Natural Language Processing
AI-829

Nested Summarization with Heading Hierarchy

Mandate-3
Syntax Analysis

Presented By:
Aparajita Kumari(MT2023037)
Nandini Yadav(MT2023096)

Contents

1. Recap of Lexical Processing and Word Embedding

In this mandate, we first break down the text into individual words to understand their distribution. Then, we clean up the text to remove any errors or inconsistencies. After that, we make different word forms uniform to improve consistency. Finally, we use Word2Vec to turn words into numbers, making it easier for computers to understand and work with the text. These steps help us prepare the textual data for analysis and modeling by ensuring it's organized, clean, and represented in a way that machines can interpret effectively.

Explanation of each step:

- **Tokenization and Zipf distribution analysis**: This step breaks down the text into individual words or tokens and helps us understand how often each word appears. By knowing which words are common and rare, we can prepare the text better for analysis.

- **Preprocessing techniques:** These techniques make sure that the text is clean and consistent, without any unnecessary information or errors. It's like tidying up the text so that machines can understand it better.

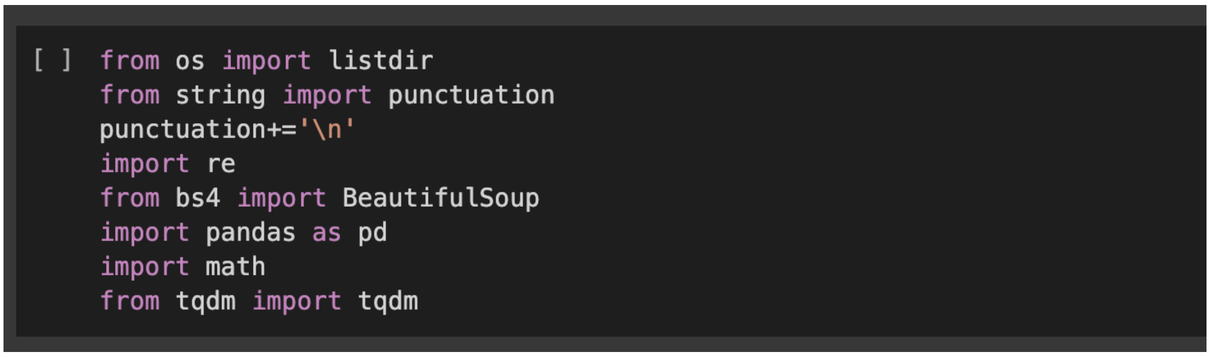
- **Canonicalization techniques:** These methods make different forms of words (like "running" and "ran") into a single, standard form (like "run"). This helps in keeping the text consistent and makes it easier for computers to understand.

2. Overview of Mandate 3

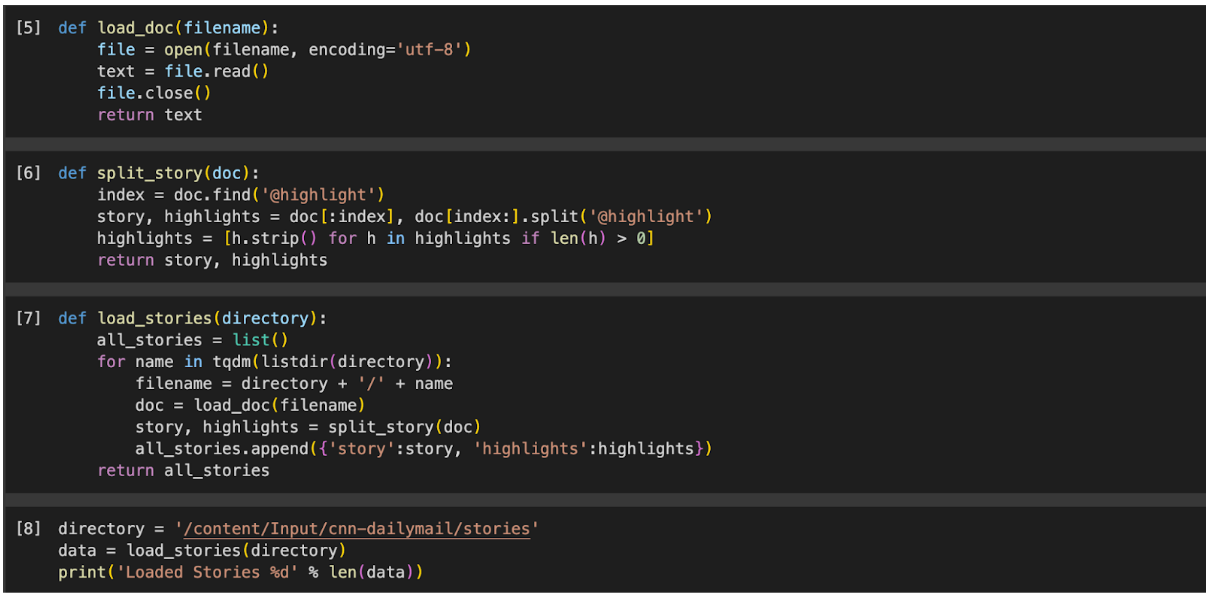
- The code utilizes natural language processing (NLP) tools such as NLTK, Stanza, and spaCy for further analysis.
- It performs part-of-speech tagging to identify the grammatical categories of words in the text.
- Additionally, it uses dependency parsing to analyze the syntactic structure of sentences.
- Finally, it visualizes the dependency parse tree using the displaCy library from spaCy, providing a graphical representation of how words in a sentence relate to each other.
- The code also aims to implement coreference resolution to identify and link coreferent mentions in the text, enhancing semantic understanding. However, integrating coreference resolution presents challenges due to compatibility issues and complexities in implementing advanced NLP techniques.

3. Loading Data

The code starts by importing necessary libraries such as os, string, pandas, tqdm, nltk, contractions, stanza, and spacy.



```
[ ]  from os import listdir
     from string import punctuation
     punctuation+='\n'
     import re
     from bs4 import BeautifulSoup
     import pandas as pd
     import math
     from tqdm import tqdm
```



```
[5] def load_doc(filename):
        file = open(filename, encoding='utf-8')
        text = file.read()
        file.close()
        return text

[6] def split_story(doc):
        index = doc.find('@highlight')
        story, highlights = doc[:index], doc[index:].split('@highlight')
        highlights = [h.strip() for h in highlights if len(h) > 0]
        return story, highlights

[7] def load_stories(directory):
        all_stories = list()
        for name in tqdm(listdir(directory)):
            filename = directory + '/' + name
            doc = load_doc(filename)
            story, highlights = split_story(doc)
            all_stories.append({'story':story, 'highlights':highlights})
        return all_stories

[8] directory = '/content/Input/cnn-dailymail/stories'
    data = load_stories(directory)
    print('Loaded Stories %d' % len(data))
```

This defines functions to load documents, split stories into main content and highlights, and load multiple stories from a directory. Then, it loads stories from a specified directory, preprocesses them by removing unnecessary characters and expanding contractions, and stores them in a pandas DataFrame.

