

---

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

**Paul Breugnot**

**Aug 13, 2018**



## CONTENTS:

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Requirements</b>	<b>3</b>
2.1	Freeling . . . . .	3
2.2	Database . . . . .	3
2.3	Utils . . . . .	3
<b>3</b>	<b>Configuration</b>	<b>5</b>
<b>4</b>	<b>Classes</b>	<b>7</b>
4.1	File . . . . .	7
4.2	Review . . . . .	7
4.3	Sentence . . . . .	8
4.4	Word . . . . .	9
4.5	Synset . . . . .	9
4.6	DepTree . . . . .	10
4.7	DepTreeNode . . . . .	10
4.8	Polarity . . . . .	11
<b>5</b>	<b>Feeding database : <i>process</i> package</b>	<b>13</b>
5.1	Raw Processes . . . . .	13
5.2	Freeling Processes . . . . .	14
5.3	Feed database . . . . .	16
<b>6</b>	<b>Load data from database : <i>load</i> package</b>	<b>17</b>
6.1	Load Files . . . . .	17
6.2	Load Reviews . . . . .	18
6.3	Load Sentences . . . . .	20
6.4	Load Words . . . . .	21
6.5	Load Synsets . . . . .	22
6.6	Load Lemmas . . . . .	23
6.7	Load DepTrees . . . . .	24
<b>7</b>	<b>Analyse data : <i>analysis</i> package</b>	<b>27</b>
7.1	Sentiment Analysis . . . . .	27
7.2	Pattern Recognition . . . . .	29
7.3	Frequencies . . . . .	34
7.4	Polarity Check . . . . .	40
<b>8</b>	<b>Machine Learning</b>	<b>41</b>
8.1	Word2Vec . . . . .	41

8.2	SVM . . . . .	41
9	Save and plot results : <i>utils</i> package	43
10	Other Useful Examples	45
	Python Module Index	47

## **INTRODUCTION**

*This project was originally lead by the NLP research group of the Technical 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 dependency 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*



## 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.

**For now**, only a Linux installation in the default folder */usr/local* is supported, but this should be fixed in next improvements.

### 2.2 Database

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.

### 2.3 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.





## CONFIGURATION

`loacore.conf.check_freeling_path()`

Print current *freeling\_path*.

`loacore.conf.check_lang()`

Print 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



## 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





## 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,  
                                                             freel-  
                                                             ing_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,    freel-  
                                                       ing_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 Senti-WordNet 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, freel-
                                                         ing_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,
                                                             freel-
                                                             ing_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 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 multicore processors architecture 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)
```

## 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_paths(file_paths_re)`

This function can be used to retrieve file ids in database from their path.

All the regular expression in the `** argument list**` will be checked.

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

.

**Parameters** `file_paths_re` (list of str) – Regular expressions 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_paths([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)`

Load the complete database as a list of *File*, with all the dependencies specified in parameters loaded in them.

**Parameters**

- **id\_files** – If specified, load only the files with the corresponding ids. Otherwise, load all the files.
- **load\_reviews** – Specify if Reviews need to be loaded if files.

- **load\_sentences** – If Reviews have been loaded, specify if Sentences need to be loaded in reviews.
- **load\_words** – If Sentences have been loaded, specify if Words need to be loaded in sentences.
- **load\_deptrees** – If Words have been loaded, specify if DepTrees need to be loaded in sentences.

**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 huele 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))
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)
```

(continues on next page)

(continued from previous page)

```
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)
```



## ANALYSE DATA : ANALYSIS PACKAGE

### 7.1 Sentiment Analysis

`loacore.analysis.sentiment_analysis.compute_extreme_files_polarity` (*files*, *pes-*  
*simistic=False*,  
*freel-*  
*ing\_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,
                                                                    com-
                                                                    mit_polarities=False,
                                                                    pes-
                                                                    simistic=False,
                                                                    freel-
                                                                    ing_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 committed 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\_Polarity - : 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 considered 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/20)
- [http://clic.ub.edu/corpus/webfm\\_send/18](http://clic.ub.edu/corpus/webfm_send/18)
- [http://clic.ub.edu/corpus/webfm\\_send/49](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 'subj' 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,
↳[['subj']])
>>> patterns_str = pattern_recognition.print_patterns(patterns, PoS_tag_
↳display=True, label_display=True)
( colmo : NCMS000 : subj )
( que : None : subj )
( que : None : subj )
( tencion : None : subj )
( señora : NCFS000 : subj )
( que : None : subj )
( turista : NCCS000 : subj )
( ella : None : subj )
( que : None : subj )
...
```

---

`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 module provides functions to compare results of polarity analysis. Comparison 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',    'pat-  
                                         tern_adj_cc',    'pattern_adj',    'pat-  
                                         tern_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,          analy-  
                                         sis_to_check=['simple',  
                                         'optimistic',    'pessimistic',  
                                         'pattern_adj_cc',    'pat-  
                                         tern_adj',    'pattern_cc'],  
                                         ref='label',    select='all',  
                                         terminal_print=True,    col-  
                                         ored_polarity=True,    direc-  
                                         tory_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.

## MACHINE LEARNING

### 8.1 Word2Vec

### 8.2 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** (`KeyedVector`) – 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* fuctions :

- `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 dictionary.

#### Parameters

- **reviews** (list of `Review`) – Learning Dataset
- **reviews\_vectors** (list of `numpy.array`) – Vector representations of reviews

**Returns** Learned model

**Return type**

LinearSVC

## 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/w  
                                                       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



## 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

### I

- `loacore.analysis.frequencies`, [34](#)
- `loacore.analysis.pattern_recognition`,  
[29](#)
- `loacore.analysis.polarity_check`, [40](#)
- `loacore.analysis.sentiment_analysis`, [27](#)
- `loacore.conf`, [5](#)
- `loacore.learning.svm`, [41](#)
- `loacore.load.deptree_load`, [24](#)
- `loacore.load.file_load`, [17](#)
- `loacore.load.lemma_load`, [23](#)
- `loacore.load.review_load`, [18](#)
- `loacore.load.sentence_load`, [20](#)
- `loacore.load.synset_load`, [22](#)
- `loacore.load.word_load`, [21](#)
- `loacore.process.deptree_process`, [15](#)
- `loacore.process.file_process`, [16](#)
- `loacore.process.lemma_process`, [14](#)
- `loacore.process.review_process`, [13](#)
- `loacore.process.sentence_process`, [14](#)
- `loacore.process.synset_process`, [14](#)
- `loacore.utils.file_writer`, [43](#)
- `loacore.utils.plot_frequencies`, [43](#)
- `loacore.utils.plot_polarities`, [43](#)
- `loacore.utils.print_patterns`, [43](#)