1. What are Corpora?

Answer: A corpus (plural: corpora) is a collection of texts or documents that are used as a dataset for linguistic analysis, natural language processing (NLP), or other text-based research. Corpora serve as valuable resources for studying language patterns, extracting linguistic features, and developing and evaluating language models and algorithms.

Here are some key characteristics and types of corpora:

1. **Textual Data**: Corpora consist of textual data, which can include written texts, spoken transcripts, social media posts, emails, web pages, news articles, books, and any other form of language data.

2. **Size and Scope**: Corpora can vary greatly in size and scope, ranging from small, specialized collections to large-scale, general-purpose datasets. The size and scope of a corpus depend on the specific research goals and the availability of data.

3. **Annotation**: Some corpora are annotated with linguistic information such as part-of-speech tags, syntactic structures, named entities, sentiment labels, or semantic annotations. Annotation adds value to the corpus by providing additional information for analysis and modeling.

4. **Balanced vs. Representative**: Corpora can be balanced, meaning they represent a diverse range of linguistic features, genres, styles, and contexts. Alternatively, they can be specialized or domain-specific, focusing on a particular language variety, genre, domain, or time period.

5. **Availability**: Corpora may be publicly available or proprietary, depending on the data sources and usage restrictions. Publicly available corpora are often used in academic research and shared among researchers, while proprietary corpora may be owned by organizations or companies and used for proprietary purposes.

6. **Anonymization and Privacy**: In some cases, corpora may be anonymized to protect the privacy of individuals whose data is included in the corpus, especially in the case of spoken or written personal communications.

2. What are Tokens?

Answer: In the context of natural language processing (NLP), tokens are the individual units or elements that make up a piece of text. These units can vary depending on the level of granularity or abstraction considered. Here are some common types of tokens:

1. **Words**: In many NLP tasks, tokens correspond to individual words in a text. Words are typically delimited by whitespace or punctuation marks. For example, in the sentence "The quick brown fox jumps over the lazy dog," the tokens are: ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"].

2. **Subwords**: Subword tokens are smaller units of text that are derived from words by breaking them down into smaller components. This is particularly useful for languages with complex morphology or for handling out-of-vocabulary words. Subword tokenization techniques include Byte Pair Encoding (BPE), WordPiece, and SentencePiece.

3. **Characters**: In character-level tokenization, each character in the text is considered a token. This level of granularity is useful for tasks where character-level information is important, such as text generation, spelling correction, and language modeling.

4. Phrases: In some cases, tokens may represent multi-word phrases or chunks of text rather than individual words. This can be useful for tasks such as named entity recognition, chunking, and semantic analysis.

Tokenization is an essential preprocessing step in NLP tasks, as it breaks down raw text into manageable units that can be further processed, analyzed, and used as input to machine learning models. The choice of tokenization strategy depends on the specific task, language, and characteristics of the text data being analyzed.

3. What are Unigrams, Bigrams, Trigrams?

Answer: Unigrams, bigrams, and trigrams are types of n-grams, which are contiguous sequences of n items (usually words) in a piece of text. These n-grams are used in natural language processing (NLP) for various tasks such as language modeling, text generation, and feature extraction. Here's a brief explanation of each:

1. Unigrams: Unigrams are single words considered as individual tokens in a text. Each word in the text is treated as a separate unigram. For example, in the sentence "The quick brown fox jumps over the lazy dog," the unigrams are: ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"].

2. Bigrams: Bigrams are sequences of two adjacent words in a text. Each bigram consists of two consecutive words in the text. For example, in the same sentence, the bigrams would be: [("The", "quick"), ("quick", "brown"), ("brown", "fox"), ("fox", "jumps"), ("jumps", "over"), ("over", "the"), ("the", "lazy"), ("lazy", "dog")]. Bigrams capture some syntactic and semantic information about the text and are often used in tasks like text classification, sentiment analysis, and machine translation.

3. Trigrams: Trigrams are sequences of three adjacent words in a text. Each trigram consists of three consecutive words in the text. For example, in the same sentence, the trigrams would be: [("The", "quick", "brown"), ("quick", "brown", "fox"), ("brown", "fox", "jumps"), ("fox", "jumps", "over"), ("jumps", "over", "the"), ("over", "the", "lazy"), ("the", "lazy", "dog")]. Trigrams provide more context than bigrams and are often used in tasks like language modeling, named entity recognition, and part-of-speech tagging.

