PROGRAM-1

AIM: Preprocessing text document using NLTK of python program

1A. Stopword elimination

Source Code:

Stop word Elimination is a text preprocessing technique used in natural language processing (NLP) to eliminate common words (like "the," "a," "is") that don't carry much significant meaning for certain tasks. This process helps improve efficiency and focus on more relevant terms in text analysis.

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
# Sample text
text = "This is a sample sentence that showing the stopword removal."
# Get English stopwords and tokenize
stop_words = set(stopwords.words('english'))
tokens = word_tokenize(text.lower())
# Remove stopwords
filtered_tokens = [word for word in tokens if word not in stop_words]
print("Original:", tokens)
print("Filtered text:", filtered_tokens)
```

OUTPUT:

```
Original: ['this', 'is', 'a', 'sample', 'sentence', 'that', 'showing', 'the', 'stopword', 'removal', '.']
Filtered text: ['sample', 'sentence', 'showing', 'stopword', 'removal', '.']
```

1B. Stemming:

Stemming and **lemmatization** are both text normalization techniques used in Natural Language Processing (NLP) to reduce words to their root form.
However, they differ in their approach and output.
Stemming is a faster, rule-based process that often results in non-real words and **Stemming** just removes or stems the last few characters of a word, often leading to incorrect meanings and spelling., while **lemmatization** is slower, dictionary-based, and produces valid words (lemmas). **Lemmatization** considers the context and converts the word to its meaningful base form, which is called Lemma

Stemming:

- **Process:**

Stemming involves cutting off the prefixes and suffixes of words to obtain a root form, also known as a stem.

- **Example:**

"running," "runs," and "ran" would all be stemmed to "run".

- **Output:**

Stemming often produces stems that are not actual words (e.g., stemming "lazy" might result in "lazi").

Source Code 1 for STEMMING

```
from nltk.stem import PorterStemmer
porter_stemmer = PorterStemmer()
words = ["running", "jumps", "happily", "running", "happily"]
stemmed_words = [porter_stemmer.stem(word) for word in words]
print("Original words:", words)
print("Stemmed words:", stemmed_words)
```

OUTPUT:

```
Original words: ['running', 'jumps', 'happily', 'running', 'happily']
Stemmed words: ['run', 'jump', 'happili', 'run', 'happili']
```

Source Code 2 for STEMMING

```
import nltk
from nltk.stem import PorterStemmer
nltk.download("punkt")
# Initialize Python porter stemmer
ps = PorterStemmer()
# Example inflections to reduce
example_words = ["program","programming","programer","programs","programmed"]
# Perform stemming
print("{0:20}{1:19}".format("--Word--","--Stem--"))
for word in example_words:
  print ("{0:20}{1:20}".format(word, ps.stem(word)))
```

OUTPUT:

```
--Word--            --Stem--
Program             program
Programming         program
Programmer          program
Programs            program
Programmed          program
```

1C. Lemmatization:

- **Process:**

Lemmatization, on the other hand, reduces words to their base or dictionary form, called a lemma.

- **Example:**

"better" would be lemmatized to "good," while "studies" would be lemmatized to "study".

- **Output:**

Lemmatization produces valid dictionary words, unlike stemming.

Source Code 1 for LEMMATIZATION

```
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
text = "The cats were running faster than the dogs."
tokens = word_tokenize(text)
lemmatized_words = [lemmatizer.lemmatize(word) for word in tokens]
print(f"Original Text: {text}")
print(f"Lemmatized Words: {lemmatized_words}")
```

OUTPUT:

```
Original Text: The cats were running faster than the dogs.
Lemmatized Words: ['The', 'cat', 'were', 'running', 'faster', 'than', 'the', 'dog', '.']
```

Source Code 2 for LEMMATIZATION

```
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
sentence = "The children are running towards a better place."
tokens = word_tokenize(sentence)
lemmatized_sentence = []
for word, tag in tagged_tokens:
  lemmatized_sentence.append(lemmatizer.lemmatize(word, get_wordnet_pos(tag)))
print("Original Sentence: ", sentence)
print("Lemmatized Sentence: ", ' '.join(lemmatized_sentence))
```

OUTPUT:

```
Original Sentence:  The children are running towards a better place.
Lemmatized Sentence:  The child be run towards a good place .
```

1D. POS TAGGING

POS (Part-of-Speech) tagging in NLP is the process of assigning grammatical categories (like noun, verb, adjective, etc.) to each word in a sentence. It's a fundamental step in understanding the structure and meaning of text. Essentially, it helps determine the role each word plays within a sentence

```
#importing libraries
import spacy
# Load the English language model
nlp = spacy.load("en_core_web_sm")
# Sample text
text = "The quick brown fox jumps over the lazy dog.."
# Process the text with SpaCy
doc = nlp(text)
print("Original Text: ", text)
print("PoS Tagging Result:")
for token in doc:
    print(f"{token.text}: {token.pos_}")
```

OUTPUT:

```
Original Text:  The quick brown fox jumps over the lazy dog..
PoS Tagging Result:
The: DET
quick: ADJ
brown: ADJ
fox: NOUN
jumps: VERB
over: ADP
the: DET
lazy: ADJ
dog: NOUN
..: PUNCT
```

1E. LEXICAL ANALYSIS

LEXICAL ANALYSIS, also known as scanning or tokenization, is the initial and fundamental step in Natural Language Processing (NLP). It involves breaking down raw text into a stream of smaller, meaningful units called tokens. These tokens can be words, punctuation marks, numbers, or other symbols that carry linguistic significance.
The primary goal of lexical analysis is to transform a sequence of characters into a structured representation that can be further processed by subsequent NLP stages, such as syntax analysis (parsing) and semantic analysis.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
text1 = "NLTK is a powerful library for NLP."
text2 = "This is the first sentence. This is the second sentence."
words = word_tokenize(text1)
sentences = sent_tokenize(text2)
print("Splitting into words \n",words)
print("\nSplitting into sentences \n",sentences)
```

OUTPUT:

```
Splitting into words
 ['NLTK', 'is', 'a', 'powerful', 'library', 'for', 'NLP', '.']

Splitting into sentences
 ['This is the first sentence.', 'This is the second sentence.']
```

PROGRAM-2
Aim: SENTIMENT ANALYSIS ON CUSTOMER REVIEW ON PRODUCTS

Sentiment analysis on customer product reviews using NLTK in Python typically involves the following steps:

- **Data Collection and Preprocessing:**
  - Obtain a dataset of customer reviews (from e-commerce platforms).
  - Clean the text data by removing special characters, numbers, and irrelevant information.
  - Perform text normalization, removing stop words and stemming or lemmatization

- **Sentiment Analysis using NLTK's VADER:**
  - Import the SentimentIntensityAnalyzer from nltk.sentiment.vader.
  - Initialize the VADER analyzer.
  - For each review, use the polarity_scores() method to get sentiment scores (positive, negative, neutral, and a compound score).
  - The compound score provides an aggregated sentiment score ranging from -1 (most negative) to +1 (most positive).

CODE

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
def sentiment_analysis(text):
    analyzer = SentimentIntensityAnalyzer()
    sentiment = analyzer.polarity_scores(text)
    return sentiment

if __name__ == '__main__':
    text = "This product is absolutely amazing! I love it.."
    sentiment = sentiment_analysis(text)
    print("For positive Statement \n",sentiment)
    text2 = "I am very disappointed with this purchase.."
    sentiment = sentiment_analysis(text2)
    print("For negative Statement \n",sentiment)
```

OUTPUT:

