Loacore : Language and Opinion Analyzer for Comments and Reviews' Documentation

Paul Breugnot

Aug 13, 2018

CONTENTS:

CHAPTER

ONE

INTRODUCTION

This project was originally lead by the NLP research group of the Technichal University of Pereira, Colombia

The goal of this project is to provide a Python library that allow to easily run language processing analysis such as sentiment analysis or depency tree analysis, specialized in comments and reviews such as hotel and restaurant reviews, movie reviews, products reviews... Thus, the library provide an high level API to run backend Freeling processes or other hard coded processes involving nltk and WordNet corpus for example.

Still under development

CHAPTER

TWO

REQUIREMENTS

2.1 Freeling

This project uses an external program, called Freeling, to process language : http://nlp.lsi.upc.edu/freeling/
Check this page to install Freeling on your computer :
https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/installation/apis-linux.html
Notice that to use Python API, Freeling needs to be installed from source with the dedicated options as described in documentation. *(Not sure, this needs to be checked.)*
The folder */usr/local/* is used as the default Freeling path, because it is the default Freeling installation folder on Linux systems. This path can be set at any moment using `set_freeling_path()`, on Windows, Linux or MacOS.

2.2 Database

2.2.1 SQlite

The embedded database used to store results is an sqlite database, managed with the sqlite3 Python database API :
https://docs.python.org/3/library/sqlite3.html
The corresponding Python package should already be installed in your Python3 distribution.

2.2.2 Prebuild DB

Incomplete section
Because the database is quite heavy, it is not included in the GitHub repository. Of course you could build it on your own, but because this process can be quite long prebuild databases can be found there *enter url* with following files:

- IMDB
- UCI

2.3 Machine Learning

Machine learning modules are based on two Python libraries.

- Gensim : https://radimrehurek.com/gensim/

Installation : https://radimrehurek.com/gensim/install.html

- Scikit Learn : http://scikit-learn.org/stable/
  Installation : http://scikit-learn.org/stable/install.html

2.4 Utils

Package `utils` uses a few graphical modules to show results.

- PrettyTable : https://pypi.org/project/PrettyTable/
- Matplotlib : https://matplotlib.org/users/installing.html#linux
- Tkinter : https://wiki.python.org/moin/TkInter
  Module used to generate gui to save pdf for example.
  Even if the package should be included in Python distribution, on Linux distributions you might need to install *tk* package through your package manager.

CHAPTER

THREE

CONFIGURATION

loacore.conf.**check_freeling_path**()
Return current *freeling_path*.

loacore.conf.**check_lang**()
Return current *lang*.

loacore.conf.**set_freeling_path**(*freeling_path*)
Set Freeling path. Changes are permanent, until the next call to *set_freeling_path()*.

Parameters **freeling_path** (path-like object) – Absolute path of the folder containing the *freeling* folder.

The path must always be specified in the following format : */path/to/freeling* . Python will then automatically format it according to the current OS.

loacore.conf.**set_lang**(*user_lang*)
Set default language used in Freeling.

Changes are permanent, until the next call to *set_lang()*.

Possible values : 'as', 'ca', 'cy', 'de', 'en', 'es', 'fr', 'gl', 'hr', 'it', 'nb', 'pt', 'ru', 'sl'.

Parameters **user_lang** (*str*) – Freeling language

CHAPTER

FOUR

CLASSES

4.1 File

class loacore.classes.classes.**File**(*id_file*, *file_path*)

Variables

- **id_file** (*int*) – ID_File used in File table
- **file_path** (*path-like object*) – Path used to load file from file system.
- **reviews** (list of *Review*) – File reviews

load(*encoding='utf8'*)

Load file from file system using file_path and specified encoding.

Parameters **encoding** – Source file encoding. Default : *utf8*.

Returns file object

sentence_list()

Convenient way to get all the sentences of the file, avoiding Reviews.

Returns List of file sentences

Return type list of *Sentence*

static sentence_list_from_files(*files*)

Convenient way to get all the sentences of the specified files, avoiding Reviews.

Parameters **files** (list of *File*) – Files

Returns List of file sentences

Return type list of *Sentence*

4.2 Review

class loacore.classes.classes.**Review**(*id_review*, *id_file*, *file_index*, *review*)

Variables

- **id_review** (*int*) – ID_Review used id Review table
- **id_file** (*int*) – SQL reference to the corresponding File
- **file_index** (*int*) – Index of the Review in referenced File
- **review** (*string*) – Review represented as a string

- **sentences** (list of *Sentence*) – Review Sentences
- **polarities** – Polarities associated to the review, that can come from directly from the source file for polarity label datasets, or from the result of different analysis. Each polarity can be identified with its analysis attribute.

review_str(*colored_polarity=True*, *analysis=[]*)

Parameters

- **colored_polarity** (*boolean*) – If True, polarities are colored printed. (red = Negative, green = Positive, yellow = Objective, black = No Synset)
- **analysis** (list of str) – The polarities computed with the specified analysis will be added at the end of each review.

Returns Review string representation

Return type string

4.3 Sentence

class loacore.classes.classes.**Sentence**(*id_sentence*, *id_review*, *review_index*, *id_dep_tree*)

Variables

- **id_sentence** (*int*) – ID_Sentence used in Sentence table
- **id_review** (*int*) – SQL reference to the corresponding Review
- **review_index** (*int*) – Index of the Sentence in referenced Review
- **id_dep_tree** (*int*) – SQL reference to a possibly associated DepTree
- **words** (list of *Word*) – Sentence Words
- **dep_tree** (*DepTree*) – Possibly associated DepTree
- **freeling_sentence** (pyfreeling.sentence) – result of *compute_freeling_sentence()* when called

compute_freeling_sentence()

Generates a basic pyfreeling.sentence instance, converting words as pyfreeling.word.

This function is used to process *Sentence* with Freeling.

Example Load sentences from database and convert them into Freeling Sentences.

```
>>> import loacore.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences()
>>> freeling_sentences = [s.compute_freeling_sentence() for s␣
↪in sentences]
```

return generated Freeling Sentence instance

rtype pyfreeling.sentence

sentence_str(*colored_polarity=False*)

Convenient way of printing sentences from their word list attribute.

Parameters **colored_polarity** (*boolean*) – If True, polarities are colored printed. (red = Negative, green = Positive, yellow = Objective, black = No Synset)

Returns String representation of the sentence

Return type string

4.4 Word

class loacore.classes.classes.**Word**(*id_word*, *id_sentence*, *sentence_index*, *word*, *id_lemma*, *id_synset*, *PoS_tag*)

Variables

- **id_word** (*int*) – ID_Word used in Word table
- **id_sentence** (*int*) – SQL reference to the corresponding Sentence
- **sentence_index** (*int*) – Index of the Word in referenced Sentence
- **word** (*string*) – Word form
- **id_lemma** (*int*) – SQL references to the corresponding Lemma (Table Lemma)
- **lemma** (*string*) – Possibly associated Lemma
- **id_synset** (*int*) – SQL references to corresponding Synset
- **synset** (*Synset*) – Possibly associated Synset
- **PoS_tag** (*string*) – Possibly associated Part-of-Speech tag
- **freeling_word** (pyfreeling.word) – result of *compute_freeling_word()* when called

colored_word()

Colored representation of word. If a Synset is associated, colored are assigned as follow :

- pos_score > neg_score => red
- neg_score > pos_score => green
- neg_score = pos_score => yellow

Returns Colored word

Return type string

compute_freeling_word()

Generates a basic pyfreeling.word instance, generated by only the word form, even if some analysis could have already been realized.

Moreover, only loacore.classes.classes.File.load_sentence() (that itself uses this function) should be used, because all Freeling analysis work with pyfreeling.sentence instances.

4.5 Synset

class loacore.classes.classes.**Synset**(*id_synset*, *id_word*, *synset_code*, *synset_name*, *neg_score*, *pos_score*, *obj_score*)

Variables

- **id_synset** (*int*) – ID_Synset used in Synset table

- **id_word** (*int*) – SQL reference to the corresponding Word
- **synset_code** (*string*) – Synset as represented in Freeling (ex : 01123148-a)
- **synset_name** (*string*) – Synset as represent in WordNet and SentiWordNet (ex : good.a.01)
- **neg_score** (*float*) – Negative polarity from SentiWordNet.
- **pos_score** (*float*) – Positive polarity from SentiWordNet.
- **obj_score** (*float*) – Objective polarity from SentiWordNet.

