Unit I: Lexical Processing - I

1. Question: What are the areas of application for Natural Language Processing (NLP)?

   Answer: Natural Language Processing (NLP) finds applications in various fields, including machine translation, sentiment analysis, information retrieval, question answering, text summarization, speech recognition, and chatbots. NLP is used in industries such as healthcare, finance, customer service, and social media analysis.

3. Question: What is text encoding in NLP?

   Answer: Text encoding refers to the process of representing text data as numerical values that can be processed by computers. In NLP, text encoding converts raw text into a format suitable for various NLP tasks. Common encoding techniques include one-hot encoding, word embeddings (such as Word2Vec or GloVe), and transformer-based models (like BERT or GPT).

4. Question: Explain the concept of regular expressions (regex) in NLP.

   Answer: Regular expressions are powerful patterns used for matching and manipulating text. In NLP, regex allows us to perform tasks like pattern matching, string extraction, and text manipulation. It uses a combination of symbols and metacharacters to define patterns and search for matches within a given text.

5. Question: What are quantifiers in regular expressions? Provide examples.

   Answer: Quantifiers in regular expressions specify the number of occurrences of a character or a group of characters. Some common quantifiers include:

- "*" matches zero or more occurrences.
- "+" matches one or more occurrences.
- "?" matches zero or one occurrence.
- "{n}" matches exactly n occurrences.
- "{n,}" matches at least n occurrences.
- "{n,m}" matches between n and m occurrences.

Example: The regex "a*b" matches "b", "ab", "aab", "aaab", and so on.

6. Question: How can regular expressions be used for comprehension in NLP?

   Answer: Regular expressions allow us to extract specific patterns or information from text. They enable tasks like searching for specific words or phrases, identifying email addresses or phone numbers, finding dates or URLs, and more. By using regular expressions, we can effectively process and extract relevant information from large text datasets.

7. Question: What are anchors and wildcards in regular expressions?

   Answer: Anchors and wildcards are special characters used in regular expressions. Anchors include "^" (caret) and "$" (dollar sign) and are used to match the start and end of a line, respectively. Wildcards include "." (dot) and "\w" and match any character or any word character (alphanumeric and underscore), respectively.

8. Question: Explain the concept of character sets in regular expressions.

   Answer: Character sets, denoted by square brackets "[ ]", allow us to specify a set of characters that can match at a particular position in a regex pattern. For example, the regex "[aeiou]" matches any vowel character. Character sets can also include ranges, such as "[a-z]" for lowercase letters or "[0-9]" for digits.

9. Question: What is the difference between greedy and non-greedy search in regular expressions?

   Answer: Greedy search in regular expressions matches the longest possible substring that satisfies the pattern. In contrast, non-greedy (or lazy) search matches the shortest possible substring that satisfies the pattern. Greedy search is the default behavior in regex, but non-greedy search can be specified by adding a "?" after a quantifier.

10. Question: What are some commonly used functions for regular expressions?

    Answer: Commonly used functions for regular expressions include pattern matching functions like **match()** and **search()**, string extraction functions like **findall()** and **finditer()**, and string replacement functions like **sub()** and **subn()**. These functions are provided by regex libraries in programming languages such as Python.

Unit II: Lexical Processing - II

1. Question: What is the importance of word frequencies and stop words in NLP?

   Answer: Word frequencies represent the number of occurrences of words in a given text or corpus. They provide insights into the significance and prevalence of words. Stop words are common words (such as "the," "is," "and") that are often removed from text during preprocessing as they carry less meaning and can hinder certain NLP tasks like sentiment analysis or topic modeling.

2. Question: What is tokenization in NLP?

   Answer: Tokenization is the process of splitting text into smaller units called tokens. Tokens can be words, sentences, or even subword units. Tokenization is a crucial step in NLP as it helps in preparing text data for further analysis or processing, such as part-of-speech tagging or sentiment analysis.

3. Question: Explain the concept of the Bag-of-Words representation.

   Answer: The Bag-of-Words (BoW) representation is a simple and commonly used approach to represent text data. It represents a document as a "bag" or collection of its constituent words, disregarding word order and grammar. Each word is treated as a feature, and the frequency or presence of words in the document is used as numerical values for these features.

4. Question: What is stemming and lemmatization in NLP?

   Answer: Stemming and lemmatization are techniques used to reduce words to their base or root forms. Stemming reduces words to their stem by removing suffixes, while lemmatization reduces words to their base form using linguistic analysis. These techniques help in reducing word variations and normalizing text data for improved text processing and analysis.

