

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

19th of April 2024 University of Tartu, Estonia

Hanna-Mari Kupari hmknie@utu.fi

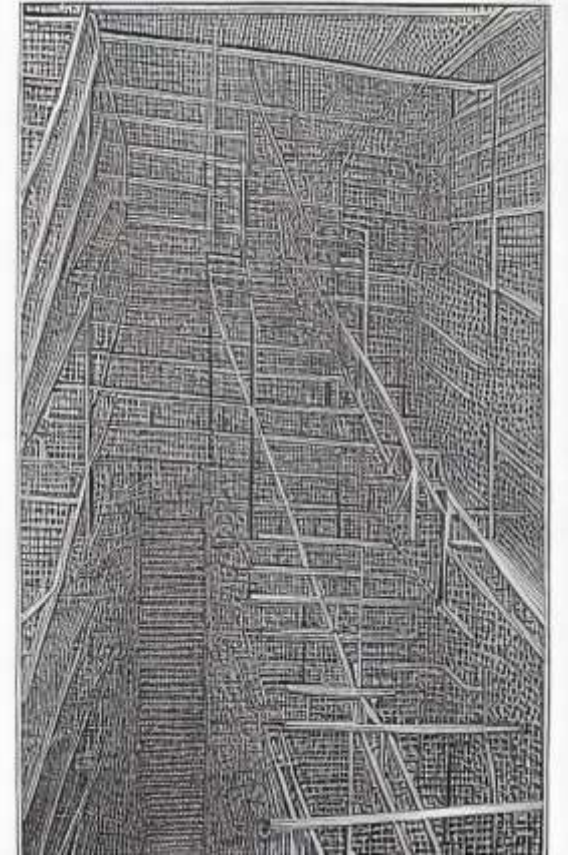
➤ Workshop Day 5

➤ Recap of ConlluEditor

Overview of next possible directions

A short presentation of my own work

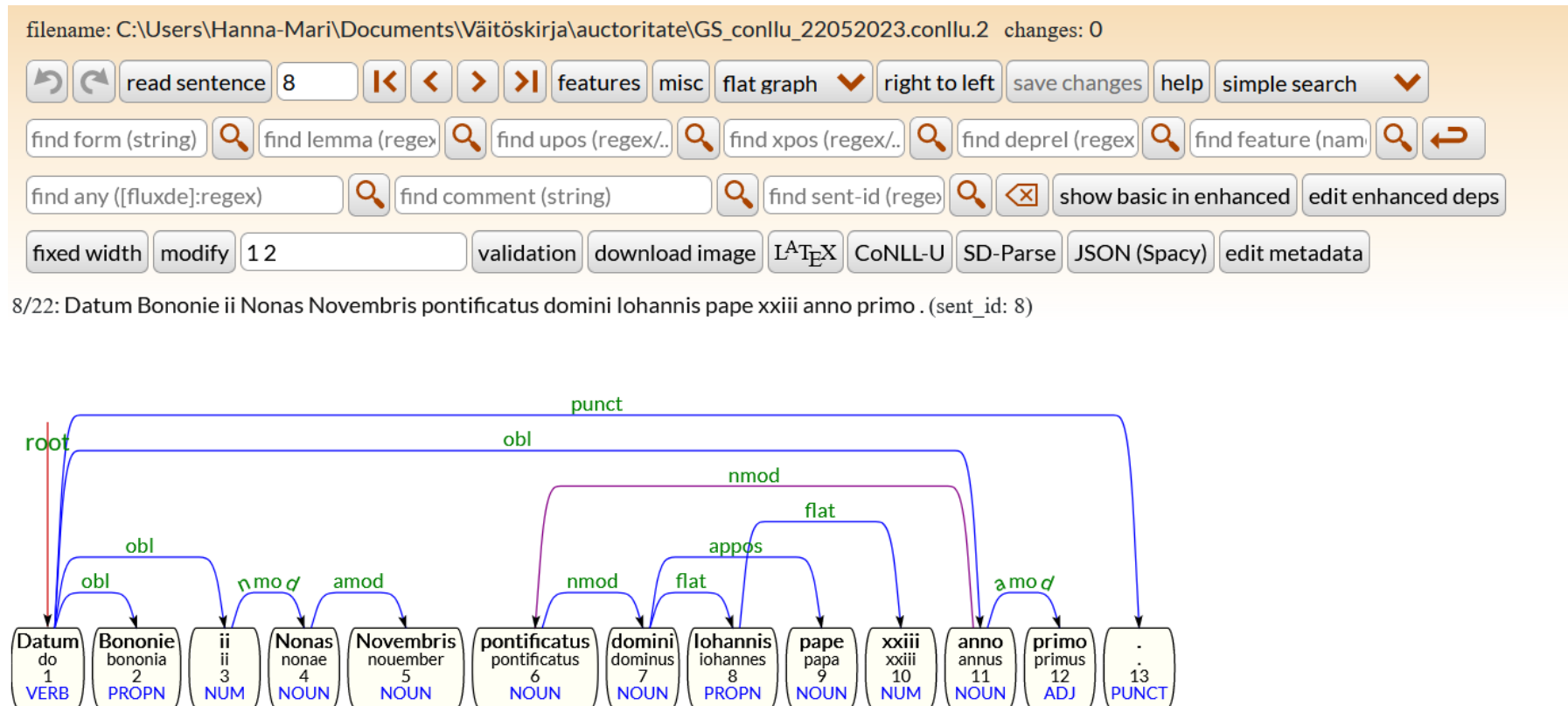
Personal projects help



Deep AI

ConlluEditor for creating a Gold Standard treebank:

