NLP data visualization project

...from experimentation to the beginning of my self-discovery & NLP-discovery

13.08.2025

🧐 background story

- **General Linguistics** (Obecná Lingvistika) & **Digital Humanities**
- **natural language processing** (zpracování přirozeného jazyka) (NLP)
- My **Github** Profile: https://github.com/kivanc57

👀 about me

- linguist & programmer
- MA student @ UPOL
- former 42 Prague student
- combat sports enthusiast
- polyglot
- i use Arch, btw

🗃️ about project(-s)

- part 1 **(spacy)** ➜ Github repo & doc
- part 2 **(nltk)** ➜ Github repo & book
- **gradual evolution**
  basic linguistic analysis ➜ various file handlings ➜ data visualization ➜ `nltk` (comparison with other library) & academic report ➜ Marp presentation ➜ `Dockerfile` & `docker-compose` ➜ Github Actions `pipeline`
- **final project(-s)** zápočet(-y)
- **datasets** ➜ large, various, traditional, extracted from real-life situations and *Kaggle* (enron, reuters, COSMIC-2 Data, Faust...).

spaCy project

project structure

```
📁 project-root
├── 📁 config
│  ├── 📄 __init__.py
│  ├── 📄 common_config.py
│  └── 📄 constants.py
│
├── 📁 scripts
│  ├── 📄 __init__.py
│  ├── 📄 entity_treemap.py
│  ├── 📄 freq_barplot.py
│  ├── 📄 len_histogram.py
│  ├── 📄 len_violin.py
│  ├── 📄 pos_cloud.py
│  └── 📄 sentiment_piechart.py
│
├── 📁 (logs)
│     ...
│
├── 📄 .gitignore
├── 📄 .gitattributes
└── 📄 main.py
```

script structure

Script	Description	Input File	Ouput File	Graph
freq_barplot.py	Generates frequency bar plots from XML data.	xml	txt	Bar Plot
len_histogram.py	Creates histograms of sentence lengths from text files.	txt	-	Histogram
sentiment_piechart.py	Generates pie charts based on sentiment analysis from text data.	sgm	-	Pie Chart
entity_treemap.py	Categorizes entities and build treemaps from XML data using them.	xml	json	Tree Map
len_violin_plot.py	Visualizes the length of email addresses using violin plots.	txt	xlsx	Violin Plot
pos_cloud.py	Generates word clouds based on parts of speech from text data.	sgm	csv	Word Cloud

config/common_config.py

```
import logging
from os.path import dirname, abspath, join, basename
from os import makedirs

#Create separate log for each file with its name
def get_log_path(script_name, log_folder='logs'):
    project_path = dirname(dirname(abspath(__file__)))
    log_folder_path = join(project_path, log_folder)
    makedirs(log_folder_path, exist_ok=True) #Create the folder if does not exist
    log_file_name = f"{script_name}.log"
    full_path = join(log_folder_path, log_file_name)
    return full_path

#Configures logging in each script
def configure_logging(script_name):
        log_file = get_log_path(script_name)
        logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s %(name)s %(levelname)s %(message)s',
        handlers=[
            logging.FileHandler(log_file, mode='a'),
            logging.StreamHandler()
        ],
    )

#Adjust this os function in each function to find inpput and output paths
def get_join_path(folder_name, file_name, is_sample=False):
    try:
        project_path = dirname(dirname(abspath(__file__)))
        if is_sample:
            folder_path = join('data', folder_name, 'sample')
        else:
            folder_path = join('data', folder_name)
        full_path = join(project_path, folder_path, file_name)
        return full_path

    except Exception as e:
        print(f"Error: {e} joining {folder_name} and {file_name}")
```

a. entity_treemap.py

This script parses an .xml file and extracts the required keywords. It then converts the output into a dictionary format, categorizes the entities, and generates a treemap to show the distribution of thematic words. Finally, the results are saved in JSON format along with the treemap graph.

Linguistic functions: Parsing entities of thematic words, which can be referred to as semantic categorization of words.

