CS 585 – Fall 2023 – Homework 5

Problem 1: Applying a Context free Grammar

Sentence: "Lucy plays with friends".
parse : (ROOT (S (NP (NNP Lucy)) (VP (VBZ plays) (PP (IN with) (NP (NNS friends))))))

Sentence: "This movie is careless and unfocused".
parse : (ROOT (S (NP (DT This) (NN movie)) (VP (VBZ is) (ADJP (JJ careless) (CC and) (JJ unfocused)))))

Sentence: "She buys a gift with gold".
1st parse :(S (NP (PRP She)) (VP (VBZ buys) (NP (DT a) (NN gift) (PP (IN with) (NP (NN gold))))))

2nd parse :(S (NP (PRP She)) (VP (VBZ buys) (NP (DT a) (NN gift)) (PP (IN with) (NP (NN gold)))))

Problem 2 : Constituency parsing

In [1]:
```
import stanza
import pandas as pd
```

In [2]:
```
nlp = stanza.Pipeline(lang='en', processors='tokenize,pos,constituency')
```

```
2023-11-20 22:21:00 INFO: Checking for updates to resources.json in case models have been updated.  Note: th
is behavior can be turned off with download_method=None or download_method=DownloadMethod.REUSE_RESOURCES
```

Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/resources_1.6.0.json: 367k/? [00:00<00:00, 6.61MB/s]

```
2023-11-20 22:21:01 INFO: Loading these models for language: en (English):
======================================
| Processor    | Package             |
--------------------------------------
| tokenize     | combined            |
| pos          | combined_charlm     |
| constituency | ptb3-revised_charlm |
======================================

2023-11-20 22:21:01 INFO: Using device: cpu
2023-11-20 22:21:01 INFO: Loading: tokenize
2023-11-20 22:21:03 INFO: Loading: pos
2023-11-20 22:21:03 INFO: Loading: constituency
2023-11-20 22:21:04 INFO: Done loading processors!
```

In [17]:
```
s = ['Lucy plays with friends','This movie is careless and unfocused','She buys a gift with gold']
print("By applying the Stanza constituency parser")
for i in s:
    print('\n->',i,":")
    z=nlp(i)
    for sentence in z.sentences:
        print(sentence.constituency)
```

```
By applying the Stanza constituency parser

-> Lucy plays with friends :
(ROOT (S (NP (NNP Lucy)) (VP (VBZ plays) (PP (IN with) (NP (NNS friends))))))

-> This movie is careless and unfocused :
(ROOT (S (NP (DT This) (NN movie)) (VP (VBZ is) (ADJP (JJ careless) (CC and) (JJ unfocused)))))

-> She buys a gift with gold :
(ROOT (S (NP (PRP She)) (VP (VBZ buys) (NP (DT a) (NN gift)) (PP (IN with) (NP (NN gold))))))
```

Problem 3 : Reading the data

In [4]:
```
climate_change_data = pd.read_csv('D:/Masters/NLP/Assignments/Assignment5/climate_change.csv', encoding='latir
gangs_data = pd.read_csv('D:/Masters/NLP/Assignments/Assignment5/Gangs.csv', encoding='latin1')
thatcher_data = pd.read_csv('D:/Masters/NLP/Assignments/Assignment5/Thatcher.csv', encoding='latin1')
# Now Filter the data( where there are empty entries in the Elementry in  3 files)
climate_change_filtered = climate_change_data[climate_change_data['Elementary'].notnull()]
gangs_filtered = gangs_data[gangs_data['Elementary'].notnull()]
thatcher_filtered = thatcher_data[thatcher_data['Elementary'].notnull()]
#Combining the 3 files
combined_datasets = pd.concat([climate_change_filtered, gangs_filtered, thatcher_filtered], ignore_index=True)
print(f"Number of rows in the combined dataset: {len(combined_datasets)}")
first_row = combined_datasets.iloc[0]
elementary_txt = first_row['Elementary']
advanced_txt = first_row['Advanced']
print(f"\nElementary Text:\n{elementary_txt}\n")
print(f"Advanced Text:\n{advanced_txt}")
```

```
Number of rows in the combined dataset: 35

Elementary Text:
Poorer countries will be most affected
by climate change in the next century.
Sea levels will rise, there will be stronger
cyclones, warmer days and nights, more
rainfall, and larger and longer heatwaves,
says a new report.

Advanced Text:
Low-income countries will remain on the front
line of human-induced climate change over the
next century, experiencing gradual sea-level
rises, stronger cyclones, warmer days and
nights, more unpredictable rainfall, and larger
and longer heatwaves, according to the most
thorough assessment of the issue yet.
```

we can see that these are 35 rows in the above filtered and combined dataset. And the Advanced text is little complex than the elementary text.

Problem 4 : Analyzing the data

In [5]:
```
import stanza
stanza.download('en')
nlp = stanza.Pipeline(lang='en', processors='tokenize,pos,lemma,depparse,constituency', use_gpu=True)
```

Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/resources_1.6.0.json: 367k/? [00:00<00:00, 13.6MB/s]

```
2023-11-20 22:21:08 INFO: Downloading default packages for language: en (English) ...
2023-11-20 22:21:10 INFO: File exists: C:\Users\seshu\stanza_resources\en\default.zip
2023-11-20 22:21:16 INFO: Finished downloading models and saved to C:\Users\seshu\stanza_resources.
2023-11-20 22:21:16 INFO: Checking for updates to resources.json in case models have been updated.  Note: th
is behavior can be turned off with download_method=None or download_method=DownloadMethod.REUSE_RESOURCES
```

Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/resources_1.6.0.json: 367k/? [00:00<00:00, 8.30MB/s]

```
2023-11-20 22:21:17 INFO: Loading these models for language: en (English):
======================================
| Processor    | Package             |
--------------------------------------
| tokenize     | combined            |
| pos          | combined_charlm     |
| lemma        | combined_nocharlm   |
| constituency | ptb3-revised_charlm |
| depparse     | combined_charlm     |
======================================

2023-11-20 22:21:17 WARNING: GPU requested, but is not available!
2023-11-20 22:21:17 INFO: Using device: cpu
2023-11-20 22:21:17 INFO: Loading: tokenize
2023-11-20 22:21:17 INFO: Loading: pos
2023-11-20 22:21:17 INFO: Loading: lemma
2023-11-20 22:21:17 INFO: Loading: constituency
2023-11-20 22:21:18 INFO: Loading: depparse
2023-11-20 22:21:18 INFO: Done loading processors!
```

In [6]:
```
elementary_txt_data = combined_datasets['Elementary'].tolist()
advanced_txt_data = combined_datasets['Advanced'].tolist()
```

In [7]:
```
def txt_parse(txt):
    parse_doc, i = [], 0
    while i < len(txt):
        doc = nlp(txt[i])
        parse_doc.append(doc)
        i += 1
    return parse_doc
```

In [8]:
```
def count_phrases_from_tree(tree, label):
    stack = [tree]
    count = 0
    while stack:
        curr_node = stack.pop()
        if curr_node is None:
            continue
        if curr_node.label == label:
            count += 1
        stack.extend(reversed(curr_node.children))
    return count
```

In [9]:
```
def con_parser_txts(txt):
    docs = txt_parse(txt)
    total_sents, total_pp, total_verb, doc_index = 0, 0, 0, 0
    while doc_index < len(docs):
        doc, sent_index = docs[doc_index], 0
        while sent_index < len(doc.sentences):
            sent = doc.sentences[sent_index]
            total_sents += 1
            total_pp += count_phrases_from_tree(sent.constituency, 'PP')
            total_verb += count_phrases_from_tree(sent.constituency, 'VP')
            sent_index += 1
        doc_index += 1
    txt_len = len(txt)
    avg_sents, avg_pps, avg_verbs = total_sents / txt_len, total_pp / txt_len, total_verb / txt_len
    return avg_sents, avg_pps, avg_verbs
```

In [10]:
```
elementary_summary = con_parser_txts(elementary_txt_data)
advanced_summary = con_parser_txts(advanced_txt_data)
print("Elementary texts summary:")
print("The average number of sentences in each text ",elementary_summary[0])
print("The average number of prepositional phrases in each text",elementary_summary[1])
print("Average number of verbs per each text ",elementary_summary[2])
print("\nAdvanced texts summary:")
print("The average number of sentences in each text ",advanced_summary[0])
print("The average number of prepositional phrases in each text",advanced_summary[1])
print("Average number of verbs per each text ",advanced_summary[2])
```

```
Elementary texts summary:
The average number of sentences in each text  3.2285714285714286
The average number of prepositional phrases in each text 4.857142857142857
Average number of verbs per each text  10.17142857142857

Advanced texts summary:
The average number of sentences in each text  3.1142857142857143
The average number of prepositional phrases in each text 6.914285714285715
Average number of verbs per each text  12.6
```

Problem 5 : Evaluating Complexity Level

The quantity of verbs in each text is the feature that I included in Problem 4.Verbs are words that describe events, conditions, or acts.This feature counts how many verbs are typically used in a given text.

The new attribute, According to my examination of the previous problem, there are 10.17 verb phrases on average in the Elementary column and 12.6 in the Advanced column. As a result, the average verb phrase for Advanced literature is somewhat greater. This suggests that advanced texts are written at a higher level because they may use verbs in sentences in a more sophisticated and diversified way.

Problem 6 :Additional Methods

These methods leverage machine learning and NLP techniques to classify texts based on their reading complexity levels. CRFs capture sequential dependencies, Bi-Directional LSTMs incorporate contextual information, and K-Means clustering provides an unsupervised approach for classification. The choice of method depends on factors such as dataset size, interpretability, and computational resources. These methods can be applied to build a tool for classifying texts, providing tailored material for students based on their reading proficiency.

K-Means Clustering: For an unsupervised approach, K-Means Clustering can be employed. Text data is transformed into tokens, and word embedding models or TF-IDF representations are applied. The number of clusters, representing different complexity levels, is chosen. Using K-Means clustering, the tool classifies texts into distinct clusters based on their similarities, providing an unsupervised means of discerning reading complexity levels. These methods collectively offer diverse strategies for developing a tool that enhances the personalized matching of texts with students' reading proficiency levels.

Conditional Random Fields(CRF): In building a tool for classifying texts by reading complexity, one effective method is Conditional Random Fields (CRFs). The process begins with annotating texts based on complexity levels, creating a labeled training dataset. Linguistic features, including part-of-speech tags, named entity recognition, and word embeddings, are extracted using NLP tools. The CRF model is then trained to capture dependencies between these features and complexity labels, effectively leveraging sequential dependencies inherent in the textual data.

Bi-Directional Long Short-Term Memory networks (Bi-LSTM): Another powerful approach involves utilizing Bi-Directional Long Short-Term Memory networks (Bi-LSTM). After annotating texts, the data is tokenized and pre-trained word embeddings like Word2Vec or BERT are applied. A custom Bi-LSTM model is constructed to capture contextual information from both forward and backward directions, enhancing the model's ability to understand the nuances of text. The model is trained on the labeled dataset, learning to classify texts based on their reading complexity.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: