fastsemsim Documentation

Release 1.0.0beta

Marco Mina

Mar 10, 2019

Contents

Release 1.0

Date Mar 10, 2019

Homepage https://sites.google.com/site/fastsemsim/

FastSemSim is a Python package for calculating semantic similarity over ontologies (yes, including the Gene Ontology).

FastSemSim has three main components:

- An underlying library of data structures to represent ontologies and annotation corpora, and algorithms to calculate semantic similarity measures.
- A comprehensive API, built on top of the library, to load and play with ontologies and annotation corpora and calculate semantic similarity scores within the Python environment.
- A command line interface (CLI) to compute semantic similarity without requiring programming skills.

This reference manual details modules and classes included in FastSemSim, as well as the command line interface usage. The follwoing sections library, API, Examlpes and CLI describe the aforementioned components and provide some examlples.

CHAPTER 1

FastSemSim library

The FastSemSim Python package implements

- a set of classes to represent and reason with ontologies (e.g. Gene Ontology)
- a set of classes to represent and reason on ACs
- the algorithms to calculate semantic similarity measures

1.1 Ontologies

Algorithms and data structure to parse and represent ontologies, and exctract information from them.

1.1.1 Ontologies

Ontology.Ontology

Ontology.GeneOntology

Ontology.CellOntology

Ontology.DiseaseOntology

1.2 Annotation Corpora (ACs)

Algorithms and data structure to parse and represent ACs, and exctract information from them.

1.2.1 Annotation Corpora (ACs)

Ontology.AnnotationCorpus

Ontology.GAF2AnnotationCorpus

Ontology.PlainAnnotationCorpus

1.3 Semantic Similarity measures

Implementation of semantic similarity measures.

1.3.1 Semantic Similarity

SemSim.TermSemSim

SemSim.ObjSemSim

SemSim.ObjSetSemSim

SemSim.SetSemSim

Specific Semantic Similarity measures

SemSim.ResnikSemSim

SemSim.CosineSemSim

SemSim.CzekanowskiDiceSemSim

SemSim.DiceSemSim

SemSim.GSESAMESemSim

SemSim.JaccardSemSim

SemSim.JiangConrathSemSim

SemSim.LinSemSim

SemSim.SimGICSemSim

SemSim.SimICNDSemSim

SemSim.SimICNPSemSim

SemSim.SimICSemSim

SemSim.SimNTOSemSim

SemSim.SimRelSemSim

SemSim.SimTOSemSim

SemSim.SimUISemSim

SemSim.MixSemSim

SemSim.avgSemSim

SemSim.maxSemSim

SemSim.BMASemSim

SemSim.SemSimUtils

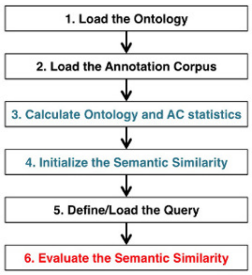
CHAPTER 2

FastSemSim API

The layer of "entrypoint functions" (API) available in FastSemSim conveniently allows interacting with ontologies, Annotation Corpora (ACs) and semantic similarity without knowing the inner workings of the FastSemSim library.

This section describes these entrypoint functions and explains how to use them. Examples and use cases are presented as well.

The standard processing workflow used when calculating semantic similarity scores can be recapitulated in the following 6 steps:



For each of these points, some entrypoint functions take care of masking all the inner workings of the package. It is noteworthy to note that albeit the primary function of FastSemSim is calculating semantic similarity, its representation of ontologies and ACs can be used as a base to extract statistics. explore the data or perform other analyses based on ontology annotations.

2.1 0. Importing FastSemSim

After installing the pacakge, you can import FastSemSim in your Python environment with:

```
import fastsemsim
```

2.2 1. API - Ontologies

2.2.1 Loading an ontology

Loading an ontology from a file or a descriptor (more about descriptors in section FastSemSim Datasets) is as simple as running a single line of code using the entrypoint function fastsemsim.load_ontology:

```
my_ontology = fastsemsim.load_ontology(...)
```

If invoked without parameters, this function will load the Gene Ontology shipped with fastsemsim. Passing the right parameters to the function allows loading other ontologies included in fastsemsim, as well as custom ontologies.