- Install Conllu-editor



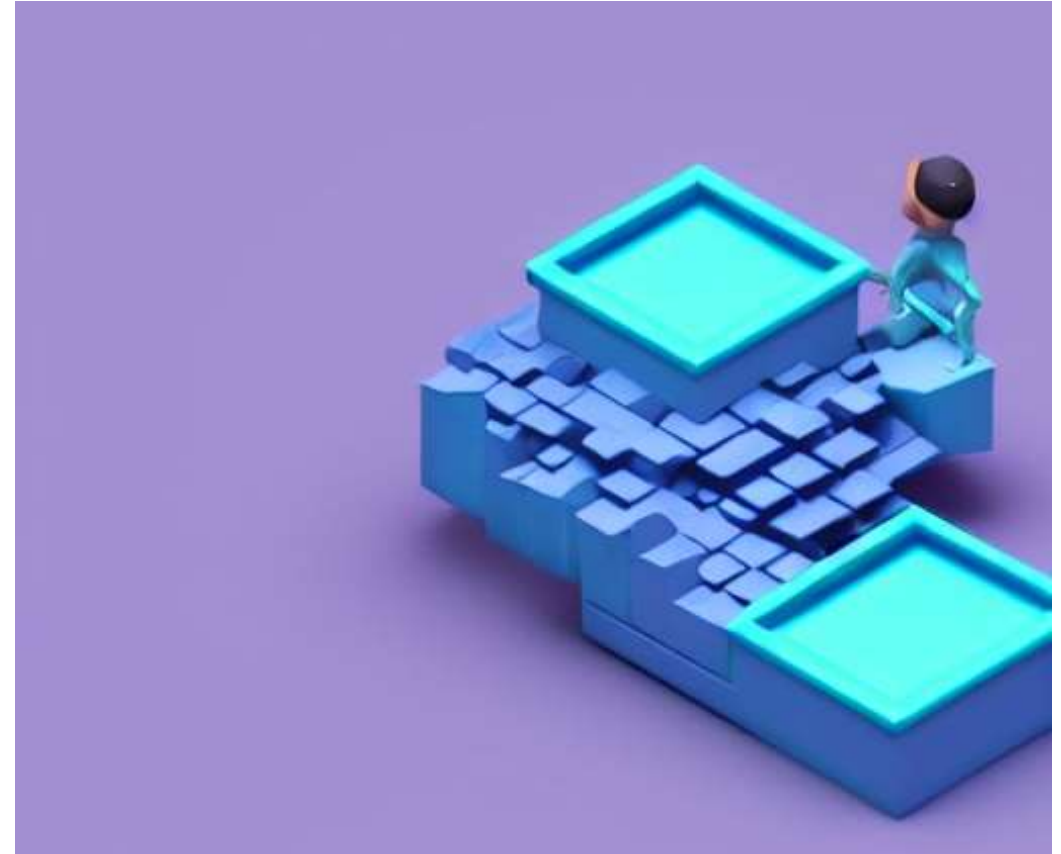
Detailed instructions to follow

- https://github.com/HannaKoo/ParsersTartu/blob/main/ConlluEditor_use.md
- clicking on a word and then clicking on the head-word creates a dependency relation. An edit window opens to enter the relation a name
- Existing relations can be renamed by clicking on their name
- Clicking twice on a word deletes its eventual dependency relation and makes it root
- <https://github.com/Orange-OpenSource/conllueditor>

Hungry for more?

- Developing your own model
- Improving your results with a voting system
- Reporting your results
- Programming skills for working better with tabular data
- Looking into machine learning

DeepAI



Developing your own model

- Needs 3 things
 - A suitable dataset
 - A suitable GPU (supercomputer)
 - Some programming skills
- Trankit detailed instructions
- <https://trankit.readthedocs.io/en/latest/training.html>
- The basis is the XML-RoBERTa Large
- Finetuned with your own treebank model

Trankit training

The screenshot shows a web browser displaying the Trankit documentation page for 'Building a customized pipeline'. The page has a blue header with the Trankit logo and a search bar. A left sidebar contains a table of contents with links to various sections. The main content area is titled 'Building a customized pipeline' and includes a 'NOTE' about supported languages, a paragraph explaining how to build pipelines using the `TrankitPipeline` class, and a bulleted list of four pipeline categories: `customized-mwt-ner`, `customized-mwt`, `customized-ner`, and `customized`. Below this, a paragraph explains how to use these category names as special identities in the `Pipeline` class. The page also features a 'Training' section with a sub-section 'Training a joint token and sentence splitter' and a code snippet `import trankit`. A right sidebar contains a 'Edit on GitHub' link. A bottom right overlay shows a version history table and a promotional message for automatic documentation previews.

trankit
latest

Search docs

INTRODUCTION

News: Trankit v1.0.0 is out

Installation

Quick examples

How Trankit works

Model performance

USAGE

Supported Languages

Sentence segmentation

Tokenization

Part-of-speech, Morphological tagging and Dependency parsing

Lemmatization

Named entity recognition

Building a customized pipeline

Training

Loading

Command-line interface

Building a customized pipeline

Edit on GitHub

Building a customized pipeline

NOTE: Please check out the list of [supported languages](#) to check if the language of your interest is already supported by Trankit.

