

A practical workshop on automatic morpho-syntactic annotation of large language corpora using the Universal Dependencies framework

15th of April 2024 University of Tartu, Estonia

Hanna-Mari Kupari hmknie@utu.fi

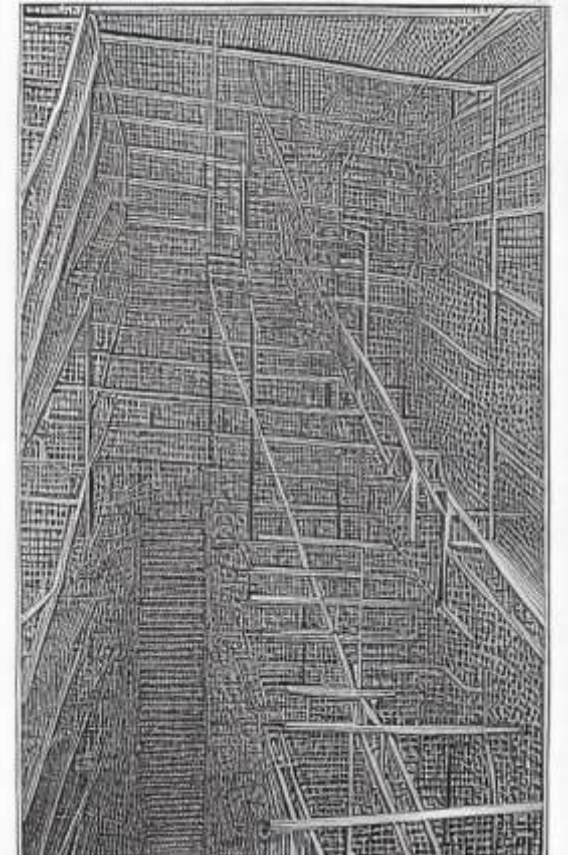
➤ Workshop Day 2

➤ Recap

A short introduction to machine learning

What technology are the parsers based on?

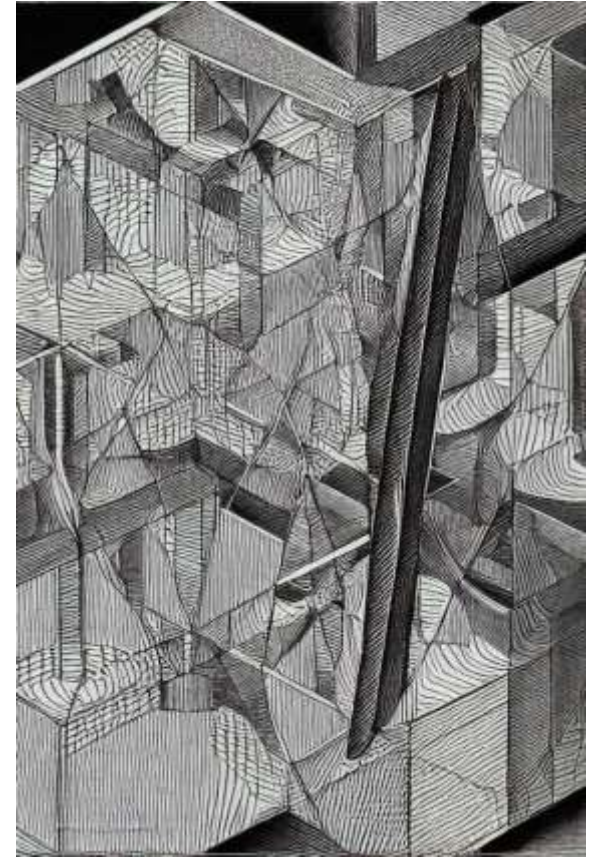
What are possible tools for parsing your own data?



Deep AI

Shortly about today's aims

- Working in a workshop style
- Some basic points from the lecture on Monday
- How the models learn to analyze language?
- Two tools to work with: Trankit and Stanza



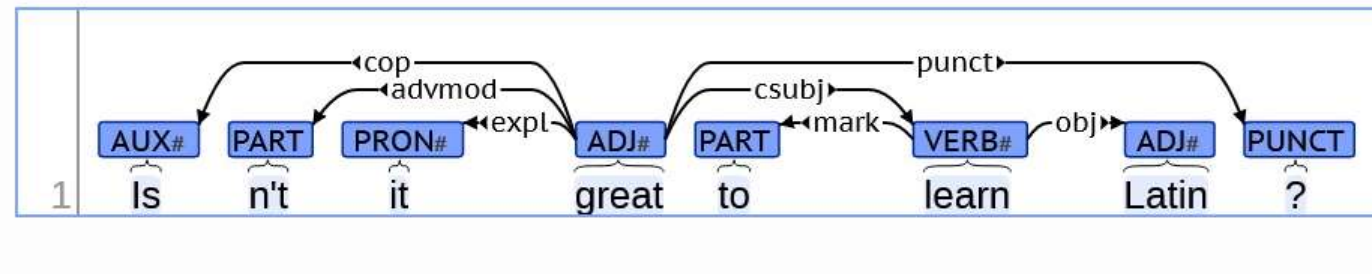


Recap (and introductions)

- Say your name (so I'll try and remember)
- What do you remember from yesterday's lecture?