4. Syntax Analysis

Syntax analysis involves understanding the structure of sentences to determine relationships between words. POS tagging assigns parts of speech (like noun, verb) to each word in a sentence. Parsing breaks down sentences into a hierarchical structure, showing how words relate to each other grammatically. Coreference resolution identifies when pronouns refer to the same entity mentioned earlier in the text. These processes are essential in natural language understanding, enabling machines to comprehend and interpret text accurately.

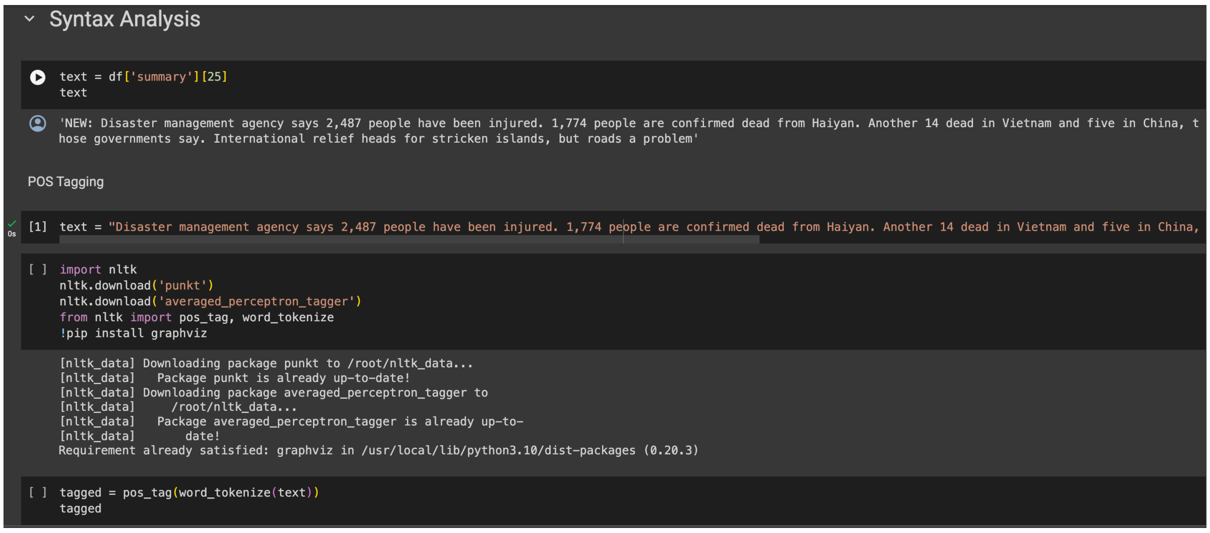
4.1 POS Tagging:

POS tags provide valuable information for syntactic parsing, where the structure of sentences is analyzed. Understanding the syntactic relationships between words is essential for tasks such as sentence parsing and grammar checking.

<u>Information Extraction:</u> POS tagging aids in extracting specific information from text. For instance, identifying nouns can help extract entities such as names, locations, and organizations from a document.

<u>Example:</u>
For example, in the text "The top YOU.S. intel official says leaks affect national security," NLTK has tagged each word as follows:

"The" as determiner (DT)
"top" as adjective (JJ)
"YOU.S." as proper noun (NNP)
"intel" as noun (NN)
"official" as noun (NN)
"says" as verb (VBZ)
"leaks" as noun (NNS)
"affect" as verb (VBP)
"national" as adjective (JJ)
"security" as noun (NN)
And so on, for each word in the sentence. This tagging provides valuable insights into the grammatical structure of the text, enabling further analysis and processing based on the roles of individual words within the sentence.


Syntax Analysis

```
text = df['summary'][25]
text
```

```
'NEW: Disaster management agency says 2,487 people have been injured. 1,774 people are confirmed dead from Haiyan. Another 14 dead in Vietnam and five in China, t
hose governments say. International relief heads for stricken islands, but roads a problem'
```

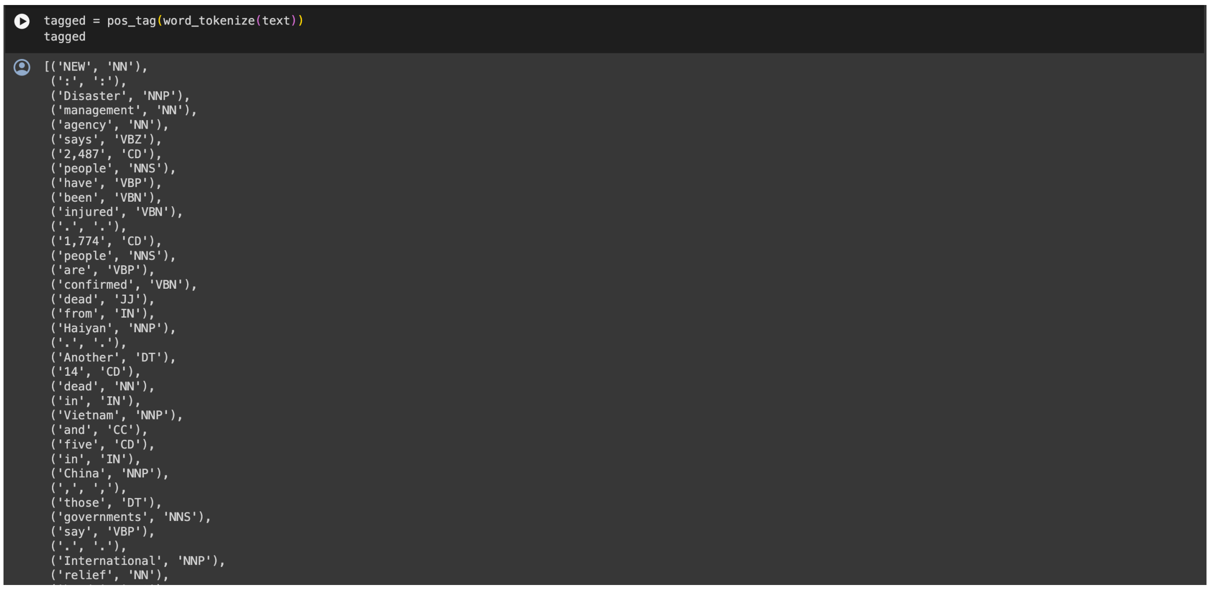
POS Tagging

```
[1] text = "Disaster management agency says 2,487 people have been injured. 1,774 people are confirmed dead from Haiyan. Another 14 dead in Vietnam and five in China,
```

```
[ ] import nltk
    nltk.download('punkt')
    nltk.download('averaged_perceptron_tagger')
    from nltk import pos_tag, word_tokenize
    !pip install graphviz
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (0.20.3)
```

```
[ ] tagged = pos_tag(word_tokenize(text))
    tagged
```

