# lingtools Documentation

**Maira B. Carvalho**

**Sep 04, 2018**

# Contents

This is a module that extracts linguistic features from texts.

To see a complete list of the features extracted by this code, see *Introduction*.

CHAPTER 1

# Introduction

This is a package that extracts linguistic features from texts.

## 1.1 Features extracted

These are the features extracted by the package:

### 1.1.1 Basic features

Some basic linguistic features, such as word count, sentence count, average sentence length, and incidence of certain grammatical classes.

Incidences are normalized by word count, so they range from 0 to 1.

- word_count
- sentence_count
- paragraph_count
- avg_word_length
- sd_word_length
- avg_sentence_length
- sd_sentence_length
- avg_paragraph_length
- sd_paragraph_length
- noun_incidence
- verb_incidence
- adjective_incidence

- adverb_incidence
- pronoun_incidence
- 1p_sing_pronoun_incidence
- 1p_pl_pronoun_incidence
- 2p_pronoun_incidence
- 3p_sing_pronoun_incidence
- 3p_pl_pronoun_incidence

### 1.1.2 Biber features

These are the 67 features selected by Douglas Biber [1] to reflect the linguistic structure of text.

Features are normalized by word count and multiplied by 1000 (range: 0-1000), with the exception of type_token_ratio and word_length.

- past_tense
- perfect_aspect_verbs
- present_tense
- place_adverbials
- time_adverbials
- first_person_pronouns
- second_person_pronouns
- third_person_pronouns
- pronoun_it
- demonstrative_pronouns
- indefinite_pronouns
- do_as_proverb
- wh_questions
- nominalizations
- gerunds
- nouns
- agentless_passives
- by_passives
- be_as_main_verb
- existential_there
- that_verb_complements
- that_adj_complements
- wh_clauses
- infinitives

- present_participial_clauses
- past_participial_clauses
- past_prt_whiz_deletions
- present_prt_whiz_deletions
- that_relatives_subj_position
- that_relatives_obj_position
- wh_relatives_subj_position
- wh_relatives_obj_position
- wh_relatives_pied_pipes
- sentence_relatives
- adv_subordinator_cause
- adv_sub_concesssion
- adv_sub_condition
- adv_sub_other
- prepositions
- attributive_adjectives
- predicative_adjectives
- adverbs
- type_token_ratio
- word_length
- conjuncts
- downtoners
- hedges
- amplifiers
- empathics
- discourse_particles
- demonstratives
- possibility_modals
- necessity_modals
- predictive_modals
- public_verbs
- private_verbs
- suasive_verbs
- seems_appear
- contractions
- that_deletion

- stranded_prepositions
- split_infinitives
- split_auxilaries
- phrasal_coordination
- non_phrasal_coordination
- synthetic_negation
- analytic_negation

### 1.1.3 MRC features

These features are the average and standard deviation scores of the words in the document according to the Medical Research Council (MRC) Psycholinguistic Database [2].

The ranges of the features are the same as informed in the MRC database. See http://websites.psychology.uwa.edu.au/school/MRCDatabase/mrc2.html for details.

- avg_Nlet (range: 1-23)
- sd_Nlet
- avg_Nphon (range: 0-19)
- sd_Nphon
- avg_Nsyl (range: 0-9)
- sd_Nsyl
- avg_K-F-freq (maximum frequency in file: 69971)
- sd_K-F-freq
- avg_K-F-ncats (maximum frequency in file: 69971)
- sd_K-F-ncats
- avg_K-F-nsamp (maximum frequency in file: 69971)
- sd_K-F-nsamp
- avg_T-L-freq
- sd_T-L-freq
- avg_Brown-freq (range of entries: 0 - 6833)
- sd_Brown-freq
- avg_Familiarity (range: 100 - 700)
- sd_Familiarity
- avg_Concreteness (range: 100 - 700)
- sd_Concreteness
- avg_Imageability (range: 100 - 700)
- sd_Imageability
- avg_Meaningfulness-Colorado (range: 100 - 700)
- sd_Meaningfulness-Colorado

- avg_Meaningfulness-Paivio (range: 100 - 700)
- sd_Meaningfulness-Paivio
- avg_Age-of-acquisition (range: 100 - 700)
- sd_Age-of-acquisition

### 1.1.4 ANEW features

These features are the average and standard deviation scores of the words in the document according to the Affective Norms for English Words (ANEW) [3].

- avg_valence (range: 1-9)
- sd_valence
- avg_arousal (range: 1-9)
- sd_arousal
- avg_dominance (range: 1-9)
- sd_dominance

### 1.1.5 SemanticVectors

These are measures of text coherence, using representations in the LSA vector space to calculate distances. The features are presented as average and standard deviations in the text.

Cosine distances can range from -1.0 to +1.0.

- avg_cosdis_adjacent_sentences
- sd_cosdis_adjacent_sentences
- avg_cosdis_all_sentences_in_paragraph
- sd_cosdis_all_sentences_in_paragraph
- avg_cosdis_adjacent_paragraphs
- sd_cosdis_adjacent_paragraphs
- avg_givenness_sentences
- sd_givenness_sentences

### 1.1.6 SentimentAnalysis

Sentiment Analysis scores using NLTK's VADER sentiment analysis tool [4].

In the Vader lexicon, words are rated from -4 to +4 in the categories positive, negative and neutral. The compound value is a normalized combined score of the first three categories, ranging from -1.0 to +1.0.

- positive
- negative
- neutral
- compound

### 1.1.7 NER (Named Entity Recognition)

Recognized entities in the text, extracted using NLTK's NER tool. The values are normalized by word count (range: 0-1).

- organization
- person
- gpe
- location
- facility

### 1.1.8 Extra features

Some extra basic features, all normalized by word count (range: 0-1).

To recognize English words, we use NLTK's *word* and *brown* corpora.

- dollarsign
- eurosign
- poundsign
- numbers
- years
- englishwords
- nonenglishwords

## 1.2 References


- [1] Biber, D. (1988). Variation across speech and writing. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511621024
- [2] MRC Psycholinguistic Database. http://websites.psychology.uwa.edu.au/school/MRCDatabase/mrc2.html
- [3] Bradley, M. M., & Lang, P. J. (1999). Affective norms for English words (ANEW): Instruction manual and affective ratings (pp. 1-45). Technical report C-1, the center for research in psychophysiology, University of Florida.
- [4] Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.


## 1.3 Author


- **Maira B. Carvalho** (m.brandaocarvalho@uvt.nl), Tilburg University

CHAPTER 2

# Installation

To run this module, you will need to do the following:

- Clone the repository
- Install requirements
- Install the module
- Install the assets

## 2.1 Clone the repository

```
cd ~
git clone git@bitbucket.org:catdevrandom/lingtools.git
```

## 2.2 Install requirements

Make sure you have the requisites installed. You can use the file *requirements.txt* to install the dependences using pip:

```
cd ~/lingtools
pip install -r requirements.txt
```

### 2.2.1 Prerequisites

Software:

- Python 2.7 (it might work with Python 3, but it has not been tested).

Python packages:

See file *requirements.txt*, in the root of the repository.

NLTK data (in a place where NLTK can find it, e.g. ~/nltk_data):

- maxent_ne_chunker
- brown
- sentiwordnet
- stopwords
- wordnet
- words
- vader_lexicon
- averaged_perceptron_tagger
- maxent_treebank_pos_tagger
- punkt

Assets (see section *Install assets*):

- ANEW database
- MRC2 database
- LSA model trained on the TASA corpus

## 2.3 Install module

You can use the setup.py script. For that, you need the module *setuptools* to be installed:

```
pip install setuptools
python setup.py install
```

## 2.4 Install assets

The assets are not distributed with the code. To be able to use the software, you need to download the assets and unzip the file in the assets folder:

```
cd ~/.lingtools
wget https://surfdrive.surf.nl/files/index.php/s/2FVDToX8NwmhHpr/download -O assets.
↪zip
unzip assets.zip
```

## 2.5 Running the tests

You can run the test suit to make sure the installation works. This software uses the built-in module *unittest*:

```
cd tests
./runtests.sh
```

CHAPTER 3

# Configuration

Lingtools will look for a configuration file at *~/.lingtools/config.yml* by default. A sample configuration file is placed at *~/.lingtools/config.yml.sample*. You can rename and edit it as needed.

Alternatively, it is possible to inform an alternative configuration file path to most lingtools classes. Check in the *lingtools API* which classes accept this option.

## 3.1 File format

The format of the configuration file is YAML (YAML Ain't Markup Language), which is a human-readable data serialization language. In YAML, key value pairs within a map are separated by a colon, and structure is shown through indentation (one or more spaces). Comments begin with the number sign (#), can start anywhere on a line and continue until the end of the line. More information about YAML can be found at the official website: http://yaml.org/start.html.