Note: neg_score + pos_score + obj_score = 1

4.6 DepTree

class loacore.classes.classes.**DepTree**(*id_dep_tree*, *id_dep_tree_node*, *id_sentence*)

Variables

- **id_dep_tree** (*int*) – Id_Dep_Tree used in DepTree table
- **id_dep_tree_node** (*int*) – SQL reference to root node (Dep_Tree_Node table)
- **id_sentence** (*int*) – SQL reference to the corresponding Sentence
- **root** (*DepTreeNode*) – Root node

dep_tree_str(*root=None*, *colored_polarity=False*)

Parameters

- **root** (*DepTreeNode*) – If set, node from which to start to print the tree. self.root otherwise.
- **colored_polarity** (*boolean*) – If True, polarities are colored printed. (red = Negative, green = Positive, yellow = Objective, black = No Synset)

Returns String representation of DepTree instance

Return type string

4.7 DepTreeNode

class loacore.classes.classes.**DepTreeNode**(*id_dep_tree_node*, *id_dep_tree*, *id_word*, *label*, *root*)

Variables

- **id_dep_tree_node** (*int*) – ID_Dep_Tree_Node used in Dep_Tree_Node table
- **id_dep_tree** (*int*) – SQL reference to the corresponding DepTree
- **id_word** (*int*) – SQL reference to corresponding id_word
- **word** (*Word*) – Possibly loaded associated word
- **label** (*string*) – Node dependency label. See annex for details.
- **root** (*boolean*) – True if and only if this is the root of the corresponding DepTree

- **children** (list of *DepTreeNode*) – Node children

4.8 Polarity

class loacore.classes.classes.**Polarity**(*id_polarity*, *analysis*, *id_review*, *pos_score*, *neg_score*, *obj_score*)

Object used to store possible polarities of a review.

Variables

- **id_polarity** (*int*) – ID_Polarity used in Polarity table.
- **analysis** (*string*) – An analysis identifier, from which the polarity come from.
- **id_review** (*int*) – SQL reference to the corresponding Review
- **pos_score** (*float*) – Positive polarity
- **neg_score** (*float*) – Negative polarity
- **obj_score** (*float*) – Objective polarity

is_negative()

Returns True if pos_score > neg_score

Return type boolean

is_objective()

Returns True if pos_score > neg_score

Return type boolean

is_positive()

Returns True if pos_score > neg_score

Return type boolean

CHAPTER

FIVE

FEEDING DATABASE : *PROCESS* PACKAGE

This package contains all the necessary modules to perform the process of new files. Notice that all the processes are automatically handled by the `file_process.add_files()` function.

5.1 Raw Processes

5.1.1 Normalization and review splitting

- Normalization : conversion to UTF-8 and lower case
- Review splitting : the file text is splitted into reviews

`loacore.process.review_process.`**`add_reviews_from_files`**(*files*, *encoding*)

Load argument files from file system and normalize their content.

Compute Reviews objects and add them to the database.

Note: This function should be used only inside the *`add_files()`* function.

Parameters

- **`files`** (`list` of *`File`*) – Files to process
- **`encoding`** (*`str`*) – Encoding used to load files.

Returns added reviews

Return type `list` of *`Review`*

`loacore.process.review_process.`**`normalize`**(*text*)

Performs raw text normalization.

- Conversion to lower case
- Review splitting using python regular expressions : each new line correspond to a new review

Parameters **`text`** (*`str`*) – text to process

Returns reviews

Return type `list` of `str`

5.2 Freeling Processes

5.2.1 tokenization

loacore.process.sentence_process.**add_sentences_from_reviews**(*reviews*, *_state_queue=None*, *_id_process=None*, *freeling_modules=None*)

Performs the first Freeling process applied to each normalized review.

Each review is tokenized, and then splitted into sentences, thanks to corresponding Freeling modules.

A representation of the Sentences and their Words (tokens) are then added to corresponding tables.

Note: This function should be used only inside the `file_process.add_files()` function.

Parameters **reviews** (`list` of *Review*) – Reviews to process

Returns added sentences

Return type `list` of *Sentence*

5.2.2 lemmatization

loacore.process.lemma_process.**add_lemmas_to_sentences**(*sentences*, *print_lemmas=False*, *_state_queue=None*, *_id_process=None*, *freeling_modules=None*)

Performs a Freeling process to add lemmas to words.

However, the argument is actually a sentence to better fit Freeling usage.

Our sentences will be converted to a Freeling Sentences before processing.

Note: This function should be used only inside the `file_process.add_files()` function.

Parameters

- **sentences** (`list` of *Sentence*) – Sentences to process
- **print_lemmas** (*boolean*) – If True, print lemmatization results

5.2.3 disambiguation

loacore.process.synset_process.**add_polarity_to_synsets**(*id_words*, *_state_queue=None*, *_id_process=None*)

Adds the positive/negative/objective polarities of all the synsets currently in the table Synset, from the SentiWordNet corpus.

Note: This function should be used only inside the `file_process.add_files()` function.

loacore.process.synset_process.**add_synsets_to_sentences**(*sentences*, *print_synsets=False*, *_state_queue=None*, *_id_process=None*, *freeling_modules=None*)

Performs a Freeling process to disambiguate words of the sentences according to their context (UKB algorithm) linking them to a unique synset (if possible).

Our sentences are converted to Freeling Sentences before processing.

Notice that even if we may have already computed the Lemmas for example, Freeling Sentences generated from our sentences are "raw sentences", without any analysis linked to their Words. So we make all the Freeling process from scratch every time, except *tokenization* and *sentence splitting*, to avoid any confusion.

Note: This function should be used only inside the file_process.add_files() function.

Parameters

- **sentences** (`list` of *Sentence*) – Sentences to process
- **print_synsets** (*boolean*) – If True, print disambiguation results

5.2.4 dependency tree generation

loacore.process.deptree_process.**add_dep_tree_from_sentences**(*sentences*, *print_result=False*, *_state_queue=None*, *_id_process=None*, *freeling_modules=None*)

Generates the dependency trees of the specified sentences and add the results to the database.

Sentences are firstly converted into "raw" Freeling sentences (without any analysis) and then all the necessary Freeling processes are performed.

The PoS_tag of words are also computed and added to the database in this function.

Note: This function should be used only inside the `file_process.add_files()` function.

Note: This process can be quite long. (at least a few minutes)

Parameters

- **sentences** (`list` of *Sentence*) – Sentences to process
- **print_result** (*boolean*) – Print PoS_tags and labels associated to each *Word*

5.3 Feed database

loacore.process.file_process.**add_files**(*file_paths*, *encoding='utf8'*, *lang=''*, *workers=1*)

This function performs the full process on all the file_paths specified, and add the results to the corresponding tables.

Parameters

- **file_paths** (list of path-like object) – Paths used to load files
- **encoding** (*str*) – Files encoding.
- **lang** (*str*) – If specify, *lang* will be used as Freeling language. Otherwise, default language is used (See loacore)

  Possible values : 'as', 'ca', 'cs', 'cy', 'de', 'en', 'es', 'fr', 'gl', 'hr', 'it', 'nb', 'pt', 'ru', 'sl')

  See https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/basics.html for more details.
- **workers** (*int*) – Number of workers used to perform Freeling processes.

  If *workers* <= 0, Multiprocessing is not used and all the programm is run in a unique process.

  if *workers* > 0, *workers* processes are created in addition of the main process. This is useful to take advantage of multi-core architectures to greatly speed up the process.

  Note: To control the state of each process when *workers* > 0, a Python curses app is opened. Notice that seems to work badly in PyCharm terminal.

Example

Process and load file from the relative directory *data/raw/*

```
file_paths = []
for dirpath, dirnames, filenames in os.walk(os.path.join('data', 'raw')):
    for name in filenames:
        file_paths.append(os.path.join(dirpath, name))

file_process.add_files(file_paths)
```

CHAPTER

SIX

LOAD DATA FROM DATABASE : *LOAD* PACKAGE

6.1 Load Files

loacore.load.file_load.**clean_db**()
Remove all files from database. Implemented references will also engender the deletion of all files dependencies in database : all the tables will be emptied.

loacore.load.file_load.**get_id_files_by_file_path**(*file_path_re*)
This function can be used to retrieve file ids in database from their path.

For more information about how Python regular expressions work, see https://docs.python.org/3/library/re.html .

Parameters **file_path_re** (str) – Regular expression to check

Returns Ids of matching files.

Return type list of int

Example

Find if files of files in an uci folder.

```
>>> import loacore.load.file_load as file_load
>>> ids = file_load.get_id_files_by_file_path(r'.*/uci/.+')
>>> print(ids)
[1, 2, 3]
```

Note: The full path of a file (as saved in the database) can be used as a regular expression.

loacore.load.file_load.**load_database**(*id_files=[]*, *load_reviews=True*, *load_polarities=True*, *load_sentences=True*, *load_words=True*, *load_deptrees=True*, *workers=1*)
Load the complete database as a list of *File* , with all the dependencies specified in parameters loaded in them.

