**NLP**

*Lower:*

Converts the data into lowercase using. lower()

*Stop words:*

These are words that are commonly filtered out before processing text data.

*Regular Expression:*

This has search and sub function.

While using search function if the text we are using to search matches the text in the data, then it shows match else it shows none.

Using Sub function, we can use to replace a word or text if the word or text is wrong.

*Tokenisation:*

Tokenisation is a process of breaking down sentence into words or words into characters.

*Stemming:*

It reduces the word to their base form.

*Lemmatization*:

Unlike stemming, which often simply cuts off word endings, lemmatization considers the context and morphological analysis of the words, ensuring that the root form is a valid word in the language.

*N-Grams:*

**N-GRAM is a sequence of n adjacent elements from a string of tokens.**

***POS Tagging:***

**It groups each word into it’s part of speech. Like whether a word is proper noun or adjective or noun.**

***Named entity recognition:***

**It groups words based on its name like whether it is an organisation or a date or a person etc.**

***Rule Based Sentiment:***

***Text Blob:***

**It decides the sentiment of the data based on index from -1 to 1. If the sentiment score is between 0 then sentiment is neutral. If the sentiment score is between 0 and 1 it is positive sentiment. If the sentiment score is between 0 and -1 it is negative sentiment**

***Vader:***

**It gives positive, negative and neutral scores for all the text.**

***Pre-Trained Transformer model:***

**It labels value as positive or negative or neutral and it also gives the index of how much it is positive or negative or neutral**

***Bag of words:***

It checks whether the word is there in the sentence and based on the result it will give 0 or 1 value in a matrix form for each word.

*Tf-idf:*

It evaluates the importance of a word in a document relative to a collection of documents in a matrix form.

It is a product of term frequency and Inverse document frequency.

Term Frequency:

Number of times term t appears in document dTotal number of terms in document /Total number of terms in document *d*

Inverse document frequency*:*

*Inverse document frequency:*

log(Total number of documents in corpus *D)*​(Number of documents containing term *t*)

*LDA:*

We go by iterations. In the first iteration we assign a document to some topic and in the second iteration we check the probability of those documents in that topic.

*LSA:*

Construct Term-Document Matrix: Create a matrix where each entry represents the frequency of a term in a document.

Apply SVD: Decompose the term-document matrix using SVD into three matrices: U, Σ, and V^T.

Reduce Dimensionality: Retain only the top k singular values and corresponding vectors to form a reduced representation of the original matrix.

**LLM**

What is LLM?

A large language model (LLM) is a type of artificial intelligence program that can recognize and generate text, among other tasks. LLMs are trained on huge sets of data — hence the name "large"

*LLM ARCHITECTURE:*

Transformer Architecture:

Transformer uses encoder and decoder architecture

Step 1: Create input embedding

- We first need to convert text into numerical representation

- Break down the input to tokens. Tokens can be words, sub words or characters

- Each token is mapped to a unique identifying number based on predefined vocabulary

- Once the tokens are mapped to their respective numbers, the model retrieves its pre-trained word embeddings

- Words that are similar to one another in meaning are mapped close to each other in vector

- However, the same word can have different meanings depending on how it's used in a sentence. This is where positional encoding comes in

Example:

Words | Number | Matrix

Love - 36 | 0.25|-0.15|0.88|...

I - 28 | 0.54|-0.19|0.68|...

Step 2: Positional encoding

- Positional encoding is given essentially where we give each token a number, recording the position of the token in the sentence

- This way we can send texts simultaneously instead of sequentially

Step 3: Multi Head attention layer

- Multi attention layer: It allows the model to weigh the importance of different tokens. It gives us attention vectors that captures contextual information between every token in the text

- For each token in the input sentence, we create three vectors. They are query vector, a key vector and a value vector.

- For each token in the sentence a query vector is calculated. This query vector represents the token's own question about the other tokens in the sentence. It's like how much attention it should pay to each of the other tokens

- Key vector holds information about all the other tokens in the sequence. Each key vector corresponds to a different token and contains information that helps determine how much attention should be given to that token

- Value vectors are associated with each token in the input sequence. They hold the actual content or knowledge about each token.