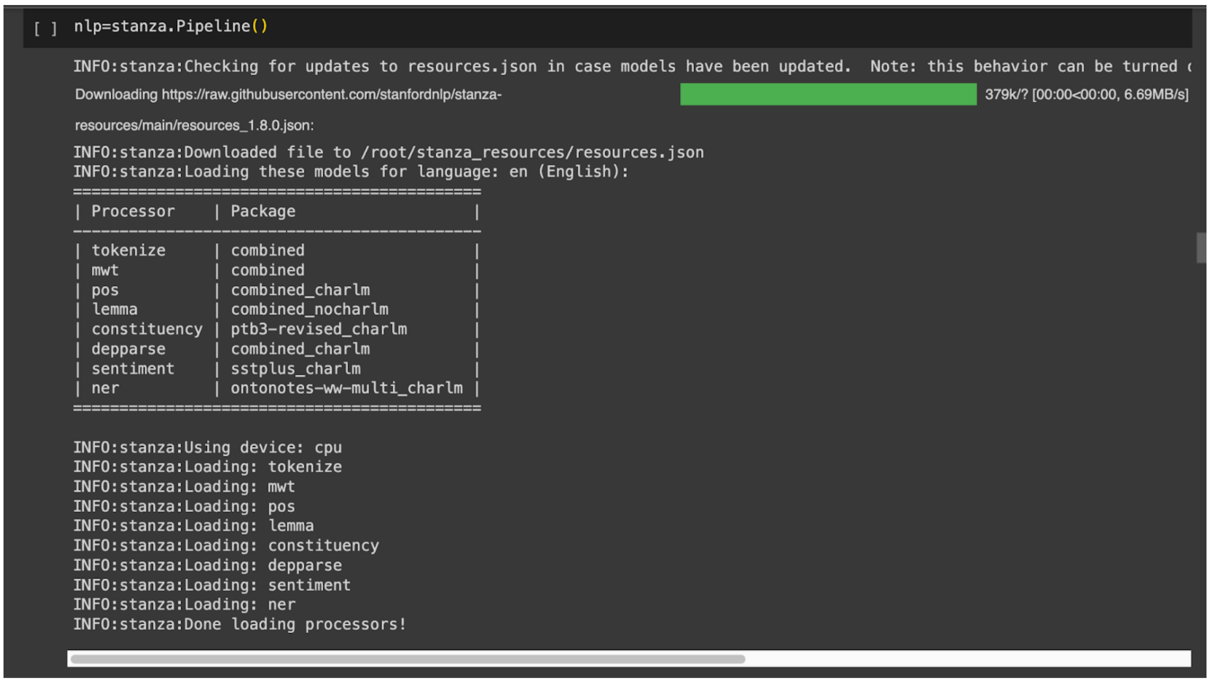


```
tagged = pos_tag(word_tokenize(text))
tagged
```

```
[('NEW', 'NN'),
 (':', ':'),
 ('Disaster', 'NNP'),
 ('management', 'NN'),
 ('agency', 'NN'),
 ('says', 'VBZ'),
 ('2,487', 'CD'),
 ('people', 'NNS'),
 ('have', 'VBP'),
 ('been', 'VBN'),
 ('injured', 'VBN'),
 ('.', '.'),
 ('1,774', 'CD'),
 ('people', 'NNS'),
 ('are', 'VBP'),
 ('confirmed', 'VBN'),
 ('dead', 'JJ'),
 ('from', 'IN'),
 ('Haiyan', 'NNP'),
 ('.', '.'),
 ('Another', 'DT'),
 ('14', 'CD'),
 ('dead', 'NN'),
 ('in', 'IN'),
 ('Vietnam', 'NNP'),
 ('and', 'CC'),
 ('five', 'CD'),
 ('in', 'IN'),
 ('China', 'NNP'),
 (',', ','),
 ('those', 'DT'),
 ('governments', 'NNS'),
 ('say', 'VBP'),
 ('.', '.'),
 ('International', 'NNP'),
 ('relief', 'NN'),
```

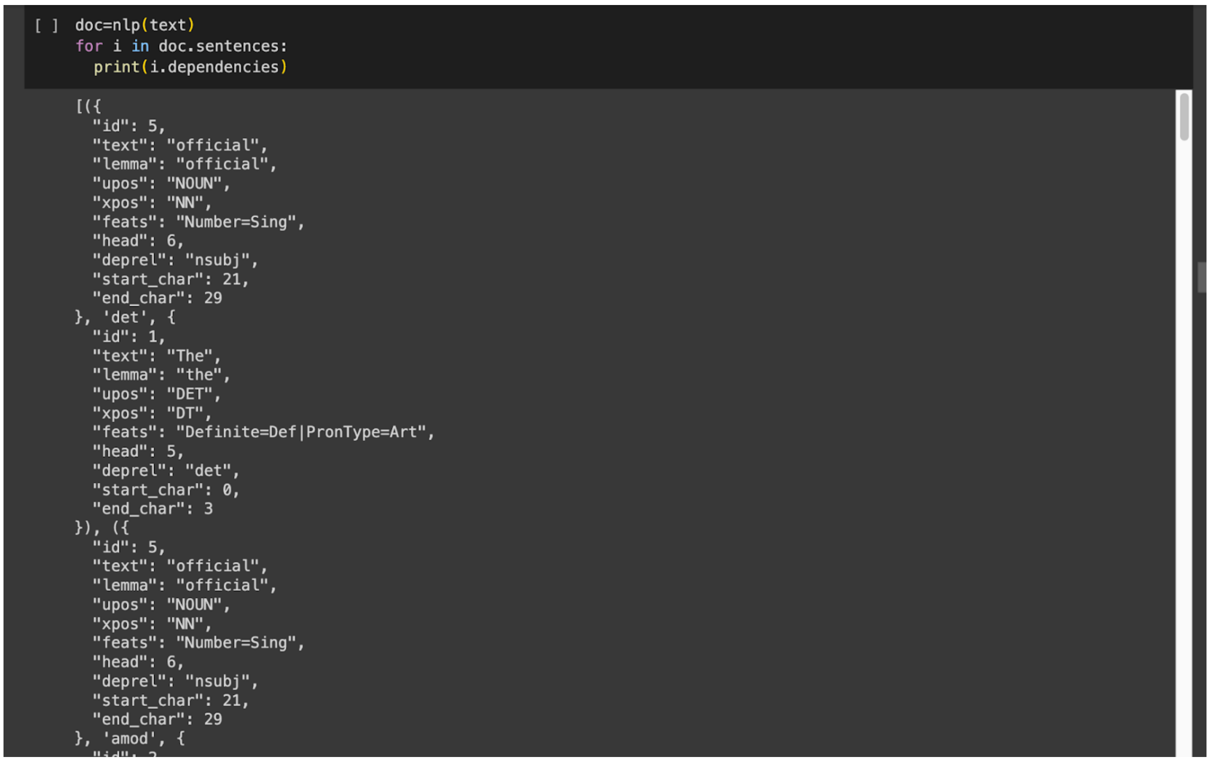
4.2 Dependency Parsing:

Dependency parsing analyzes the syntactic structure of sentences by identifying relationships between words and representing them as a tree structure. Stanza, a powerful NLP library, offers dependency parsing capabilities, allowing us to generate parse trees that illustrate the hierarchical relationships between words. Parse trees provide insights into the grammatical structure of sentences, aiding in understanding sentence semantics and relationships between words.



```
[ ] nlp=stanza.Pipeline()

INFO:stanza:Checking for updates to resources.json in case models have been updated.  Note: this behavior can be turned
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-          379k/? [00:00<00:00, 6.69MB/s]
resources/main/resources_1.8.0.json:
INFO:stanza:Downloaded file to /root/stanza_resources/resources.json
INFO:stanza:Loading these models for language: en (English):
============================================
| Processor    | Package                   |
--------------------------------------------
| tokenize     | combined                  |
| mwt          | combined                  |
| pos          | combined_charlm           |
| lemma        | combined_nocharlm         |
| constituency | ptb3-revised_charlm       |
| depparse     | combined_charlm           |
| sentiment    | sstplus_charlm            |
| ner          | ontonotes-ww-multi_charlm |
============================================

INFO:stanza:Using device: cpu
INFO:stanza:Loading: tokenize
INFO:stanza:Loading: mwt
INFO:stanza:Loading: pos
INFO:stanza:Loading: lemma
INFO:stanza:Loading: constituency
INFO:stanza:Loading: depparse
INFO:stanza:Loading: sentiment
INFO:stanza:Loading: ner
INFO:stanza:Done loading processors!
```

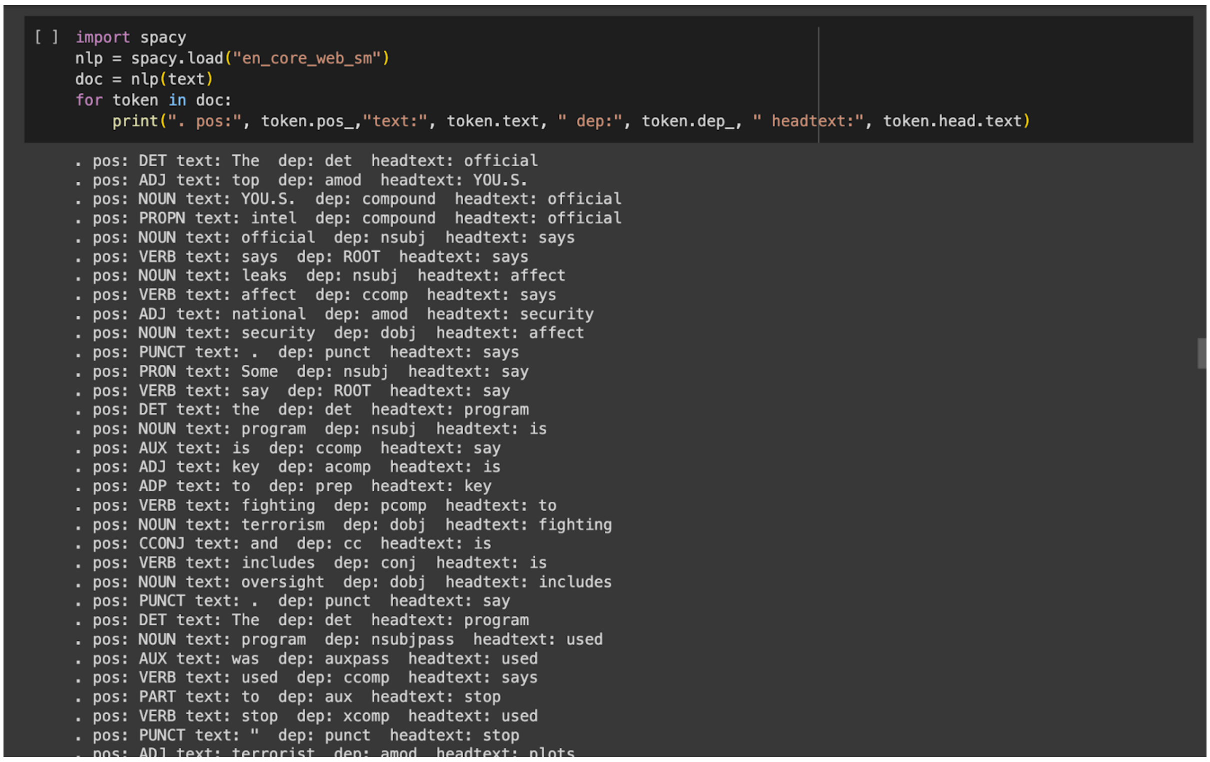
This is the following structure we get through stanza pipeline:



```
[ ] doc=nlp(text)
    for i in doc.sentences:
      print(i.dependencies)

    [({
      "id": 5,
      "text": "official",
      "lemma": "official",
      "upos": "NOUN",
      "xpos": "NN",
      "feats": "Number=Sing",
      "head": 6,
      "deprel": "nsubj",
      "start_char": 21,
      "end_char": 29
    }, 'det', {
      "id": 1,
      "text": "The",
      "lemma": "the",
      "upos": "DET",
      "xpos": "DT",
      "feats": "Definite=Def|PronType=Art",
      "head": 5,
      "deprel": "det",
      "start_char": 0,
      "end_char": 3
    }), ({
      "id": 5,
      "text": "official",
      "lemma": "official",
      "upos": "NOUN",
      "xpos": "NN",
      "feats": "Number=Sing",
      "head": 6,
      "deprel": "nsubj",
      "start_char": 21,
      "end_char": 29
    }, 'amod', {
```

