nafigator Documentation

Release 0.1.31

Willem Jan Willemse

Aug 22, 2021

CONTENTS:

CHAPTER

ONE

NAFIGATOR

DISCLAIMER - BETA PHASE

This package is currently in a beta phase.

1.1 to nafigate [naf-i-geyt]

v.intr, **nafigated**, **nafigating**

1. To process one of more text documents through a NLP pipeline and output results in the NLP Annotation Format.

1.2 Features

The Nafigator package allows you to store (intermediate) results and processing steps from custom made spaCy and stanza pipelines in one format.

- Convert text files to naf-files that satisfy the NLP Annotation Format (NAF)
  - Supported input media types: application/pdf (.pdf), text/plain (.txt), text/html (.html), MS Word (.docx)
  - Supported output formats: naf-xml (.naf.xml), naf-rdf in turtle-syntax (.ttl) and xml-syntax (.rdf) (experimental)
  - Supported NLP processors: spaCy, stanza
  - Supported NAF layers: raw, text, terms, entities, deps, multiwords
- Read naf-files and access data as Python lists and dicts

When reading naf-files Nafigator stores data in memory as lxml ElementTrees. The lxml package provides a Pythonic binding for C libaries so it should be very fast.

1.3 The NLP Annotation Format (NAF)

Key features:

- Multilayered extensible annotations;
- Reproducible NLP pipelines;
- NLP processor agnostic;
- Compatible with RDF

References:

- NAF: the NLP Annotation Format
- NAF documentation on Github

Current changes to NAF:

- a 'formats' layer is added with text format data (font and size) to allow text classification like header detection
- a 'model' attribute is added to LinguisticProcessors to record the model that was used
- all attributes of public are Dublin Core elements and mapped to the dc namespace
- attributes in a dependency relation are renamed 'from_term' and 'to_term' ('from' is a Python reserved word)

The code of the SpaCy converter to NAF is partially based on SpaCy-to-NAF

1.4 Installation

To install the package

```
pip install nafigator
```

To install the package from Github

```
pip install -e git+https://github.com/wjwillemse/nafigator.git#egg=nafigator
```

1.5 How to run

1.5.1 Command line interface

To parse a pdf, .docx, .txt or .html-file from the command line interface run in the root of the project:

```
python -m nafigator.cli
```

1.5.2 Function calls

To convert a .pdf, .docx, .txt or .html-file in Python code you can use:

```
from nafigator.parse2naf import generate_naf

doc = generate_naf(input = "../data/example.pdf",
                   engine = "stanza",
                   language = "en",
                   naf_version = "v3.1",
                   dtd_validation = False,
                   params = {'fileDesc': {'author': 'anonymous'}},
                   nlp = None)
```

- input: document to convert to naf document
- engine: pipeline processor, i.e. 'spacy' or 'stanza'
- language: for example 'en' or 'nl'
- naf_version: 'v3' or 'v3.1'
- dtd_validation: True or False (default = False)
- params: dictionary with parameters (default = {})
- nlp: custom made pipeline object from spacy or stanza (default = None)

The returning object, doc, is a NafDocument from which layers can be accessed.

Get the document and processors metadata via:

```
doc.header
```

Output of doc.header of processed data/example.pdf:

```
{
  'fileDesc': {
    'author': 'anonymous',
    'creationtime': '2021-04-25T11:28:58UTC',
    'filename': 'data/example.pdf',
    'filetype': 'application/pdf',
    'pages': '2'},
  'public': {
    '{http://purl.org/dc/elements/1.1/}uri': 'data/example.pdf',
    '{http://purl.org/dc/elements/1.1/}format': 'application/pdf'},
...
```

Get the raw layer output via:

```
doc.raw
```

Output of doc.raw of processed data/example.pdf:

```
The Nafigator package allows you to store NLP output from custom made spaCy and␣
↪stanza  pipelines with (intermediate) results and all processing steps in one␣
↪format.  Multiwords like in 'we have set that out below' are recognized (depending␣
↪on your NLP  processor).
```

Get the text layer output via:

```
doc.text
```

Output of doc.text of processed data/example.pdf:

```
[
  {'text': 'The', 'page': '1', 'sent': '1', 'id': 'w1', 'length': '3', 'offset': '0'},
  {'text': 'Nafigator', 'page': '1', 'sent': '1', 'id': 'w2', 'length': '9', 'offset
↪': '4'},
  {'text': 'package', 'page': '1', 'sent': '1', 'id': 'w3', 'length': '7', 'offset':
↪'14'},
  {'text': 'allows', 'page': '1', 'sent': '1', 'id': 'w4', 'length': '6', 'offset':
↪'22'},
...
```

Get the terms layer output via:

```
doc.terms
```

Output of doc.terms of processed data/example.pdf:

```
[
  {'id': 't1', 'lemma': 'the', 'pos': 'DET', 'type': 'open', 'morphofeat':
↪'Definite=Def|PronType=Art', 'targets': [{'id': 'w1'}]},
  {'id': 't2', 'lemma': 'Nafigator', 'pos': 'PROPN', 'type': 'open', 'morphofeat':
↪'Number=Sing', 'targets': [{'id': 'w2'}]},
  {'id': 't3', 'lemma': 'package', 'pos': 'NOUN', 'type': 'open', 'morphofeat':
↪'Number=Sing', 'targets': [{'id': 'w3'}]},
  {'id': 't4', 'lemma': 'allow', 'pos': 'VERB', 'type': 'open', 'morphofeat':
↪'Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin',
...
```