What is automatic parsing?

- “to analyze the input sentence in terms of grammatical constituents, identifying the parts of speech and their syntactic relations”
- producing tokens: basic blocks of text (words, characters, or subwords)



Dependency grammar

- grammatical theory that is based on the dependency relation
- words are connected to each other by directed links
- Finite verb is the starting point (i.e. root)
- All other words are either directly or indirectly connected to the verb in terms of the directed links i.e. dependencies
- A dependency structure is determined by the relation between a word (a head) and its dependents.
(https://en.wikipedia.org/wiki/Dependency_grammar)

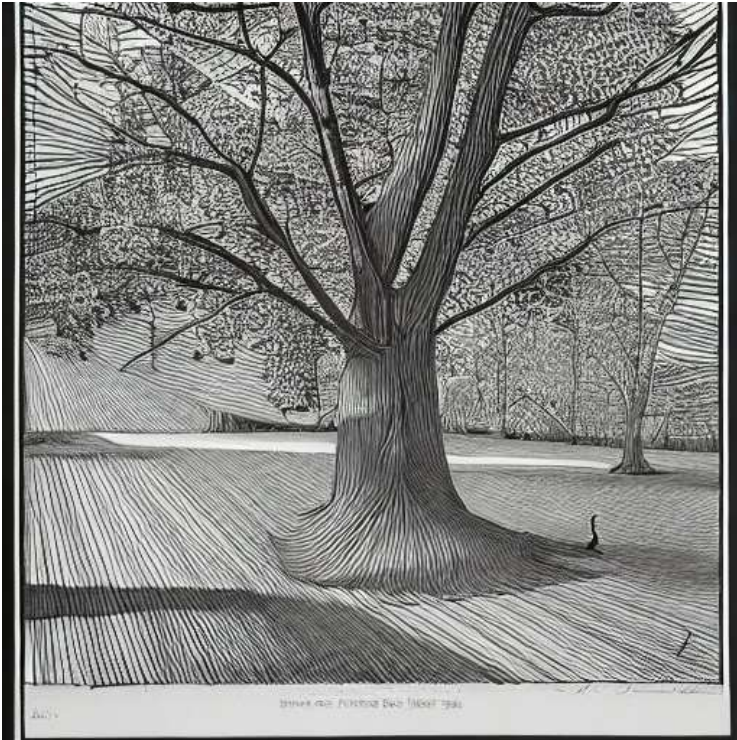
Universal Dependencies

<https://universaldependencies.org/>

- framework for consistent annotation of grammar (parts of speech, morphological features, and syntactic dependencies) across different human languages
- open community effort:
 - 300 contributors
 - producing nearly 200 **treebanks**
 - over 100 languages

https://applied-language-technology.mooc.fi/html/notebooks/part_iii/02_universal_dependencies.html

What is a treebank?



- In linguistics, a treebank is a parsed text corpus that annotates syntactic sentence structure
- The term **parsed corpus** is often used interchangeably with the term treebank

CoNLL-U output

<https://lindat.mff.cuni.cz/services/udpipe/>

generator = UDPipe 2, <https://lindat.mff.cuni.cz/services/udpipe>

udpipe_model = latin-**perseus**-ud-2.10-220711

sent_id = 1

text = Puella canes amat.

1 Puella puella NOUN

2 canes canis NOUN

2 amat amo VERB

3 4 . . PUNCT

The screenshot shows the UDPipe web interface. At the top, there is a disclaimer about the service being for testing and a license notice. Below this, the 'Model' is set to 'latin-perseus-ud-2.10-220711'. The 'Actions' section has 'Tag and Lemmatize' and 'Parse' checked. The 'Advanced Options' section is collapsed. The 'Input Text' field contains the sentence 'Puella canes amat.'. The 'Process Input' button is highlighted. Below the button, the 'Output Text' tab is selected, showing the sentence 'Puella canes amat.'. The 'Show Trees' tab is also visible, showing a syntax tree for the sentence. The tree structure is: root (NOUN) -> amat (VERB) -> Puella (NOUN) and canes (NOUN). The root node is also labeled 'punct'.

+ other columns with information about syntax,
semantic roles Proto-Indo-European
roots etc

Case study: expressions of violence

- Penitentiary documents of the late middle ages
- How do people report cases of violence to the Vatican office when asking for an absolution
- 20 000 word corpus



Semantic roles can also be annotated

- Basic semantic roles in a sentence can be for example agent (the one who is actively doing something) and patient (the object of actions)
- In the penitentiary documents people have the objective of showing their own action in a good light
- What kind of verbs are used when the suppliant is the one reporting about their own actions and the actions of other parties involved



Case study

- A case study combining parsers with traditional close reading
- Use a parser to find all the verbs in a corpus
 - A search for POS column with POS label VERB
- Close reading this dataset and picking verbs related to violence (e.g. to hit, to shoot with an arrow, punch, kill)
- Taking these sentences and adding semantic roles (close reading)
- The annotation can be kept consistent and tidy with digital methods

BRAT

annotation tool

https://brat.nlpab.org/introduction.html


home | introduction | examples | features | manual | site map | contact | brat

mini-introduction to brat

brat is a web-based tool for text annotation; that is, for adding notes to existing text documents.