Parameters

- **id_files** (list of int) – If specified, load only the files with the corresponding ids. Otherwise, load all the files.
- **load_reviews** (*bool*) – Specify if Reviews need to be loaded if files.
- **load_polarities** (*bool*) – If Reviews have been loaded, specify if Polarities need to be loaded in reviews.

- **load_sentences** (*bool*) – If Reviews have been loaded, specify if Sentences need to be loaded in reviews.
- **load_words** (*bool*) – If Sentences have been loaded, specify if Words need to be loaded in sentences.
- **load_deptrees** (*bool*) – If Words have been loaded, specify if DepTrees need to be loaded in sentences.
- **workers** (*int*) – Number of workers used to load database.

  If *workers* <= 0, Multiprocessing is not used and all the program is run in a unique process.

  if *workers* > 0, *workers* processes are created in addition of the main process. This is useful to take advantage of multi-core architectures to greatly speed up the process.

Returns loaded files

Return type list of *File*

Note: Among the dependencies, only the load_deptrees should be set to False to significantly reduce processing time if they are not needed. Loading other structures is quite fast.

Example Load files 1,2,3 with only their id_file and id_path.

```
>>> import loacore.load.file_load as file_load
>>> files = file_load.load_database(id_files=[1, 2, 3], load_
↪reviews=False)
>>> print([f.file_path for f in files])
['../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_EO.txt',
'../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_CC.txt',
'../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_GR.txt']
```

Example Load the first 10 files without Dep Trees.

```
>>> import loacore.load.file_load as file_load
>>> files = load_database(id_files=range(1, 11), load_deptrees=False)
>>> print(files[3].reviews[8].review)
que sea mas grande el parqueadero
```

Example Load the complete database.

```
>>> import loacore.load.file_load as file_load
>>> files = load_database()
>>> print(len(files))
33
```

loacore.load.file_load.**remove_files**(*files*)

Remove specified files from database. Implemented references will also engender the deletion of all files dependencies in database.

Parameters **files** – list of *File*

6.2 Load Reviews

loacore.load.review_load.**load_reviews**(*id_reviews=[]*, *load_polarities=False*, *load_sentences=False*, *load_words=False*, *load_deptrees=False*)

Load reviews from database.

Parameters

- **id_reviews** (list of int) – If specified, load only the reviews with corresponding ids. Otherwise, load all the reviews.
- **load_sentences** (*boolean*) – Specify if Sentences need to be loaded in reviews.
- **load_words** (*boolean*) – If Sentences have been loaded, specify if Words need to be loaded in sentences.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in sentences.

Returns Loaded reviews

Return type list of *Review*

Example Load all reviews with sentences and words

```
>>> import loacore.load.review_load as review_load
>>> reviews = review_load.load_reviews(load_sentences=True, load_
↪words=True)
>>> reviews[0].sentences[0].sentence_str(print_sentence=False)
'teleferico'
```

loacore.load.review_load.**load_reviews_by_id_files**(*id_files*, *load_polarities=False*, *load_sentences=False*, *load_words=False*, *load_deptrees=False*)

Load reviews of files specified by their ids.

Parameters

- **id_files** (list of int) – Ids of files from which reviews should be loaded.
- **load_sentences** (*boolean*) – Specify if Sentences need to be loaded in reviews.
- **load_words** (*boolean*) – If Sentences have been loaded, specify if Words need to be loaded in sentences.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in sentences.

Returns Loaded reviews

Return type list of *Review*

Example Load reviews from the first file as "raw" reviews, without sentences.

```
>>> import loacore.load.review_load as review_load
>>> reviews = review_load.load_reviews_by_id_files([1])
>>> print(reviews[0].review)
teleferico
```

loacore.load.review_load.**load_reviews_in_files**(*files*, *load_sentences=False*, *load_words=False*, *load_deptrees=False*)

Load reviews into corresponding *files*, setting up their attribute `reviews`.

Also return all the loaded reviews.

Note: This function is automatically called by `file_load.load_database()` when *load_reviews* is set to `True`. In most of the cases, this function should be used to load files and reviews in one go.

Parameters

- **files** (`list` of *File*) – Files in which corresponding reviews will be loaded.
- **load_sentences** (*boolean*) – Specify if Sentences need to be loaded in reviews.
- **load_words** (*boolean*) – If Sentences have been loaded, specify if Words need to be loaded in sentences.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in sentences.

Returns Loaded reviews

Return type `list` of *Review*

6.3 Load Sentences

loacore.load.sentence_load.**load_sentences**(*id_sentences=[]*, *load_words=False*, *load_deptrees=False*)

Load sentences from database.

Parameters

- **id_sentences** (`list` of `int`) – If specified, load only the sentences with corresponding ids. Otherwise, load all the sentences.
- **load_words** (*boolean*) – Specify if Words need to be loaded in sentences.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in sentences.

Returns Loaded sentences

Return type `list` of *Sentence*

Example Load sentences 1,2 and their words.

```
>>> import loacore.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences([1,2], load_words=True)
>>> sentences[0].sentence_str(print_sentence=False)
'teleferico'
>>> sentences[1].sentence_str(print_sentence=False)
'toboganvy que el agua huela a asufre'
```

loacore.load.sentence_load.**load_sentences_by_id_files**(*id_files*, *load_words=True*, *load_deptrees=True*)

Ids of files from which sentences should be loaded.

Parameters

- **id_files** (list of int) – Ids of files from which reviews should be loaded.
- **load_words** (*boolean*) – Specify if Words need to be loaded in sentences.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in sentences.

Returns Loaded sentences

Return type list of *Sentence*

Example

Load all the sentences from file 1.

```
>>> import loacore.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([1])
>>> sentences[0].sentence_str(print_sentence=False)
'teleferico'
```

loacore.load.sentence_load.**load_sentences_in_reviews**(*reviews*, *load_words=False*, *load_deptrees=False*)

Load sentences into corresponding *reviews*, setting up their attribute sentences.

Also return all the loaded sentences.

Note: This function is automatically called by file_load.load_database() or review_load.load_reviews() when *load_sentences* is set to True. In most of the cases, those functions should be used instead to load reviews and sentences in one go.

Parameters

- **reviews** (list of *Review*) – Reviews in which corresponding sentences should be loaded.
- **load_words** (*boolean*) – Specify if Words need to be loaded in sentences.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in sentences.

Returns Loaded sentences

Return type list of *Sentence*

6.4 Load Words

loacore.load.word_load.**load_words**(*id_words=[]*, *load_lemmas=True*, *load_synsets=True*)

Load words from database.

Parameters

- **id_words** (list of int) – If specified, load only the words with corresponding ids. Otherwise, load all the words.
- **load_lemmas** (*boolean*) – Specify if Lemmas need to be loaded in words.
- **load_synsets** (*boolean*) – Specify if Synsets need to be loaded in words.

Returns loaded words

Return type list of *Word*

Example Load all words and their lemmas, synsets.

```
>>> import loacore.load.word_load as word_load
>>> words = word_load.load_words()
>>> print([w.word for w in words[0:11]])
['teleferico', 'toboganvy', 'que', 'el', 'agua', 'huela', 'a', 'asufre
↪', 'pista', 'de', 'baile']
>>> print([w.lemma for w in words[0:11]])
['', '', 'que', 'el', 'agua', 'oler', 'a', '', 'pista', 'de', 'bailar
↪']
```

loacore.load.word_load.**load_words_in_dep_trees**(*dep_trees*, *load_lemmas=True*, *load_synsets=True*)

Load words into corresponding *dep_trees*, setting up the attribute word of each node.

Note: This function is automatically called by file_load.load_database() when *load_deptrees* is set to True, or by dep_tree.load_deptrees() when *load_words* is set to True. In most of the cases, those functions should be used instead to load dep_trees and words in one go.

Parameters

- **dep_trees** (list of *DepTree*) – DepTrees in which corresponding words should be loaded.
- **load_lemmas** (*boolean*) – Specify if Lemmas need to be loaded in words.
- **load_synsets** (*boolean*) – Specify if Synsets need to be loaded in words.

loacore.load.word_load.**load_words_in_sentences**(*sentences*, *load_lemmas=True*, *load_synsets=True*)

Load words into corresponding *sentences*, setting up their attribute words.

Also return all the loaded words.

Note: This function is automatically called by file_load.load_database() or sentence_load.load_sentences() when *load_words* is set to True. In most of the cases, those functions should be used instead to load sentences and words in one go.

Parameters

- **sentences** (list of *Sentence*) – Sentences in which corresponding words should be loaded.
- **load_lemmas** (*boolean*) – Specify if Lemmas need to be loaded in words.
- **load_synsets** (*boolean*) – Specify if Synsets need to be loaded in words.

