COIMBATORE INSTITUTE OF TECHNOLOGY

M.SC DATA SCIENCE

Natural Language Processing Laboratory

(21MDSEL2)

BY ,

HARINNE R KUMAR (71762232015)

PRITHIKA J (71762232039)

S.VEDHAPRIYAA (71762232059)

Ex: 1 **Access Text Corpora and Lexical Resources**

2.7.24

Problem Statement:

To access text corpora and lexical resources using

i) Conditional frequency distribution

ii)Word Net

Procedure:

Sure! Here's a step-by-step procedure based on a typical Jupyter notebook for NLP tasks, as I infer from the context of the file:

Step 1: Load Libraries

- Definition: The first step involves loading essential Python libraries such as nltk , numpy , pandas , or sklearn for data manipulation, text processing, and machine learning.

Step 2: Load Dataset

- Definition: Import the dataset you will use for your NLP task. This could be text data in formats like .csv , .txt , or directly from NLTK's corpora.

Step 3: Data Preprocessing

- Definition: Clean and preprocess the text data. This step often includes tasks like:

- Tokenization (breaking the text into words or sentences).
- Lowercasing the text.
- Removing stopwords (common words like "the", "is").
- Stemming or Lemmatization (reducing words to their root forms).

Step 4: Text Tokenization

- Definition: Tokenization is the process of converting a text into individual words or sentences. NLTK provides various tokenizers like word_tokenize() and sent_tokenize() for this purpose.

Step 5: POS Tagging (Part-of-Speech Tagging)

- Definition: Assigning parts of speech (noun, verb, adjective, etc.) to each tokenized word using tools like nltk.pos_tag() .

Step 6: Chunking or Shallow Parsing

- Definition: The process of grouping words into meaningful chunks like noun phrases or verb phrases, usually based on POS tags.

- Mathematical Formula: Uses regular expressions to define chunking patterns.

Step 7: Named Entity Recognition (NER)

- Definition: Identifying and classifying proper names, such as person names, organizations, and locations from the text using NER models.

Step 8: Vectorization (Text to Numbers)

- Definition: Convert the text into numerical format using methods like Bag of Words (BoW) or TF-IDF (Term Frequency-Inverse Document Frequency).

- Mathematical Formula for TF-IDF:

TF-IDF(t,d)=TF(t,d)×log|{d∈D:t∈d}|N

Where:

- TF(t,d) is the term frequency of term t in document d .
- N is the total number of documents.
- |\{d \in D: t \in d\}| is the number of documents containing term t .

Step 9: Text Classification

- Definition: Apply machine learning algorithms like Naive Bayes, SVM, or neural networks to classify text into categories. These models learn from the vectorized text data.

- No specific mathematical formula unless a model is used.

Step 10: Model Evaluation

- Definition: Evaluate the performance of the NLP model using metrics such as accuracy, precision, recall, and F1 score.

- These metrics are based on the comparison of predicted and actual labels in classification tasks.

Step 11: Visualize Results

- Definition: Use plots to visualize model performance, such as confusion matrices or graphs showing the importance of features.

Step 12: Save and Export Results

- Definition: Save your trained model or processed results for future use, possibly by exporting them to a file or displaying results in an organized format.

Code:

```
import nltk
nltk.corpus.gutenberg.fileids()
from nltk.corpus import webtext
for fileid in webtext.fileids():
  print(fileid, webtext.raw(fileid)[:33], '...')
from nltk.corpus import nps_chat
chatroom = nps_chat.posts('10-19-20s_706posts.xml')
chatroom[12]
from nltk.corpus import reuters
```

```
print(reuters.fileids())
a=reuters.words('test/14826')
print(a[:44])
from nltk.corpus import brown
print(brown.fileids())
b=brown.words('ca01')
print(b[:20])
from nltk.corpus import inaugural
print(inaugural.fileids())
c=inaugural.words('1789-Washington.txt')
print(c[:2])
emma=nltk.corpus.gutenberg.words('austen-emma.txt')
len(emma)
shakespeare=nltk.corpus.gutenberg.words('shakespeare-macbeth.txt')
len(shakespeare)
emma=nltk.Text(nltk.corpus.gutenberg.words('austen-emma.txt'))
emma.concordance("surprize")
from nltk.corpus import gutenberg
gutenberg.fileids()
emma = gutenberg.words('austen-emma.txt')
for fileid in gutenberg.fileids():
  num_chars = len(gutenberg.raw(fileid))
  num_words = len(gutenberg.words(fileid))
  num_sents = len(gutenberg.sents(fileid))
  num_vocab = len(set(w.lower() for w in gutenberg.words(fileid)))
  print(round(num_chars/num_words), round(num_words/num_sents),
round(num_words/num_vocab), fileid)
blake=gutenberg.raw('blake-poems.txt')
```

```
blake
len(gutenberg.raw('blake-poems.txt'))
blake_sentences[0]
longest_len=max(len(s) for s in blake_sentences)
[s for s in blake_sentences if len(s) == longest_len]
from nltk.probability import FreqDist

text = gutenberg.words('austen-emma.txt')
fdist = FreqDist(text)
print(fdist.most_common(10))
count=len(fdist)
count
from nltk.corpus import brown
brown.categories()
brown.words(categories='news')
brown.words(fileids=['cg22'])
brown.sents(categories=['news', 'editorial', 'reviews'])
from nltk.corpus import brown
cfd = nltk.ConditionalFreqDist((genre, word) for genre in brown.categories()
for word in brown.words(categories=genre))
genre_word = [(genre, word) for genre in ['news', 'romance'] for word in
brown.words(categories=genre)]
len(genre_word)
cfd = nltk.ConditionalFreqDist(genre_word)
cfd
cfd.conditions()
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']
```

```
cfd = nltk.ConditionalFreqDist((genre, word) for genre in ['news', 'romance'] for word in brown.words(categories=genre)  if word in days )
print("Tabulation:")
cfd.tabulate(samples=days)
print("\nPlot:")
cfd.plot(samples=days, title="Days of the Week in News and Romance Genres")
from nltk.corpus import wordnet as wn
synonyms=wn.synsets('like')
print(synonyms)
for syn in synonyms:
    print(syn.definition())
    print(syn.examples())
from nltk.tokenize import word_tokenize, sent_tokenize

text = "Hi all.This is fun."
print(word_tokenize(text))
print(sent_tokenize(text))
macbeth_sentences = gutenberg.sents('shakespeare-macbeth.txt')
print(macbeth_sentences)
wn.lemmas('like')
motorcar=wn.synset('car.n.01')
types=motorcar.hyponyms()
types[0]
sorted(lemma.name() for synset in types for lemma in synset.lemmas())
```

Output:

firefox.txt Cookie Manager: "Don't allow site ... grail.txt SCENE 1: [wind] [clop clop clop] ... overheard.txt White guy: So, do you have any pl ... pirates.txt PIRATES OF THE CARRIBEAN: DEAD MA ... singles.txt 25 SEXY MALE, seeks attrac older ... wine.txt Lovely delicate, fragrant Rhone w ...

['JOIN']

['ca01', 'ca02', 'ca03', 'ca04', 'ca05', 'ca06', 'ca07', 'ca08', 'ca09', 'ca10', 'ca11', 'ca12', 'ca13', 'ca14', 'ca15', 'ca16', 'ca17', 'ca18', 'ca19', 'ca20', 'ca21', 'ca22', 'ca23', 'ca24', 'ca25', 'ca26', 'ca27', 'ca28', 'ca29', 'ca30', 'ca31', 'ca32', 'ca33', 'ca34', 'ca35', 'ca36', 'ca37', 'ca38', 'ca39', 'ca40', 'ca41', 'ca42', 'ca43', 'ca44', 'cb01', 'cb02', 'cb03', 'cb04', 'cb05', 'cb06', 'cb07', 'cb08', 'cb09', 'cb10', 'cb11', 'cb12', 'cb13', 'cb14', 'cb15', 'cb16', 'cb17', 'cb18', 'cb19', 'cb20', 'cb21', 'cb22', 'cb23', 'cb24', 'cb25', 'cb26', 'cb27', 'cc01', 'cc02', 'cc03', 'cc04', 'cc05', 'cc06', 'cc07', 'cc08', 'cc09', 'cc10', 'cc11', 'cc12', 'cc13', 'cc14', 'cc15', 'cc16', 'cc17', 'cd01', 'cd02', ' 'ck10', 'ck11', 'ck12', 'ck13', 'ck14', 'ck15', 'ck16', 'ck17', 'ck18', 'ck19', 'ck20', 'ck21', 'ck22', 'ck23', 'ck24', 'ck25', 'ck26', 'ck27', 'ck28', 'ck29', 'cl01', 'cl02', 'cl03', 'cl04', 'cl05', 'cl06', 'cl07', 'cl08', 'cl09', 'cl10', 'cl11', 'cl12', 'cl13', 'cl14', 'cl15', 'cl16', 'cl17', 'cl18', 'cl19', 'cl20', 'cl21', 'cl22', 'cl23', 'cl24', 'cm01', 'cm02', 'cm03', 'cm04', 'cm05', 'cm06', 'cn01', 'cn02', 'cn03', 'cn04', 'cn05', 'cn06', 'cn07', 'cn08', 'cn09', 'cn10', 'cn11', 'cn12', 'cn13', 'cn14', 'cn15', 'cn16', 'cn17', 'cn18', 'cn19', 'cn20', 'cn21', 'cn22', 'cn23', 'cn24', 'cn25', 'cn26', 'cn27', 'cn28', 'cn29', 'cp01', 'cp02', 'cp03', 'cp04', 'cp05', 'cp06', 'cp07', 'cp08', 'cp09', 'cp10', 'cp11', 'cp12', 'cp13', 'cp14', 'cp15', 'cp16', 'cp17', 'cp18', 'cp19', 'cp20', 'cp21', 'cp22', 'cp23', 'cp24', 'cp25', 'cp26', 'cp27', 'cp28', 'cp29', 'cr01', 'cr02', 'cr03', 'cr04', 'cr05', 'cr06', 'cr07', 'cr08', 'cr09'] ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', "Atlanta's", 'recent', 'primary', 'election', 'produced', ' ', 'no', 'evidence', '"""', 'that']

['1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt', '1801-Jefferson.txt', '1805-Jefferson.txt', '1809-Madison.txt', '1813-Madison.txt', '1817-Monroe.txt', '1821-Monroe.txt', '1825-Adams.txt', '1829-Jackson.txt', '1833-Jackson.txt', '1837-VanBuren.txt', '1841-Harrison.txt', '1845-Polk.txt', '1849-Taylor.txt', '1853-Pierce.txt', '1857-Buchanan.txt', '1861-Lincoln.txt', '1865-Lincoln.txt', '1869-Grant.txt', '1873-Grant.txt', '1877-Hayes.txt', '1881-Garfield.txt', '1885-Cleveland.txt', '1889-Harrison.txt', '1893-Cleveland.txt', '1897-McKinley.txt', '1901-McKinley.txt', '1905-Roosevelt.txt', '1909-Taft.txt', '1913-Wilson.txt', '1917-Wilson.txt', '1921-Harding.txt', '1925-Coolidge.txt',