5. Question: How is the final Bag-of-Words representation created?

   Answer: The final Bag-of-Words representation is created by transforming the text data into a matrix or vector format. Each document is represented as a numerical vector, where each element corresponds to the frequency or presence of a specific word in the document. The resulting matrix can be used as input for various machine learning algorithms.

6. Question: What is TF-IDF representation in NLP?

   Answer: TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical representation that reflects the importance of a word in a document within a larger corpus. It calculates the product of the word's term frequency (TF) and its inverse document frequency (IDF). TF-IDF helps in highlighting words that are both frequent in a document and rare in the overall corpus.

7. Question: How can a spam detector be built using NLP techniques?

   Answer: A spam detector can be built using NLP techniques by using supervised learning algorithms such as Naive Bayes, Support Vector Machines (SVM), or decision trees. The detector can be trained on a labeled dataset of spam and non-spam emails, where features can include word frequencies, presence of certain words, or other linguistic patterns.

8. Question: What is canonicalization in NLP?

   Answer: Canonicalization is the process of converting text data into a standard or canonical form. It involves steps like removing punctuation, converting text to lowercase, handling contractions, expanding abbreviations, and normalizing characters. Canonicalization helps in reducing text variations and improving text processing and analysis.

9. Question: Explain the concept of phonetic hashing in NLP.

   Answer: Phonetic hashing is a technique used to transform words into phonetic representations. It helps in capturing the phonetic similarity of words, allowing for tasks like pronunciation matching or finding rhymes. Phonetic hashing algorithms like Soundex or Metaphone map words to phonetic codes based on their pronunciation.

10. Question: What is edit distance in NLP?

    Answer: Edit distance, also known as Levenshtein distance, is a metric used to measure the difference between two strings. It quantifies the minimum number of operations (insertions, deletions, or substitutions) required to transform one string into another. Edit distance has applications in spell correction, DNA sequence alignment, and approximate string matching.

Unit III: Syntactic Processing - I

1. Question: What is syntax in natural language processing?

   Answer: Syntax refers to the set of rules and principles that govern the structure of sentences and phrases in a language. It defines how words and phrases can be combined to form grammatically correct sentences. Syntax plays a crucial role in syntactic processing tasks such as part-of-speech tagging, parsing, and grammatical analysis.

2. Question: What are parts of speech (PoS) in NLP?

Answer: Parts of speech (PoS) are categories or tags assigned to words based on their grammatical roles and relationships in a sentence. Common PoS tags include nouns, verbs, adjectives, adverbs, pronouns, prepositions, conjunctions, and interjections. PoS tagging is the process of assigning these tags to words in a given sentence.

3. Question: How is PoS tagging performed in NLP?

   Answer: PoS tagging is typically performed using statistical models or rule-based approaches. Statistical models use machine learning algorithms trained on labeled corpora, where words are annotated with their corresponding PoS tags. Rule-based approaches rely on handcrafted rules and patterns to assign PoS tags based on linguistic properties and context.

4. Question: What is a Hidden Markov Model (HMM) in PoS tagging?

   Answer: A Hidden Markov Model (HMM) is a statistical model used for sequence labeling tasks, including PoS tagging. It assumes that the underlying states (PoS tags) generating observed outputs (words) form a Markov chain. HMMs utilize probabilities to estimate the most likely sequence of PoS tags given an input sentence.

5. Question: What is a PoS tagging application in NLP?

   Answer: PoS tagging has various applications in NLP, including information retrieval, machine translation, sentiment analysis, named entity recognition, and syntactic parsing. It provides valuable linguistic information for downstream tasks and helps in improving the accuracy of language processing algorithms.

6. Question: Can you provide a case study of PoS tagging in NLP?

   Answer: One case study of PoS tagging is part-of-speech disambiguation in machine translation. PoS tags help disambiguate the meaning of words in the source language, aiding in accurate translation. For example, correctly identifying whether a word is a noun or a verb can significantly impact the translation quality.

7. Question: What is constituency parsing in NLP?

   Answer: Constituency parsing is a syntactic parsing technique that aims to analyze the sentence structure by identifying constituents or phrases and their hierarchical relationships. It represents sentences as parse trees, where nodes represent constituents (such as noun phrases or verb phrases), and edges represent the syntactic relationships between them.

8. Question: What is dependency parsing in NLP?

   Answer: Dependency parsing is a syntactic parsing technique that focuses on analyzing the grammatical relationships between words in a sentence. It represents sentences as directed graphs, where words are nodes, and the edges indicate the dependency relationships between the words. Dependency parsing is widely used for tasks like information extraction and question answering.

