**SRM Institute of Science and Technology**

**College of Engineering and Technology**

**School of Computing**

SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamilnadu

**Academic Year: 2024 - 25 EVEN**

**B.Tech-School of Computing**

**Test: FT2** **Date: 21.2.2025**

**Course Code & Title****: 21CSE356T – Natural Language Processing Duration: 2 periods**

**Sem: VI Sem** **Max. Marks: 50**

**Batch II SET : A**

ANSWER KEY

1.A

2.D

3.C

4.A

5.D

6.C

7.C

8.D

9.A

10.C

11.A.

**1. Unigrams: (2m)**

Unigrams are individual words. In this sentence, the unigrams are:

* I
* am
* feeling
* hungry

**2. Bigrams(2m)**

Bigrams are sequences of two adjacent words. In this sentence, the bigrams are:

* I am
* am feeling
* feeling hungry

**3. Trigrams: (1m)**

Trigrams are sequences of three adjacent words. In this sentence, the trigrams are:

* I am feeling
* am feeling hungry

11.B

a)Lexical (2.5m)

**a) Lexical Analysis:**

Lexical analysis is the foundational layer of NLP. It focuses on the individual words or morphemes (the smallest meaningful units of language) within a text. Think of it as the process of understanding the building blocks of language. This phase doesn't delve into the relationships between words or the overall meaning of a sentence; it's concerned with the form and characteristics of individual words.

Key tasks in lexical analysis include:

* **Tokenization:** Breaking down the text into individual words, punctuation marks, or other meaningful units called tokens. For example, "The cat sat on the mat." would be tokenized into "The," "cat," "sat," "on," "the," "mat," "."
* **Morphological Analysis:** Analyzing the structure of words to identify their root forms (stems or lemmas), prefixes, suffixes, and other morphemes. This helps to understand the meaning of inflected words. For example, "running" can be broken down into "run" (root) and "-ing" (suffix).
* **Part-of-Speech Tagging (POS Tagging):** Assigning grammatical tags (noun, verb, adjective, adverb, etc.) to each token based on its context. This helps to understand the grammatical role of each word in a sentence. For example, "run" can be a verb ("I run") or a noun ("a run").
* **Stemming:** Reducing words to their root form (stem) by chopping off suffixes. This is a heuristic approach and may not always produce a valid word. For example, "running," "runs," and "ran" might all be stemmed to "run."
* **Lemmatization:** Similar to stemming, but it aims to find the dictionary form of a word (lemma), considering its meaning and context. Lemmatization produces valid words. For example, the lemma of "running," "runs," and "ran" is "run."

b)Semantic (2.5 m)

**b) Semantic Analysis:**

Semantic analysis builds upon the output of lexical analysis to understand the meaning of words, phrases, and sentences. It goes beyond the surface form of words to extract the actual meaning conveyed by the text. This phase deals with the relationships between words and their meanings, as well as the overall meaning of a sentence or document.

Key tasks in semantic analysis include:

* **Word Sense Disambiguation (WSD):** Determining the correct meaning of a word that has multiple senses (polysemy) in a given context. For example, "bank" can refer to a financial institution or a river bank. WSD helps to choose the right meaning.
* **Semantic Role Labeling (SRL):** Identifying the roles that different words or phrases play in a sentence, such as agent, patient, instrument, etc. This helps to understand who did what to whom.
* **Lexical Relation Analysis:** Identifying semantic relationships between words, such as synonymy (words with similar meanings), antonymy (words with opposite meanings), hypernymy/hyponymy (is-a relationships), meronymy (part-whole relationships), etc.
* **Sentiment Analysis:** Determining the emotional tone (positive, negative, neutral) expressed in a text.
* **Understanding Sentence Meaning:** Combining the meanings of individual words and phrases to derive the overall meaning of a sentence. This often involves considering the sentence structure and context.

12.A Definition: (1m) Part-of-Speech (POS) tagging is the process of assigning grammatical tags (such as noun, verb, adjective, adverb, etc.) to each word in a sentence or text. It's a fundamental step in many Natural Language Processing (NLP) tasks because understanding the grammatical role of words is crucial for understanding the meaning of the text. For example, the word "run" can be a verb ("I run") or a noun ("a 5k run"). POS tagging helps to distinguish between these different uses.