The lingtools configuration file accepts the following settings, divided in three groups:

- **memory**
  - limit
- **assets**
  - anew
  - mrc
  - lcm
- **vector_spaces**
  - [vector space entries]

## 3.2 Memory

Check the maximum memory allocation for the module (in bytes). The default value is 2 GiB, which might not be enough to analyze large files.

```
memory:
    limit: 2147483648
```

## 3.3 Assets

### 3.3.1 Assets location

You can configure the path in which lingtools assets can be found. By default, they will be placed in the folder *.lingtools*, in the user's home folder.

```
assets:
    path: /home/USER/.lingtools
```

### 3.3.2 Assets

Lingtools requires some assets to run. These should be downloaded (see installation instructions) and placed in the configured path (see above).

The ANEW, MRC and LCM dictionaries are expected to be provided as a marisa trie (http://marisa-trie.readthedocs.io) pickled object.

#### ANEW dictionary

The ANEW norms of English Language. Configuration key: *anew*.

#### MRC dictionary

The MRC Psycholinguistic Database _http://websites.psychology.uwa.edu.au/school/MRCDatabase/mrc2.html. Configuration key *mrc*.

#### LCM dictionary

Linguistic category Model dictionary. Configuration key *lcm*.

## 3.4 Vector spaces configuration

In the configuration file, you can inform vector spaces to be made available to lingtools. It is possible to inform as many vector spaces as desired. For more information about vector spaces and the file formats that can be used, see *Vector spaces* in the page *How to code with lingtools*.

Each vector space entry (marked with indentation and an hyphen) must contain the following fields:

- **name**: the vector space name (without spaces)

- **description**: a one-line description of the vector space
- **dict**: dictionary for the vector space, which can be a text file with one word per line or a gensim dictionary (https://radimrehurek.com/gensim/).
- **vectors**: vector space file, a dense matrix of N-dimensional vectors, in which each row represents a word. There must be a 1:1 match between the vectors in this file and the words in the dictionary file, in correct order.

---

**Hint:** Loading gensim dictionaries is significantly faster than loading dictionaries from plain text. If speed is a concern, convert the dictionary file to a gensim dictionary. Important: ensure that the indices of the gensim dictionary match the vector space exactly!

---

There are many pre-trained vector spaces available online that can be used with lingtools, in English and in other languages. For example:

- GloVe: Global Vectors for Word Representation: https://nlp.stanford.edu/projects/glove/
- FastText: https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md
- LSA spaces for R: http://www.lingexp.uni-tuebingen.de/z2/LSAspaces/ and https://sites.google.com/site/fritzgntr/software-resources

In most cases, the pre-trained vector spaces are available in the format of rows of a dense matrix stored in tab-delimited format (first element of each line corresponds to a word, followed by the values in the vector representing it). This format has to be processed as to extract the first column (the words) into a separate file, leaving the dense matrix on its own to be imported directly by numpy.

An example of configuration for two vector spaces in YAML format:

```
vector_spaces:
    - name: fasttext
      description: Fast Text, trained on Wikipedia English, 300 dimensions
      dict: PATH_TO_DICT_FILE
      vectors: PATH_TO_VECTOR_FILE
    - name: glove
      description: Global Vectors, 6B tokens, uncased, 300 dimensions
      dict: PATH_TO_DICT_FILE
      vectors: PATH_TO_VECTOR_FILE
```

These vector spaces are made available for the modules in addition to the default LSA space (see *Default vector space*).

### 3.4.1 Default vector space

The default vector space is an 300 dimension LSA model trained using the TASA corpus, lemmas only. For details, see the description of the model TASA 1 in the paper: Ştefănescu, D., Banjade, R., Rus, V.: Latent Semantic Analysis Models on Wikipedia and TASA, LREC (2014), available at http://deeptutor2.memphis.edu/Semilar-Web/public/lsa-models-lrec2014.html.

## 3.5 Example configuration file

```
memory:
    limit: 2147483648 # 2 * 1024 * 1024 * 1024 = 2 GiB

assets:
```

(continues on next page)

(continued from previous page)

```
    path: /home/USER/.lingtools
    anew: /home/USER/.lingtools/assets/anew.marisa.pickle
    mrc: /home/USER/.lingtools/assets/mrc2.marisa.pickle
    lcm: /home/USER/.lingtools/assets/lcm.marisa.pickle

vector_spaces:
    - name: fasttext
      description: Fast Text, trained on Wikipedia English, 300 dimensions
      dict: /home/USER/.lingtools/assets/fasttext_dict
      vectors: /home/USER/.lingtools/assets/fasttext_model
    - name: glove
      description: Global Vectors, 6B tokens, uncased, 300 dimensions
      dict: /home/USER/.lingtools/assets/glove6B_dict
      vectors: /home/USER/.lingtools/assets/glove6B_model
```

CHAPTER 4

# Command line usage

***Table of contents***



## 4.1 Lingtools helper script

If all you need is to extract features from text, you can use the *lingtools_script.py* script in the *bin* directory.

After installing the lingtools package (see *Installation*), go to the bin directory:

```
cd ~/src/lingtools/bin
```

Check the script help for options:

```
python lingtools_script.py -h
```

Each functionality of the script has its own help page. For example:

```
python lingtools_script.py features -h
```

### 4.1.1 Generate a configuration file

For more control regarding how the text is processed, you can inform extra settings to the script. These settings are the same as described in the API (see, for example, *lingtools.tokenizer.LocalTokenizer*).

You can generate a sample configuration file with the default options:

```
python lingtools_script.py generate --output config.yml
```

The sample configuration file can be edited as needed and informed to the script, for example:

```
python lingtools_script.py features INPUT_PATH --config=config.yml
```

### 4.1.2 Extract features

You can extract features from text using the command "features".

By default, the script takes a simple text file in which each line is a document and outputs the results to the console:

```
python lingtools_script.py features INPUT_PATH
```

You can also treat the input file as one single document:

```
python lingtools_script.py features INPUT_PATH --type txt_single
```

You can also use CSV files as input, with any separator character (for example a semi-colon) and skipping one line to ignore the header:

```
python lingtools_script.py features INPUT_PATH --type csv --sepin=";" --skip=1
```

If you want, the script can write the results to an output file. You can choose which separator character will be used:

```
python lingtools_script.py features INPUT_PATH --output="output.csv" --sepout="tab"
```

Logging messages can be directed to a file:

```
python lingtools_script.py features INPUT_PATH --log="log.txt"
```

And the debug flag can turn on extensive debugging messages (a log file path must also be provided):

```
python lingtools_script.py features INPUT_PATH --log="log.txt" --debug
```

For more configuration options, use the configuration file generated by the command *generate* (see *Generate a configuration file*).

### 4.1.3 Dynamic themes

You can calculate cosine distances between a list of theme words (split by commas) and one or more documents:

```
python lingtools_script.py themes INPUT_PATH "theme, words, write"
```

Other options are similar as described in *Extract features*. For a list of all options, consult the command help:

```
python lingtools_script.py themes -h
```

### 4.1.4 Cosine similarity matrix

With this command, you can calculate a matrix of cosine distances between the lines of a file. Each line is considered a document, and each document can contain one word or multiple words:

```
python lingtools_script.py cosines INPUT_PATH
```

Other options are similar as described in *Extract features*. For a list of all options, consult the command help:

```
python lingtools_script.py cosines -h
```

### 4.1.5 Nearest neighbors

With this command, you can get the nearest neighbors from a piece of text (one or more words):

```
python lingtools_script.py neighbors TEXT
```

Note that, unlike other commands, this command takes the raw text and not a file path. If you want to pass the contents of a file, you can pipe in the file:

```
python lingtools_script.py neighbors < INPUT_PATH
```

Other options are similar as described in *Extract features*. For a list of all options, consult the command help:

```
python lingtools_script.py neighbors -h
```

### 4.1.6 Distinctive words

With this command, given two sets of texts, you can see the words that most distinguish the two:

```
python lingtools_script.py words INPUT_PATH_1 INPUT_PATH_2
```

Other options are similar as described in *Extract features*, but with different options for file 1 and file 2. For a list of all options, consult the command help:

```
python lingtools_script.py words -h
```

CHAPTER 5

# How to code with lingtools

***Table of contents***



## 5.1 Coding with lingtools

Lingtools is a package that provides the following functions:

- A tokenizer wrapper class, which is used consistently by the other classes (class `lingtools.tokenizer.LocalTokenizer`).

