Typecraft XML Python Documentation

Release 0.11.0

Tormod Haugland

Jul 09, 2018

Contents

Contents:

CHAPTER 1

Typecraft Python

This repository contains an IGT model based on the Typecraft IGT format. It also contains a simple CLI for performing various NLP tasks, interfacing with both NLTK and other tools such as the TreeTagger.

- Free software: MIT license
- Full Documentation: https://typecraft_python.readthedocs.io.

1.1 Installation

pip install typecraft_python

1.2 Features

- Parsing of the Typecraft XML format.
- **Manipulation of the Typecraft IGT model format.**
  - Integrating with NLTK
  - Integrating with TreeTagger
- Provides a CLI that can be used to load, convert and manipulate raw text and Typecraft XML files.

1.3 Usage

```
Usage: tpy [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.
```

(continues on next page)

(continued from previous page)

```
Commands:
  convert
  ntexts   This command lists the number of texts in a...
  raw
  xml
```

1.3.1 Examples

Load a raw file, tokenize and tag it, and output xml (to stdout):

```
$ tpy raw your_file.txt
```

To save to a file

```
$ tpy raw your_file.txt -o output.xml
# or
$ tpy raw your_file.txt > output.xml
```

To tag using a specific tagger:

```
$ tpy raw your_file.txt --tagger=tree  # Tags using the tree tagger
```

To load a Typecraft xml file and tag it:

```
$ tpy xml your_file.xml --tag --tagger=nltk -o tagged_output.xml
```

1.4 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Typecraft XML Python, run this command in your terminal:

```
$ pip install typecraft_python
```

This is the preferred method to install Typecraft XML Python, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Typecraft XML Python can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/Typecraft/tc_xml_python
```

Or download the tarball:

```
$ curl  -OL https://github.com/Typecraft/tc_xml_python/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

The system can be used in two ways: As a CLI, and as a library.

3.1 CLI

The CLI currently has 4 active commands:

- raw
- xml
- ntexts
- par

raw loads raw texts, and performs a number of operations on it. It will always convert the result to a TC-XML file. **xml** loads TC-xml files, and performs a number of operations on it. **ntexts** loads TC-xml files, and reports how many text-objects exist in the file. **par** parses parallel corpora.

All file inputs in the commands accepts "-" as input, which specifies that the input should be read from stdin. The examples in *Combined examples* gives some examples of this.

3.1.1 raw

The **raw** command always loads raw text, and always outputs TC-XML files.

```
Usage: tpy raw [OPTIONS] [INPUT]...

Options:
  --sent-tokenize / --no-sent-tokenize
                                  Will sentence tokenize if true.
  --tokenize / --no-tokenize      Will tokenize if true.
  --tag / --no-tag                Will tag if true.
  --tagger TEXT                   The tagger to use.
```

(continues on next page)

(continued from previous page)

```
  --title TEXT                    Title to attach to generated texts.
  --language TEXT                 The language of the input text(s).
  --meta <TEXT TEXT>...           Metadata to attach to generated text(s)
  --tagset TEXT                   If set, the tags in the output will be
                                  converted into this tagset.
  -o, --output PATH               If given, the output will be written to this
                                  file, instead of stdout.
  --help                          Show this message and exit.
```

By default, the command will perform

- Sentence-tokenization
- Tokenization
- Tagging using NLTK
- All the above assuming the language is English.

Examples

Load a raw file, tokenize and tag it, and output xml (to stdout):

```
$ tpy raw your_file.txt
```

To save to a file

```
$ tpy raw your_file.txt -o output.xml
# or
$ tpy raw your_file.txt > output.xml
```

To tag using a specific tagger:

```
$ tpy raw your_file.txt --tagger=tree  # Tags using the tree tagger
```

Attach "Annotator" metadata:

```
$ tpy raw your_file.txt --meta Annotator "Tormod Haugland"
```

Tags a german text

```
$ tpy raw your_file.txt --language=de
```

Tags a german text using the TreeTagger and converts all tags to the Typecraft tagset:

```
$ tpy raw your_file.txt --tagger=tree --tagset=tc --language=de
```

Suppose you have the file *input.txt* with the following contents:

```
Ich bin glucklich.
```

You now run the command

```
$ tpy raw input.txt --tagger=tree --language=de --tagset=tc
```

Your output (after prettifying) will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<typecraft xmlns="http://typecraft.org/typecraft" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance" xsi:schemaLocation="https://typecraft.org/typecraft.xsd">
   <text lang="de">
      <title>Automatically generated text from tpy</title>
      <titleTranslation />
      <body />
      <phrase valid="EMPTY">
         <original>Ich bin glucklich.</original>
         <translation />
         <translation2 />
         <globaltags id="1" tagset="DEFAULT" />
         <description />
         <word head="false" text="Ich">
            <pos>PN</pos>
            <morpheme baseform="ich" meaning="" text="Ich" />
         </word>
         <word head="false" text="bin">
            <pos>AUX</pos>
            <morpheme baseform="sein" meaning="" text="bin" />
         </word>
         <word head="false" text="glücklich">
            <pos>ADJ</pos>
            <morpheme baseform="glücklich" meaning="" text="glücklich" />
         </word>
         <word head="false" text=".">
            <pos>PUN</pos>
            <morpheme baseform="." meaning="" text="." />
         </word>
      </phrase>
   </text>
</typecraft>
```

3.1.2 xml

The **xml** command loads a TC-XML file, and performs a number of specified operations on it.