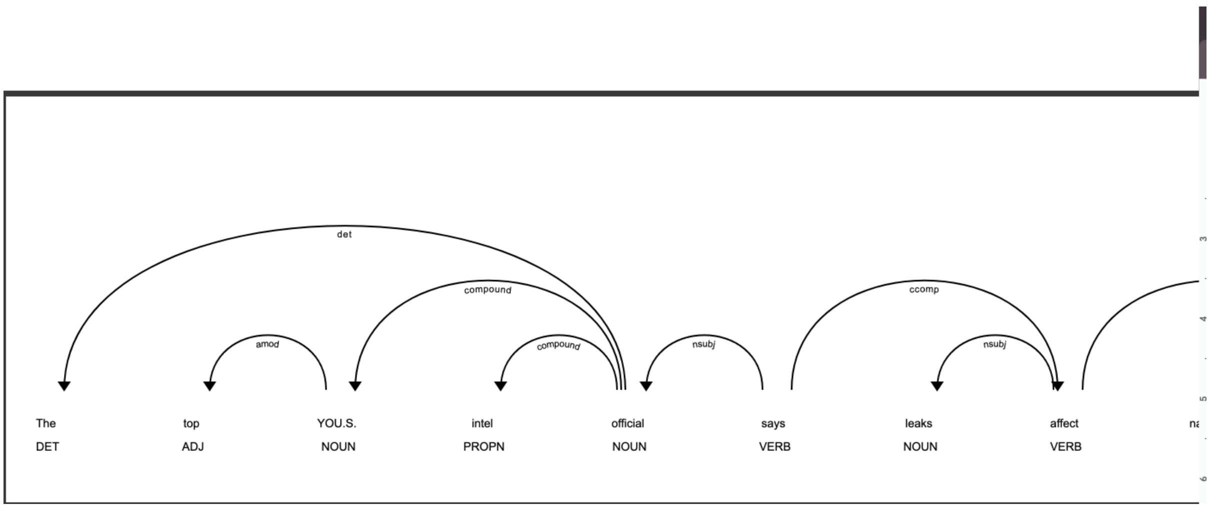
4.3 Parse Tree:

We utilized the Stanza library for syntactic analysis, parsing the text to understand its grammatical structure. Additionally, we employed Spacy to visualize the parse tree, providing a graphical representation of how words in the sentence relate to each other syntactically.



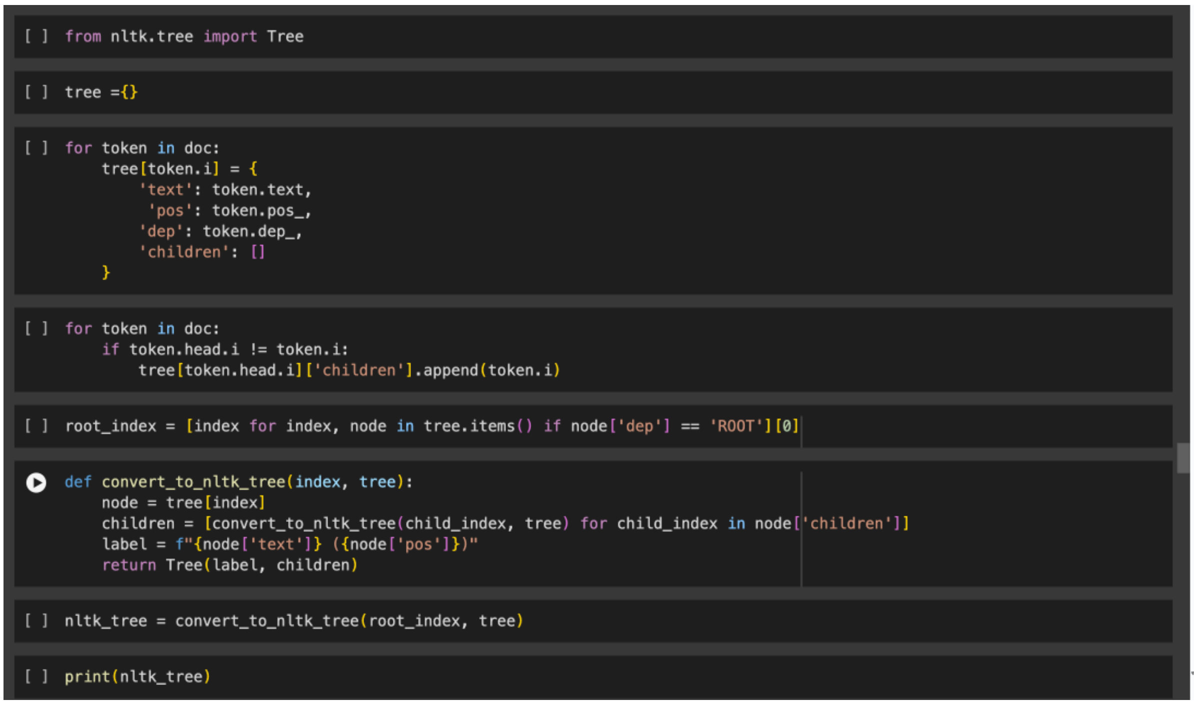
```python
[ ] import spacy
    nlp = spacy.load("en_core_web_sm")
    doc = nlp(text)
    for token in doc:
        print(". pos:", token.pos_,"text:", token.text, " dep:", token.dep_, " headtext:", token.head.text)
```

```
. pos: DET text: The  dep: det  headtext: official
. pos: ADJ text: top  dep: amod  headtext: YOU.S.
. pos: NOUN text: YOU.S.  dep: compound  headtext: official
. pos: PROPN text: intel  dep: compound  headtext: official
. pos: NOUN text: official  dep: nsubj  headtext: says
. pos: VERB text: says  dep: ROOT  headtext: says
. pos: NOUN text: leaks  dep: nsubj  headtext: affect
. pos: VERB text: affect  dep: ccomp  headtext: says
. pos: ADJ text: national  dep: amod  headtext: security
. pos: NOUN text: security  dep: dobj  headtext: affect
. pos: PUNCT text: .  dep: punct  headtext: says
. pos: PRON text: Some  dep: nsubj  headtext: say
. pos: VERB text: say  dep: ROOT  headtext: say
. pos: DET text: the  dep: det  headtext: program
. pos: NOUN text: program  dep: nsubj  headtext: is
. pos: AUX text: is  dep: ccomp  headtext: say
. pos: ADJ text: key  dep: acomp  headtext: is
. pos: ADP text: to  dep: prep  headtext: key
. pos: VERB text: fighting  dep: pcomp  headtext: to
. pos: NOUN text: terrorism  dep: dobj  headtext: fighting
. pos: CCONJ text: and  dep: cc  headtext: is
. pos: VERB text: includes  dep: conj  headtext: is
. pos: NOUN text: oversight  dep: dobj  headtext: includes
. pos: PUNCT text: .  dep: punct  headtext: say
. pos: DET text: The  dep: det  headtext: program
. pos: NOUN text: program  dep: nsubjpass  headtext: used
. pos: AUX text: was  dep: auxpass  headtext: used
. pos: VERB text: used  dep: ccomp  headtext: says
. pos: PART text: to  dep: aux  headtext: stop
. pos: VERB text: stop  dep: xcomp  headtext: used
. pos: PUNCT text: "  dep: punct  headtext: stop
. pos: ADJ text: terrorist  dep: amod  headtext: plots
```

