NLP Data Analysis Illustration and Repository

Author: Kıvanç Gördü

Course: KOL/TEK

Introduction

In this report, I demonstrate two projects that I have built predominantly using two mainstream Python NLP (Natural Language Processing) libraries: *spaCy* and *NLTK*, alongside other supporting libraries. The choice between spaCy and NLTK depends entirely on the business case and specific scenario. While NLTK is well-known for its versatility and academic rigor, spaCy is better suited for real-world applications, shining with its efficiency and performance. Both repositories are listed below and can be further explored for more detailed information:

1. nlp_data_visualization (spacy): https://github.com/kivanc57/nlp_data_visualization
2. nltk_complete (nltk): https://github.com/kivanc57/nltk_complete

Although, functions are fixed for rigid cases, they are easily adjustable for an extensive frame of inner mechanism, input and output format with a little knowledge of Python programming. I use multiple document formats for input and output on ‘nlp_data_visualization’ repository that indeed proves this statement. This repository is a result-based while ‘nltk_complete’ focuses more on the process itself. Threfore, their exploration in this paper also will be slightly different.

Ultimately, these repositories generate graphical representations to illustrate the results in a user-friendly manner. Unlike standard programming documentation, this report emphasizes the *linguistic mechanisms* of the implemented functions. All mentioned scripts can be found in the GitHub repositories under the ‘src’ or ‘source’ folders, which conventionally contain project source code.

1. nlp_data_visualization (spaCy)

This repository contains a collection of Python scripts designed to help you analyze and visualize text data in various ways. Each script focuses on a specific type of analysis or visualization, providing a comprehensive toolkit for text data exploration.

This project utilizes a range of powerful Python libraries, such as matplotlib, seaborn, wordcloud, spaCy, and textblob, to perform tasks like frequency analysis, sentiment analysis, and part-of-speech tagging. The resulting visualizations, including bar plots, histograms, pie charts, treemaps, violin plots, and word clouds, offer clear and insightful representations of the underlying data.

Some of the libraries used include matplotlib, seaborn, wordcloud, spaCy, and textblob. The graph types include bar plots, histograms, pie charts, treemaps, violin plots, and word clouds. Input and output formats consist of .txt, .xml, .json, .xlsx, .csv, and .sgm. Each individual script is designed to work with one of these visualization techniques and formats.

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.

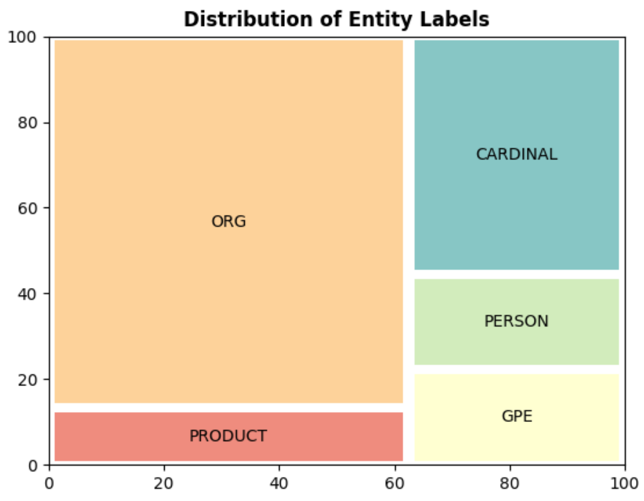


Figure 1.1: Treemap

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.

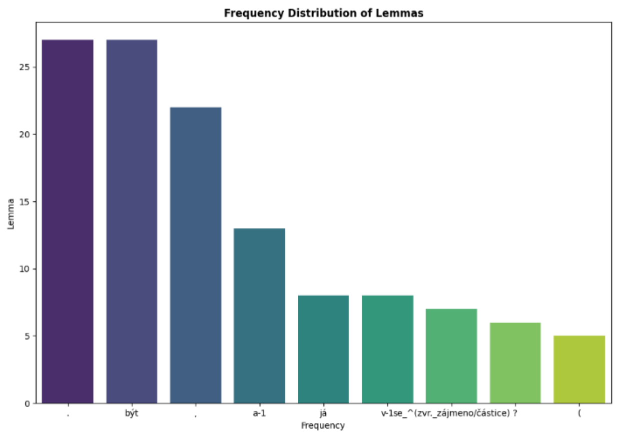


Figure 1.2: Bar Plot

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.