- A feature extractor class, which processes text into numerical features (class *lingtools.featureextractor.FeatureExtractor*). This class can be extended with add-ons.
- A class to calculate cosine similarity values and nearest neighbors (class *lingtools.cosinedists.CosineDists*).
- A class to evaluate documents using dynamically defined themes (class *lingtools.dynamicthemes.DynamicThemes*).
- A frequency analyzer class, which compares texts or groups of texts to extract most distinctive words (class *lingtools.freqanalyzer.FreqAnalyzer*).

The lingtools package has other auxiliary classes, but the user does not need to know them to use the module. If you want to know more about the remaining classes, you can check the *lingtools API*.

Roughly speaking, to use lingtools, you need to perform the following steps:

1. Instantiate a tokenizer object with the desired settings;
2. Instantiate an object of the class providing the functionality you want (feature extractor, dynamic themes, etc) and loading the desired vector space;
3. Load a file or a list of documents in the class;
4. Call the appropriate processing function.

These steps will be described in detail below.

## 5.2 Local tokenizer

The class *lingtools.tokenizer.LocalTokenizer* ensures that the user has control over how the document is pre-processed and that the pre-processing steps are the same across different functionalities.

This class can use either NLTK's (https://www.nltk.org/) pos-tagging and lemmatizing functions, or a server running Stanford CoreNLP (https://stanfordnlp.github.io/CoreNLP/). NLTK is the default option because it runs locally without any further configuration.

This is how you instantiate a tokenizer object using the default NLTK tagger:

```
from lingtools.tokenizer import LocalTokenizer

tokenizer_settings = {"language":"english",
                      "encoding": "utf-8",
                      "tagger": "nltk",
                      "stanford_url": None,
                      "simple_tokenizer": False,
                      "remove_stopwords": True,
                      "stopwords": None, # Use NLTK default stopwords
                      "replace_digits": True,
                      "remove_punctuation": True,
                      "replace_not": True,
                      "lowercase": False,
                      "use_lemmas": True}
tokenizer_obj = LocalTokenizer(**tokenizer_settings)
```

To use a Stanford CoreNLP server, it is necessary to inform the URL in the format *<protocol>://<server>:<port>* .

```
tokenizer_settings_stanford = {"language":"english",
                               "encoding": "utf-8",
```

(continues on next page)

(continued from previous page)

```
                                "tagger": "stanford",
                                "stanford_url": "http://localhost:9000",
                                "simple_tokenizer": False,
                                "remove_stopwords": True,
                                "stopwords": None, # Use NLTK default stopwords
                                "replace_digits": True,
                                "remove_punctuation": True,
                                "replace_not": True,
                                "lowercase": False,
                                "use_lemmas": True}
tokenizer_stanford_obj = LocalTokenizer(**tokenizer_settings_stanford)
```

Make sure the CoreNLP server is configured to output documents in the 'json' format and that it uses the following annotators, in this order: 'tokenize,pos,lemma,ssplit'.

The stanford-corenlp-full-2018-02-27 server can be started (locally, in port 9002, with 4Gb memory) in the following manner:

```
# Run the server using all jars in the current directory (e.g., the CoreNLP home␣
↪directory)
java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9002 -
↪timeout 15000
```

For more information on how to run the CoreNLP server, see https://stanfordnlp.github.io/CoreNLP/corenlp-server.html.

### 5.2.1 Deep structure pos-tagged document

The *LocalTokenizer* class generates a specially formatted tagged document, the "deep structure pos-tagged document". Rarely, if ever, this format will be used directly by the user when utilizing the lingtools package.

The "deep structure pos-tagged document" is a 3-dimensional list of PennBank POS-tagged and lemmatized tokens (list of paragraphs, which contains a list of sentences, which contains a list of tagged tokens). The tagged tokens have three components: the original token, its POS-tag and its lemma.

For example, take the following raw text (double line break indicates a new paragraph):

```
The sky is blue.

My name is John.
```

The LocalTokenizer processes this document into the following deep structure pos-tagged document:

```
[ # document level
  [ # paragraph level
    [ # sentence level
       (u'The', 'DT', 'the'),
       (u'sky', 'NN', u'sky'),
       (u'is', 'VBZ', u'be'),
       (u'blue', 'JJ', u'blue'),
       (u'.', '.', u'.')]
    ],
  [
    [
       (u'My', 'PRP$', u'my'),
       (u'name', 'NN', u'name'),
```

(continues on next page)

(continued from previous page)

```
        (u'is', 'VBZ', u'be'),
        (u'John', 'NNP', u'john'),
        (u'.', '.', u'.')
    ]
  ]
]
```

## 5.3 Vector spaces

Lingtools requires a vector space to perform most of its functions. Lingtools comes with a *Default vector space*, which will be used if no other option is selected. Other vector spaces can be configured and called by name when instantiating a class object. See *Vector spaces configuration* for details.

## 5.4 Loading files or lists

For most lingtools classes, you have to first load the instantiated object with a file or word/document list that will be processed on the fly. The appropriate processing function can only be called once the file or the list of documents is loaded in the object.

### 5.4.1 Files

The `load_file()` function allows lingtools to process large files without having to load everything in memory first.

Lingtools accepts the following file formats:

- Multiple documents as comma separated values (.csv)
- Multiple documents as plain text (.txt)
- Single document as plain text (.txt)

For CSV files, it is possible to choose which separator character to use.

For multiple document files, it is possible to indicate how many lines to skip (to account for a header, for example).

For single document text files, it is possible to indicate which character should be consider a paragraph splitter.

For all options, it is possible to inform the encoding (default: utf-8).

---

**Important:** In CSV files, it is expected that the first column contains the ID of the document, and the second column contains the text. All remaining columns are ignored.

---

### 5.4.2 Lists

It is also possible to load simple lists of documents instead of files, using the function `load_list()`.

---

**Important:** Always use unicode strings! Using byte strings will cause lingtools to throw an Exception. For more information, see https://docs.python.org/2/howto/unicode.html.

---

## 5.5 Extracting features

To extract features from text, you need to instantiate a feature extractor object from class *FeatureExtractor*, using a previously instantiated LocalTokenizer object (see *Local tokenizer*):

```
from lingtools.featureextractor import FeatureExtractor

# Create a dictionary with settings, including the tokenizer object created previously
feature_extractor_settings = {"feature_sets": ["anew", "biber", "mrc"], # Choose␣
↪which modules to extract
                              "vectorspace_name": 'glove', # Non-default vector space␣
↪(optional)
                              "tokenizer_obj": tokenizer_obj
                              }

# Instantiate feature extractor object
feature_extractor = FeatureExtractor(**feature_extractor_settings)
```

Then, you are ready to load a file for extracting the features using the function *load_file()*:

```
feature_extractor.load_file('test.csv',
                            sep=';',
                            encoding='utf-8',
                            skip=1,     # Skip the header
                            file_type='csv')
```

Alternatively, a list of strings can be loaded using the function *load_list()*:

```
input_docs = [  u"This is the first document",
                u"This is the second document"  ]

# It is possible to load named indices for the documents
input_docs_indices = [u"doc1", u"doc2"]

feature_extractor.load_list(input_docs,
                            doc_index=input_docs_indices)
```

The loaded file is processed document by document by the function *process()*:

```
# Process results
for idx, result in feature_extractor.process():

    # Get a result tuple from the object:
    # (group_id, group_code, feature_id, feature_code, feature_value)
    print (idx, result.get_results_as_tuples())
```

An output file can be created with the help of the function *get_feature_names()*, which can be used to generate a header, and function *process_simple()*, which outputs a simple list of values (instead of a whole feature container):

```
# Create a header first
with open('result.csv', 'w') as f:
    header_items = ['docid'] + feature_extractor.get_feature_names()
    header_line = ";".join(header_items)
    f.write(header_line+"\n")

# Iterate over the documents and append values to the output file
```

(continues on next page)

(continued from previous page)

```
for idx, result_list in feature_extractor.process_simple():

    # Format the line using the feature value
    result_items = [idx] + [str(x) for x in result_list]
    result_line = ";".join(result_items)

    # Write the result
    with open('result.csv', 'a') as f:
        f.write(result_line + "\n")
```

### 5.5.1 Add-ons

The *FeatureExtractor* object accepts add-on modules. Add-ons are classes that implement at least the following methods:

- get_group_name(): returns a name for the feature set, so that it can be identified among the other groups. Make sure this name doesn't clash with the existing feature sets' names.
- get_feature_names(): returns a list of feature names processed by the add-on.
- get_features(): receives a deep structure document (see *Deep structure pos-tagged document*) and returns a list of the features processed by the class. The features must be returned in the same order as the list given by get_feature_names().

The add-on needs to be placed in the PYTHON_PATH so that lingtools is able to find and import it. Alternatively, it may be available in another package installed in the system.

