Q1: What is tokenization in NLP?

Answer: Tokenization is the process of splitting text into smaller units, such as words or sentences, which makes it easier to process and analyze the text. Word tokenization breaks text into individual words, while sentence tokenization splits text into individual sentences.

Q2: Why is stop word removal important in text preprocessing?

Answer: Stop word removal is essential because common words like "the", "is", and "in" do not carry significant meaning in text analysis and can add unnecessary noise to models. Removing them helps improve the performance of machine learning algorithms.

Q3: What is stemming, and how does it differ from lemmatization?

Answer: Stemming is a process that reduces words to their root form by removing prefixes and suffixes. However, stemming can produce non-existent words (e.g., "running" → "run"). Lemmatization, on the other hand, converts words to their dictionary form (e.g., "running" → "run"), ensuring that the results are valid words.

Q4: What is POS tagging, and why is it used?

Answer: Part-of-speech (POS) tagging involves assigning each word in a sentence a tag that represents its part of speech (e.g., noun, verb, adjective). It helps in understanding the grammatical structure of the sentence and is important for tasks like text parsing and syntactic analysis.

Q5: What is TF-IDF and how is it calculated?

Answer: TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents. It is calculated by multiplying the Term Frequency (TF) of a word in a document by the Inverse Document Frequency (IDF) of the word across the entire corpus.

Q6: Why is IDF used in the TF-IDF formula?

Answer: IDF is used to reduce the weight of common words that appear in many documents and increase the weight of words that are rare across the corpus. This makes words that appear in fewer documents more important, which can help improve the relevance of features in models.

Q7: What is the difference between stemming and lemmatization?

Answer: Stemming is a simpler process that removes prefixes and suffixes to reduce words to their root form, which might not always be valid words. Lemmatization, on the other hand, uses linguistic knowledge to convert words into their valid dictionary form based on their meaning.

Q8: How does TF-IDF improve document representation in NLP?

Answer: TF-IDF improves document representation by providing a more accurate measure of word importance. Words that are frequent in a document but rare in the corpus are given higher importance, while common words across many documents are down-weighted. This allows for a better distinction of important terms in the document.

Q9: What is the role of `TfidfVectorizer` in the code?

Answer: `TfidfVectorizer` is used to convert a collection of text documents into a matrix of TF-IDF features. It computes the TF and IDF scores for each word in the documents and transforms the text data into a numerical representation that can be used for machine learning.

Q10: Why do we need to remove stopwords in text processing?

Answer: Removing stopwords helps reduce the dimensionality of the data and eliminates words that do not contribute meaningful information to text analysis. This enhances the performance of machine learning models and allows them to focus on more relevant terms.

Q11: How is tokenization done in the provided code?

Answer: In the provided code, tokenization is done using NLTK's `word_tokenize()` and `sent_tokenize()`. `word_tokenize()` breaks text into individual words, while `sent_tokenize()` splits the text into sentences.

Q12: What does the `FreqDist` class do in the code?

Answer: `FreqDist` is a class in NLTK that calculates the frequency distribution of words in a given text. It counts how often each word appears in the text, helping in the analysis of word frequency and importance.

Q13: What is the advantage of using lemmatization over stemming?

Answer: Lemmatization provides more accurate results than stemming because it converts words to their actual base form, ensuring that the output is a valid word. It considers the context and meaning of the word, making it more linguistically correct than stemming.

Q14: What does the function `word_tokenize()` do in the code?

Answer: `word_tokenize()` from the NLTK library splits the input text into individual words (tokens). This process helps in analyzing each word separately for further text processing tasks like stopword removal, stemming, and lemmatization.

Q15: Why is the `PorterStemmer` used in the code?

Answer: The `PorterStemmer` is used in the code to perform stemming. It reduces words to their root form by removing common suffixes. For example, "running" becomes "run," which helps in reducing word complexity for analysis.

Q16: What is the significance of POS tagging in NLP?

Answer: POS tagging helps in understanding the grammatical role of each word in a sentence. It provides information about whether a word is a noun, verb, adjective, etc., which is useful for syntactic parsing, named entity recognition, and other advanced NLP tasks.

Q17: What do the terms `tfidf_matrix` and `vectorizer.get_feature_names_out()` represent?

Answer: The `tfidf_matrix` represents the TF-IDF values for each word in the documents, while `vectorizer.get_feature_names_out()` provides the list of unique words (features) in the corpus that are used to construct the matrix.

Q18: How does TF-IDF help in identifying important words in a document?

Answer: TF-IDF helps by giving higher importance to words that appear frequently in a document but less frequently in other documents. This helps in identifying terms that are more specific and significant to a particular document or topic.

Q19: Why is the `nltk.download()` function used in the code?

Answer: The `nltk.download()` function is used to download necessary NLTK datasets and resources, such as the stopwords, punkt tokenizer models, and POS tagger, that are required for various NLP tasks like tokenization, stop word removal, and POS tagging.

Q20: What is the role of `ConfusionMatrixDisplay` in the code?

Answer: `ConfusionMatrixDisplay` is used to visually display the confusion matrix, which shows the performance of the classification model by comparing the predicted and actual values. It helps in understanding the model's accuracy and error rates.

Q21: How does `word_tokenize()` differ from `sent_tokenize()`?

Answer: `word_tokenize()` splits a text into individual words, while `sent_tokenize()` splits the text into sentences. Both are used for breaking down text, but at different levels of granularity.

Q22: What is the output of the `fdist.plot()` function in the code?

Answer: The `fdist.plot()` function plots the frequency distribution of words in the text, showing how often each word appears. This helps in visualizing the most common words in the document.

Q23: How do we calculate Term Frequency (TF)?

Answer: Term Frequency (TF) is calculated by dividing the number of times a word appears in a document by the total number of words in that document. It gives an indication of the importance of a word in the document.

Q24: What is the importance of downloading corpora like `stopwords` and `punkt`?

Answer: Downloading corpora like `stopwords` and `punkt` ensures that the necessary resources for tokenization and stop word removal are available. These corpora contain predefined lists of stop words and tokenization rules needed for text preprocessing.

Q25: How does the `TfidfVectorizer` transform text into a numerical format?

Answer: The `TfidfVectorizer` transforms text into a numerical format by calculating the TF-IDF score for each word in the document. This results in a sparse matrix where each row represents a document, and each column represents a word in the vocabulary, with values corresponding to the TF-IDF score.