Here's a discussion of various approaches to perform POS tagging in NLP: Any 2(2 m each)

**1. Rule-Based Tagging:**

* **Concept:** This approach relies on a set of handcrafted rules based on grammar and lexicon. These rules specify the possible tags for a word based on its form, context, and surrounding words.
* **Process:** A lexicon (dictionary) stores the possible tags for each word. Rules are then applied to disambiguate the tags based on the context. For example, a rule might state that a word ending in "-ing" is usually a verb (gerund or present participle) unless it's preceded by an article (in which case it's likely a noun).
* **Advantages:** Simple to implement for small lexicons and grammars. Can be effective for specific domains with well-defined rules.
* **Disadvantages:** Difficult and time-consuming to create and maintain a comprehensive set of rules. Doesn't handle unseen words well. Limited ability to capture complex grammatical contexts.

**2. Statistical Tagging:**

* **Concept:** This approach uses statistical models trained on large corpora of tagged text to predict the most likely tag for each word. It leverages probabilities and frequencies of word-tag co-occurrences.
* **Types:**
  + **Hidden Markov Models (HMMs):** HMMs are probabilistic models that assume the tag of a word depends only on the previous tag (first-order Markov assumption). They use the Viterbi algorithm to find the most likely sequence of tags for a sentence.
  + **Conditional Random Fields (CRFs):** CRFs are more powerful than HMMs as they can consider dependencies on multiple previous tags and even features of the words themselves (e.g., prefixes, suffixes). They are discriminative models, directly learning the conditional probability of tags given the words.
* **Advantages:** Can handle unseen words and complex grammatical contexts. More accurate than rule-based tagging, especially for large corpora.
* **Disadvantages:** Requires large amounts of training data. Computationally more intensive than rule-based tagging.

**3. Transformation-Based Tagging (Brill Tagging):**

* **Concept:** This approach starts with an initial tagging (often using a simple lexicon) and then applies a set of learned rules (transformations) to correct the tags. The rules are learned from a training corpus.
* **Process:** The algorithm learns a set of transformations that improve the accuracy of the tagging. These transformations might change a tag based on the surrounding words or other features.
* **Advantages:** Can achieve high accuracy with a relatively small number of rules. Easy to interpret the learned rules.
* **Disadvantages:** Can be slower than statistical tagging.

**4. Neural Network-Based Tagging:**

* **Concept:** Recurrent Neural Networks (RNNs), especially LSTMs and GRUs, are well-suited for sequence labeling tasks like POS tagging. They can capture long-range dependencies in the text and learn complex patterns. More recently, Transformer models have achieved state-of-the-art results.
* **Advantages:** Can achieve very high accuracy, especially with large training datasets. Can capture complex linguistic patterns.
* **Disadvantages:** Requires significant computational resources and large amounts of training data. Can be difficult to interpret the model's decisions.

12B. Any 2 in detail

 **Machine Translation:** Translating text or speech from one language to another.

 **Text Summarization:** Condensing large amounts of text into shorter summaries.

 **Sentiment Analysis:** Determining the emotional tone (positive, negative, neutral) in text.

 **Chatbots and Virtual Assistants:** Creating conversational agents that can interact with humans.

 **Information Retrieval:** Finding relevant information from a large collection of text.

 **Spam Detection:** Identifying and filtering out spam emails or messages.

 **Speech Recognition:** Converting spoken language into text.

 **Text Generation:** Generating human-like text, such as creative writing or news articles.

 **Named Entity Recognition (NER):** Identifying and classifying named entities (people, organizations, locations, etc.) in text.

 **Question Answering:** Providing answers to questions posed in natural language.

 **Spell Checking and Grammar Correction:** Identifying and correcting errors in text.

 **Content Recommendation:** Recommending relevant content to users based on their past interactions.

 **Fake News Detection:** Identifying and flagging potentially false or misleading news articles.

13A.

