1) Explain Topdown parsing Technique with an example.

It shows how to construct an efficient known backtracking from topdown parsing called a Predictive Parser.

1) first 2) follow.

first :-

To compile first(x) for all grammar symbols 'x', apply the following rules until no more terminals (or) $\in$ (epsilon) can be added to only first.

Rules :-

* If x is a terminal then first(x) is set{x}.
* If $x \to \in$ is a production then add $\in$ to first(x)
* If x is a non-terminals $x \to Y_1, Y_2, Y_3 \ldots Y_n$ is a production then place a first(x). If for some i in first(Y_i)

Ex :- $E \longrightarrow TE'$

$E' \longrightarrow +TE' | \in$

$T \longrightarrow FT'$

$T' \longrightarrow *FT' | \in$

$F \longrightarrow (E) | id.$

1) first(E) = first(T) = first(F) = {c, id}

(E: Non Terminal, T: Non Terminal, F: non Terminal, c, id: Terminal)

2) first(E') = {+, $\in$}

3) first(T') = {*, $\in$}.

follow :- To compute follow(A) apply the following rules until nothing can be added to any follow set.

Rules :-

* $ in follow(S) where S is the start symbol and $ is the input right and marker.
* If there is a production in the form of $A \to \alpha B\beta$ then everything in first(β) except for $\in$ is placed in follow(B)
* If there is a production $A \to \alpha B\beta$ (or) $A \to \alpha B$ containing $\in$ ($B \to \in$) then everything in follow(β) is in follow(B)

Eg:- $E \to TE'$
$E' \to +TE' | \in$
$T \to FT'$
$T' \to *FT' | \in$
$F \to (E) | id.$

follow(E) = { $,) }
follow(E') = { $,) }
follow(T) = { +, $,) }
follow(T') = { +, $,) }
follow(F) = { *, +, $,) }.

Construction of predictive parsing Table
I/p :- context free grammar
O/p :- predictive parsing table.

method :-

1) for each production is in the form $A \to \alpha$ of the grammar do step 2 & 3.
2) for each terminal in first(α) and $A \to \alpha$ to M[A, a]
3) If "$\in$" is in first(α) add $A \to \alpha$ to M[A, b] for each terminal b in follow(A). If $\in$ is in first(α) and $ in follow(A) add $A \to \alpha$ to M[A, $]

4) Make each undefined entry 'm' be error.

Predictive parser Table / Top Down Parsing.

Non Terminals	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \in$	$E' \to \in$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \in$	$T' \to *FT'$		$T' \to \in$	$T' \to \in$
F	$F \to id$			$F \to (E)$		

Rules:—

* If $X = a = \$$, the parser halts and announces successful completion of parsing.
* If $X = a \neq \$$, then parser pops X off the stack and advances the input pointer to the next input symbol.
* If X is a non-terminal the program consult entry $m[X, a) = X \to UVW$, the parser replaces X on top of the stack WVU.

Stack	Input	output.
$E	id + id * id $	-
$E'T	id + id * id $	$E \to TE'$
$E'T'F	id + id * id $	$T \to FT'$
$E'T'id	id + id * id $	$F \to id$
$E'T'	+ id * id $	-
$E'	+ id * id $	$T' \to \in$
$E'T+	+ id * id $	$E' \to +TE'$
$E'T	id * id $	-
$E'T'F	id * id $	$T \to FT'$
$E'T'id	id * id $	$F \to id$

$E'T'	*id$	-
$E'T'F*	*id$	$T' \rightarrow *FT'$
$E'T'F	id$	-
$E'T'id	id$	$F \rightarrow id$
$E'T'	$	-
$E'∈	$	$T' \rightarrow ∈$
$∈'	$	
$	$	

2) How can we convert Text to Bag of words, Text to TFIDF, Text to Ngrams with suitable examples

In NLP, text data needs to be converted into numbers so that ML Algorithm can understand it. One common method to do this is Bag of words (BoW) model. It turns next text like sentence, paragraph (or) document into a collection of words and counts how often each word appears but ignoring the order of words. It does not consider the order of words (or) their grammar but focus on counting how often each word appears in the text.

Key components of Bag of words :-

1) Vocabulary :- It is a list of all unique words from entire dataset each word in the vocabulary corresponds to a feature in the model.

2) Document Representation :- Each document is represented as a vector where each element shows the frequency of words from the vocabulary in that document. The frequency of each word is used as a feature for the model.

Eg:- Sent 1 → He is a good boy
Sent 2 → She is a good girl
Sent 3 → Boy and girl are good.

Removing stop words,

Sent 1 → good boy
Sent 2 → good girl
Sent 3 → boy girl good.

Words	frequency
good	3
boy	2
girl	2

Binary Bag of words.

	f_1	f_2	f_3	o/p.
	good	boy	girl	
S1	1	1	0	1
S2	1	0	1	1
S3	1	1	1	1

⟹ TF IDF :-

* Spelling errors are common in user generated text (Social media, chatbot).
* Errors can be Non word errors (There instead of their)
* Detection involves identifying words not found in reference are contextually wrong.

$$TF = \frac{\text{No. of repitition of words in sentence}}{\text{No. of words in sentence}}$$

$$IDF = \log_e\left(\frac{\text{No. of sentence}}{\text{No. of sentence containing words}}\right)$$

TF means Term frequency and
IDF means Inverse Document frequency.

