, see chapter 15 of Jurafsky and James 2020).

Table 1 shows the output of the spaCy pipeline for the sentence "Trump said that Biden is the dumbest of all candidates" in a data frame format. Each row represents a unique *token* (i.e. word or punctuation), and the columns contain information from several text preprocessing techniques. The *lemma* (lemmatization) and *pos* (part-of-speech tagging) columns are relatively straightforward to use because they contain information about individual tokens. The dependency tree information, which is represented in the *parent* and *relation* columns, is more complicated to use because it contains information about the relations between tokens. To work with this data, we need to understand the general properties of this graph (i.e. network) data, and how it is represented in the data frame.

A dependency tree is a set of labeled dependency relations between the tokens of a sentence (Kübler, McDonald, & Nivre, 2009, 12), that is represented as a directed acyclic graph (see e.g., Chen & Manning, 2014; Manning et al., 2014). Since each token in the tree can only have a single parent, it is possible to represent the complete graph in the tokens data frame. The number in the *parent* column refers to the *token_id* of the parent, and the *relation* column indicates the type of dependency relation. For example, for the "Trump" token the parent value is 2, which indicates that "said" is the parent. The relation is *nsubj*, which indicates that "Trump" is the *nominal subject* of the verb "said". From this information we can infer that Trump is the one who said something.

Having data on these syntactic structures allows us to analyze texts based on general patterns in how people say something rather than what they say specifically. People can be very creative in stringing words together, but sentences do need to adhere to certain syntactic structures in order to make sense. Instead of "Trump said that Biden is the dumbest candidate", one could

Table 1 Output of the spaCy parser

sentence	token_id	token	lemma	pos	parent	relation
1	1	Trump	trump	PROPN	2	nsubj
1	2	said	say	VERB		ROOT
1	3	that	that	ADP	5	mark
1	4	Biden	Biden	PROPN	5	nsubj
1	5	is	be	VERB	2	ccomp
1	6	the	the	DET	7	det
1	7	dumbest	dumb	ADJ	5	attr
1	8	of	of	ADP	7	prep
1	9	all	all	DET	10	det
1	10	candidates	candidate	NOUN	8	pobj

also have said "Biden is the dumbest candidate, says Trump", or even "The 'dumbest candidate ', Trump was saying, 'is Biden ' ". Despite differences in word order, the subject-verb structure of these sentences can be used to determine who-said-what. Moreover, the syntactic structure can be the same even if the specific words change, for instance in "Biden said that Trump is the worst president". Therefore, by developing queries for a few common syntactic patterns, we can extract information from a huge variety of specific sentences.

The rsyntax package facilitates this methodological approach in three ways. First, dependency trees can be visualized in a way that makes it easier to see and compare the syntactic structures of sentences. Second, a versatile system for creating, combining and applying queries for syntactic patterns is provided. Third, for more advanced applications, tools are provided for transforming sentence structures (Harris, 1957), which can help dealing with subordinate clauses (i.e. nested sentences).

Visualizing dependency trees

To make working with dependency data easier rsyntax can visualize token data in a way that conveys both the data frame structure and the dependency tree. The following code produces Figure 1 based on output from the spaCy pipeline.

```
library(spacyr)
library(rsyntax)

tokens = spacy_parse("Trump said that Biden is the dumbest of
all candidates", dependency=T)
plot_tree(tokens)
```

The words of the sentence are presented horizontally at the bottom, optionally with other columns such as the *lemma* and *pos* printed below them. The tree illustrates the syntactic relations between these tokens. The nodes of the graph contain the token_id, and the edges indicate the dependency relations between tokens. The higher node is the parent token, and the lower node is the child token. The type of relation is printed above the child node in italics.

