NAME – Mansi Dobriyal
Phone no - 9311453903
Email id – mansidobriyal50@gmail.com

Assignments: NLP Introduction and Text Preprocessing

1. What is the primary goal of Natural Language Processing (NLP)
   The primary goal of Natural Language Processing (NLP) is to enable computers to understand, interpret, generate, and respond to human language in a way that is both meaningful and useful. This involves tasks like language translation, sentiment analysis, text summarization, speech recognition, and chatbot development.
2. What does "tokenization" refer to in text processing
   In text processing, **tokenization** refers to the process of breaking down a piece of text into smaller units called **tokens**. Tokens can be words, phrases, sentences, or even characters, depending on the level of granularity required.
   For example:
   - Input: "Natural Language Processing is fascinating."
   - Tokenized (word-level): ["Natural", "Language", "Processing", "is", "fascinating", "."]

   It is a crucial step in NLP as it prepares the text for further analysis or processing.
3. What is the difference between lemmatization and stemming
   Lemmatization and **stemming** are text normalization techniques used to reduce words to their base or root form, but they differ in their approach and precision:
   1. Stemming:
   - **Definition**: Reduces words to their root form by chopping off prefixes or suffixes without considering the word's meaning.
   - **Method**: Uses heuristic rules (e.g., removing "ing", "ed", etc.).
   - **Output**: May produce non-meaningful roots.
   - **Example**:
     - Input: "running", "runner", "ran"
     - Output: "run", "runner", "ran"

   2. Lemmatization:
   - **Definition**: Reduces words to their base or dictionary form (lemma) based on their meaning and context.
   - **Method**: Uses vocabulary and morphological analysis.
   - **Output**: Produces meaningful root words.
   - **Example**:
     - Input: "running", "runner", "ran"
     - Output: "run", "run", "run"

   Key Difference: Lemmatization is more accurate as it considers the word's context, while stemming is faster but less precise.
4. What is the role of regular expressions (regex) in text processing
   1. **Primary goal of NLP**: Enable computers to understand and process human language.
   2. **Tokenization**: Breaking text into smaller units like words or sentences.
   3. **Lemmatization vs Stemming**: Lemmatization is precise and context-aware; stemming is faster but less accurate.
   4. **Regex in text processing**: Used for pattern matching, extraction, replacement, cleaning, and validation.
5. What is Word2Vec and how does it represent words in a vector space
   Word2Vec is an NLP model that represents words as vectors in a continuous vector space based on their context in a text. It uses two architectures: **CBOW** (predicts a word from its context) and **Skip-gram** (predicts context from a word).
   Words with similar meanings have closer vectors, capturing semantic relationships.
6. How does frequency distribution help in text analysis
   Frequency distribution counts how often each word or token appears in a text, helping identify key terms, trends, and patterns, such as the most common words or outliers.
7. Why is text normalization important in NLP

Text normalization ensures consistency in text data by converting it into a standard form, improving the accuracy of NLP tasks like tokenization, stemming, and lemmatization.

8. What is the difference between sentence tokenization and word tokenization
   - **Sentence Tokenization**: Splits text into sentences.
     Example: "Hello world. NLP is fun." → ["Hello world.", "NLP is fun."]
   - **Word Tokenization**: Splits text into words.
     Example: "Hello world." → ["Hello", "world", "."]
9. What are co-occurrence vectors in NLP

   Co-occurrence vectors represent words based on how often they appear near other words in a text, capturing contextual relationships.
10. What is the significance of lemmatization in improving NLP tasks

    Lemmatization improves NLP tasks by reducing words to their base form, ensuring consistency and enhancing understanding of word meaning, which helps in tasks like text classification, sentiment analysis, and information retrieval.
11. What is the primary use of word embeddings in NLP

    The primary use of word embeddings in NLP is to represent words as dense vectors in a continuous vector space, capturing semantic relationships and meanings, which improves tasks like language modeling, translation, and sentiment analysis.
12. What is an annotator in NLP

    An **annotator** in NLP is a tool or person that adds labels or metadata (such as part-of-speech tags, named entities, or sentiment) to text data to assist in training models or analyzing text.
13. What are the key steps in text processing before applying machine learning models

    Key steps in text processing before applying machine learning models include:
    1. **Text Cleaning**: Removing noise, special characters, and irrelevant data.
    2. **Tokenization**: Breaking text into words or sentences.
    3. **Lowercasing**: Converting all text to lowercase for consistency.
    4. **Removing Stopwords**: Eliminating common words (e.g., "and", "the") that don't add meaning.
    5. **Lemmatization/Stemming**: Reducing words to their base forms.
    6. **Vectorization**: Converting text into numerical format (e.g., TF-IDF, Word2Vec).
    7. **Feature Engineering**: Creating additional features, like word counts or sentence length.