'1929-Hoover.txt', '1933-Roosevelt.txt', '1937-Roosevelt.txt', '1941-Roosevelt.txt', '1945-Roosevelt.txt', '1949-Truman.txt', '1953-Eisenhower.txt', '1957-Eisenhower.txt', '1961-Kennedy.txt', '1965-Johnson.txt', '1969-Nixon.txt', '1973-Nixon.txt', '1977-Carter.txt', '1981-Reagan.txt', '1985-Reagan.txt', '1989-Bush.txt', '1993-Clinton.txt', '1997-Clinton.txt', '2001-Bush.txt', '2005-Bush.txt', '2009-Obama.txt', '2013-Obama.txt', '2017-Trump.txt', '2021-Biden.txt']
['Fellow', '-']

192427

23140

D isplaying 25 of 37 matches: er father , was sometimes taken by surprize at his being still able to pity hem do the other any good ." " You surprize me ! Emma must do Harriet good : a Knightley actually looked red with surprize and displeasure , as he stood up , r . Elton , and found to his great surprize , that Mr . Elton was actually on d aid ." Emma saw Mrs . Weston ' s surprize , and felt that it must be great , father was quite taken up with the surprize of so sudden a journey , and his f y , in all the favouring warmth of surprize and conjecture . She was , moreove he appeared , to have her share of surprize , introduction , and pleasure . Th ir plans ; and it was an agreeable surprize to her , therefore , to perceive t talking aunt had taken me quite by surprize , it must have been the death of m f all the dialogue which ensued of surprize , and inquiry , and congratulation the present . They might chuse to surprize her ." Mrs . Cole had many to agre the mode of it , the mystery , the surprize , is more like a young woman ' s s to her song took her agreeably by surprize -- a second , slightly but correct " " Oh ! no -- there is nothing to surprize one at all .-- A pretty fortune ; t to be considered . Emma ' s only surprize was that Jane Fairfax should accep of your admiration may take you by surprize some day or other ." Mr . Knightle ation for her will ever take me by surprize .-- I never had a thought of her i expected by the best judges , for surprize -- but there was great joy . Mr . sound of at first , without great surprize . " So unreasonably early !" she w d Frank Churchill , with a look of surprize and displeasure .-- " That is easy ; and Emma could imagine with what surprize and mortification she must be retu tled that Jane should go . Quite a surprize to me ! I had not the least idea ! . It is impossible to express our surprize . He came to speak to his father o g engaged !" Emma even jumped with surprize ;-- and , horror - struck , exclai

5 25 26 austen-emma.txt

5 26 17 austen-persuasion.txt

5 28 22 austen-sense.txt

4 34 79 bible-kjv.txt

5 19 5 blake-poems.txt

4 19 14 bryant-stories.txt

4 18 12 burgess-busterbrown.txt

4 20 13 carroll-alice.txt

5 20 12 chesterton-ball.txt

5 23 11 chesterton-brown.txt

5 18 11 chesterton-thursday.txt

4 21 25 edgeworth-parents.txt

5 26 15 melville-moby_dick.txt

5 52 11 milton-paradise.txt

4 12 9 shakespeare-caesar.txt

4 12 8 shakespeare-hamlet.txt

4 12 7 shakespeare-macbeth.txt

5 36 12 whitman-leaves.txt

[['[', 'Poems', 'by', 'William', 'Blake', '1789', ']'], ['SONGS', 'OF', 'INNOCENCE', 'AND', 'OF', 'EXPERIENCE', 'and', 'THE', 'BOOK', 'of', 'THEL'], ...]

['[', 'Poems', 'by', 'William', 'Blake', '1789', ']']

[['I', 'fear', 'that', 'I', 'am', 'not', 'like', 'thee', ':', 'For', 'I', 'walk', 'through',])

[(',', 11454), ('.', 6928), ('to', 5183), ('the', 4844), ('and', 4672), ('of', 4279), ('I', 3178), ('a', 3004), ('was', 2385), ('her', 2381)]

7811

['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor,'learned','lore', 'mystery', 'news', 'religion', 'reviews', 'romance', 'science_fiction']

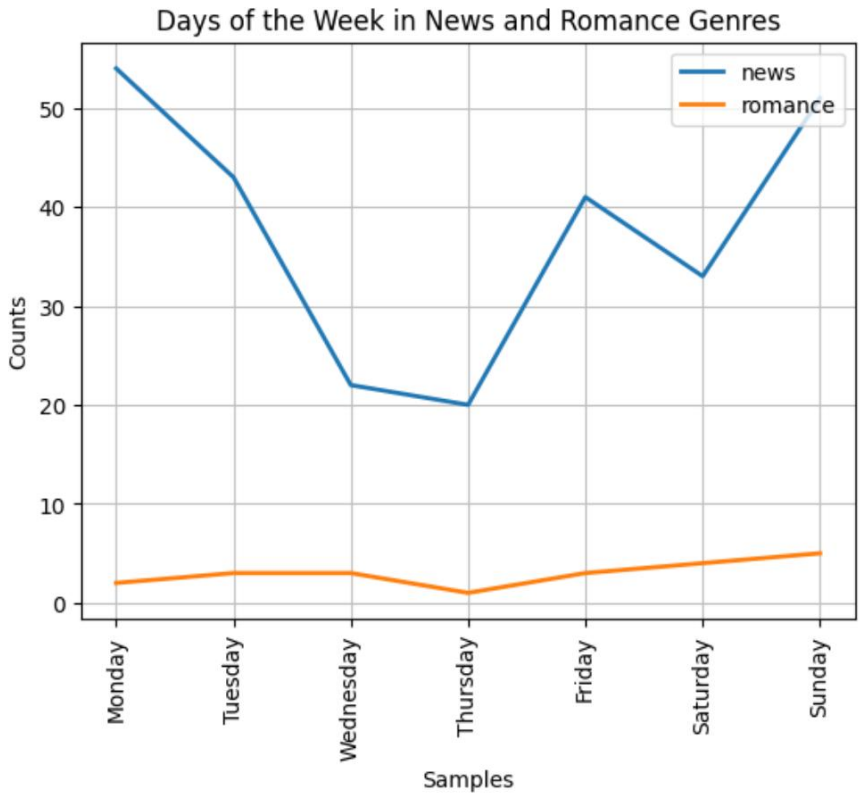
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]

['Does', 'our', 'society', 'have', 'a', 'runaway', ',', ...]

170576

['news', 'romance']

Tabulation: Monday Tuesday Wednesday Thursday Friday Saturday Sunday news 54 43 22 20 41 33 51 romance 2 3 3 1 3 4 5 Plot:



['Hi', 'all.This', 'is', 'fun', '.'] ['Hi all.This is fun.']

[['[', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare', '1603', ']'], ['Actus', 'Primus', '.'], ...]

Synset('ambulance.n.01')

['Model_T','S.U.V.','SUV','Stanley_Steamer','ambulance', 'beach_waggon' 'beach_wagon' 'bus', 'cab', 'compact', 'compact_car','convertible']

Ex: 2 **Processing Raw Text**

9.7.24

Problem Statement:

Write a program to process Raw Text files with below operations

1. Accessing Text from Web
2. Regular Expression for Detecting word pattern
3. Normalizing the text
4. Regulars Expression for tokenizing text

Procedure:

Step 1: Install Required Libraries

- Definition: Install necessary Python libraries (requests , BeautifulSoup , feedparser , re , nltk) that are used for web scraping, RSS feed parsing, and NLP tasks.

- pip install commands might be needed to install these libraries.

Step 2: Load Libraries

- Definition: Import required libraries:

- requests for making HTTP requests.

- BeautifulSoup from bs4 for parsing HTML content.

- feedparser for parsing RSS feeds.

- re for regular expressions.

- nltk for natural language processing tasks like tokenization, stemming, and lemmatization.

Step 3: Scrape Web Page Content

- Definition: Make a GET request to a webpage using requests.get(url) and parse its content using BeautifulSoup .

- Task: Extract all the paragraph (<p>) elements from the HTML and combine them into a single text block.

Step 4: Process the Scraped Text

- Definition: Process the extracted text by determining its data type (str) and measuring its length.

- Print a snippet of the first 100 characters as a preview.

Step 5: Parse RSS Feed

- Definition: Use the feedparser library to parse an RSS feed.

- Task: Retrieve and display the titles and descriptions of the first few entries from the RSS feed.

Step 6: Pattern Matching with Regular Expressions

- Definition: Use regular expressions (re) to find specific patterns in text.

Sub-step 6.1: Find Words Starting with Vowels

- Definition: Use regular expressions to find all words in the text that begin with a vowel (case-insensitive).

Sub-step 6.2: Identify Past Tense Verbs

- Definition: Extract words that appear in the past tense by finding all words ending with "ed."

Sub-step 6.3: Find Phone Numbers

- Definition: Use regular expressions to detect phone numbers in different formats from the text.

Sub-step 6.4: Extract Email Addresses

- Definition: Use regular expressions to extract valid email addresses from the text.

Step 7: Normalize Text

- Definition: Normalize text by:

  - Removing digits and converting it to lowercase.

  - Removing punctuation and extra whitespace using regular expressions.

Step 8: Stemming

- Definition: Use PorterStemmer from NLTK to reduce words to their root form (stem). This process helps to group similar words together.

Step 9: Lemmatization

- Definition: Use WordNetLemmatizer from NLTK to reduce words to their base or dictionary form (lemma). This is more advanced than stemming as it considers the part of speech (POS).

Step 10: Text Tokenization

- Definition: Tokenize text, i.e., break the text into individual words or tokens using different methods:

Sub-step 10.1: Whitespace Tokenization

- Definition: Split the text into tokens based on whitespace.

Sub-step 10.2: Regex Tokenization

- Definition: Use regular expressions to tokenize the text, which is more flexible than whitespace splitting.

Sub-step 10.3: NLTK Tokenization

- Definition: Use nltk.regexp_tokenize for advanced tokenization, which allows tokenizing based on a defined pattern.

