Open in Colab

Assignment 3

EAI6010 - Applications of Artificial Intelligence

Mohammad Hossein Movahedi

Lecturer: Prof. Vladimir Shapiro

Fall 2023

Table of Contents

1. Introduction
2. Data Cleaning
3. Data Analysis
4. Results and Discussion
5. Conclusion
6. References

Introduction

This assignment delves into the fascinating world of Natural Language Processing (NLP), a field at the intersection of computer science, artificial intelligence, and linguistics. NLP enables computers to understand, interpret, and respond to human language in a meaningful way. Through this assignment, you will gain hands-on experience in using NLP techniques for text analysis, employing tools such as the Gutenberg corpus and the Inaugural corpus within the NLTK package.

Data Cleaning

For this assignment, I will be using the Gutenberg corpus, a collection of texts from the 18th century, to explore the relationship between language and the world around us. It doen't need a data cleaning step as the corpus is already clean.

Data Analysis

I'm embarking on an NLP assignment focused on the text courpus. The first crucial step involves data cleaning, ensuring the text is primed for precise analysis. This process will include standardizing formats, removing irrelevant elements, and preparing the data for in-depth exploration of its linguistic features.

Below is the code for Q1

First, let's import the necessary libraries.

In [78]:
```
#Import Necessary Libraries
```

```
import nltk
from nltk.corpus import gutenberg
from collections import Counter
import pandas as pd
```

Then let's load the text data. by downloading the Gutenberg corpus tool from the NLTK website.

In [79]:
```
nltk.download('gutenberg')
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
```

Out[79]: True

After that, let's load the text data from the corpus.

In [4]:
```
#Use the texts in the corpus
corpus = nltk.corpus.gutenberg.words()
corpus
```

Out[4]: ['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', ...]

Next, let's count the frequency of each word in the corpus. I will use the FreqDist function from the nltk library to count the frequency of each word in the corpus. and then calculate the relative frequency of each modal.

In [46]:
```
#Create a table displaying relative frequency
FeqDist = pd.Series(nltk.FreqDist(corpus)).sort_values(ascending=False)

#filter the table on  which “modals” (can, could, may, might, will, would, and should) appear in each of the tex
modals = ['can', 'could', 'may', 'might', 'will', 'would', 'should']
ModalsDist = FeqDist[FeqDist.index.isin(modals)]
ModalsDist = pd.DataFrame(ModalsDist, columns=['frequency'])
ModalsDist
```

Out[46]:

	frequency
will	7130
would	3932
could	3528
should	2496
may	2435
can	2163
might	1938

Now let's find the two modals with the largest span of relative frequencies.

In [65]:
```
#Determine two modals with the largest span of relative frequencies'
Modals = ['can', 'could', 'may', 'might', 'will', 'would', 'should']
#"Relative frequency" = the number of occurrences of a given modal divided by the total number of modals in the
Dic = {}
for modal in Modals:
  Dic[modal] = ModalsDist.loc[modal, 'frequency'] / ModalsDist['frequency'].sum()
#print(Dic)
#Determine two modals with the largest span of relative frequencies
Dic = sorted(Dic.items(), key=lambda kv: kv[1], reverse=True)
#print(Dic)
# Print the two modal with the largest span of relative frequencies
print(Dic[0])
print(Dic[-1])
```

```
('will', 0.30183727034120733)
('might', 0.08204216408432817)
```

Here is the Code for Part E

In [106…

```
modal_frequencies = {modal: Counter() for modal in modals}
#print(modal_frequencies)

# Analyze each text in the Gutenberg corpus
for fileid in gutenberg.fileids():
    words = [word.lower() for word in gutenberg.words(fileid)]
    for modal in modals:
        modal_frequencies[modal][fileid] = words.count(modal)

#print(modal_frequencies)

# Create a DataFrame to display relative frequencies
df = pd.DataFrame(modal_frequencies)

# Calculate total modals for each text
df['total_modals'] = df.sum(axis=1)

# Calculate relative frequencies
for modal in modals:
    df[modal] = df[modal] / df['total_modals']

# Find two modals with the largest span of relative frequencies
spans = {modal: df[modal].max() - df[modal].min() for modal in modals}
largest_span_modals = sorted(spans, key=spans.get, reverse=True)[:2]

# Select the texts for the most frequently used modal among the two
most_used_modal = largest_span_modals[0]
most_text = df[most_used_modal].idxmax()
least_text = df[most_used_modal].idxmin()

# Print results
print(f"Two modals with the largest span: {largest_span_modals}")
print(f"Text with the most usage of '{most_used_modal}': {most_text}")
print(f"Text with the least usage of '{most_used_modal}': {least_text}")
```

```
{'can': Counter(), 'could': Counter(), 'may': Counter(), 'might': Counter(), 'will': Counter(), 'would': Counter
(), 'should': Counter()}
Two modals with the largest span: ['will', 'can']
Text with the most usage of 'will': bible-kjv.txt
Text with the least usage of 'will': blake-poems.txt
```