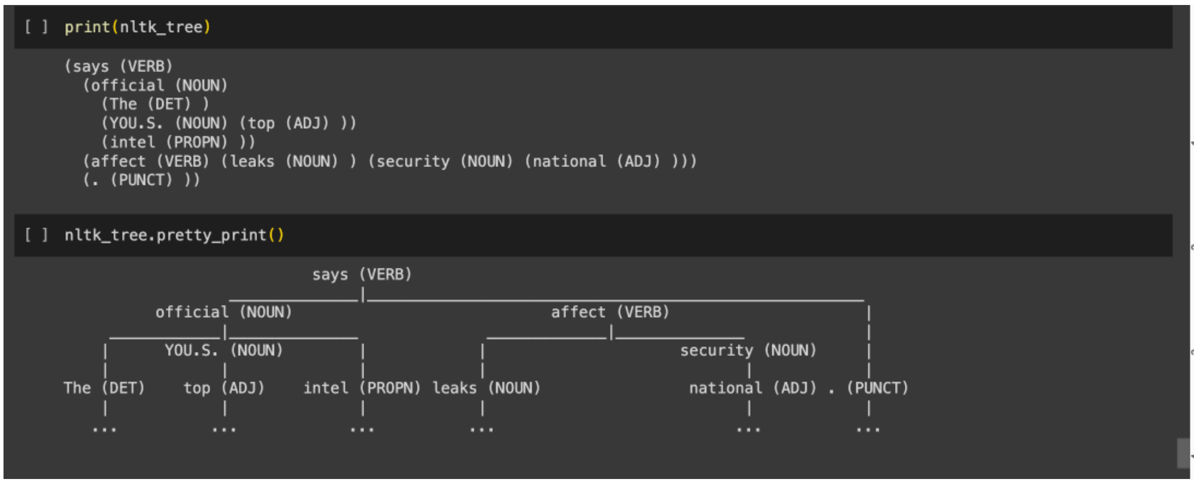


We have tried to explore various other methods such as :

NLTK:



```
[ ] from nltk.tree import Tree

[ ] tree ={}

[ ] for token in doc:
        tree[token.i] = {
            'text': token.text,
             'pos': token.pos_,
            'dep': token.dep_,
            'children': []
        }

[ ] for token in doc:
        if token.head.i != token.i:
            tree[token.head.i]['children'].append(token.i)

[ ] root_index = [index for index, node in tree.items() if node['dep'] == 'ROOT'][0]

    def convert_to_nltk_tree(index, tree):
        node = tree[index]
        children = [convert_to_nltk_tree(child_index, tree) for child_index in node['children']]
        label = f"{node['text']} ({node['pos']})"
        return Tree(label, children)

[ ] nltk_tree = convert_to_nltk_tree(root_index, tree)

[ ] print(nltk_tree)
```



```
[ ] print(nltk_tree)

    (says (VERB)
      (official (NOUN)
        (The (DET) )
        (YOU.S. (NOUN) (top (ADJ) ))
        (intel (PROPN) ))
      (affect (VERB) (leaks (NOUN) ) (security (NOUN) (national (ADJ) )))
      (. (PUNCT) ))

[ ] nltk_tree.pretty_print()

                               says (VERB)
                      ______________|______________________________________________________
              official (NOUN)                              affect (VERB)                     |
         ____________|______________              _____________|______________             |
        |      YOU.S. (NOUN)        |            |                     security (NOUN)     |
        |            |              |            |                            |            |
    The (DET)    top (ADJ)    intel (PROPN) leaks (NOUN)                national (ADJ) . (PUNCT)
        |            |              |            |                            |            |
       ...          ...            ...          ...                          ...          ...
```

4.4 Coreference Resolution:

Coreference refers to the phenomenon where different words or phrases in a text refer to the same entity. Coreference resolution aims to identify and link these references together.
The coreference scores matrix simulates the likelihood of coreference between pairs of tokens in the text. Higher scores indicate a higher likelihood of coreference.

BERT-MODEL: this was the model that worked for us.

Importing Libraries and Loading BERT Model:

The code begins by importing necessary libraries such as transformers and tensorflow.
It loads the pre-trained BERT model and tokenizer from the 'bert-base-uncased' variant. BERT (Bidirectional Encoder Representations from Transformers) is a powerful pre-trained language model widely used in various NLP tasks.

Tokenization and Encoding:

The target text from df['summary'][25] (which presumably is a summary from a dataset) is tokenized and encoded using the BERT tokenizer.
Tokenization breaks the text into individual tokens (words or subwords), and encoding converts these tokens into numerical representations suitable for BERT.

Inference and Embedding Extraction:

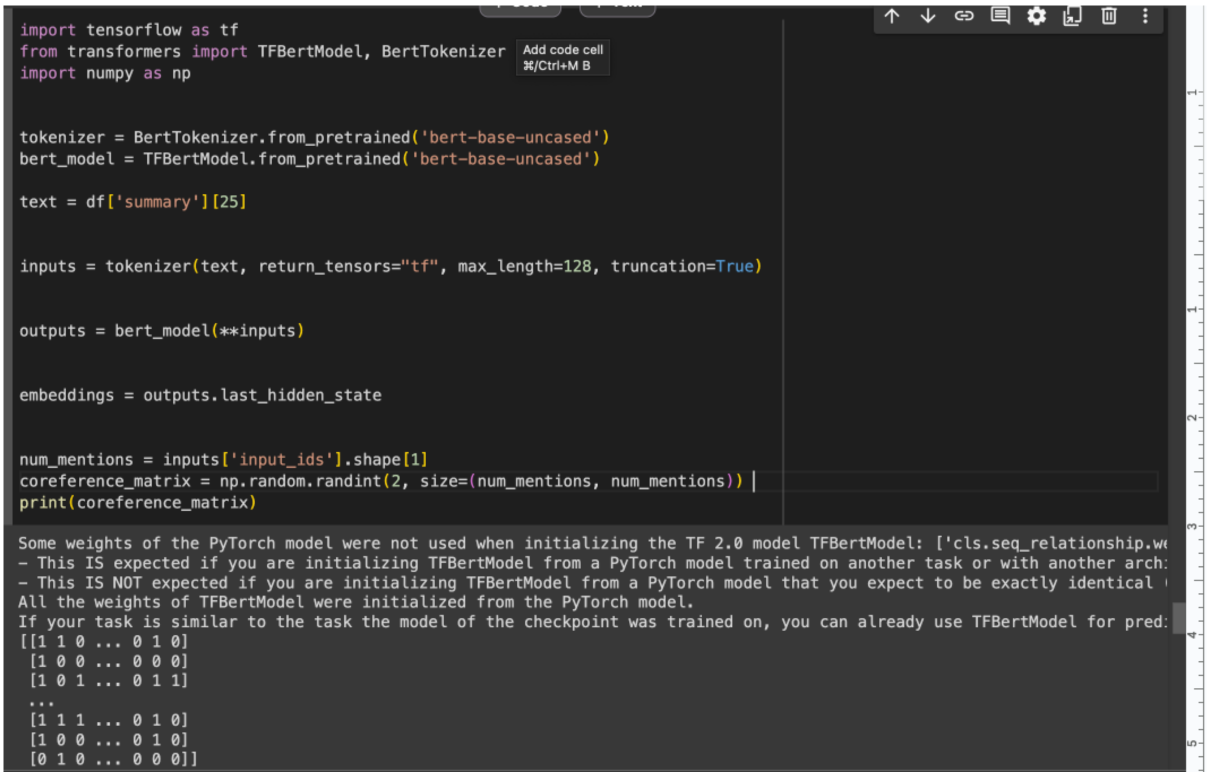
The encoded inputs are fed into the BERT model for inference, which generates output embeddings. Specifically, the last hidden states are extracted from the BERT model outputs. These last hidden states represent contextualized embeddings for each token in the input text.By capturing contextual information, these embeddings aid in understanding the semantic relationships between tokens.

