Hotel Reviews Language Processing Documentation

Paul Breugnot

Jul 16, 2018

CONTENTS:

CHAPTER

ONE

REQUIREMENTS

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.

CHAPTER

TWO

CLASSES

2.1 File

class src.database.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='windows-1252'*)

Load file from file system using file_path and specified encoding.

Parameters **encoding** – Source file encoding. Default is set to *windows-1252*, the encoding obtained from .txt conversion in Excel.

Returns file object

2.2 Review

class src.database.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

2.3 Sentence

class src.database.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 *Sentence* s from database and convert them into Freeling Sentences.

```
import src.database.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

print_sentence(*print_sentence=True*)

Convenient way of printing sentences from their word list attribute.

Parameters **print_sentence** – Can be set to False to compute and return the string corresponding to the sentence, without printing it.

Returns String representation of the sentence

Return type string

2.4 Word

class src.database.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

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

2.5 Synset

class src.database.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

2.6 DepTree

class src.database.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

print_dep_tree(*root=None*, *print_dep_tree=True*)

Parameters

- **root** (*DepTreeNode*) – If set, node from which to start to print the tree. self.root otherwise.
- **print_dep_tree** (*boolean*) – Can be set to False to compute and return the string corresponding to the tree, without printing it.

Returns String representation of DepTree instance

Return type string

2.7 DepTreeNode

class src.database.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

CHAPTER

THREE

FEEDING DATABASE : *PROCESS* PACKAGE

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

3.1 Raw Processes

3.1.1 Normalization and review splitting

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

`src.database.process.review_process.`**`add_reviews_from_files`**(*files*)
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 `file_process.add_files()` function.

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

Returns added `Review` s

Return type `list` of `Review`

`src.database.process.review_process.`**`normalize`**(*text*)
Performs raw text normalization.

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

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

Returns reviews

Return type `list` of `string`

3.2 Freeling Processes

3.2.1 tokenization

src.database.process.sentence_process.**add_sentences_from_reviews**(*reviews*)
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`) – `Review` s to process

Returns added `Sentence` s

Return type `list` of `Sentence`

3.2.2 lemmatization

src.database.process.lemma_process.**add_lemmas_to_sentences**(*sentences*, *print_lemmas=False*)
Performs a Freeling process to add lemmas to `Word` s.

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

Our `Sentence` s 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`) – `Sentence` s to process
- **print_lemmas** (*boolean*) – If True, print lemmatization results

3.2.3 disambiguation

src.database.process.synset_process.**add_polarity_to_synsets**()
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.

src.database.process.synset_process.**add_synsets_to_sentences**(*sentences*, *print_synsets=False*)
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 `Sentence` s 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 `Sentence` s 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`) – `Sentence` s to process
- **print_synsets** (*boolean*) – If True, print disambiguation results

3.2.4 dependency tree generation

`src.database.process.deptree_process.`**`add_dep_tree_from_sentences`**(*sentences*, *print_result=False*)

Generates the dependency trees of the specified `Sentence` s 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`) – `Sentence` s to process
- **print_result** (*boolean*) – Print PoS_tags and labels associated to each `Word`

3.3 Feed database

`src.database.process.file_process.`**`add_files`**(*file_paths*)

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

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

FOUR

LOAD DATA FROM DATABASE : *LOAD* PACKAGE

4.1 Load Files

src.database.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.

src.database.load.file_load.**load_database**(*id_files=[]*, *load_reviews=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 `File` s.
- **load_sentences** – If Reviews have been loaded, specify if Sentences need to be loaded in `Review` s.
- **load_words** – If Sentences have been loaded, specify if Words need to be loaded in `Sentence` s.
- **load_deptrees** – If Words have been loaded, specify if DepTrees need to be loaded in `Sentence` s.

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 src.database.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',
```

(continues on next page)

(continued from previous page)

```
'../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_CC.txt',
'../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_GR.txt']
```

Example Load the first 10 files without `DepTree` s.

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

`src.database.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`

4.2 Load Reviews

`src.database.load.review_load.`**`load_reviews`**(*id_reviews=[]*, *load_sentences=False*, *load_words=False*, *load_deptrees=False*)

Load `Review` s 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 `Review` s.
- **`load_words`** (*boolean*) – If Sentences have been loaded, specify if Words need to be loaded in `Sentence` s.
- **`load_deptrees`** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in `Sentence` s.

Returns Loaded reviews

Return type `list` of `Review` s

Example Load all reviews with sentences and words

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

src.database.load.review_load.**load_reviews_by_id_files**(*id_files*, *load_sentences=False*, *load_words=False*, *load_deptrees=False*)

Load Review s 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 Review s.
- **load_words** (*boolean*) – If Sentences have been loaded, specify if Words need to be loaded in Sentence s.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in Sentence s.

Returns Loaded reviews

Return type list of Review s

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

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

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

Load Review s 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 Review s.
- **load_words** (*boolean*) – If Sentences have been loaded, specify if Words need to be loaded in Sentence s.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in Sentence s.

Returns Loaded reviews

Return type list of Review s

4.3 Load Sentences

src.database.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 Sentence s.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in Sentence s.

Returns Loaded sentences

Return type list of Sentence

Example Load sentences 1,2 and their words.

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

src.database.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 Sentence s.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in Sentence s.

Returns Loaded sentences

Return type list of Sentence

Example

Load all the sentences from file 1.