Add-ons can be included in the FeatureExtractor object using the following dictionary format:

```
addon_extra = {
'name': 'Extra', # Human-readable name for the feature set
'code': 'extra', # Code for the feature set
'package': None, # Package where the addon comes from, if any
'module': 'extra', # Name of the module
'class': 'ExtraFeatures', # Class providing the add-on
'options': {} # Parameters to be passed to the class, if any
}
```

And the feature extractor object can be initialized with one or more add-ons:

```
# Create a dictionary with settings, including the tokenizer object created previously
settings_with_addon = { "feature_sets": ["anew", "biber", "mrc"], # Choose which␣
↪modules to extract
                        "vectorspace_name": 'glove', # Non-default vector space␣
↪(optional)
                        "tokenizer_obj": tokenizer_obj,
                        "addons": [addon_extra] }

# Instantiate feature extractor object
feature_extractor_2 = FeatureExtractor(**settings_with_addon)
```

## 5.6 Calculate cosine similarities

The class *lingtools.cosinedists.CosineDists* can be used to extract cosine similarity measurements between words/words, words/documents and documents/documents.

To generate a cosine similarity matrix between a list of words:

```
import pandas
from lingtools.cosinedists import CosineDists

# Create a dictionary with settings, including the tokenizer object created previously
cosdist_settings = {"vectorspace_name": 'glove', # Non-default vector space (optional)
                    "tokenizer_obj": tokenizer_obj
                   }

# Instantiate object
cosdist_obj = CosineDists(**cosdist_settings)

# Load a list of words
cosdist_obj.load_list([u"dog", u"cat", u"pizza"])

# Get matrix as a dictionary
results = cosdist_obj.get_matrix_as_dict()

# Use pandas to load results as a table
df = pandas.DataFrame.from_dict(results)
```

## 5.7 Nearest neighbors

The class *lingtools.cosinedists.CosineDists* can also be used to find nearest neighbors from a given word or document.

To extract nearest neighbors from a word or document:

```
from lingtools.cosinedists import CosineDists

# Create a dictionary with settings, including the tokenizer object created previously
cosdist_settings = {"vectorspace_name": 'glove', # Non-default vector space (optional)
                    "tokenizer_obj": tokenizer_obj
                   }

# Instantiate object
cosdist_obj = CosineDists(**cosdist_settings)

# Get nearest neighbors
result = cosdist_obj.get_nearest_neighbors(u"dog", n=10)
```

## 5.8 Dynamic themes

The class *lingtools.dynamicthemes.DynamicThemes* calculates cosine similarity values between a dynamically constructed theme and a list of documents.

To use the class:

```
from lingtools.dynamicthemes import DynamicThemes

# Create a dictionary with settings, including the tokenizer object created previously
dynthemes_settings = {"vectorspace_name": 'glove', # Non-default vector space␣
↪(optional)
```

(continues on next page)

(continued from previous page)

```
                      "tokenizer_obj": tokenizer_obj
                     }

# Instantiate object
dynthemes_obj = DynamicThemes(**dynthemes_settings)

# Load the theme
dynthemes_obj.load_theme("Coffee", ["coffee", "bean", "beverage", "hot"])

# Load the file
dynthemes_obj.load_file('test.csv',
                        sep=';',
                        encoding='utf-8',
                        skip=1,     # Skip the header
                        file_type='csv')

# Get the results
for idx, r in dynthemes_obj.process():
    print("Doc index: %s" % idx)
    print("Result tuple: %s" % r)
```

**Important:** Note that theme words will NOT be preprocessed. So, for example, if the vector space was trained using uses lemmas (as is the case with the *Default vector space*), theme words need to be informed already as lemmas.

The results are tuples in the following format: (theme_word, cosine_similarity, normalized_count). The first item refers to all theme words combined, while the following items list the similarity and count for each theme word separately:

```
# output:
Doc index: 1
Result tuple: [ (u'ALL', 0.10212916346547951, 0.04452212307975347),
                (['coffee'], 0.1343591231224161, 0.036151228037899),
                (['bean'], 0.1163471933552616, 0.004967344310551007),
                (['beverage'], 0.08517727367333683, 0.0018397571520559286),
                (['hot'], 0.06555257055307234, 0.0015637935792475394)
              ]
```

## 5.9 Extract distinctive words

The class *lingtools.freqanalyzer.FreqAnalyzer* can be used to compare two groups of documents and extract the most distinctive words between them using the chi-square test. For details on this approach, see: https://de.dariah.eu/tatom/feature_selection.html.

One additional step that needs to be performed to use this class is calling *lingtools.freqanalyzer.FreqAnalyzer.create_termdocmatrix()* after both group files have been loaded and before the function *lingtools.freqanalyzer.FreqAnalyzer.get_distinctive_words()* is called. See the example:

```
from lingtools.freqanalyzer import FreqAnalyzer

# Create a dictionary with settings, including the tokenizer object created previously
freqanalyze_settings = {"tokenizer_obj": tokenizer_obj}
```

(continues on next page)

(continued from previous page)

```
# Instantiate object
freqanalyze_obj = FreqAnalyzer(**freqanalyze_settings)

# Load file from the first group
freqanalyze_obj.load_file('group1.csv',
                          group_name="group1",
                          sep=';',
                          encoding='utf-8',
                          skip=1,     # Skip the header
                          file_type='csv')

# Load file from the second group
freqanalyze_obj.load_file('group2.csv',
                          group_name="group2",
                          sep='\t',
                          encoding='utf-8', # Must be the same as the other file
                          skip=1,     # Skip the header
                          file_type='csv')

# Generate term-document matrix
freqanalyze_obj.create_termdocmatrix()

# Get the result
result = freqanalyze_obj.get_distinctive_words('group1', 'group2')
```

**Important:** Both files must have the same encoding!

The function returns a list of tuples in the following format: (word, chi-square, pval, word rate (per 1000 words) in group 1, word rate (per 1000 words) in group 2, group where word appears most). If there are no distinctive words between the two groups (that is, the files are too similar), the list will be empty.

Example result:

```
# print result
# (word, chi-square, pval, word_rate_group1, word_rate_group2, from_group)
[ (u'use', 15.777249575551785, 7.125418302648217e-05, 0.032749304077288356, 0.
↪6091370558375634, 2),
  (u'good', 14.004340900039825, 0.00018238907279415775, 0.13099721630915342, 0.
↪77834179357022, 2),
  (u'product', 12.342331635540466, 0.0004428016171161547, 0.06549860815457671, 0.
↪5752961082910322, 2),
  (u'one', 6.531100082712987, 0.010600437725800424, 0.06549860815457671, 0.
↪37225042301184436, 2),
  (u'size', 5.940777502067824, 0.014794490914363133, 0.360242344850172, 0.
↪0676818950930626, 1),
  (u'look', 5.408346134152585, 0.020040694334323265, 0.5239888652366138, 0.
↪1692047377326565, 1),
  (u'pretty', 5.218757467144564, 0.022344511137499933, 0.26199443261830685, 0.
↪0338409475465313, 1),
  (u'small', 4.475395319418088, 0.0343862467947065, 0.4257409530047487, 0.
↪1353637901861252, 1),
  (u'long', 4.306586021505376, 0.03796507910969707, 0.22924512854101853, 0.
↪0338409475465313, 1)  ]
```

# CHAPTER 6

# lingtools API

## 6.1 lingtools package

### 6.1.1 Submodules

#### lingtools.anew module

**class** lingtools.anew.**ANEWFeatures**(*tokenizer_obj=None*, *config_file=None*)

Bases: object

Extracts the average and standard deviation scores of the words in the document according to the Affective Norms for English Words (ANEW).

- avg_valence (range: 1-9)
- sd_valence
- avg_arousal (range: 1-9)
- sd_arousal
- avg_dominance (range: 1-9)
- sd_dominance

Reference: Bradley, M. M., & Lang, P. J. (1999). Affective norms for English words (ANEW): Instruction manual and affective ratings (pp. 1-45). Technical report C-1, the center for research in psychophysiology, University of Florida.

**Parameters**

- **tokenizer_obj** (LocalTokenizer) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()

Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)
Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()
Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

## lingtools.basicfeatures module

**class** lingtools.basicfeatures.**BasicFeatures**(*tokenizer_obj=None*, *config_file=None*)
Bases: object

Extracts some basic linguistic features, such as word count, sentence count, average sentence length, and incidence of certain grammatical classes.

Incidences are normalized by word count, so they range from 0 to 1.