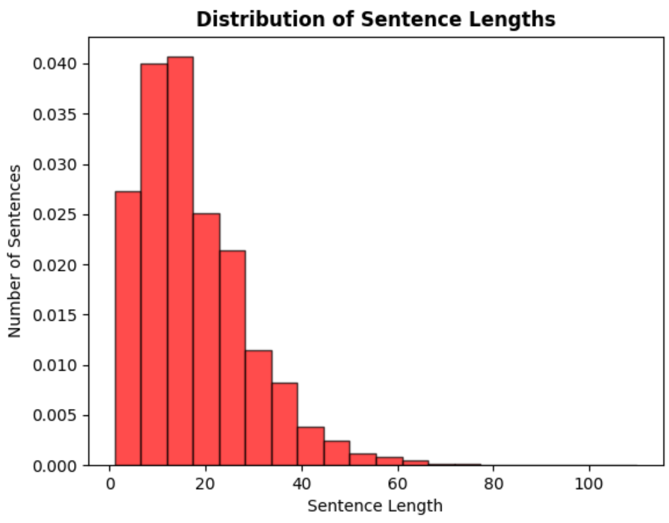


Figure 1.3: Histogram

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.

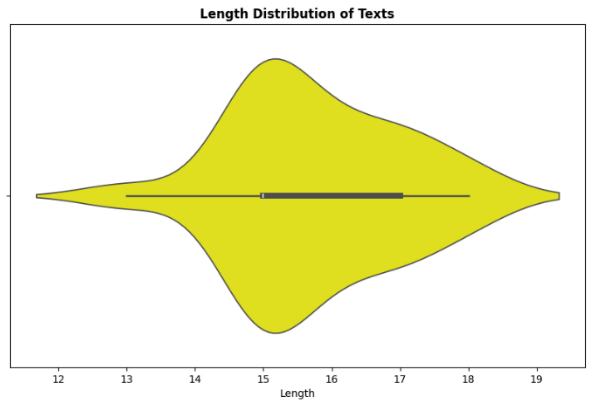


Figure 1.4: Violin Plot

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.

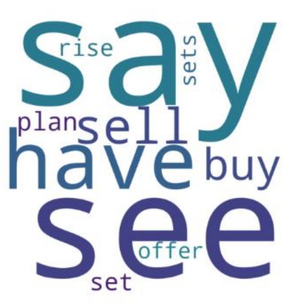


Figure 1.5: Word Cloud

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.

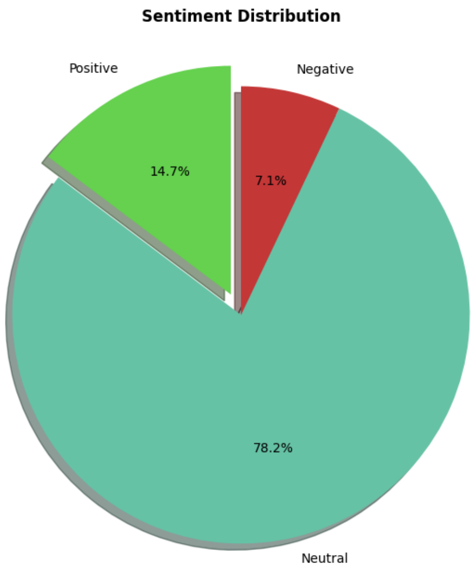


Figure 1.6: Pie Chart

2. nltk_complete (NLTK)

This repository is a practical implementation of the book *Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit*, authored by Steven Bird, Ewan Klein, and Edward Loper. It presents my interpretation and application of the book's examples, providing a hands-on approach to exploring a variety of NLP concepts and techniques. The repository serves as a learning tool for understanding core NLP tasks, such as tokenization, frequency distribution analysis, and syntactic parsing.

The repository encompasses a wide array of linguistic methods, including the calculation of linguistic feature indexes, identifying common words, and visualizing data through plots, histograms, and WordNet graphics. It also covers part-of-speech tagging, length comparisons of different text units, and other methods that build a comprehensive understanding of text processing and analysis. These tools and techniques allow users to gain deeper insights into how textual data can be explored and interpreted.