Returns loaded words

Return type list of *Word*

6.5 Load Synsets

loacore.load.synset_load.**load_synsets**(*id_synsets=None*)

Load *Synset* s from database.

Parameters **id_synsets** (list of *Word*) – If specified, load only the synsets with corresponding ids. Otherwise, load all the synsets.

Returns loaded synsets

Return type list of *Synset*

Example

Load all synsets from database.

```
>>> import loacore.load.synset_load as synset_load
>>> synsets = synset_load.load_synsets()
>>> print(synsets[0].synset_code)
14845743-n
>>> print(synsets[0].synset_name)
water.n.01
```

loacore.load.synset_load.**load_synsets_in_words**(*words*)

Load *Synset* s into corresponding *words*, setting up their attribute synset.

Also return all the loaded synsets.

Note: This function is automatically called by file_load.load_database() when *load_words* is set to True or by word_load.load_words() when *load_synsets* is set to True. In most of the cases, those functions should be used instead to load words and synsets in one go.

Parameters **words** (list of *Word*) – Words in which corresponding synsets should be loaded.

Returns loaded synsets

Return type list of *Synset*

6.6 Load Lemmas

loacore.load.lemma_load.**load_lemmas**(*id_lemmas=[]*)

Load lemmas from database.

Parameters **id_lemmas** (list of int) – If specified, load only the lemmas with corresponding ids. Otherwise, load all the lemmas.

Returns loaded lemmas

Return type list of str

Example

Load all lemmas from database.

```
>>> import loacore.load.lemma_load as lemma_load
>>> lemmas = lemma_load.load_lemmas()
>>> print(len(lemmas))
```

(continues on next page)

(continued from previous page)

```
103827
>>> print(lemmas[10])
bailar
```

loacore.load.lemma_load.**load_lemmas_in_words**(*words*)

Load lemmas into corresponding *words*, setting up their attribute `lemma`.

Also return all the loaded lemmas.

Note: This function is automatically called by `file_load.load_database()` when *load_words* is set to `True` or by `word_load.load_words()` when *load_synsets* is set to `True`. In most of the cases, those functions should be used instead to load words and synsets in one go.

Parameters **words** (`list` of *Word*) – Words in which corresponding synsets should be loaded.

Returns loaded lemmas

Return type `list` of `str`

6.7 Load DepTrees

loacore.load.deptree_load.**load_dep_tree_in_sentences**(*sentences*, *load_words=True*)

Load Dep Trees into corresponding *sentences*, setting up their attribute `dep_tree`.

Also return all the loaded deptrees.

Note: This function is automatically called by `file_load.load_database()` or `sentence_load.load_sentences()` when *load_deptrees* is set to `True`. In most of the cases, those functions should be used instead to load sentences and deptrees in one go.

Parameters

- **sentences** (`list` of *Sentence*) – Sentences in which corresponding DepTrees should be loaded.
- **load_words** (*boolean*) – Specify if Words need to be loaded in Dep Trees.

Returns loaded deptrees

Return type `list` of *DepTree*

loacore.load.deptree_load.**load_dep_trees**(*id_dep_trees=[]*, *load_words=True*)

Load Dep Trees from database.

Parameters

- **id_dep_trees** (`list` of `int`) – If specified, load only the deptrees with corresponding ids. Otherwise, load all the deptrees.
- **load_words** (*boolean*) – Specify if Words need to be loaded in Dep Trees.

Returns loaded deptrees

Return type `list` of *DepTree*

Example

Load all deptrees from database : can take a few moments.

```
>>> import loacore.load.deptree_load as deptree_load
>>> deptrees = deptree_load.load_dep_trees()
>>> deptree_str = deptrees[500].dep_tree_str()
instalaciones (sentence, NCFP000, instalación)
    las (spec, None, el)
    agua (sn, NCCS000, agua)
        el (spec, None, el)
        fria (s.a, None, )
            y (coord, None, y)
            caliente (grup.a, AQ0CS00, calentar)
        caminata (sn, NCFS000, caminata)
            la (spec, None, el)
        tranquilidad (sn, NCFS000, tranquilidad)
            la (spec, None, el)
        servicio (sn, NCMS000, servicio)
            el (spec, None, el)
```

CHAPTER

SEVEN

ANALYSE DATA : *ANALYSIS* PACKAGE

7.1 Sentiment Analysis

loacore.analysis.sentiment_analysis.**compute_extreme_files_polarity**(*files*, *pessimistic=False*, *freeling_lang='es'*)

Compute the extreme polarity of all the reviews in each file, and then compute the normalized sum of polarities of all reviews of each file, and return them as a dictionary that map id_files to polarity tuples (pos_score, neg_score, obj_score).

Parameters

- **files** (list of *File*) – Files to process
- **pessimistic** (*boolean*) – Specify if pessimistic computing should be used. Optimistic is used if set to False.

Returns Score dictionary

Return type dict of int : tuple

Example Compute optimistic and pessimistic polarities of uci files.

```
>>> import loacore.load.file_load as file_load
>>> import loacore.analysis.sentiment_analysis as sentiment_analysis
>>> from loacore.utils import plot_polarities
>>> ids = file_load.get_id_files_by_file_paths([r'.*/uci/.+'])
>>> files = file_load.load_database(id_files=ids, load_deptrees=False)
>>> polarities = sentiment_analysis.compute_extreme_files_
↪polarity(files, freeling_lang='en')
>>> plot_polarities.print_polarity_table(polarities)
+--------------------------+-----------+-----------+-----------+
|           File           | Pos_Score | Neg_Score | Obj_Score |
+--------------------------+-----------+-----------+-----------+
|     imdb_labelled.txt    |   0.467   |   0.066   |   0.467   |
|     yelp_labelled.txt    |   0.465   |   0.069   |   0.465   |
| amazon_cells_labelled.txt |   0.462   |   0.076   |   0.462   |
+--------------------------+-----------+-----------+-----------+
>>> polarities = sentiment_analysis.compute_extreme_files_
↪polarity(files, pessimistic=True, freeling_lang='en')
>>> plot_polarities.print_polarity_table(polarities)
+--------------------------+-----------+-----------+-----------+
|           File           | Pos_Score | Neg_Score | Obj_Score |
+--------------------------+-----------+-----------+-----------+
```

(continues on next page)

(continued from previous page)

```
|     imdb_labelled.txt     |   0.091   |   0.819   |   0.091   |
|     yelp_labelled.txt     |   0.056   |   0.887   |   0.056   |
| amazon_cells_labelled.txt |   0.049   |   0.901   |   0.049   |
+---------------------------+-----------+-----------+-----------+
```

loacore.analysis.sentiment_analysis.**compute_extreme_reviews_polarity**(*reviews*, *commit_polarities=False*, *pessimistic=False*, *freeling_lang='es'*)

Performs *extreme* reviews polarity computation : only the most pessimistic or optimistic sense (according to pessimistic argument) is considered. Those values tend to show how the disambiguation process is important, due to the huge difference between pessimistic and optimistic scores.

Also notice that this function is an interesting example of how other processes could be applied to data already computed in database. Here, the disambiguated synsets are not used, and all senses re-computed with Freeling are used.

Check source code for more detailed explanations about this example.

Polarities are then stored in reviews, with the analysis label 'optimistic' or 'pessimistic' according to *pessimistic* parameter, and eventually committed to database if *commit_polarities* is set to True.

Parameters

- **reviews** (list of *Review*) – Reviews to process
- **commit_polarities** (*boolean*) – If set to True, computed polarities are commited to database.
- **pessimistic** (*boolean*) – Specify if pessimistic computing should be used. Optimistic is used if set to False.
- **freeling_lang** (*String*) – Specify language used by Freeling. Default : 'es'.

  Possible values : 'as', 'ca', 'cs', 'cy', 'de', 'en', 'es', 'fr', 'gl', 'hr', 'it', 'nb', 'pt', 'ru', 'sl')

  See https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/basics.html for more details.

Example Commit pessimistic polarities of uci files to database.

```
import loacore.analysis.sentiment_analysis as sentiment_analysis
import loacore.load.file_load as file_load
import loacore.load.review_load as review_load

ids = file_load.get_id_files_by_file_paths([r'.*/uci/.+'])
reviews = review_load.load_reviews_by_id_files(id_files=ids, load_
↪sentences=True, load_words=True)
sentiment_analysis.compute_extreme_reviews_polarity(reviews, commit_
↪polarities=True, pessimistic=True,
                                                    freeling_lang='en
↪')
```

loacore.analysis.sentiment_analysis.**compute_simple_files_polarity**(*files*)

Compute the simple polarity of all the reviews in each file, and then compute the normalized sum of polarities of all reviews of each file, and return them as a dictionary that map id_files to polarity tuples (pos_score, neg_score, obj_score).