Code:

```
import requests

from bs4 import BeautifulSoup

url = https://en.wikipedia.org/wiki/Natural_language_processing

response = requests.get(url)

soup = BeautifulSoup(response.content, 'html.parser')

paragraphs = soup.find_all('p') text = ' '.join([para.get_text() for para in paragraphs])

print(type(text)) print(len(text)) print(text[:100]) !pip install feedparser import feedparser

url = http://feeds.bbci.co.uk/news/rss.xml

feed = feedparser.parse(url)

entries = feed.entries

for entry in entries[:5]:

        print("Title:", entry.title)

        print("Description:", entry.description)

         print() import re

text = "An apple a day keeps the doctor away. Understanding nature is important."

vowel_words = re.findall(r'\b[AEIOUaeiou][a-z] \b', text)

print(vowel_words)

text = "He walked to the store. She smiled and greeted her friends. The cat purred contentedly."

 past_tense_words = re.findall(r'\b\w+ed\b', text)

print(past_tense_words)

import re
```

```
text = """Customer service can be reached at 1-800-123-4567 during business hours.For urgent inquiries, call (555) 789-0123 or email support@example.com."""

phone_numbers = re.findall(r'\b(?:\+?(\d{1,3}))?[-. (] (\d{3})[-. )] (\d{3})[-. ] (\d{4})\b', text)

 print("Phone numbers found:")

 for phone in phone_numbers:

        print("".join(phone))

text_no_punctuation = re.sub(r'[^\w\s]', '', text)

text_normalized = re.sub(r'\s+', ' ', text_no_punctuation).strip()
print(text_normalized)

import nltk

from nltk.stem

import PorterStemmer

words = nltk.word_tokenize("The striped bats are hanging on their feet for best"
)

ps = PorterStemmer()

 stemmed_words = [ps.stem(word) for word in words]

print("Stemmed Words:", stemmed_words)

import nltk from nltk.stem

import WordNetLemmatizer

words = nltk.word_tokenize("The striped bats are hanging on their feet for best"
)

lemmatizer = WordNetLemmatizer()

lemmatized_words = [lemmatizer.lemmatize(word, pos='v')

for word in words] print("Lemmatized Words:", lemmatized_words)

 import re import nltk def tokenize_text(text):

  Method 1: Simple whitespace tokenization

tokens_whitespace = text.split()
```

```
print("Whitespace Tokenization:")
 print(tokens_whitespace)
  Method 2: Regex-based tokenization
 tokens_regex = re.findall(r'\b\w+\b', text)
 print("\nRegex Tokenization:")
print(tokens_regex)
  Method 3: NLTK's regexp_tokenize for advanced tokenization
pattern = r'\b\w+\b'
tokens_nltk = nltk.regexp_tokenize(text, pattern)
print("\nNLTK Tokenization:")
print(tokens_nltk)
quote_text = "Imagination is more important than knowledge."   Tokenize the text using different methods tokenize_text(quote_text)
```

Output:

<class 'str'> 7725 Natural language processing (NLP) is an interdisciplinary subfield of computer science and artificial intelligence. It is primarily concerned with providing computers the ability to process data encoded in natural language and is thus closely related to information retrieval, knowledge representation and computational linguistics, a subfield of linguistics. Typically data is collected in text corpora, using either rule-based, statistical or neural-based approaches of machine learning and deep

Title: 'Levelling up' phrase to be erased, says minister Description: Jim McMahon says it was "only ever a slogan" and would be taken out of his department's name. Title: US blocks British court from British territory Description: The island hosts a secretive UK-US military base and access is heavily restricted. Title: This Nato summit could save or sink Biden's candidacy Description: The gathering of leaders will inflict more scrutiny on the president. Will it be a reprieve or his last stand? Title: Dyson to cut about 1,000 jobs in the UK Description: The firm, known for the invention of the bag-less vacuum cleaner, made the announcement in response to global markets. Title: Europe's Ariane-6 rocket poised for debut launch Description: It's taken 10 years to

develop and cost billions but the new launch vehicle is finally ready to fly.
['An', 'apple', 'a', 'away', 'Understanding', 'is', 'important']

['walked', 'smiled', 'greeted', 'purred']

Phone numbers found: 18001234567 5557890123 Email addresses:
['support@example.com', 'info@company.co.uk']

nlp with nltk is fun! it was developed in the s.

NLP with NLTK is fun It was developed in the 1990s

Stemmed Words: ['the', 'stripe', 'bat', 'are', 'hang', 'on', 'their', 'feet', 'for', 'best']

Whitespace Tokenization:
['Imagination', 'is', 'more', 'important', 'than', 'knowledge.']

Regex Tokenization:
['Imagination', 'is', 'more', 'important', 'than', 'knowledge']

NLTK Tokenization:
['Imagination', 'is', 'more', 'important', 'than', 'knowledge']

Ex: 3 **Categorizing and Tagging Words**

22.7.24

Problem Statement:

Categorizing and Tagging Words

I) Automatic Tagging

II) N Gram Tagging

III) Transformation Based Tagging

Procedure:

Step 1: Load Libraries

- Definition: Import necessary libraries for POS tagging and corpus data:
  - `nltk`: For Natural Language Processing tasks.
  - `word_tokenize`: For tokenizing text into words.
  - `brown`: The Brown corpus for training POS taggers.
  - `brill` and `brill_trainer`: For Brill tagging.

Step 2: Train a Lookup Tagger

- Definition: Use a predefined dictionary of word-tag pairs for a simple lookup-based POS tagging method.
  - Lookup Tagger Model: Train a `UnigramTagger` using the dictionary that defines tags for specific words like `'The'`, `'quick'`, `'fox'`, etc.
  - Task: Tokenize a sentence like `"The quick brown fox jumps over the lazy dog."` and apply the lookup tagger to it.

Step 3: Use Default POS Tagger

- Definition: Load the default POS tagger using `nltk.PerceptronTagger()`, which tags words based on a pre-trained model.
  - Task: Tokenize a sentence and apply the default tagger to it.

Step 4: Train a Unigram Tagger

- Definition: Train a Unigram Tagger using tagged sentences from the Brown corpus.

- Unigram Tagger: This tagger assigns a POS tag to each word based solely on the word itself, using the most frequent tag for each word.

- Task: Tokenize a sentence like `"Data science is an interdisciplinary field."` and apply the Unigram tagger to it.

Step 5: Train a Bigram Tagger (with Backoff to Unigram Tagger)

- Definition: Train a Bigram Tagger that uses word pairs to improve accuracy, with backoff to the Unigram tagger when no bigram is found.

- Bigram Tagger: It assigns POS tags based on pairs of words (the current word and the previous word).

- Task: Tokenize a sentence and apply the Bigram tagger with backoff to the Unigram tagger.

Step 6: Train a Trigram Tagger (with Backoff to Bigram and Unigram Taggers)

- Definition: Train a Trigram Tagger that uses triples of words for even more accurate tagging, with backoff to the Bigram and Unigram taggers.

- Trigram Tagger: It assigns POS tags based on triples of words (the current word and the two previous words).

- Task: Tokenize a sentence and apply the Trigram tagger with backoff.

Step 7: Define Brill Tagger

- Definition: Define a `BrillTagger` with initial backoff to the Unigram Tagger.

- Brill Tagger: A rule-based POS tagger that adjusts tags based on transformational rules learned from the training data.

- Task: Train the Brill Tagger using the Brown corpus and predefined templates for rule generation.

Step 8: Train Brill Tagger

- Definition: Train the Brill Tagger using the Brown corpus and limit the number of transformation rules to 10.

- Task: Tokenize sentences like `"Machine learning is a branch of artificial intelligence."` and `"Natural language processing enables computers to understand human language."` and apply the Brill tagger.

Step 9: Display Tagged Output

- Definition: Display the POS tagging results for each of the taggers:

- Lookup Tagger

- Default Perceptron Tagger

- Unigram-Bigram-Backoff Tagger

- Unigram-Bigram-Trigram-Backoff Tagger

- Brill Tagger

Code:

```
import nltk

from nltk.corpus

import brown

from nltk

import word_tokenize from nltk.data

 import load # Train a lookup tagger with some predefined word-tag pairs
train_data = brown.tagged_sents(categories='news')

lookup_dict = { 'The': 'DT', 'quick': 'JJ', 'brown': 'JJ', 'fox': 'NN', 'jumps': 'VBZ', 'over': 'IN', 'the': 'DT', 'lazy': 'JJ', 'dog': 'NN' } # Load default POS tagger
default_tagger = nltk.PerceptronTagger()

text2 = "The quick brown fox jumps over the lazy dog."
```

```
tokens2 = word_tokenize(text2)

tagged2 = default_tagger.tag(tokens2) print("Default Tagger Output 2:",
tagged2)

 lookup_tagger = nltk.UnigramTagger(model=lookup_dict)

 text1 = "The quick brown fox jumps over the lazy dog."

 tokens1 = word_tokenize(text1)

tagged1 = lookup_tagger.tag(tokens1)

 print("Lookup Tagger Output 1:", tagged1) # Load default POS tagger
default_tagger = nltk.PerceptronTagger()

text2 = "The quick brown fox jumps over the lazy dog."

tokens2 = word_tokenize(text2)

tagged2 = default_tagger.tag(tokens2)

print("Default Tagger Output 2:", tagged2)

 import nltk from nltk.corpus

 import brown from nltk import word_tokenize # Train data from the Brown
corpus train_data = brown.tagged_sents(categories='news') # Train a unigram
tagger unigram_tagger = nltk.UnigramTagger(train_data) # Train a bigram
tagger with backoff to the unigram tagger

 bigram_tagger = nltk.BigramTagger(train_data, backoff=unigram_tagger)

 text1 = "Data science is an interdisciplinary field."

 tokens1 = word_tokenize(text1)

 tagged1 = bigram_tagger.tag(tokens1)

 print("Combined Unigram-Bigram Tagger Output 1:", tagged1) # Train a
trigram tagger with backoff to the bigram tagger, which backs off to the
unigram tagger

trigram_tagger = nltk.TrigramTagger(train_data, backoff=bigram_tagger)

text2 = "Data science is an interdisciplinary field"

tokens2 = word_tokenize(text2)

tagged2 = trigram_tagger.tag(tokens2)

 print("Combined Unigram-Bigram-Trigram Tagger Output 2:", tagged2)
```

```
from nltk.tag import brill, brill_trainer # Define initial backoff tagger (unigram)
initial_tagger = nltk.UnigramTagger(train_data) # Define Brill Tagger templates
templates = nltk.tag.brill.nltkdemo18()

trainer = brill_trainer.BrillTaggerTrainer(initial_tagger, templates) # Train the
Brill tagger brill_

tagger = trainer.train(train_data, max_rules=10)

text5 = "Machine learning is a branch of artificial intelligence."

 tokens5 = word_tokenize(text5)

tagged5 = brill_tagger.tag(tokens5)

print("Brill Tagger Output 5:", tagged5)

text6 = "Natural language processing enables computers to understand human
language."

 tokens6 = word_tokenize(text6)

tagged6 = brill_tagger.tag(tokens6)

 print("Brill Tagger Output 6:", tagged6)
```

Output:

```
Lookup Tagger Output 1: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'),
('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', None)]

Default Tagger Output 2: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox',
'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.',
'.')]

Combined Unigram-Bigram Tagger Output 1: [('Data', None), ('science', 'NN'),
('is', 'BEZ'), ('an', 'AT'), ('interdisciplinary', None), ('field', 'NN'), ('.', '.')]

Brill Tagger Output 5: [('Machine', None), ('learning', 'NN'), ('is', 'BEZ'), ('a',
'AT'), ('branch', 'NN'), ('of', 'IN'), ('artificial', 'JJ'), ('intelligence', None), ('.', '.')]

Brill Tagger Output 6: [('Natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'),
('enables', 'VBZ'), ('computers', 'NNS'), ('to', 'TO'), ('understand', 'VB'),
('human', 'NN'), ('language', 'NN'), ('.', '.')]
```

Ex: 4 **Text Classifier**

31.7.24

Problem Statement:

Text Classification using supervised learning classifier and neural network

Procedure:

Step 1: Import Libraries

- Definition: Import necessary libraries for the tasks:

- numpy: For mathematical operations (e.g., calculating RMSE).

- MLPClassifier: A neural network classifier from scikit-learn.

- NearestCentroid: A simple classifier that classifies based on centroids of the data.