    These steps prepare the text for effective machine learning model training.
14. What is the history of NLP and how has it evolved

    The history of **Natural Language Processing (NLP)** spans several decades and has evolved through various stages:
    1. **1950s-1960s**: Early efforts like **machine translation** (e.g., Georgetown-IBM experiment) focused on translating text between languages using rule-based systems.
    2. **1970s-1980s**: The development of **linguistic theories** and **parsing algorithms** such as **context-free grammars** aimed at understanding sentence structure.
    3. **1990s**: Shift to **statistical models** using large corpora, introducing methods like **Hidden Markov Models (HMMs)** and **n-grams** for tasks like part-of-speech tagging and speech recognition.
    4. **2000s**: Rise of **machine learning** algorithms for NLP, such as **Support Vector Machines (SVMs)**, and **maximum entropy models**, and the development of the **Word2Vec** embedding method in 2013.
    5. **2010s-Present**: The emergence of **deep learning** models, including **Recurrent Neural Networks (RNNs)**, **Long Short-Term Memory (LSTM)** networks, and the revolutionary **Transformer** models (e.g., **BERT**, **GPT**) that excel in language understanding and generation tasks.

    Overall, NLP has evolved from rule-based systems to machine learning and deep learning techniques, significantly improving the accuracy and capabilities of language understanding and generation.
15. Why is sentence processing important in NLP

    Sentence processing is important in NLP because it helps in understanding the structure and meaning of sentences, enabling tasks like **syntactic analysis**, **sentiment analysis**, and **machine translation**. It allows the model to capture relationships between words, handle ambiguity, and improve the accuracy of downstream tasks.

16. How do word embeddings improve the understanding of language semantics in NLP

Word embeddings improve the understanding of language semantics in NLP by representing words as dense vectors in a continuous vector space, where words with similar meanings are positioned closer together. This captures semantic relationships, such as synonyms and analogies, enabling models to understand context, similarity, and word associations, leading to better performance in tasks like translation, sentiment analysis, and question answering.

17. How does the frequency distribution of words help in text classification

    The frequency distribution of words helps in text classification by identifying the most common or important words in a document. These word frequencies can serve as features for machine learning models, aiding in distinguishing between different classes (e.g., spam vs. non-spam). Words that frequently appear in specific classes are often strong indicators for classification.

18. What are the advantages of using regex in text cleaning

    Advantages of using **regex** in text cleaning include:

    1. **Flexibility**: Can match complex patterns like email addresses, URLs, or special characters.
    2. **Efficiency**: Allows quick search, replace, and extraction operations on large text data.
    3. **Precision**: Enables fine control over the types of characters or words to remove or retain.
    4. **Automation**: Reduces manual intervention by automating repetitive text-cleaning tasks.
    5. **Versatility**: Can handle various text formats and structures across different domains.

19. What is the difference between word2vec and doc2vec

    The key difference between **Word2Vec** and **Doc2Vec** lies in the level of representation:

    1. **Word2Vec**:
        - **Purpose**: Represents individual words as vectors.
        - **Output**: Each word in the vocabulary is mapped to a vector that captures its semantic meaning based on context.
        - **Usage**: Primarily used for tasks where word-level representation is needed (e.g., word similarity, analogy tasks).
    2. **Doc2Vec**:
        - **Purpose**: Extends Word2Vec to represent entire documents (or sentences) as vectors.
        - **Output**: Each document or sentence is represented by a unique vector, capturing the overall semantic meaning of the text.
        - **Usage**: Useful for document-level tasks like text classification, sentiment analysis, and information retrieval.

    In essence, **Word2Vec** focuses on words, while **Doc2Vec** focuses on representing larger text units like documents.

20. Why is understanding text normalization important in NLP

    Understanding **text normalization** is important in NLP because it ensures consistency and prepares raw text for analysis by reducing variations (like different spellings or case) into a standard form. This improves the accuracy of tasks such as tokenization, lemmatization, and machine learning, making it easier for models to process and understand the text.

