TRUNAJOD

Release 0.1.1

Diego Palma

Apr 21, 2021

CONTENTS:

TRUNAJOD is a Python library for text complexity analysis build on the high-performance spaCy library. With all the basic NLP capabilities provided by spaCy (dependency parsing, POS tagging, tokenizing), TRUNAJOD focuses on extracting measurements from texts that might be interesting for different applications and use cases. While most of the indices could be computed for different languages, currently we mostly support Spanish. We are happy if you contribute with indices implemented for your language!

CHAPTER

ONE

FEATURES

- Utilities for text processing such as lemmatization, POS checkings.
- Semantic measurements from text such as average coherence between sentences and average synonym overlap.
- Giveness measurements such as pronoun density and pronoun noun ratio.
- Built-in emotion lexicon to compute emotion calculations based on words in the text.
- Lexico-semantic norm dataset to compute lexico-semantic variables from text.
- Type token ratio (TTR) based metrics, and tunnable TTR metrics.
- A built-in syllabizer (currently only for spanish).
- Discourse markers based measurements to obtain measures of connectivity inside the text.
- Plenty of surface proxies of text readability that can be computed directly from text.
- Measurements of parse tree similarity as an approximation to syntactic complexity.
- Parse tree correction to add periphrasis and heuristics for clause count, all based on linguistics experience.
- Entity Grid and entity graphs model implementation as a measure of coherence.
- An easy to use and user-friendly API.

CHAPTER

TWO

INSTALLATION

`TRUNAJOD` can be installed by running `pip install trunajod`. It requires Python 3.6.2+ to run.

CHAPTER

THREE

GETTING STARTED

Using this package has some other pre-requisites. It assumes that you already have your model set up on spacy. If not, please first install or download a model (for Spanish users, a spanish model). Then you can get started with the following code snippet.

You can download pre-build `TRUNAJOD` models from the repo, under the `models` directory.

Below is a small snippet of code that can help you in getting started with this lib. Don´t forget to take a look at the documentation.

The example below assumes you have the `es_core_news_sm` spaCy Spanish model installed. You can install the model running: `python -m spacy download es_core_news_sm`. For other models, please check spaCy docs.

```
from TRUNAJOD import surface_proxies
from TRUNAJOD.entity_grid import EntityGrid
from TRUNAJOD.lexico_semantic_norms import LexicoSemanticNorm
import pickle
import spacy
import tarfile


class ModelLoader(object):
    """Class to load model."""
    def __init__(self, model_file):
        tar = tarfile.open(model_file, "r:gz")
        self.crea_frequency = {}
        self.infinitive_map = {}
        self.lemmatizer = {}
        self.spanish_lexicosemantic_norms = {}
        self.stopwords = {}
        self.wordnet_noun_synsets = {}
        self.wordnet_verb_synsets = {}

        for member in tar.getmembers():
            f = tar.extractfile(member)
            if "crea_frequency" in member.name:
                self.crea_frequency = pickle.loads(f.read())
            if "infinitive_map" in member.name:
                self.infinitive_map = pickle.loads(f.read())
            if "lemmatizer" in member.name:
                self.lemmatizer = pickle.loads(f.read())
            if "spanish_lexicosemantic_norms" in member.name:
                self.spanish_lexicosemantic_norms = pickle.loads(f.read())
            if "stopwords" in member.name:
                self.stopwords = pickle.loads(f.read())
```

(continues on next page)

(continued from previous page)

```
            if "wordnet_noun_synsets" in member.name:
                self.wordnet_noun_synsets = pickle.loads(f.read())
            if "wordnet_verb_synsets" in member.name:
                self.wordnet_verb_synsets = pickle.loads(f.read())


# Load TRUNAJOD models
model = ModelLoader("trunajod_models_v0.1.tar.gz")

# Load spaCy model
nlp = spacy.load("es_core_news_sm", disable=["ner", "textcat"])

example_text = (
    "El espectáculo del cielo nocturno cautiva la mirada y suscita preguntas"
    "sobre el universo, su origen y su funcionamiento. No es sorprendente que "
    "todas las civilizaciones y culturas hayan formado sus propias "
    "cosmologías. Unas relatan, por ejemplo, que el universo ha"
    "sido siempre tal como es, con ciclos que inmutablemente se repiten; "
    "otras explican que este universo ha tenido un principio, "
    "que ha aparecido por obra creadora de una divinidad."
)

doc = nlp(example_text)

# Lexico-semantic norms
lexico_semantic_norms = LexicoSemanticNorm(
    doc,
    model.spanish_lexicosemantic_norms,
    model.lemmatizer
)

# Frequency index
freq_index = surface_proxies.frequency_index(doc, model.crea_frequency)

# Clause count (heurístically)
clause_count = surface_proxies.clause_count(doc, model.infinitive_map)

# Compute Entity Grid
egrid = EntityGrid(doc)

print("Concreteness: {}".format(lexico_semantic_norms.get_concreteness()))
print("Frequency Index: {}".format(freq_index))
print("Clause count: {}".format(clause_count))
print("Entity grid:")
print(egrid.get_egrid())
```

This should output:

```
Concreteness: 1.95
Frequency Index: -0.7684649336888104
Clause count: 10
Entity grid:
{'ESPECTÁCULO': ['S', '-', '-'], 'CIELO': ['X', '-', '-'], 'MIRADA': ['O', '-', '-'],
↪'UNIVERSO': ['O', '-', 'S'], 'ORIGEN': ['X', '-', '-'], 'FUNCIONAMIENTO': ['X', '-',
↪ '-'], 'CIVILIZACIONES': ['-', 'S', '-'], 'CULTURAS': ['-', 'X', '-'], 'COSMOLOGÍAS
↪': ['-', 'O', '-'], 'EJEMPLO': ['-', '-', 'X'], 'TAL': ['-', '-', 'X'], 'CICLOS': [
↪'-', '-', 'X'], 'QUE': ['-', '-', 'S'], 'SE': ['-', '-', 'O'], 'OTRAS': ['-', '-',
↪'S'], 'PRINCIPIO': ['-', '-', 'O'], 'OBRA': ['-', '-', 'X'], 'DIVINIDAD': ['-', '-',
↪ 'X']}
```

(continues on next page)

(continued from previous page)