Parameters **files** (list of *File*) – Files to process

Returns Polarity dict

Return type dict of int : tuple

Example Load uci files, compute basic polarities, and show results with utils.print_polarity_table().

```
>>> import loacore.load.file_load as file_load
>>> import loacore.analysis.sentiment_analysis as sentiment_analysis
>>> ids = file_load.get_id_files_by_file_paths([r'.*/uci/.+'])
>>> files = file_load.load_database(id_files=ids, load_deptrees=False)
>>> polarities = sentiment_analysis.compute_simple_files_
↪polarity(files)
>>> from loacore.utils import plot_polarities
>>> plot_polarities.print_polarity_table(polarities)
+--------------------------+-----------+-----------+-----------+
|           File           | Pos_Score | Neg_Score | Obj_Score |
+--------------------------+-----------+-----------+-----------+
|     imdb_labelled.txt    |   0.117   |   0.088   |   0.795   |
|     yelp_labelled.txt    |   0.123   |   0.086   |   0.790   |
| amazon_cells_labelled.txt |   0.118   |   0.090   |   0.792   |
+--------------------------+-----------+-----------+-----------+
```

loacore.analysis.sentiment_analysis.**compute_simple_reviews_polarity**(*reviews*, *commit_polarities=False*)

Perform the easiest sentiment analysis possible : a normalized sum of the positive/negative/objective polarities available in all synsets of each review, setting the polarity in the review.polarities dict, with the entry "simple".

Parameters **reviews** (list of *Review*) – Files to process

Example Commit simple polarities of uci files to database.

```
import loacore.analysis.sentiment_analysis as sentiment_analysis
import loacore.load.file_load as file_load
import loacore.load.review_load as review_load

ids = file_load.get_id_files_by_file_paths([r'.*/uci/.+'])
reviews = review_load.load_reviews_by_id_files(id_files=ids, load_
↪sentences=True, load_words=True)
sentiment_analysis.compute_simple_reviews_polarity(reviews, commit_
↪polarities=True)
```

7.2 Pattern Recognition

Patterns recognitions are realized on the dependency trees computed with Freeling. This means that *parent-child* structures will be matched, what **don't necessarily correspond to adjacent words in the original sentence**.

loacore.analysis.pattern_recognition.**adj_pattern_table**(*sentences*, *polarity_commuter_only=True*, *lang='en'*)

Return a string table with the following fiels :

- Noun : word with a Noun PoS_tag, with an associated adjective
- N_Polarity + : noun positive polarity

- N_Polatity - : noun negative polarity
- Adj : word with an Adjective PoS_tag, associated to the Noun according to the sentence dependency tree
- A_Polarity + : adjective positive polarity
- A_Polarity - : adjective negative polarity

Only the words associated to a synset are considered.

Parameters

- **sentences** (list of *Sentence*) – Sentences to process
- **polarity_commuter_only** (*boolean*) – If True, only the patterns that are subject to a polarity commutation are displayed, i.e. those constituted by a positive noun associated to a negative adjective.
- **lang** (*string*) – Language used. (PoS_tags and dependency labels to match are chosen accordingly) Possible values : 'en', 'es'

Returns Table as a str

Return type string

Example Load Spanish files from corrected folder (previously added to database) and print their adj pattern table, with polarity commutation only.

```
>>> import loacore.load.file_load as file_load
>>> import loacore.analysis.pattern_recognition as pattern_recognition
>>> ids = file_load.get_id_files_by_file_paths([r'.*/corrected/.+'])
>>> files = file_load.load_database(id_files=ids)
>>> for file in files:
...     print(pattern_recognition.adj_pattern_table(file.sentence_
↪list(), lang='es'))
...
```

loacore.analysis.pattern_recognition.**general_pattern_recognition**(*sentences*, *pattern*, *types*)

Recognize a general pattern, compound of PoS_tags and dependency labels, in the DepTrees associated to specified sentences.

Parameters

- **sentences** (list of *Sentence*) – Sentences to process
- **pattern** (list of list of str) – A 2 dimensional list of strings representing patterns. The patterns list pattern[i] represents the label that will match at position i. ex : *pattern = [['V'], ['cc', 'ci', 'cd']]* will match all the *Verb/complement* structures.
- **types** (list of str. Allowed value are 'PoS_tag' and 'label'. Otherwise, nothing will match.) – Specify what type of match to use, such that *types[i]* specifies if elements of *pattern[i]* have to be condidered as PoS_tag or label. Notice that types is unidimensional, whereas pattern can be 2 dimensional : this means that for consistency reason, we assume that all the tags that can match in a position *i* are of the same nature.

Returns Matching patterns in specified sentences, as node tuples.

Return type list of tuple of *DepTreeNode*

Example Find all the Verb(PoS_tag)/complement(label) patterns in file 28(_PQRS.txt).

(classically, a negation that applies to the parent verb)

```
>>> import loacore.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import loacore.analysis.pattern_recognition as pattern_recognition
>>> patterns = pattern_recognition.general_pattern_
↪recognition(sentences, [['V'], ['cc', 'ci', 'cc']], ['PoS_tag',
↪'label'])
>>> patterns_str = pattern_recognition.print_patterns(patterns, PoS_
↪tag_display=True, label_display=True)
( parece : VMIP3S0 : sentence, me : None : ci )
( promueven : VMIP3P0 : S, en : None : cc )
( atiende : VMIP3S0 : S, de : None : cc )
( atiende : VMIP3S0 : S, como : None : cc )
( atiendan : VMSP3P0 : S, de : None : cc )
( atiende : VMIP3S0 : S, en : None : cc )
( viniera : VMSI3S0 : S, con : None : cc )
( orientar : VMN0000 : S, en : None : cc )
( orientar : VMN0000 : S, al : None : cc )
( poner : VMN0000 : S, le : None : ci )
( poner : VMN0000 : S, a : None : ci )
( establecer : VMN0000 : S, uun : None : cc )
...
```

loacore.analysis.pattern_recognition.**label_patterns_recognition**(*sentences*, *pattern*)

Recognize a dependency label pattern in the DepTrees associated to specified sentences.

Labels used for Spanish can be found there :

- http://clic.ub.edu/corpus/webfm_send/20
- http://clic.ub.edu/corpus/webfm_send/18
- http://clic.ub.edu/corpus/webfm_send/49

Parameters

- **sentences** (list of *Sentence*) – Sentences to process
- **pattern** (list of list of str) – A 2 dimensional list of strings representing patterns. The patterns list pattern[i] represents the label that will match at position i. ex : *pattern = [['sentence', 'v'], ['']]** could be used to find all the dependency functions that could follow *sentence* of *v* function.

Returns Matching patterns in specified sentences, as node tuples.

Return type list of tuple of *DepTreeNode*

Example Find all node to which a verbal modifier is applied in file 28 (_PQRS.txt).

(classically, a negation that applies to the parent verb)

```
>>> import loacore.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import loacore.analysis.pattern_recognition as pattern_recognition
>>> patterns = pattern_recognition.label_patterns_
↪recognition(sentences, [['*'], ['mod']])
>>> patterns_str = pattern_recognition.print_patterns(patterns, label_
↪display=True)
( bajar : ao, ya : mod )
( bajar : ao, no : mod )
( podia : S, tampoco : mod )
```

(continues on next page)

(continued from previous page)

```
( quiere : ao, no : mod )
( dejan : S, no : mod )
...
```

Note: This function can also be used to recognize unigram patterns.

Example : Find all the nodes with dependency label 'suj' in file 28 (_PQRS.txt)

```
>>> import loacore.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import loacore.analysis.pattern_recognition as pattern_recognition
>>> patterns = pattern_recognition.label_patterns_recognition(sentences,␣
↪[['suj']])
>>> patterns_str = pattern_recognition.print_patterns(patterns, PoS_tag_
↪display=True, label_display=True)
( colmo : NCMS000 : suj )
( que : None : suj )
( que : None : suj )
( tencion : None : suj )
( señora : NCFS000 : suj )
( que : None : suj )
( turista : NCCS000 : suj )
( ella : None : suj )
( que : None : suj )
...
```

---

loacore.analysis.pattern_recognition.**pos_tag_patterns_recognition**(*sentences*, *pattern*)

Recognize a PoS_tag pattern in the DepTrees associated to specified sentences.

PoS_tags corresponding to each language can be found there : https://talp-upc.gitbooks.io/freeling-4-1-user-manual/content/tagsets.html

Parameters

- **sentences** (list of *Sentence*) – Sentences to process
- **pattern** (list of list of str) – A 2 dimensional list of strings representing patterns. The patterns list pattern[i] represents the PoS_tags that will match at position i. ex : *pattern = [['V'], ['A', 'NC']]* recognizes verbs followed by an adjective or a common noun.