21. How does word count help in text analysis

    Word count helps in text analysis by providing a simple measure of text length, helping identify key features like document size, word frequency, and text complexity. It can be used to:

    1. **Measure content length**: Analyze whether a document is too short or too long.
    2. **Feature extraction**: For machine learning models, word count can serve as a feature to classify or categorize texts.
    3. **Frequency analysis**: Identifying frequently occurring words, aiding in tasks like topic modeling or keyword extraction.

22. How does lemmatization help in NLP tasks like search engines and chatbots

    Lemmatization helps in NLP tasks like search engines and chatbots by reducing words to their base or dictionary form, improving consistency and accuracy in understanding user queries.

    1. **Search Engines**: It allows the search engine to treat different forms of a word (e.g., "running" and "run") as the same, improving the relevance of search results.
    2. **Chatbots**: Lemmatization helps the chatbot understand variations of a word (e.g., "went" vs. "go") and respond appropriately, improving user interaction and comprehension.

    In both cases, lemmatization enhances matching and context understanding.

23. What is the purpose of using Doc2Vec in text processing

    The purpose of using **Doc2Vec** in text processing is to represent entire documents (or sentences) as fixed-length vectors, capturing the semantic meaning of the text. This helps in tasks such as:

    1. **Document Classification**: Assigning labels to entire documents based on their content.
    2. **Similarity Analysis**: Comparing the semantic similarity between different documents.
    3. **Information Retrieval**: Improving search results by finding documents that are semantically similar to a query.

    Unlike word embeddings (Word2Vec), which represent individual words, Doc2Vec captures the broader context of an entire document.

24. What is the importance of sentence processing in NLP

    Sentence processing is important in NLP because it helps understand the structure and meaning of sentences. It enables models to:

    1. **Disambiguate Meaning**: Resolve ambiguity by understanding sentence context (e.g., "bank" as a financial institution vs. riverbank).
    2. **Syntactic Analysis**: Analyze sentence structure, aiding in tasks like parsing and grammatical analysis.
    3. **Contextual Understanding**: Capture the relationships between words, improving tasks like sentiment analysis, machine translation, and question answering.

    Sentence processing ensures that NLP models interpret and generate text accurately and contextually.

25. What is text normalization, and what are the common techniques used in it

    Text normalization is the process of converting text into a consistent, standard form to make it easier for NLP tasks. Common techniques used in text normalization include:

    1. **Lowercasing**: Converting all text to lowercase to avoid distinguishing between the same word in different cases (e.g., "Apple" and "apple").
    2. **Removing Punctuation**: Eliminating punctuation marks to focus on the actual content (e.g., commas, periods).
    3. **Removing Stopwords**: Eliminating common words (e.g., "the", "is") that don't contribute significant meaning.
    4. **Stemming**: Reducing words to their root form (e.g., "running" to "run").
    5. **Lemmatization**: Converting words to their base form using a dictionary (e.g., "better" to "good").
    6. **Expanding Contractions**: Replacing contractions with their full forms (e.g., "don't" to "do not").
    7. **Removing Numbers or Special Characters**: Depending on the task, removing non-alphabetic characters may be necessary.

    These techniques help prepare text for more accurate analysis and modeling in NLP.

26. Why is word tokenization important in NLP

    Word tokenization is important in NLP because it breaks down text into individual words, enabling models to process and analyze language at a granular level. It helps in:

    1. **Text Preprocessing**: Essential for further steps like stopword removal, stemming, or lemmatization.
    2. **Feature Extraction**: Enables the creation of features for machine learning models (e.g., word frequency).
    3. **Improved Analysis**: Allows the model to focus on individual words and their relationships, improving tasks like sentiment analysis, translation, and topic modeling.

    In essence, tokenization forms the foundation for most NLP tasks.

27. How does sentence tokenization differ from word tokenization in NLP

    Sentence tokenization and **word tokenization** differ in the level at which they break down text:

    1. **Sentence Tokenization**:
        - Splits text into **sentences**.
        - Focuses on sentence boundaries (e.g., period, question mark).
        - Example: "I love NLP. It's amazing." → ["I love NLP.", "It's amazing."]
    2. **Word Tokenization**:
        - Splits text into **words**.
        - Focuses on separating words by spaces and punctuation.
        - Example: "I love NLP." → ["I", "love", "NLP", "."]