CHAPTER

FOUR

A REAL WORLD EXAMPLE

TRUNAJOD lib was used to make TRUNAJOD web app, which is an application to assess text complexity and to check the adquacy of a text to a particular school level. To achieve this, several TRUNAJOD indices were analyzed for multiple Chilean school system texts (from textbooks), and latent features were created. Here is a snippet:

```
"""Example of TRUNAJOD usage."""
import glob

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import spacy
import textract  # To read .docx files
import TRUNAJOD.givenness
import TRUNAJOD.ttr
from TRUNAJOD import surface_proxies
from TRUNAJOD.syllabizer import Syllabizer

plt.rcParams["figure.figsize"] = (11, 4)
plt.rcParams["figure.dpi"] = 200


nlp = spacy.load("es_core_news_sm", disable=["ner", "textcat"])

features = {
    "lexical_diversity_mltd": [],
    "lexical_density": [],
    "pos_dissimilarity": [],
    "connection_words_ratio": [],
    "grade": [],
}
for filename in glob.glob("corpus/*/*.docx"):
    text = textract.process(filename).decode("utf8")
    doc = nlp(text)
    features["lexical_diversity_mltd"].append(
        TRUNAJOD.ttr.lexical_diversity_mtld(doc)
    )
    features["lexical_density"].append(surface_proxies.lexical_density(doc))
    features["pos_dissimilarity"].append(
        surface_proxies.pos_dissimilarity(doc)
    )
    features["connection_words_ratio"].append(
        surface_proxies.connection_words_ratio(doc)
    )
```

(continues on next page)

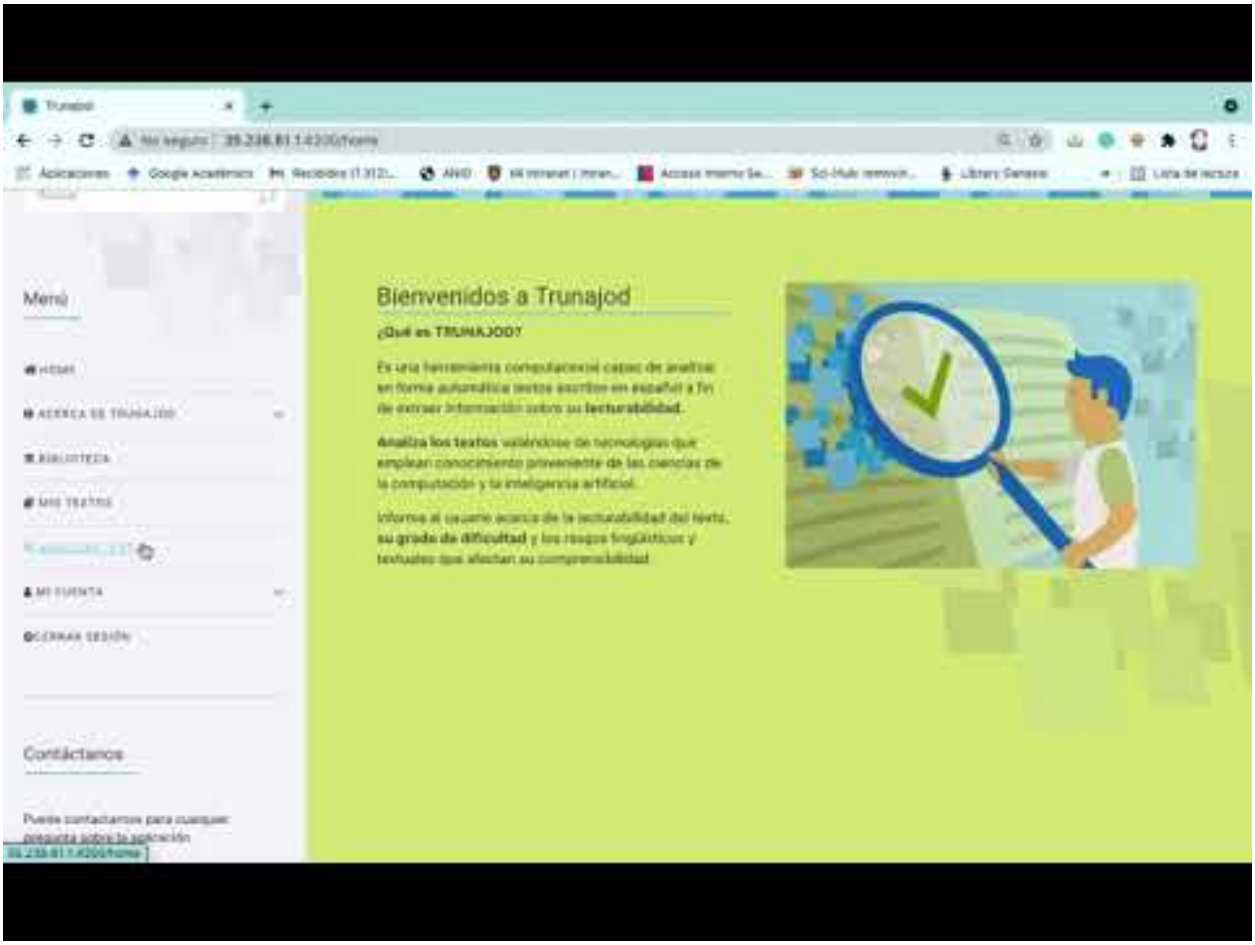
(continued from previous page)

```
    # In our case corpus was organized as:
    # corpus/5B/5_2_55.docx where the folder that
    # contained the doc, contained the school level, in
    # this example 5th grade
    features["grade"].append(filename.split("/")[1][0])

df = pd.DataFrame(features)


fig, axes = plt.subplots(2, 2)

sns.boxplot(x="grade", y="lexical_diversity_mltd", data=df, ax=axes[0, 0])
sns.boxplot(x="grade", y="lexical_density", data=df, ax=axes[0, 1])
sns.boxplot(x="grade", y="pos_dissimilarity", data=df, ax=axes[1, 0])
sns.boxplot(x="grade", y="connection_words_ratio", data=df, ax=axes[1, 1])
```

Which yields:

4.1 *TRUNAJOD* web app example

TRUNAJOD web app backend was built using TRUNAJOD lib. A demo video is shown below (it is in Spanish):



CHAPTER

FIVE

CONTRIBUTING TO *TRUNAJOD*

Bug reports and fixes are always welcome! Feel free to file issues, or ask for a feature request. We use `Github` issue tracker for this. If you'd like to contribute, feel free to submit a pull request. For more questions you can contact me at `dipalma (at) udec (dot) cl`.

More details can be found in CONTRIBUTING.

CHAPTER

SIX

REFERENCES

If you find anything of this useful, feel free to cite the following papers, from which a lot of this python library was made for (I am also in the process of submitting this lib to an open software journal):

1. Palma, D., & Atkinson, J. (2018). Coherence-based automatic essay assessment. IEEE Intelligent Systems, 33(5), 26-36.
2. Palma, D., Soto, C., Veliz, M., Riffo, B., & Gutiérrez, A. (2019, August). A Data-Driven Methodology to Assess Text Complexity Based on Syntactic and Semantic Measurements. In International Conference on Human Interaction and Emerging Technologies (pp. 509-515). Springer, Cham.

```
@article{palma2018coherence,
  title={Coherence-based automatic essay assessment},
  author={Palma, Diego and Atkinson, John},
  journal={IEEE Intelligent Systems},
  volume={33},
  number={5},
  pages={26--36},
  year={2018},
  publisher={IEEE}
}

@inproceedings{palma2019data,
  title={A Data-Driven Methodology to Assess Text Complexity Based on Syntactic and␣
↪Semantic Measurements},
  author={Palma, Diego and Soto, Christian and Veliz, M{\'o}nica and Riffo, Bernardo␣
↪and Guti{\'e}rrez, Antonio},
  booktitle={International Conference on Human Interaction and Emerging Technologies},
  pages={509--515},
  year={2019},
  organization={Springer}
}
```

6.1 API Reference

6.1.1 Discourse Markers

TRUNAJOD discourse markers module.

