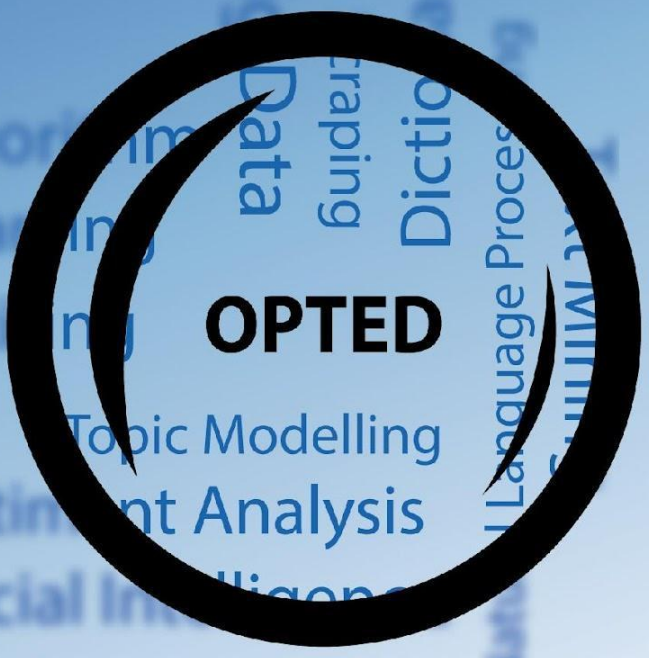

OPTED
Topic Modelling
Sentiment Analysis
Machine Learning
Lemmatizing
Data
Dictionary
Natural Language Processing

Disclaimer

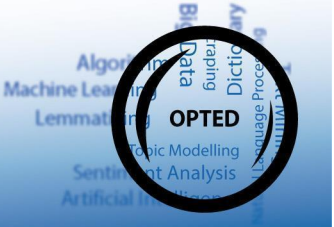
This project has received funding from the European Union's Horizon 2020 research & innovation programme under grant agreement No 951832. The document reflects only the authors' views. The European Union is not liable for any use that may be made of the information contained herein.

Dissemination level

Public

Type

Report



OPTED
Observatory for Political Texts in European Democracies:
A European research infrastructure

Text preprocessing module

Deliverable 7.3

Authors: Wouter van Atteveldt[1], Farzam Fanitabasi[1], Kasper Welbers[1],

[1] Department of Communication Science, Faculty of Social Science, Vrije Universiteit Amsterdam

- Executive Summary

The overall objective of WP7 is to establish routines and protocols to standardize the pre-processing of text, depending on the source and usage purpose. This work package focuses on assessing and providing prototypes of open science and open data structures in terms of data storage.

As the third step to achieve this objective, D7.3 introduces a collection of multilingual Natural Language Processing (NLP) tools. As text-as-data techniques are an increasingly important part of social science research, access to text (pre)processing tools is important for conducting valid and high-quality research. Although tools for English are widely available, tools for other languages are not always easy to find or run for users with less technical expertise. Moreover, the computers and environments used for analysis are not always suitable for processing large amounts of texts.