- word_count
- sentence_count
- paragraph_count
- avg_word_length
- sd_word_length
- avg_sentence_length
- sd_sentence_length
- avg_paragraph_length
- sd_paragraph_length
- noun_incidence
- verb_incidence
- adjective_incidence
- adverb_incidence
- pronoun_incidence
- 1p_sing_pronoun_incidence
- 1p_pl_pronoun_incidence
- 2p_pronoun_incidence
- 3p_sing_pronoun_incidence
- 3p_pl_pronoun_incidence

**Parameters**

- **tokenizer_obj** (LocalTokenizer) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()
Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)
Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()
Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

### lingtools.cosinedists module

**class** lingtools.cosinedists.**CosineDists**(*logging_interval=None*, *dictionary_file=None*, *vectorspace_file=None*, *vectorspace_name=None*, *tokenizer_obj=None*, *config_file=None*)
Bases: object

Calculates matrices of cosine similarity values between documents.

**Parameters**

- **logging_interval** (*int*) – output logging info every *logging_interval* documents
- **tokenizer_obj** (LocalTokenizer) – initialized tokenizer object
- **vectorspace_name** (*string*) – Name of the vector space to be used (see *Configuration*).
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**get_matrix**()
Calculates a vector for each document (previously loaded with *load_list()* or *load_file()*) and gets the cosine similarity between all documents.

**Returns** a NxN matrix (as a list of tuples) of cosine similarity values. Values range from -1.0 (most dissimilar) to 1.0 (most similar)

**Return type** list of tuples (docid1, docid2, cosine_similarity)

**get_matrix_as_array**()
Calculates a vector for each document (previously loaded with *load_list()* or *load_file()*) and gets the cosine similarity between all documents.

**Returns** a list with the field names and a NxN matrix (as a numpy array) of cosine similarity values. Values range from -1.0 (most dissimilar) to 1.0 (most similar)

**Return type** (field_names, array)

**get_matrix_as_dict**()
Calculates a vector for each document (previously loaded with *load_list()* or *load_file()*) and gets the cosine similarity between all documents.

**Returns** a NxN matrix (as a dictionary) of cosine similarity values. Values range from -1.0 (most dissimilar) to 1.0 (most similar)

**Return type** dict (matrix[docid1][docid2] = cosine_similarity)

**get_nearest_neighbors**(*input_doc*, *n=5*, *algorithm=u'brute'*)
Get the n nearest neighbor words for an input document.

**Parameters**

- **input_doc** (*unicode*) – the raw word or document
- **n** (*int*) – the number of nearest neighbors to retrieve. Default: 5
- **algorithm** (*string*) – which algorithm to use (see `sklearn.neighbors.NearestNeighbors` for possible values).

**load_file**(*input_file*, *file_type=None*, *sep=None*, *encoding=None*, *skip=None*, *paragraph_sep=None*, *clean_extra_spaces=False*)
Loads a file for processing. See *lingtools.filereader.FileReader.load_file()* for a description of the options.

**load_list**(*doc_list*, *doc_index=None*, *paragraph_sep=None*, *clean_extra_spaces=False*)
Loads a list of (unicode) documents for processing. See *lingtools.filereader.FileReader.load_list()* for a description of the options.

Note that the list of documents loaded here is ignored if a file has been loaded with *load_file()*.

**unload_file**()
Clear a previously loaded file. This function needs to be called if a file has been loaded before, but the user wants to process a list of documents (loaded with *load_list()*) instead.

### lingtools.dynamicthemes module

**class** lingtools.dynamicthemes.**DynamicThemes**(*logging_interval=None*, *vectorspace_name=None*, *tokenizer_obj=None*, *config_file=None*)

Bases: `object`

Calculates cosine similarity values between a dynamically constructed theme and a list of documents.

**Parameters**

- **logging_interval** (*int*) – output logging info every *logging_interval* documents
- **tokenizer_obj** (`LocalTokenizer`) – initialized tokenizer object
- **vectorspace_name** (*string*) – Name of the vector space to be used (see *Configuration*).
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**get_theme_words**()
Get a clean list of the theme words. Only returns words that are recognized in the dictionary.

**Returns** A list of words that will be considered in the theme.

**Return type** list

**load_file**(*input_file*, *file_type=None*, *sep=None*, *encoding=None*, *skip=None*, *paragraph_sep=None*, *clean_extra_spaces=False*)
Loads a file for processing. See *lingtools.filereader.FileReader.load_file()* for a description of the options.

**load_list**(*doc_list*, *doc_index=None*, *paragraph_sep=None*, *clean_extra_spaces=False*)
Loads a list of (unicode) documents for processing. See *lingtools.filereader.FileReader.load_list()* for a description of the options.

Note that the list of documents loaded here is ignored if a file has been loaded with *load_file()*.

**load_theme**(*theme_name*, *informed_theme_words*, *add_theme_name_to_list=False*, *remove_duplicates=True*)
Load a theme in the object so that the documents can be processed.

Note that words will NOT be preprocessed. So, for example, if the vector space was trained using uses lemmas, theme words need to be informed already as lemmas.

**Parameters**

- **theme_name** (*unicode*) – A name for the theme.
- **informed_theme_words** (*list*) – A list of theme words (as unicode). Compound words are allowed.
- **add_theme_name_to_list** (*boolean*) – Should the theme name be considered a theme word? Default: False
- **remove_duplicates** (*boolean*) – Should duplicate theme words be removed? Default: True

**process**()
Processes previously loaded file or documents on the fly.

Calculates a vector for a user-given theme and gets the cosine similarity value between the document and the theme as a whole, and to each theme word separately.

The results are tuples in the following format: (theme_word, cosine_similarity, normalized_count). The first item refers to all theme words combined, while the following items list the similarity and count for each theme word separately.

**Yields** tuple (docid,(results))

**process_semantic_vectors**()
Process previously loaded documents on the fly.

Returns the representation of the document in the vector space.

**Yields** a tuple (docid,(semantic_vector))

**unload_file**()
Clear a previously loaded file. This function needs to be called if a file has been loaded before, but the user wants to process a list of documents (loaded with *load_list()*) instead.

**unload_theme**()
Clear a previously loaded theme.

### lingtools.extra module

**class** lingtools.extra.**ExtraFeatures**(*tokenizer_obj=None*, *config_file=None*)
Bases: object

Some extra basic features (numbers, special characters, English vs. non-English words), all normalized by word count (range: 0-1).

To recognize English words, we use NLTK's *word* and *brown* corpora.

- dollarsign
- eurosign
- poundsign
- numbers
- years
- englishwords
- nonenglishwords

**Parameters**

- **tokenizer_obj** (LocalTokenizer) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()
Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)
Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()
Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

### lingtools.featureextractor module

**class** lingtools.featureextractor.**FeatureExtractor**(*features=None*, *feature_sets=None*, *addons=None*, *logging_interval=None*, *vectorspace_name=None*, *tokenizer_obj=None*, *config_file=None*)
Bases: object

Extracts linguistic features from text. It is possible to load an input file, or to give a single text document to process.

**Parameters**

- **features** – DEPRECATED
- **feature_sets** (*list*) – list of features to be extracted from the text. Possible values: anew, basic, biber, extra, lcm, mrc, ner, semanticvectors, sentimentanalysis
- **addons** (*list<dict>*) – a list of dictionary entries with the information about extra add-on modules to be loaded. See *How to code with lingtools*.
- **logging_interval** (*int*) – output logging info every *logging_interval* documents
- **tokenizer_obj** (LocalTokenizer) – initialized tokenizer object
- **vectorspace_name** (*string*) – Name of the vector space to be used (see *Configuration*).
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**get_feature_names**()
Returns the ordered, flat list of feature names that will be generated.

**Returns** feature names

**Return type** list

**load_file**(*input_file*, *file_type=None*, *sep=None*, *encoding=None*, *skip=None*, *paragraph_sep=None*, *clean_extra_spaces=False*)
Loads a file for processing. See *lingtools.filereader.FileReader.load_file()* for a description of the options.

**load_list**(*doc_list*, *doc_index=None*, *paragraph_sep=None*, *clean_extra_spaces=False*)
Loads a list of (unicode) documents for processing. See *lingtools.filereader.FileReader.load_list()* for a description of the options.

Note that the list of documents loaded here is ignored if a file has been loaded with *load_file()*.

**process**()
Processes previously loaded file or documents on the fly.

Yields a tuple (docid,featureContainer), which is loaded with a result container from the document.

**Yields** a tuple (docid, featureContainer)

**process_simple**()
Processes previously loaded file or documents on the fly.

The result is in the format of a tuple (docid, result_list), with result_list being a simple list with the results for each feature, in the order given by get_feature_names().

**Yields** a tuple (docid, list_feature_values)

**unload_file**()
Clear a previously loaded file. This function needs to be called if a file has been loaded before, but the user wants to process a list of documents (loaded with *load_list()*) instead.

**class** lingtools.featureextractor.**FeaturesContainer**(*qualified_feature_sets*)
Bases: object

An object to contain extracted features from a text document.