Here the full description of the function:

2.3 2. API - Annotation Corpora

2.3.1 Loading an annotation corpus

Loading an annotation corpus (AC) is as easy as loading ontologies. The only additional requirement is that an ontology must be passed as parameter to the parser, that will take care of matching the terms in the ontology to the terms in the annotation corpus. The AC can be loaded from a file or from a descriptor (more about descriptors in section FastSemSim Datasets) using the entrypoint function fastsemsim.load_ac:

```
ac = fastsemsim.load_ac(ontology = my_ontology, ...)
```

If invoked without parameters, beside the mandatory ontology parameter, this function will load one of the ACs shipped with fastsemsim. Passing the right parameters to the function allows loading other ontologies included in fastsemsim (see Dataset section), as well as custom ontologies from a file. In general, fastsemsim is able to load any file coming from the geneontology community (gaf2 file format).

Here the full description of the function:

2.4 3. Data embedded in FastSemSim

Fastsemsim includes some of the standard broadly used ontologies and annotation corpora. The corresponding files are automatically installed with the library.

All the available ACs and ontologies are managed by the Dataset class. The Dataset class works around the concept of Descriptor. Each ontology or AC available is referenced with a Descriptor indicating where the file is stored and the parameters necesary to correctly load it. Upon importing fastsemsim, an instance of the class Dataset is automatically created and filled with the descriptors of the Acs and ontologies embedded in fatsemsim. Such Dataset object is available as:

```
fastsemsim.dataset
```

The Dataset class exposes some convenient functions to easily interrogate the available ACs and ontologies and load them. It is possible, for instance, to list the available ontologies of a given type (e.g. GeneOntology, DiseaseOntology, ...) and get a valid descriptor. The descriptor can be passed to the ontology or AC loading functions (see API - Ontoliges and API - Annotation Corpora sections).

The main functions exposed by Dataset are:

- list_ontologies(): list the ontologies available in fastsemsim.

```
fastsemsim.dataset.list_ontologies()
```

- list_acs(): list the ACs available in fastsemsim, you can use the function .

```
fastsemsim.dataset.list_acs()
```

- get_ontology(): returns the descriptor(s) of the ontologies satisfying the passed parameters (if any).

Note that more than one descriptor might be returned.

```
fastsemsim.dataset.get_ontology(ontology_type=...)
fastsemsim.dataset.get_ontology(ontology_type='GeneOntology') # to look specifically␣
↪for the Gene Ontology
```

- get_annotation_corpus(): returns the descriptor(s) of the annotation corpora satisfying the passed parameters (if any).

Note that more than one descriptor might be returned.

```
fastsemsim.dataset.get_annotation_corpus(ontology_type=..., ac_species=...)
fastsemsim.dataset.get_annotation_corpus(ontology_type='GeneOntology', ac_species=
↪'human') # look for ACs compatible with the Gene Ontology, for the human species
```

Here the full description of the four functions list above:

For more details about the Dataset class, check the full documentation of the data module available at the follwing page:

2.4.1 Ontologies and annotation corpora included in fastsemsim

Class Dataset

2.5 4. Calculating the Semantic similarity

2.6 5. Calculating the Semantic similarity - batch mode

CHAPTER 3

Examples and use cases

In this section real example code and use case scenarios are proposed.

3.1 Loading the Gene Ontology

In this example the Gene Ontology embedded in FastSemSim is loaded. The quick example loads the Gene Ontology in one simple call. The more complete example explores some of the parameters that can be used.

Please refer to the FastSemSim API section for a detailed description of the function to load ontologies.

The complete example code is available in the file fastsemsim/examples/load_ontology.py within the source tarball of FastSemSim.

3.1.1 The quick way

```
# Import fastsemsim
import fastsemsim

ontology = fastsemsim.load_ontology(ontology_type = 'GeneOntology')
```

3.1.2 A more comprehensive example

```
# Import fastsemsim
import fastsemsim

# Select the type of ontology (GeneOntology, ...)
ontology_type = 'GeneOntology'
# ontology_type = 'CellOntology'
# ontology_type = 'DiseaseOntology'

```

(continues on next page)

(continued from previous page)

```
# Select the relatioships to be ignored. For the GeneOntology, has_part is ignore by default, for CellOntology, lacks_plasma_membrane_part is ignored by default
# ontology_parameters = {}
# ontology_parameters = {'ignore':{}}
# ontology_parameters = {'ignore':{'has_part':True, 'occurs_in':True, 'happens_during':True}}
ignore_parameters =     {'ignore':{'regulates':False, 'has_part':True, 'negatively_regulates':False, 'positively_regulates':False, 'occurs_in':False, 'happens_during':True, 'lacks_plasma_membrane_part':True}}

# Select the source file type (obo or obo-xml)
ontology_file_type = 'obo'

# Select the ontology source file name. If None, the default ontology_type included in fastsemsim will be used
ontology_source_file = None


ontology = fastsemsim.load_ontology(source_file = ontology_source_file, file_type = ontology_file_type, ontology_type = ontology_type, ontology_descriptor = None, parameters=ignore_parameters)
```