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

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

Load Sentence s 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 `Sentence` s.
- **load_deptrees** (*boolean*) – If Words have been loaded, specify if DepTrees need to be loaded in `Sentence` s.

Returns Loaded sentences

Return type `list` of `Sentence`

4.4 Load Words

`src.database.load.word_load.`**load_words**(*id_words=[]*, *load_lemmas=True*, *load_synsets=True*)

Load `Word` s 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 `Word` s.
- **load_synsets** (*boolean*) – Specify if Synsets need to be loaded in `Word` s.

Returns loaded words

Return type `list` of `Word`

Example Load all words and their lemmas, synsets.

```
>>> import src.database.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
↪']
```

`src.database.load.word_load.`**load_words_in_dep_trees**(*dep_trees*, *load_lemmas=True*, *load_synsets=True*)

Load `Word` s 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 Word s.
- **load_synsets** (*boolean*) – Specify if Synsets need to be loaded in Word s.

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

Load Word s 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 Word s.
- **load_synsets** (*boolean*) – Specify if Synsets need to be loaded in Word s.

Returns loaded words

Return type list of Word

4.5 Load Synsets

src.database.load.synset_load.**load_synsets**(*id_synsets=[]*)

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 src.database.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
```

src.database.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`

4.6 Load Lemmas

`src.database.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 `string`

Example

Load all lemmas from database.

```
>>> import src.database.load.lemma_load as lemma_load
>>> lemmas = lemma_load.load_lemmas()
>>> print(len(lemmas))
103827
>>> print(lemmas[10])
bailar
```

`src.database.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 `string`

4.7 Load DepTrees

`src.database.load.deptree_load.`**load_dep_tree_in_sentences**(*sentences*, *load_words=True*)

Load `DepTree` s 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 `DepTree` s.

Returns loaded deptrees

Return type `list` of `DepTree`

`src.database.load.deptree_load.`**load_dep_trees**(*id_dep_trees=[]*, *load_words=True*)

Load `DepTree` s 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 `DepTree` s.

Returns loaded deptrees

Return type `list` of `DepTree`

Example

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

```
>>> import src.database.load.deptree_load as deptree_load
>>> deptrees = deptree_load.load_dep_trees()
>>> deptree_str = deptrees[500].print_dep_tree()
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

FIVE

ANALYSE DATA : *ANALYSIS* PACKAGE

5.1 Sentiment Analysis

src.database.analysis.sentiment_analysis.**compute_files_polarity**(*files*)

Perform the easiest sentiment analysis possible : a normalized sum of the positive/negative/objective polarities available in all synsets of each file.

Return a dictionnary that map file_paths to a polarity tuple. A polarity tuple is a tuple of length 3, with this form : (positive_score, negative_score, objective_score)

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

Returns IdFile/Scores dictionnary

Return type dict of int : tuple

Example

Load all files and compute basic polarities

```
>>> import src.database.load.file_load as file_load
>>> import src.database.analysis.sentiment_analysis as sentiment_analysis
>>> file_load.load_database(load_deptrees=False)
>>> polarities = sentiment_analysis.compute_files_polarity(files)
>>> sentiment_analysis.print_polarity_table(polarities)
+-------------------------------------------------------------------------------
↪------------------+-----------+...
|                                                  File
↪                  | Pos_Score |...
+-------------------------------------------------------------------------------
↪------------------+-----------+...
|           ../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_EO.txt          |   0.000   |...
|           ../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_CC.txt          |   0.069   |...
|           ../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_GR.txt          |   0.000   |...
|           ../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_JA.txt          |   0.060   |...
|           ../../data/raw/TempBaja/Balneario2/
↪EncuestaTemporadaBajafinalbalneario2_CD.txt          |   0.080   |...
|           ../../data/raw/TempBaja/Balneario3/
↪EncuestaTemporadaBajafinalbalneario3_JA.txt          |   0.055   |...
|           ../../data/raw/TempBaja/Balneario3/
↪EncuestaTemporadaBajafinalbalneario3_CD.txt          |   0.019   |...
...
```

src.database.analysis.sentiment_analysis.**print_polarity_table**(*file_score_dict*)
Print a table with columns File path, Positive Score, Negative Score and Objective Score.

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

5.2 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**.

src.database.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 :obj'string') – 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 string. 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 src.database.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import src.database.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 )
```

(continues on next page)

(continued from previous page)

```
( poner : VMN0000 : S, le : None : ci )
( poner : VMN0000 : S, a : None : ci )
( establecer : VMN0000 : S, uun : None : cc )
...
```

src.database.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 :obj'string') – 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 src.database.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import src.database.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 )
( 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 src.database.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import src.database.analysis.pattern_recognition as pattern_
↪recognition
```

(continues on next page)

(continued from previous page)

```
>>> 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 )
```

---

src.database.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 :obj'string') – 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 src.database.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import src.database.analysis.pattern_recognition as pattern_
↪recognition
>>> patterns = pattern_recognition.pos_tag_patterns_
↪recognition(sentences, [['N'], ['A']])
>>> 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 the nodes with dependency label 'suj' in file 28 (_PQRS.txt)

```
>>> import src.database.load.sentence_load as sentence_load
>>> sentences = sentence_load.load_sentences_by_id_files([28])
>>> import src.database.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 )
...
```

PYTHON MODULE INDEX

S