brat is designed in particular for structured annotation, where the notes are not freeform text but have a fixed form that can be automatically processed and interpreted by a computer.

the following screenshot shows a simple example where a sentence has been annotated to identify mentions of some real-world entities (things) and their types, and a relation between two.




example annotations (following in part the [ACE 2005](#) entity and relation annotation guidelines)

this example illustrates two basic categories of annotation:

- **text span** annotations, such as those marked with the *Organization* and *Person* types in the example
- **relation** annotations, such as the *family* relation in the example

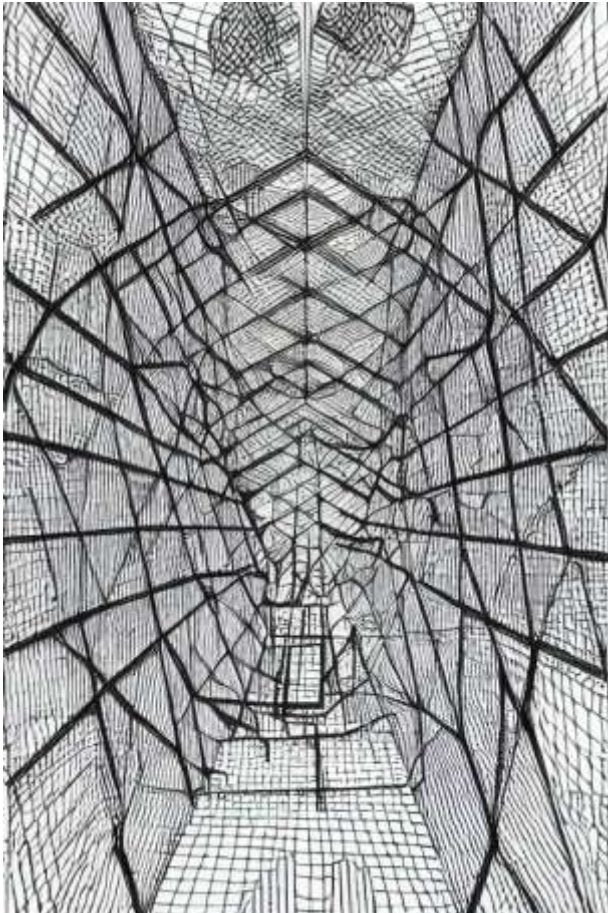
the simple typed text span category is suitable for creating annotations for [named entity recognition](#), and binary relations for simple relational [information extraction](#) tasks, among others.

brat also supports the annotation of **n-ary associations** that can link together any number of other annotations participating in specific roles. This category of annotation can be used for example for event annotation, such as *TRANSFER* in the following example:



Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou and Jun'ichi Tsujii (2012). brat: a Web-based Tool for NLP-Assisted Text Annotation. In *Proceedings of the Demonstrations Session at EACL 2012*.

Modern parsing tools



- Based on machine learning
- Natural language processing uses the two types
 - Supervised machine learning
 - Unsupervised

ChatGPT 3.5

In summary, supervised learning relies on labeled data with predefined output labels, while unsupervised learning works with unlabeled data to discover patterns or structures within the data itself.



ChatGPT 3.5 ▾



You

what is the difference between supervised and unsupervised machine learning



ChatGPT

Supervised and unsupervised machine learning are two fundamental approaches to training machine learning models, and they differ primarily in the type of data they use and the learning objectives they pursue.

1. **Supervised Learning:**

- In supervised learning, the algorithm is trained on a labeled dataset, meaning that each input data point is associated with a corresponding output label.
- The goal of supervised learning is to learn a mapping function from input variables to output variables, based on the labeled training data.
- During training, the algorithm adjusts its internal parameters to minimize the difference between its predicted output and the actual labels provided in the training data.
- Examples of supervised learning algorithms include linear regression, logistic regression, decision trees, support vector machines (SVM), and neural networks.
- Common applications of supervised learning include classification (predicting discrete labels) and regression (predicting continuous values).