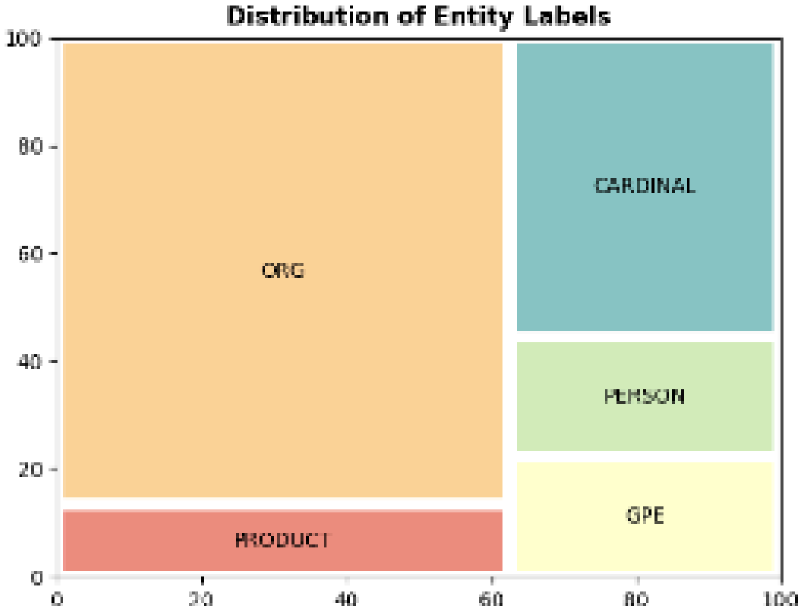

Figure 1.1: Treemap

b- freq_barplot.py

This script retrieves data from an .xml file, converts it into a frequency list of lemmas, and generates a bar plot. Finally, both the bar plot and the numerical results are saved as plain text.

Linguistic function: Creation of a frequency list and lemmatization.

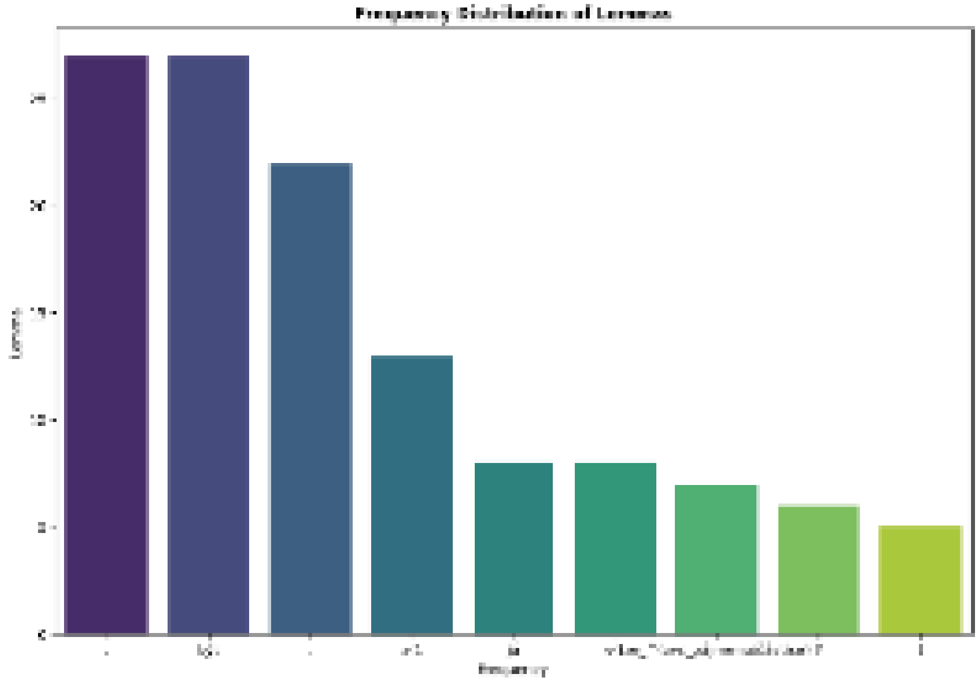

Figure 1.2: Bar Plot

c- len_histogram.py

In this script, I read the data from a .txt file, then calculate the sentence lengths. The results are visualized as a histogram and saved to the local device.