N-grams of higher orders, such as 4-grams (four-word sequences) or higher, can also be used, but they become less common as the size of the n-gram increases due to the increased sparsity of longer sequences in text data. The choice of n-gram size depends on the specific task, the characteristics of the text data, and the desired level of context or granularity.

4. How to generate n-grams from text?

Answer: Generating n-grams from text involves breaking down the text into sequences of n contiguous items, typically words or characters. Here's a general approach to generate n-grams from text:

1. Tokenization: First, tokenize the text into individual units such as words or characters. This involves splitting the text into its constituent elements, usually based on whitespace for word-level tokenization or individual characters for character-level tokenization.

2. Constructing n-grams: Iterate through the tokenized text to create n-grams. For each position in the text, create a sequence of n tokens starting from that position. For example:
    - For unigrams (1-grams), each token is considered as a separate n-gram.
    - For bigrams (2-grams), create pairs of consecutive tokens.

- For trigrams (3-grams), create triples of consecutive tokens.
- For higher-order n-grams, create sequences of n consecutive tokens.

3. Handling Boundary Cases: Depending on the desired behavior, you may need to handle boundary cases where the sequence of tokens spans the beginning or end of the text. Options include:

    - Padding the text with special tokens to ensure that n-grams can be generated consistently across all positions.
    - Discarding n-grams that overlap with the beginning or end of the text if partial sequences are not desired.

Here's a Python code example to generate n-grams from a given text using the NLTK library:

```
import nltk
from nltk.util import ngrams

def generate_ngrams(text, n):
    # Tokenize the text
    tokens = nltk.word_tokenize(text)
    # Generate n-grams
    n_grams = list(ngrams(tokens, n))
    return n_grams

# Example usage
text = "This is a sample text for generating n-grams."
n = 2  # Generating bigrams
result = generate_ngrams(text, n)
print(result)
```

5. Explain Lemmatization.

Answer: Lemmatization is the process of reducing words to their base or canonical form, known as the lemma, which represents the dictionary form of a word. The goal of lemmatization is to normalize words so that different inflected forms of the same word are treated as the same word.

Here's how lemmatization works:

1. Word Analysis: Lemmatization takes into account the morphological analysis of words, considering factors such as part-of-speech (POS) tags, grammatical features, and context to determine the base form of a word.

2. Lookup in Lexicon: Lemmatization typically involves using a lexicon or dictionary, known as a lemma list or lemmatization dictionary, which contains mappings of words to their corresponding lemmas. These mappings are based on the rules of the language and may include information about irregular forms, conjugations, and derivations.

3. Normalization: For each word in the text, the lemmatization process looks up the word in the lemma list to find its base form or lemma. If the word is found in the dictionary, its lemma is returned. If not, the word may be left unchanged or mapped to a default lemma based on its POS tag or context.

4. **Example**: For example, the word "running" has the lemma "run," and the word "mice" has the lemma "mouse." By lemmatizing these words, both "running" and "runs" would be reduced to "run," and "mice" and "mouses" would be reduced to "mouse."

Lemmatization is particularly useful in natural language processing (NLP) tasks where the semantic meaning of words is important, such as information retrieval, document classification, sentiment analysis, and machine translation. By reducing words to their base forms, lemmatization helps to improve the accuracy and effectiveness of these tasks by reducing the vocabulary size and capturing the inherent meaning of words. Popular NLP libraries such as NLTK (Natural Language Toolkit), spaCy, and Stanford CoreNLP provide lemmatization functionality for various languages.

6. Explain Stemming.

Answer: Stemming is the process of reducing words to their base or root form, known as the stem, by removing affixes such as prefixes and suffixes. The goal of stemming is to normalize words so that different inflected forms of the same word are treated as the same word.

Here's how stemming works:

1. **Rule-Based Process**: Stemming typically involves applying a set of rules or algorithms to strip away affixes from words. These rules are based on linguistic heuristics and patterns and may vary depending on the language and the stemming algorithm used.

2. **Suffix Stripping**: In most stemming algorithms, suffixes are removed from words to generate the stem. For example, common English suffixes such as "-ing," "-ed," "-s," and "-es" are stripped away to reduce words to their base form.

3. **Prefix Stripping**: Some stemming algorithms also remove prefixes from words, although this step is less common than suffix stripping. Prefixes such as "un-" or "re-" may be removed to generate the stem.

4. **Example**: For example, applying stemming to the words "running," "runs," and "ran" would result in the common stem "run." Similarly, stemming the words "happiness," "happier," and "happiest" would result in the stem "happi."