3.2 Loading the human Gene Ontology annotation corpus

In this example the human Gene Ontology annotation corpus for Uniprot Proteins embedded in FastSemSim is loaded. The quick example loads the Annotation Corpus in one simple line. The more complete example explores some of the parameters that can be used. Note that in any case an ontology has to be already loaded.

Please refer to the FastSemSim API section for a detailed description of the function to load acs.

The complete example code is available in the file fastsemsim/examples/load_go_annotation_corpus.py within the source tarball of FastSemSim.

3.2.1 The quick way

```
ac = fastsemsim.load_ac(ontology, ac_species = 'human')
```

3.2.2 A more comprehensive example

```
# Select the ac source file name. If None, the default ac included in fastsemsim for the ac_species will be used
ac_source_file = None

ac_species = 'human'
# ac_species = 'arabidopsis'
# ac_species = 'fly'
# ac_species = 'mouse'
# ac_species = 'rat'
# ac_species = 'worm'
# ac_species = 'zebrafish'
```

(continues on next page)

(continued from previous page)

```
# ac_source_file_type = 'plain'
ac_source_file_type = 'gaf-2.0'

ac_params = {}

# gaf-2.0 ac
ac_params['filter'] = {} # filter section is useful to remove undesired annotations
ac_params['filter']['EC'] = {} # EC filtering: select annotations depending on their␣
↪EC
# ac_params['filter']['EC']['EC'] = EC_include # select which EC accept or reject
# ac_params['filter']['EC']['inclusive'] = True # select which EC accept or reject
# ac_params['filter']['EC'] = {} # EC filtering: select annotations depending on␣
↪their EC
# ac_params['filter']['EC']['EC'] = EC_ignore # select which EC accept or reject
# ac_params['filter']['EC']['inclusive'] = False # select which EC accept or reject

ac_params['filter']['taxonomy'] = {}
# ac_params['filter']['taxonomy']['taxonomy'] = tax_include # set properly this field␣
↪to load only annotations involving proteins/genes of a specific species
# ac_params['filter']['taxonomy']['inclusive'] = True # select which taxonomy accept␣
↪or reject
# ac_params['filter']['taxonomy'] = {}
# ac_params['filter']['taxonomy']['taxonomy'] = tax_ignore
# ac_params['filter']['taxonomy']['inclusive'] = False # select which EC accept or␣
↪reject
# ac_params['simplify'] = True # after parsing and filtering, removes additional␣
↪information such as taxonomy or EC. Useful if you have a huge amount of annotations␣
↪and not enough memory


ac_descriptor = fastsemsim.dataset.get_default_annotation_corpus(ontology_
↪type=ontology_type, ac_species=ac_species)
ac = fastsemsim.load_ac(ontology, source_file=None, file_type=None, species=None, ac_
↪descriptor=ac_descriptor, params=ac_params)
```

3.3 Calculating a semantic similarity

Once an ontology (and possibly an annotation corpus) has been loaded, semantic similarity scores can be calculated. As in the previous examples, a first quick glance at some simple code is presented, followed by a more complex example.

Note that in any case an ontology (and the annotation corpus, if required) has to be already loaded.

Please refer to the FastSemSim API section for a detailed description of the function to initialize semantic similarity measures and calculate semantic similarity scores.

The complete example code is available in the file fastsemsim/examples/calculate_ss_on_go.py within the source tarball of FastSemSim.