Linguistic function: Calculation of sentence length based on the number of tokens.

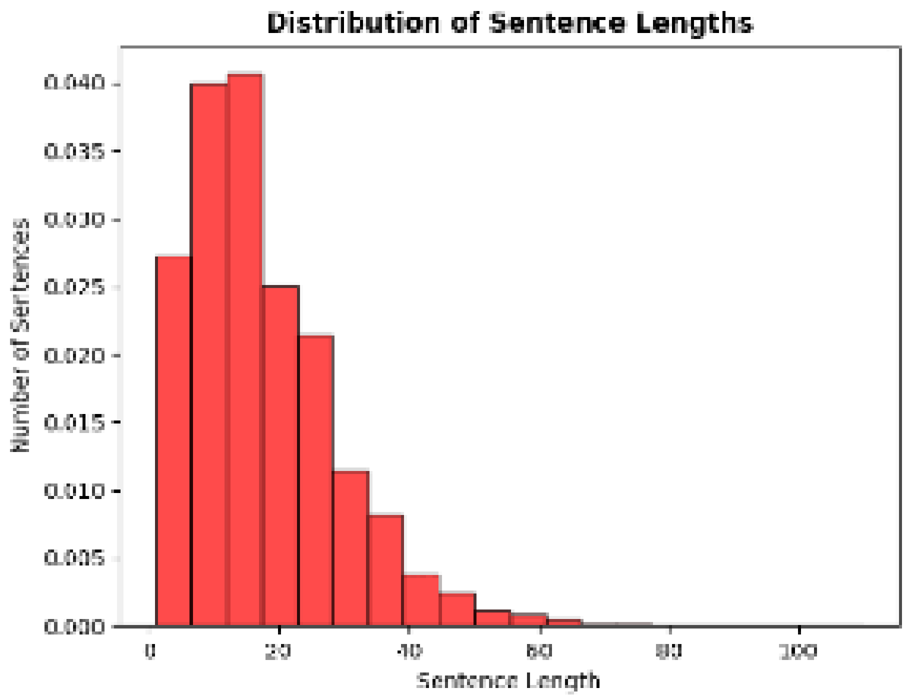

Figure 1.3: Histogram

d.- len_violin_plot.py

The script above retrieves the emails as plain text. It then extracts the lexemes that match a given regular expression (e.g., "\w+@\w+.\w+" for emails in this case). Next, it calculates the lengths of these lexemes and generates a violin plot. The results are saved in both .png and Excel file formats.

Linguistic functions: Matching regular expressions and calculating the lengths of tokens.