Building customized pipelines are easy with Trankit. The training is done via the class `TrankitPipeline` and the loading is done via the class `Pipeline` as usual. To achieve this, customized pipelines are currently organized into 4 categories:

- `customized-mwt-ner` : a pipeline of this category consists of 5 models: (i) joint token and sentence splitter, (ii) multi-word token expander, (iii) joint model for part-of-speech tagging, morphological feature tagging, and dependency parsing, (iv) lemmatizer, and (v) named entity recognizer.
- `customized-mwt` : a pipeline of this category doesn't have the (v) named entity recognizer.
- `customized-ner` : a pipeline of this category doesn't have the (ii) multi-word token expander.
- `customized` : a pipeline of this category doesn't have the (ii) multi-word token expander and the (v) named entity recognizer.

The category names are used as the special identities in the `Pipeline` class. Thus, we need to choose one of the categories to start training pipelines. Below we show the example for training and loading a `customized-mwt-ner` pipeline. Training procedure for customized pipelines of be obtained by obmitting the steps related to the models that those pipelines don't have. For data format, `TrankitPipeline` accepts training data in the Universal Dependencies tasks, and training data in [BIOES format](#) for the NER task.

Training

Training a joint token and sentence splitter

```
import trankit
```

Version 3.1

✓ PR #379

✗ PR #378

Automatic doc previews for every PR when you host your documentation on Read the Docs. [Sign up today.](#)

Ad by Phosidek

Requirements for finetuning

- All data must be in CoNLL-U format
- A requirement of training and development data
 - `.train`
 - `.dev`
- Typically, you also have a `.test` dataset to compare the predicted results to a Gold Standard
- Training datasets are in essence Gold Standard annotated
 - This can be split into 3 parts, e.g. 80 % for train, 10 % dev and 10 % test

What to consider in training?

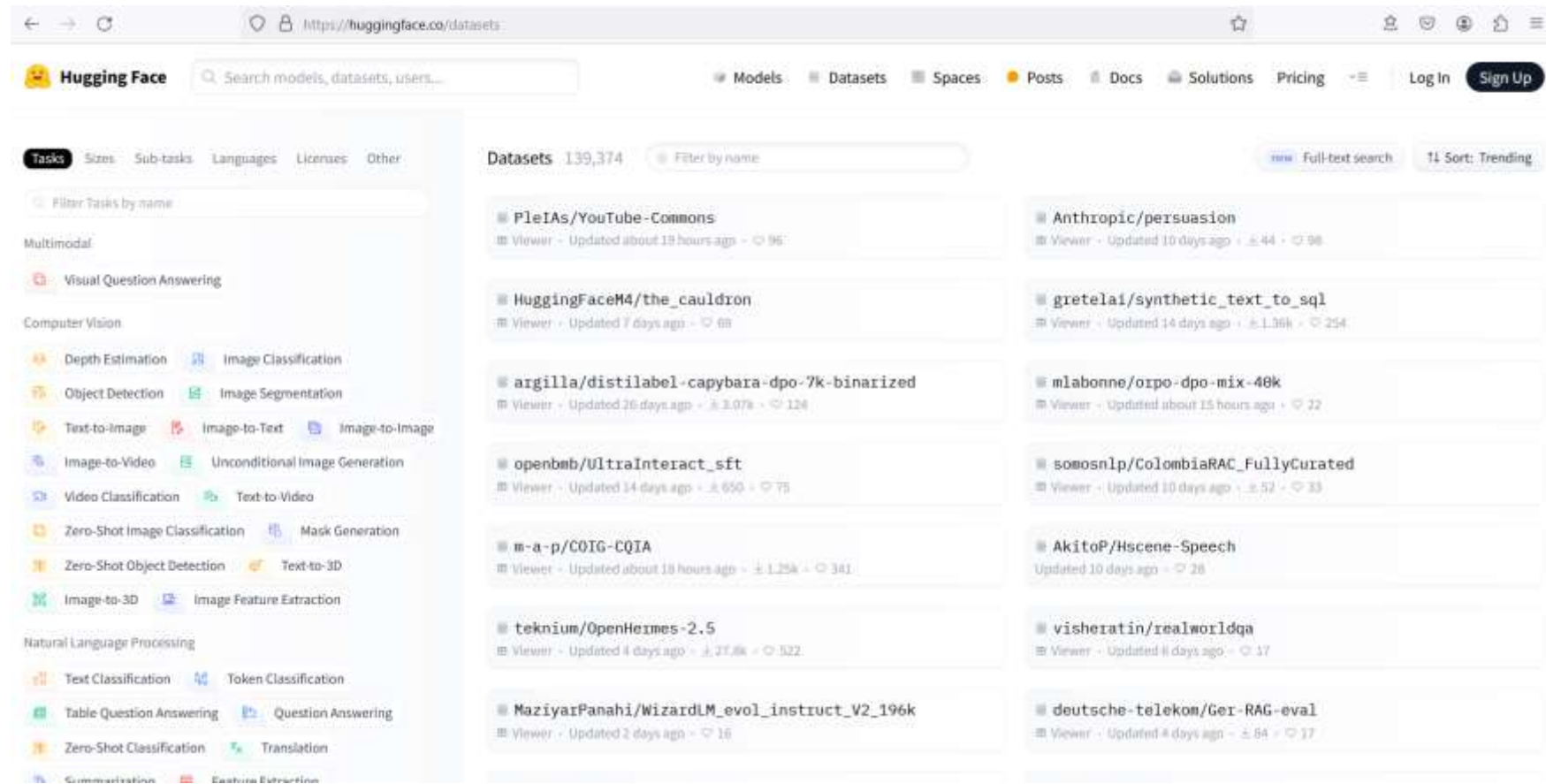
- Find the biggest amount of data that is suitable for your own needs
- Think about the characteristics of your own corpus regarding the training data (genre, time period)
- Expect some noise
- If you don't take into consideration the computational resources, the more the better (impact on cost, environmental burden)