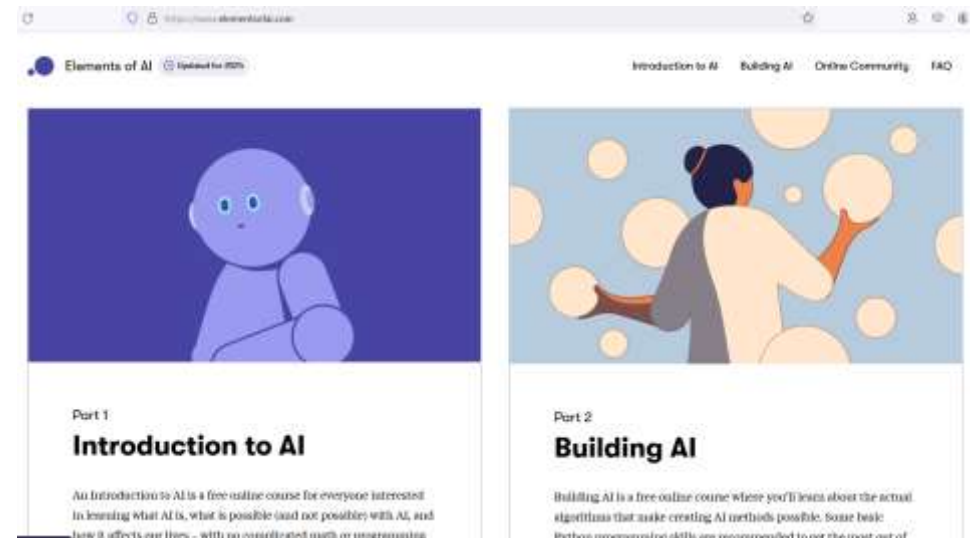
2. **Unsupervised Learning:**



Some practical examples

- Supervised
- Based on given labels
- A **parser** trained on Gold Standard training data
- Named-Entity Recognition
- Unsupervised
- Based on statistics
- **Topic modeling**
 - **is a type of statistical modeling used to identify topics or themes within a collection of documents.** It involves automatically clustering words that tend to co-occur frequently across multiple documents, with the aim of identifying groups of words that represent distinct topics.
(upenn.edu)

A free MOOC to learn more about AI



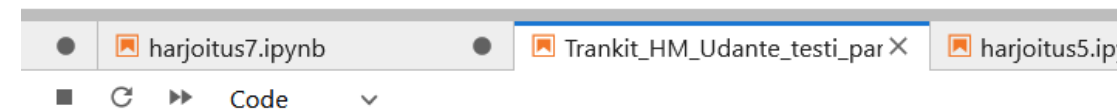
Today's main focus: HOW

- Trankit
- Stanza
- UD pipe 2 on web interface
- Main idea is to choose a tool (or do a comparison between the models?)
- You need just need some “cleaned” text

Advanced level

- Use Trankit
- Or perhaps even train Trankit?
 - Combining the existing UD treebanks to form new training data
- Needs some computational power, I have used CSC Puhti

/node/r18c01.bullx/61715/lab/tree/kupariha/trankit/HM_udante/Trankit_HM_Udante_testi_par



```
<it
```

```
object_2008402/kupariha/venv/lib64/python3.9/site-packages/tqdm/auto.py:21: TqdmW
tps://ipywidgets.readthedocs.io/en/stable/user_install.html
nernotebook import tqdm as notebook_tqdm
```

```
ify_customized_pipeline(
    lang='customized-mwt', # pipeline category
    cache_dir='./save_dir', # directory used for saving models in previous steps
    embedding_name='xlm-roberta-base' # embedding version that we use for training our cust
```

```
line is ready to use!
alized as follows:
```

```
from import Pipeline
p = Pipeline(lang='customized-mwt', cache_dir='./save_dir')
```

```
https://raw.githubusercontent.com/HannaKoo/Latin-variability/main/harmonization/f
```

```
udante-ud-test.conllu > test.conllu
```

```
/conllu_to_text.pl --lang la < test.conllu > test.txt
```

```
'udante-ud-test.conllu' already there; not retrieving.
```

```
actions for processing the trankit output format
```

```
def line(token):
    return "\n".join([str(token["id"]), token["text"], token.get("lemma", "_"), token.get("up", "_")])

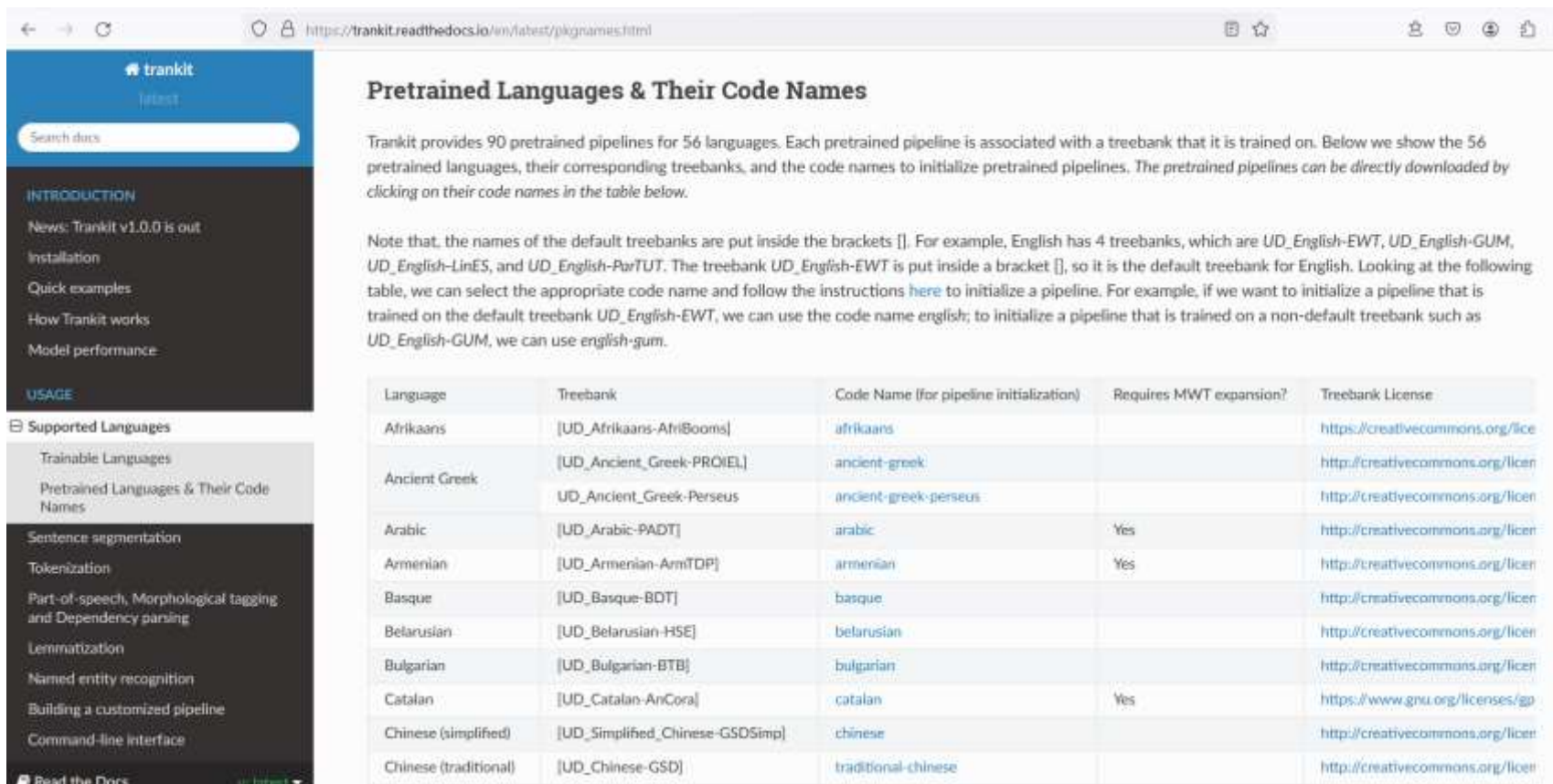
def conllu(d):
    return "\n".join([line(token) for token in d])
```

```
def conllu_to_text(conllu_file, text_file, encoding="UTF8"):
    with open(conllu_file, "r", encoding=encoding) as f:
        conllu_data = f.read()
        conllu_lines = conllu_data.split("\n")
        conllu_lines = [line for line in conllu_lines if line]
```