- Next is we need to calculate the similarity scores between each query vector and all key vectors using dot product. This produces a set of similarity scores, one for each token, indicating how much attention each token should receive from the currently attended to query token.

- To ensure stability, we scale the similarity scores by dividing them by the square root of dimension of the key vectors. This scaling factor helps control the magnitude of the scores

- SoftMax function is applied to obtain the attention weight of scaled similarity scores

- This SoftMax function is called attention weights where it indicates how much attention query token should given to each of the token

Step 4: Feed forward layer

- Input for feed forward layer contains information about how the token relates to other tokens in the sequence

- Linear transformation of the input information is the first step in feed forward layer. This means reshaping and projecting input into a new space with potentially higher dimensions

- This transformation is usually followed by activation function, which introduces non-linearity into the model

- Another linear transformation is applied, this transformation reshaped and projects the data further, often reducing the dimensionality.

- The output of this second linear transformation is the final representation of each token.

Summary of step 4:

- Acts as a set of neural network operations applied independently to each token representation

- Helps the transformer model capture and encode complex, on-linear relationships between token in the input sequence

- Applied independently

- It can be run in parallel

Step 5: Masked multi-head attention

- The desired outputs we want our model to learn are fed into decoder block

- Let us take an example of translating from French to English. French will be fed as input in encoder block and English would be fed in the decoder block.

- These outputs go through similar embedding transformations and get their positional encodings before being fed to the decoder block.

- However we don't feed all output embeddings to the decoder block at once. They go through the masked multi-head attention layer which is similar to multi-head attention layer which was there in the encoder block but we say it as masked because we need to hide some of the information on the outputs at this stage to make the model learn.

- Going back to the example, we allow the model to see all the French input words, and words here really means the attention vectors for these words.

- But the model will only see the English words that have come before the currently attended to word in the sentence.

- The model has to learn what the correct next word would be rather than just looking at the next word in the given outputs

Step 6: Multi-head attention in decoder

- The multi head attention in the decoder receives input from both the encoder block and the masked multi head attention layer.

- In addition to self attention in encoder, multi-head attention also calculates attention scores between the current output token and the encoder outputs

- This steps helps the model determine which parts of the input sequence are relevant when generating the next token for the output

Step 7: Context vector

- This context vector represents the relevant information from the input sequence that should be considered when generating the current output token

Step 8: The feed forward neural network again makes the output more digestible for the next step in the process

Step 9: Linear layer is another feed forward neural network, used again to manipulate the output into a useful format

Step 10: The SoftMax layer transforms the output into a probability distribution. This probability distribution is then used to predict the next word in the output sequence

Step 11: The whole process of this decoder block is repeated for each token in the output sentence, with the previously generated tokens being included as token for each subsequent step.

**PROMPT ENGINEERING**

Temperature - It is a value range from 0 to 1. It controls the randomness of the model. If the value is near 0 then it is less random. If the value is nearer to 1 then it is more random

Top P - This is a sampling technique with temperature, called nucleus sampling. If we are looking for exact and factual answers we need to keep this low. If we are looking for more diverse responses then we need increase to a higher value

Max Length - We can manager the number of tokens the model generates by adjusting the max length

Stop sequence - Specifying stop sequence is another way to control the length and the structure of the model's response

Frequency penalty - It applies a penalty on the next token proportional to how many times that token already appeared in the response and prompt. The higher the frequency penalty, the less likely a word will appear again.

Presence Penalty - The presence penalty also applies a penalty on repeated tokens but, unlike the frequency penalty, the penalty is the same for all repeated tokens. A token that appears twice and a token that appears 10 times are penalized the same. If you want the model to generate diverse or creative text, you might want to use a higher presence penalty. Or, if you need the model to stay focused, try using a lower presence penalty.

System Role:

The System role is used to provide setup information or context that informs the behavior of the model.