A key feature of this repository is its focus on the NLTK library, which offers a broader range of options than other libraries like spaCy. NLTK provides users with powerful functionalities, such as syntactic analysis, that are not available in spaCy. In addition to NLTK, the repository also incorporates modules like urllib, b4, feedparser, re, collections, pylab, and pickle, along with various NLTK corpora. This combination allows for a thorough exploration of advanced NLP methods, making it a valuable resource for anyone looking to gain practical experience with natural language processing.

a. 01-language_processing.ipynb

This Jupyter notebook lists the concordance of a given word, finds lexically similar words, and retrieves a dispersion plot for the input words. It also generates a random text. Later, I analyze the token count and divide it by the dictionary size to calculate the Type-Token Ratio (TTR), which is an indicator of vocabulary richness. I also convert it to a percentage. Next, I generate a frequency distribution list to find the most common words. I then sort the tokens and filter them based on a character length limit. Additionally, I retrieve bigrams and collocations. In the final cell, I filter tokens to include only alphabetic ones and count the number of words in the given text.

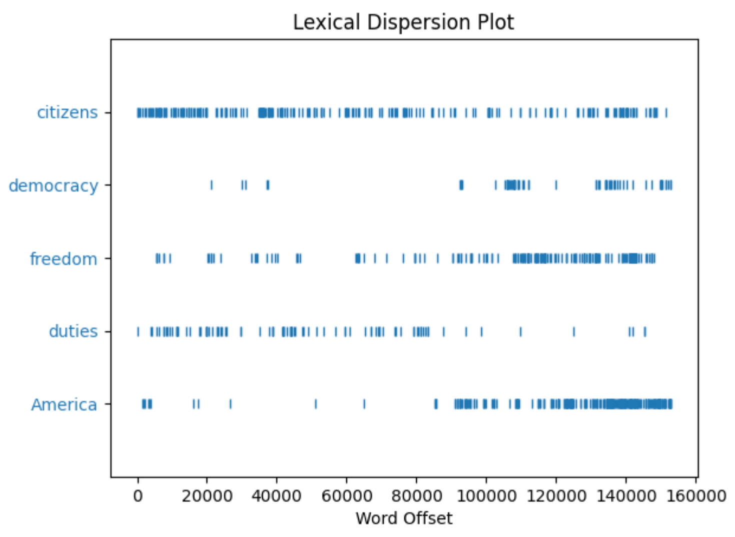


Figure 2.1.1: Dispersion Plot

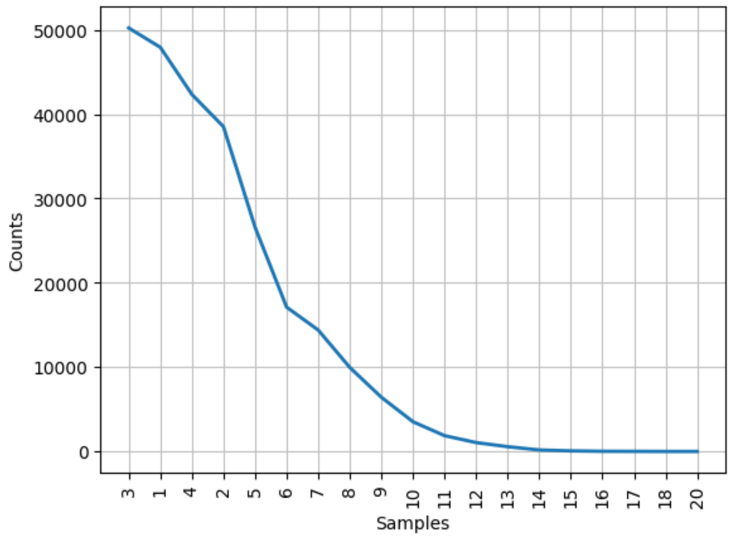


Figure 2.1.2: Frequency Distribution Plot

b. 02-accessing_text_corpora_and_lexical_resources.ipynb

I explore built-in libraries from NLTK. Next, I calculate the average word length, average sentence length, and lexical diversity score (or Type-Token Ratio, TTR). I also examine different genres from various corpora. I generate a frequency list once more and apply additional filters before plotting the results. I compare the word lengths across the same text in different languages and create plots to visualize these differences. I then split the text into bigrams. Additionally, I identify unusual words and plot another frequency distribution list with specific conditions. Finally, I generate an entity list and analyze hyponyms, hypernyms, meronyms, holonyms, antonyms, and the relationships between substances and members based on their syntactic similarity.