3.3.1 The quick way

In this quick example we tell fastSemSim we wish to initialize the Resnik semantic similarity measure between proteins in the annotation corpus (using the parameter semsin_type='obj'. as Resnik is a term pairwise measure, we also

need to say how to mix the single term-term similarities (mixing_strategy parameter). The returned object can then be used (by invoking its method SemSim) to calculate similarities between two proteins.

```
# Parameters for the SS
semsim_type='obj'
semsim_measure='Resnik'
mixing_strategy='max'

# Initializing semantic similarity
ss = fastsemsim.init_semsim(ontology = ontology, ac = ac, semsim_type = semsim_type,␣
↪semsim_measure = semsim_measure, mixing_strategy = mixing_strategy)

# Calculating SS for some pairs of proteins...
res = ss.SemSim('O75884', 'Q9NQB0')
print(res)
```

3.4 Calculating semantic similarity - batch mode

The previous minimal example is sufficient to calculate a semantic similarity score. Suppose we want to calculate the semantic similarity in a pairwise fashion between all the pairs of proteins within a given set. Of course, we could use the object built in the quick example from above, and repeteadly invoke it within a loop cycling through all the protein pairs. Albeit feasible, this is not very efficient and requires some programming effort. For this reasons, FastSemSim provides a 'batch mode' to calculate semantic simialrity between several pairs/groups of protein.

```
# Parameters for the SS
semsim_type='obj'
semsim_measure='Resnik'
mixing_strategy='max'
ss_util=None
semsim_do_log=False
semsim_params={}

# Initializing batch Semantic Similarity onject...
ssbatch = fastsemsim.init_batchsemsim(ontology = ontology, ac = ac, semsim_type =␣
↪semsim_type, semsim_measure = semsim_measure, mixing_strategy = mixing_strategy, ss_
↪util = ss_util, do_log = semsim_do_log, params = semsim_params)

# Same as before, using the pairwise ss as template...
ssbatch2 = fastsemsim.init_batchsemsim(ontology = ontology, ac = ac, semsim=ss)


# Calculating pairwise SS in batch mode for a list of proteins...

batch_query_pairs = [['O75884', 'Q9NQB0'], ['Q14206', 'Q8IUH3' ]]
res = ssbatch.SemSim(query=batch_query_pairs, query_type='pairs')

batch_query_pairwise = ['O75884', 'Q9NQB0', 'Q14206', 'Q8IUH3' ]
res2 = ssbatch.SemSim(query=batch_query_pairwise, query_type='pairwise')

res_long = ssbatch.SemSim(query= 10*batch_query_pairwise, query_type='pairwise')
res_very_long = ssbatch.SemSim(query= 30*batch_query_pairwise, query_type='pairwise')
res_very_very_long = ssbatch.SemSim(query= 100*batch_query_pairwise, query_type=
↪'pairwise')

res_long_v2 = ssbatch2.SemSim(query= 10*batch_query_pairwise, query_type='pairwise')
```

(continues on next page)

(continued from previous page)

```
%%timeit
res_very_very_long = ssbatch.SemSim(query= 10*batch_query_pairwise, query_type=
↪'pairwise')
```

3.5 Putting all together

Here is a complete minimal example to calculate the semantic similaity between two proteins over the gene ontology

```
# Import fastsemsim
import fastsemsim

ontology = fastsemsim.load_ontology(ontology_type = 'GeneOntology')
ac = fastsemsim.load_ac(ontology, ac_species = 'human')
ssbatch = fastsemsim.init_batchsemsim(ontology = ontology, ac = ac, semsim_type = 'obj
↪', semsim_measure = 'resnik', mixing_strategy = 'BMA')

# Calculating pairwise SS in batch mode for a list of proteins...
ssbatch.set_root('molecular_function')
ssbatch.set_output(output = 'console')
batch_query_pairs = [['O75884', 'Q9NQB0'], ['Q14206', 'Q8IUH3' ]]
res = ssbatch.SemSim(query=batch_query_pairs, query_type='pairs'


ssbatch.set_root('biological_process')
ssbatch.set_output(output = None)
batch_query_pairwise = ['O75884', 'Q9NQB0', 'Q14206', 'Q8IUH3' ]
res2 = ssbatch.SemSim(query=batch_query_pairwise, query_type='pairwise')
```

3.5.1 Divers

CHAPTER 4

FastSemSim Command Line Interface (CLI)

The FastSemSim package includes a command line Interface to load ontologies and annotation corpora, and evaluate semantic similarity, without requiring any programming effort.

This section explains how to start and use FastSemSim from the command line.

4.1 Running the Command Line Interface

FastSemSim installations takes care to create the executable of the command line Interface. To run it, just type `fastsemsim` in a console.

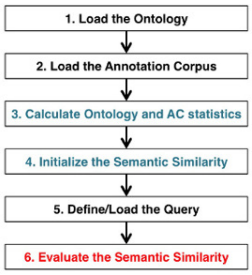
Run `fastsemsim -h` to visualize the full set of parameters.