    Key Difference: Sentence tokenization breaks text into sentences, while word tokenization breaks text into individual words.

28. What is the primary purpose of text processing in NLP

The primary purpose of **text processing** in NLP is to clean, transform, and prepare raw text data for analysis, making it easier for models to understand and extract meaningful insights. It involves steps like tokenization, normalization, and vectorization to convert text into a format suitable for machine learning and other NLP tasks.

29. What are the key challenges in NLP

    Key challenges in NLP include:

    1. **Ambiguity**: Words or sentences with multiple meanings depending on context (e.g., "bank" can mean a financial institution or a riverbank).
    2. **Context Understanding**: Grasping the intended meaning of words based on surrounding text or conversation.
    3. **Sarcasm and Irony**: Identifying the true meaning when the text conveys the opposite of the literal meaning.
    4. **Out-of-Vocabulary Words**: Dealing with new or rare words not seen in training data.
    5. **Language Variability**: Handling different dialects, slang, and informal language.
    6. **Named Entity Recognition (NER)**: Correctly identifying and classifying entities like names, dates, or locations.
    7. **Multilingual Challenges**: Managing translation and understanding across different languages with unique structures and rules.

30. How do co-occurrence vectors represent relationships between words

    Co-occurrence vectors represent relationships between words by capturing how often two words appear near each other within a specified context (e.g., a window of surrounding words). The frequency of co-occurrence is stored in a vector, where each element corresponds to a specific word in the vocabulary.

    How it works:

    - Words that frequently appear together in similar contexts will have similar co-occurrence vectors, reflecting semantic relationships.
    - For example, the words "cat" and "dog" might co-occur frequently in similar contexts, so their co-occurrence vectors will be close in the vector space, indicating they are related.

    These vectors help capture semantic similarity, analogies, and associations between words, useful in tasks like word similarity, clustering, and recommendation systems.

31. What is the role of frequency distribution in text analysis

    The **frequency distribution** in text analysis plays a key role by counting how often each word or token appears in a text. It helps in:

    1. **Identifying Important Words**: Highlights frequently occurring words, which can indicate key topics or themes.
    2. **Feature Extraction**: Provides features for machine learning models, such as word counts or term frequencies (TF).
    3. **Text Classification**: Helps distinguish between classes by analyzing the frequency of specific words associated with each class.
    4. **Pattern Recognition**: Detects patterns and trends in text, useful for tasks like topic modeling and sentiment analysis.

    In summary, frequency distribution aids in summarizing text data, feature generation, and improving model performance in NLP tasks.

32. What is the impact of word embeddings on NLP tasks

    Word embeddings significantly impact NLP tasks by transforming words into dense vectors that capture semantic relationships. Their key impacts include:

    1. **Improved Semantic Understanding**: Words with similar meanings are placed closer together in the vector space, enhancing models' ability to understand context and word similarity.
    2. **Better Generalization**: Embeddings allow models to generalize across synonyms and related terms, improving performance in tasks like sentiment analysis, translation, and classification.
    3. **Dimensionality Reduction**: Word embeddings reduce the high dimensionality of one-hot encoding, making models more efficient and scalable.
    4. **Enhanced Performance**: They improve accuracy and effectiveness in tasks like named entity recognition (NER), machine translation, and question answering.

    Overall, word embeddings enable more powerful, context-aware models, making them a foundational component of modern NLP.

33. What is the purpose of using lemmatization in text preprocessing?

The purpose of using **lemmatization** in text preprocessing is to reduce words to their base or root form (called the "lemma"), ensuring consistency and improving the accuracy of NLP tasks. This helps by:

1. **Grouping Word Variants**: Converts different forms of a word (e.g., "running," "ran," "runs") to a single base form ("run"), allowing the model to treat them as the same word.
2. **Improving Meaning Understanding**: It helps models understand the underlying meaning of words, improving tasks like text classification, sentiment analysis, and information retrieval.
3. **Reducing Noise**: Removes irregularities in word forms, leading to cleaner and more consistent data for analysis.

Lemmatization provides more accurate and meaningful word representations compared to simpler techniques like stemming.