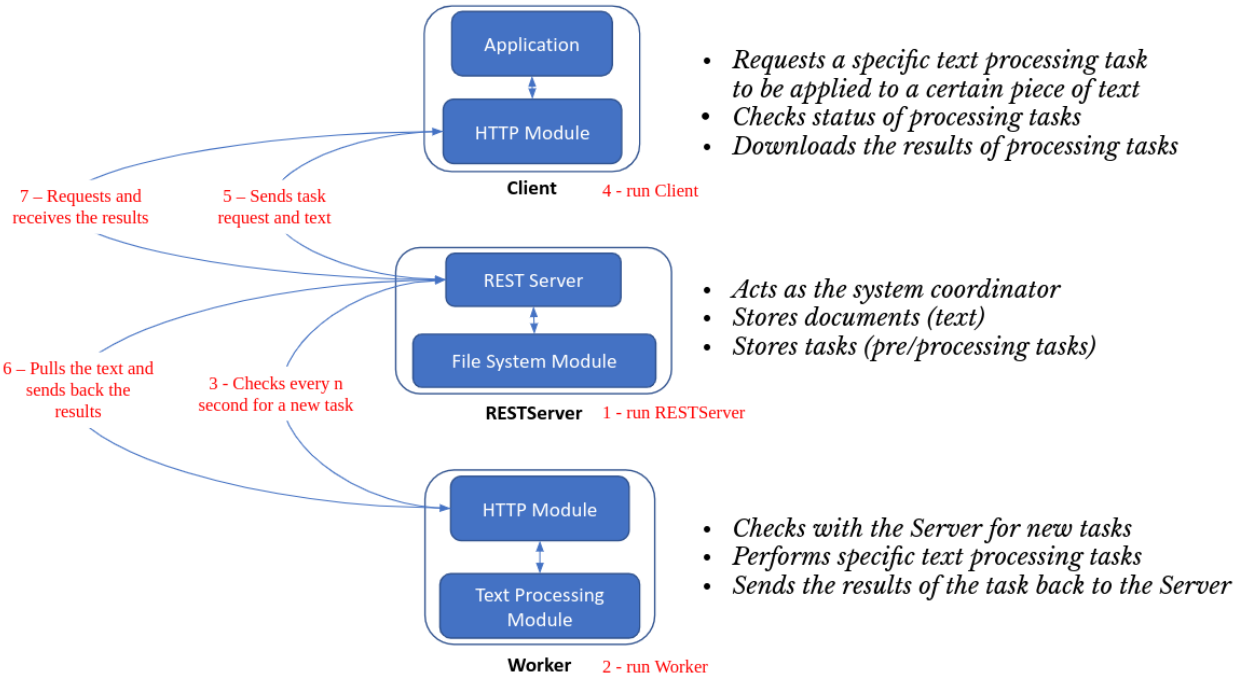
This deliverable hopes to mitigate these two problems by providing NLPipe, an easily installable collection of NLP tools that can be run either locally or via a server-worker architecture. By offering a standard interface for a number of open-source tools that offer multilingual processing, such as UDPipe, Spacy and gensim, NLPipe will make it easier for both non-technical users and power users to use state of the art processing tools. Moreover, it will allow the easy integration of other (possibly language-dependent) tools into the same interface, helping to mitigate the resource scarcity problem of many non-English languages.

This deliverable consists of the NLPipie codebase, which includes several *worker modules* that can be run in conjunction with the *server* module. This code can easily be run locally, be installed on a lab server, or as part of the AmCAT 4.0 text analysis infrastructure (D7.1). This codebase is published via the OPTED website and publicly accessible on the GitHub repository.

-

- NLPipe Overview

The figure below shows the overall architecture of NLPipe:



In short, the central role is played by the REST *server*, which acts as the system coordinator and stores documents to process and results. This server also allocates the tasks to workers and keeps track of progress.

The user connects to the server through one of the available *clients*, which can be the web client, a command line client included with the tool, or an API client in Python or R. Using this client, the user can request a document or set of documents to be processed, which causes it to be sent to the server. The user can also ask for the status of a job and download the results.

The main advantage of this division is that the server can be managed by more technical researchers or research engineers, while the client interface remains the same, no matter what political, legislative or journalistic texts are processed with whatever preprocessing tools. This also means that pre-processing stays reproducible between researchers who use the same server and is not dependent on the specific computational environment (operating system, processor architecture and installed software).

The actual processing is performed by the *workers*. Workers are tied to a specific processing tool (e.g. Spacy or UDPipe) and connect to the REST server to receive tasks. If a task is received, the worker runs the processing and sends the result back to the server.

- Installing NLPipe

NLPipe can be used as a standalone tool by installing it via *pip* (the Python package management tool). This will run the API on a local port, which can be interacted with via a web browser, the command line NLPipe interface, and/or API clients in Python, R, or other languages. This options works well for relatively small amounts of text or individual researchers.

For more demanding setups (see *using NLPipe* below), the server can be exposed through a standard web server, and workers can be run on the same or different machines.