Sentence 1 :- He is a good boy
Sentence 2 :- She is a good girl
Sentence 3 :- boy girl are good.

After lowering and stop words removing :-

S1 → good boy
S2 → good girl
S3 → boy girl are good.

TF	S1	S2	S3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

	f1 good	f2 boy	f3 girl
sent 1	0	$\frac{1}{2} \times \log_e(\frac{3}{2})$ $= 0.08$	0
sent 2	0	0	$\frac{1}{2} \times \log_e(\frac{3}{2})$ $= 0.08$.
Sent 3	0	$\frac{1}{3} \times \log_e(\frac{3}{2})$ $= 0.05$	$\frac{1}{3} \times \log_e(\frac{3}{2})$ $= 0.05$.

⇒ Next to ngrams

In NLP, ngrams are continuous sequences of n word from a given text.

A unigram considers only individual words.

A bigram considers two word sequences.

A Trigram consider three word sequences.

1) Unigram.

$$P(w) = \frac{\text{count}(w)}{\text{Total words}}.$$

2) Bigram

$$P(B/A) = \frac{\text{count}(AB)}{\text{count}(A)}$$

3) Trigram

$$P(C/A,B) = \frac{\text{count}(ABC)}{\text{count}(AB)}.$$

3) Write short note on

1) First 2) follow 3) Shift Reduce Passing 4) Syntax ambiguity.

1) First :—

To compile first(x) for all grammar symbols 'x', apply the following rules until no more terminals (or) ∈ (epsilon) can be added to only first.

Rules:-

→ If x is a terminal then first(x) is set {x}

→ If $x \to \in$ is a production then add $\in$ to first(x)

→ If x is a non terminal $x \to Y_1, Y_2, Y_3 \cdots Y_n$ is a production then place a first(x). If for some i in first(Y_i).

2) follow.

To compute follow(A) apply the following rules until nothing can be added to any followset

① $ in follow(S) where s is the start symbol and $ is the input right and marker

② If there is a production in the form of $A \to \alpha B \beta$ then everything in first(β) except for $\in$ is placed in follow(B)

③ If there is production $A \to \alpha B \beta$ (or) $A \to \alpha B$ containing $\in$ ($B \to \in$) then everything in follow(β) is in follow(B)

3) Shift reduce parsing.

* A shift reduce parser is a type of bottom up parser used in NLP & compiler construction.

* The process the input from left to right and uses a stack code and manages symbols.

* The parsing process involves 2 main operations shift and reduce.

* shift operation :- This involves moving the next input symbol onto the stack.

* Reduce operation :- This involves replacing a sequence of symbols on the stack with non-terminal symbol on the left hand side of rule.

4) Syntax ambiguity :-

Syntax ambiguity happens when a sentence can be parsed in more than 1 way Because its structure is unclear.

for eg:- I saw the man with glasses.
This has two meanings.

4) What are the differences between top down parsing and bottom up parsing with examples.

Top down parsing:-

1) Begins from the start symbol of the grammar and tries to derive the input string.
2) Derivation is from the root to leaves.
3) Parses tree from the top (start symbol) by expanding productions.
4) Used in recursive descent parsers, LL parsers
5) may involve backtracking (unless predictive parsing is used).

Bottom up parsing:-

1) Begins from the input string and tries to reduce it to the start symbol
2) Derivation is from leaves to root (reverse of rightmost derivation)
3) Builds the parse tree from the bottom, reducing substring to non terminals.
4) Used in shift reduced parsers, LR parsers.
5) Generally does not involve Backtracking

Examples for Top-down parsing:-

① Start with S
② Apply S→aSb: Now have aSb
③ Apply S→aSb: Now have aaSbb.
④ Apply S→∈ : Now have aabb.

⑤ Example for Bottom-up parsing:-

① look for substring matching the right side of rules
② Replace ∈ with nothing.

③ Find asb → Reduce asb to S
But we must build from bottom.

④ Start reducing.
* Identify ab as a Potential Reduction
* But there's no rule ab → S, so we try to reduce from aabb to S
* Parse tree construct in reverse, combining asb patterns.

5) What is meant by NLP? What are the challenges of NLP?

Natural Language Processing is a branch of AI that deals with the ability of machines to understand, analyze, manipulate and generate human language.

Challenges of NLP :-

① Ambiguity :-
* words or sentences can have multiple meanings
* Types :- Lexical, Syntactic, and Semantic ambiguity.

② Sarcasicm and Irony :-
* Difficult for machines to detect non literal expressions

③ Data Scarcity :-
* Lack of labelled data for many languages and domains.

④ Named Entity Recognition :-
* Identifying names, places, dates, etc., can be difficult due to variations.

⑤ Code Switching :-
* Mixing languages in one sentence

⑥ Bias and fairness :-
* NLP models may inherit societal biases from training data.

Applications of NLP:-

1. Machine Translation.
   eg:- Google, Translate, DeepL.
2. chatbots and virtual assistants
   eg:- Siri, Alexa, chatGpt, customer Service bots.
3. Sentiment Analysis
   Analyzing opinions in social media, reviews etc.
4. Speech Recognition
   converting spoken language into text.
5. Text Summarization.
   Creating concise summaries of longer documents
6. Question Answering Systems
   eg:- Search engines, AI tutors.
7. Spam Detection.
   Identifying and filtering out unwanted messages or emails
8. Text classification.
   categorizing Texet.