Practical

1. How can you perform word tokenization using NLTK

```
import nltk

nltk.download('punkt')

sentence = "Hello, how are you doing today?"
words = nltk.word_tokenize(sentence)

print(words)
```

2. How can you perform sentence tokenization using NLTK:

```
import nltk
nltk.download('punkt')  # Download required resources

text = "Hello, how are you? I hope you're doing well. Have a great day!"
sentences = nltk.sent_tokenize(text)

print(sentences)
```

3. How can you remove stopwords from a sentence

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')  # Download stopwords

# Sample sentence
sentence = "This is an example sentence showing off the stopwords removal."

# Tokenize the sentence
words = nltk.word_tokenize(sentence)

# Get the list of stopwords
stop_words = set(stopwords.words('english'))

# Remove stopwords from the tokenized words
filtered_words = [word for word in words if word.lower() not in stop_words]

print(filtered_words)
```

4. How can you perform stemming on a word

```
import nltk
from nltk.stem import PorterStemmer
nltk.download('punkt')  # Download required resources

# Initialize the stemmer
stemmer = PorterStemmer()

# Sample word
word = "running"

# Perform stemming
stemmed_word = stemmer.stem(word) : How can you perform lemmatization on a word
print(stemmed_word)
```

5. How can you perform lemmatization on a word

```
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')  # Download WordNet resource
nltk.download('omw-1.4')  # Download WordNet lexical resource

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Sample word
word = "running"

# Perform lemmatization
lemmatized_word = lemmatizer.lemmatize(word, pos='v')  # 'v' for verb

print(lemmatized_word)
```

6. How can you normalize a text by converting it to lowercase and removing punctuation

```
import nltk
import string
nltk.download('punkt')  # Download required resources

# Sample text
text = "Hello, How are you? Welcome to NLP."

# Convert text to lowercase
text_lower = text.lower()

# Remove punctuation
text_normalized = ''.join([char for char in text_lower if char not in string.punctuation])

print(text_normalized)
```

7. How can you create a co-occurrence matrix for words in a corpus

```
import nltk
import numpy as np
from nltk import FreqDist
from nltk.util import ngrams
```

```
from collections import defaultdict
nltk.download('punkt')  # Download required resources

# Sample corpus (list of sentences)
corpus = [
    "I love programming in Python",
    "Python is great for data science",
    "I enjoy learning new programming languages"
]

# Tokenize the corpus
tokenized_corpus = [nltk.word_tokenize(sentence.lower()) for sentence in corpus]

# Define a function to create a co-occurrence matrix
def create_co_occurrence_matrix(corpus, window_size=2):
    co_occurrence_matrix = defaultdict(lambda: defaultdict(int))

    for sentence in corpus:
        for i, word in enumerate(sentence):
            # Consider a window of words around the current word
            for j in range(i + 1, i + window_size + 1):
                if j < len(sentence):
                    co_occurrence_matrix[word][sentence[j]] += 1
                    co_occurrence_matrix[sentence[j]][word] += 1

    return co_occurrence_matrix

# Create the co-occurrence matrix
co_matrix = create_co_occurrence_matrix(tokenized_corpus)

# Convert to a more readable format (list of tuples)
co_occurrence_list = [(key, inner_key, count) for key, inner_dict in co_matrix.items() for
inner_key, count in inner_dict.items()]

# Print the co-occurrence list
for entry in co_occurrence_list:
    print(entry)
```

Explanation:

- The function create_co_occurrence_matrix takes a corpus and a window size as input, and calculates how often each pair of words occurs together within the defined window.
- The output is a list of co-occurring word pairs and their counts.

8. How can you apply a regular expression to extract all email addresses from a text

```
import re

# Sample text
text = "You can reach out to john.doe@example.com or jane.smith123@mydomain.org for more
information."

# Regular expression pattern for matching email addresses
email_pattern = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

# Find all email addresses in the text
```

```
email_addresses = re.findall(email_pattern, text)

print(email_addresses)
```

Explanation:

- The regular expression pattern `r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'` matches valid email formats.
  - `[a-zA-Z0-9._%+-]+` matches the username part of the email.
  - `@[a-zA-Z0-9.-]+` matches the domain name.
  - `\.[a-zA-Z]{2,}` matches the top-level domain (e.g., `.com`).
- `re.findall()` is used to find all occurrences of the email pattern in the text.

9. How can you perform word embedding using Word2Vec

```
import nltk
from gensim.models import Word2Vec
nltk.download('punkt')  # Download required resources

# Sample corpus (list of sentences)
corpus = [
    "I love programming in Python",
    "Python is great for data science",
    "I enjoy learning new programming languages"
]