9. Question: Can you demonstrate parsing in Python?

   Answer: Sure! Here's a Python code demonstration using the spaCy library for dependency parsing:

import spacy nlp = spacy.load('en_core_web_sm') sentence = "The cat chased the mouse." doc = nlp(sentence) for token in doc: print(token.text, token.dep_, token.head.text)

In this example, we load the spaCy English model, process a sentence, and iterate over the tokens to print their text, dependency labels, and the head word they depend on.

10. Question: What are some limitations of parsing techniques in NLP?

    Answer: Parsing techniques in NLP can face challenges when dealing with ambiguous sentences, out-of-vocabulary words, or languages with free word order. Parsing accuracy heavily relies on the quality of the training data and the complexity of the sentence structures. Additionally, parsing can be computationally expensive for large-scale applications.

Unit IV: Syntactic Processing - II

1. Question: What is Named Entity Recognition (NER) in NLP?

   Answer: Named Entity Recognition (NER) is a task in NLP that involves identifying and classifying named entities in text into predefined categories such as person names, organizations, locations, dates, and more. NER plays a vital role in information extraction, question answering, and text summarization.

2. Question: How is Named Entity Recognition performed in NLP?

   Answer: Named Entity Recognition is typically approached as a sequence labeling task. Machine learning models, such as Conditional Random Fields (CRF) or deep learning models like Recurrent Neural Networks (RNN) or Transformers, are trained on labeled data to predict the named entity labels for each word in a given sentence.

3. Question: What is IOB labeling in Named Entity Recognition?

   Answer: IOB labeling (Inside, Outside, Beginning) is a common tagging scheme used in Named Entity Recognition. Each word in a sentence is labeled as either B-ENTITY, I-ENTITY, or O. B-ENTITY indicates the beginning of a named entity, I-ENTITY indicates words inside a named entity, and O indicates words that are not part of any named entity.

4. Question: Can you provide a Python demonstration for Named Entity Recognition?

   Answer: Certainly! Here's an example of using the spaCy library for Named Entity Recognition in Python:

import spacy nlp = spacy.load('en_core_web_sm') sentence = "Apple Inc. was founded by Steve Jobs in Cupertino, California." doc = nlp(sentence) for ent in doc.ents: print(ent.text, ent.label_)

In this example, we load the spaCy English model, process a sentence, and iterate over the entities in the document to print the text and label of each named entity.

5. Question: What is Conditional Random Field (CRF) in NER?

Answer: Conditional Random Field (CRF) is a popular probabilistic graphical model used for sequence labeling tasks like Named Entity Recognition. CRF models consider the dependencies between neighboring labels and capture the contextual information to make accurate predictions. CRFs are trained using labeled data and can achieve good performance in NER.

6. Question: How is the CRF model trained for NER? (Part I)

   Answer: Training a CRF model for NER involves several steps. First, a labeled dataset is prepared, where each word in the training sentences is annotated with its corresponding named entity label. Next, features such as word context, part-of-speech tags, and gazetteer information are extracted. Finally, the CRF model is trained using these features and the labeled data to learn the patterns and dependencies between words and labels.

7. Question: How is the CRF model used for prediction in NER? (Part II)

   Answer: Once the CRF model is trained, it can be used for prediction on new, unseen sentences. For each input sentence, features are extracted, and the trained CRF model predicts the most likely sequence of named entity labels for each word. The predicted labels can then be used for downstream tasks or analysis.

8. Question: Can you provide a Python implementation for custom Named Entity Recognition?
   Answer: Certainly! Here's an example of a simple Python implementation for custom NER using regular expressions:

import re def extract_entities(text): entities = [] pattern = r"\b[A-Z][a-zA-Z]+\b" # Matches capitalized words matches = re.findall(pattern, text) for match in matches: entities.append((match, 'ENTITY')) return entities sentence = "John works at Apple Inc. in California." entities = extract_entities(sentence) for entity in entities: print(entity[0], entity[1])

In this example, we define a function **extract_entities** that uses a regular expression pattern to match capitalized words in the input text. The matched words are considered as entities and are returned with a generic label 'ENTITY'.

9. Question: What are the potential applications of Named Entity Recognition?

   Answer: Named Entity Recognition has numerous applications, including information extraction, question answering, sentiment analysis, chatbots, recommendation systems, and text summarization. It enables the identification and extraction of important entities from unstructured text, facilitating various NLP tasks.

10. Question: Can you explain the concept of customizing NER models for domain-specific data?
    Answer: Customizing NER models involves training the models on domain-specific data to improve their performance in a particular domain. This process typically requires annotating a labeled dataset specific to the domain of interest. By training the NER model on this domain-specific data, it can learn domain-specific patterns, terminology, and entities, resulting in more accurate entity recognition in that specific domain.