This discourse markers module is based on the lexicon described in [[iAMC05]], and at the point of writing they are under https://cs.famaf.unc.edu.ar/~laura/shallowdisc4summ/discmar/. TRUNAJOD comes with this lexicon bundled, and this module has the following set ‘constants’:

Type of discourse marker	Discourse marker
cause	dado que, porque, debido a, gracias a, por si, por, por eso, en conclusión, así que, como consecuencia, para, para que, por estar razón, por tanto, en efecto
context	teniendo en cuenta, después, antes, originalmente, a condición de, durante, mientras, a no ser que, cuando, donde, de acuerdo con, lejos de, tan pronto como, por el momento, entre, hacia, hasta, mediante, según, en cualquier caso, entonces, respecto a, en ese caso, si, siempre que, sin duda, a la vez
equality	en resumen, concretamente, en esencia, en comparación, en otras palabras, en particular, es decir, por ejemplo, precisamente, tal como, por último, por un lado, por otro lado, a propósito, no sólo, sino también, en dos palabras, además, también, aparte, aún es más, incluso, especialmente, sobretodo
highly polysemic	como, desde, sobre, antes que nada, para empezar
Revision	a pesar de, aunque, excepto, pese a, no obstante, sin embargo, en realidad, de hecho, al contrario, el hecho es que, es cierto que, pero', con todo, ahora bien, de todos modos
Vague meaning closed class words	y, e, ni, o, u, que, con, sin, contra, en, a,

The following module constants are defined as sets `CAUSE_DISCOURSE_MARKERS`, `CONTEXT_DISCOURSE_MARKERS`, `EQUALITY_DISCOURSE_MARKERS`, `HIGHLY_POLYSEMIC_DISCOURSE_MARKERS`, `REVISION_DISCOURSE_MARKERS`, `VAGUE_MEANING_CLOSED_CLASS_WORDS`.

`TRUNAJOD.discourse_markers.`**`find_matches`**(*text: str, list: List[str]*) → int

Return matches of words in list in a target text.

Given a text and a list of possible matches (in this module, discourse markers list), returns the number of matches found in text. This ignores case.

Hint: For non-Spanish users You could use this function with your custom list of discourse markers in case you need to compute this metric. In that case, the way to call the funcion would be: `find_matches(YOUR_TEXT, ["dm1", "dm2", etc])`

Parameters

- **`text`** (*`string`*) – Text to be processed
- **`list`** (*`Python list of strings`*) – list of discourse markers

Returns Number of ocurrences

Return type int

`TRUNAJOD.discourse_markers.`**`get_cause_dm_count`**(*text: spacy.tokens.doc.Doc*) → float

Count discourse markers associated with cause.

Parameters **`text`** (*`Spacy Doc`*) – The text to be analized

Returns Average of revision cause markers over sentences

Return type float

TRUNAJOD.discourse_markers.**get_closed_class_vague_meaning_count**(*text: spacy.tokens.doc.Doc*) → float

Count words that have vague meaning.

Parameters **text** (*Spacy Doc*) – The text to be analized

Returns Average of vague meaning words over sentences

Return type float

TRUNAJOD.discourse_markers.**get_context_dm_count**(*text: spacy.tokens.doc.Doc*) → float

Count discourse markers associated with context.

Parameters **text** (*Spacy Doc*) – The text to be analized

Returns Average of context discourse markers over sentences

Return type float

TRUNAJOD.discourse_markers.**get_equality_dm_count**(*text: spacy.tokens.doc.Doc*) → float

Count discourse markers associated with equality.

Parameters **text** (*Spacy Doc*) – The text to be analized

Returns Average of equality discourse markers over sentences

Return type float

TRUNAJOD.discourse_markers.**get_overall_markers**(*text: spacy.tokens.doc.Doc*) → float

Count all types of discourse markers.

Parameters **text** (*Spacy Doc*) – The text to be analized

Returns Average discourse markers over sentences

Return type float

TRUNAJOD.discourse_markers.**get_polysemic_dm_count**(*text: spacy.tokens.doc.Doc*) → float

Count discourse markers that are highly polysemic.

Parameters **text** (*Spacy Doc*) – The text to be analized

Returns Average of highly polysemic discourse markers over sentences

Return type float

TRUNAJOD.discourse_markers.**get_revision_dm_count**(*text: spacy.tokens.doc.Doc*) → float

Count discourse markers associated with revisions.

Parameters **text** (*Spacy Doc*) – The text to be analized

Returns Average of revision discourse markers over sentences

Return type float

6.1.2 Emotions

TRUNAJOD emotions module.

These measurements are based on the lexicon from [[RSG14]]. TRUNAJOD already comes bundled with this lexicon, in the following module constant `TRUNAJOD.spanish_emotion_lexicon.SPANISH_EMOTION_LEXICON`.

> **Caution:** Keep in mind this module is still under development. Moreover only Spanish language is supported by this module as we are directly feeding the class with the lexicon.

The following emotions are captured: alegría, enojo, miedo, repulsión, sorpresa, tristeza.

class `TRUNAJOD.emotions.`**Emotions**(*doc: spacy.tokens.doc.Doc*, *lemmatizer: Optional[Dict[str, str]] = None*)

Compute emotions from SPANISH EMOTION LEXICON [[RSG14]].

Computes emotions based on Probability Factor of Affective use (PFA) and averaging over the total analized words from the text (that are found in the lexicon).

Ideas:

- How about not using PFA?
- Averaging over emotions and not the total count?

get_alegria() → float
Get alegria.

Returns Average alegria over number of tokens

Return type float

get_enojo() → float
Get enojo.

Returns Average enojo over number of tokens

Return type float

get_miedo() → float
Get miedo.

Returns Average miedo over number of tokens

Return type float

get_repulsion() → float
Get repulsion.

Returns Average repulsion over number of tokens

Return type float

get_sorpresa() → float
Get sorpresa.

Returns Average sorpresa over number of tokens

Return type float

get_tristeza() → float
Get tristeza.

Returns Average tristeza over number of tokens

Return type float

6.1.3 Entity Grids

Entity grid module for TRUNAJOD.

In this module, entity grid based features are implemented. On one side, an entity grid [[BL08]] implementation is provided. We also provide an implementation of the entity graph coherence modeling [[GS13]].

Danger: These set of features or measurements Really depends on the dependency parsing accuracy, which relies on the CORPUS the dependency parsed was trained. There is no guarantee that this will work with all types of texts. On the other hand, the implementation is simple and we do not do any coreference resolution for noun-phrases and just rely on simple heuristics.