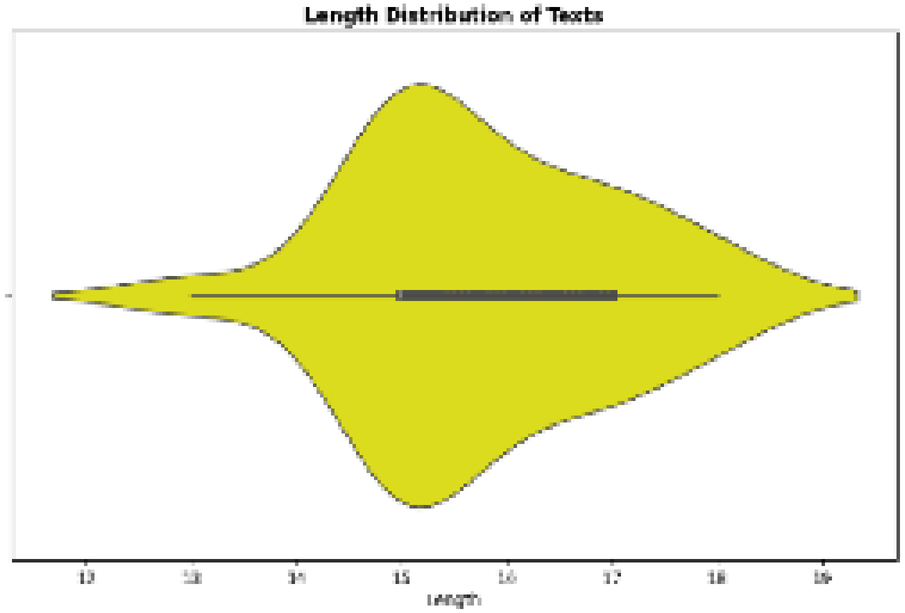

Figure 1.4: Violin Plot

e- pos_cloud.py

This program reads multiple .sgm files, parses them, and lemmatizes each token. Each lemma is categorized and filtered based on its part-of-speech tagging. The resulting output highlights entities belonging to the classes of verbs, subjects, and objects. Finally, the results are saved as a .csv file and visualized as a word cloud.

Linguistic functions: Lemmatization, part-of-speech tagging, and extracting specific lexical classes.

Figure 1.5: Word Cloud

d- sentiment_piechart.py

This program takes a .sgm file as input, calculates the sentiment based on the semantic meaning of each token, and categorizes them accordingly. The results are visualized as a pie chart and saved to a local folder.

Linguistic functions: Sentiment analysis and semantic analysis.