REMAINING QUESTIONS

Unit IV: Syntactic Processing - II

4. Question: Can you demonstrate Named Entity Recognition in Python?

   Answer: Certainly! Here's a Python code demonstration using the spaCy library for Named Entity Recognition:

import spacy nlp = spacy.load('en_core_web_sm') sentence = "Apple Inc. was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne." doc = nlp(sentence) for ent in doc.ents: print(ent.text, ent.label_)

In this example, we load the spaCy English model, process a sentence, and iterate over the entities recognized by the model. For each entity, we print its text and the corresponding entity label.

5. Question: What is Conditional Random Field (CRF) in NER?

   Answer: Conditional Random Field (CRF) is a probabilistic model often used in Named Entity Recognition (NER). It is a type of graphical model that models the dependencies between sequential labels, such as the labels assigned to each word in a sentence. CRFs capture both local and global contextual information to make accurate predictions for NER tasks.

6. Question: How is a CRF model trained for Named Entity Recognition (NER)?

   Answer: Training a CRF model for NER involves providing annotated training data with labeled entities. The model learns the patterns and dependencies between words and entity labels in the training data. Features such as word representations, part-of-speech tags, and context windows around words are used to capture relevant information. The CRF model is then trained using optimization techniques, such as maximum likelihood estimation or gradient descent, to maximize the likelihood of the correct label sequence given the input sentence.

7. Question: How does a CRF model make predictions in Named Entity Recognition (NER)?

   Answer: Once a CRF model is trained, it can make predictions by calculating the probabilities of different entity label sequences for a given input sentence. The model considers various factors, such as word features and contextual information, to estimate the most likely label sequence using the Viterbi algorithm or other decoding algorithms. The predicted label sequence represents the recognized named entities in the input sentence.

8. Question: Can you provide a Python implementation for custom Named Entity Recognition (NER)?

   Answer: Certainly! Here's a simplified Python implementation for custom Named Entity Recognition using CRF and the sklearn-crfsuite library:

import sklearn_crfsuite from sklearn_crfsuite import metrics # Prepare training data with features and labels X_train = [...] # Feature representation for each word y_train = [...] # Corresponding entity labels for each word # Train CRF model crf = sklearn_crfsuite.CRF() crf.fit(X_train, y_train) # Prepare test data X_test = [...] # Feature representation for each word # Make predictions with CRF model y_pred = crf.predict(X_test) # Evaluate the model metrics.flat_f1_score(y_test, y_pred, average='weighted')

In this example, you would need to prepare training data with feature representations (**X_train**) and corresponding entity labels (**y_train**). Then, a CRF model is initialized and trained using the **fit()** function. For testing, you need to prepare test data (**X_test**) and make predictions using the **predict()** function. Finally, you can evaluate the model using metrics such as F1 score.

Unit V: Semantic Processing - I

1. Question: What are Knowledge Graphs in NLP?

   Answer: Knowledge Graphs are structured representations of knowledge that capture relationships between entities. They consist of nodes (representing entities) and edges (representing relationships). Knowledge Graphs provide a way to organize and represent factual information in a machine-readable format, enabling efficient knowledge retrieval and inference.

2. Question: How are Knowledge Graphs created in NLP?

   Answer: Knowledge Graphs can be created using various methods. One common approach is to extract information from unstructured text using Natural Language Processing techniques such as named entity recognition, relation extraction, and semantic role labeling. The extracted information is then structured into a graph format, where entities become nodes and relationships become edges. Additionally, existing structured knowledge sources, such as databases and ontologies, can be integrated to enrich the Knowledge Graph.

3. Question: What are the advantages of using Knowledge Graphs in NLP?

   Answer: Knowledge Graphs offer several advantages in NLP:

- Enriched representation: Knowledge Graphs provide a structured representation that captures semantic relationships between entities, enabling more accurate and context-aware understanding of textual data.
- Knowledge integration: Knowledge Graphs can integrate information from multiple sources, such as text, databases, and ontologies, allowing for comprehensive knowledge exploration and reasoning.
- Efficient querying: Knowledge Graphs facilitate efficient information retrieval by enabling complex queries that traverse the graph and leverage relationships between entities.
- Inference and reasoning: Knowledge Graphs support reasoning mechanisms that allow for logical deductions, making it easier to derive new knowledge and answer complex questions.
- Knowledge-driven applications: Knowledge Graphs serve as a foundation for various NLP applications, including question-answering systems, recommendation systems, and semantic search engines.