It's important to note, In the King James Bible, "will" is used to show God's power and plans, fitting its serious and old-fashioned style. In William Blake's poems, "will" shows personal wishes and feelings, matching his dreamy and emotional writing. So, "will" in the Bible is about God's commands, but in Blake's poetry, it's about personal hopes and challenging norms.

Below is the Code for Q2

Nunc in velit neque. Cras dui nunc, maximus non ornare a, tempor quis lorem. Morbi feugiat sodales magna quis lacinia. Pellentesque porttitor ex id nisi pretium lacinia. Proin nec sapien volutpat, porttitor purus nec, elementum enim.

Q1 Pellentesque at vestibulum augue, non gravida tellus?

Pellentesque sed metus risus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Donec cursus eleifend neque, id interdum metus eleifend in.

A1 Pellentesque at vestibulum augue, non gravida tellus.

In [109…

```
# nltk is a tool that helps us play with words in books or speeches
from nltk.corpus import inaugural
from nltk.probability import FreqDist
from nltk.corpus import wordnet as wn
nltk.download('wordnet')
# A:  download the inaugural speeches
nltk.download('inaugural')

# B: We're going to look at President Kennedy's speech, which is like picking out his book from a library
kennedy_speech = inaugural.words('1961-Kennedy.txt')

# C: Now we want to find the 10 longest and most popular words in his speech
long_words = [word.lower() for word in kennedy_speech if len(word) > 7]  # Only pick words longer than 7 letters
fdist = FreqDist(long_words)  # Count how many times each word appears
```

```python
most_common_long = fdist.most_common(10)  # Pick the 10 words that appear the most

# D: We're going to use WordNet, which is like a thesaurus to find synonyms
max_syn_count = 0  # This is placeholder for the count of the word with the most synonyms
word_with_max_syn = ''  # This is palaceholder for the word with the largest number of synonyms

# E: We're going to make a list of all the words that mean the same thing for our top 10 words
synonyms = {}  # This is like our notebook to write down all the synonyms
for word, _ in most_common_long:
    synsets = wn.synsets(word)  # Find all the groups of synonyms for our word
    all_synonyms = set(lemma.name() for synset in synsets for lemma in synset.lemmas())  # Write down all the sy
    synonyms[word] = all_synonyms  # Put them in our list
    if len(all_synonyms) > max_syn_count:  # If this word has more synonyms than the current champion...
        max_syn_count = len(all_synonyms)  # ...update the synonym scoreboard...
        word_with_max_syn = word  # ...and write down the new winner

# Print the word with the largest number of synonyms and the count
print(f"The word '{word_with_max_syn}' has the most synonyms: {max_syn_count}")

# F: We also want to find words that are more specific types of our word, like if 'vehicle' was our word, 'car'
hyponyms = {}  # This is another notebook for the specific types of words
for word, _ in most_common_long:
    synsets = wn.synsets(word)  # Again we find all the groups of synonyms
    all_hyponyms = set(lemma.name() for synset in synsets for hyponym in synset.hyponyms() for lemma in hyponym.
    hyponyms[word] = all_hyponyms  # And put them in our notebook

# G: Now let's find out which word has the most specific types
max_hypo_count = 0  # Our scoreboard for the most specific types
word_with_max_hypo = ''  # The name of the winner

for word, hypo_set in hyponyms.items():
    if len(hypo_set) > max_hypo_count:  # If this word has more specific types than the current champion...
        max_hypo_count = len(hypo_set)  # ...update the scoreboard...
        word_with_max_hypo = word  # ...and write down the new winner

# Print the word with the largest number of hyponyms and the count
print(f"The word '{word_with_max_hypo}' has the most hyponyms: {max_hypo_count}")

# H: Reflect on the results - This is where you think about what you found, like looking back at your adventure
print("Synonyms:")
for word, syn_set in synonyms.items():
    print(f"{word}: {syn_set}")

print("\nHyponyms:")
for word, hypo_set in hyponyms.items():
    print(f"{word}: {hypo_set}")
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
The word 'supporting' has the most synonyms: 37
The word 'americans' has the most hyponyms: 109
Synonyms:
citizens: {'citizen'}
president: {'chair', 'President', 'United_States_President', 'president', 'Chief_Executive', 'chairwoman', 'chairp
erson', 'President_of_the_United_States', 'prexy', 'chairman'}
americans: {'American_English', 'American_language', 'American'}