- accuracy_score, precision_score, recall_score, mean_squared_error, classification_report: Metrics for evaluating the models.

- matplotlib: For plotting the comparison between classifiers.

Step 2: Initialize Neural Network Classifier

Definition: Initialize the MLPClassifier with a neural network architecture containing multiple hidden layers (256, 128, 64, 32), using 2000 maximum iterations and enabling early stopping to prevent overfitting.

- MLPClassifier: This classifier uses backpropagation and gradient-based optimization to train the neural network.

Step 3: Train Neural Network Classifier

- Definition: Fit the Neural Network model to the training data (train_features_scaled, train_labels_numeric).

- Task: The model will learn from the input features and corresponding labels to minimize the classification error.

Step 4: Make Predictions with Neural Network

- Definition: Predict the labels on the test set (test_features_scaled) using the trained Neural Network classifier.

- Task: The model outputs predicted labels for the unseen test data.

Step 5: Evaluate Neural Network Classifier

- Definition: Calculate performance metrics for the Neural Network model:

- Accuracy: Proportion of correct predictions.

- Precision: Proportion of true positive predictions out of all positive predictions.

- Recall: Proportion of true positives out of all actual positive cases.

- MSE (Mean Squared Error): Average squared difference between predicted and actual labels.

- RMSE (Root Mean Squared Error): Square root of the MSE, providing a measure of prediction error in the same units as the labels.

Step 6: Display Neural Network Classifier Results

- Definition: Print the calculated metrics for the Neural Network model and display a classification report, showing detailed precision, recall, and F1-scores for each class.

Step 7: Initialize Nearest Centroid Classifier

Step 8: Evaluate Nearest Centroid Classifier

- Definition: Calculate performance metrics for the Nearest Centroid model, similar to the Neural Network evaluation:

- Accuracy, Precision, Recall, MSE, RMSE.

Step 9: Display Nearest Centroid Classifier Results

- Definition: Print the calculated metrics for the Nearest Centroid model and display a classification report.

Step 10: Compare Classifiers

- Definition: Create a comparison between the Neural Network and Nearest Centroid classifiers using a bar chart.

- Metrics for Comparison: Accuracy, Precision, Recall, MSE, RMSE.

- Visualization: Create a bar chart using matplotlib with two bars for each metric, one for the Neural Network and one for the Nearest Centroid classifier.

Step 11: Annotate Bar Chart

- Definition: Add annotations to the bars in the chart to display the actual metric values above each bar for better visualization.

Step 12: Display the Plot

- Definition: Display the bar chart comparing the two classifiers.

Code:

```
import nltk
nltk.download('webtext')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')

from nltk.corpus import webtext
from nltk import pos_tag, word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import DictVectorizer
from sklearn.preprocessing import LabelEncoder, StandardScaler

text = webtext.raw('pirates.txt')
words = word_tokenize(text)
tagged_words = pos_tag(words)

filtered_words = [(word, pos) for word, pos in tagged_words if pos in ('NN',
'VB', 'JJ', 'RB')]

def word_features(word, context, window=2):
    features = {}
    for i in range(1, window + 1):
        if i <= len(context):
```

```
            features[f'prev_word_{i}'] = context[-i]
        else:
            features[f'prev_word_{i}'] = '<START>'
    return features

dataset = []
for i, (word, pos) in enumerate(filtered_words[:1000]):
    context = filtered_words[max(0, i-2):i]
    context_words = [cw for cw, cp in context]
    features = word_features(word, context_words)
    dataset.append((features, pos))

train_set, test_set = train_test_split(dataset, test_size=0.25, random_state=42)
train_features, train_labels = zip(*train_set)
test_features, test_labels = zip(*test_set)

label_encoder = LabelEncoder()
train_labels_numeric = label_encoder.fit_transform(train_labels)
test_labels_numeric = label_encoder.transform(test_labels)

vectorizer = DictVectorizer(sparse=False)
train_features_vectorized = vectorizer.fit_transform(train_features)
test_features_vectorized = vectorizer.transform(test_features)

scaler = StandardScaler()
train_features_scaled = scaler.fit_transform(train_features_vectorized)
test_features_scaled = scaler.transform(test_features_vectorized)
```

```
import numpy as np

from sklearn.neural_network import MLPClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, mean_squared_error, classification_report

import matplotlib.pyplot as plt

clf_nn = MLPClassifier(hidden_layer_sizes=(256, 128, 64, 32), max_iter=2000, random_state=42, early_stopping=True)

clf_nn.fit(train_features_scaled, train_labels_numeric)

predicted_labels_nn = clf_nn.predict(test_features_scaled)

accuracy_nn = accuracy_score(test_labels_numeric, predicted_labels_nn)

precision_nn = precision_score(test_labels_numeric, predicted_labels_nn, average='weighted', zero_division=1)

recall_nn = recall_score(test_labels_numeric, predicted_labels_nn, average='weighted')

mse_nn = mean_squared_error(test_labels_numeric, predicted_labels_nn)

rmse_nn = np.sqrt(mse_nn)

print("\nNeural Network Classifier:")

print(f'Accuracy: {accuracy_nn * 100:.2f}%')

print(f'Precision: {precision_nn:.2f}')

print(f'Recall: {recall_nn:.2f}')

print(f'MSE: {mse_nn:.2f}')

print(f'RMSE: {rmse_nn:.2f}')

print(classification_report(test_labels_numeric, predicted_labels_nn)) from sklearn.neighbors import NearestCentroid

clf_nc = NearestCentroid()

clf_nc.fit(train_features_scaled, train_labels_numeric)


predicted_labels_nc = clf_nc.predict(test_features_scaled)

accuracy_nc = accuracy_score(test_labels_numeric, predicted_labels_nc)
```

```
precision_nc = precision_score(test_labels_numeric, predicted_labels_nc,
average='weighted', zero_division=1)

recall_nc = recall_score(test_labels_numeric, predicted_labels_nc,
average='weighted')

mse_nc = mean_squared_error(test_labels_numeric, predicted_labels_nc)

rmse_nc = np.sqrt(mse_nc)

print("\nNearest Centroid Classifier:")

print(f'Accuracy: {accuracy_nc * 100:.2f}%')

print(f'Precision: {precision_nc:.2f}')

print(f'Recall: {recall_nc:.2f}')

print(f'MSE: {mse_nc:.2f}')

print(f'RMSE: {rmse_nc:.2f}')

print(classification_report(test_labels_numeric, predicted_labels_nc))
```

Output:

```
Neural Network Classifier:
Accuracy: 61.60%
Precision: 0.62
Recall: 0.62
MSE: 0.67
RMSE: 0.82
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        36
           1       0.63      0.97      0.76       158
           2       0.00      0.00      0.00        32
           3       0.00      0.00      0.00        24

    accuracy                           0.62       250
   macro avg       0.16      0.24      0.19       250
weighted avg       0.40      0.62      0.48       250

Nearest Centroid Classifier:
```

Accuracy: 52.80%

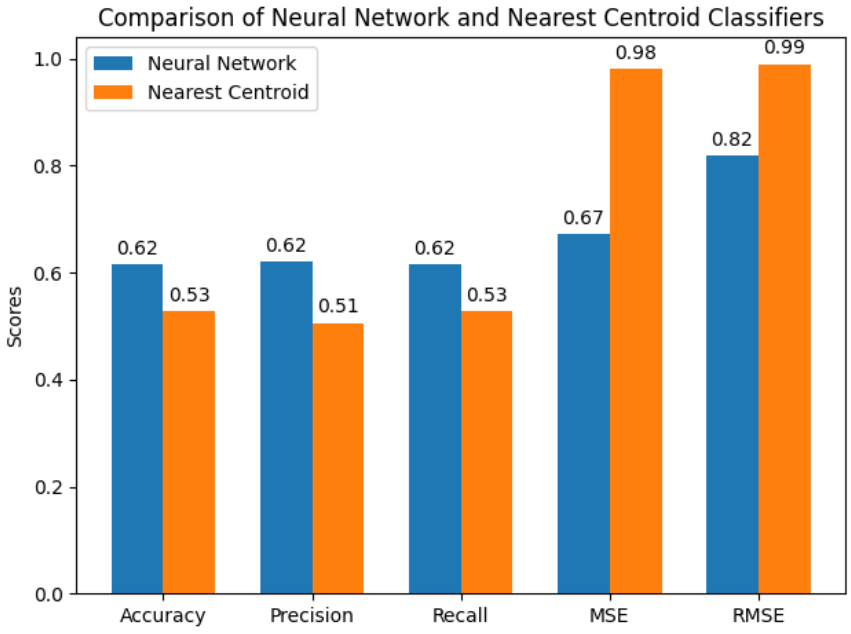
Precision: 0.51

Recall: 0.53

MSE: 0.98

RMSE: 0.99

	precision	recall	f1-score	support
0	0.19	0.22	0.20	36
1	0.65	0.73	0.68	158
2	0.33	0.12	0.18	32
3	0.29	0.21	0.24	24
accuracy			0.53	250
macro avg	0.36	0.32	0.33	250
weighted avg	0.51	0.53	0.51	250



EXERCISE :05 **TEXT EXTRACTION FROM TEXT**

PROBLEM STATEMENT :

The designing of a program to extracting information from text.

(i) Chunking
(ii) Named entity recognition
(iii) Relation extraction

PROCEDURE:

Step 01: Import Libraries
Start by importing necessary libraries from NLTK for tokenization, POS tagging, named entity recognition (NER), and chunking.

Step 02: Download NLTK Resources
Download required datasets and models, like tokenizers, named entity recognizers, and taggers, to perform the analysis.

Step 03: Input Text
Define the input text, which contains information about people, places, and organizations.

Step 04: Tokenization
Split the text into individual tokens (words and punctuation) to prepare it for further processing.

Step 05: POS Tagging
Assign part-of-speech tags to each token to identify its grammatical category (noun, verb, etc.).

Step 06: Named Entity Recognition (NER)
Perform named entity recognition to detect entities such as people, places, and organizations, classifying them by type (e.g., "PERSON", "ORGANIZATION").

Step 07: Display Named Entities
Extract and print the recognized named entities and their categories.

Step 08: Convert Named Entity Tree to IOB Tags
Convert the entity tree into IOB format to label each word as inside, outside, or beginning of a named entity, making it easier for relation extraction.

Step 09: Relation Extraction
Use a rule-based method to find relationships between consecutive entities, like pairs of people or places.

Step 10: Chunking with Defined Grammar
Apply a chunking grammar to group tokens into phrases (noun phrases, verb phrases, etc.) based on their POS tags.

Step 11: Display Chunks
Display the identified chunks, showing how the text is structured in terms of grammatical phrases.

Step 12: Results
Named entities, relationships, and phrase chunks are now extracted, providing structured information from the raw text.