Combining treebanks to improve performance

Training set	ITTB	LLCT	Perseus	Proiel	UDante
Classical Latin			x	x	
Late and Medieval Latin	x	x			x
Later and Christian Latin	x	x		Vulgate only	x
Merged	x	x	x	x	x

Trankit



The screenshot shows the Trankit website interface. The left sidebar contains a search bar and a list of navigation links: INTRODUCTION, USAGE, Supported Languages, Trainable Languages, Pretrained Languages & Their Code Names, Sentence segmentation, Tokenization, Part-of-speech, Morphological tagging and Dependency parsing, Lemmatization, Named entity recognition, Building a customized pipeline, and Command-line interface. The main content area is titled 'Pretrained Languages & Their Code Names' and contains a paragraph explaining that Trankit provides 90 pretrained pipelines for 56 languages, each associated with a treebank. It then provides a table of these languages and their code names. Below the table, there is a note about the names of the default treebanks and how to initialize a pipeline.



Pretrained Languages & Their Code Names

Trankit provides 90 pretrained pipelines for 56 languages. Each pretrained pipeline is associated with a treebank that it is trained on. Below we show the 56 pretrained languages, their corresponding treebanks, and the code names to initialize pretrained pipelines. The pretrained pipelines can be directly downloaded by clicking on their code names in the table below.

Note that, the names of the default treebanks are put inside the brackets []. For example, English has 4 treebanks, which are `UD_English-EWT`, `UD_English-GUM`, `UD_English-LinES`, and `UD_English-ParTUT`. The treebank `UD_English-EWT` is put inside a bracket [], so it is the default treebank for English. Looking at the following table, we can select the appropriate code name and follow the instructions [here](#) to initialize a pipeline. For example, if we want to initialize a pipeline that is trained on the default treebank `UD_English-EWT`, we can use the code name `english`; to initialize a pipeline that is trained on a non-default treebank such as `UD_English-GUM`, we can use `english-gum`.

Language	Treebank	Code Name (for pipeline initialization)	Requires MWT expansion?	Treebank License
Afrikaans	[UD_Afrikaans-AfriBooms]	afrikaans		https://creativecommons.org/licenses/by/4.0/
Ancient Greek	[UD_Ancient_Greek-PROIEL]	ancient-greek		http://creativecommons.org/licenses/by/4.0/
	UD_Ancient_Greek-Perseus	ancient-greek-perseus		http://creativecommons.org/licenses/by/4.0/
Arabic	[UD_Arabic-PADT]	arabic	Yes	http://creativecommons.org/licenses/by/4.0/
Armenian	[UD_Armenian-ArmTDP]	armenian	Yes	http://creativecommons.org/licenses/by/4.0/
Basque	[UD_Basque-BDT]	basque		http://creativecommons.org/licenses/by/4.0/
Belarusian	[UD_Belarusian-HSE]	belarusian		http://creativecommons.org/licenses/by/4.0/
Bulgarian	[UD_Bulgarian-BTB]	bulgarian		http://creativecommons.org/licenses/by/4.0/
Catalan	[UD_Catalan-AnCorà]	catalan	Yes	https://www.gnu.org/licenses/gp
Chinese (simplified)	[UD_Simplified_Chinese-GSDSimp]	chinese		http://creativecommons.org/licenses/by/4.0/
Chinese (traditional)	[UD_Chinese-GSD]	traditional-chinese		http://creativecommons.org/licenses/by/4.0/