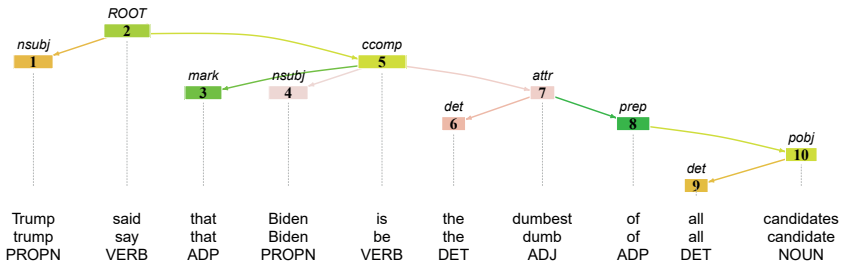

Figure 1 Dependency tree for "Trump said that Biden is the dumbest of all candidates"

This visualization contains a lot of information, but once familiarized it makes it much easier to detect patterns in syntactic structure. We can see in one glance that "Trump" is the nominal subject (nsubj) of the verb "said". We can also see longer and more complicated patterns, such as the fact that the rest of the sentence is directly or indirectly a child of "said". Being able to quickly see the dependency tree for a given sentence is a great help in developing and testing queries. Using example sentences to determine what common patterns to look for also makes the method more approachable for users without advanced linguistic knowledge.

Querying dependency trees

The core functionality of rsyntax is the system for querying syntactic patterns. Here we briefly discuss the general design. As an example case, consider that we would want to compare statements made by Trump and Biden in the news. To do this, we need to query syntactic patterns that indicate that someone is the source of a statement.

Developing a query. Queries in rsyntax need to account for two things: the selection of tokens and the relations between these tokens. For extracting statements made by Trump and Biden, a good place to start is to find verbs that indicate speech. The following code creates a *tquery* (tree query) for finding all tokens with the part-of-speech (pos) tag "VERB" and a lemma that indicates speech.

```
speech_verbs = c("say", "state") ## etc.

source_said = tquery(pos = "VERB", lemma = speech_verbs)
```

In Table 1, this query would match the token "said". To go from this single token to a pattern of tokens, we add conditions for its parent or children. In the following code we say that the verb needs to have a child with a "nsubj" relation. This is achieved by nesting the *children* function within the *tquery*.

```
source_said = tquery(pos = "VERB", lemma = speech_verbs,
    children(relation = "nsubj"))
```

Notice how the *children* function itself only specifies selection criteria for rows from the tokens data frame. By applying these selection criteria to the children of another selection, which can also be within another *children* function, any tree shaped pattern can be created. Thus, querying a pattern in a dependency tree can be understood as nested row selection operations, and does not require learning a full graph querying language.

The next step in our example is to also add the quote or paraphrase to the pattern. A rough but effective approach is to take all remaining children of the verb, which we can do by adding a second children function without any conditions. In addition, we are now going to add labels to the query. This way, we will not only look whether a source-verb-quote pattern exists, but also distinguish these three components.

```
source_said = tquery(label = "quote", pos = "VERB", lemma =
speech_verbs,
    children(label = "source", relation = "nsubj"),
    children(label = "quote"))
```

In summary, we now have a query that matches (1) a verb that indicates speech, that has (2) a child with a subject relation and (3) other children. These components are labeled "verb", "source" and "quote", respectively. We can now use this query to annotate the tokens data, using the 'annotate_tqueries()' function.

```
tokens = annotate_tqueries(tokens, "verb", source_said)
```

To verify that the query works as intended, the annotations can also be added to the tree visualization.

```
plot_tree(tokens, annotation="quote")
```

Figure 2 shows this visualization for a new example sentence. The outer box indicates which tokens are part of the pattern, and the inner boxes indicate the labels assigned to the pattern components. The speech indicating verb of this sentence is "said". The subject of this verb is "Biden", and the remaining children are "cloaked" and the comma. Our query matches these tokens, and assigns labels to indicate that "Joe Biden" (source) is the one who "said" (verb) that "Trump has 'cloaked America in darkness for much too long ',"" (quote).