CODE:

```
!pip install nltk

import nltk

from nltk import word_tokenize, pos_tag, ne_chunk

from nltk.chunk import conlltags2tree, tree2conlltags, RegexpParser

from nltk.corpus import conll2002


nltk.download('punkt')

nltk.download('maxent_ne_chunker')

nltk.download('words')
```

```
nltk.download('conll2002')

nltk.download('averaged_perceptron_tagger')



text = """

Barack Obama was born in Honolulu, Hawaii. He served as the 44th President of the United States from 2009 to 2017.

Apple Inc. is an American multinational technology company headquartered in Cupertino, California. It was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne.

"""



tokens = word_tokenize(text)

tagged_tokens = pos_tag(tokens)



named_entities = ne_chunk(tagged_tokens)



print("Named Entities:")

for subtree in named_entities:

    if isinstance(subtree, nltk.Tree):

        entity_name = " ".join([token for token, pos in subtree.leaves()])

        entity_type = subtree.label()

        print(f"{entity_name}: {entity_type}")



conlltags = tree2conlltags(named_entities)
```

```
print("\nCONLL Tagging:")
for tag in conlltags:
    print(tag)


def extract_relations(tags):
    relations = []
    for i in range(len(tags) - 1):
        if tags[i][2] == 'B-PERSON' and tags[i+1][2] == 'I-PERSON':
            relations.append((tags[i][0], tags[i+1][0]))
        elif tags[i][2] == 'B-GPE' and tags[i+1][2] == 'I-GPE':
            relations.append((tags[i][0], tags[i+1][0]))
    return relations

relations = extract_relations(conlltags)
print("\nExtracted Relations:")
for rel in relations:
    print(rel)
grammar = r"""
  NP: {<DT|JJ|NN. >+}          Chunk sequences of DT, JJ, NN
  PP: {<IN><NP>}               Chunk prepositions followed by NP
  VP: {<VB. ><NP|PP|CLAUSE>+$}   Chunk verbs and their arguments
  CLAUSE: {<NP><VP>}           Chunk NP, VP
  """
```

```
chunk_parser = RegexpParser(grammar)
chunks = chunk_parser.parse(tagged_tokens)

print("\nChunks:")
for subtree in chunks:
    if isinstance(subtree, nltk.Tree):
        chunk = " ".join([token for token, pos in subtree.leaves()])
        print(f"{subtree.label()}: {chunk}")
```

OUTPUT:

Named Entities:

Barack: PERSON

Obama: PERSON

Honolulu: GPE

Hawaii: GPE

United States: GPE

Apple Inc.: PERSON

American: GPE

Cupertino: GPE

California: GPE

Steve Jobs: PERSON

Steve Wozniak: PERSON

Ronald Wayne: PERSON

CONLL Tagging:

('Barack', 'NNP', 'B-PERSON')

('Obama', 'NNP', 'B-PERSON')

('was', 'VBD', 'O')

('born', 'VBN', 'O')

('in', 'IN', 'O')

('Honolulu', 'NNP', 'B-GPE')

(',', ',', 'O')

('Hawaii', 'NNP', 'B-GPE')

('.', '.', 'O')

('He', 'PRP', 'O')

('served', 'VBD', 'O')

('as', 'IN', 'O')

('the', 'DT', 'O')

('44th', 'CD', 'O')

('President', 'NNP', 'O')

('of', 'IN', 'O')

('the', 'DT', 'O')

('United', 'NNP', 'B-GPE')

('States', 'NNPS', 'I-GPE')

('from', 'IN', 'O')

('2009', 'CD', 'O')

('to', 'TO', 'O')

('2017', 'CD', 'O')

('.', '.', 'O')

('Apple', 'NNP', 'B-PERSON')

('Inc.', 'NNP', 'I-PERSON')

('is', 'VBZ', 'O')

('an', 'DT', 'O')

('American', 'JJ', 'B-GPE')

('multinational', 'NN', 'O')

('technology', 'NN', 'O')

('company', 'NN', 'O')

('headquartered', 'VBD', 'O')

('in', 'IN', 'O')

('Cupertino', 'NNP', 'B-GPE')

(',', ',', 'O')

('California', 'NNP', 'B-GPE')

('.', '.', 'O')

('It', 'PRP', 'O')

('was', 'VBD', 'O')

('founded', 'VBN', 'O')

('by', 'IN', 'O')

('Steve', 'NNP', 'B-PERSON')

('Jobs', 'NNP', 'I-PERSON')

(',', ',', 'O')

('Steve', 'NNP', 'B-PERSON')

('Wozniak', 'NNP', 'I-PERSON')

(',', ',', 'O')

('and', 'CC', 'O')

('Ronald', 'NNP', 'B-PERSON')

('Wayne', 'NNP', 'I-PERSON')

('.', '.', 'O')

Extracted Relations:

('United', 'States')

('Apple', 'Inc.')

('Steve', 'Jobs')

('Steve', 'Wozniak')

('Ronald', 'Wayne')

Chunks:

NP: Barack Obama

PP: in Honolulu

NP: Hawaii

PP: as the

NP: President

PP: of the United States

NP: Apple Inc.

NP: an American multinational technology company

PP: in Cupertino

NP: California

PP: by Steve Jobs

NP: Steve Wozniak

NP: Ronald Wayne

EXERCISE : 06 **CONTEXT FREE GRAMMAR**

PROBLEM STATEMENT:

Analyzing the sentence structure using Context Free Grammar.

PROCEDURE :

Step 01: Import Libraries
Import required libraries from NLTK, including CFG, PCFG, and parsers like ChartParser, ViterbiParser, and EarleyChartParser.

Step 02: Simple CFG Method
Define simple_cfg() to demonstrate parsing using a basic context-free grammar (CFG).

Step 03: Initialize CFG
Set up a CFG with rules for sentence structures like S -> NP VP.

Step 04: Parse Sentences Using ChartParser
Parse sentences using ChartParser and print the resulting parse trees.

Step 05: Probabilistic CFG Method
Define probabilistic_cfg() for parsing using a probabilistic CFG (PCFG) where grammar rules have probabilities.

Step 06: Initialize PCFG
Set up a PCFG with rules and probabilities, e.g., NP -> DT NN [0.8].

Step 07: Parse Sentences Using ViterbiParser
Parse sentences using ViterbiParser to find the most probable parse tree.

Step 08: Earley Parsing Method
Define earley_parsing() to demonstrate parsing using the Earley parser for CFGs.

Step 09: Parse Sentences Using EarleyChartParser
Use EarleyChartParser to handle more complex or ambiguous sentences.

Step 10: Conclusion
Run all methods to compare parse trees generated by different parsing approaches.

CODE:

```
import nltk
from nltk import CFG, PCFG
from nltk.parse import ChartParser, ViterbiParser, EarleyChartParser

def simple_cfg():
    print("Method 1: Simple Context Free Grammar")

    grammar1 = CFG.fromstring("""
      S -> NP VP
      NP -> DT NN | DT NNS
      VP -> VBZ NP
      DT -> 'the'
      NN -> 'dog' | 'cat'
      NNS -> 'dogs' | 'cats'
      VBZ -> 'chases' | 'sees'
    """)

    parser1 = ChartParser(grammar1)

    sentences = [
        "the dog chases the cat",
        "the dogs sees the cats"
    ]

    for sentence in sentences:
```

```
        print(f"Parsing: '{sentence}'")
        for tree in parser1.parse(sentence.split()):
            print(tree)
        print("\n")

def probabilistic_cfg():
    print("Method 2: Probabilistic Context Free Grammar")

    grammar2 = PCFG.fromstring("""
      S -> NP VP [1.0]
      NP -> DT NN [0.8] | DT NNS [0.2]
      VP -> VBZ NP [1.0]
      DT -> 'the' [1.0]
      NN -> 'dog' [0.5] | 'cat' [0.5]
      NNS -> 'dogs' [0.6] | 'cats' [0.4]
      VBZ -> 'chases' [0.7] | 'sees' [0.3]
    """)

    parser2 = ViterbiParser(grammar2)

    sentences = [
        "the dog chases the cat",
        "the dogs sees the cat"
    ]

    for sentence in sentences:
        print(f"Parsing: '{sentence}'")
```

```
        for tree in parser2.parse(sentence.split()):
            print(tree)
        print("\n")

def earley_parsing():
    print("Method 3: Earley Parser")

    grammar3 = CFG.fromstring("""
      S -> NP VP
      NP -> DT NN | DT NNS
      VP -> VBZ NP
      DT -> 'the'
      NN -> 'dog' | 'cat'
      NNS -> 'dogs' | 'cats'
      VBZ -> 'chases' | 'sees'
    """)

    parser3 = EarleyChartParser(grammar3)

    sentences = [
        "the dog chases the cat",
        "the dogs sees the cats"
    ]

    for sentence in sentences:
        print(f"Parsing: '{sentence}'")
        for tree in parser3.parse(sentence.split()):
```

```
            print(tree)
        print("\n")

  Run all methods
simple_cfg()
probabilistic_cfg()
earley_parsing()
```

OUTPUT :

Method 1: Simple Context Free Grammar

```
Parsing: 'the dog chases the cat'
(S (NP (DT the) (NN dog)) (VP (VBZ chases) (NP (DT the) (NN cat))))

Parsing: 'the dogs sees the cats'
(S (NP (DT the) (NNS dogs)) (VP (VBZ sees) (NP (DT the) (NNS cats))))
```

Method 2: Probabilistic Context Free Grammar

```
Parsing: 'the dog chases the cat'
(S
  (NP (DT the) (NN dog))
  (VP (VBZ chases) (NP (DT the) (NN cat)))) (p=0.112)


Parsing: 'the dogs sees the cat'
(S
  (NP (DT the) (NNS dogs))
  (VP (VBZ sees) (NP (DT the) (NN cat)))) (p=0.0144)
```

Method 3: Earley Parser

Parsing: 'the dog chases the cat'

(S (NP (DT the) (NN dog)) (VP (VBZ chases) (NP (DT the) (NN cat))))

Parsing: 'the dogs sees the cats'

(S (NP (DT the) (NNS dogs)) (VP (VBZ sees) (NP (DT the) (NNS cats))))

EXERCISE :07 **SYNONYM OF SENTENCES**

PROBLEM STATEMENT :

Analyzing the synonym of sentences.

PROCEDURE :

Step 01: Import Libraries
Import Expression from nltk.sem to handle logical expressions.

Step 02: Define Propositional Logic Introduction Method
Create the propositional_logic_intro() function to introduce propositional logic with basic expressions P and Q.

Step 03: Print Basic Propositions
Print the propositional logic statements for P and Q.

Step 04: Define Basic Connectives Method
Create propositional_logic_basic_connectives() to demonstrate propositional connectives: conjunction (&), disjunction (|), implication (->), and negation (¬).

Step 05: Print Logical Operations
Print results of logical operations (e.g., P & Q, P | Q, P -> Q, and ¬P).

Step 06: Define Expression Evaluation Method
Create evaluate_expr() to evaluate logical expressions based on a given valuation for P and Q.

Step 07: Propositional Logic Semantics
Define propositional_logic_semantics() to evaluate logical expressions (e.g., P & Q, P | Q, ¬P, and P -> Q) under a specified valuation of P=True and Q=False.

Step 08: Define Propositional Logic Models
Create propositional_logic_models() to evaluate expressions under different valuations and print the results.

Step 09: First-Order Logic Introduction
Define first_order_logic_intro() to introduce first-order logic with predicates P(x) and Q(x) over a domain.