Reading on Trankit





 ACL Anthology [FAQ](#) [Corrections](#) [Submissions](#) [Github](#) 

Trankit: A Light-Weight Transformer-based Toolkit for Multilingual Natural Language Processing

Minh Van Nguyen, Viet Dac Lai, Amir Pouran Ben Veyseh, Thien Huu Nguyen

Abstract

We introduce Trankit, a light-weight Transformer-based Toolkit for multilingual Natural Language Processing (NLP). It provides a trainable pipeline for fundamental NLP tasks over 100 languages, and 90 pretrained pipelines for 56 languages. Built on a state-of-the-art pretrained language model, Trankit significantly outperforms prior multilingual NLP pipelines over sentence segmentation, part-of-speech tagging, morphological feature tagging, and dependency parsing while maintaining competitive performance for tokenization, multi-word token expansion, and lemmatization over 90 Universal Dependencies treebanks. Despite the use of a large pretrained transformer, our toolkit is still efficient in memory usage and speed. This is achieved by our novel plug-and-play mechanism with Adapters where a multilingual pretrained transformer is shared across pipelines for different languages. Our toolkit along with pretrained models and code are publicly available at: <https://github.com/nlp-uoregon/trankit>. A demo website for our toolkit is also available at: <http://nlp.uoregon.edu/trankit>. Finally, we create a demo video for Trankit at: <https://youtu.be/q0KGP3zGjGc>.

 PDF
 Cite
 Search
 Code

Anthology ID: 2021.eacl-demos.10
Volume: Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations
Month: April
Year: 2021
Address: Online

<https://aclanthology.org/2021.eacl-demos.10/>

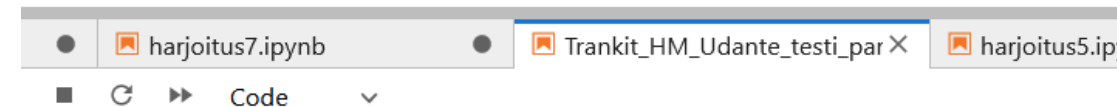
Using Trankit

- https://github.com/HannaKoo/ParsersTartu/blob/main/Trankit_use.md

Beginner's level

- Use Stanza
- Or continue with UD pipe 2
- Starting point a Github MD document
- Working together step by step

/node/r18c01.bullx/61715/lab/tree/kupariha/trankit/HM_udante/Trankit_HM_Udante_testi_par



```
<it
```

```
object_2008402/kupariha/venv/lib64/python3.9/site-packages/tqdm/auto.py:21: Tqdmw  
tps://ipywidgets.readthedocs.io/en/stable/user\_install.html  
notebook import tqdm as notebook_tqdm
```

```
ify_customized_pipeline(  
    lang='customized-mwt', # pipeline category  
    cache_dir='./save_dir', # directory used for saving models in previous steps  
    lang_name='xlm-roberta-base' # embedding version that we use for training our cust
```

```
    \line is ready to use!  
    \alized as follows:
```

```
from Pipeline  
p = Pipeline(lang='customized-mwt', cache_dir='./save_dir')
```

```
https://raw.githubusercontent.com/HannaKoo/Latin-variability/main/harmonization/
```

```
udante-ud-test.conllu > test.conllu
```

```
/conllu_to_text.pl --lang la < test.conllu > test.txt
```

```
'udante-ud-test.conllu' already there; not retrieving.
```