5. **Approximate Process**: Stemming is an approximate process and may produce stems that are not actual words or may result in stems that are different from the base form of the word in some cases. Stemming algorithms prioritize simplicity and speed over accuracy, so they may not always produce linguistically correct stems.

Stemming is particularly useful in natural language processing (NLP) tasks where the precise meaning of words is less important, such as information retrieval, document clustering, and text classification. By reducing words to their stems, stemming helps to reduce the vocabulary size and improve the efficiency of text processing and analysis. Popular stemming algorithms include the Porter stemming algorithm, the Snowball stemming algorithm, and the Lancaster stemming algorithm. These algorithms are implemented in various NLP libraries such as NLTK (Natural Language Toolkit), spaCy, and TextBlob.

7. Explain Part-of-speech (POS) tagging.

Answer: Part-of-speech (POS) tagging, also known as grammatical tagging or word-category disambiguation, is the process of assigning a grammatical category (or part of speech) to each word in a sentence based on its syntactic role and context within the sentence. The goal of POS tagging is to label each word with its

appropriate part of speech, such as noun, verb, adjective, adverb, pronoun, preposition, conjunction, or interjection.

Here's how POS tagging works:

1. **Word Tokenization**: POS tagging begins with tokenizing the input text into individual words or tokens. Each token represents a word in the sentence.

2. **POS Tagging**: For each word in the sentence, a POS tagger analyzes its context, surrounding words, and grammatical features to determine the appropriate part of speech. This is typically done using pre-trained models or statistical algorithms that have learned patterns from annotated corpora.

3. **POS Tag Set**: POS taggers use a predefined set of tags, often based on linguistic conventions or universal standards such as the Penn Treebank tag set. Each tag represents a specific grammatical category, and different tag sets may have different levels of granularity and detail.

4. **Example**: For example, consider the sentence "The quick brown fox jumps over the lazy dog." A POS tagger would assign POS tags to each word in the sentence, resulting in a tagged sequence like this: [("The", "DT"), ("quick", "JJ"), ("brown", "JJ"), ("fox", "NN"), ("jumps", "VBZ"), ("over", "IN"), ("the", "DT"), ("lazy", "JJ"), ("dog", "NN")]. In this example, "DT" represents a determiner, "JJ" represents an adjective, "NN" represents a noun, and "VBZ" represents a verb in the third person singular form.

POS tagging is an essential preprocessing step in many natural language processing (NLP) tasks and applications, including syntactic parsing, information extraction, machine translation, sentiment analysis, and text-to-speech synthesis. Accurate POS tagging helps improve the performance of downstream NLP tasks by providing valuable syntactic and grammatical information about the text.

8. Explain Chunking or shallow parsing.

Answer: Chunking, also known as shallow parsing, is a natural language processing (NLP) technique used to identify and extract phrases or "chunks" of words in a sentence that belong together and serve a common grammatical role. Unlike full syntactic parsing, which involves analyzing the entire sentence's grammatical structure, chunking focuses on identifying smaller syntactic units or phrases based on patterns of words and their part-of-speech tags.

Here's how chunking works:

1. **POS Tagging**: Chunking typically begins with part-of-speech (POS) tagging, where each word in the sentence is assigned a grammatical category or POS tag, such as noun (NN), verb (VB), adjective (JJ), preposition (IN), etc.

2. **Chunking Patterns**: Chunking relies on predefined patterns or rules that specify which sequences of POS tags correspond to particular types of phrases or chunks. These patterns are often expressed using regular expressions or finite-state grammar rules.

3. **Identifying Chunks**: The chunker applies these patterns to the sequence of POS tags generated from the POS tagging step to identify and extract chunks that match the specified patterns. Common types of chunks include noun phrases (NP), verb phrases (VP), prepositional phrases (PP), etc.

4. **Example**: Consider the sentence "The quick brown fox jumps over the lazy dog." A simple chunking pattern for noun phrases might be "adjective noun" (JJ NN). Applying this pattern to the POS-tagged sentence would result in the extraction of two noun phrases: "the quick brown fox" and "the lazy dog."

5. **Output**: The output of the chunking process is a sequence of chunks or phrases extracted from the sentence, each with a label indicating its type or grammatical role. These chunks can then be used as input to further processing steps or analysis tasks, such as information extraction, named entity recognition, or sentiment analysis.