Step 10: Define First-Order Logic Predicates and Quantifiers
Create first_order_logic_predicates_quantifiers() to introduce quantifiers (forall, exists) and predicates over a domain and print them.

Step 11: Evaluate First-Order Logic Expressions
Create evaluate_expression() to evaluate first-order logic expressions under a specified valuation.

Step 12: First-Order Logic Semantics
Define first_order_logic_semantics() to evaluate expressions like P(a) and forall x (P(x) -> Q(x)) under a given valuation.

Step 13: Define Universal Quantifier Evaluation Method
Create evaluate_forall() to handle expressions with universal (forall) and existential (exists) quantifiers over a domain.

Step 14: First-Order Logic Models
Define first_order_logic_models() to evaluate expressions under different valuations for a specified domain and print the results.

CODE :

```
from nltk.sem import Expression

read_expr = Expression.fromstring


def propositional_logic_intro():

    Define propositional logic expressions

   P = read_expr('P')

   Q = read_expr('Q')


   print("Proposition P:", P)

   print("Proposition Q:", Q)


propositional_logic_intro()
```

```
read_expr = Expression.fromstring

def propositional_logic_basic_connectives():
      Define propositional logic expressions
    P = read_expr('P')
    Q = read_expr('Q')

      Define logical operations
    conjunction = read_expr('P & Q')        p AND q
    disjunction = read_expr('P | Q')        p OR q
    implication = read_expr('P -> Q')       p IMPLIES q
    negation = read_expr('¬P')              NOT p

    print("P & Q:", conjunction)
    print("P | Q:", disjunction)
    print("P -> Q:", implication)
    print("¬P:", negation)

propositional_logic_basic_connectives()

read_expr = Expression.fromstring

def evaluate_expr(expr, valuation):
    """
    Evaluate a logical expression based on the given valuation.
    """
    if expr == read_expr('P'):
```

```
        return valuation['P']
    elif expr == read_expr('Q'):
        return valuation['Q']
    elif expr == read_expr('P & Q'):
        return valuation['P'] and valuation['Q']
    elif expr == read_expr('P | Q'):
        return valuation['P'] or valuation['Q']
    elif expr == read_expr('¬P'):
        return not valuation['P']
    elif expr == read_expr('P -> Q'):
        return not valuation['P'] or valuation['Q']
    else:
        raise ValueError(f"Unknown expression: {expr}")

def propositional_logic_semantics():
     Define propositions
    P = read_expr('P')
    Q = read_expr('Q')

     Define valuation
    valuation = {
        'P': True,
        'Q': False
    }

     Define expressions
    expr1 = read_expr('P & Q')      p AND q
```

```
    expr2 = read_expr('P | Q')      p OR q
    expr3 = read_expr('¬P')         NOT p
    expr4 = read_expr('P -> Q')     p IMPLIES q

      Evaluate expressions under the valuation
    print("P & Q evaluates to:", evaluate_expr(expr1, valuation))
    print("P | Q evaluates to:", evaluate_expr(expr2, valuation))
    print("¬P evaluates to:", evaluate_expr(expr3, valuation))
    print("P -> Q evaluates to:", evaluate_expr(expr4, valuation))

propositional_logic_semantics()

read_expr = Expression.fromstring

def evaluate_expression(expr, valuation):
    """
    Evaluate the logical expression based on the given valuation.
    """
    if expr == read_expr('P'):
        return valuation['P']
    elif expr == read_expr('Q'):
        return valuation['Q']
    elif expr == read_expr('P & Q'):
        return valuation['P'] and valuation['Q']
    elif expr == read_expr('P | Q'):
        return valuation['P'] or valuation['Q']
    elif expr == read_expr('¬P'):
```

```
        return not valuation['P']
    elif expr == read_expr('P -> Q'):
        return not valuation['P'] or valuation['Q']
    else:
        raise ValueError(f"Unknown expression: {expr}")

def propositional_logic_models():
    # Define propositions and expressions
    expr = read_expr('P & Q')

    # Define different valuations
    valuation1 = {'P': True, 'Q': True}
    valuation2 = {'P': True, 'Q': False}

    # Evaluate expression under different valuations
    result1 = evaluate_expression(expr, valuation1)
    result2 = evaluate_expression(expr, valuation2)

    print("Valuation1 (P=True, Q=True) satisfies 'P & Q':", result1)
    print("Valuation2 (P=True, Q=False) satisfies 'P & Q':", result2)

propositional_logic_models()

read_expr = Expression.fromstring

def first_order_logic_intro():
    # Define domain and predicates
```

```
    domain = ['a', 'b']
    P = read_expr('P(x)')
    Q = read_expr('Q(x)')

    print("Domain:", domain)
    print("Predicate P:", P)
    print("Predicate Q:", Q)

first_order_logic_intro()

read_expr = Expression.fromstring

def first_order_logic_predicates_quantifiers():
      Define domain and predicates
    domain = ['a', 'b']
    P = read_expr('P(x)')
    Q = read_expr('Q(x)')

      Define expressions with quantifiers
    expr1 = read_expr('P(a)')
    expr2 = read_expr('forall x (P(x) -> Q(x))')
    expr3 = read_expr('exists x (P(x) & Q(x))')

    print("P(a):", expr1)
    print("forall x (P(x) -> Q(x)):", expr2)
    print("exists x (P(x) & Q(x)):", expr3)
```

```
first_order_logic_predicates_quantifiers()


read_expr = Expression.fromstring


def evaluate_expression(expr, valuation):
    """
    Evaluate the logical expression based on the given valuation.
    """
    if expr == read_expr('P(a)'):
        return valuation.get('P(a)', False)
    elif expr == read_expr('P(b)'):
        return valuation.get('P(b)', False)
    elif expr == read_expr('Q(a)'):
        return valuation.get('Q(a)', False)
    elif expr == read_expr('Q(b)'):
        return valuation.get('Q(b)', False)
    elif expr == read_expr('forall x (P(x) -> Q(x))'):
          Evaluates universally quantified statements
        for item in ['a', 'b']:
            if not (valuation.get(f'P({item})', False) <= valuation.get(f'Q({item})',
False)):
                return False
        return True
    else:
        raise ValueError(f"Unknown expression: {expr}")


def first_order_logic_semantics():
     Define expressions
```

```
    expr1 = read_expr('P(a)')
    expr2 = read_expr('forall x (P(x) -> Q(x))')

     Define valuations
    valuation = {
        'P(a)': True,
        'P(b)': False,
        'Q(a)': False,
        'Q(b)': True
    }

     Evaluate expressions
    print("P(a) is:", evaluate_expression(expr1, valuation))
    print("forall x (P(x) -> Q(x)) is:", evaluate_expression(expr2, valuation))

first_order_logic_semantics()

read_expr = Expression.fromstring

def evaluate_forall(expr, valuation, domain):
    """
    Evaluates expressions with a universal quantifier.
    """
    if expr == read_expr('forall x (P(x) -> Q(x))'):
        return all(valuation.get(f'P({d})', False) <= valuation.get(f'Q({d})', False)
for d in domain)
    elif expr == read_expr('exists x (P(x) & Q(x))'):
```

```
        return any(valuation.get(f'P({d})', False) and valuation.get(f'Q({d})', False)
for d in domain)
    else:
        raise ValueError(f"Unknown expression {expr}")


def first_order_logic_models():
     Define domain and predicates
    domain = ['a', 'b']


     Define first-order logic expressions
    expr1 = read_expr('forall x (P(x) -> Q(x))')
    expr2 = read_expr('exists x (P(x) & Q(x))')


     Define a valuation function
    valuation = {
        'P(a)': True,
        'P(b)': False,
        'Q(a)': False,
        'Q(b)': True
    }


    print("forall x (P(x) -> Q(x)) is:", evaluate_forall(expr1, valuation, domain))
    print("exists x (P(x) & Q(x)) is:", evaluate_forall(expr2, valuation, domain))


first_order_logic_models()
```

OUTPUT :

Proposition P: P

Proposition Q: Q

P & Q: (P & Q)

P | Q: (P | Q)

P -> Q: (P -> Q)

¬P: ¬P

P & Q evaluates to: False

P | Q evaluates to: True

¬P evaluates to: False

P -> Q evaluates to: False

Valuation1 (P=True, Q=True) satisfies 'P & Q': True

Valuation2 (P=True, Q=False) satisfies 'P & Q': False

Domain: ['a', 'b']

Predicate P: P(x)

Predicate Q: Q(x)

P(a): P(a)

forall x (P(x) -> Q(x)): all x.(P(x) -> Q(x))

exists x (P(x) & Q(x)): exists x.(P(x) & Q(x))

P(a) is: True

forall x (P(x) -> Q(x)) is: False

forall x (P(x) -> Q(x)) is: False

exists x (P(x) & Q(x)) is: False

MINI-PROJECT REPORT

AutoCorrect and Keyword Suggestion System Using NLP with both text and speech input system

Problem Statement:

In today's digital era, generating context-specific keywords is essential for content creation, optimizing search engines, and enhancing user engagement. Manually identifying the correct spelling and relevant keywords can be time-consuming and prone to error. There is a need for an intelligent system that not only autocorrects user inputs but also provides relevant, contextually appropriate keyword suggestions using NLP techniques.

This project addresses this problem by developing an intelligent system that uses text analysis, tokenization, and part-of-speech (POS) tagging to suggest relevant keywords, enhancing the overall content creation and search optimization process.

Objective:

The primary objective of this project is to develop an NLP-powered tool that:

- Automatically corrects spelling errors.
- Suggests relevant keywords based on context and part-of-speech tagging.
- Enhances user experience and reduces manual efforts in content generation.
- Improves search optimization by categorizing keywords according to their grammatical usage.

Scope:

The system offers the following functionalities:

- **Spelling Correction**: Uses a corpus-based spell-checking algorithm to suggest corrections.
- **Keyword Suggestion**: Utilizes tokenization and word embedding techniques to suggest contextually appropriate keywords.
- **POS-Based Search**: Categorizes search results based on part-of-speech (POS) tagging.
- **Voice Input**: Integrates a speech recognition system to convert voice inputs into text and process them for spell correction and keyword suggestion.

Tools and Technologies:

- **Programming Language**: Python
- **Framework**: Streamlit (for the user interface)
- **NLP Libraries**: Hugging Face Transformers, NLTK
- **Speech Recognition**: Google Speech API, SpeechRecognition package
- **Other Libraries**: Autocorrect, re (Regular expressions), collections (for Counter)
- **Corpus Data**: A text corpus file (`big.txt`) for spell correction and word probability calculations.

System Design:

Architecture Overview:

1. **User Input**:
    - Text Input: Users can type in a word or phrase to be corrected or analyzed.
    - Voice Input: Users can provide voice input, which is transcribed into text for processing.
2. **Spell Checker**:
    - A spelling correction module processes the input text and suggests corrections using word-level edits and corpus-based probability calculations.
3. **Keyword Suggestion**:
    - Based on the corrected word, the system suggests relevant keywords using tokenization and context analysis.
4. **POS Categorization**:
    - The results from the corpus search are categorized according to POS tags (Nouns, Verbs, Adjectives, etc.) using a pre-trained POS tagger from the Hugging Face library.