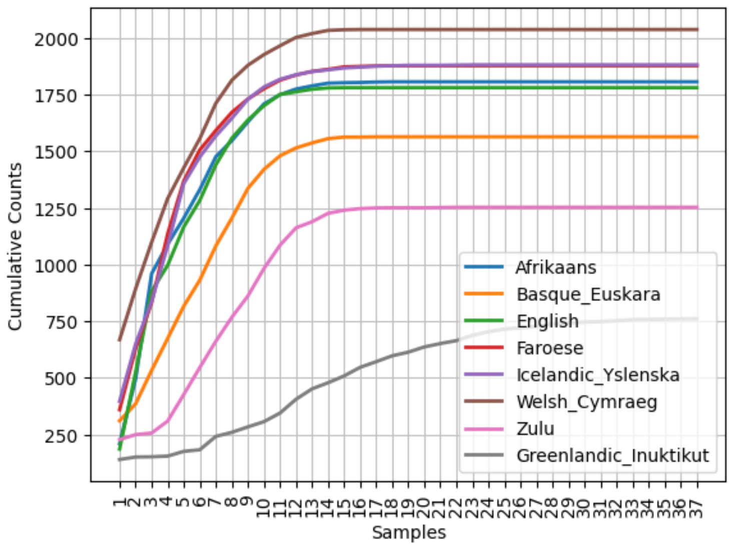


Figure 2.2.1: Word Length Plot of Different Languages

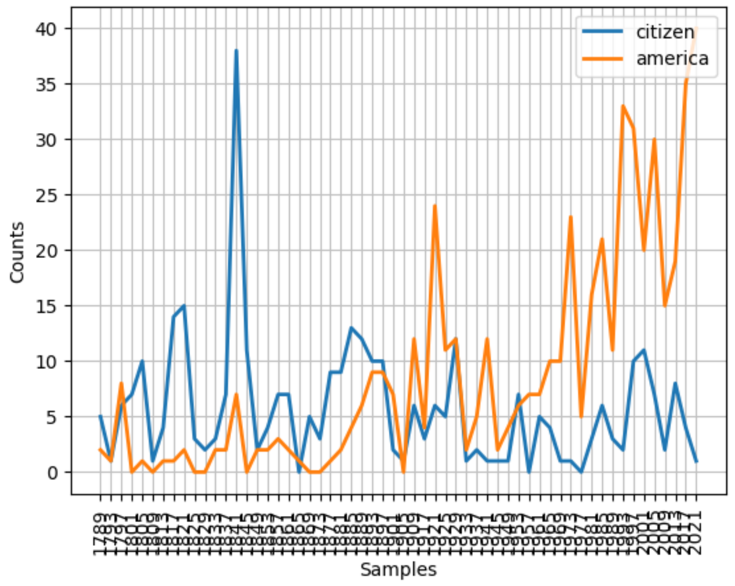
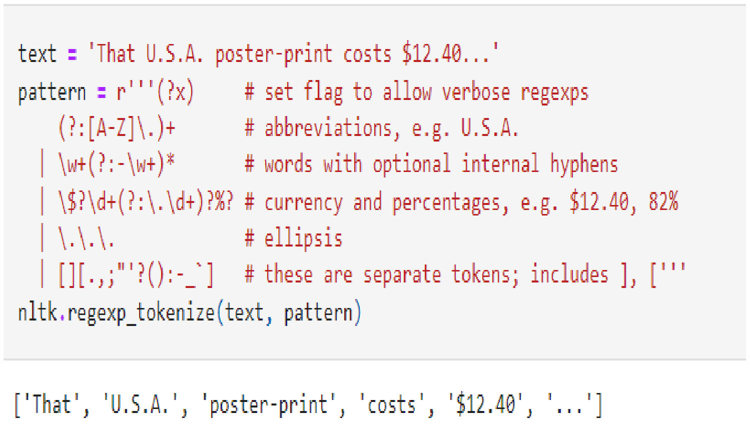