  Note: Matches are performed with the beginning of the PoS_tag, according to the length of the specified tags. For example, 'A' will match 'AQ0CS00', 'AQ0MS00'...

Returns Matching patterns in specified sentences, as node tuples.

Return type list of tuple of *DepTreeNode*

Example Find all Noun/Adjective patterns in file 28 (_PQRS.txt).

```
>>> import loacore.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import loacore.analysis.pattern_recognition as pattern_recognition
>>> patterns = pattern_recognition.pos_tag_patterns_
↪recognition(sentences, [['N'], ['A']])
```

(continues on next page)

(continued from previous page)

```
>>> patterns_str = pattern_recognition.print_patterns(patterns, PoS_
↪tag_display=True)
( manera : NCFS000, grosera : AQ0FS00 )
( manera : NCFS000, igual : AQ0CS00 )
( señora : NCFS000, irrespetuosa : AQ0FS00 )
( zona : NCFS000, visible : AQ0CS00 )
( manera : NCFS000, grosera : AQ0FS00 )
( espacio : NCMS000, mejor : AQ0CS00 )
( atencion : NCFS000, mejor : AQ0CS00 )
...
```

Note: This function can also be used to recognize unigram patterns.

Example : Find all verbs in file 28 (_PQRS.txt)

```
>>> import loacore.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import loacore.analysis.pattern_recognition as pattern_recognition
>>> patterns = pattern_recognition.pos_tag_patterns_recognition(sentences,
↪ [['V']])
>>> patterns_str = pattern_recognition.print_patterns(patterns, PoS_tag_
↪display=True, label_display=True)
( parece : VMIP3S0 : sentence )
( promueven : VMIP3P0 : S )
( atiende : VMIP3S0 : S )
( atiendan : VMSP3P0 : S )
( atiende : VMIP3S0 : S )
( viniera : VMSI3S0 : S )
( orientar : VMN0000 : S )
( contratar : VMN0000 : S )
( era : VSII3S0 : sentence )
( argumentando : VMG0000 : gerundi )
( poner : VMN0000 : S )
...
```

---

loacore.analysis.pattern_recognition.**verb_context_table**(*sentences*, *context_length=2*)

Return a string table with the following fields :

- Verb : a word with a verb PoS_tag
- Context : the context of the words. *context_length* defines the number of words to display before and after the verb.
- synset : name of the synset associated to the verb
- polarity + : positive polarity
- polarity - : negative polarity
- polarity = : objective polarity

Parameters

- **sentences** (list of *Sentence*) – Sentences to process
- **context_length** (*int*) – Number of words to display before and after the verb.

Returns Table as a `str`

Return type string

Example Load files from uci and corrected folders (after they have been added to database) and print their verb context tables.

```
>>> import loacore.load.file_load as file_load
>>> import loacore.analysis.pattern_recognition as pattern_recognition
>>> ids = file_load.get_id_files_by_file_paths([r'.*/uci/.+', r'.*/
↪corrected/.+'])
>>> files = file_load.load_database(id_files=ids, load_deptrees=False)
>>> for file in files:
...     print(pattern_recognition.verb_context_table(file.sentence_
↪list()))
...
```

7.3 Frequencies

This module provides functions to compute simple and bigram dependency labels frequencies. The objective of such analysis could be to identify relevant structures among the most commons.

`loacore.analysis.frequencies.`**`bigram_label_frequencies`**(*files*)

Compute bigram dependency label frequencies if files.

A "bigram label" is defined as a tuple constituted by the label of a parent node, and the label of one of its children.

Results are returned in a dictionary that map file names to frequencies.

Frequencies are represented has dictionaries that map dependency labels to frequencies.

A list of dependency labels is also returns.

Parameters **`files`** (`list` of *`File`*) – Files to process

Returns Dependency labels, and a dictionary that maps file names to frequencies

Return type `list` of `str` , `dict` of `str` : `dict` of `str` : `float`

Example Compute bigram label frequencies of uci files.

`loacore.analysis.frequencies.`**`bigram_pos_tag_frequencies`**(*files*, *tag_len=2*)

Same idea as :func:'label_frequencies', but with Part Of Speech tags.

The number of characters of the PoS_tags to consider can be specified with tag_len.

Parameters

- **`files`** (`list` of *`File`*) – Files to process
- **`tag_len`** – Number of letters kept in each PoS_tag
- **`tag_len`** – int

Returns PoS_tags, and a dictionary that maps file names to frequencies

Return type `list` of `str` , `dict` of `str` : `dict` of `str` : `float`

Example Compute bigram PoS_tags frequencies of uci files.

loacore.analysis.frequencies.**count_bigram_label**(*file*, *bigram_label*, *c*)
Count the number of occurrence of *bigram_label* if all the dependency trees of *file*, through an SQL request.

Note: For a more efficient call from *bigram_label_frequencies()*, this function takes an already initialized SQL cursor as an argument. If you want to use it, you can initialize *c* with the following code :

```
import sqlite3 as sql
from loacore.conf import DB_PATH
conn = sql.connect(DB_PATH)
c = conn.cursor()
```

Parameters

- **file** (*File*) – File to process
- **bigram_label** (tuple of str) – Tuple of two dependency labels.
- **c** – SQL cursor

Returns Number of bigram_label occurrences

Return type int

loacore.analysis.frequencies.**count_bigram_pos_tag**(*file*, *bigram_pos_tag*, *c*)
Count the number of occurrence of *bigram_pos_tag* if all the dependency trees of *file*, through an SQL request.

Note: For a more efficient call from *bigram_pos_tag_frequencies()*, this function takes an already initialized SQL cursor as an argument. If you want to use it, you can initialize *c* with the following code :

```
import sqlite3 as sql
from loacore.conf import DB_PATH
conn = sql.connect(DB_PATH)
c = conn.cursor()
```

Parameters

- **file** (*File*) – File to process
- **bigram_pos_tag** (tuple of str) – Tuple of two pos_tags.
- **c** – SQL cursor

Returns Number of bigram_pos_tag occurrences

Return type int

loacore.analysis.frequencies.**count_label**(*file*, *label*, *c*)
Count the number of occurrence of *label* if all the dependency trees of *file*, through an SQL request.

Note: For a more efficient call from *label_frequencies()*, this function takes an already initialized SQL cursor as an argument. If you want to use it, you can initialize *c* with the following code :

```
import sqlite3 as sql
from loacore.conf import DB_PATH
conn = sql.connect(DB_PATH)
c = conn.cursor()
```

Parameters

- **file** (*File*) – File to process
- **label** (*str*) – Dependency label
- **c** – SQL cursor

Returns Number of label occurrences

Return type int

loacore.analysis.frequencies.**count_pos_tag**(*file*, *pos_tag*, *c*)

Count the number of occurrence of *pos_tag* if all the words of *file*, through an SQL request.

Note: For a more efficient call from *pos_tag_frequencies()*, this function takes an already initialized SQL cursor as an argument. If you want to use it, you can initialize *c* with the following code :

```
import sqlite3 as sql
from loacore.conf import DB_PATH
conn = sql.connect(DB_PATH)
c = conn.cursor()
```

Parameters

- **file** (*File*) – File to process
- **pos_tag** (*string*) – Part of Speech tag
- **c** – SQL cursor

Returns Number of pos_tag occurrences

Return type int

loacore.analysis.frequencies.**get_bigram_label_set**(*files*, *c*)

Returns all the existing bigram dependency labels in files, through a SQL request.

A "bigram label" is defined as a tuple constituted by the label of a parent node, and the label of one of its children.

Note: For a more efficient call from *bigram_label_frequencies()*, this function takes an already initialized SQL cursor as an argument. If you want to use it, you can initialize *c* with the following code :

```
import sqlite3 as sql
from loacore.conf import DB_PATH
conn = sql.connect(DB_PATH)
c = conn.cursor()
```

Parameters

- **files** (list of *File*) – Files to process. Notice that only the id_files are needed.
- **c** – SQL cursor

Returns Bigram labels as string tuples.

Return type list of tuple of str

loacore.analysis.frequencies.**get_bigram_pos_tag_set**(*files*, *c*, *tag_len*)
Returns all the existing bigram PoS_tag in files, through a SQL request.

A "bigram PoS_tag" is defined as a tuple constituted by the PoS_tag of a parent node, and the PoS_tag of one of its children.

Note: For a more efficient call from *bigram_pos_tag_frequencies()*, this function takes an already initialized SQL cursor as an argument. If you want to use it, you can initialize *c* with the following code :

```
import sqlite3 as sql
from loacore.conf import DB_PATH
conn = sql.connect(DB_PATH)
c = conn.cursor()
```

Parameters

- **files** (list of *File*) – Files to process. Notice that only the id_files are needed.
- **c** – SQL cursor

Returns Bigram PoS_tags as string tuples.