Modules:

- **AutoCorrect**: Corrects misspelled words by generating potential word edits and ranking them based on probability from the corpus.
- **Search Corpus**: Searches for words in the corpus and returns the surrounding context.
- **POS Tagger**: Applies part-of-speech tagging to categorize the search results

Implementation:

1. Data Loading and Preprocessing:

The project loads a large corpus file (big.txt) to be used for spell checking and word probability calculation.

 # Load the corpus and compute probabilities

```
def load_corpus(filename):
    corpus = read_corpus(filename)
    vocab = set(corpus)
    words_count = Counter(corpus)
    total_words_count = float(sum(words_count.values()))
    word_probabs = {word: words_count[word] / total_words_count for word in words_count.keys()}
    return corpus, vocab, word_probabs
```

2. Spelling Correction:

The correct_spelling function generates possible corrections for misspelled words using techniques such as deletion, insertion, replacement, and swapping of letters.

 # Spell corrector function

```
def correct_spelling(word, vocab, word_probabs):
    if word in vocab:
        return f"'{word}' is already correct."

    suggestions = level_one_edits(word) or level_two_edits(word) or [word]
    best_guesses = [w for w in suggestions if w in vocab]

    if not best_guesses:
```

```
        return f"Sorry, no suggestions found for '{word}'."

    # Rank suggestions based on probabilities
    suggestions_with_probabs = [(w, word_probabs[w]) for w in best_guesses]
    suggestions_with_probabs.sort(key=lambda x: x[1], reverse=True)

    return f"Suggestions for '{word}':\n" + "\n".join([f"{w}: {p:.4f}" for w, p in suggestions_with_probabs])
```

3. POS Tagging and Categorization:

Using Hugging Face's pre-trained POS tagger, the search results are categorized into different POS tags such as Noun, Verb, Adjective, etc.

```
# Load Hugging Face POS tagger model
pos_tagger = pipeline("token-classification", model="vblagoje/bert-english-uncased-finetuned-pos", aggregation_strategy="simple")

# Categorize search results by POS
for res in search_results:
    pos_tags = pos_tagger(res)
    # Categorize and display results by POS
```

4. Speech Recognition:

The system allows voice input for hands-free usage, which is transcribed into text for further processing.

```
def transcribe_audio():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        audio = recognizer.listen(source)
    try:
```

```
        text = recognizer.recognize_google(audio)
        return text
    except sr.UnknownValueError:
        return ""
```

APP.PY

```
import streamlit as st
from autocorrect.corrector import correct_spelling, load_corpus
from search.search_engine import search_corpus
from transformers import pipeline
import speech_recognition as sr

# Load corpus data
corpus, vocab, word_probabs = load_corpus('data/big.txt')

# Load Hugging Face POS tagger model
pos_tagger = pipeline("token-classification", model="vblagoje/bert-english-uncased-finetuned-pos", aggregation_strategy="simple")

# Set page config for better appearance
st.set_page_config(page_title="AutoCorrect Dashboard", layout="wide")

# Function to transcribe audio input
def transcribe_audio():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        st.sidebar.text("Listening...")
```

```
        audio = recognizer.listen(source)
    try:
        text = recognizer.recognize_google(audio)
        st.sidebar.text(f"Transcribed text: {text}")
        return text
    except sr.UnknownValueError:
        st.sidebar.error("Could not understand audio.")
        return ""
    except sr.RequestError:
        st.sidebar.error("Could not request results from Google Speech Recognition service.")
        return ""

# Sidebar for user input
st.sidebar.header("AutoCorrect Settings")
input_method = st.sidebar.selectbox("Choose Input Method:", ["Text Input", "Voice Input"])

if input_method == "Voice Input":
    word = transcribe_audio()
else:
    word = st.sidebar.text_input("Enter a word to check spelling:", "")

# Main title
st.title("🔍 AutoCorrect & POS Categorized Search System")

# If the user has entered a word, show suggestions and search results
if word:
```

```
    # Correct spelling
    result = correct_spelling(word.lower(), vocab, word_probabs)

    # Display the original and corrected word
    st.subheader("Spelling Correction")
    if ":" in result:
        corrected_word = result.split(":")[1].strip()
        st.markdown(f"Original Word: {word}")
        st.markdown(f"Corrected Suggestion: {corrected_word}")
    else:
        st.markdown(f"Original Word: {word}")
        st.markdown("No corrections found.")
        corrected_word = word  # Use the original word if no correction is found

    # Perform search using corrected spelling (first suggestion)
    search_results = search_corpus(corrected_word, corpus)

    # Display search results categorized by POS
    st.subheader("Search Results Categorized by POS")

    # Define the POS categories you want to include
    pos_categories = ["NOUN", "VERB", "ADJ", "ADV", "PROPN", "PRON",
"NUM", "INTJ"]

    # Create a dictionary to hold results for each POS category
    pos_results = {pos: [] for pos in pos_categories}

    if search_results:
```

```
        for res in search_results:
            # Apply POS tagging to each result
            pos_tags = pos_tagger(res)

            # Check which POS the corrected word appears as
            for tag in pos_tags:
                entity = tag['entity_group']
                if entity in pos_results:  # Check if entity is in pos_results
                    if tag['word'].lower() == corrected_word:
                        pos_results[entity].append(res)
                        break  # Exit the loop after finding the first match

        # Display results for each POS category
        for pos in pos_categories:
            st.subheader(f"Results where '{corrected_word}' is used as a {pos}:")
            if pos_results[pos]:
                for i, res in enumerate(pos_results[pos], start=1):
                    st.markdown(f"Result {i}: {res}")
            else:
                st.markdown(f"No results found where '{corrected_word}' is used as a {pos}.")
    else:
        st.warning(f"No search results found for {corrected_word}.")

# Footer for additional info
st.markdown("---")
st.markdown("### About This Tool")
st.write(
```

"This AutoCorrect & POS Categorized Search System helps you find corrections for your misspelled words and search relevant information from a corpus, while categorizing the results based on the grammatical usage of the word."

)

SEARCH_ENGINE.PY

```
# Basic search function that searches for a word in the corpus and returns surrounding words (context)
def search_corpus(word, corpus, context_size=5):
    results = []
    for i, w in enumerate(corpus):
        if w == word:
            # Extract context around the found word
            start = max(i - context_size, 0)
            end = min(i + context_size + 1, len(corpus))
            context = " ".join(corpus[start:end])
            results.append(context)

    return results if results else None
```

CORRECTOR.PY

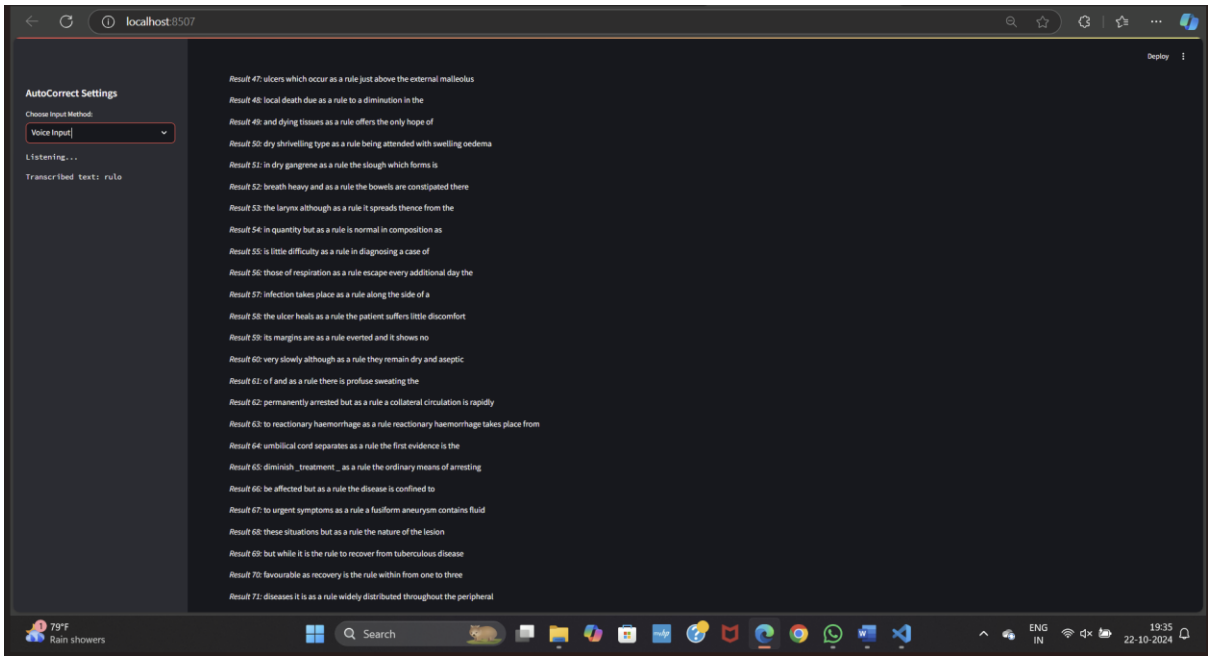
```
import re
import string
from collections import Counter

# Function to read the corpus from a text file and extract words
def read_corpus(filename):
    with open(filename, 'r', encoding='utf-8') as file:
```

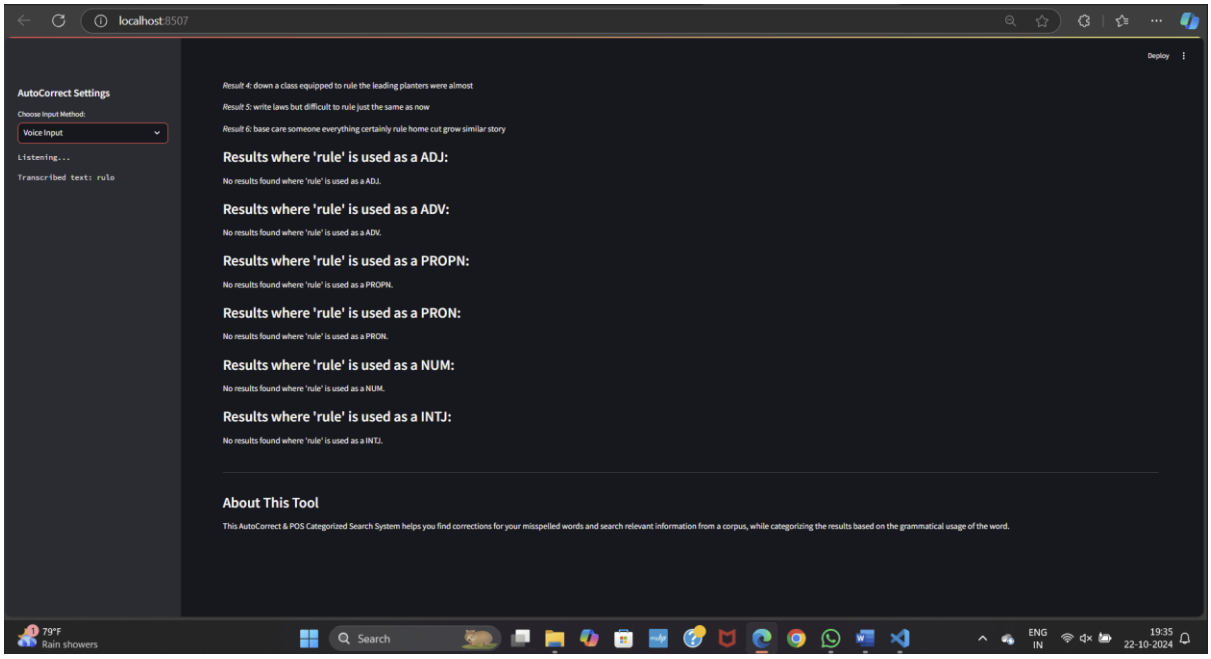
```
        lines = file.readlines()
        words = []
        for line in lines:
            words += re.findall(r'\w+', line.lower())
    return words

# Load the corpus and compute probabilities
def load_corpus(filename):
    corpus = read_corpus(filename)
    vocab = set(corpus)
    words_count = Counter(corpus)
    total_words_count = float(sum(words_count.values()))
    word_probabs = {word: words_count[word] / total_words_count for word in
words_count.keys()}
    return corpus, vocab, word_probabs

# Helper functions for generating word edits
def split(word):
    return [(word[:i], word[i:]) for i in range(len(word) + 1)]

def delete(word):
    return [left + right[1:] for left, right in split(word) if right]

def swap(word):
    return [left + right[1] + right[0] + right[2:] for left, right in split(word) if
len(right) > 1]

def replace(word):
```