A screenshot of a computer

AI-generated content may be incorrect. 13B. **a) One-Hot Encoding:**

One-hot encoding is a technique used to represent categorical variables as numerical vectors. It's particularly useful in machine learning when dealing with features that are not ordinal (i.e., there's no inherent order). Imagine you have a feature called "color" which can be "red," "green," or "blue." One-hot encoding would create three new binary (0 or 1) features, one for each color: "color\_red," "color\_green," and "color\_blue." If the original value was "red," then "color\_red" would be 1, and the other two would be 0. This representation prevents the model from interpreting the categories as having a numerical relationship (e.g., blue > red). One-hot encoding is essential for many machine learning algorithms that require numerical input.

**b) N-grams:**

N-grams are contiguous sequences of *n* items from a given text or speech. These items can be characters, words, or even sub-word units. N-grams are used in various NLP tasks, including language modeling, text classification, and information retrieval. For example, in the sentence "The cat sat on the mat," the bigrams (2-grams) are "The cat," "cat sat," "sat on," "on the," and "the mat." Trigrams (3-grams) would be "The cat sat," "cat sat on," "sat on the," and "on the mat." N-grams capture some local contextual information and are useful for tasks where word order is important, but not the entire sentence structure. They help to represent text numerically, which is crucial for statistical and machine-learning models.

14A.

Grammar rules govern how words are combined to form phrases, clauses, and sentences in English. They dictate the structure and organization of the language, ensuring clarity and meaning. While a comprehensive treatment of English grammar is extensive, here's a brief overview of key elements:

**1. Parts of Speech:**

English words fall into different categories called parts of speech, each with specific functions:

* **Nouns:** Represent people, places, things, or ideas (e.g., *cat, house, freedom*).
* **Verbs:** Express actions or states of being (e.g., *run, think, exist*).
* **Adjectives:** Describe nouns (e.g., *red, big, happy*).
* **Adverbs:** Modify verbs, adjectives, or other adverbs (e.g., *quickly, very, often*).
* **Pronouns:** Replace nouns (e.g., *he, she, it, they*).
* **Prepositions:** Show relationships between words (e.g., *on, in, at, to*).
* **Conjunctions:** Connect words, phrases, or clauses (e.g., *and, but, or, because*).
* **Determiners:** Specify nouns (e.g., *the, a, this, some*).

**2. Phrase Structure:**

Words are grouped into phrases, which serve as building blocks for larger structures:

* **Noun Phrase (NP):** Contains a noun and its modifiers (e.g., *the big cat, my friend John*).
* **Verb Phrase (VP):** Contains a verb and its related elements (e.g., *runs quickly, ate the cake*).
* **Prepositional Phrase (PP):** Contains a preposition and its object (e.g., *on the table, in the house*).

**3. Sentence Structure:**

Sentences typically follow a Subject-Verb-Object (SVO) order, though variations exist:

* **Subject:** The noun phrase that performs the action (e.g., *The cat* sat on the mat).
* **Verb:** The verb phrase expressing the action or state (e.g., The cat *sat* on the mat).
* **Object:** The noun phrase that receives the action (e.g., The cat sat on *the mat*).

**4. Clauses:**

Clauses are groups of words containing a subject and a verb. Sentences can consist of one or more clauses:

* **Independent Clause:** Can stand alone as a sentence (e.g., *The cat sat on the mat*).
* **Dependent Clause:** Cannot stand alone and requires an independent clause to form a complete sentence (e.g., *because the cat was tired*).

**5. Grammar Rules (Examples):**

Grammar rules, often represented using formal notations like Backus-Naur Form (BNF), describe how these elements combine:

* S -> NP VP (A sentence consists of a noun phrase followed by a verb phrase).
* NP -> Det N (A noun phrase can be a determiner followed by a noun).
* VP -> V NP (A verb phrase can be a verb followed by a noun phrase).