Nvidia's greenhouse gas emissions in 2023 amounted to **73,017 metric tons of CO2 equivalent**

Huggingface resources

- <https://huggingface.co/datasets>

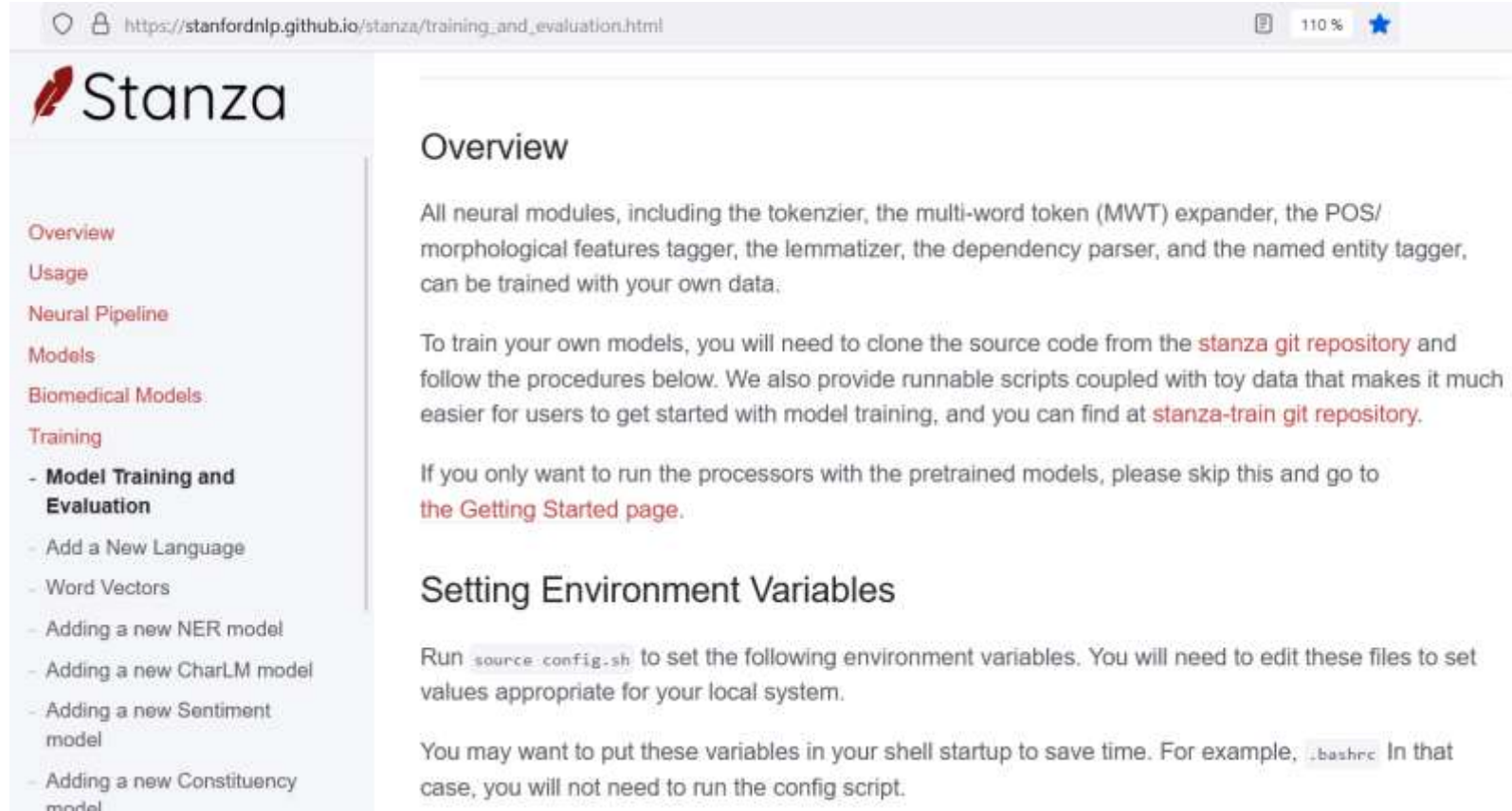


What to consider in training?