Figure 2.2.2: Word Count Plot of Different Words

c. 03-processing_raw_text.ipynb

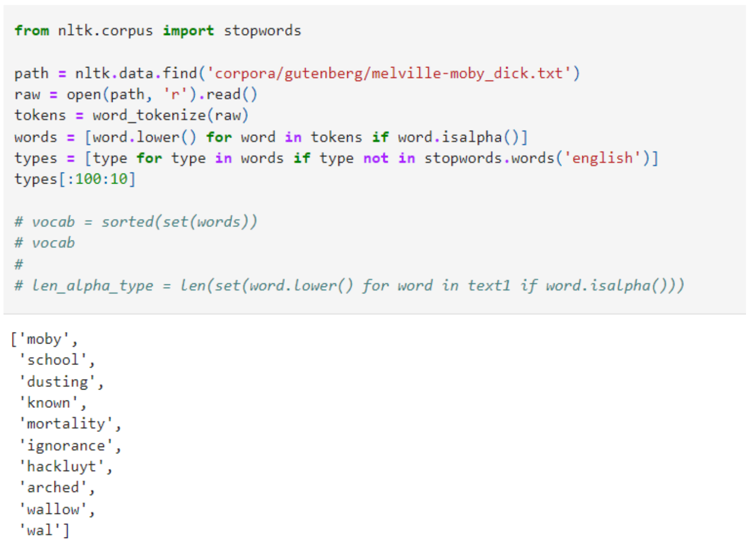
I tokenize the text. Next, I identify collocations and retrieve the concordance matches. In another example, I find which characters are alphabetic, convert them to lowercase, and filter out stop words before manually tokenizing the text. I explore string methods and raw text processing techniques. I then provide an illustration of regular expressions. I identify the most common instances of vowel pairs and create a conditional frequency list. Finally, I stem the tokens.



```
text = 'That U.S.A. poster-print costs $12.40...'
pattern = r'''(?x)     # set flag to allow verbose regexps
    (?:[A-Z]\.)+       # abbreviations, e.g. U.S.A.
  | \w+(?:-\w+)*       # words with optional internal hyphens
  | \$?\d+(?:\.\d+)?%? # currency and percentages, e.g. $12.40, 82%
  | \.\.\.             # ellipsis
  | [][.,;"'?():-_`]  # these are separate tokens; includes ], ['''
nltk.regexp_tokenize(text, pattern)
```

```
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Figure 2.3.1: Regular Expression Patterns



```
from nltk.corpus import stopwords

path = nltk.data.find('corpora/gutenberg/melville-moby_dick.txt')
raw = open(path, 'r').read()
tokens = word_tokenize(raw)
words = [word.lower() for word in tokens if word.isalpha()]
types = [type for type in words if type not in stopwords.words('english')]
types[:100:10]