Return type list of tuple of str

loacore.analysis.frequencies.**get_label_set**(*files*, *c*)
Returns all the existing dependency labels in files, through a SQL request.

Note: For a more efficient call from *label_frequencies()*, this function takes an already initialized SQL cursor as an argument. If you want to use it, you can initialize *c* with the following code :

```
import sqlite3 as sql
from loacore.conf import DB_PATH
conn = sql.connect(DB_PATH)
c = conn.cursor()
```

Parameters

- **files** (list of *File*) – Files to process. Notice that only the id_files are needed.
- **c** – SQL cursor

Returns Dependency labels

Return type list of str

loacore.analysis.frequencies.**get_polarity_label_set**(*files*, *c*, *polarity*)
Select all the possible dependency labels for words in files with the specified polarity, thanks to an SQL request.

Parameters

- **files** (*File*) – File to process
- **c** – SQL cursor
- **polarity** (str : {'positive', 'negative'}) – Word polarity to consider.

Returns Dependency labels

Return type list of str

loacore.analysis.frequencies.**get_polarity_pos_tag_set**(*files*, *c*, *tag_len*, *polarity*)
Select possible Part of Speech tags for words in files with the specified polarity, thanks to an SQL request.

Parameters

- **files** (*File*) – File to process
- **tag_len** – Number of letters kept in each PoS_tag
- **tag_len** – int
- **c** – SQL cursor
- **polarity** (str : {'positive', 'negative'}) – Word polarity to consider.

Returns PoS tags

Return type list of str

loacore.analysis.frequencies.**get_pos_tag_set**(*files*, *c*, *tag_len*)
Returns all the existing pos_tags in files, through a SQL request.

Only the first *tag_len* characters of the tags are considered.

Note: For a more efficient call from *pos_tag_frequencies()*, this function takes an already initialized SQL cursor as an argument. If you want to use it, you can initialize *c* with the following code :

```
import sqlite3 as sql
from loacore.conf import DB_PATH
conn = sql.connect(DB_PATH)
c = conn.cursor()
```

Parameters

- **files** (list of *File*) – Files to process. Notice that only the id_files are needed.
- **c** – SQL cursor
- **tag_len** – Number of letters kept in each PoS_tag
- **tag_len** – int

Returns PoS_tags

Return type list of str

loacore.analysis.frequencies.**label_frequencies**(*files*)
Compute simple dependency label frequencies if files.

Results are returned in a dictionary that map file names to frequencies.

Frequencies are represented has dictionaries that map dependency labels to frequencies.

A list of dependency labels is also returns.

Parameters **files** (list of *File*) – Files to process

Returns Dependency labels, and a dictionary that maps file names to frequencies

Return type list of str , dict of str : dict of str : float

Example Compute simple label frequencies of uci files.

loacore.analysis.frequencies.**polarity_word_label_frequencies**(*files*, *polarity*)

Compute label frequencies for words with the specified polarity.

Parameters

- **files** (list of *File*) – Files to process
- **polarity** (str : {'positive', 'negative'}) – Word polarity to consider.

Returns Labels, and a dictionary that maps file names to frequencies

Return type list of str , dict of str : dict of str : float

Example

```
import loacore.load.file_load as file_load
import loacore.analysis.frequencies as frequencies

ids = file_load.get_id_files_by_file_paths([r'.*/uci/.+'])
files = file_load.load_database(id_files=ids, load_reviews=False)
labels, freq = frequencies.polarity_word_label_frequencies(files,
↪'positive')
```

loacore.analysis.frequencies.**polarity_word_pos_tag_frequencies**(*files*, *polarity*, *tag_len=2*)

Compute PoS_tag frequencies only for words with the specified polarity.

Parameters

- **files** (list of *File*) – Files to process
- **polarity** (str : {'positive', 'negative'}) – Word polarity to consider.
- **tag_len** – Number of letters kept in each PoS_tag
- **tag_len** – int

Returns PoS_tags, and a dictionary that maps file names to frequencies

Return type list of str , dict of str : dict of str : float

Example Compute pos_tag (first 2 characters) frequencies for positive words in uci files.

```
import loacore.load.file_load as file_load
import loacore.analysis.frequencies as frequencies

ids = file_load.get_id_files_by_file_paths([r'.*/uci/.+'])
files = file_load.load_database(id_files=ids, load_reviews=False)
labels, freq = frequencies.polarity_word_pos_tag_frequencies(files,
↪'positive')
```

loacore.analysis.frequencies.**pos_tag_frequencies**(*files*, *tag_len=2*)

Same idea as :func:'label_frequencies', but with Part Of Speech tags.

The number of characters of the PoS_tags to consider can be specified with tag_len.

Parameters

- **files** (list of *File*) – Files to process
- **tag_len** – Number of letters kept in each PoS_tag
- **tag_len** – int

Returns PoS_tags, and a dictionary that maps file names to frequencies

Return type `list` of `str` , `dict` of `str` : `dict` of `str` : `float`

Example Compute simple PoS_tags frequencies of uci files.

7.4 Polarity Check

This modules provides functions to compare results of polarity analysis. Comparation with the analysis that correspond to labelled reviews can notably be used to detect/show/count false positive and false negative of other analysis.

`loacore.analysis.polarity_check.`**`check_polarity`**(*files, analysis_to_check=['simple', 'optimistic', 'pessimistic', 'pattern_adj_cc', 'pattern_adj', 'pattern_cc'], ref='label'*)

Prints tables with the rates, in percentages, of correct, false positive and false negative classifications of reviews by file for each analysis in *analysis_to_check*, compared with *ref* analysis.

Parameters

- **`files`** (`list` of *`File`*) – Files to process.
- **`analysis_to_check`** (`list` of `str`) – Analysis to check
- **`ref`** (*`string`*) – Reference analysis, with which analysis to check are compared.

`loacore.analysis.polarity_check.`**`write_polarity_check`**(*files, analysis_to_check=['simple', 'optimistic', 'pessimistic', 'pattern_adj_cc', 'pattern_adj', 'pattern_cc'], ref='label', select='all', terminal_print=True, colored_polarity=True, directory_path='/home/paulbreugnot/Development/Python/w*

Write polarity in .txt files. If *select* is set to all, each review is written with polarities corresponding to *ref* and *analysis_to_check*. If *select* is set to *false_positive* or *false_negative*, only the review with a false positive or false negative for at least one analysis of *analysis_to_check* are written, with the corresponding(s) polarities.

Parameters

- **`files`** (`list` of *`File`*) – Files to process.
- **`analysis_to_check`** (`list` of `str`) – Analysis to compare with *ref*
- **`ref`** (*`string`*) – Reference analysis
- **`select`** (`list` of `str` : {'all', 'false_positive', 'false_negative'}) – Select option
- **`terminal_print`** (*`boolean`*) – If True, print results in the terminal.
- **`colored_polarity`** – If True, write words with colored polarity. Notice that colors are not display in most of the .txt editors. For example, in Linux, use

  ```
  cat file_name.txt
  ```

  To show the colored file in terminal.
- **`directory_path`** (path-like object) – Path of the directory in which to write files.

CHAPTER

EIGHT

MACHINE LEARNING

8.1 Word2Vec

loacore.learning.word2vec.**get_tokens_list**(*reviews*)

Returns a list of all tokens in specified reviews. For each word, if a lemma is found, lemma is used. Otherwise, word form is used.

Parameters **reviews** (list of *Review*) – Reviews to process

Returns List of tokens

Return type list of str

loacore.learning.word2vec.**review_2_vec**(*review*, *wv*)

Returns a vector representations review, according to the specified Word2Vec dictionary.

(Currently : mean of the Word2Vec vectors of words)

Parameters

- **review** (list of *Review*) – Reviews to process
- **wv** (KeyedVector) – Word2Vec dictionary

Returns Vector list

Return type list of numpy.array

loacore.learning.word2vec.**reviews_2_vec**(*reviews*, *wv*)

Returns a list of vector representations of the reviews, according to the specified Word2Vec dictionary.

(Currently : mean of the Word2Vec vectors of words)

Parameters

- **reviews** (list of *Review*) – Reviews to process
- **wv** – Word2Vec dictionary

Returns Vector list

Return type

list of numpy.array

loacore.learning.word2vec.**word_2_vec**(*reviews*)

Learn a Word2Vec dictionary from the tokens of specified reviews (tokens obtained with *get_tokens_list()*.

Parameters **reviews** (list of *Review*) – Reviews to process

Returns Word2Vec dictionary

Return type

KeyedVector

8.2 Latent Semantic Indexing

loacore.learning.lsi.**lsi_model**(*reviews*)

Train a Latent Semantic Indexing model from input *reviews*, with gensim.models.LsiModel.