generation: {'contemporaries', 'generation', 'coevals', 'genesis', 'propagation', 'multiplication'}
forebears: {'forebear', 'forbear'}
revolution: {'rotation', 'revolution', 'gyration'}
committed: {'trust', 'institutionalize', 'attached', 'entrust', 'put', 'charge', 'confide', 'place', 'give', 'pul
l', 'committed', 'practice', 'perpetrate', 'invest', 'institutionalise', 'dedicate', 'send', 'devote', 'intrust',
'consecrate', 'commit'}
powerful: {'muscular', 'right', 'powerful', 'mighty', 'hefty', 'brawny', 'herculean', 'knock-down', 'potent', 'mig
htily', 'sinewy'}
supporting: {'subscribe', 'patronize', 'plump_for', 'patronage', 'affirm', 'endorse', 'hold', 'corroborate', 'keep
_going', 'bear', 'bear_out', 'digest', 'indorse', 'endure', 'confirm', 'sustain', 'hold_up', 'patronise', 'defen
d', 'stick_out', 'plunk_for', 'stand', 'back_up', 'put_up', 'load-bearing', 'tolerate', 'underpin', 'brook', 'enco
uraging', 'back', 'stomach', 'suffer', 'support', 'substantiate', 'fend_for', 'abide', 'supporting'}
themselves: set()

Hyponyms:
citizens: {'freewoman', 'repatriate', 'voter', 'active_citizen', 'civilian', 'thane', 'freeman', 'private_citize
n', 'elector'}
president: {'ex-president', 'vice_chairman', 'Kalon_Tripa'}
americans: {'Keystone_Stater', 'Badger', 'Alaskan', 'Delawarean', 'Sooner', 'Mainer', 'Carolinian', 'Missourian',
'Appalachian', 'AAVE', 'West_Indian', 'Latino', 'Tory', 'New_Jerseyite', 'New_Jerseyan', 'South_Dakotan', 'Bostoni
```

```
an', 'Alabaman', 'Oklahoman', 'Bluegrass_Stater', 'Hawaiian', 'West_Virginian', 'Michigander', 'Kansan', 'Utahan',
'Rhode_Islander', 'Wisconsinite', 'Louisianan', 'North_Dakotan', 'Northerner', 'Wolverine', 'Mississippian', 'Neva
dan', 'Franco-American', 'Hispanic', 'Afro-American', 'Iowan', 'Arkansan', 'Californian', 'New_Yorker', 'Hoosier',
'Beaver', 'Volunteer', 'Illinoisan', 'Creole', 'Virginian', 'Nebraskan', 'African-American', 'Louisianian', 'Arizo
nian', 'Tennessean', 'North_Carolinian', 'Arizonan', 'Georgian', 'Montanan', 'Yank', 'South_American', 'New_Hampsh
irite', 'German_American', 'Yankee-Doodle', 'Ohioan', 'South_Carolinian', 'Southerner', 'Yankee', 'Black_Vernacula
r', 'Marylander', 'Ebonics', 'Arkansawyer', 'Black_English_Vernacular', 'Puerto_Rican', 'Asian_American', 'Garden_
Stater', 'Alabamian', 'New_Englander', 'Latin_American', 'Kentuckian', 'Mesoamerican', 'North_American', 'Tarhee
l', 'Hispanic_American', 'Coloradan', 'Pennsylvanian', 'Connecticuter', 'African_American_English', 'African_Ameri
can_Vernacular_English', 'Gopher', 'Granite_Stater', 'Oregonian', 'Delawarian', 'Floridian', 'Spanish_American',
'Bay_Stater', 'Buckeye', 'Vermonter', 'Down_Easter', 'Washingtonian', 'Idahoan', 'Wyomingite', 'Minnesotan', 'Angl
o-American', 'New_Mexican', 'Texan', 'Indianan', 'Black_English', 'African_American', 'Cornhusker', 'Black_America
n', 'Black_Vernacular_English', 'Nisei'}
generation: {'biogenesis', 'baby-boom_generation', 'gen_X', 'posterity', 'baby_boom', 'youth_culture', 'peer_grou
p', 'generation_X', 'biogeny'}
forebears: {'grandparent', 'great_grandparent'}
revolution: {'spin', 'levorotation', 'orbital_motion', 'axial_rotation', 'axial_motion', 'roll', 'counterclockwise
_rotation', 'green_revolution', 'dextrorotation', 'orbital_rotation', 'Cultural_Revolution', 'clockwise_rotation',
'counterrevolution', 'Great_Proletarian_Cultural_Revolution'}
committed: {'obligate', 'shelter', 'rededicate', 'tie_up', 'consign', 'charge', 'roll_over', 'speculate', 'job',
'fund', 'hospitalize', 'apply', 'hospitalise', 'vow', 'buy_into', 'make', 'consecrate', 'recommit', 'commend'}
powerful: set()
supporting: {'patronize', 'pole', 'sponsor', 'stick_up', 'stand_up', 'propping_up', 'bear_up', 'further', 'carry',
'patronise', 'prop', 'rationalize', 'underpin', 'bring_home_the_bacon', 'hold_still_for', 'block', 'boost', 'under
gird', 'brace', 'excuse', 'truss', 'assist', 'shore_up', 'indorse', 'sit_out', 'justify', 'suspension', 'prop_up',
'see_through', 'fund', 'hanging', 'bracket', 'back_up', 'shop', 'back', 'chock', 'apologize', 'take_a_joke', 'swal
low', 'shore', 'champion', 'second', 'uphold', 'apologise', 'take_lying_down', 'shew', 'vouch', 'scaffold', 'dangl
ing', 'provide', 'promote', 'defend', 'encourage', 'frequent', 'subsidise', 'buoy', 'subsidize', 'pay', 'show', 'a
dvance', 'help', 'live_with', 'shoring_up', 'endorse', 'shoring', 'warrant', 'buy_at', 'rationalise', 'verify', 's
tand_for', 'validate', 'buoy_up', 'prove', 'accept', 'document', 'aid', 'guarantee', 'establish', 'shop_at', 'demo
nstrate'}
themselves: set()
```