It is also worth noting, that we consider an entity grid of two-sentence sequence and the API currently does not provide any hyper-parameter tunning to change this.

class TRUNAJOD.entity_grid.**EntityGrid**(*doc*, *model_name='spacy'*)

Entity grid class.

Class Entity Grid, creates an entity grid from a doc, which is output of applying spacy.nlp(text) to a text. Thus, this class depends on spacy module. It only supports 2-transitions entity grid.

get_egrid() → dict

Return obtained entity grid (for debugging purposes).

Returns entity grid represented as a dict

Return type dict

get_nn_transitions() → float

Get – transitions.

Returns Ratio of transitions

Return type float

get_no_transitions() → float

Get -O transitions.

Returns Ratio of transitions

Return type float

get_ns_transitions() → float

Get -S transitions.

Returns Ratio of transitions

Return type float

get_nx_transitions() → float

Get -X transitions.

Returns Ratio of transitions

Return type float

get_on_transitions() → float

Get O- transitions.

Returns Ratio of transitions

Return type float

get_oo_transitions() → float
Get OO transitions.

Returns Ratio of transitions

Return type float

get_os_transitions() → float
Get OS transitions.

Returns Ratio of transitions

Return type float

get_ox_transitions() → float
Get OX transitions.

Returns Ratio of transitions

Return type float

get_sentence_count() → int
Return sentence count obtained while processing.

Returns Number of sentences

Return type int

get_sn_transitions() → float
Get S- transitions.

Returns Ratio of transitions

Return type float

get_so_transitions() → float
Get SO transitions.

Returns Ratio of transitions

Return type float

get_ss_transitions() → float
Get SS transitions.

Returns Ratio of transitions

Return type float

get_sx_transitions() → float
Get SX transitions.

Returns Ratio of transitions

Return type float

get_xn_transitions() → float
Get X- transitions.

Returns Ratio of transitions

Return type float

get_xo_transitions() → float
Get XO transitions.

Returns Ratio of transitions

Return type float

get_xs_transitions() → float
Get XS transitions.

Returns Ratio of transitions

Return type float

get_xx_transitions() → float
Get XX transitions.

Returns Ratio of transitions

Return type float

TRUNAJOD.entity_grid.**dependency_mapping**(*dep: str*) → str
Map dependency tag to entity grid tag.

We consider the notation provided in [[BL08]]:

EGrid Tag	Dependency Tag
S	nsub, csubj, csubjpass, dsubjpass
O	iobj, obj, pobj, dobj
X	For any other dependency tag

Parameters **dep** (*string*) – Dependency tag

Returns EGrid tag

Return type string

TRUNAJOD.entity_grid.**get_local_coherence**(*egrid: TRUNAJOD.entity_grid.EntityGrid*) → [<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>]
Get local coherence from entity grid.

This method gets the coherence value using all the approaches described in [[GS13]]. This include:

- local_coherence_PU
- local_coherence_PW
- local_coherence_PACC
- local_coherence_PU_dist
- local_coherence_PW_dist
- local_coherence_PACC_dist

Parameters **egrid** (EntityGrid) – An EntityGrid object.

Returns Local coherence based on different heuristics

Return type tuple of floats

TRUNAJOD.entity_grid.**weighting_syntactic_role**(*entity_role: str*) → int
Return weight given an entity grammatical role.

Weighting scheme for syntactic role of an entity. This uses the heuristic from [[GS13]], which is:

EGrid Tag	Weight
S	3
O	2
X	1
dash	0

Parameters **entity_role** (*string*) – Entity grammatical role (S, O, X, -)

Returns Role weight

Return type int

6.1.4 Givenness

Givenness module.

Giveness is defined as the amount of given information a text exposes over successive constituents [[HDM+05]]. Givenness is can be used as a proxy of text complexity.

TRUNAJOD.givenness.**pronoun_density**(*doc: spacy.tokens.doc.Doc*) → float
Compute pronoun density.

This is a measurement of text complexity, in the sense that a text with a higher pronoun density will be more difficult to read than a text with lower pronoun density (due to inferences needed). The way this is computed is taking the ratio between third person pronouns and total words in the text.

Parameters **doc** (*Spacy Doc*) – Document to be processed.

Returns Pronoun density

Return type float

TRUNAJOD.givenness.**pronoun_noun_ratio**(*doc: spacy.tokens.doc.Doc*) → float
Compute Pronoun Noun ratio.

This is an approximation of text complexity/readability, since pronouns are co-references to a proper noun or a noun. This is computed as the taking the ratio between third person pronouns and total nouns.

Parameters **doc** (*Spacy doc*) – Text to be processed

Returns pronoun-noun ratio

Return type float

6.1.5 Lexico-Semantic Norms

TRUNAJOD lexico semantic norms module.

Lexico-Semantic norms do also require external knowledge to be computed. We compute the following lexico-semantic variables:

- Arousal
- Concreteness
- Context Availability
- Familiarity
- Imageability
- Valence

We provide two downloadable models of these variables, which come from [[DPSebastianGalles+13]] and [[GFerreF16]].

class TRUNAJOD.lexico_semantic_norms.**LexicoSemanticNorm**(*doc*, *lexico_semantic_norm_dict*, *lemmatizer=None*)

Create a lexico semantic norm calculator for text.

This requires a lexico semantic norm dict, with key-value pairs specified as `word -> {"arousal", "concreteness", "context_availability", "familiarity", "imageability", "valence"}`. Average over number of tokens will be computed. The values are obtained from [[GFerreF16]].

get_arousal()
Get arousal.

Returns Average arousal.

Return type float

get_concreteness()
Get concreteness.

Returns Average concreteness.

Return type float

get_context_availability()
Get context_availability.

Returns Average context_availability.

Return type float

get_familiarity()
Get familiarity.

Returns Average familiarity.

Return type float

get_imageability()
Get imageability.

Returns Average imageability.

Return type float

get_valence()
Get valence.

Returns Average valence.

Return type float

TRUNAJOD.lexico_semantic_norms.**get_conc_imag_familiarity**(*doc*)
Get lexico-semantic variables.

Computes three lexico-semantic variables: Concreteness, Imageability and Familiarity. The values are obtained from the EsPal dictionary (Spanish) and average of each metric is computed over sentences. To get each metric, the best practice is using *LSNorm Enum* defined in *lexicosemantic_norms_espal* module. The enums are *CONCRETENESS*, *IMAGEABILITY* and *FAMILIARITY*. This implementation uses values of the lexico-semantic norms from [[DPSebastianGalles+13]].

Parameters **doc** (*Spacy Doc*) – Tokenized text

Returns Concreteness imageability and familiarity averaged over sentences

Return type List of float

6.1.6 Semantic Measures

Semantic measures TRUNAJOD methods.

The dimensions defined in this module, require external knowledge, for example synonym overlap measurement requires knowledge from a word onthology, and semantic measurements require word vectors (word embeddings) obtained from CORPUS semantics.

TRUNAJOD.semantic_measures.**avg_w2v_semantic_similarity**(*docs*, *N*)
Compute average semantic similarity between adjacent sentences.