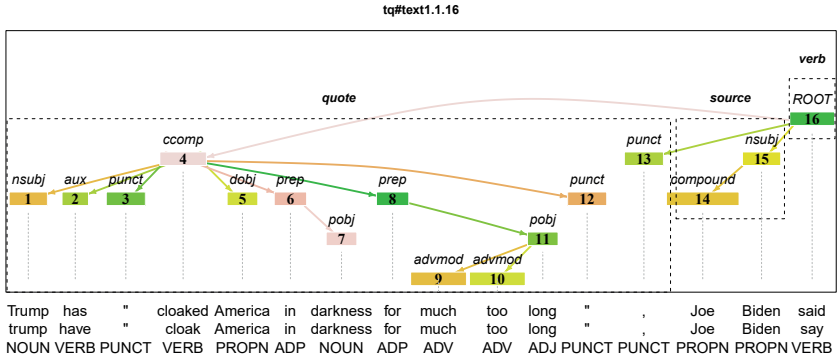

Figure 2 Dependency tree with annotations for source, verb and quote.

Notice that the entire sentence is labelled, even though the query only matched four tokens (Biden, said, cloaked, and the comma). This is because of a special functionality for recursively annotating children of a matched token with the same semantic role label, which we call the *fill* heuristic. The default behavior of this heuristic often makes sense because a child in a dependency tree contains information related to the parent. However, labelling *all* children is sometimes too crude. For better precision it is therefore possible and recommended to customize the fill heuristic. For instance, we can make a special fill heuristic for subjects that only labels compound relations (like "Joe" in "Joe Biden") or that stops when it reaches a subordinate clause.

Here we only demonstrated a relatively simple query to illustrate the gist of the method. The purpose of the tquery function is that any pattern can be matched, and more features such as the fill heuristic are provided to make it easier to match useful patterns. An ongoing development goal is to make this as easy and flexible as possible, building on user feedback from the community.

Chaining multiple queries. Our example query captures a common syntactic structure for indicating sources, but for better recall on our task of finding statements from Trump and Biden, we will need to add queries for common alternatives. For example, Figure 3 shows the dependency tree of a sentence where the source is indicated with "according to". This is a rather common pattern, but to match it we need a slightly more complicated query that contains several nested children.

```
according_to_source = tquery(label = "quote", pos = "VERB",
    children(label = "verb", lemma="accord",
        children(lemma = "to",
            children(label = "source"))))
```

To obtain good accuracy we need to at least develop queries for the most common syntactic structures within a given use case (e.g., quotes from politicians in news) but any number of queries can be used to improve accuracy. An important functionality of rsyntax is the possibility to combine multiple queries in a *chain*.

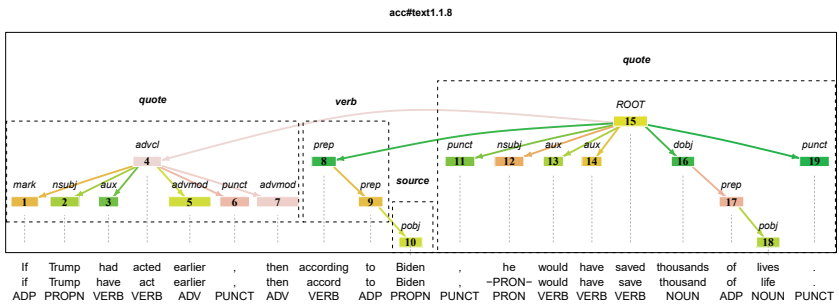

Figure 3 Dependency tree for "If Trump had acted earlier, then according to Biden, he would have saved thousands of lives."

```
chain = list(source_said, according_to_source)
tokens = annotate_tqueries(tokens, "quote", chain)
```

All results are then combined, and certain priority rules are applied so that tokens can only be annotated once. Most importantly, queries earlier in the chain are given priority, which can be used to systematically improve accuracy. New queries can be added to the end of the chain to improve recall without intervening with prior queries. If there are patterns that are

incorrectly matched, a more specific query can be placed in front of the responsible query to improve precision.

Comparing statements from Trump and Biden. After developing and validating a chain of queries for extracting source-verb-quote relations, we can return to our hypothetical task of comparing statements from Trump and Biden. By applying the queries on the tokens data frame, two columns are added: one specifying the unique id of the pattern and one specifying the label (Table 2).