During the process, a vocabulary and a dictionary mapping this vocabulary to unique integers (because LsiModel use this representation of texts) is computed from input *reviews*, and this dictionary is also returned.

To compute vocabulary, word lemmas are considered if defines, otherwise word forms are used.

Parameters **reviews** (list of *Review*) – Input reviews.

Returns A dictionary mapping vocabulary to integers, and the trained LSI model.

Return type

dict of string : int, LsiModel

loacore.learning.lsi.**review_2_vec**(*review*, *model*, *word2int_dict*)

Return a vector representation of *review* according to the given model.

Note: *word2int_dict* should be consistent with *model*, i.e. *word2int_dict* and *model* should be the results of the same call of *lsi_model()*.

Parameters

- **review** – Input review
- **review** – *Review*
- **model** – Trained LsiModel
- **word2int_dict** (dict of string : int) – A dictionary mapping vocabulary to integers.

Returns A vector representation of *review*, according to *model*.

Return type

numpy.array

loacore.learning.lsi.**reviews_2_vec**(*reviews*, *model*, *word2int_dict*)

List of vector representations of *reviews*, as returned by *review_2_vec()*.

Parameters

- **reviews** (list of *Review*) – Input reviews.
- **model** – Trained LsiModel
- **word2int_dict** (dict of string : int) – A dictionary mapping vocabulary to integers.

Returns Vector representations of *reviews*.

Return type

list of numpy.array

8.3 SVM

loacore.learning.svm.**commit_analysis**(*clf*, *reviews*, *wv*, *analysis='svm'*, *db_commit=False*)
Predict polarity of each review in reviews and add the result to their *review.polarity* dictionary with the *analysis* label.

Parameters

- **clf** (LinearSVC) – LinearSVC model
- **reviews** (list of *Review*) – Reviews to process
- **wv** – Word vectors dictionary
- **analysis** (*string*) – Analysis label
- **db_commit** – If True, results are commit to the database.

loacore.learning.svm.**get_labels_vector**(*reviews*, *ref='label'*)
Return reviews classification based on specified ref analysis. Classes are represented as -1, 0, +1, according to the results of *Polarity* fucntions :

- *is_negative()*
- *is_objective()*
- *is_positive()*

Especially used with reviews as the learning dataset, to return labels to give to the LinearSVC learning model.

Parameters

- **reviews** (list of *Review*) – Reviews to process
- **ref** (*string*) – Reference analysis

Returns Reviews classification

loacore.learning.svm.**learn_model**(*reviews*, *reviews_vectors*)
Learn and return a LinearSVC model from specified labelled reviews and Word2Vec dictionnary.

Parameters

- **reviews** (list of *Review*) – Learning Dataset
- **reviews_vectors** – Vector representations of reviews

Returns Learned model

Return type

LinearSVC

8.4 Validation

loacore.learning.validation.**k_fold_validation**(*reviews*, *reviews_vectors*, *k*)
Performs a K-Fold validation from *reviews* and their associated vector representation using sklearn function cross_val_score.

Parameters

- **reviews** (list of *Review*) – Reviews to check.
- **reviews_vectors** – Vectors representations associated to reviews.

- **k** (*int*) – K-Fold model parameter

Returns Cross validation results.

Return type

array-like object, as returned by cross_val_score.

loacore.learning.validation.**leave_p_out_validation**(*reviews*, *reviews_vectors*, *p*)

Performs a Leave-P-out validation from *reviews* and their associated vector representation using sklearn function cross_val_score.

Parameters

- **reviews** (list of *Review*) – Reviews to check.
- **reviews_vectors** – Vectors representations associated to reviews.
- **p** (*int*) – Leave-P-out model parameter

Returns Cross validation results.

Return type

array-like object, as returned by cross_val_score.

CHAPTER

NINE

SAVE AND PLOT RESULTS : *UTILS* PACKAGE

Provides useful functions to plot and save various results.

loacore.utils.plot_polarities.**print_polarity_table**(*file_score_dict*)

Print a table with columns File path, Positive Score, Negative Score and Objective Score.

Notice that displayed scores are rounded values.

Parameters **file_score_dict** (dict of int : tuple) – A dict that maps file_paths to a score tuple.

loacore.utils.plot_polarities.**save_polarity_pie_charts**(*file_score_dict*, *gui=False*, *file_path='/home/paulbreugnot/Development/Python*
file_name='polarity_pie_charts.pdf')

Plot polarity pie charts using Matplotlib, and save them into a .pdf file.

Parameters

- **file_score_dict** (dict of int : tuple) – Data to plot and save. The dict maps ID_Files to a polarity tuple (pos_score, neg_score, obj_score).
- **gui** (*boolean*) – Specify if a gui should be used to save file.
- **file_path** (path-like object) – If gui is not called : path of the directory in which plots will be saved. If directory doesn't exist, will be created. Default is set to *RESULT_PATH/sentiment_analysis/*
- **file_name** (*string*) – Name of the saved file.

loacore.utils.plot_frequencies.**frequencies_bar_chart**(*files_frequencies*, *plot=False*, *save=True*, *gui=False*, *file_path='/home/paulbreugnot/Development/Python/w*
file_name='frequencies_bar_chart.pdf', *val_number=0*)

This function allows to plot or save results of *loacore.analysis.frequencies*.

If plot is set to True, results will be presented in a matplotlib figure, but *save* will be set to False. (matplotlib restriction)

Notice that all the results are arbitrarily re-ordered in the order of the first file of files_frequencies. This means for example that the right-most label of the graph corresponds to the most common of the first file, but that is not necessarily true for the others. This could also be useful to consider when *val_number* is used to show only "interesting" labels. Those labels actually correspond to the *val_number* most common labels of the first file.

Parameters

- **files_frequencies** (dict of str : dict of label : float .) – Dictionary that maps file names to frequencies.
- **plot** (*boolean*) – If True, plot results as a figure.

- **save** (*boolean*) – If not *plot*, save figure as a PDF.
- **gui** (*boolean*) – Specify if a GUI should be used to set directory.
- **file_path** (path-like object) – Path of the directory used.

  Default : 'RESULT_PATH/frequencies/'
- **file_name** (*string*) – File name
- **val_number** – Number of labels to show. Only the most commons in the first file will be kept.

Example Save the 60 most commons bigram labels frequencies of uci files in a PDF.

```
import loacore.load.file_load as file_load
import loacore.analysis.frequencies as frequencies
import loacore.utils.plot_frequencies as plot_frequencies
import os
from loacore.conf import RESULT_PATH

ids = file_load.get_id_files_by_file_paths([r'.*/uci/.+'])
files = file_load.load_database(id_files=ids, load_reviews=False)
labels, freq = frequencies.bigram_label_frequencies(files)

plot_frequencies.frequencies_bar_chart(
    freq,
    plot=False,
    save=True,
    file_path=os.path.join(RESULT_PATH, 'frequencies', 'label_
↪frequencies', 'uci'),
    file_name="uci_label_bigrams_frequencies.pdf",
    val_number=60)
```

loacore.utils.plot_frequencies.**write_frequencies**(*files_frequencies*, *file_path='/home/paulbreugnot/Development/Python/workspa*

Write frequencies to a txt file in file_path folder. Original raw file names are used.

Parameters

- **files_frequencies** (dict of str : dict of label : float .) – Dictionary that maps file names to frequencies.
- **file_path** (path-like object) – Directory path

CHAPTER

TEN

OTHER USEFUL EXAMPLES

Display dependency trees from node that have at least one adjective as a child, from all the database file.

```
import loacore.load.file_load as file_load
files = file_load.load_database()

import loacore.analysis.pattern_recognition as pattern_recognition
for file in files:
    for review in file.reviews:
        for sentence in review.sentences:
            patterns = pattern_recognition.pos_tag_patterns_recognition([sentence], [[
↪'*'], ['A']])
            dt = sentence.dep_tree
                for pattern in patterns:
                    sentence.print_sentence()
                    dt.print_dep_tree(root=pattern[0])
                    print('')
```

Results:

```
la reserva natural
reserva (sentence, NCCS000, reserva)
    la (spec, None, el)
    natural (s.a, AQ0CS00, natural)

muy buena atencion el paisaje
atencion (sentence, NCFS000, )
buena (s.a, AQ0FS00, bueno)
    muy (spec, RG, muy)
paisaje (sn, NCMS000, paisaje)
    el (spec, None, el)

termales y calidad
calidad (sentence, NCFS000, calidad)
    termales (s.a, AQ0CP00, termal)
    y (coord, None, y)

el agua termal
agua (sentence, NCCS000, agua)
    el (spec, None, el)
    termal (s.a, AQ0CS00, termal)
...
```

PYTHON MODULE INDEX

l