14B. The CKY (Cocke-Kasami-Younger) algorithm is a parsing algorithm for context-free grammars (CFGs). It's particularly efficient for grammars in Chomsky Normal Form (CNF). Let's break down how it works with an example. (2m)

**1. Chomsky Normal Form (CNF):**

The CKY algorithm requires the grammar to be in CNF. A CFG is in CNF if all production rules are in one of the following forms:

* A → B C (where A, B, and C are non-terminal symbols)
* A → a (where A is a non-terminal and 'a' is a terminal symbol)
* S → ε (where S is the start symbol and ε represents an empty string – rarely needed)

**2. The CKY Table:**

The algorithm uses a dynamic programming approach, filling in a table (usually triangular) to store intermediate results. For a sentence of length *n*, the table has dimensions *(n+1) x (n+1)*.

**3. Example Grammar (in CNF):**

Let's use a simple grammar for parsing:

* S → NP VP
* NP → Det N
* VP → Verb NP
* Det → the
* N → cat | mat
* Verb → sat

**4. Example Sentence:**

"The cat sat on the mat"

For simplicity, let's parse the shorter sentence: "The cat sat"

**5. CKY Table Construction:**

The table is filled in a bottom-up manner.

* **Step 1 (Base Case):** Fill the bottom diagonal of the table with the terminal symbols (words) and their corresponding non-terminals.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **0 (The)** | **1 (cat)** | **2 (sat)** |
| **0** | Det |  |  |
| **1** |  | N |  |
| **2** |  |  | Verb |
| **3** |  |  |  |

* **Step 2:** Look for rules that derive pairs of adjacent symbols in the row below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **0 (The)** | **1 (cat)** | **2 (sat)** |
| **0** | Det | NP |  |
| **1** |  | N | VP |
| **2** |  |  | Verb |
| **3** |  |  |  |

* **Explanation:**
  + Cell (0, 1): We see "Det" in cell (0, 0) and "N" in cell (1, 1). The rule NP → Det N applies, so we put "NP" in cell (0, 1).
  + Cell (1, 2): We see "N" in cell (1, 1) and "Verb" in cell (2, 2). The rule VP → Verb does *not* apply. We need a rule VP -> Verb NP. Since we don't have an NP starting at position 2, we can't fill cell (1, 2) yet.
* **Step 3:** Now, we consider spans of length 3.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **0 (The)** | **1 (cat)** | **2 (sat)** |
| **0** | Det | NP | S |
| **1** |  | N | VP |
| **2** |  |  | Verb |
| **3** |  |  |  |

* **Explanation:**
  + Cell (0, 2): We have "Det" in (0,0), "N" in (1,1) making NP at (0,1). Then "Verb" in (2,2). The rule S → NP VP applies, so we add "S" to cell (0, 2).

**6. Result:**

The presence of the start symbol "S" in the top-right cell of the table indicates that the sentence is grammatical according to the given CFG.

**7. Parse Tree Reconstruction (Backtracking): (3m)**

To get the actual parse tree, you backtrack through the table. Starting from the "S" in the top-right cell, you look for the rules that led to that entry.

* S (0, 2) was derived from NP (0, 1) and VP (1, 2) using rule S → NP VP.
* NP (0, 1) was derived from Det (0, 0) and N (1, 1) using rule NP → Det N.
* VP (1, 2) was derived from Verb (2, 2) using rule VP → Verb.

This gives you the parse tree:

S

/ \

NP VP

/ \ |

Det N Verb

| | |

The cat sat

15A.

**N-gram Modeling? (2m)**

N-gram modeling is a simple yet effective technique in Natural Language Processing (NLP) used for predicting the next word in a sequence. It's based on the idea that the probability of a word depends on the preceding *n-1* words. An "n-gram" is a contiguous sequence of *n* items (words, characters, etc.) from a given text.

**Types of N-grams: ( 2m)**

* **Unigram (1-gram):** A single word. Example: "I," "am," "Henry"
* **Bigram (2-gram):** A sequence of two adjacent words. Example: "I am," "am Henry," "Henry I"
* **Trigram (3-gram):** A sequence of three adjacent words. Example: "I am Henry," "am Henry </S>," "Henry I am"

**Example Sentence Breakdown (from your corpus):**

Let's take the sentence "<S> I am Henry </S>" and break it down into n-grams:

* Unigrams: <S>, I, am, Henry, </S>
* Bigrams: <S> I, I am, am Henry, Henry </S>
* Trigrams: <S> I am, I am Henry, am Henry </S>

**Trigram Probability and Prediction: (4m)**

To predict the next word using trigram probability, we calculate the probability of a word given the previous two words. The formula is:

P(Wn | Wn-2, Wn-1) = Count(Wn-2, Wn-1, Wn) / Count(Wn-2, Wn-1)

Where:

* Wn is the word we want to predict.
* Wn-2 and Wn-1 are the two preceding words.
* Count() is the number of times that sequence appears in the corpus.