# Tokenize the corpus
tokenized_corpus = [nltk.word_tokenize(sentence.lower()) for sentence in corpus]

# Create and train a Word2Vec model
model = Word2Vec(tokenized_corpus, vector_size=50, window=3, min_count=1, workers=4)

# Get the vector for a word
word_vector = model.wv['python']

print(word_vector)
```

10. How can you use Doc2Vec to embed documents

```
import nltk
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
nltk.download('punkt')  # Download required resources

# Sample corpus (list of sentences or documents)
corpus = [
    "I love programming in Python",
    "Python is great for data science",
    "I enjoy learning new programming languages"
]

# Tokenize the corpus and tag each document
```

```
tagged_corpus = [TaggedDocument(words=nltk.word_tokenize(doc.lower()), tags=[str(i)]) for
i, doc in enumerate(corpus)]

# Create and train a Doc2Vec model
model = Doc2Vec(vector_size=50, window=3, min_count=1, workers=4)
model.build_vocab(tagged_corpus)
model.train(tagged_corpus, total_examples=model.corpus_count, epochs=10)

# Get the vector for a specific document
doc_vector = model.dv['0']  # '0' is the tag for the first document

print(doc_vector)
```

11. How can you perform part-of-speech tagging

```
import nltk
nltk.download('punkt')  # Download required resources
nltk.download('averaged_perceptron_tagger')  # Download POS tagger

# Sample text
text = "NLTK is a leading platform for building Python programs to work with human
language data."

# Tokenize the text
tokens = nltk.word_tokenize(text)

# Perform POS tagging
pos_tags = nltk.pos_tag(tokens)

print(pos_tags)
```

12. How can you find the similarity between two sentences using cosine similarity

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Sample sentences
sentence1 = "I love programming in Python"
sentence2 = "Python programming is great for data science"

# Initialize the TfidfVectorizer
vectorizer = TfidfVectorizer()

# Combine the sentences into a list
sentences = [sentence1, sentence2]

# Transform the sentences into TF-IDF vectors
tfidf_matrix = vectorizer.fit_transform(sentences)

# Calculate cosine similarity
cosine_sim = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])

print(cosine_sim[0][0])
```

Explanation:

1. **TF-IDF Vectorization**: TfidfVectorizer converts the sentences into vector representations, considering the importance of each word in the context of the entire corpus.
2. **Cosine Similarity**: cosine_similarity calculates the cosine of the angle between the vectors of the two sentences, which measures their similarity. A value closer to 1 indicates high similarity, and 0 indicates no similarity.

Output:

The output will be a value between 0 and 1, where a higher value indicates more similarity between the sentences.

13. How can you extract named entities from a sentence

```
import nltk
nltk.download('punkt')  # Download required resources
nltk.download('maxent_ne_chunker')  # Named entity chunker
nltk.download('words')  # Words corpus

# Sample sentence
sentence = "Barack Obama was born in Hawaii and was the 44th President of the United
States."

# Tokenize the sentence and perform POS tagging
tokens = nltk.word_tokenize(sentence)
pos_tags = nltk.pos_tag(tokens)

# Perform Named Entity Recognition (NER)
named_entities = nltk.chunk.ne_chunk(pos_tags)

# Extract named entities
named_entities_list = []
for chunk in named_entities:
    if isinstance(chunk, nltk.Tree):  # Check if it's a named entity
        entity = " ".join([word for word, tag in chunk])
        named_entities_list.append(entity)

print(named_entities_list)
```

14. How can you split a large document into smaller chunks of text

```
# Sample large document
document = """
Natural language processing (NLP) is a subfield of artificial intelligence (AI) focused on
the interaction between computers and humans through natural language. It involves tasks
such as
language modeling, machine translation, and sentiment analysis. NLP combines linguistics
and computer science
to enable computers to process and analyze human language data.
"""

# Define the chunk size (number of characters per chunk)
chunk_size = 100  # You can adjust the chunk size as needed

# Split the document into smaller chunks
chunks = [document[i:i+chunk_size] for i in range(0, len(document), chunk_size)]
```

```
# Print the smaller chunks
for chunk in chunks:
    print(chunk)
    print("-----")
```

Explanation:

1. **Document**: A sample large document is provided.
2. **Chunk Size**: The document is split into smaller chunks based on the defined chunk size (100 characters in this case). You can adjust the chunk_size as needed.
3. **Splitting Logic**: The document is split by iterating over it in steps of chunk_size and creating a list of text chunks.
4. **Printing Chunks**: Each chunk is printed with a separator.