- Using NLPipe

The client/server/worker model described above allows for NLPipe to be used in a very versatile way:

- A single researcher might just install nlpipe on their local computer, which would include the server and workers and allow the researcher to directly process texts from e.g. R or Python. Used in this way, the tool is simply a standalone service for the user.
- In a research lab, one central computer might be used as an NLPipe server, and multiple researchers could share this resource and the server could also function as a local storage of results (especially if connected to an AmCAT interface).
- For very demanding tasks, e.g. full syntactic parsing of hundreds of thousands or millions of documents, a central computer might act as server, and workers can be activated on multiple computers to each process part of the corpus. As workers pull jobs from the server one by one, jobs are automatically distributed over the workers. Since the workers do not need to be reachable from outside (the workers communicate with the server, not the other way around), these workers can be run on e.g. idle workstations or High Power Computing (HPC) cluster nodes.
- Given access to the API of AmCAT as described in D7.1, NLPipe can autonomously pull relevant selected documents, process them and upload the results to the storage server.

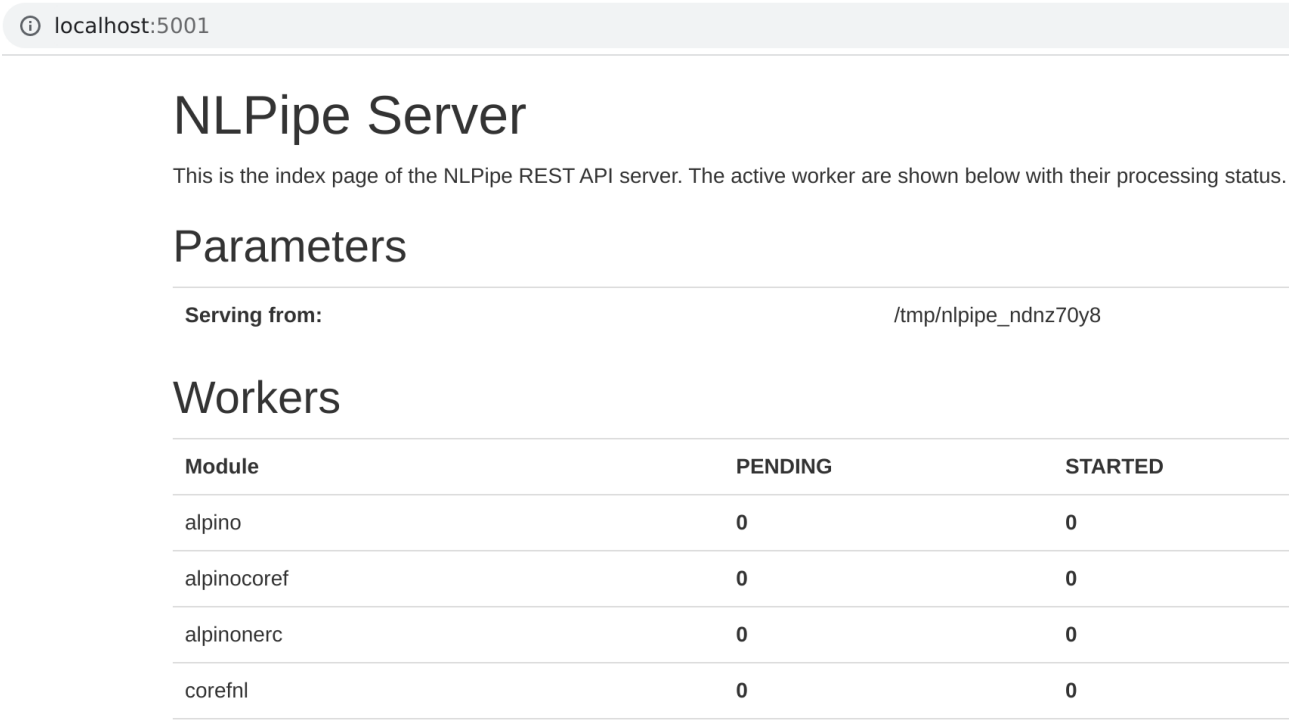
- Technologies used

All code is written in Python (>=3.6) using only open source dependencies that are available in the PyPI repository. The REST server uses Flask (web server) and Peewee (DB manager)[1]

- NLPipe Components

- NLPipe REST server

All functionality of NLPipe is exposed through a REST server, which allows clients to assign documents for processing, check status, and download results. This server also exposes an informational HTML index which can be used to check which tools are available and processing status. The screenshot below shows this index page running on a local installation:



[1] This will be refactored to fastAPI+SQLAlchemy/Elastic for a next version to be released via GitHub

- NLPipe workers

NLPipe workers are responsible for processing documents using a single tool (e.g. Spacy). The worker connects with the server and asks for available tasks. If a task is available, it retrieves it from the server, processes it, and stores the result. Note that this is generally not exposed to the end user, they simply see that their task is being processed. By separating the workers from the server it is possible to run workers on different or multiple machines, e.g. to connect the server with workers on a High Power Computing resource.