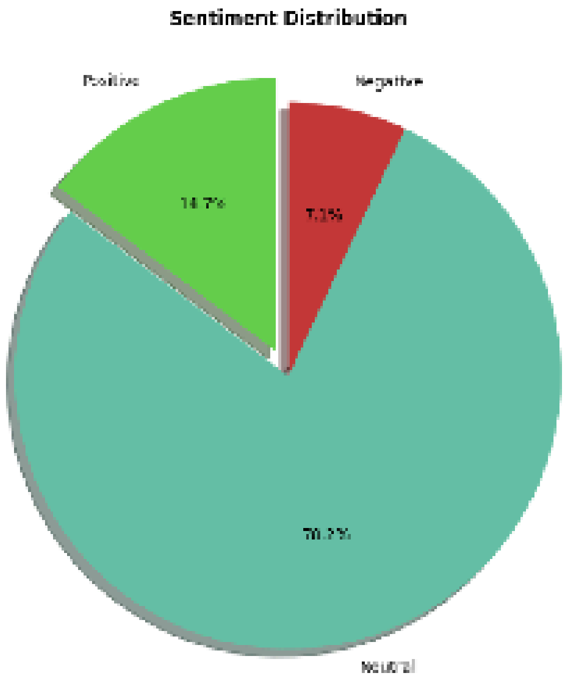

Figure 1.6: Pie Chart

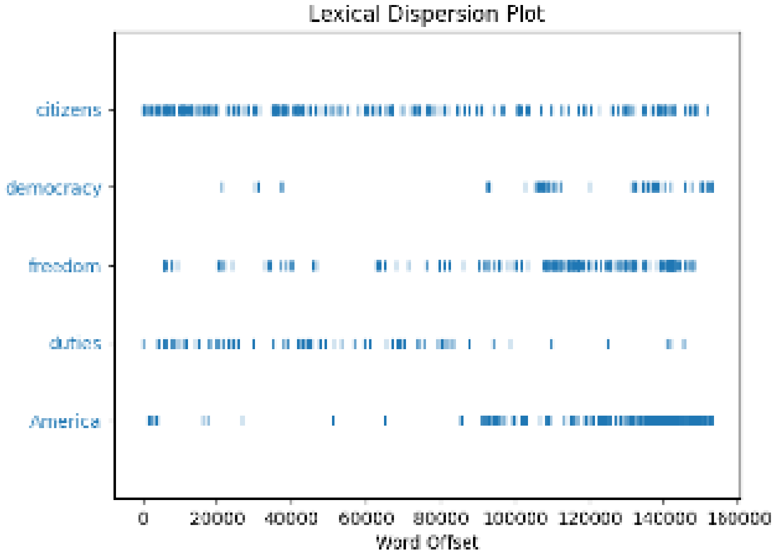

Figure 2.1.1: Dispersion Plot

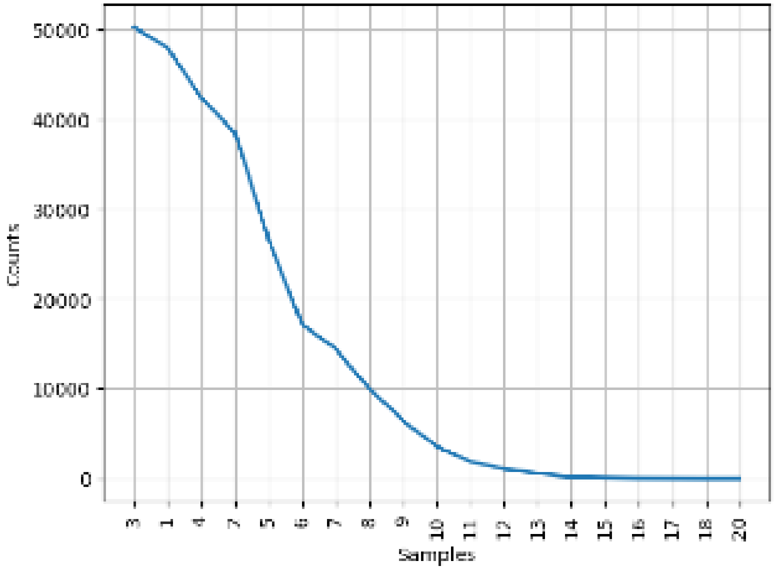

Figure 2.1.2: Frequency Distribution Plot

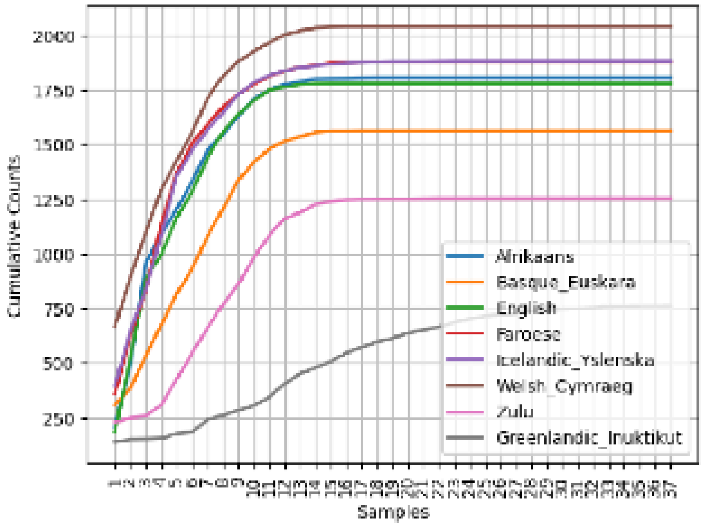

Figure 2.2.1: Word Length Plot of Different Languages

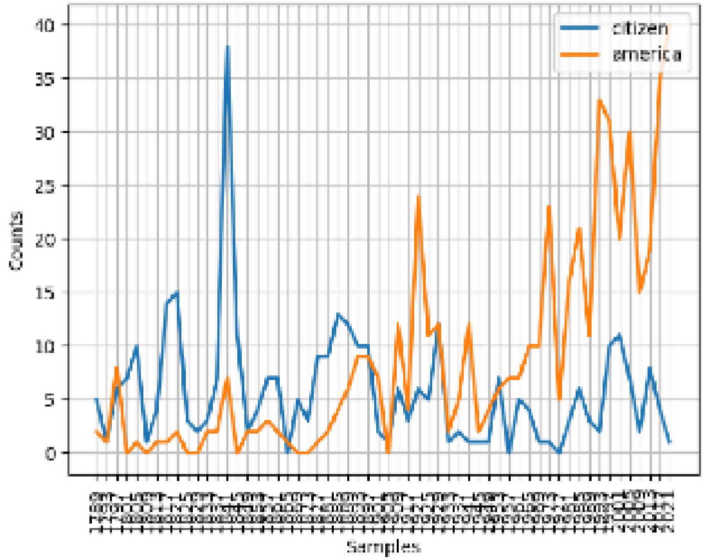

Figure 2.2.2: Word Count Plot of Different Words

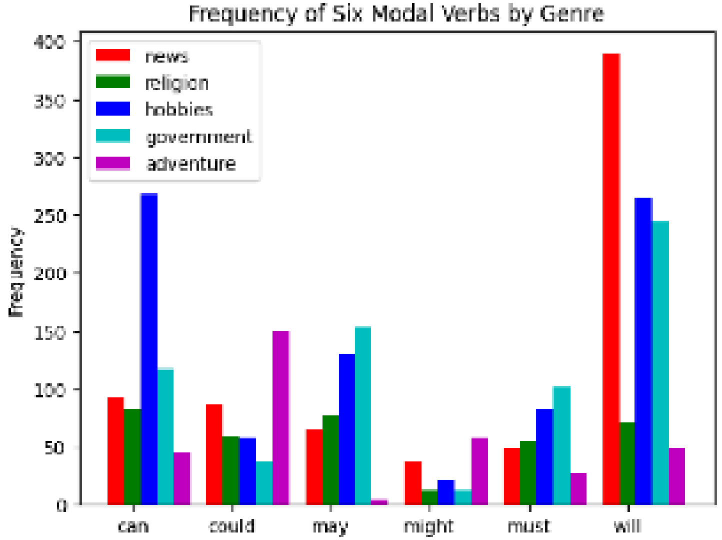

Figure 2.4.1: Frequency Distribution Graph of Model Verbs by Ger

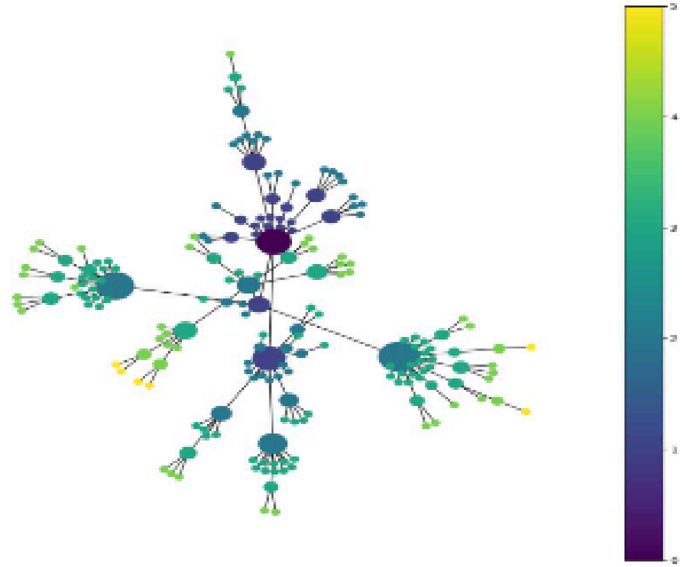

Figure 2.4.2: WordNet Graph

spaCy & nltk

🚀 tech stack

part 1 - spaCy

vscode, **os**, **re**, **dotenv**, **bs4**, **xml**, **pandas**, **spacy**
csv, json, logging, seaborn, squarify, matplotlib, wordcloud, textblob

part 2 - nltk

jupyter notebook, nltk, re, xml, bs4, urllib, pickle, collections, matplotlib, numpy

🔥 nltk vs nlp

nlp

- endustry standard
- newer, robust and *deep learning* accompanied
- fast & efficient
- easy-to-learn
- general purposes

nltk

- academy standard
- old-school, however has *machine learning*
- slower
- steeper learning curve
- tailored for *linguistic analysis*

what did i learn?

- reading & searching by **documentation**
- **the longest journey:** the entire process took me months
- **self-reliance** & **self-confidence**
- **scalability** concerns for big data
- **hands-on experience** with multiple data science, utility, file libraries, some **CI/CD** tools & a few **IDEs**
- **observation** of main NLP libraries' comparison
- **12.12.2025:** neovim python setup + Marp documentation
- **13.12.2025:** Dockerfile, docker-compose.yml and Github Actions
- **credits** 🤗

technical volume vs. complexity

- **dilemma**
- one of a kind and **first stand-alone project**
- tiny **technical details**
- **real value** because of previously mentioned points
- fast-paced learning in **last two days(!?)**

💼🤝 let's connect!

📩 kivancgordu@hotmail.com