```
Usage: tpy xml [OPTIONS] [INPUT]...

Options:
  --tokenize / --no-tokenize  Will re-tokenize all phrases if true.
  --tag / --no-tag            Will tag if true.
  --tagger TEXT               The tagger to use.
  --split INTEGER             If greater than 1, the output will be split into
                              the given value number of texts.
  --merge / --no-merge        If true, will merge all files.
  --title TEXT                Title to attach to generated texts.
  --override-language TEXT    If set, will override the language used in all
                              calculations and set the language for all texts.
  --meta <TEXT TEXT>...       Metadata to attach to generated text(s)
  --tagset TEXT               If set, the tags in the output will be converted
                              into this tagset.
  -o, --output PATH           If given, the output will be written to this
                              file, instead of stdout.
  --help                      Show this message and exit.
```

By default the command will do nothing but re-output the input. The "-o" flag behaves identically to the one in **raw**.

Notes

- Split will split into the given number of files, even if the given number is larger than the number of phrases.

Examples

Load a text and splits it into 10 smaller texts (all contained in one file):

```
$ tpy xml your_file.xml --split 10
```

Load a text and convert the tagset:

```
$ tpy xml your_file.xml --tagset=tc
```

Tag or re-tag a text:

```
$ tpy xml your_file.xml --tag --tagger=tree
```

Change language and set some metadata:

```
$ tpy xml --override-language=nob \
    --meta Annotator "Tormod Haugland" \
    --meta "Content description" "This is some cool content"
```

3.1.3 par

par will parse parallel corpora files. Currently there is only one supported format, named *continuous* or *linear*. The output is always Typecraft XML. This format requires there to to be *n* consecutive lines in the file, one per language, for each phrase that is to be translated.

Note that the Typecraft XML format only supports two translation tiers.

```
Usage: tpy par [OPTIONS] [INPUT]...
↪                                                                               ␣
↪                                                                               ␣
  The `par` command attempts to parse raw text as parallel corpora.
↪                                                                               ␣
↪                                                                               ␣
  The input is one or more files containing raw text, in some parallel
↪                                                                               ␣
  format.
↪                                                                               ␣
↪                                                                               ␣
Options:
↪                                                                               ␣
  -f, --format TEXT        The format of the parallel file.
↪                                                                               ␣
  -n, --num-langs INTEGER  The number of languages present.
↪                                                                               ␣
```

(continues on next page)

(continued from previous page)

```
  -o, --output PATH        If given, the output will be written to this file,
↪                                                                            |
                           instead of stdout.
↪                                                                            |
  --help                   Show this message and exit.
```

Examples

Given the file *input.txt* with the contents below:

```
Hi this is a nice sentence.
Hei dette er en fin setning.
This is sentence number two.
Dette er setning nummber to.
```

Which is a parallel corpus file with two languages (Norwegian and English). We can call the command

```
$ tpy par -n 2 input.txt
```

The resulting output will be Typecraft XML with a single text with two phrases. The phrases will not be tokenized, with the appropriate amount of free translations tiers set.

3.1.4 ntexts

ntexts will output the number of texts in a TC-XML file.

Examples

```
Usage: tpy ntexts [OPTIONS] INPUT

  This command lists the number of texts in a TCXml file. :param input:
  :return:

Options:
  --help  Show this message and exit.
```

Examples

```
$ tpy ntexts input_with_10_texts.xml
10
```

3.1.5 Combined examples

Load and treat a raw file, then split it into 10 texts:

```
# The "-" in the xml command reads the piped input
$ tpy raw input.txt | tpy xml - --split 10
```

Load and treat a raw file, then merge it with an existing files texts.

```
$ tpy raw append_this.txt | tpy xml - to_this.xml --merge
```

Make sure ntexts behaves correctly:

```
$ tpy raw input.txt | tpy xml - --split 50 | tpy ntexts -
100
```

Merge files then re-split:

```
$ tpy xml corpus{1..100}.xml --merge | tpy xml - -split 1000
```

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/Typecraft/typecraft_python/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Typecraft/typecraft_python/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/typecraft_python.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 typecraft_python
$ cd typecraft_python/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b feature/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 typecraft_python tests
$ py.test
$ 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 feature/name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 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 2.7, 3.5 and 3.6. Check https://travis-ci.org/Typecraft/typecraft_python/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_typecraft_python
```

CHAPTER 5

Credits

5.1 Development Lead

- Tormod Haugland <tormod.haugland@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.1 (2016-08-15)

- Fixed some small bugs.

6.2 0.1.0 (2016-08-14)

- **First release. Added main bulk of initial code:**
  - Parser works in its most basic inception and parses TC-XML documents into its object-tree

CHAPTER 7

Indices and tables

- genindex
- modindex
- search