This is using word2vec [[MCC+13]] model based on SPACY implementation. The semantic similarity is based on [[FKL98]] approach to compute text coherence.

Parameters

- **docs** (*Doc Generator*) – Docs generator provided by SPACY API
- **N** (*int*) – Number of sentences

Returns Average sentence similarity (cosine)

Return type float

TRUNAJOD.semantic_measures.**get_synsets**(*lemma*, *synset_dict*)
Return synonym set given a word lemma.

The function requires that the synset_dict is passed into it. In our case we provide downloadable models from MCR (Multilingual-Central-Repository). [[GALR12]]. If the lemma is not found in the synset_dict, then this function returns a set with the lemma in it.

Parameters

- **lemma** (*string*) – Lemma to be look-up into the synset
- **synset_dict** (*Python dict*) – key-value pairs, lemma to synset

Returns The set of synonyms of a given lemma

Return type Python set of strings

TRUNAJOD.semantic_measures.**overlap**(*lemma_list_group*, *synset_dict*)
Compute average overlap in a text.

Computes semantic synset overlap (synonyms), based on a lemma list group and a dictionary containing synsets. Note that the computations are carried out dividing by number of text segments considered; matches TAACO implementation. For more details about this measurement, refer to [[CKM16]]

Parameters

- **lemma_list_group** (*List of List of strings*) – List of tokenized and lemmatized sentences
- **synset_dict** (*Python dict*) – key-value pairs for lemma-synonyms

Returns Average overlap between sentences

Return type float

6.1.7 Surface Proxies

Surface proxies of TRUNAJOD.

These surface proxies are measurements from text that consists on shallow measures (proxies) that approximate to intrinsic properties of the text such as cohesion, coherence, complexity. Examples of these measurements include but are not limited to: Number of sentences, number of syllables, etc.

TRUNAJOD.surface_proxies.**add_periphrasis**(*doc*, *periphrasis_type*, *periphrasis_list*)
Add periphrasis to SPACY tags.

One of the drawbacks that spaCy has, is that it does not address properly periphrasis of texts (in our case Spanish text). This function adds periphrasis to the text in order to improve further analysis such as clause segmentation, and clause count. This is used by *TRUNAJOD.surface_proxies.fix_parse_tree()*.

Parameters

- **doc** (*Spacy Doc*) – Tokenized text
- **type** (*string*) – Periphrasis type
- **periphrasis_list** (*List of strings*) – List of periphrasis

Returns Corrected doc

Return type Spacy Doc

TRUNAJOD.surface_proxies.**average_clause_length**(*doc*, *infinitive_map*)
Return average clause length (heuristic).

This measurement is computed as the ratio of # of words / # of clauses. To count clauses we do it heuristically, and you can refer to *TRUNAJOD.surface_proxies.clause_count()* for more details.

Parameters

- **doc** (*Spacy Doc*) – Text to be processed
- **infinitve_map** – Lexicon containing maps from conjugate to infinitive.

Returns Average clause length

Return type float

TRUNAJOD.surface_proxies.**average_sentence_length**(*doc*)
Return average sentence length.

This measurement is computed as the ratio of: # of words / # of sentences.

Parameters **doc** (*Spacy Doc*) – Text to be processed.

Returns average sentence length

Return type float

TRUNAJOD.surface_proxies.**average_word_length**(*doc*)
Return average word length.

Computed as the ratio of: # number of chars / # of words

Parameters **doc** (*Spacy Doc*) – Text to be processed.

Returns Average word length

Return type float

TRUNAJOD.surface_proxies.**char_count**(*doc*)
Return number of chars in a text.

This count does not consider anything that its `Token.pos_` tag is either `PUNCT` or `SPACE`.

Parameters **doc** (*Spacy Doc*) – Text to be processed.

Returns Char count

Return type int

TRUNAJOD.surface_proxies.**clause_count**(*doc*, *infinitive_map*)
Return clause count (heuristic).

This function is decorated by the `TRUNAJOD:surface_proxies.fix_parse_tree()` function, in order to heuristically count clauses.

Parameters

- **doc** (*Spacy Doc*) – Text to be processed.
- **infinitve_map** – Lexicon containing maps from conjugate to infinitive.

Returns Clause count

Return type int

TRUNAJOD.surface_proxies.**connection_words_ratio**(*doc*)
Get ratio of connecting words over total words of text.

This function computes the ratio of connective words over the total number of words. This implementation is only supported in Spanish and we consider the following lemmas: `y`, `o`, `no`, `si`.

Parameters **doc** (*Spacy Doc*) – Tokenized text

Returns Connection word ratio

Return type float

TRUNAJOD.surface_proxies.**first_second_person_count**(*doc*)
Count first|second person tokens.

Parameters **doc** (*Spacy Doc*) – Processed text

Returns First and second person count

Return type int

TRUNAJOD.surface_proxies.**first_second_person_density**(*doc*)
Compute density of first|second person.

Parameters **doc** (*Spacy Doc*) – Processed text

Returns Density 1,2 person

Return type float

TRUNAJOD.surface_proxies.**fix_parse_tree**(*doc*, *infinitive_map*)
Fix SPACY parse tree.

We found that for Spanish texts, spaCy tags do not deal appropiately with periphrasis and other lingüistic cues. This function address this shortcome by modifying the parse tree computed by spaCy adding periphrasis for Gerunds, Infinitive and Past tense verbs.

Parameters

- **doc** (*Spacy Doc*) – Processed text
- **infinitve_map** – Lexicon containing maps from conjugate to infinitive.

Returns Fixed Doc

Return type Spacy Doc

TRUNAJOD.surface_proxies.**frequency_index**(*doc*, *frequency_dict*)
Return frequency index.

The frequency index is defined as the average frequency of the rarest word over sentences. To compute this, we use a dictionary. In the case of this Spanish implementation we could use RAE dictionary CREA.

Parameters **doc** (*Spacy Doc*) – Tokenized text.

Returns Frequency index

Return type float

TRUNAJOD.surface_proxies.**get_word_depth**(*index*, *doc*)
Get word depth in the parse tree given a sentence and token index.

The ROOT of the sentence is considered level 1. This method traverses the parse tree until reaching the ROOT, and counts all levels traversed.

Parameters

- **index** (*int*) – Position of the token in the sentence
- **doc** (*Spacy Doc*) – Tokenized text

Returns Depth of the of the token

Return type int

TRUNAJOD.surface_proxies.**infinitve**(*conjugate*, *infinitive_map*)
Get infinitive form of a conjugated verb.

Given a mapping of conjugate to infinitive, this function computes the infinitive form of a conjugate verb. We provide models available for downloading, so you do not have to worry about the infinitive_map. Regretfully we only provide models for Spanish texts.

Parameters

- **conjugate** (*string*) – Verb to be processed
- **infinitve_map** – Lexicon containing maps from conjugate to infinitive.

Returns Infinitive form of the verb, None if not found

Return type string

TRUNAJOD.surface_proxies.**lexical_density**(*doc*)
Compute lexical density.