- Experiment with several options
- Continue with the most accurate one and create your own treebanks by correcting manually some output
- Use this Gold Standard to develop a new model (save some of course for .dev and .test)



Stanza can also be trained



The screenshot shows a web browser window with the URL `https://stanfordnlp.github.io/stanza/training_and_evaluation.html`. The page features the Stanza logo (a red quill) and a sidebar with navigation links: Overview, Usage, Neural Pipeline, Models, Biomedical Models, and Training. The Training section is expanded, showing a list of links: Model Training and Evaluation, Add a New Language, Word Vectors, Adding a new NER model, Adding a new CharLM model, Adding a new Sentiment model, and Adding a new Constituency model. The main content area is titled "Overview" and contains text about training neural modules with user data, cloning the Stanza git repository, and following procedures for training. It also mentions a `stanza-train` git repository for easier model training. A section titled "Setting Environment Variables" provides instructions on running `source config.sh` to set environment variables and putting them in the shell startup file `.bashrc`.

Overview

All neural modules, including the tokenizer, the multi-word token (MWT) expander, the POS/morphological features tagger, the lemmatizer, the dependency parser, and the named entity tagger, can be trained with your own data.

To train your own models, you will need to clone the source code from the [stanza git repository](#) and follow the procedures below. We also provide runnable scripts coupled with toy data that makes it much easier for users to get started with model training, and you can find at [stanza-train git repository](#).

If you only want to run the processors with the pretrained models, please skip this and go to [the Getting Started page](#).