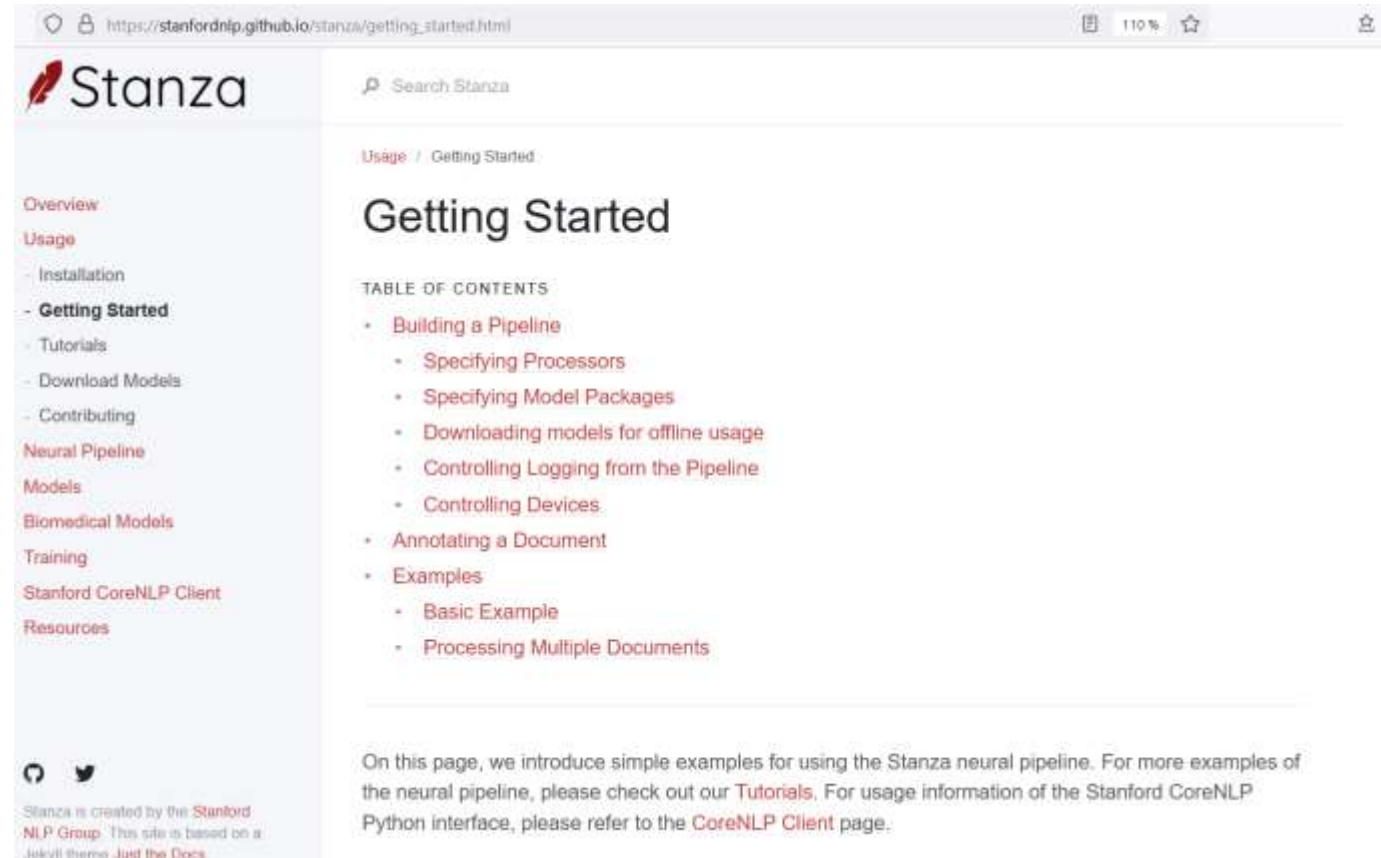
```
actions for processing the trankit output format
```

```
def line(token):  
    t = token.join([str(token["id"]), token["text"], token.get("lemma", "_"), token.get("up  
    re
```

```
def conllu(d):
```

```
    p = ("test_output_conllu" + "w" + encoding="UTF8") as f:
```

Stanza



The screenshot shows a web browser displaying the Stanza Getting Started page. The browser's address bar shows the URL `https://stanfordnlp.github.io/stanza/getting_started.html`. The page has a light blue sidebar on the left with a search bar and a navigation menu. The main content area is white and features a breadcrumb trail, a title, a table of contents, and a paragraph of introductory text.

