**CATMA – ANNOTATION WORKFLOW**

**Kaushik Sadhu – 22.08.2020 | University of Stuttgart**

RwTagger tool: <https://github.com/redewiedergabe/tagger>

These recognizers were developed by the DFG-funded project "Redewiedergabe - eine literatur- und sprachwissenschaftliche Korpusanalyse" (Leibniz Institute for the German Language / University of Würzburg) ([www.redewiedergabe.de](http://redewiedergabe.de)) and (mostly) trained on data from [Corpus Redewiedergabe](https://github.com/redewiedergabe/corpus).

They can be used to automatically detect and annotate the following 4 types of speech, thought and writing representation in German texts, direct, free indirect ('erlebte Rede'), indirect and reported.

***Recognizer models:***

1. ***KONVENS 2020 models***

These are the models discussed in the[***KONVENS 2020 paper.***](http://ceur-ws.org/Vol-2624/paper5.pdf)

1. **Alternative models (KONVENS 2020)**

As an alternative, we also provide the most successful models using an alternative language embedding. These were used in the comparisons in the [KONVENS 2020 paper](http://ceur-ws.org/Vol-2624/paper5.pdf).

**SETUP ENVIRONMENT TO RUN THE TAGGER:**

A Python environment (Python 3.7.0 +), with the necessary modules has to be set up. We provide a requirements file and give some instructions how to set up a [Python virtual environment](https://github.com/redewiedergabe/tagger#environment) to facilitate this.

**Setup with Anaconda Python under Windows**

* Make sure the you have at least Python 3.7.0 installed (newer versions should work as well)
* If you have no experience with Python, we recommend installing [Anaconda Python](https://www.anaconda.com/); then proceed in the 'Anaconda Powershell Prompt' console to avoid problems with path variables (NOTE: Anaconda has two different 'Prompt' consoles. These instructions assume you use 'Anaconda **Powershell** Prompt')
* If Python **virtualenv** is not already installed, execute the following code in the console:

pip install virtualenv

* Download this Github project
* Change into the directory **tagger** and execute the following code
  + **NOTE:** The code below installs the **CPU version of pytorch**, which works for all computers. If you want to use a GPU instead, uncomment the alternative line in the code. However, for the GPU to work with pytorch your also have to make sure you have CUDA installed and configured correctly. For this, please refer to other guides, e.g. [this one](https://medium.com/datadriveninvestor/installing-pytorch-and-tensorflow-with-cuda-enabled-gpu-f747e6924779).
    - virtualenv venv
    - cd venv
    - .\Scripts\activate
    - # --> you should now see '(base) (venv)' at the beginning of your prompt line
    - # install pytorch:
    - # if your computer does not have a GPU:
    - pip3 install torch==1.3.1+cpu torchvision==0.4.2+cpu -f https://download.pytorch.org/whl/torch\_stable.html
    - # alternatively, if your computer has a GPU you want to use, remove the line above and uncomment the following:
    - # pip3 install torch===1.3.1 torchvision===0.4.2 -f https://download.pytorch.org/whl/torch\_stable.html
    - # install all other required modules:
    - pip install -r ..\requirements.txt
    - # change to the rwtagger directory
    - cd ..\rwtagger
* To tokenize input texts, you need additional libraries for the NLTK module. We recommend installing them in the interactive mode:
  + type python to open the Python interpreter. Then type the following:

import nltk

nltk.download('punkt')

exit()

* You can now execute the recognizers in this console window (after you have downloaded the Recognizer models). Make sure that **venv** is the active environment (should be visible in your prompt line). If you want to switch back to your regular Python environment, type:

Deactivate

**RUN THE MODELS:**

After placing the respective models that you want in the model’s folder, place the text (plain utf-8) inside input\_dir folder. Then follow below steps.

The directory **test** contains some folders you can use for testing. Note that the data in the output folder will be overwritten whenever you call the script again.

python rwtagger.py input\_dir output\_dir

simplest call: expects an input folder of plain text UTF-8 coded files, tags all 4 STWR types and outputs tsv files with columns for each type; runs the tagger on CPU (Note: This call might take some time, as it loads and executes all 4 taggers one after the other)

python rwtagger.py input\_dir output\_dir -t direct indirect -conf

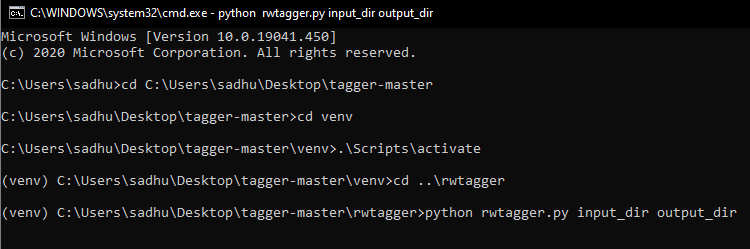
annotates only the types direct and indirect; outputs confidence values for each annotation; expects an input folder of plain text UTF-8 coded files

python rwtagger.py input\_dir output\_dir -gpu -f tsv

runs the tagger on GPU; input format is not plain text but tsv (similar to the output format of the tagger: one token per line and markers for sentence start; column names must be 'tok' and 'sentstart'); annotates all 4 STWR types

python rwtagger.py input\_dir output\_dir -m test -t reported

runs the tagger and also calculates test scores for the STWR type reported; input files must be tsv format and contain a column called 'reported' containing the gold standard annotations.



**USEFUL LINKS AND RESOURCES:**

**[1] CATMA:** (<https://github.com/mpetris/catma>) | <https://app.catma.de/catma/>

**[2] CATMA TEI-XML format** : (<https://catma.de/documentation/tei-export-format/>)

**[3] Gitlab**: <https://catma.de/git-access/>

**[4] GITLAB Community Edition – Docker:** <https://hub.docker.com/r/gitlab/gitlab-ce/>