Table 2 Tokens data frame after applying tqueries (abbreviated).

sent	token_id	token	lemma	pos	parent	relation	pattern_id	pattern
1	1	Trump	trump	PROPN	2	nsubj	dir#1.1.2	source
1	2	said	say	VERB		ROOT	dir#1.1.2	verb
1	3	that	that	ADP	5	mark		
1	4	Biden	Biden	PROPN	5	nsubj	dir#1.1.2	quote
1	5	is	be	VERB	2	ccomp	dir#1.1.2	quote
…	…	…	…	…	…	…	…	…

We can now select the patterns for which Trump or Biden was the source, and use the quote tokens to create separate corpora or document-term matrices containing only the words from their statements. We can then use any (bag-of-words type) text analysis technique to analyze and compare statements from Biden and Trump.

Reshaping dependency trees

A complication for querying dependency trees is that sentences often contain implicit relations. For example, the sentence "Jebediah reformed the economy and balanced the budget" (Figure 4) contains two semantic relations: "Jebediah reformed the economy" and "Jebediah balanced the budget". For the second relation the subject (Jebediah) is redundant by convention, and therefore not repeated. In the dependency tree this implicit subject relation is also not included. This means that a query that looks for a verb with a direct subject relation would not find the relation between "Jebediah" and "balanced".

It is possible to write additional queries to extract these implicit relations. Also, for broadly extracting text of things done by Jebediah, the fill heuristic (in combination with design details not discussed here) also provides an easy solution with decent accuracy. However, a more powerful solution would be to first reshape the dependency trees so that implicit relations

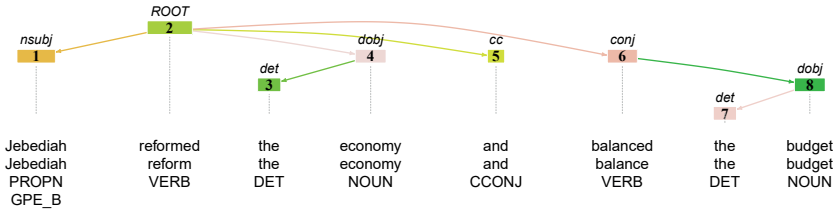

Figure 4 Dependency graph for the sentence "Jebediah reformed the economy and balanced the budget.".

become explicit. The Universal Dependencies framework has an enhanced representation that "aims to make implicit relations between content words more explicit by adding relations and augmenting relation names" (Schuster & Manning, 2016, 2372). The implicit subject relation between "Jebediah" and "balanced" is then made explicit by adding an additional *nsubj* relation from "balanced" to "Jebediah".

A consequence of this solution is that it violates the single parent property. This means that the enhanced dependency tree cannot be represented as a data frame (without having nested values). To include implicit relations, the implicit tokens would also have to be copied. This type of transformation is very similar to *text simplification*, in which sentences are broken up into less complicated components. Text simplification can be performed by applying certain operations on the dependency tree (see e.g., Siddharthan & Mandya, 2014). The rsyntax package therefore includes tools for performing these types of operations.

An example is given in Figure 5, where the example sentence is split into two sentences by resolving the conjunction. The visualization function in rsyntax automatically plots the now disconnected trees side-by-side, with the copied words colored red. To demonstrate how this makes it easier to query the dependency tree, we applied a simple query for subject-verb-object relations.

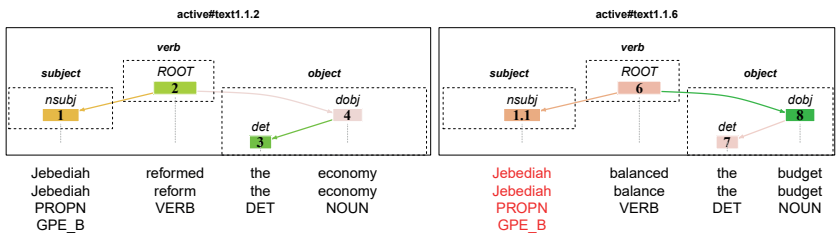

Figure 5 Dependency graph after resolving conjunctions and annotating subject-verb-object patterns.