Stanza

Search Stanza

Usage / Getting Started

Getting Started

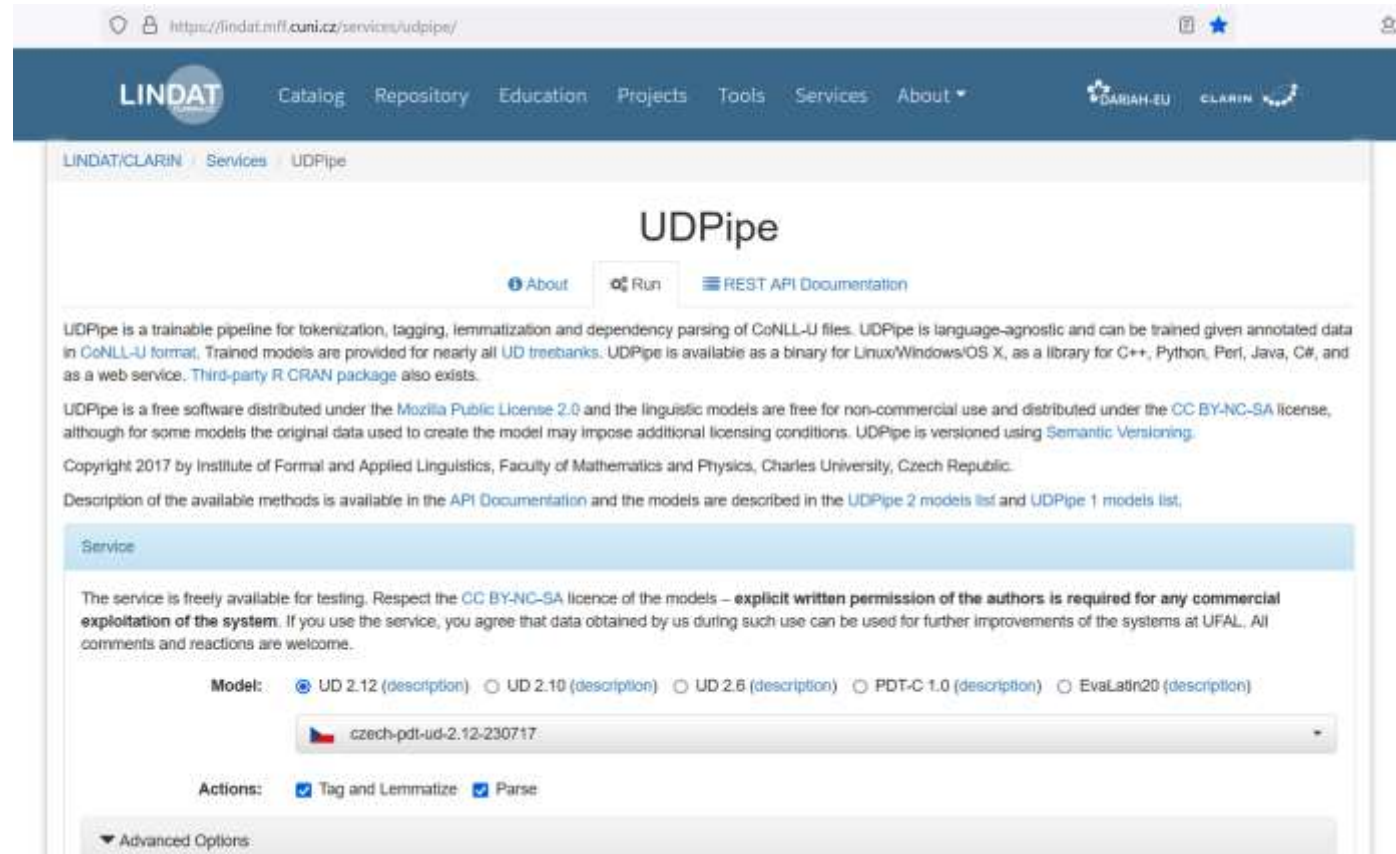
TABLE OF CONTENTS

- [Building a Pipeline](#)
 - [Specifying Processors](#)
 - [Specifying Model Packages](#)
 - [Downloading models for offline usage](#)
 - [Controlling Logging from the Pipeline](#)
 - [Controlling Devices](#)
- [Annotating a Document](#)
- [Examples](#)
 - [Basic Example](#)
 - [Processing Multiple Documents](#)

On this page, we introduce simple examples for using the Stanza neural pipeline. For more examples of the neural pipeline, please check out our [Tutorials](#). For usage information of the Stanford CoreNLP Python interface, please refer to the [CoreNLP Client](#) page.

Stanza is created by the [Stanford NLP Group](#). This site is based on a [Jekyll](#) theme [Just the Docs](#).

UD pipe 2



The screenshot shows the UDPipe web interface. The header includes the LINDAT logo and navigation links: Catalog, Repository, Education, Projects, Tools, Services, and About. The main content area is titled "UDPipe" and has tabs for "About", "Run", and "REST API Documentation". The "Run" tab is active. The text describes UDPipe as a trainable pipeline for tokenization, tagging, lemmatization, and dependency parsing of CoNLL-U files. It mentions that trained models are provided for nearly all UD treebanks and that UDPipe is available as a binary for Linux/Windows/OS X, as a library for C++, Python, Perl, Java, C#, and as a web service. A third-party R CRAN package also exists. The text also states that UDPipe is free software distributed under the Mozilla Public License 2.0 and the linguistic models are free for non-commercial use and distributed under the CC BY-NC-SA license, although for some models the original data used to create the model may impose additional licensing conditions. UDPipe is versioned using Semantic Versioning. Copyright 2017 by Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University, Czech Republic. Description of the available methods is available in the API Documentation and the models are described in the UDPipe 2 models list and UDPipe 1 models list.

Service

The service is freely available for testing. Respect the CC BY-NC-SA licence of the models – **explicit written permission of the authors is required for any commercial exploitation of the system.** If you use the service, you agree that data obtained by us during such use can be used for further improvements of the systems at UFAL. All comments and reactions are welcome.

Model: ☒ UD 2.12 (description) ☐ UD 2.10 (description) ☐ UD 2.6 (description) ☐ PDT-C 1.0 (description) ☐ EvaLatin20 (description)

Actions: ☒ Tag and Lemmatize ☒ Parse

▼ Advanced Options

What is Github?

- It is easy to set up a free account
- An easy way to share code
- For **advanced** level: Git is a distributed version control system that tracks changes in code
- Get notifications of things you are interested in

What are Jupyter notebooks?

- JupyterLab is the latest web-based interactive development environment for notebooks, code, and data
- **Jupyter Notebook** provides a fast interactive environment for prototyping and explaining code, exploring and visualizing data, and sharing ideas with others
- <https://jupyter-notebook.readthedocs.io/en/latest/>

Using Stanza step by step beginner's level

- Have a look at <https://stanfordnlp.github.io/stanza/>
- Open the .md file from https://github.com/HannaKoo/ParsersTartu/blob/main/Stanza_use.md
- We will first work with English and then:
- https://stanfordnlp.github.io/stanza/performance.html#system_performance-on-ud-treebanks