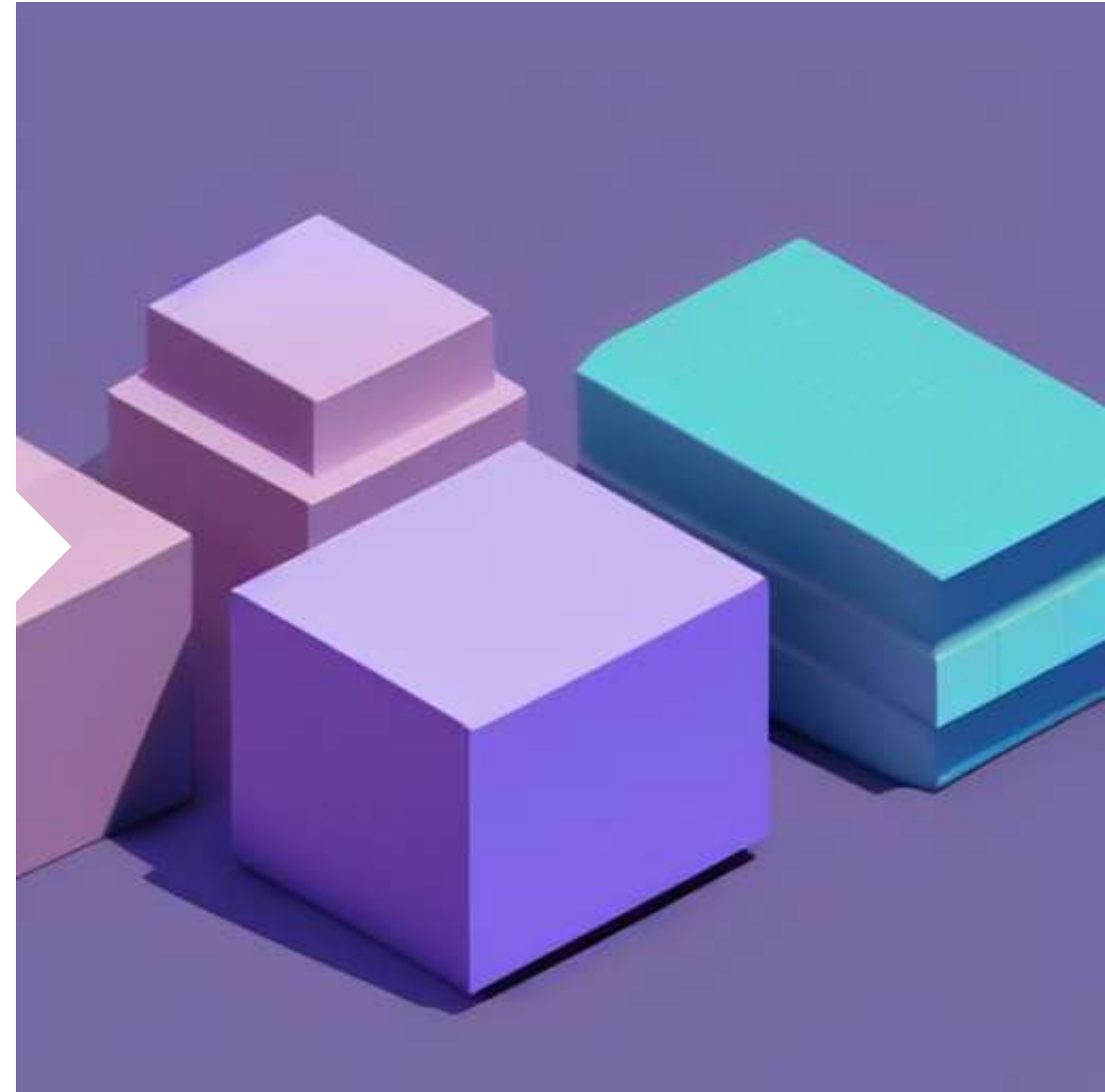
Setting Environment Variables

Run `source config.sh` to set the following environment variables. You will need to edit these files to set values appropriate for your local system.

You may want to put these variables in your shell startup to save time. For example, `.bashrc`. In that case, you will not need to run the config script.

Take home message of training

- Read the documentation
- Don't be frustrated when the documentation is poor
- Make high quality notes
- Consider sharing your models and training data



Improve your results with voting

- Use 3 models or more
- Take the most common result
- <https://aclanthology.org/W05-1518.pdf>
- Improving Parsing Accuracy by Combining Diverse Dependency Parsers, Zeman et al, 2005

Report your results

- Publish all numbers in a GitHub repo
- Take your most important numbers up in the paper
- Think about the reader of the paper

	ittb.udp		llct.udp		perseus.udp		proiel.udp		udante.udp	
	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS
ITTB	84.51%	86.23%	44.25%	52.16%	29.54%	40.56%	30.54%	45.43%	59.93%	65.77%
LLCT	44.22%	50.16%	93.02%	93.85%	28.92%	37.44%	40.37%	52.10%	45.57%	53.42%
Perseus	33.28%	44.21%	39.85%	48.71%	61.80%	67.18%	38.93%	55.16%	35.64%	45.79%
PROIEL	39.10%	50.86%	43.16%	53.08%	41.52%	52.36%	73.51%	77.45%	39.43%	48.62%
UDante	50.78%	58.51%	36.95%	45.78%	22.44%	32.41%	26.72%	40.41%	50.81%	57.32%

Table 3: UDPipe scores before treebank alignment. Columns correspond to trained models, rows to test data.

	ittb.udp		llct.udp		perseus.udp		proiel.udp		udante.udp	
	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS
ITTB	83.83%	85.51%	43.80%	51.45%	43.17%	53.12%	40.46%	51.33%	61.68%	67.39%
LLCT	43.12%	48.55%	93.11%	93.88%	47.31%	54.13%	46.69%	55.23%	41.56%	49.05%
Perseus	42.73%	53.54%	48.69%	55.24%	63.80%	68.38%	49.98%	59.25%	43.59%	54.23%
PROIEL	46.77%	55.39%	50.37%	57.48%	53.11%	59.88%	75.78%	78.87%	46.13%	55.15%
UDante	53.06%	59.95%	38.51%	46.69%	35.59%	45.64%	30.72%	44.11%	54.50%	61.02%

Table 4: UDPipe scores after treebank alignment. Columns correspond to trained models, rows to test data.