**Predicting the Next Word for "<S> DO I Like ?"**

We want to predict the next word after "I Like." We'll consider trigrams ending with "I Like" and see what words follow in your corpus.

1. **Identify Relevant Trigrams:**
   * <S> Do I
   * Do I like
   * I like college
   * I like Henry
2. **Calculate Probabilities:**
   * P(college | Do, I, Like) = Count(Do I like college) / Count(Do I like) = 2/2 = 1
   * P(Henry | Do, I, Like) = Count(Do I like Henry) / Count(Do I like) = 1/2 = 0.5
3. **Prediction: (1m)**

Since the probability of "college" given "Do I like" is 1, and the probability of "Henry" is 0.5, our trigram model would predict "college" as the next word.

**Therefore, the trigram model predicts the next word to be "college," resulting in the sequence "<S> DO I Like college."**

15B.

Definition:1m

Text normalization is a crucial preprocessing step in Natural Language Processing (NLP). It aims to transform text into a consistent and standardized form, making it easier for NLP models to understand and process. Raw text often contains variations in spelling, capitalization, punctuation, and formatting, which can hinder the performance of NLP algorithms.

**Need for Text Normalization: (3m)**

1. **Improving Model Accuracy:** NLP models often rely on statistical patterns in the text. Variations in word forms (e.g., "run," "running," "runs") can lead the model to treat them as different words, even though they have the same root meaning. Normalization reduces these variations, allowing the model to generalize better and improve accuracy.
2. **Reducing Data Sparsity:** The vocabulary size of a raw text corpus can be very large due to the presence of different forms of the same word. Normalization reduces the vocabulary size by grouping related words together, which helps to alleviate the data sparsity problem.
3. **Enhancing Efficiency:** Normalizing text reduces the computational cost of NLP tasks. A smaller vocabulary and consistent representation leads to faster processing.
4. **Handling Noise and Inconsistency:** Raw text often contains noise, such as typos, inconsistent capitalization, and special characters. Normalization helps to clean the text and ensure consistency.
5. **Improving Cross-Lingual Analysis:** Normalization can be particularly important when dealing with multiple languages, as different languages may have different conventions for capitalization, punctuation, and word forms.

**Steps Involved in Text Normalization: (6m) with examples**

Text normalization typically involves several steps, which may be applied in different orders depending on the specific NLP task and the characteristics of the text:

1. **Tokenization**
2. **Lowercasing**
3. **Punctuation Removal**
4. **Stop Word Removal**
5. **Handling Special Characters**
6. **Number Normalization**
7. **Stemming**
8. **Lemmatization**

16A.

**What is Dependency Parsing? (2m)**

Dependency parsing is a method of syntactic analysis that focuses on the relationships between words in a sentence. Unlike constituency parsing (which groups words into phrases), dependency parsing represents the sentence structure by showing how words depend on each other. These dependencies are directed, meaning they show which word is the *head* (governor) and which is the *dependent*.

* **Head:** The word that governs the other word.
* **Dependent:** The word that is governed by the head.

Dependencies are typically represented as labeled arrows or arcs connecting the head to the dependent. The label indicates the grammatical relation (e.g., subject, object, modifier).

**Example Dependency Relations: (2m)**

* **nsubj (nominal subject):** The noun that performs the action of the verb. (e.g., "The dog barks." - "dog" is the nsubj of "barks").
* **dobj (direct object):** The noun that receives the action of the verb. (e.g., "She eats the cake." - "cake" is the dobj of "eats").
* **amod (adjectival modifier):** An adjective that modifies a noun. (e.g., "the red car" - "red" is the amod of "car").
* **advmod (adverbial modifier):** An adverb that modifies a verb, adjective, or another adverb. (e.g., "He runs quickly." - "quickly" is the advmod of "runs").
* **prep (prepositional modifier):** A preposition and its object. (e.g., "The book on the table." - "on the table" is the prep of "book").