```
For positive Statement
 {'neg': 0.0, 'neu': 0.361, 'pos': 0.639, 'compound': 0.8709}
For negative Statement
 {'neg': 0.404, 'neu': 0.596, 'pos': 0.0, 'compound': -0.5256}
```

PROGRAM-3
Aim:
3A. Web analytics on Web usage data (web, clickstreamanalysis)

Web analytics involving web usage data, including clickstream analysis, can be enhanced using the Natural Language Toolkit (NLTK) for tasks such as text processing of user-generated content or analyzing textual patterns within clickstream data.

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from collections import Counter

# Download required NLTK resources
nltk.download('stopwords')
nltk.download('punkt_tab')

# Sample clickstream data with search queries
search_queries = [
    "best laptop for students",
    "how to improve website performance",
    "buy new phone deals",
    "laptop reviews 2025",
    "website analytics tools"

# Preprocessing
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()

# Frequency analysis
print("Most common words in search queries:")
for word, count in word_counts.most_common(5):
    print(f"- {word}: {count}")

# N-gram analysis (example for bigrams)
bigrams = list(nltk.ngrams(processed_queries, 2))
bigram_counts = Counter(bigrams)
print("\nMost common bigrams:")
for bigram, count in bigram_counts.most_common(3):
    print(f"- {bigram}: {count}")
```

OUTPUT:

```
Most common words in search queries:
- laptop: 2
- websit: 2
- best: 1
- student: 1
- improv: 1

Most common bigrams:
- ('best', 'laptop'): 1
- ('laptop', 'student'): 1
- ('student', 'improv'): 1
```

3.B Web analytics on hyperlink data

Web analytics on hyperlink data, particularly within the context of an NLTK program, involves analyzing the structure and content of hyperlinks to gain insights into website usage, content relationships, and search engine optimization.

While NLTK is primarily a Natural Language Toolkit, it can be integrated into a web analytics workflow for text-based analysis of hyperlink attributes.

Process for analyzing hyperlink data with NLTK:

- **Data Acquisition:**
  - **Web Crawling:** Utilize a web crawler (e.g., Scrapy, Beautiful Soup with Requests, or even NLTK's own nltk.corpus.webtext for small-scale examples) to extract HTML content from web pages.
  - **Hyperlink Extraction:** Parse the HTML to identify and extract hyperlink elements (<a> tags) and their attributes, such as href (the URL), title, and the anchor text.

CODE:

```
import requests
import bs4

def hyperlink_analysis(url):
    response = requests.get(url)
    soup = bs4.BeautifulSoup(response.content, 'html.parser')
    links = soup.find_all('a')

    # Analyze the links
    link_counts = {}
    for link in links:

        url = link['href']
        if url not in link_counts:
            link_counts[url] = 0
        link_counts[url] += 1

    # Print the results
    for url, count in link_counts.items():
        print(f'{url}: {count}')

if __name__ == '__main__':
    url = 'https://www.aceec.ac.in/'
    hyperlink_analysis(url)
```

OUTPUT:

The output of the program will depend on the page at the URL that you specify. However, the output might include the various sample things from given URL as follows

https://www.aceec.ac.in/aicte-lite/: 1
https://alumni.aceec.ac.in/: 3
https://fee.aceec.ac.in/#/public-app/feePayment: 1
https://www.aceec.ac.in/ace-launchpad/: 1
https://www.aceec.ac.in/about-pals/: 1
https://www.aceec.ac.in/admissions/: 2
https://www.aceec.ac.in/principal/: 2
https://www.aceec.ac.in/administration-staff-list/: 2
https://www.aceec.ac.in/committees/: 2
https://www.aceec.ac.in/our-departments/: 2
https://www.aceec.ac.in/sports/: 2
https://www.aceec.ac.in/nss-2/: 2
https://www.aceec.ac.in/placement-cell/: 2
https://www.aceec.ac.in/toppers-interview-videos/: 3
https://www.aceec.ac.in/placement-cell/#PlacementData: 1