The lexical density is defined as the Part of Speech ratio of the following tags: VERB, AUX, ADJ, NOUN, PROPN and ADV over the total number of words.

Parameters **doc** (*Spacy Doc*) – Tokenized text

Returns Lexical density

Return type Float

TRUNAJOD.surface_proxies.**negation_density**(*doc*)
Compute negation density.

This is defined as the ratio between number of occurrences of TRUNAJOD.surface_proxies.NEGATION_WORDS in the text over the total word count.

Parameters **doc** (*Spacy Doc*) – Tokenized text

Returns Negation density

Return type float

TRUNAJOD.surface_proxies.**node_similarity**(*node1*, *node2*, *is_central_node=False*)
Compute node similarity recursively, based on common children POS.

This function is called inside *TRUNAJOD.surface_proxies.syntactic_similarity()* so is an auxiliary function. In the common use case, is unlikely you will need to call this function directly, but we provide it for debugging purposes.

Parameters

- **node1** (*Spacy Token*) – Node of the parse tree.
- **node2** (*Spacy Token*) – Node of the parse tree
- **is_central_node** (*bool, optional*) – Whether is the central node, defaults to False

Returns Total childs in common between node1 and node2.

Return type int

TRUNAJOD.surface_proxies.**noun_count**(*doc*)
Count nouns in the text.

Count all tokens which Part of Speech tag is either NOUN or PROPN.

Parameters **doc** (*Spacy Doc*) – Text to be processed

Returns Noun count

Return type int

TRUNAJOD.surface_proxies.**noun_phrase_density**(*doc*)
Compute NP density.

To compute NP density we do it heuristically. We might improve it in the future by using some NP-chunking strategy. For counting noun phrases, we check that for a node in the parse tree, its head is a Noun. Then, we check if either of the following conditions is met:

- The token is the article del or al
- The token dependency is not cc, case or cop, and the token is not a punctuation and the token is not the ROOT

Then we compute the ratio between # of NP / Noun count.

Parameters **doc** (*Spacy Doc*) – Tokenized text.

Returns NP density

Return type float

TRUNAJOD.surface_proxies.**pos_dissimilarity**(*doc*)

Measure Part of Speech disimilarity over sentences.

The dissimilarity of POS between two sentences is the difference between POS distribution over the total population of POS tags. It is computed as follows:

- For each sentence, PoS tag distribution is computed.
- For each tag in either of the two sentences, we compute the difference in distributions (absolute value)
- This difference is divided by the total population of the sentences

This is done for each pair of sentences (`N - 1` sentences) and the results are averaged (again, over `N - 1`)

Parameters **doc** (*Spacy Doc*) – Processed text

Returns Part of Speech dissimilarity

Return type float

TRUNAJOD.surface_proxies.**pos_distribution**(*doc*)

Get POS distribution from a processed text.

Let us suppose that a given sentence has the following pos tags: `[NOUN, VERB, ADJ, VERB, ADJ]`. The PoS distribution would be

- `NOUN: 1`
- `VERB: 2`
- `ADJ: 2`

This function returns this distrubution as a dict.

Parameters **doc** (*Spacy Doc*) – Processed text

Returns POS distribution as a dict key POS, value Count

Return type dict

TRUNAJOD.surface_proxies.**pos_ratio**(*doc*, *pos_types*)

Compute POS ratio given desired type of ratio.

The `pos_types` might be a regular expression if a composed ratio is needed. An example of usage would be `pos_ratio(doc, "VERB|AUX")`.

Parameters

- **doc** (*Spacy Doc*) – Spacy processed text
- **pos_types** (*string*) – POS to get the ratio

Returns Ratio over number of words

Return type float

TRUNAJOD.surface_proxies.**sentence_count**(*doc*)

Return number of sentences in a text.

Parameters **doc** (*Spacy Doc*) – Text to be processed

Returns Number of sentences in the text

Return type int

TRUNAJOD.surface_proxies.**subordination**(*doc*, *infinitive_map*)
Return subordination, defined as the clause density.

The subordination is defined as the ratio between # of clauses and the # of sentences. To compute number of clauses, a heuristic is used.

Parameters

- **doc** (*Spacy Doc*) – Text to be processed.
- **infinitve_map** – Lexicon containing maps from conjugate to infinitive.

Returns Subordination index

Return type float

TRUNAJOD.surface_proxies.**syllable_count**(*doc*)
Return number of syllables of a text.

Parameters **doc** (*Spacy Doc*) – Text to be processed.

Returns Number of syllables in the text

Return type int

TRUNAJOD.surface_proxies.**syllable_word_ratio**(*doc*)
Return average syllable word ratio.

It is computed as # Syllables / # of words.

Parameters **doc** (*Spacy Doc*) – Text to be processed.

Returns syllable word ratio

Return type float

TRUNAJOD.surface_proxies.**syntactic_similarity**(*doc*)
Compute average syntactic similarity between sentences.

For each pair of sentences, compute the similarity between each pair of nodes, using *TRUNAJOD.surface_proxies.node_similarity()* Then, the result is averaged over the N - 1 pair of sentences.

Parameters **doc** (*Spacy Doc*) – Processed text

Returns Average syntactic similarity over sentences.

Return type float

TRUNAJOD.surface_proxies.**verb_noun_ratio**(*doc*)
Compute Verb/Noun ratio.

Parameters **doc** (*Spacy Doc*) – Processed text

Returns Verb Noun ratio

Return type float

TRUNAJOD.surface_proxies.**word_count**(*doc*)
Return number of words in a text.

Parameters **doc** (*Spacy Doc*) – Text to be processed.

Returns Word count

Return type int

TRUNAJOD.surface_proxies.**words_before_root**(*doc*, *max_depth=4*)

Return average word count of words before root.

For each sentence, word count before root is computed in the case that the root is a verb. Otherwise, the root is considered to be the verb in the highest node in the parse tree.

Parameters **doc** (*Spacy Doc*) – Text to be processed.

Returns Average words before root

Return type float

6.1.8 Syllabizer

Syllabizer module.

This syllabizator is for spanish texts. It is based on http://sramatic.tripod.com/silabas.html

And based on mabodo's implementation: https://github.com/mabodo/sibilizador

- Strong vowels are `a-e-o`
- Weak vowels are `i-u`

The following rules are applied:

Rule	Description
v	The smallest syllabe is formed by one vowel.
V+ - V+	Two vowels are separated if both are strong vowels.
V-V+ and V-V-	Two vowels are not separated if one is strong and the other
CV	is weak nor if both are weak. Most common syllable in Spanish is the one that has a
C-C	consonant and a vowel. Two consonants joined are usually separated.
CC/C = l,r	Two join consonants are maintained joint if the second is an l or r.
CC/ = ch,ll,rr	Two consonants are joined if they represent the sounds ch, ll,rr.
C-CC	If three consonants are joined, the first one is separated from the rest.
CC-C/CsC	In the situation of three joined consonants, the first two are separated from the last one if the one in the middle
CC-CC	is an s. If four consonants are joined, they are halved.

class TRUNAJOD.syllabizer.**CharLine**(*word*)

Auxiliary object to set char types on a word.