Coreference Score Calculation:

The code generates coreference scores between pairs of tokens.Coreference scores between pairs of tokens are calculated based on the embeddings generated by BERT. These scores indicate the likelihood of coreference between tokens, with higher scores suggesting a stronger association.

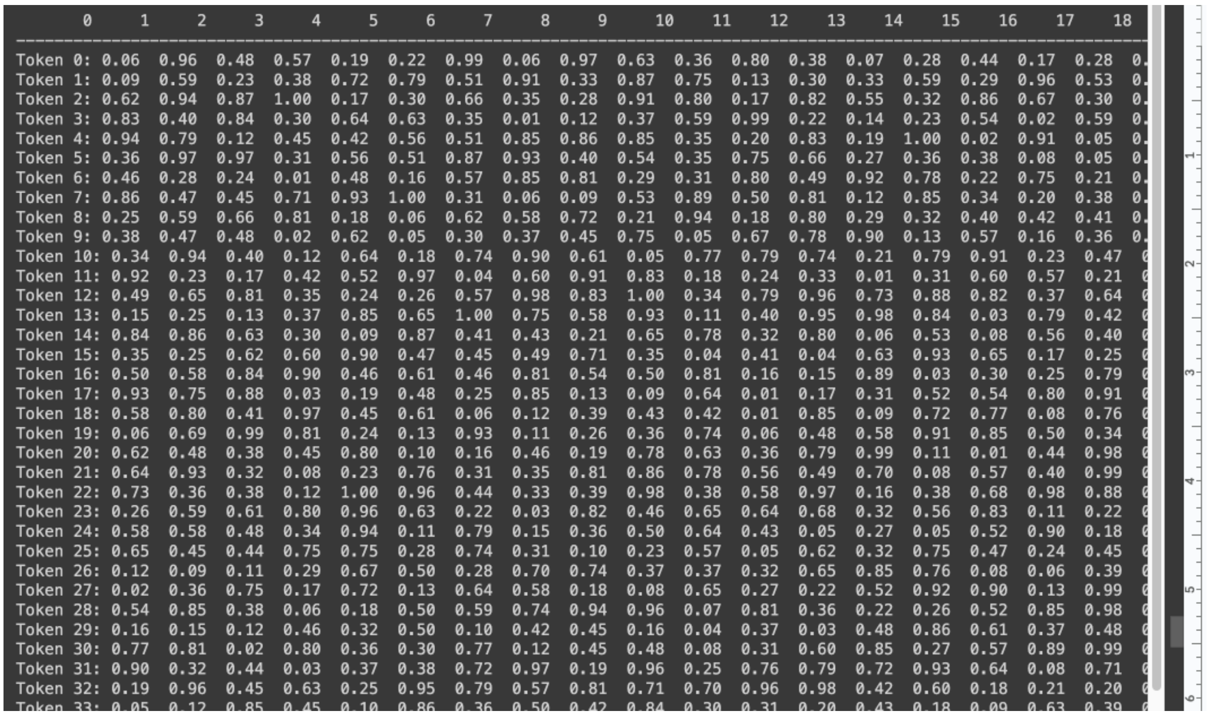
Printing Coreference Scores Matrix:

Finally, the code prints the coreference scores matrix, providing a visual representation of potential coreference relationships between tokens.Each cell in the matrix represents the coreference score between a pair of tokens, with higher scores indicating a stronger likelihood of coreference.



```python
import tensorflow as tf
from transformers import TFBertModel, BertTokenizer
import numpy as np


tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = TFBertModel.from_pretrained('bert-base-uncased')

text = df['summary'][25]


inputs = tokenizer(text, return_tensors="tf", max_length=128, truncation=True)


outputs = bert_model(**inputs)


embeddings = outputs.last_hidden_state


num_mentions = inputs['input_ids'].shape[1]
coreference_matrix = np.random.randint(2, size=(num_mentions, num_mentions))
print(coreference_matrix)
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.seq_relationship.we
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another arch:
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for pred:
[[1 1 0 ... 0 1 0]
 [1 0 0 ... 0 0 0]
 [1 0 1 ... 0 1 1]
 ...
 [1 1 1 ... 0 1 0]
 [1 0 0 ... 0 1 0]
 [0 1 0 ... 0 0 0]]
```



```
          0     1     2     3     4     5     6     7     8     9     10    11    12    13    14    15    16    17    18
--------------------------------------------------------------------------------------------------------------------------
Token 0: 0.06  0.96  0.48  0.57  0.19  0.22  0.99  0.06  0.97  0.63  0.36  0.80  0.38  0.07  0.28  0.44  0.17  0.28  0.
Token 1: 0.09  0.59  0.23  0.38  0.72  0.79  0.51  0.91  0.33  0.87  0.75  0.13  0.30  0.33  0.59  0.29  0.96  0.53  0.
Token 2: 0.62  0.94  0.87  1.00  0.17  0.30  0.66  0.35  0.28  0.91  0.80  0.17  0.82  0.55  0.32  0.86  0.67  0.30  0.
Token 3: 0.83  0.40  0.84  0.30  0.64  0.63  0.35  0.01  0.12  0.37  0.59  0.99  0.22  0.14  0.23  0.54  0.02  0.59  0.
Token 4: 0.94  0.79  0.12  0.45  0.42  0.56  0.51  0.85  0.86  0.85  0.35  0.20  0.83  0.19  1.00  0.02  0.91  0.05  0.
Token 5: 0.36  0.97  0.97  0.31  0.56  0.51  0.87  0.93  0.40  0.54  0.35  0.75  0.66  0.27  0.36  0.38  0.08  0.05  0.
Token 6: 0.46  0.28  0.24  0.01  0.48  0.16  0.57  0.85  0.81  0.29  0.31  0.80  0.49  0.92  0.78  0.22  0.75  0.21  0.
Token 7: 0.86  0.47  0.45  0.71  0.93  1.00  0.31  0.06  0.09  0.53  0.89  0.50  0.81  0.12  0.85  0.34  0.20  0.38  0.
Token 8: 0.25  0.59  0.66  0.81  0.18  0.06  0.62  0.58  0.72  0.21  0.94  0.18  0.80  0.29  0.32  0.40  0.42  0.41  0.
Token 9: 0.38  0.47  0.48  0.02  0.62  0.05  0.30  0.37  0.45  0.75  0.05  0.67  0.78  0.90  0.13  0.57  0.16  0.36  0.
Token 10: 0.34  0.94  0.40  0.12  0.64  0.18  0.74  0.90  0.61  0.05  0.77  0.79  0.74  0.21  0.79  0.91  0.23  0.47  0
Token 11: 0.92  0.23  0.17  0.42  0.52  0.97  0.04  0.60  0.91  0.83  0.18  0.24  0.33  0.01  0.31  0.60  0.57  0.21  0
Token 12: 0.49  0.65  0.81  0.35  0.24  0.26  0.57  0.98  0.83  1.00  0.34  0.79  0.96  0.73  0.88  0.82  0.37  0.64  0
Token 13: 0.15  0.25  0.13  0.37  0.85  0.65  1.00  0.75  0.58  0.93  0.11  0.40  0.95  0.98  0.84  0.03  0.79  0.42  0
Token 14: 0.84  0.86  0.63  0.30  0.09  0.87  0.41  0.43  0.21  0.65  0.78  0.32  0.80  0.06  0.53  0.08  0.56  0.40  0
Token 15: 0.35  0.25  0.62  0.60  0.90  0.47  0.45  0.49  0.71  0.35  0.04  0.41  0.04  0.63  0.93  0.65  0.17  0.25  0
Token 16: 0.50  0.58  0.84  0.90  0.46  0.61  0.46  0.81  0.54  0.50  0.81  0.16  0.15  0.89  0.03  0.30  0.25  0.79  0
Token 17: 0.93  0.75  0.88  0.03  0.19  0.48  0.25  0.85  0.13  0.09  0.64  0.01  0.17  0.31  0.52  0.54  0.80  0.91  0
Token 18: 0.58  0.80  0.41  0.97  0.45  0.61  0.06  0.12  0.39  0.43  0.42  0.01  0.85  0.09  0.72  0.77  0.08  0.76  0
Token 19: 0.06  0.69  0.99  0.81  0.24  0.13  0.93  0.11  0.26  0.36  0.74  0.06  0.48  0.58  0.91  0.85  0.50  0.34  0
Token 20: 0.62  0.48  0.38  0.45  0.80  0.10  0.16  0.46  0.19  0.78  0.63  0.36  0.79  0.99  0.11  0.01  0.44  0.98  0
Token 21: 0.64  0.93  0.32  0.08  0.23  0.76  0.31  0.35  0.81  0.86  0.78  0.56  0.49  0.70  0.08  0.57  0.40  0.99  0
Token 22: 0.73  0.36  0.38  0.12  1.00  0.96  0.44  0.33  0.39  0.98  0.38  0.58  0.97  0.16  0.38  0.68  0.98  0.88  0
Token 23: 0.26  0.59  0.61  0.80  0.96  0.63  0.22  0.03  0.82  0.46  0.65  0.64  0.68  0.32  0.56  0.83  0.11  0.22  0
Token 24: 0.58  0.58  0.48  0.34  0.94  0.11  0.79  0.15  0.36  0.50  0.64  0.43  0.05  0.27  0.05  0.52  0.90  0.18  0
Token 25: 0.65  0.45  0.44  0.75  0.75  0.28  0.74  0.31  0.10  0.23  0.57  0.05  0.62  0.32  0.75  0.47  0.24  0.45  0
Token 26: 0.12  0.09  0.11  0.29  0.67  0.50  0.28  0.70  0.74  0.37  0.37  0.32  0.65  0.85  0.76  0.08  0.06  0.39  0
Token 27: 0.02  0.36  0.75  0.17  0.72  0.13  0.64  0.58  0.18  0.08  0.65  0.27  0.22  0.52  0.92  0.90  0.13  0.99  0
Token 28: 0.54  0.85  0.38  0.06  0.18  0.50  0.59  0.74  0.94  0.96  0.07  0.81  0.36  0.22  0.26  0.52  0.85  0.98  0
Token 29: 0.16  0.15  0.12  0.46  0.32  0.50  0.10  0.42  0.45  0.16  0.04  0.37  0.03  0.48  0.86  0.61  0.37  0.48  0
Token 30: 0.77  0.81  0.02  0.80  0.36  0.30  0.77  0.12  0.45  0.48  0.08  0.31  0.60  0.85  0.27  0.57  0.89  0.99  0
Token 31: 0.90  0.32  0.44  0.03  0.37  0.38  0.72  0.97  0.19  0.96  0.25  0.76  0.79  0.72  0.93  0.64  0.08  0.71  0
Token 32: 0.19  0.96  0.45  0.63  0.25  0.95  0.79  0.57  0.81  0.71  0.70  0.96  0.98  0.42  0.60  0.18  0.21  0.20  0
Token 33: 0.05  0.12  0.85  0.45  0.10  0.86  0.36  0.50  0.42  0.84  0.30  0.31  0.20  0.43  0.18  0.09  0.63  0.39  0
```

5. Challanges Faced

We encountered challenges in implementing various techniques due to version compatibility issues and package downloading errors. This report highlights the hurdles faced and the attempted solutions in integrating coreference resolution into our NLP pipeline.

1. Stanford Parser Integration:

Initially, we aimed to utilize the Stanford Parser for coreference resolution due to its robust capabilities. However, implementation hurdles arose due to version discrepancies between the parser and other dependencies in our environment. Despite efforts to reconcile the versions, compatibility issues persisted, preventing successful integration.

2. CORENLP Package Download Error:

As an alternative, we explored integrating the CORENLP package, which offers comprehensive NLP functionalities, including coreference resolution. Regrettably, we encountered difficulties in downloading the package using the provided URL. Despite multiple attempts and troubleshooting, the download error persisted, impeding our progress.

3. BERTCoreferenceResolution Library:

To circumvent the challenges encountered with traditional coreference resolution approaches, we explored leveraging the BERTCoreferenceResolution library imported through the transformers package. This library utilizes BERT, a state-of-the-art language model, for coreference resolution tasks. However, upon attempting to import the library, we encountered an error, which hindered further exploration of this approach.

6.Conclusion

Incorporating POS tagging, parse tree generation through Stanza dependency parsing, and coreference resolution matrix enhances semantic analysis capabilities in NLP. These techniques provide valuable insights into the syntactic and semantic structure of textual data, enabling more accurate interpretation and understanding. By leveraging these NLP techniques, we can unlock the full potential of semantic analysis for various applications, including information extraction, sentiment analysis, and question answering.

7. Collab Notebook

https://colab.research.google.com/drive/1L0glXYqVMwTu4Mluelb5OxRAUYzx5iio?usp=sharing