PROGRAM-4

Aim: Search engine optimization- implement spamdexing

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize # Import word_tokenize

def spamdexing(text):
    stopwords = nltk.corpus.stopwords.words('english')
    keywords = ['keyword1', 'keyword2', 'keyword3']
    tokens = word_tokenize(text) # Tokenize the input text
    filtered_text = [word for word in tokens if word not in stopwords] # Corrected 'notin' to 'not
in'
    for keyword in keywords:
        filtered_text.append(keyword * 10)
    return filtered_text # Corrected indentation of return statement

if __name__ == '__main__': # Corrected indentation and spacing
    text = "This is a sample text with stopwords."
    filtered_text = spamdexing(text)
    print(filtered_text)
```

OUTPUT:

```
['This', 'sample', 'text', 'stopwords', '.',
'keyword1keyword1keyword1keyword1keyword1keyword1keyword1keyword1keywor
d1keyword1',
'keyword2keyword2keyword2keyword2keyword2keyword2keyword2keyword2keywor
d2keyword2',
'keyword3keyword3keyword3keyword3keyword3keyword3keyword3keyword3keywor
d3keyword3']
```

PROGRAM-5

Aim: Use Google analytics tools to implement the following

5A. Conversion Statistics

```
import requests
def get_conversion_data(conversion_id):
 url = 'https://analytics.google.com/analytics/v3/data/ga?'
 params = {
'ids': 'ga:{conversion_id}',
'start-date': '2023-01-01',
'end-date': '2023-08-01',
'metrics': 'ga:conversions',
'dimensions': 'ga:date',
'samplingLevel': '1'
 }
 response = requests.get(url, params=params) return
response.json()

if    name    == '  main  ':
 conversion_id = '1234567890'
 conversion_data = get_conversion_data(conversion_id)
 print(conversion_data)
```

OUTPUT:

The output of the program will depend on the data in the data file. However, the output might include the following information:

- The conversion rate
- The number of conversions
- The number of visitors

B. Visitor Profiles

```
import requests

def get_visitor_profiles(profile_ids):
 url = 'https://analytics.google.com/analytics/v3/data/ga?'
 params = {
'ids': 'ga:{profile_ids}',
'start-date': '2023-01-01',
'end-date': '2023-08-01',
'metrics': 'ga:sessions,ga:bounceRate,ga:pageviews', 'dimensions':
'ga:source,ga:medium,ga:deviceCategory', 'samplingLevel': '1'
 }
 response = requests.get(url, params=params)
 return response.json()

if __name__ == '__main__':
 profile_ids = '1234567890,1234567891'
 visitor_profiles = get_visitor_profiles(profile_ids)
 print(visitor_profiles)
```

OUTPUT:

- ga:sessions: The number of sessions that the visitor had.
- ga:bounceRate: The bounce rate of the visitor.
- ga:pageviews: The number of pageviews that the visitor had.
- ga:source: The source of the visitor.
- ga:medium: The medium of the visitor.
- ga:deviceCategory: The device category of the visitor.

PROGRAM-6

Aim: Use Google analytics tools to implement the Traffic Sources.

SOURCE CODE:

```
import requests
def get_traffic_sources(profile_id):
url = 'https://analytics.google.com/analytics/v3/data/ga?'
params = {
'ids': 'ga:{profile_id}',
'start-date': '2023-01-01',
'end-date': '2023-08-01',
'metrics': 'ga:sessions',
'dimensions': 'ga:source,ga:medium', 'samplingLevel': '1'
 }
 response = requests.get(url, params=params)
 return response.json()

if __name__ == '__main__':
 profile_id = '1234567890'
 traffic_sources = get_traffic_sources(profile_id)
 print(traffic_sources)
```

OUTPUT:

- ga:sessions: The number of sessions from the traffic source.
- ga:source: The source of the traffic.
- ga:medium: The medium of the traffic.

```
{
"rows": [
  {
"ga:sessions": 100, "ga:source": "google", "ga:medium": "organic"
  },
  {
"ga:sessions": 50, "ga:source": "facebook", "ga:medium": "social"
  },
 { "ga:sessions": 20,
"ga:source": "twitter", "ga:medium": "social"
  },
 { "ga:sessions": 10,
"ga:source": "direct",
"ga:medium": "none"
  }
 ]
}
```