A word string is processed and converted into a char sequence, consisting on consonants, vowels that are used to apply rules for syllabizing Spanish words. This is a helper class used by the Syllabizator class and it is unlikely the user will need to explicitly instanitate an object of this class.

static char_type(*char: str*) → str

Get char type (vowel, consonant, etc).

This method checks a `char` type based on syllabization rules. If the `char` is in `STRONG_VOWELS` this returns `'V'`. If the `char` is in `WEAK_VOWELS` it will return `'v'`. If the `char` is an `'x'` or `'s'` it will return `'x'` and `'s'` respectively. Otherwise it will return `'c'` representing a consonant.

Parameters **char** (*string*) – Char from were to get the type

Returns Char type

Return type string

find (*finder: str*) → int
Find string occurrence in the type representation.

Parameters **finder** (*string*) – String to be searched

Returns Position of occurrence of the finder

Return type int

split (*pos: int*, *where: int*) → [~CharLine, ~CharLine]
Split the object into two Charline objects.

Parameters

- **pos** (*int*) – Start position of the split
- **where** (*int*) – End position of the split

Returns Tuple with two charlines split

Return type Tuple (*CharLine*, *CharLine*)

split_by (*finder: str*, *where: int*) → [~CharLine, ~CharLine]
Split charline by *finder* occurrence on *type_char*.

Parameters

- **finder** (*string*) – Type char string
- **where** (*int*) – End position to look for.

Returns Split of two charlines based on match.

Return type Tuple (*CharLine*, *CharLine*)

class TRUNAJOD.syllabizer.**Syllabizer**
Syllabizer class to process syllables from a word.

It has methods that take a word split it into syllables using different rules. This class is mainly used for counting syllables.

static number_of_syllables (*word: str*) → int
Return number of sillables of a word.

Parameters **word** (*string*) – Word to be processed

Returns Syllable count for the word.

Return type int

static split (*chars: TRUNAJOD.syllabizer.CharLine*) → [<class 'TRUNAJOD.syllabizer.CharLine'>]
Split CharLine into syllabes.

Parameters **chars** (CharLine) – Word to be syllabized

Returns Syllabes

Return type List [*CharLine*]

6.1.9 Type Token Ratios

Type Token Ratios module.

Type token ratios (TTR) are a measurement of lexical diversity. They are defined as the ratio of unique tokens divided by the total number of tokens. This measurement is bounded between 0 and 1. If there is no repetition in the text this measurement is 1, and if there is infinite repetition, it will tend to 0. This measurement is not recommended if analyzing texts of different lengths, as when the number of tokens increases, the TTR tends flatten.

TRUNAJOD.ttr.**d_estimate**(*doc: spacy.tokens.doc.Doc*, *min_range: int = 35*, *max_range: int = 50*, *trials: int = 5*) → float

Compute D measurement for lexical diversity.

The measurement is based in [[RM00]]. We pick `n` numbers of tokens, varying `N` from `min_range` up to `max_range`. For each `n` we do the following:

1. Sample `n` tokens without replacement
2. Compute `TTR`
3. Repeat steps 1 and 2 `trials` times
4. Compute the average `TTR`

At this point, we have a set of points `(n, ttr)`. We then fit these observations to the following model:

$$TTR = \frac{D}{N}\left[\sqrt{1+2\frac{N}{D}}-1\right]$$

The fit is done to get an estimation for the `D` parameter, and we use a least squares as the criteria for the fit.

Parameters

- **doc** (*Doc*) – SpaCy doc of the text.
- **min_range** (*int, optional*) – Lower bound for n, defaults to 35
- **max_range** (*int, optional*) – Upper bound for n, defaults to 50
- **trials** (*int, optional*) – Number of trials to estimate TTR, defaults to 5

Raises **ValueError** – If invalid range is provided.

Returns D metric

Return type float

TRUNAJOD.ttr.**lexical_diversity_mtld**(*doc: spacy.tokens.doc.Doc*, *model_name: str = 'spacy'*, *ttr_segment: float = 0.72*) → float

Compute MTLD lexical diversity in a bi-directional fashion.

Parameters

- **doc** (*NLP Doc*) – Processed text
- **model_name** (*str*) – Determines which model is used (spacy or stanza)
- **ttr_segment** (*float*) – Threshold for TTR mean computation

Returns Bi-directional lexical diversity MTLD

Return type float

TRUNAJOD.ttr.**one_side_lexical_diversity_mtld**(*doc: spacy.tokens.doc.Doc*, *model_name: str = 'spacy'*, *ttr_segment: float = 0.72*) → float

Lexical diversity per MTLD.

Parameters

- **doc** (*NLP Doc*) – Tokenized text
- **model_name** (*str*) – Determines which model is used (spacy or stanza)
- **ttr_segment** (*float*) – Threshold for TTR mean computation

Returns MLTD lexical diversity

Return type float

TRUNAJOD.ttr.**type_token_ratio**(*word_list: List[str]*) → float
Return Type Token Ratio of a word list.

Parameters **word_list** (*List of strings*) – List of words

Returns TTR of the word list

Return type float

TRUNAJOD.ttr.**yule_k**(*doc: spacy.tokens.doc.Doc*) → float
Compute Yule's K from a text.

Yule's K is defined as follows [[Yul14]]:

$$K = 10^{-4} \frac{\sum r^2 V_r - N}{N^2}$$

Where *Vr* is the number of tokens ocurring *r* times. This is a measurement of lexical diversity.

Parameters **doc** (*Doc*) – Processed spaCy Doc

Returns Texts' Yule's K

Return type float

6.1.10 Utils

Utility functions for TRUNAJOD library.

class TRUNAJOD.utils.**SupportedModels**
Enum for supported Doc models.

TRUNAJOD.utils.**flatten**(*list_of_lists*)
Flatten a list of list.

This is a utility function that takes a list of lists and flattens it. For example the list `[[1, 2, 3], [4, 5]]` would be converted into `[1, 2, 3, 4, 5]`.

Parameters **list_of_lists** (*Python List of Lists*) – List to be flattened

Returns The list flattened

Return type Python List

TRUNAJOD.utils.**get_sentences_lemmas**(*docs*, *lemma_dict*, *stopwords=[]*)
Get lemmas from sentences.

Get different types of lemma measurements, such as noun lemmas, verb lemmas, content lemmas. It calls *TRUNAJOD.utils.get_token_lemmas()* internally to extract different lemma types for each sentence. This function extract the following lemmas:

- Noun lemmas
- Verb lemmas

- Function lemmas (provided as `stopwords`)
- Content lemmas (anything that is not in `stopwords`)
- Adjective lemmas
- Adverb lemmas
- Proper pronoun lemmas

Parameters

- **docs** (*List of Spacy Doc (Doc.sents)*) – List of sentences to be processed.
- **lemma_dict** (*dict*) – Lemmatizer dictionary
- **stopwords** (*list, optional*) – List of stopwords (function words), defaults to []

Returns List of lemmas from text

Return type List of Lists of str

TRUNAJOD.utils.**get_stopwords**(*filename*)

Read stopword list from file.