Conclusion

In this paper we introduced the rsyntax R package, as a flexible and relatively accessible tool for using syntactic information in computational text analysis. It offers the core tools for working with syntactic dependencies in a graph format, but represented in a data frame format that communication scientists are more accustomed to working with.

The most interesting application for communication science is the extraction of semantic relations regarding who said or did what to whom, but there are limitations to what can be achieved with a rule-based approach. Language is extremely complex and constantly changing, making it practically impossible to develop rules for every possible way of saying something. A rule-based approach is therefore mainly useful for more specific types of relations (e.g., quotes from politicians, conflicts between known parties) in a not too diverse context (e.g., journalistic texts, parliamentary documents). As the task gets more complex or the language more diverse, machine learning approaches will often be more successful, under the condition that sufficient training data is available. If training data cannot be found or crafted, a rule-based approach might be the only feasible option, but it should be considered that for some tasks this has limited attainable accuracy.

More generally, the achievable accuracy of rsyntax depends on the performance of available dependency parsers. Good quality dependency parsers are not yet available for every language, and even good parser will not perform well if texts have strange of faulty spelling and grammar. Thankfully, open treebank data for many languages are actively being developed for the purpose of training more and better dependency parsers. Moreover, many of these treebanks use the Universal Dependencies annotation scheme, that is designed to be cross-lingually consistent (Nivre et al., 2016). Thus, the accuracy of rules developed in rsyntax is not only likely to increase, but rules developed for one language will to some extent be transferable to other languages. Aside from the core package that offers the tools, we will also provide a separate *cookbook* repository where we and other researchers can share rules.

In evaluating the merit of current techniques for working with syntax, we should not forget that our firmly rooted reliance on the bag-of-words assumption was borne of necessity rather than merit. Even with current limitations, querying dependency trees enables us to use more of the information encoded in texts. With rsyntax we aim to make the use of this information more accessible, so that more researchers might explore how this can advance our field. The stable release version is hosted on the Comprehensive R Archive Network (CRAN), and the development version

is available on GitHub. The main API is stable, but the software will be maintained and currently remains in active development.

References

Barberá, P., Boydstun, A., Linn, S., McMahon, R., & Nagler, J. (2016). Methodological challenges in estimating tone: Application to news coverage of the us economy. In *Meeting of the midwest political science association, chicago, il.*.

Benoit, K., & Matsuo, A. (2017). spacyr: R wrapper to the spacy nlp library [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=spacyr (R package version 0.9.0).

Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant automated content analysis approaches and techniques for digital journalism scholars. *Digital Journalism*, 4 (1), 8–23.

Chen, D., & Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 740–750).

Fundel, K., Küffner, R., & Zimmer, R. (2007). Relex—relation extraction using dependency parse trees. *Bioinformatics*, 23 (3), 365–371.

Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10 (1), 1–309.

Grimmer, J., & Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis*, 21 (3), 267–297. doi: 10.1093/pan/mps028.

Harris, Z. S. (1957). Co-occurrence and transformation in linguistic structure. *Language*, 33 (3), 283–340.

He, L., Lee, K., Lewis, M., & Zettlemoyer, L. (2017). Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: Long papers)* (Vol. 1, pp. 473–483).

Honnibal, M., & Johnson, M. (2015, September). An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1373–1378). Lisbon, Portugal: Association for Computational Linguistics. Retrieved from https://aclweb.org/anthology/D/D15/D15-1162.

Jurafsky, D., & James, H. M. (2020). Speech and language processing. 3rd edn. draft. *Online: https://web.stanford.edu/~jurafsky/slp3*.

Kleinnijenhuis, J., van Hoof, A. M., & van Atteveldt, W. (2019). The combined effects of mass media and social media on political perceptions and preferences. *Journal of Communication*, 69 (6), 650–673.

Koopmans, R., & Statham, P. (1999). Political claims analysis: Integrating protest event and political discourse approaches. *Mobilization: an international quarterly*, 4 (2), 203–221.

Kübler, S., McDonald, R., & Nivre, J. (2009). Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1 (1), 1–127.

Lasswell, H. D., & Leites, N. (1965). *Language of politics: Studies in quantitative semantics* (*2nd ed.*). Cambridge, MA: MIT Press.

Luotolahti, J., Kanerva, J., Pyysalo, S., & Ginter, F. (2015). Sets: Scalable and efficient tree search in dependency graphs. In *Proceedings of the 2015 conference of the north american chapter of the association for computational linguistics: Demonstrations* (pp. 51–55).

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., & McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations* (pp. 55–60).

Nakov, P., Ritter, A., Rosenthal, S., Sebastiani, F., & Stoyanov, V. (2016). Semeval-2016 task 4: Sentiment analysis in twitter. In *Proceedings of the 10th international workshop on semantic evaluation* (*semeval-2016*) (pp. 1–18).

Nivre, J., De Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., ... others (2016). Universal dependencies v1: A multilingual treebank collection. In *Lrec*.

Osgood, C. E., Saporta, S., & Nunnally, J. C. (1956). Evaluative assertion analysis. *Litera*.

Poria, S., Cambria, E., Ku, L.-W., Gui, C., & Gelbukh, A. (2014). A rule-based approach to aspect extraction from product reviews. In *Proceedings of the second workshop on natural language processing for social media* (*socialnlp*) (pp. 28–37).

Roberts, M. E., Stewart, B. M., Tingley, D., Lucas, C., Leder-Luis, J., Gadarian, S. K., ... Rand, D. G. (2014). Structural topic models for open-ended survey responses. *American Journal of Political Science*, 58 (4), 1064–1082.

Schrodt, P. A. (2015). *Comparing methods for generating large scale political event data sets.* presented at the Text as Data meetings, New York University, 16-17 October 2015.

Schuster, S., & Manning, C. D. (2016). Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the tenth international conference on language resources and evaluation* (*lrec '16*) (pp. 2371–2378).

Shlain, M., Taub-Tabib, H., Sadde, S., & Goldberg, Y. (2020). Syntactic search by example. *arXiv preprint arXiv:2006.03010*.

Siddharthan, A., & Mandya, A. A. (2014). Hybrid text.simplification using synchronous dependency grammars with hand-written and automatically harvested

rules. In *Proceedings of the 14th conference of the european chapter of the association for computational linguistics* (*eacl 2014*)..

Straka, M., & Straková, J. (2017, August). Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the conll 2017 shared task: Multilingual parsing from raw text to universal dependencies* (pp. 88–99). Vancouver, Canada: Association for Computational Linguistics. Retrieved from http://www.aclweb.org/anthology/K/K17/K17-3009.pdf.

Van Atteveldt, W. (2008). *Semantic network analysis: Techniques for extracting, representing, and querying media content* (*dissertation*). BookSurge.

Van Atteveldt, W., Sheafer, T., Shenhav, S. R., & Fogel-Dror, Y. (2017). Clause analysis: using syntactic information to automatically extract source, subject, and predicate from texts with an application to the 2008–2009 gaza war. *Political Analysis*, 1–16.

Van Cuilenburg, J. J., Kleinnijenhuis, J., & De Ridder, J. A. (1986). A theory of evaluative discourse: Towards a graph theory of journalistic texts. *European Journal of Communication*, 1 (1), 65–96.

Wijffels, J. (2019). udpipe: Tokenization, parts of speech tagging, lemmatization and dependency parsing with the 'udpipe' 'nlp' toolkit [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=udpipe (R package version 0.8.3).

Wueest, B., Clematide, S., Bünzli, A., Laupper, D., & Frey, T. (2011). Electoral campaigns and relation mining: Extracting semantic network data from newspaper articles. *Journal of Information Technology & Politics*, 8 (4), 444–463.

Zhou, J., & Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing* (*volume 1: Long papers*) (Vol. 1, pp. 1127–1137).

About the authors

Kasper Welbers, Department of Communication Science, Vrije Universiteit Amsterdam

Wouter van Atteveldt, Department of Communication Science, Vrije Universiteit Amsterdam

Jan Kleinnijenhuis, Department of Communication Science, Vrije Universiteit Amsterdam