Package 'semgram'

July 23, 2025

Type Package

Title Extracting Semantic Motifs from Textual Data

Version 0.1.0

Description A framework for extracting semantic motifs around entities in textual data. It implements an entity-centered semantic grammar that distinguishes six classes of motifs: actions of an entity, treatments of an entity, agents acting upon an entity, patients acted upon by an entity, characterizations of an entity, and possessions of an entity. Motifs are identified by applying a set of extraction rules to a parsed text object that includes part-of-speech tags and dependency annotations - such as those generated by 'spacyr'. For further reference, see: Stuhler (2022) <doi:10.1177/00491241221099551>.

URL https://github.com/omstuhler/semgram

License GPL-3

Encoding UTF-8

BugReports https://github.com/omstuhler/semgram/issues

RoxygenNote 7.1.2

Depends R (>= 3.5.3)

Imports data.table, rsyntax (>= 0.1.2), stringr

Author Oscar Stuhler [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7391-1743>)

Maintainer Oscar Stuhler <semgram.r@gmail.com>

NeedsCompilation no

Repository CRAN

Date/Publication 2022-05-31 10:30:02 UTC

Contents

`extract_motifs` *Extract semantic motifs from parsed text object*

Description

This function extracts semantic motifs from text. The input is a data.frame representing a parsed text such as those returned by spacyr::spacy_parse(). The output is a list of data.frames containing semantic motifs such as actions or characterizations of textual entities. For a detailed explanation, see Stuhler (2022).

Usage

```
extract_motifs(
  tokens,
  entities = "*",
  motif_classes = c("t", "a", "be", "H", "At", "aP"),
  custom_cols,
  fast = F,
  parse_multi_token_entities = T,
  extract = "lemma",
  markup = F,
  add_sentence = F,
  be_entity = T,
  get_aux_verbs = F,
  aux_verb_markup = T,
  pron_as_ap = F,
  use_appos = T,
  lowercase = F,
  verbose = F
)
```

Arguments

`tokens`	A tokens data.frame with predicted dependencies as generated, for instance, by spacyr::spacy_parse(). Dependencies need to be in ClearNLP style. This tag set is used by all English language models implemented in spaCy. Other languages or dependency grammars are currently not supported.
`entities`	Specifies the core entities around which to extract motifs. This can be a single character string or a vector of character strings. By default, multi-token strings such as "Harry Potter" will be parsed and considered. Note that this parameter is case-sensitive. It defaults to "*" in which case any token is treated as a potential entity.
`motif_classes`	A character vector specifying which motif classes should be considered in the extraction. This can include "t" for treatments, "a" for actions, "be" for characterizations, "H" for possessions, as well as "At" and "aP" for agent-treatment

and action-patient motifs respectively. By default, all motif classes are considered. Note, however, that runtime increases with the number of motif classes considered.

`custom_cols` Generally, the columns in the tokens object should be labeled as follows: "doc_id", "sentence_id", "token_id", "token", "lemma", "pos", "head_token_id" , "dep_rel". If the columns in your tokens object are not labeled according to this scheme, provide the matching column names to custom_cols in the corresponding order.

`fast` If set to true, some of the more specific extraction rules are not applied. This results in fewer extractions but faster run time. Defaults to FALSE.

`parse_multi_token_entities` Should multi-token entities (e.g., "Harry Potter") be considered? Defaults to TRUE. When using multi-token entities, it is crucial that tokens are separated by a space character. Input should match the tokenized version in the tokens object. For instance, hyphens are usually considered a token in tokenization, so that "Claude Levi-Strauss" should be passed to the function as "Claude Levi - Strauss".

`extract` Defines whether extracted motifs are represented in "lemma" or "token" form. Defaults to "lemma" which reduces sparsity and is preferable for most purposes.

`markup` If TRUE, motifs will also be provided as collapsed markup tokens (e.g., "aP_ask_Harry"). Defaults to FALSE.

`add_sentence` If TRUE, the sentence for each motif is added to the extracted motif. Note that this is done by pasting together the tokens of the sentence, so that the representation might differ minimally from the original text. Nonetheless, this can be helpful for validation and for a mode of analyses that switches between distant and close readings of the text. Defaults to FALSE. Note that setting this to TRUE will noticeably increase runtime.

`be_entity` Should things that are linked to an entity via "being" (or one of its lemmas) be considered as characterization motifs? For example, if we are extracting characterizations around the "immigrants" in the sentence "my parents are immigrants", should we extract the characterization motif "be_parent"? Defaults to TRUE.

`get_aux_verbs` Should auxiliary verbs (e.g., can, could, may, must) be considered actions? Defaults to FALSE.

`aux_verb_markup` Should auxiliary verbs with "to" be marked up so that "going" in "going to eat" becomes "going-to". Note that this will not affect cases of the sort "going to the bar." This can be useful for analyses concerning modality. Defaults to TRUE.

`pron_as_ap` Should pronouns be considered agents and patients? Defaults to FALSE.

`use_appos` Should things linked to an entity via an appositional modifier be considered as equivalent to the entity? For example, if we specify our entity to be "Peter" in the sentence "My brother Peter left.", should "brother" be considered equivalent to "Peter"? Only if use_appos = TRUE, we can extract "leaving" as action motif associated with Peter, as the subject associated with "leaving" is "brother". Defaults to TRUE.

`lowercase` Should all tokens and lemmas be lowercased? Defaults to FALSE.

`verbose` Should progress be reported during execution? Defaults to FALSE.

Details

This is the main function for extracting semantic motifs around entities. Extraction is done by applying a set of extraction rules to the parsed text object that includes part-of-speech tags and dependency relations. Details on the scope of these rules, the theoretical reasoning behind them, and the markup used for motifs can be found in Stuhler (2022). For a recent application, see Stuhler (2021). The following is an abbreviated explanation of the motif classes from Stuhler (2022: 22-23).