Get the entities layer output via:

```
doc.entities
```

Output of doc.entities of processed data/example.pdf:

```
[
  {'id': 'e1', 'type': 'PRODUCT', 'text': 'Nafigator', 'targets': [{'id': 't2'}]},
  {'id': 'e2', 'type': 'CARDINAL', 'text': 'one', 'targets': [{'id': 't28'}]}]
]
```

Get the entities layer output via:

```
doc.deps
```

Output of doc.deps of processed data/example.pdf:

```
[
  {'from_term': 't3', 'to_term': 't1', 'from_orth': 'package', 'to_orth': 'The',
↪'rfunc': 'det'},
  {'from_term': 't4', 'to_term': 't3', 'from_orth': 'allows', 'to_orth': 'package',
↪'rfunc': 'nsubj'},
  {'from_term': 't3', 'to_term': 't2', 'from_orth': 'package', 'to_orth': 'Nafigator',
↪ 'rfunc': 'compound'},
  {'from_term': 't4', 'to_term': 't5', 'from_orth': 'allows', 'to_orth': 'you', 'rfunc
↪': 'obj'},
...
```

Get the multiwords layer output via:

```
doc.multiwords
```

Output of doc.multiwords:

```
[
  {'id': 'mw1', 'lemma': 'set_out', 'pos': 'VERB', 'type': 'phrasal', 'components': [
    {'id': 'mw1.c1', 'targets': [{'id': 't37'}]},
    {'id': 'mw1.c2', 'targets': [{'id': 't39'}]}]}
]
```

Get the formats layer output via:

```
doc.formats
```

Output of doc.formats:

```
[
  {'length': '268', 'offset': '0', 'textboxes': [
    {'textlines': [
      {'texts': [
        {'font': 'CIDFont+F1', 'size': '12.000', 'length': '87', 'offset': '0', 'text
↪': 'The Nafigator package allows you to store NLP output from custom made spaCy and␣
↪stanza '
        }]
      },
      {'texts': [
        {'font': 'CIDFont+F1', 'size': '12.000', 'length': '77', 'offset': '88', 'text
↪': 'pipelines with (intermediate) results and all processing steps in one format.'
...
```

1.6 Adding new annotation layers

To add a new annotation layer with elements, start with registering the processor of the new annotations:

```
lp = ProcessorElement(name="processorname", model="modelname", version="1.0",␣
↪timestamp=None, beginTimestamp=None,   endTimestamp=None, hostname=None)

doc.add_processor_element("recommendations", lp)
```

Then get the layer and add subelements:

```
layer = doc.layer("recommendations")

data_recommendation = {'id': "recommendation1", 'subjectivity': 0.5, 'polarity': 0.25,
↪ 'span': ['t37', 't39']}

element = doc.subelement(element=layer, tag="recommendation", data=data_
↪recommendation)

doc.add_span_element(element=element, data=data_recommendation)
```

Retrieve the recommendations with:

```
doc.recommendations
```

1.6.1 Convert NAF file to RDF in turtle syntax

Just run:

```
python -m nafigator.convert2rdf
```

No ontology or vocabulary of NAF exists yet. For now, we map xml tags and attributes to RDF predicates using provisional prefixes and namespaces, for example base attributes are mapped to the prefix naf-base.

Below are some excerpts.

From the nafHeader:

```
_:nafHeader
    naf-base:hasFileDesc [
        naf-fileDesc:hasCreationtime "2021-05-24T11:29:44UTC"^^xsd:dateTime ;
        naf-fileDesc:hasFilename "data/example.pdf"^^rdf:XMLLiteral ;
        naf-fileDesc:hasFiletype "application/pdf"^^rdf:XMLLiteral ;
] ;
```

A word:

```
_:w1
    xl:type naf-base:wordform ;
    naf-base:hasText """The"""^^rdf:XMLLiteral ;
    naf-base:hasSent "1"^^xsd:integer ;
    naf-base:hasPage "1"^^xsd:integer ;
    naf-base:hasOffset "0"^^xsd:integer ;
    naf-base:hasLength "3"^^xsd:integer .
```

A term:

```
_:t1
    xl:type naf-base:term ;
    naf-base:hasType naf-base:close ;
    naf-base:hasLemma "the" ;
    naf-base:hasPos <http://purl.org/olia/olia.owl#Determiner> ;
    naf-morphofeat:hasDefinite "Def" ;
    naf-morphofeat:hasPronType "Art" ;
    naf-base:hasSpan [
        naf-base:ref _:w1
] .
```

An entity:

```
_:e1
    xl:type naf-base:entity ;
    naf-base:hasType naf-entity:PRODUCT ;
    naf-base:hasSpan [
        naf-base:ref _:t2
] .
```

A dependency:

```
_:t3 naf-rfunc:det _:t1
```

CHAPTER

TWO

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.1 Types of Contributions

2.1.1 Report Bugs

Report bugs at https://github.com/wjwillemse/nafigator/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

2.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

2.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

2.1.4 Write Documentation

Nafigator could always use more documentation, whether as part of the official nafigator docs, in docstrings, or even on the web in blog posts, articles, and such.

2.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/wjwillemse/nafigator/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.2 Get Started!

Ready to contribute? Here's how to set up *nafigator* for local development.

1. Fork the *nafigator* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/nafigator.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv nafigator
$ cd nafigator/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 nafigator tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

2.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/wjwillemse/nafigator/pull_requests and make sure that the tests pass for all supported Python versions.

2.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_nafigator
```

2.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

THREE

CREDITS

3.1 Development Lead

- Willem Jan Willemse <w.j.willemse@xs4all.nl>

3.2 Contributors

None yet. Why not be the first?

CHAPTER

FOUR

HISTORY

4.1 0.1.0 (2021-03-13)

- First release on PyPI.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search