Results and Discussion

In analyzing President Kennedy's inaugural speech through NLTK's corpus and WordNet, we find that the word 'supporting' stands out with the most synonyms, suggesting a rich variety of ways to convey the concept of aid or backing within the English language. This diversity reflects the speech's emphasis on the collective effort and mutual assistance, resonating with the theme of national unity and cooperation.

On the other hand, 'americans' has the most hyponyms, indicating a broad spectrum of identities and groups that constitute the American populace. This underlines the speech's inclusive nature, acknowledging the diverse tapestry of people that form the United States.

Conclusion

In this NLP assignment, I explored how language reflects themes and styles in different texts. Using the Gutenberg corpus, I found that the modal verb "will" varies significantly between texts, illustrating God's command in the King James Bible and personal desires in William Blake's poetry. This highlights how the same word can carry different meanings depending on the context. In President Kennedy's inaugural speech analysis, the word 'supporting' emerged with the most synonyms, emphasizing the speech's focus on collective effort and aid. The word 'americans' showed the most hyponyms, reflecting the diversity of the American population, aligning with the speech's inclusive nature. These analyses using NLP tools like NLTK and WordNet demonstrate the complexity and richness of language, revealing how word choice in literature and speeches can mirror societal values and historical contexts. This assignment underlines the challenges and nuances in NLP, showcasing its potential to unravel deeper linguistic and thematic insights from texts

References

James, K. (2023). OFFICIAL KING JAMES BIBLE ONLINE. [online] Kingjamesbibleonline.org. Available at: https://www.kingjamesbibleonline.org/ [Accessed 19 Nov. 2023].

Blakearchive.org. (2023). The William Blake Archive. [online] Available at: https://www.blakearchive.org/ [Accessed 19 Nov. 2023].