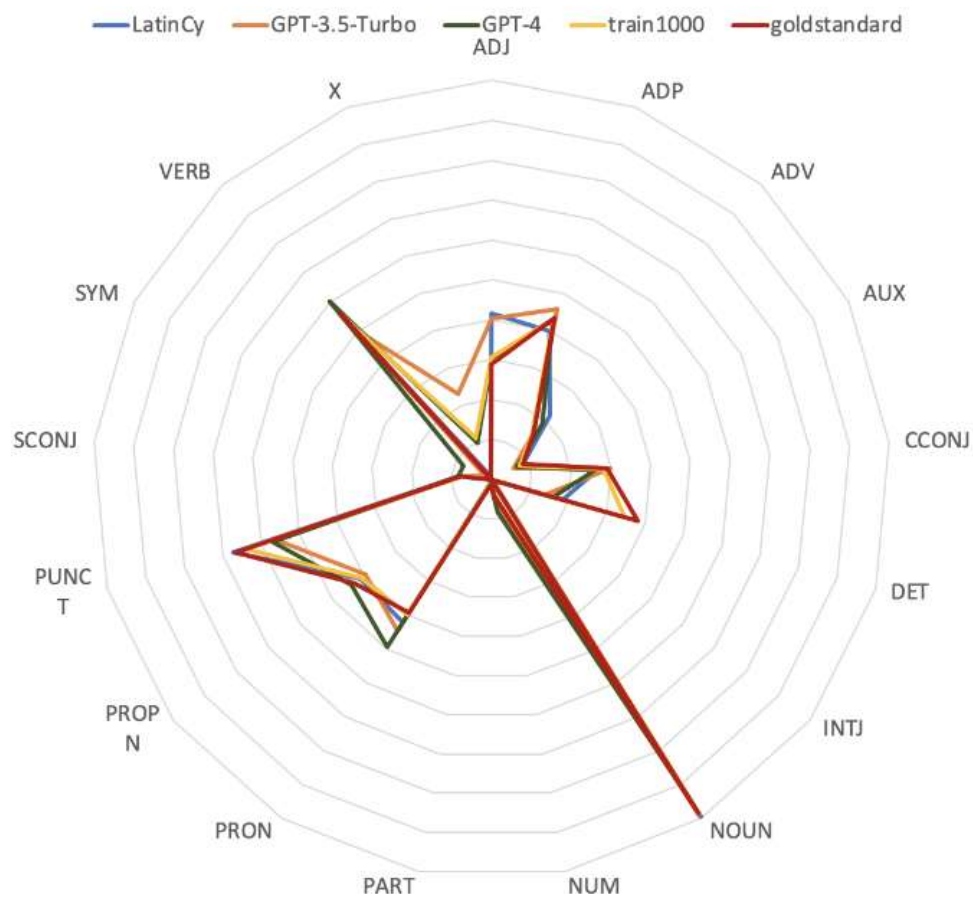


Figure 6: POS tag distribution in the LLCT test set.

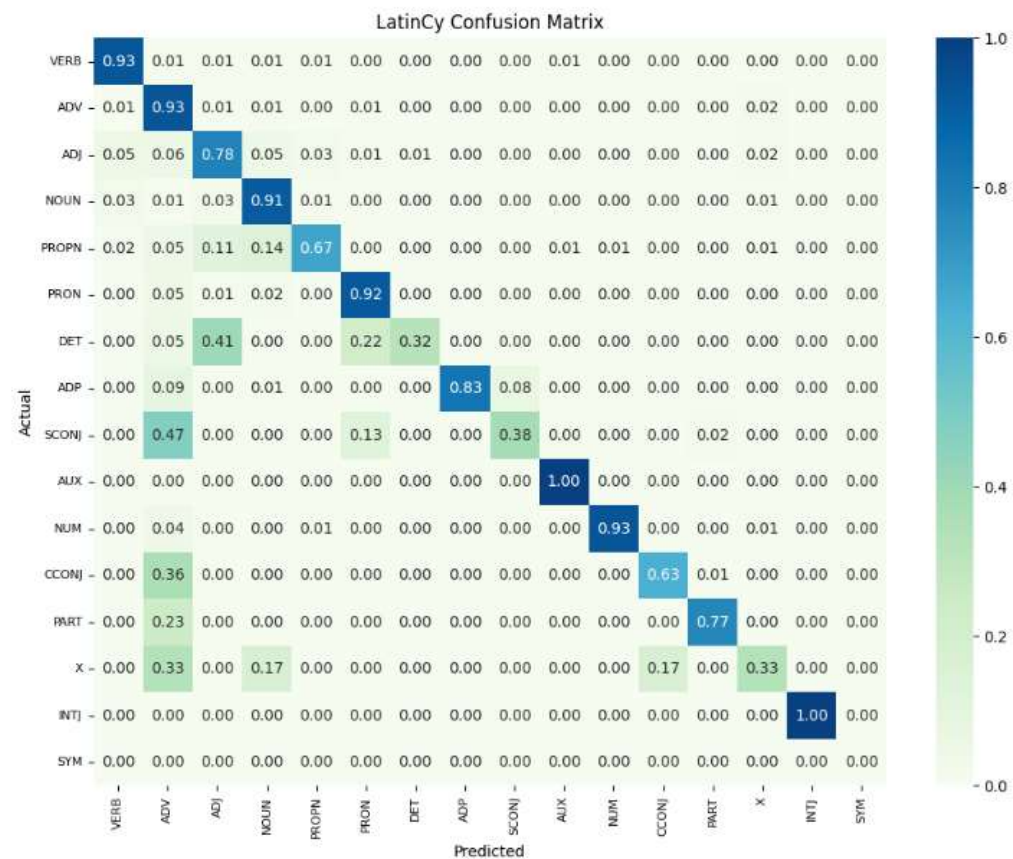
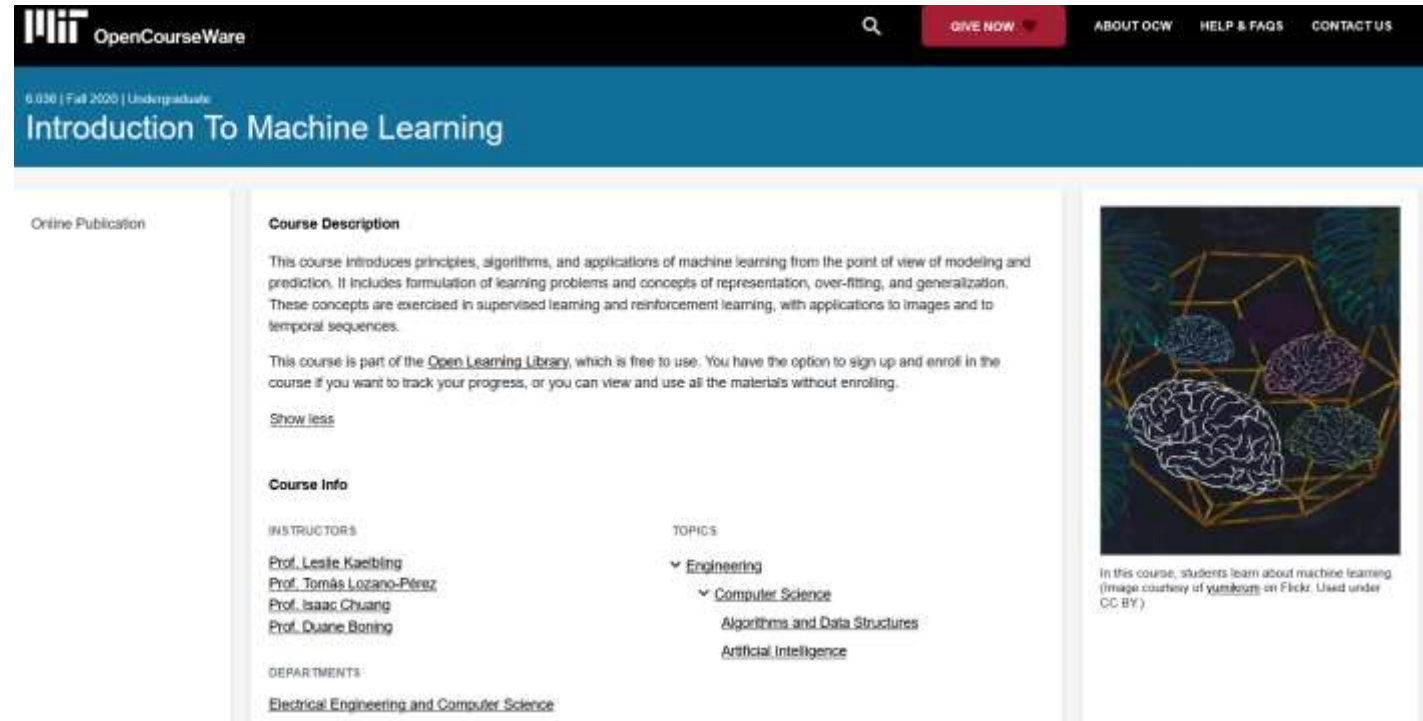