Assumes that the list is defined as a newline separated words. It is a utility in case you'd like to provide your own stopwords list. Assumes encoding `utf8`.

Parameters **filename** (*string*) – Name of the file containing stopword list.

Returns List of stopwords

Return type set

TRUNAJOD.utils.**get_token_lemmas**(*doc*, *lemma_dict*, *stopwords=[]*)

Return lemmas from a sentence.

From a sentence, extracts the following lemmas:

- Noun lemmas
- Verb lemmas
- Function lemmas (provided as `stopwords`)
- Content lemmas (anything that is not in `stopwords`)
- Adjective lemmas
- Adverb lemmas
- Proper pronoun lemmas

Parameters

- **doc** (*Spacy Doc*) – Doc containing tokens from text
- **lemma_dict** (*Dict*) – Lemmatizer key-value pairs
- **stopwords** (*set/list, optional*) – list of stopwords, defaults to []

Returns All lemmas for noun, verb, etc.

Return type tuple of lists

TRUNAJOD.utils.**is_adjective**(*pos_tag*)

Return `True` if `pos_tag` is `ADJ`, False otherwise.

Parameters **pos_tag** (*string*) – Part of Speech tag

Returns True if POS is adjective

Return type boolean

TRUNAJOD.utils.**is_adverb**(*pos_tag*)
Return True if pos_tag is ADV, False otherwise.

Parameters **pos_tag** (*string*) – Part of Speech tag

Returns True if POS is adverb

Return type boolean

TRUNAJOD.utils.**is_noun**(*pos_tag*)
Return True if pos_tag is NOUN or PROPN, False otherwise.

Parameters **pos_tag** (*string*) – Part of Speech tag

Returns True if POS is noun or proper noun

Return type boolean

TRUNAJOD.utils.**is_pronoun**(*pos_tag*)
Return True if pos_tag is PRON, False otherwise.

Parameters **pos_tag** (*string*) – Part of Speech tag

Returns True if POS is pronoun

Return type boolean

TRUNAJOD.utils.**is_stopword**(*word*, *stopwords*)
Return True if word is in stopwords, False otherwise.

Parameters

- **word** (*string*) – Word to be checked
- **stopwords** (*List of strings*) – stopword list

Returns True if word in stopwords

Return type boolean

TRUNAJOD.utils.**is_verb**(*pos_tag*)
Return True if pos_tag is VERB, False otherwise.

Parameters **pos_tag** (*string*) – Part of Speech tag

Returns True if POS is verb

Return type boolean

TRUNAJOD.utils.**is_word**(*pos_tag*)
Return True if pos_tag is not punctuation, False otherwise.

This method checks that the pos_tag does not belong to the following set: {'PUNCT', 'SYM', 'SPACE'}.

Parameters **pos_tag** (*string*) – Part of Speech tag

Returns True if POS is a word

Return type boolean

TRUNAJOD.utils.**lemmatize**(*lemma_dict*, *word*)
Lemmatize a word.

Lemmatizes a word using a lemmatizer which is represented as a dict that has (word, lemma) as (key, value) pair. An example of a lemma list can be found in https://github.com/michmech/lemmatization-lists.

If the word is not found in the dictionary, the lemma returned will be the word.

Parameters

- **lemma_dict** (*Python dict*) – A dict (word, lemma)
- **word** (*string*) – The word to be lemmatized

Returns Lemmatized word

Return type string

TRUNAJOD.utils.**process_text**(*text*, *sent_tokenize*)
Process text by tokenizing sentences given a tokenizer.

Parameters

- **text** (*string*) – Text to be processed
- **sent_tokenize** (*Python callable that returns list of strings*) – Tokenizer

Returns Tokenized sentences

Return type List of strings

TRUNAJOD.utils.**read_text**(*filename*)
Read a utf-8 encoded text file and returns the text as string.

This is just a utily function, that is not recommended to use if the text file does not fit your available RAM. Mostly used for small text files.

Parameters **filename** (*string*) – File from which is the text to be read

Returns Text in the file

Return type string

BIBLIOGRAPHY

[iAMC05] Laura Alonso i Alemany, Irene Castellón Masalles, and Lluıs Padró Cirera. Representing discourse for automatic text summarization via shallow nlp techniques. *Unpublished PhD thesis, Universitat de Barcelona*, 2005.

[RSG14] Ismael Díaz Rangel, Grigori Sidorov, and Sergio Suárez Guerra. Creación y evaluación de un diccionario marcado con emociones y ponderado para el español. *Onomazein*, pages 31–46, 2014.

[BL08] Regina Barzilay and Mirella Lapata. Modeling local coherence: an entity-based approach. *Computational Linguistics*, 34(1):1–34, 2008.

[GS13] Camille Guinaudeau and Michael Strube. Graph-based local coherence modeling. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 93–103. 2013.

[HDM+05] Christian F Hempelmann, David Dufty, Philip M McCarthy, Arthur C Graesser, Zhiqiang Cai, and Danielle S McNamara. Using lsa to automatically identify givenness and newness of noun phrases in written discourse. In *Proceedings of the 27th annual conference of the Cognitive Science Society*, 941–946. Erlbaum Mahwah, NJ, 2005.

[DPSebastianGalles+13] Andrew Duchon, Manuel Perea, Nuria Sebastián-Gallés, Antonia Martí, and Manuel Carreiras. Espal: one-stop shopping for spanish word properties. *Behavior research methods*, 45(4):1246–1258, 2013.

[GFerreF16] Marc Guasch, Pilar Ferré, and Isabel Fraga. Spanish norms for affective and lexico-semantic variables for 1,400 words. *Behavior Research Methods*, 48(4):1358–1369, 2016.

[CKM16] Scott A Crossley, Kristopher Kyle, and Danielle S McNamara. The tool for the automatic analysis of text cohesion (taaco): automatic assessment of local, global, and text cohesion. *Behavior research methods*, 48(4):1227–1237, 2016.

[FKL98] Peter W Foltz, Walter Kintsch, and Thomas K Landauer. The measurement of textual coherence with latent semantic analysis. *Discourse processes*, 25(2-3):285–307, 1998.

[GALR12] Aitor Gonzalez-Agirre, Egoitz Laparra, and German Rigau. Multilingual central repository version 3.0. In *LREC*, 2525–2529. 2012.

[MCC+13] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, L Sutskever, and G Zweig. Word2vec. *URL https://code. google. com/p/word2vec*, 2013.

[RM00] Brian Richards and David Malvern. Measuring vocabulary richness in teenage learners of french. *British Educational Research Association Annual Conference*, 2000.

[Yul14] C Udny Yule. *The statistical study of literary vocabulary*. Cambridge University Press, 2014.

PYTHON MODULE INDEX

t

INDEX

I

L

N

O

P