Action motifs imply that an entity is doing something. The most straightforward example of this is when the entity serves as a nominal subject of a verb ("ENTITY calls." - a_call). There are various syntactic constructions, however, in which a verb is considered an action despite the entity not being its nominal subject. This includes instances in which the entity is the conjunct of a nominal subject ("John and ENTITY called." - a_call), there are multiple verbs ("ENTITY calls and asks." - a_call, a_ask), the entity serves as an appositional modifier of a nominal subject (My friend ENTITY called. - a_call), and passive constructions ("John was called by ENTITY." - a_call). All actions are either lexical verbs or, if explicitly specified, auxiliary verbs.

Patient motifs are things that the entity of interest acts towards. They are usually objects of transitive verbs that were identified as an entity's action. These objects can be in accusative case ("ENTITY asks John." - aP_ask_John) or in dative case if the verb is ditransitive ("ENTITY asks John a question." - aP_ask_John, aP_ask_question). Any action motif can lead to multiple Patient motifs – as any transitive verb can have multiple conjunct objects ("ENTITY calls John, Jane, and Steve." - aP_call_John, aP_call_Jane, aP_call_Steve). Beyond objects, nominal passive subjects are also considered patients ("John is asked by ENTITY." - aP_ask_John).

Treatment motifs imply that something is done to an entity of interest. This is the case when the entity is the object of a transitive verb. The relationship between treatments and the entity is analogous to that of actions and patients. The entity can function as accusative ("John calls ENTITY" - t_call) or dative ("John gives ENTITY an apple." - t_give) object, as nominal passive subject ("ENTITY was called." - t_call), or as conjunct of any of these ("John calls Peter and ENTITY" - t_call).

Agent motifs are things that act towards the entity of interest via a treatment motif. In most cases, agents are the nominal subject of a verb that has been identified as a treatment motif ("John calls ENTITY." - t_call). However, agents need not take that position and can be conjuncts ("Peter and John ask ENTITY." - At_Peter_ask, At_John_ask) or appositional modifiers ("My friend John asked your brother ENTITY." - At_friend_ask, At_John_ask) of the nominal subject. Generally, the relationship between agents and treatments is analogous to that of the entity and actions, so that the transitive verb may take different positions ("John came and asked ENTITY." - At_John_ask; "John wants to ask ENTITY." At_John_ask), and passive constructions in which the entity serves as nominal passive subject ("ENTITY is asked by John." - At_John_ask) are considered.

Beyond these process motifs, there are two classes of stasis motifs. Characterizations are characteristics ascribed to the entity of interest. There are several ways in which this can happen. The most common one is via a copular verb, that has either an adjectival ("ENTITY is kind." - be_kind; "ENTITY looks sad." - be_sad; "ENTITY is kind and honest." - be_kind, be_honest) or nominal ("ENTITY is the winner." - be_winner; "ENTITY hopes to remain president." - be_president) attribute dependent. However, adjectives can also be direct dependents of the entity ("John bought a cheap, new ENTITY." - be_cheap, be_new) to be considered characterizations. Furthermore, nominal subjects of copular verbs with the entity as attribute dependent ("The winner was ENTITY." - be_winner) and heads with the entity as appositional modifier ("My brother ENTITY won." - be_brother) are considered characterizations.

Possessions are things that the entity of interest is said to possess. The rule set accounts for three ways in which this can be expressed. First, when the entity serves as a possession modifier to a noun, said noun and its conjunct dependents are considered possessions ("ENTITY's spouse, friends, and parents were shocked." - H_spouse, H_friend, H_parent). Second, constructions where the entity serves as object dependent of the preposition "of" can lead to possessions ("The breaks and wheels of the ENTITY were old." - H_breaks, H_wheels). Third, if the entity serves as nominal subject of "have" or one of its inflections, its direct object and nominal conjunctions thereof are considered possessions ("ENTITY has friends and enemies." - H_friend, H_enemy). Note that "have" is a transitive verb, but within the grammar, it is not considered an action, and consequently its objects aren't considered patients.

Value

A list with six dataframes, one for each motif class. List elements of motif classes not specified in the motif_classes parameter will be empty.

References

Stuhler, O. (2022) "Who Does What To Whom? Making Text Parsers Work for Sociological Inquiry." Sociological Methods and Research. <doi: 10.1177/00491241221099551>.

Stuhler, O. (2021) "What's in a category? A new approach to Discourse Role Analysis." Poetics 88. <doi:10.1016/j.poetic.2021.101568>.

Examples

```
# Given data.frame with parsed sentence - as can be generated with spacyr::spacy_parse().
tokens_df = data.frame(doc_id = rep("text1", 4),
                       sentence_id = rep(1, 4),
                       token_id = 1:4,
                       token = c("Emil", "chased", "the", "thief"),
                       lemma = c("Emil", "chase", "the", "thief"),
                       pos = c("PROPN", "VERB", "DET", "NOUN"),
                       head_token_id = c(2,2,4,2),
                       dep_rel = c("nsubj", "ROOT", "det", "dobj")
                       )

# Extract motifs around specific entities, here "Emil"
extract_motifs(tokens = tokens_df, entities = c("Emil"))

# Extract all possible motifs
extract_motifs(tokens = tokens_df, entities = "*")
```

---

semgram *semgram: R package for extracting semantic motifs from text*

Description

semgram extracts semantic motifs from textual data. It builds on an entity-centered semantic grammar that distinguishes six classes of motifs: actions of an entity, treatments of an entity, agents acting upon an entity, patients acted upon by an entity, characterizations of an entity, and possessions of an entity.

Index