Figure 7: Confusion matrix for LatinCy on the Bullinger sample.

Most important programming skills

- Basic Python
- Skills to work with tabulator separated values
- Some insight into ML
- Folgert Karsdorp:
 - <http://www.karsdorp.io/python-course/>



The image shows a screenshot of the MIT OpenCourseWare website for the course "Introduction To Machine Learning". The page has a dark blue header with the MIT logo and "OpenCourseWare" text. Below the header, there's a blue banner with the course title "Introduction To Machine Learning". The main content area is white and contains several sections: "Online Publication", "Course Description", "Course Info", and "TOPICS". The "Course Description" section provides a brief overview of the course content. The "Course Info" section lists the instructors: Prof. Leslie Kaelbling, Prof. Tomás Lozano-Pérez, Prof. Isaac Chuang, and Prof. Duane Boning. The "TOPICS" section lists the topics: Engineering, Computer Science, Algorithms and Data Structures, and Artificial Intelligence. On the right side of the page, there is a large image of a brain with a network of connections, and a caption below it stating: "In this course, students learn about machine learning. (Image courtesy of yunbozhu on Flickr. Used under CC BY.)"

MIT OpenCourseWare

6.036 | Fall 2020 | Undergraduate

Introduction To Machine Learning

Online Publication

Course Description

This course introduces principles, algorithms, and applications of machine learning from the point of view of modeling and prediction. It includes formulation of learning problems and concepts of representation, over-fitting, and generalization. These concepts are exercised in supervised learning and reinforcement learning, with applications to images and to temporal sequences.

This course is part of the [Open Learning Library](#), which is free to use. You have the option to sign up and enroll in the course if you want to track your progress, or you can view and use all the materials without enrolling.

[Show less](#)

Course Info

INSTRUCTORS

Prof. Leslie Kaelbling
Prof. Tomás Lozano-Pérez
Prof. Isaac Chuang
Prof. Duane Boning

TOPICS


▼ Engineering

▼ Computer Science

Algorithms and Data Structures
Artificial Intelligence

DEPARTMENTS

Electrical Engineering and Computer Science



In this course, students learn about machine learning. (Image courtesy of yunbozhu on Flickr. Used under CC BY.)

My own use of parsing tools

- To analyze a large amount of texts
- We have countless studies on differences between Classical and Medieval Latin
- Usually these discuss one feature of language or are based in the heuristic understanding of the researcher
- What can we say quantitatively that makes medieval Latin medieval?

Penitentiary documents provide us an insight to the communicative means of persuasive language use



The situation – e.g. violent assault

description



+
Linguistic
features

Thanks to University of Tartu



EMIL AALTOSEN SÄÄTIÖ

- LinkedIn
- hmknie@utu.fi



Kieli- ja käännöstieteiden laitos 2024