# vocab = sorted(set(words))
# vocab
#
# len_alpha_type = len(set(word.lower() for word in text1 if word.isalpha()))
```

```
['moby',
 'school',
 'dusting',
 'known',
 'mortality',
 'ignorance',
 'hackluyt',
 'arched',
 'wallow',
 'wal']
```

Figure 2.3.2: Raw Text Processing

d- 04-writing_structured_programs.ipynb

Initially, I examine list indexing and traversing methods. Secondly, I create a conditional frequency list and plot the modal verbs based on their usage genre. I then analyze hyponyms, create a new WordNet graph, and traverse it. Later, I draw the graph. Finally, I demonstrate the transposition of an array with a simple example and apply the SVD decomposition method to it.

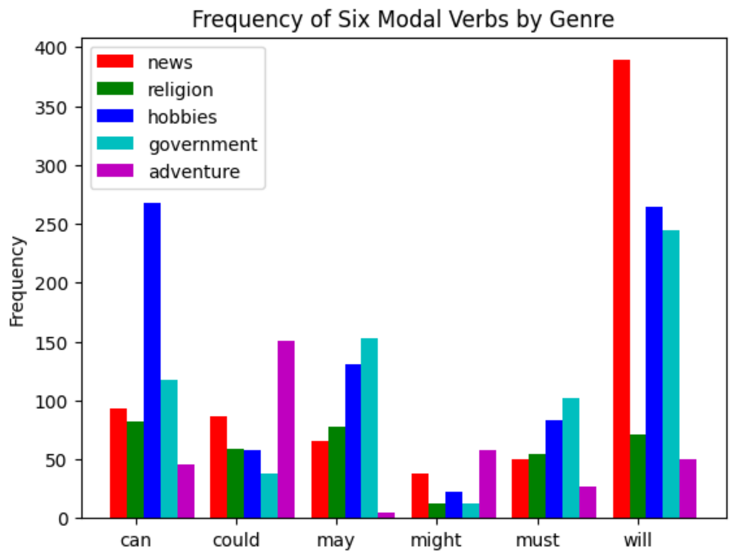


Figure 2.4.1: Frequency Distribution Graph of Model Verbs by Genre

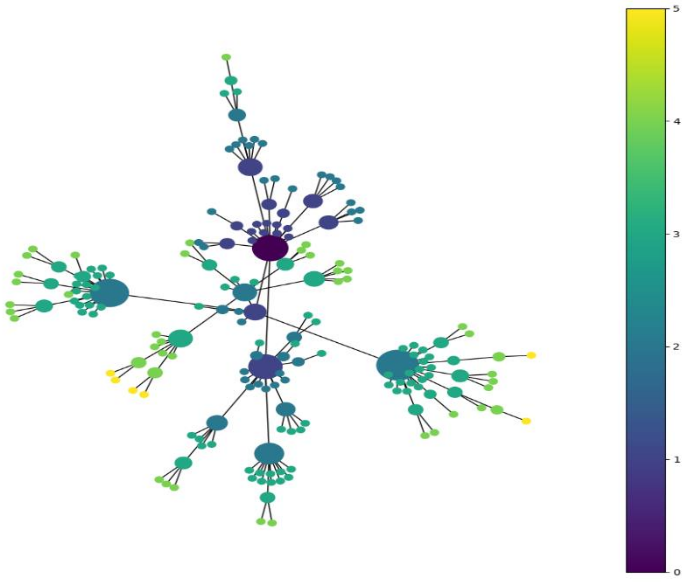
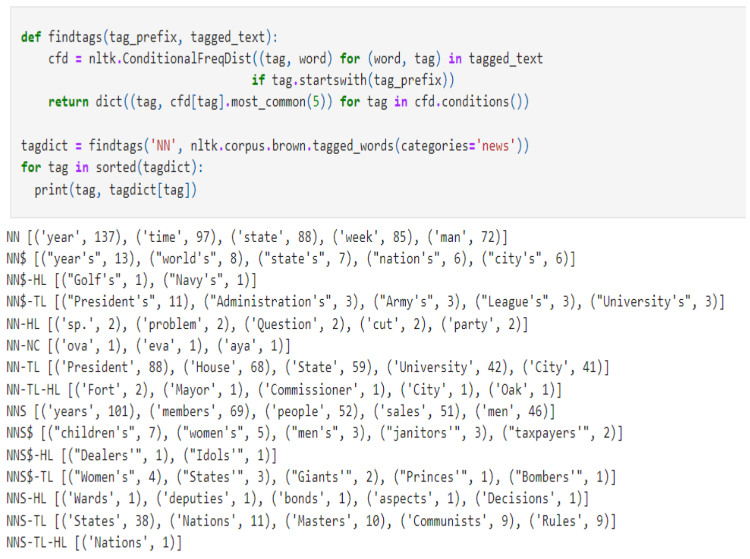


Figure 2.4.2: WordNet Graph

d. 05-categorizing_and_tagging_words.ipynb

I tokenize a text and parse the words based on their part-of-speech roles. I then extract the frequency distribution list and identify the most common words. I apply the same procedure to bigrams as well. I work with the words from the Treebank corpus. Next, I create a conditional frequency distribution list and analyze their part-of-speech tags. I count the occurrences of these roles in the text. For more advanced queries, I use regular expressions and create a unigram tagger. I then build upon it to develop bigram and trigram taggers. Finally, I evaluate their accuracy using a built-in testing method.



```python
def findtags(tag_prefix, tagged_text):
    cfd = nltk.ConditionalFreqDist((tag, word) for (word, tag) in tagged_text
                                  if tag.startswith(tag_prefix))
    return dict((tag, cfd[tag].most_common(5)) for tag in cfd.conditions())

tagdict = findtags('NN', nltk.corpus.brown.tagged_words(categories='news'))
for tag in sorted(tagdict):
  print(tag, tagdict[tag])
```