4.2 CLI capabilities and standard workflow

The CLI main function is designed to evaluate semantic similarity scores.

It currently allows to load an arbitrary obo ontology and annotation corpus. If no custom ontology is provided, the embedded version will be used.

The standard processing workflow performed by the CLI to evaluate the semantic similarity scores can be recapitulated in the following 6 steps



4.3 Examples of semantic similarity calculation with the CLI

This section provides several CLI use cases. You may take them as exampels and modify the parameters to fit your objectives.

More examples of CLI usage are included in the file fastsemsim/examples/example_cmdline.sh

4.3.1 Exampe 1. Evaluate the Resnik BMA semantic similarity over the Gene Ontology between all the pairs of human proteins annotated in the Uniprot GOA:

```
fastsemsim --ontology_type GeneOntology --query_ss_type obj --tss Resnik --mix
BMA --query_input ac --ac_species human -vv
```

Parameter description:

`--ontology_type GeneOntology`: instructs fastSemSim to load the embedded version of the GeneOntology

`--ac_species human`: instructs fastSemSim to load the embedded version of the Uniprot human gene ontology annotation corpus

`--tss Resnik`: selects the term semantic similarity to use

`--mix BMA`: selects the mixing strategy to use (required by pairwise semantic similarity measures, e.g. Resnik)

`--query_ss_type obj`: tells fastSemSim that the query will be a list of objects annotated with the ontology terms

`--query_input ac`: tells fastSemSim to use all the objects in the annotation corpus as query

`-vv`: the verbosity level

4.3.2 Exampe 2. Evaluate the SimGIC semantic similarity over the Disease Ontology between all the pairs of ontology terms, using the human DOA (Disease Ontology Annotation) dataset for establishing the Information Content statistics:

```
fastsemsim --ontology_type DiseaseOntology --query_ss_type term --tss SimGIC
--query_input ontology --ac_species human -vv
```

Parameter description:

`--ontology_type DiseaseOntology`: instructs fastSemSim to load the embedded version of the DiseaseOntology

`--ac_species human`: instructs fastSemSim to load the embedded version of the human Disease Ontology Annotation corpus

`--tss SimGIC`: selects the term semantic similarity to use (as SimGIC is a pairwise semantic similarity, the –mix parameter is not required)

`--query_ss_type term`: tells fastSemSim that the query will be a list of ontology terms

`--query_input ontology`: tells fastSemSim to use all the terms in the ontology as query. Use `--query_input ac` to use all the ontology terms associated to at least one object in the annotation corpus as query.

`-vv`: the verbosity level

4.3.3 Exampe 3. Evaluate the SimGIC semantic similarity over the Gene Ontology (loaded from file *GO_FILE*) and the Uniprot human gene ontology annotation corpus.

The query is a list of sets of objects loaded from the *QUERY_FILE* file. See `this file` for an example of a valid objset query file. The output will be written in the *OUTPUT_FILE* file. See `this file` for an example of the output generated.

```
fastsemsim --ontology_type GeneOntology --ontology_file GO_FILE --ac_species
human --query_ss_type objset --tss SimGIC --query_input file --query_file
QUERY_FILE -vv --output_file OUTPUT_FILE --query_mode list
```

*Parameter description *:

`--ontology_file GO_FILE`: the file with the ontology to load

`--ontology_type GeneOntology`: instructs fastSemSim about the type of ontology that will be loaded

`--ac_species human`: instructs fastSemSim to load the embedded version of the human Disease Ontology Annotation corpus

`--tss SimGIC`: selects the term semantic similarity to use

`--query_ss_type objset`: tells fastSemSim that the query will be a list of sets of objects.

`--query_input file`: tells fastSemSim to load the query from file. Requires the parameter –query_file to be specified.

`--query_file QUERY_FILE`: the name of the query file

`--query_mode list`: tells fastSemSim that the query is a list of elements. The pairwise semantic similarity between each element will be evaluated.

`--output_file OUTPUT_FILE`: the output file where the scores will be stored.

4.4 Troubleshooting

If you want to run fastSemSim Command Line Interface without installing the FastSemSim library, you can run the Python script `fastsemsim_cmdline.py` located in the folder `fastsemsim`. You should include the FastSemSim library in your PYTHONPATH before running the script, by running `export PYTHONPATH=path-to-fastsemsim-folder`.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

F