Chunking is a useful technique in NLP for extracting structured information from text data and identifying meaningful units of information within sentences. It helps to capture syntactic relationships and dependencies between words and can be used to identify relevant information for downstream NLP tasks.

9. Explain Noun Phrase (NP) chunking.

Answer: Noun Phrase (NP) chunking is a specific type of chunking or shallow parsing in natural language processing (NLP) that focuses on identifying and extracting noun phrases from sentences. A noun phrase is a phrase that consists of a noun (or pronoun) and optionally any associated modifiers such as adjectives, determiners, and prepositional phrases.

Here's how noun phrase (NP) chunking works:

1. **POS Tagging**: NP chunking typically begins with part-of-speech (POS) tagging, where each word in the sentence is assigned a grammatical category or POS tag, such as noun (NN), verb (VB), adjective (JJ), preposition (IN), etc.

2. **Chunking Patterns**: NP chunking relies on predefined patterns or rules that specify which sequences of POS tags correspond to noun phrases. These patterns often specify sequences of POS tags that represent the structure of noun phrases, including optional modifiers and determiners.

3. **Identifying NP Chunks**: The NP chunker applies these patterns to the sequence of POS tags generated from the POS tagging step to identify and extract noun phrases that match the specified patterns. Common patterns for noun phrases include sequences such as "determiner + adjective* + noun," where * denotes zero or more occurrences of adjectives.

4. **Example**: Consider the sentence "The quick brown fox jumps over the lazy dog." A simple NP chunking pattern might be "determiner + adjective* + noun" (DT JJ* NN). Applying this pattern to the POS-tagged sentence would result in the extraction of two noun phrases: "the quick brown fox" and "the lazy dog."

5. **Output**: The output of NP chunking is a sequence of noun phrase chunks extracted from the sentence, each with a label indicating its type or grammatical role (e.g., NP). These chunks represent meaningful units of information within the sentence and can be used for further analysis or processing in various NLP tasks.

NP chunking is a useful technique in NLP for extracting and identifying the main subjects and objects in sentences, as well as other noun-based entities. It helps to capture the syntactic structure of sentences and can be used as a preprocessing step for tasks such as information extraction, named entity recognition, and syntactic parsing.

10. Explain Named Entity Recognition.

Answer: Named Entity Recognition (NER) is a natural language processing (NLP) task that involves identifying and categorizing named entities in text into predefined categories such as person names, organization names, location names, dates, numerical expressions, and more. The goal of NER is to extract and classify specific entities mentioned in text data to provide structured information for downstream NLP tasks.

Here's how Named Entity Recognition works:

1. **Tokenization**: NER typically begins with tokenizing the input text into individual words or tokens. Each token represents a word in the text.

2. **Part-of-Speech Tagging**: The text is then annotated with part-of-speech (POS) tags, which classify each word according to its grammatical category (e.g., noun, verb, adjective, etc.). POS tagging helps provide additional context for identifying named entities.

3. **Named Entity Recognition**: NER models use various techniques such as rule-based approaches, machine learning algorithms (e.g., conditional random fields, recurrent neural networks, transformer-based models), or a combination of both to identify and classify named entities in the text. These models leverage features such as word embeddings, POS tags, syntactic dependencies, and contextual information to make predictions.

4. **Entity Categorization**: Once a named entity is identified, it is categorized into one of several predefined types or classes, such as person names, organization names, location names, date expressions, numerical values, etc. Some systems may use a generic "other" category for entities that do not fit into any predefined type.

5. **Output**: The output of Named Entity Recognition is a sequence of labeled entities, where each entity is associated with its corresponding type or class. For example, given the input text "Apple Inc. is headquartered in Cupertino, California," the output of NER might include the following named entities: ["Apple Inc."] labeled as an organization and ["Cupertino, California"] labeled as a location.

   Named Entity Recognition is widely used in various NLP applications and information extraction tasks, including:

   - Information Retrieval: Identifying relevant entities in text to improve search results.
   - Question Answering: Extracting named entities mentioned in questions or documents to provide accurate answers.
   - Named Entity Linking: Linking recognized entities to knowledge bases or databases to retrieve additional information.
   - Sentiment Analysis: Analyzing opinions and sentiments expressed towards specific entities mentioned in text.
   - Event Extraction: Identifying events, dates, and locations mentioned in news articles or social media posts.

NER plays a crucial role in structuring unstructured text data and extracting valuable information from large text corpora, making it an essential component in many NLP pipelines and systems.