Sample Output:

```
Natural language processing (NLP) is a subfield of artificial intelligenc
e (AI) focused on the interaction between computers and humans through n
atural language. It involves tasks such as
-----
language modeling, machine translation, and sentiment analysis. NLP comb
ines linguistics and computer science
-----
to enable computers to process and analyze human language data.
-----
```

15. How can you calculate the TF-IDF (Term Frequency - Inverse Document Frequency) for a set of documents

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample set of documents
documents = [
    "I love programming in Python",
    "Python is great for data science",
    "I enjoy learning new programming languages"
]

# Initialize the TfidfVectorizer
vectorizer = TfidfVectorizer()

# Fit and transform the documents to calculate the TF-IDF values
tfidf_matrix = vectorizer.fit_transform(documents)

# Convert the TF-IDF matrix to an array and display it
tfidf_array = tfidf_matrix.toarray()

# Get the feature names (words)
feature_names = vectorizer.get_feature_names_out()

# Print the TF-IDF values for each word in each document
for i, doc in enumerate(tfidf_array):
    print(f"Document {i+1}:")
    for j, score in enumerate(doc):
        print(f"{feature_names[j]}: {score:.4f}")
    print("-----")
```

Explanation:

1. **Documents**: A set of documents is provided.
2. **TfidfVectorizer**: TfidfVectorizer is used to compute the TF-IDF scores for the words in the documents.
    - **Fit**: It learns the vocabulary from the documents.
    - **Transform**: It converts the documents into TF-IDF feature vectors.
3. **TF-IDF Matrix**: The result is a sparse matrix of TF-IDF values, which is converted to a dense array for easier inspection.
4. **Feature Names**: The feature names represent the words corresponding to each column in the TF-IDF matrix.

Sample Output:

yaml

Copy code

```
Document 1:
i: 0.4472
in: 0.4472
love: 0.4472
programming: 0.4472
python: 0.4472
-----
Document 2:
data: 0.4472
for: 0.4472
is: 0.4472
python: 0.4472
science: 0.4472
-----
Document 3:
enjoy: 0.4472
languages: 0.4472
learning: 0.4472
new: 0.4472
programming: 0.4472
-----
```

16. How can you apply tokenization, stopword removal, and stemming in one go

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
nltk.download('punkt')  # Download required resources
nltk.download('stopwords')  # Download stopwords list

# Sample sentence
sentence = "I love programming in Python. It's great for data science!"

# Step 1: Tokenization
tokens = word_tokenize(sentence.lower())

# Step 2: Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]

# Step 3: Apply stemming
```

```
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]

print(stemmed_tokens)
```

Explanation:

1. **Tokenization**: The sentence is tokenized into words using word_tokenize.
2. **Stopword Removal**: The stopwords (common words like "and", "is", etc.) are removed using stopwords.words('english') from the NLTK corpus.
3. **Stemming**: The Porter stemmer is applied to reduce words to their root form (e.g., "programming" becomes "program").

Sample Output:

```
['love', 'program', 'python', 'great', 'data', 'scienc']
```

17. How can you visualize the frequency distribution of words in a sentence?

```
import nltk
import matplotlib.pyplot as plt
from nltk.probability import FreqDist
from nltk.tokenize import word_tokenize
nltk.download('punkt')  # Download required resources

# Sample sentence
sentence = "I love programming in Python. Python is great for data science and
programming."

# Tokenize the sentence
tokens = word_tokenize(sentence.lower())

# Calculate the frequency distribution
fdist = FreqDist(tokens)

# Plot the frequency distribution
fdist.plot(title="Word Frequency Distribution", xlabel="Words", ylabel="Frequency")
plt.show()
```

Explanation:

1. **Tokenization**: The sentence is tokenized into words using `word_tokenize`.
2. **Frequency Distribution**: The `FreqDist` function from NLTK calculates the frequency of each token in the sentence.
3. **Visualization**: The `plot` method of `FreqDist` uses Matplotlib to visualize the frequency distribution as a bar plot, showing how often each word appears.

Sample Output:

The output will be a bar chart showing the frequency of each word in the sentence. Words like "python" and "programming" will appear more frequently than others.

This allows you to visualize the most common words in the sentence.

THE END.....