**Parameters** **qualified_feature_sets** – Dictionary of features to be contained

**clear_results**()
Empty the results list in case it is needed to reuse the object.

**get_results**()
Returns the structure with values. If called before "load_results()", it will return the empty structure.

**get_results_as_tuples**()
Returns the results as a list of tuples: (group_id, group_code, feature_id, feature_code, feature_value).

**load_results**(*result_tuple*)
Receives a result tuple and returns a dictionary with feature values.

**Parameters** **result_tuple** – a list of (feature_code, feature_value) tuples

### lingtools.filereader module

**class** lingtools.filereader.**FileReader**(*logging_interval=None*)
Bases: *lingtools.filereader.FileReaderBase*

Base class for reading files or lists

**Parameters** **logging_interval** (*int*) – output logging info every *logging_interval* documents

**get_counter**()
Get the internal counter so that it is possible to keep track of how many documents have been read.

**get_docs**()
Generator function to iterate over the document source (whether list or file) while keeping track of the number of documents read so far via the internal counter.

**Yields** (docid, document)

**increment_counter**(*incr=1*)
Increment the internal counter by *incr* (default 1)

**Parameters** **incr** (*int*) – increment step (default 1)

**load_file**(*input_file*, *file_type=None*, *sep=None*, *encoding=None*, *skip=None*, *quotechar=None*, *paragraph_sep=None*, *clean_extra_spaces=False*)
Loads a file for processing.

In CSV files, it is expected that the first column contains the ID of the document, and the second column contains the text. All remaining columns are ignored.

**Parameters**

- **input_file** – path to the file.
- **type** – txt_single|txt_multiple|csv. Default: txt_single
- **skip** – How many lines to skip (in txt_multiple and csv files). Default: 0
- **paragraph_sep** – New paragraph separator (in txt_single and zip files). Default:
- **sep** – Separator in the CSV file. Default: ;
- **clean_extra_spaces** – boolean. If true, double spaces and extra spaces between new lines will be removed. Default: False
- **encoding** – Encoding of input file. Default: utf-8
- **quotechar** – Quoting character (for csv). Default: b""

**load_list**(*doc_list*, *doc_index=None*, *paragraph_sep=None*, *clean_extra_spaces=False*)
Loads a list of documents for processing. Note that if a file is loaded, the documents loaded by this function are ignored.

**Parameters**

- **doc_list** – a list of documents (unicode)
- **encoding** – Encoding of input file. Default: utf-8
- **paragraph_sep** – New paragraph separator (in txt_single and zip files). Default:
- **clean_extra_spaces** – boolean. If true, double spaces and extra spaces between new lines will be removed. Default: False

**reset_counter**()
Reset internal counter.

**unload_file**()
Clear a previously loaded file (so that the object can be used with a list of documents instead)

**class** lingtools.filereader.**FileReaderBase**
Bases: object

Base object for reading input files. Not to be used directly.

**class** lingtools.filereader.**MultipleFileReader**(*logging_interval=100*)
Bases: *lingtools.filereader.FileReaderBase*

A class that is able to read from multiple sources, particularly to allow document comparison (as in *lingtools.freqanalyzer.FreqAnalyzer*).

**get_docs**()
Generator function that iterates over all documents from all loaded document sources.

If there are multiple groups of documents, the group id will be prepended to the document id.

**Yields** (docid, document)

**load_file**(*input_file*, *group_name=None*, *file_type=None*, *paragraph_sep=None*, *sep=None*, *encoding=None*, *skip=None*, *quotechar=None*, *clean_extra_spaces=False*)
Loads a file for processing.

**Parameters**

- **input_file** – a CSV file containing an ID and the text itself. Remaining columns are ignored.
- **group_name** – the name identifying this group of documents.
- **type** – txt_single|txt_multiple|csv|zip. Default: txt_single
- **skip** – How many lines to skip (in txt_multiple and csv files). Default: 0
- **paragraph_sep** – New paragraph separator (in txt_single and zip files). Default:
- **sep** – Separator in the CSV file. Default: ;
- **encoding** – Encoding of input file. Default: utf-8
- **quotechar** – Quoting character. Default: b””

### lingtools.freqanalyzer module

**class** lingtools.freqanalyzer.**FreqAnalyzer**(*logging_interval=None*, *tokenizer_obj=None*, *config_file=None*)

Bases: object

Analyzes word frequencies to compare two groups of documents.

**Parameters**

- **logging_interval** (*int*) – output logging info every *logging_interval* documents
- **tokenizer_obj** (LocalTokenizer) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**create_termdocmatrix**()

Creates a term-document matrix using the documents in all the files loaded with *load_file()*.

This function must be called before *get_distinctive_words()*.

**get_distinctive_words**(*group1*, *group2*, *alpha=0.05*)

Compares two groups of documents and extracts the most distinctive words between them using Chi-square.

Before calling this function, you need to load at least two files with *load_file()* and call *create_termdocmatrix()*.

**Parameters**

- **group1** (*string*) – the name of the first group
- **group2** (*string*) – the name of the second group

**Returns** a tuple (word, chi-square, pval, word rate (per 1000 words) in group 1, word rate (per 1000 words) in group 2, group where word appears most)

**load_file**(*input_file*, *group_name=None*, *file_type=None*, *paragraph_sep=None*, *sep=None*, *encoding=None*, *skip=None*, *clean_extra_spaces=False*)

Loads a file for processing.

**Parameters**

- **input_file** – path to the file
- **group_name** (*string*) – a group name (as an identifier) for the documents in the file
- **type** – txt_single|txt_multiple|csv. Default: txt_single
- **skip** – How many lines to skip (in txt_multiple and csv files). Default: 0
- **paragraph_sep** – New paragraph separator (in txt_single and zip files). Default:
- **sep** – Separator in the CSV file. Default: ;
- **clean_extra_spaces** – boolean. If true, double spaces and extra spaces between new lines will be removed. Default: False
- **encoding** – Encoding of input file. Default: utf-8

### lingtools.keylookup module

**class** lingtools.keylookup.**KeyLookUp**(*lookup_obj*)

Bases: object

Wraps around the call for looking up a key in a dictionary and returning the associated value. Current implementation uses marisa_trie (http://marisa-trie.readthedocs.io/).

**Parameters** **lookup_obj** – The trie object where the look up will be performed.

**get**(*key*, *wildcards=False*)
Get the entry for *key*.

**Parameters**

- **key** – The key to lookup in the object
- **wildcards** – Should wildcards (*) be considered? Default: False

### lingtools.lcm module

**class** lingtools.lcm.**LCMFeatures**(*tokenizer_obj=None*, *config_file=None*)
Bases: object

Extract Linguistic Category Model features from text.

- Descriptive Action Verbs (DAV) refer to single, specific action with a clear beginning and end, such as hit, yell, and walk.
- Interpretative Action Verbs (IAV) refer to different actions with a clear beginning and end, but do not share a physical invariant feature, such as help, tease, avoid.
- State Action Verbs (SAV) refer to behavioral events, but refer to the emotional consequence of an action rather than the action itself, such as surprise, amaze, anger.
- State Verbs (SV) refer to enduring cognitive or emotional states with no clear beginning or end, such as hunger, trust, understand.
- Adjectives (ADJ) refer to a characteristic or feature qualifying a person or concept, such as distraught, optimal.

These five categories can be seen as a continuum from concreteness (DAV) to abstractness (ADJ). Semin and Fiedler (1991) proposed an aggregate of the five categories in the form of an abstractness score. This score was formed by the following straightforward formula:

$$abstractness = \frac{(DAV + (2 * (IAV + SAV)) + (3 * SV) + (4 * ADJ))}{(DAV + IAV + SAV + SV + ADJ)}$$

**Parameters**

- **tokenizer_obj** (*LocalTokenizer*) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()
Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)
Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()
Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

### lingtools.mrc module

**class** lingtools.mrc.**MRCFeatures**(*tokenizer_obj=None*, *config_file=None*)
Bases: object

Extract average and standard deviation scores of the words in the document according to the Medical Research Council (MRC) Psycholinguistic Database.

The ranges of the features are the same as informed in the MRC database.