**Dependency Parse Tree for "The quick brown fox jumps over the lazy dog." (4m)**

jumps

/ | \

fox over .

/ | \

the quick brown dog

/ |

the lazy

**Dependency Relations (with labels):**

* jumps --nsubj--> fox
* fox --det--> The
* fox --amod--> quick
* fox --amod--> brown
* jumps --prep--> over
* over --pobj--> dog
* dog --det--> the
* dog --amod--> lazy

**Python Implementation (using spaCy): (1 m)**

import spacy

nlp = spacy.load("en\_core\_web\_sm") # Load a small English language model

text = "The quick brown fox jumps over the lazy dog."

doc = nlp(text)

for token in doc:

print(token.text, "-->", token.head.text, token.dep\_)

# To visualize the dependency tree (requires displacy):

from spacy import displacy

displacy.render(doc, style="dep", jupyter=True)

* **Applications:** Dependency parsing is crucial for tasks like: (1m)
  + **Machine translation**
  + **Information extraction**
  + **Question answering**
  + **Sentiment analysis**

16B.

**What is Top-Down Parsing? (2m)**

Top-down parsing is a parsing strategy that starts with the root of the parse tree (the start symbol, 'S' in this case) and attempts to derive the input sentence by applying the grammar rules. It's a predictive process, exploring possible parse trees until one matches the input. If a path leads to a dead end (doesn't match the input), the parser backtracks and tries another path.

**Grammar: (3m)**

* S -> NP VP
* PP -> P NP
* VP -> V NP
* VP -> VP PP
* P -> with
* V -> drink
* NP -> NP PP
* NP -> N
* N -> ram
* N -> milk
* N -> chocolate
* N -> coffee

**Sentence:** "Ram drink milk with chocolate."

**Top-Down Parsing Steps: (3m)**

1. **Start with the Start Symbol (S):**

We begin with the rule S -> NP VP.

1. **Expand NP:**

We have two options for NP: NP -> NP PP or NP -> N. Let's try NP -> N first.

1. **Expand N:**

N -> ram matches the first word "Ram" in the sentence.

1. **Expand VP:**

We have two options for VP: VP -> V NP or VP -> VP PP. Let's try VP -> V NP.

1. **Expand V:**

V -> drink matches the second word "drink."

1. **Expand NP:**

Again, we have NP -> NP PP or NP -> N. Let's try NP -> N.

1. **Expand N:**

N -> milk matches the third word "milk."

1. **Expand PP:**

We are left with "with chocolate." We need to expand the PP. PP -> P NP.

1. **Expand P:**

P -> with matches the fourth word "with."

1. **Expand NP:**

NP -> N.

1. **Expand N:**

N -> chocolate matches the fifth word "chocolate."

1. **Success!**

We have now successfully derived the entire sentence "Ram drink milk with chocolate" from the start symbol 'S' using the grammar rules.

**Parse Tree: (2m)**

The resulting parse tree looks like this:

S

/ \

NP VP

| / \

N V NP

| | / \

ram drink NP PP

| / \

N P NP

| | |

milk with N

|

chocolate

**Key Characteristics of Top-Down Parsing:**

* **Starts with the start symbol:** The derivation begins from the top of the tree.
* **Predictive:** It uses the grammar rules to predict what it expects to see in the input.
* **Recursive:** It often uses recursion to handle nested structures.
* **Backtracking:** It may need to backtrack if a chosen path doesn't lead to a successful parse.

**Advantages:**

* Conceptual simplicity: Easy to understand and implement the basic algorithm.

**Disadvantages:**

* Inefficiency: Can be inefficient due to excessive backtracking, especially for complex grammars or ambiguous sentences.
* Left recursion: Struggles with left-recursive grammars (e.g., A -> A b). These can lead to infinite loops.