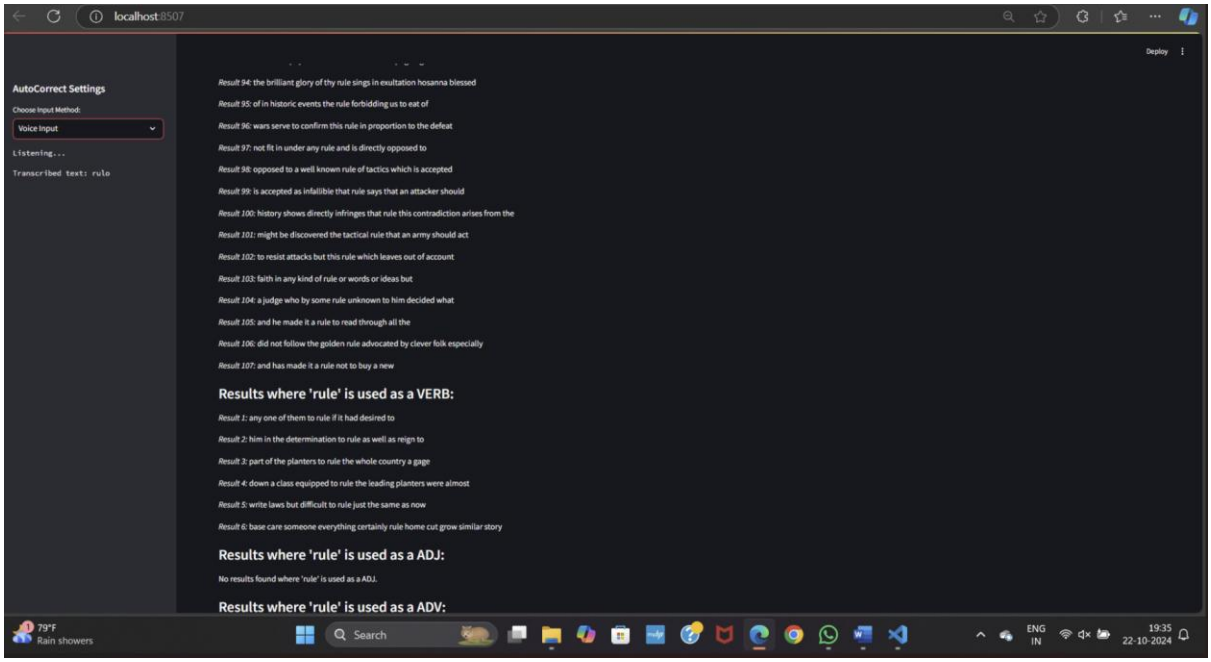
```
    return [left + char + right[1:] for left, right in split(word) if right for char in string.ascii_lowercase]

def insert(word):
    return [left + char + right for left, right in split(word) for char in string.ascii_lowercase]

# Function to generate level one edits
def level_one_edits(word):
    return set(delete(word) + swap(word) + replace(word) + insert(word))

# Function to generate level two edits
def level_two_edits(word):
    return set(e2 for e1 in level_one_edits(word) for e2 in level_one_edits(e1))

# Spell corrector function
def correct_spelling(word, vocab, word_probabs):
    if word in vocab:
        return f"'{word}' is already correct."

    # Generate suggestions
    suggestions = level_one_edits(word) or level_two_edits(word) or [word]
    best_guesses = [w for w in suggestions if w in vocab]

    if not best_guesses:
        return f"Sorry, no suggestions found for '{word}'."

    # Rank suggestions based on probabilities
```

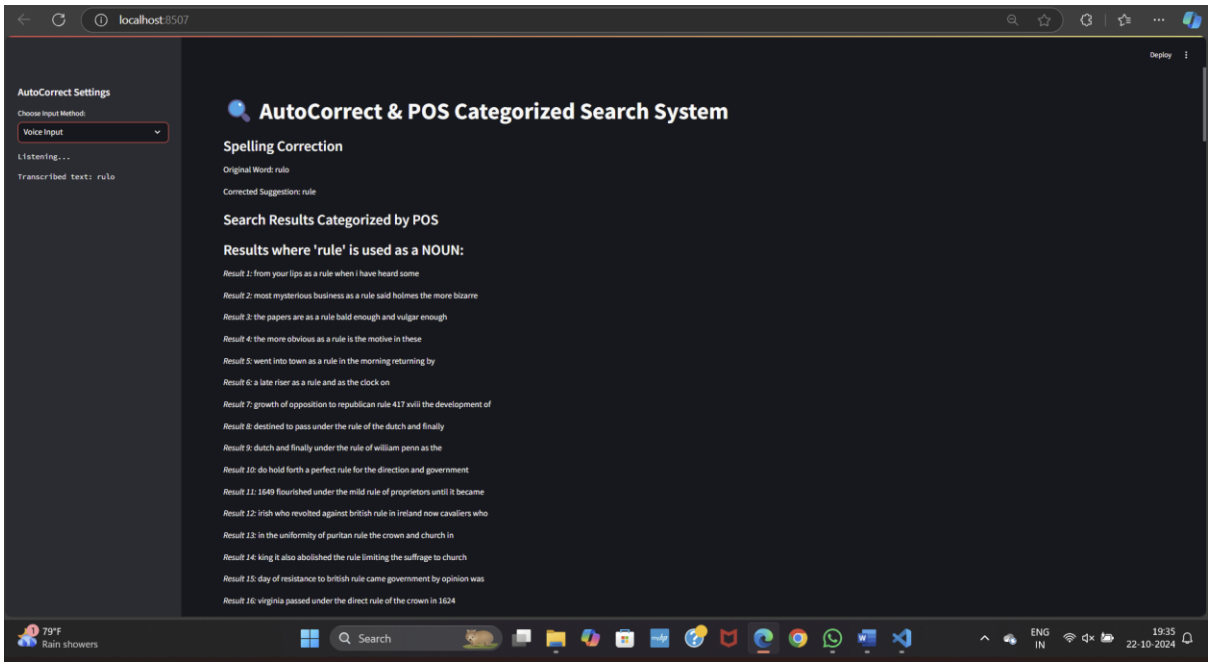
```
    suggestions_with_probabs = [(w, word_probabs[w]) for w in best_guesses]
    suggestions_with_probabs.sort(key=lambda x: x[1], reverse=True)

    return f"Suggestions for '{word}':\n" + "\n".join([f"{w}: {p:.4f}" for w, p in suggestions_with_probabs])
```

OUTPUT:






localhost:8507
Deploy
AutoCorrect Settings
Choose Input Method:
Voice Input
Listening...
Transcribed text: rulo
Result 94: the brilliant glory of thy rule sings in exultation hosanna blessed
Result 95: of in historic events the rule forbidding us to eat of
Result 96: wars serve to confirm this rule in proportion to the defeat
Result 97: not fit in under any rule and is directly opposed to
Result 98: opposed to a well known rule of tactics which is accepted
Result 99: is accepted as infallible that rule says that an attacker should
Result 100: history shows directly infringes that rule this contradiction arises from the
Result 101: might be discovered the tactical rule that an army should act
Result 102: to resist attacks but this rule which leaves out of account
Result 103: faith in any kind of rule or words or ideas but
Result 104: a judge who by some rule unknown to him decided what
Result 105: and he made it a rule to read through all the
Result 106: did not follow the golden rule advocated by clever folk especially
Result 107: and has made it a rule not to buy a new
Results where 'rule' is used as a VERB:
Result 1: any one of them to rule if it had desired to
Result 2: him in the determination to rule as well as reign to
Result 3: part of the planters to rule the whole country a gage
Result 4: down a class equipped to rule the leading planters were almost
Result 5: write laws but difficult to rule just the same as now
Result 6: base care someone everything certainly rule home cut grow similar story
Results where 'rule' is used as a ADJ:
No results found where 'rule' is used as a ADJ.
Results where 'rule' is used as a ADV:
79°F
Rain showers
Search
ENG
IN
19:35
22-10-2024


localhost:8507
Deploy
AutoCorrect Settings
Choose Input Method:
Voice Input
Listening...
Transcribed text: rulo
AutoCorrect & POS Categorized Search System
Spelling Correction
Original Word: rulo
Corrected Suggestion: rule
Search Results Categorized by POS
Results where 'rule' is used as a NOUN:
Result 1: from your lips as a rule when i have heard some
Result 2: most mysterious business as a rule said holmes the more bizarre
Result 3: the papers are as a rule bald enough and vulgar enough
Result 4: the more obvious as a rule is the motive in these
Result 5: went into town as a rule in the morning returning by
Result 6: a late riser as a rule and as the clock on
Result 7: growth of opposition to republican rule 417 xviii the development of
Result 8: destined to pass under the rule of the dutch and finally
Result 9: dutch and finally under the rule of william penn as the
Result 10: do hold forth a perfect rule for the direction and government
Result 11: 1649 flourished under the mild rule of proprietors until it became
Result 12: irish who revolted against british rule in ireland now cavaliers who
Result 13: in the uniformity of puritan rule the crown and church in
Result 14: king it also abolished the rule limiting the suffrage to church
Result 15: day of resistance to british rule came government by opinion was
Result 16: virginia passed under the direct rule of the crown in 1624
79°F
Rain showers
Search
ENG
IN
19:35
22-10-2024

Testing and Validation:

The system was tested with various inputs, including misspelled words, and was validated to ensure that the spelling correction was accurate and keyword suggestions were contextually appropriate. Additionally, POS tagging was checked to ensure correct categorization of search results.

Conclusion:

The AutoCorrect & Keyword Suggestion System successfully automates the process of correcting spelling errors and suggesting relevant keywords. By incorporating NLP techniques such as tokenization, word embedding, and POS tagging, the system significantly improves content creation workflows and search optimization efforts. The addition of speech-to-text functionality enhances accessibility and user experience, making the system versatile for a wide range of use cases.