As the user does not interact directly with the workers, there is no graphical user interface. The session below shows running a Spacy worker locally, which automatically downloads the correct language model and processes a document:

```
$ python -m nlpipe.Workers.worker http://localhost:5001 spacy
[2022-06-23 17:14:54,656 root INFO ] Workers active and waiting for input
[2022-06-23 17:14:59,687 root INFO ] Received task spacy/0x9c..53 (25 bytes)
Collecting en-core-web-sm==3.3.0
[2022-06-23 17:14:59,687 root INFO ] Task completed: spacy/0x9c..53
```

- NLPipe clients

To interact with NLPipe, a user uses one of the available *clients* to initiate processing, check status, and download results. The session below shows the command line client used to parse a small sentence using Spacy:

```
$ python -m nlpipe.Clients.client http://localhost:5001 spacy process "Wouter lives in Amsterdam!"
0x9c..53

$ python -m nlpipe.Clients.client http://localhost:5001 spacy doc_status 0x9c..53
DONE

$ python -m nlpipe.Clients.client http://localhost:5001 spacy result 0x9c..53
text       lang_  left_edge  right_edge  ent_type_  lemma_     morph           pos_   dep_
Wouter     en     Wouter     Wouter                 Wouter     Number=Sing     PROPN  nsubj
lives      en     Wouter     !                      live       Number=Sing|..  VERB   ROOT
in         en     in         Amsterdam              in                         ADP    prep
Amsterdam  en     Amsterdam  Amsterdam   GPE        Amsterdam  Number=Sing     PROPN  pobj
!          en     !          !                      !          PunctType=Peri  PUNCT  punct
```

-

- Included tools

The currently released version of NLPipe (0.58) includes the following tools:

Language-independent tools:

- GenSim [https://radimrehurek.com/gensim/], a widely-used module for topic modeling and word embeddings

Multilingual tools:

- UDPipe [https://ufal.mff.cuni.cz/udpipe] - A widely used open source toolkit for tagging and parsing in 63 different languages
- Spacy [https://spacy.io] - Another widely used open source toolkit for parsing in 22 languages.
- NewsReader [http://www.newsreader-project.eu/] - A pipeline developed in the ERC NewsReader project that includes multiple processing steps to parse and enrich data, including disambiguation and semantic role labeling.

- CoreNLP [https://stanfordnlp.github.io/CoreNLP/] - A pipeline built and maintained by the Stanford NLP group that offers preprocessing in 8 languages.

Language-specific tools:

- Alpino [http://www.let.rug.nl/vannoord/alp/Alpino/] - a Dutch syntax parser developed at the University of Groningen
- Frog [http://languagemachines.github.io/frog/] - a Dutch part-of-speech tagger and lemmatizer
- ParZu [https://github.com/rsennrich/ParZu] - a German syntax parser developed at the University of Zurich

The selection was made based on what is currently widely used and what we consider to be a good fit for multilingual pre-processing. However, NLPipe was created with flexibility in mind and the currently available tools are also meant as a blueprint to be extended in the future.

- Contributing to NLPipe and license information

NLPipe is distributed under the MIT license[2], which is a widely used permissive Free and Open Source license approved by the OSI (Open Source Initiative). In brief, this license allows anyone to use, modify, and distribute the software in any way, including in commercial settings. We actively encourage contributions to NLPipe and all other CCS-Amsterdam software through GitHub issues and pull requests.

In particular, regular users are encouraged to contribute to NLPipe by using and testing the software, and reporting any issues, bugs, or feature requests as GitHub issues. Users are also encouraged to contribute to documentation and examples, either on the GitHub page or as external resources. Technical users are also encouraged to contribute bugfixes or enhancements as GitHub pull requests. In particular, requests that contribute to fixing existing bugs, adding more tools, and improving ease of use and accessibility are greatly appreciated.

[2] https://github.com/ccs-amsterdam/nlpipe/blob/main/LICENSE