- avg_Nlet (range: 1-23)
- sd_Nlet
- avg_Nphon (range: 0-19)
- sd_Nphon
- avg_Nsyl (range: 0-9)
- sd_Nsyl
- avg_K-F-freq (maximum frequency in file: 69971)
- sd_K-F-freq
- avg_K-F-ncats (maximum frequency in file: 69971)
- sd_K-F-ncats
- avg_K-F-nsamp (maximum frequency in file: 69971)
- sd_K-F-nsamp
- avg_T-L-freq
- sd_T-L-freq
- avg_Brown-freq (range of entries: 0 - 6833)
- sd_Brown-freq
- avg_Familiarity (range: 100 - 700)
- sd_Familiarity
- avg_Concreteness (range: 100 - 700)
- sd_Concreteness
- avg_Imageability (range: 100 - 700)
- sd_Imageability
- avg_Meaningfulness-Colorado (range: 100 - 700)
- sd_Meaningfulness-Colorado
- avg_Meaningfulness-Paivio (range: 100 - 700)

- sd_Meaningfulness-Paivio
- avg_Age-of-acquisition (range: 100 - 700)
- sd_Age-of-acquisition

Based on code from https://github.com/chbrown/lexicons

Reference: MRC Psycholinguistic Database. http://websites.psychology.uwa.edu.au/school/MRCDatabase/mrc2.html

**Parameters**

- **tokenizer_obj** (*LocalTokenizer*) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()

Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)

Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()

Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

### lingtools.ner module

**class** lingtools.ner.**NERFeatures**(*tokenizer_obj=None*, *config_file=None*)

Bases: object

Recognized entities in the text, extracted using NLTK's NER tool. The values are normalized by word count (range: 0-1).

- organization
- person
- gpe
- location
- facility

**Parameters**

- **tokenizer_obj** (*LocalTokenizer*) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()
Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)
Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()
Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

### lingtools.pybiber module

**class** lingtools.pybiber.**PyBiber**(*tokenizer_obj=None*, *config_file=None*)
Bases: object

These are the 67 features selected by Douglas Biber to reflect the linguistic structure of text.

Features are normalized by word count and multiplied by 1000 (range: 0-1000), with the exception of type_token_ratio and word_length.

- past_tense
- perfect_aspect_verbs
- present_tense
- place_adverbials
- time_adverbials
- first_person_pronouns
- second_person_pronouns
- third_person_pronouns
- pronoun_it
- demonstrative_pronouns
- indefinite_pronouns
- do_as_proverb
- wh_questions
- nominalizations
- gerunds
- nouns
- agentless_passives

- by_passives
- be_as_main_verb
- existential_there
- that_verb_complements
- that_adj_complements
- wh_clauses
- infinitives
- present_participial_clauses
- past_participial_clauses
- past_prt_whiz_deletions
- present_prt_whiz_deletions
- that_relatives_subj_position
- that_relatives_obj_position
- wh_relatives_subj_position
- wh_relatives_obj_position
- wh_relatives_pied_pipes
- sentence_relatives
- adv_subordinator_cause
- adv_sub_concesssion
- adv_sub_condition
- adv_sub_other
- prepositions
- attributive_adjectives
- predicative_adjectives
- adverbs
- type_token_ratio
- word_length
- conjuncts
- downtoners
- hedges
- amplifiers
- empathics
- discourse_particles
- demonstratives
- possibility_modals
- necessity_modals

- predictive_modals
- public_verbs
- private_verbs
- suasive_verbs
- seems_appear
- contractions
- that_deletion
- stranded_prepositions
- split_infinitives
- split_auxilaries
- phrasal_coordination
- non_phrasal_coordination
- synthetic_negation
- analytic_negation

Reference: Biber, D. (1988). Variation across speech and writing. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511621024

**Parameters**

- **tokenizer_obj** (*LocalTokenizer*) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()
Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)
Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()
Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

**get_named_features**(*input_document*)
Deprecated.

**update_rule_objects**(*rules_list*, *processed_doc*)
Iterates over the tokens in the processed document and applies all Biber rules, in the correct processing order.

**Parameters**

- **rules_list** –
- **processed_doc** –

**class** lingtools.pybiber.**WordInfo**(*word*, *lemma*, *penn_tag*, *biber_tags=None*)
Bases: object

Wrapper class for words that include lemmas, POS-tags and Biber tags.

Any type of bracket tags – L/RRB for parenthesis (), L/RSB for square brackets [] and L/RCB for curly brackets {} – are converted to NLTK tags "(" and ")", and the word and lemma are converted to round parenthesis too.

**Parameters**

- **word** – The original word
- **lemma** – The lemma
- **penn_tag** – the original penn tag
- **biber_tags** – the biber tags added to the word in the initial state, if any

**activate**(*i*)
Activate Biber rule for this word.

**Parameters** **(int)** (*i*) – the rule_id to activate

**display_tags**()
Display biber tags.

**ends_with**(*ending*)
Checks if the word ends with *ending*

**Parameters** **(unicode)** (*ending*) – the ending to check

**get_biber_tags**()
Get all biber tags

**get_lemma**()
Get the word lemma

**get_penn_tag**()
Get the word Penn tag

**get_word**()
Get the word in the original form.

**has_biber_tag**(*biber_tag*)
Checks if this Wordobject has *biber_tag*

**Parameters** **biber_tag** – tag to checl

**has_lemma**(*lemma*)
Checks if the object has this lemma

**Parameters** **lemma** – the lemma to verify

**is_activated**(*i*)
Check if rule is activated for this word.

**Parameters** **(int)** (*i*) – the rule_id to check

**set_biber_tag**(*biber_tag*)
Append biber tag to list of tags.

**Parameters** **biber_tag** – the biber tag to append.

**set_lemma**(*lemma*)
Set the lemma

**Parameters** **lemma** – the word's lemma

**set_word**(*word*)
Set the word.

**Parameters** **word** – original format of the word

### lingtools.semanticvectors module

**class** lingtools.semanticvectors.**SemanticVectorsFeatures**(*dictionary_file=None*, *vectorspace_file=None*, *vectorspace_name=None*, *logging_interval=None*, *tokenizer_obj=None*, *config_file=None*)

Bases: object

These are measures of text coherence, using representations in a vector space to calculate cosine similarities. The features are presented as average and standard deviations in the text.

Cosine similarities can range from -1.0 to +1.0, where higher values indicate most similar documents.

- avg_cosdis_adjacent_sentences
- sd_cosdis_adjacent_sentences
- avg_cosdis_all_sentences_in_paragraph
- sd_cosdis_all_sentences_in_paragraph
- avg_cosdis_adjacent_paragraphs
- sd_cosdis_adjacent_paragraphs
- avg_givenness_sentences
- sd_givenness_sentences

**Parameters**

- **logging_interval** (*int*) – output logging info every *logging_interval* documents
- **tokenizer_obj** (LocalTokenizer) – initialized tokenizer object
- **vectorspace_name** (*string*) – Name of the vector space to be used (see *Configuration*).
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()
Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)
Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()
Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

### lingtools.sentimentanalysis module

**class** lingtools.sentimentanalysis.**SentimentAnalysisFeatures**(*tokenizer_obj=None*, *config_file=None*)

Bases: object

Sentiment Analysis scores using NLTK's VADER sentiment analysis tool.

In the Vader lexicon, words are rated from -4 to +4 in the categories positive, negative and neutral. The compound value is a normalized combined score of the first three categories, ranging from -1.0 to +1.0.

- positive
- negative
- neutral
- compound

Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

**Parameters**

- **tokenizer_obj** (*LocalTokenizer*) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**static get_feature_names**()
Returns a list of feature names in the same order as the features returned by *get_features()*.

**Returns** list of feature names

**Return type** list

**get_features**(*deepstruc_doc*)
Returns features for one document, in the same order as returned by *get_feature_names()*.

**Parameters** **deepstruc_doc** (*list*) – The incoming document, pre-processed as a *Deep structure pos-tagged document*.

**Returns** list of features

**Return type** list

**static get_group_name**()
Get the human-readable name of the feature set.

**Returns** unique identifier of the feature set

**Return type** string

### lingtools.tokenizer module

**class** lingtools.tokenizer.**LocalTokenizer**(*language=None*, *encoding=None*, *tagger=None*, *stanford_url=None*, *simple_tokenizer=None*, *remove_stopwords=True*, *stopwords=None*, *replace_digits=True*, *remove_punctuation=True*, *lowercase=False*, *use_lemmas=False*, *replace_not=True*, *config_file=None*)

Bases: object

Utility class to tokenize and tag documents. By default uses NLTK, but it is possible to use also a running Stanford POS Tagger server.

**Parameters**

- **language** – Choose between 'english', 'dutch', 'russian'. Default: 'english'
- **encoding** – Incoming text encoding. Default: utf-8
- **tagger** – nltk or stanford. Default: nltk
- **stanford_url** – If using stanford tagger, address of the server. Default: http://localhost:9000
- **simple_tokenizer** – Use a simple white space tokenizer
- **remove_stopwords** – Default True
- **stopwords** – The list of stopwords to ignore. Default: NLTK's list
- **replace_digits** – Replaces digits with #. Default: True
- **remove_punctuation** – Removes punctuation. Default: True
- **replace_not** – Replaces "n't" with "not". Default: True
- **lowercase** – Converts text to lowercase before processing. Default: False
- **use_lemmas** – Uses lemmatized version of words. Default: False
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**get_deep_structure**(*input_document*, *transliterate=True*, *paragraph_sep=None*, *clean_extra_spaces=False*)

Takes a raw text and converts into a deep-structure document (paragraphs, sentences, tagged and lemmatized tokens).