```
NN [('year', 137), ('time', 97), ('state', 88), ('week', 85), ('man', 72)]
NN$ [("year's", 13), ("world's", 8), ("state's", 7), ("nation's", 6), ("city's", 6)]
NN$-HL [("Golf's", 1), ("Navy's", 1)]
NN$-TL [("President's", 11), ("Administration's", 3), ("Army's", 3), ("League's", 3), ("University's", 3)]
NN-HL [('sp.', 2), ('problem', 2), ('Question', 2), ('cut', 2), ('party', 2)]
NN-NC [('ova', 1), ('eva', 1), ('aya', 1)]
NN-TL [('President', 88), ('House', 68), ('State', 59), ('University', 42), ('City', 41)]
NN-TL-HL [('Fort', 2), ('Mayor', 1), ('Commissioner', 1), ('City', 1), ('Oak', 1)]
NNS [('years', 101), ('members', 69), ('people', 52), ('sales', 51), ('men', 46)]
NNS$ [("children's", 7), ("women's", 5), ("men's", 3), ("janitors'", 3), ("taxpayers'", 2)]
NNS$-HL [("Dealers'", 1), ("Idols'", 1)]
NNS$-TL [("Women's", 4), ("States'", 3), ("Giants'", 2), ("Princes'", 1), ("Bombers'", 1)]
NNS-HL [('Wards', 1), ('deputies', 1), ('bonds', 1), ('aspects', 1), ('Decisions', 1)]
NNS-TL [('States', 38), ('Nations', 11), ('Masters', 10), ('Communists', 9), ('Rules', 9)]
NNS-TL-HL [('Nations', 1)]
```

Figure 2.5.1: Conditional Frequency List with POS tagging

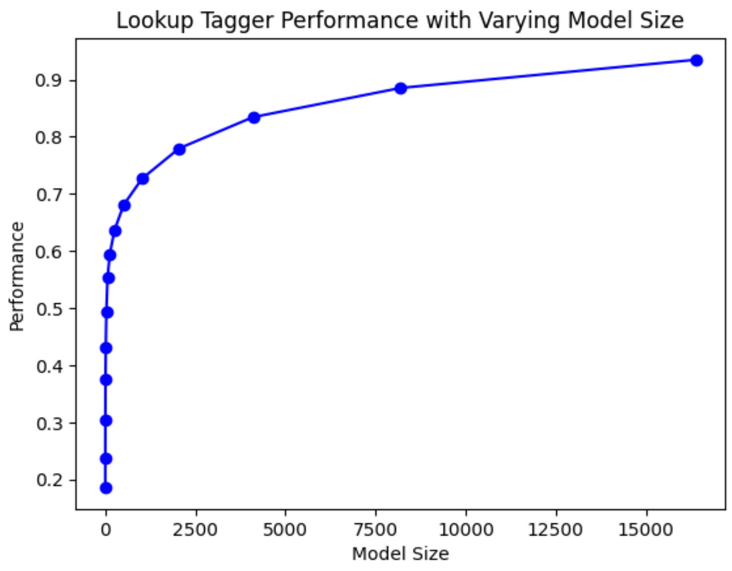


Figure 2.5.2: Performance Evaluation Graph

The subsequent notebooks following **05-categorizing_and_tagging_words.ipynb** are beyond the immediate scope of this discussion, as they delve into more formal methods and advanced techniques in natural language processing. These include topics such as grammars, decision trees, formal grammars, classification algorithms, chunking methods, and logical approaches like propositional and first-order logic. Additionally, they cover the preparation and processing of corpora for various tasks.

These formal methodologies are crucial for developing more sophisticated models and solutions in NLP, but they require a deeper understanding of computational linguistics, machine learning, and logic. While they are not part of the current focus, they represent an important aspect of the broader field of text analysis and can serve as the foundation for more complex systems.

For readers interested in exploring these advanced topics further, I highly encourage consulting the project's Github repository, where you can find the relevant notebooks, detailed explanations, and additional resources. The repository contains valuable information and code examples for anyone looking to expand their knowledge or apply these techniques in practice.

By reviewing these materials, readers will gain a stronger understanding of how formal methods contribute to the development of NLP models and systems, enabling them to tackle more complex linguistic challenges and improve their proficiency in computational linguistics.

Conclusion

This report concluded by showcasing the capabilities of two GitHub repositories, nlp_data_visualization and nltk_complete, which are both based on popular NLP Python libraries. These libraries are essential resources for Python programming that deal with Natural Language Processing (NLP).

Instead, I focused on each script's linguistic evidence rather than just the programming part. By going into more detail about these characteristics, I wanted to give readers a better grasp of how different approaches and analyses aid in the exploration of language data.

All things considered, the repositories are excellent tools for anyone wishing to learn or hone their NLP abilities. In order to efficiently process and analyze text data within linguistic circles, they offer both the computational tools, and the linguistic insights required.