**Parameters**

- **input_document** – Raw document (unicode)
- **lowercase** – boolean. If true, words will be converted to lowercase. Default: False
- **remove_punctuation** – boolean. If true, punctuation will be removed. Default: False
- **transliterate** – boolean. If true, special characters will be transliterated to ascii version. Default: True
- **paragraph_sep** – The paragraph boundary to split a document into paragraphs. Default: "nn"

**simplify_deepstruc**(*deepstruc_doc*, *flatten=False*)

Convert the input deep-structure document into a 3D list of tokens

**Parameters** **deepstruc_doc** – 3D list of (word, tag, lemma) (see *Deep structure pos-tagged document*)

### lingtools.tokenizer_nltk module

**class** lingtools.tokenizer_nltk.**NLTKTokenizer**(*language*, *encoding*, *simple_tokenizer*)
Bases: object

Utility class to tokenize and tag documents. By default uses NLTK, but it is possible to use also a running Stanford POS Tagger server.

**Parameters**

- **language** – Choose between 'english', 'dutch', 'russian'. Default: 'english'
- **encoding** – Incoming text encoding. Default: utf-8
- **simple_tokenizer** – Use a simple white space tokenizer

**get_deep_structure_paragraph**(*input_document*, *lowercase*, *remove_punctuation*)
Converts a raw paragraph into a list of sentences with a list of tagged tokens

**Parameters**

- **input_document** – Raw document (unicode)
- **lowercase** – boolean. If true, words will be converted to lowercase. Default: False
- **remove_punctuation** – boolean. If true, punctuation will be removed. Default: False

### lingtools.tokenizer_stanford module

**class** lingtools.tokenizer_stanford.**StanfordTokenizer**(*language=None*, *encoding=None*, *stanford_url=None*)
Bases: object

Utility class to tokenize, tag and lemmatize documents using a running Stanford CoreNLP server.

**Parameters**

- **language** – Choose between 'english', 'dutch', 'russian'. Default: 'english'
- **encoding** – Incoming text encoding. Default: utf-8
- **stanford_url** – If using stanford tagger, address of the server. Default: http://localhost:9000

**get_deep_structure_paragraph**(*raw_paragraph*, *lowercase*, *remove_punctuation*)
Converts a raw paragraph into a list of sentences, which contain a list of tagged tokens.

**Parameters**

- **raw_paragraph** – Raw paragraph (unicode)
- **lowercase** – boolean. If true, words will be converted to lowercase. Default: False
- **remove_punctuation** – boolean. If true, punctuation will be removed. Default: False

### lingtools.util module

lingtools.util.**get_words_from_deepstruc**(*deepstruc_doc*, *punctuation=False*)
Receives a 3d list (a tagged doc) and returns a flat list of words.

**Parameters**

- **deepstruc_doc** – 3D list of (word, tag, lemma) (see *Deep structure pos-tagged document*)
- **punctuation** – boolean. When True, includes punctuation in the list. Default: False

lingtools.util.**get_sentences_from_deepstruc**(*deepstruc_doc*)
Receives a 3d list (a tagged doc) and returns a flat list of sentences

**Parameters** **deepstruc_doc** – 3D list of (word, tag, lemma) (see *Deep structure pos-tagged document*)

lingtools.util.**get_wc_from_deepstruc**(*deepstruc_doc*, *punctuation=False*)
Receives a 3d list (a deep-structure doc) and returns the word count

**Parameters**

- **deepstruc_doc** – 3D list of (word, tag, lemma) (see *Deep structure pos-tagged document*)
- **punctuation** – boolean. When True, includes punctuation in the count. Default: False

lingtools.util.**get_sentence_count_from_deepstruc**(*deepstruc_doc*)
Receives a 3d list (a tagged doc) and returns the word count

**Parameters** **deepstruc_doc** – 3D list of (word, tag, lemma) (see *Deep structure pos-tagged document*)

lingtools.util.**is_doc_deepstruc**(*input_document*)
Checks if an input document is a deep-structure document. See *Deep structure pos-tagged document*)

**Parameters** **input_document** – the document to check

lingtools.util.**is_punctuation**(*token*)
Checks if a token is punctuation. Returns a boolean.

**Parameters** **token** – input token

lingtools.util.**remove_punctuation_chars**(*word*, *replacement=None*, *regular_expression=None*)
Replaces punctuation characters with 'replacement'

**Parameters**

- **word** – input word
- **replacement** – character that will substitute punctuation. Default = None
- **regular_expression** – The (regex module) regular expression to look for. Default = ur"[p{Punct}'~]+"

lingtools.util.**remove_special_chars**(*word*, *replacement=None*, *regular_expression=None*)
Replaces special characters with 'replacement'

**Parameters**

- **word** – input word
- **replacement** – character that will substitute special chars. Default = None
- **regular_expression** – The (regex module) regular expression to look for. Default = ur"p{S}+|[p{Punct}']+|p{M}+|p{N}+"

lingtools.util.**replace_digits_in_word**(*word*, *replacement=None*)
Replaces digits with another character.

**Parameters**

- **word** – the input word.
- **replacement** – a string. Default: #

lingtools.util.**remove_punctuation**(*token*)
If a token consists of only punctuation character, returns an empty string.

**Parameters** **token** – string

lingtools.util.**transliterate_special_chars**(*document*)
Transliterates special characters to ASCII using unidecode (https://pypi.org/project/Unidecode/). Euro (€) and Pound (£) symbols are mainained!

**Parameters** **document** – input document

lingtools.util.**memory_limit**(*config_file=None*)
Limit memory usage according to limit defined in the config file. See *Memory*.

### lingtools.wordvectorizer module

**class** lingtools.wordvectorizer.**WordVectorizer**(*dictionary_file=None*, *vectorspace_file=None*, *vectorspace_name=None*, *tokenizer_obj=None*, *config_file=None*)
Bases: object

The base class for the other feature extraction classes that rely on word embeddings.

**Parameters**

- **dictionary_file** (*string*) – Path for the dictionary for the vector space, which can be a text file with one word per line or a gensim dictionary
- **vectorspace_file** (*string*) – Path for a numpy matrix of N-dimensional vectors, in which each row represents a word. There must be a 1:1 match between the vectors in this file and the words in the dictionary file, in correct order.
- **tokenizer_obj** (LocalTokenizer) – initialized tokenizer object
- **config_file** (*string*) – Configuration file to use (optional. See *Configuration*).

**calculate_cosine_similarity**(*vec1*, *vec2*)
Calculates a cosine similarity value between two vectors. Vectors must have the same dimensionality.

**Parameters**

- **vec1** (*numpy.array or list that can be converted to numpy.array*) – first vector
- **vec2** (*numpy.array or list that can be converted to numpy.array*) – second vector

**combine_vectors**(*list_of_vectors*, *normalize=False*)
Combines vectors into one using vector addition, with optional normalizing to a unit vector.

**Parameters**

- **list_of_vectors** – the list of vectors to combine
- **normalize** (*boolean*) – Should the vectors be normalized to a unit vector?

**get_indices_from_tokens**(*list_of_tokens*)
Receives a flat list of tokens and returns their indices in the dictionary

**Parameters** **list_of_tokens** (*flat list*) – list of tokens to process

**get_structured_vectors**(*structure_of_tokens*)
Converts a 3-dimensional list of tokens (document > paragraph > sentences) into a 3-dimensional list of vectors

**Parameters** **structure_of_tokens** (*list*) – 3-dimensional list of tokens

**get_vector_from_raw_doc**(*raw_doc*, *normalize=False*)
Receives a raw document and returns the vectorized version of the document.

**Parameters**

- **raw_doc** (*unicode*) – raw document
- **normalize** (*boolean*) – Should the vectors be normalized to a unit vector?

**get_vector_from_tokens**(*list_of_tokens*, *normalize=False*)
Receives a list of tokens and returns the vectorized version of the document.

**Parameters**

- **list_of_tokens** (*list*) – flat list of tokens
- **normalize** (*boolean*) – Should the vectors be normalized to a unit vector?

**get_vectors_from_indices**(*list_of_indices*)
Gets the vectors from the indices.

**Parameters** **list_of_indices** (*list*) – list of indices of words in the dictionary

### 6.1.2 Module contents

CHAPTER 7

# License

This project is licensed under the MIT License.

The MIT License

CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## l

# Index

## H



## I



## K



## L

## M



## N



## P



## R



## S



## T



## U



## W