Example: System: The assistant should always maintain a professional tone and avoid discussing personal opinions on politics.

User role:

This role represents the human user in the conversation. Inputs from the user guide the conversation and prompt responses from the assistant.

Example: User: Can you explain how to integrate Open Ai's API with my existing Python application?

Assistant Role:

This is the role of the model itself, responding to user inputs based on the context set by the system.

*Zero Shot Prompting:*

Zero-shot prompting means that the prompt used to interact with the model won't contain examples or demonstrations. We will give a text and ask it do something with it without giving any example on how to perform the task

*Few Shot Prompting:*

Few-shot prompting can be used as a technique to enable in-context learning where we provide demonstrations in the prompt to steer the model to better performance. The demonstrations serve as conditioning for subsequent examples where we would like the model to generate a response.

*Chain of thought:*

Chain-of-thought (CoT) prompting is a technique that allows large language models to solve a problem as a series of intermediate steps before giving a final answer.

Keyword: Let's think step by step.

*Self-Consistency:*

Self-consistency simply asks a model the same prompt multiple times and takes the majority result as the final answer.

*General Knowledge prompting:*

General Knowledge Prompting can initially yield incorrect responses based on how a question is phrased. A second, more informed prompt can correct and refine the AI's response, demonstrating the importance of context and clarification in prompting.

This two-step approach underlines the importance of context and the manner in which questions are posed to ensure accurate responses from General Knowledge Prompting.

*Prompt Chaining:*

It breaks tasks into its subtasks. Once those subtasks have been identified, the LLM is prompted with a subtask and then its response is used as input to another prompt.

*Tree of thoughts:*

Thought decomposition: It breaks the complex ideas into smaller components and organize them in ranking structure.

Thought generation: The process by which generation model generates potential thoughts. This generates a set of thoughts that could potentially lead to complete solution. Key word while giving prompt is "possible next steps”.

State evaluator: So here the language model evaluates different states or partial solutions so that it can determine potential usefulness in the problem. The evaluator assigns a value to each state so that the algorithm can decide which state to explore further based on the evaluation.

// Search Algorithm: The search algorithm determines which state or partial solution to explore and in what order based on the value assigned by the state evaluator. Different search algorithm can be use within ToT framework depending on the structure of the tree and the nature of the problem. Eg. Depth first or breadth first algorithm.

*Retrieval Augmented Generation:*

Input Query: The user inputs a query or prompt.

Retrieval: The retrieval component searches a large corpus to find relevant documents or data.

Generation: The generation component uses the retrieved information to generate a contextually appropriate response.

// Output: The final output is presented to the user, combining the strengths of both retrieval and generation.

*Automatic reasoning and tool use:*

Automatic reasoning tools facilitate the process of logical reasoning and problem-solving by automating the inference process. Also ART is extensible as it also enables humans to fix mistakes in the reasoning steps or add new tools by simply updating the task and tool libraries.

*Automatic prompt engineering:*

1. Prompt Generation: The system generates initial prompts based on

predefined criteria or templates

2. Response Generation: The language model generates responses based on

the initial prompts

3. Prompt Evaluation: The generated prompts are evaluated based on the

quality of the responses they elicit from the language model. Evaluation

metrics can include relevance, coherence, informativeness, and user

satisfaction.

4. Prompt Optimization: Based on the evaluation results, the system refines

and optimizes the prompts to improve their effectiveness. Optimization

techniques can include reinforcement learning, genetic algorithms, or other

machine learning methods.

*Active prompt engineering:*

The first step is to query the LLM with or without a few CoT examples. k possible answers are generated for a set of training questions. An uncertainty metric is calculated based on the k answers (disagreement used). The most uncertain questions are selected for annotation by human.

*Direction stimulus:*

Directional stimulus prompting is a technique used to guide behaviour or responses by providing specific cues or stimuli.

*Programming aided language:*

It is mixture of few shot and chain of thought. A sample input is given step by step but here the step-by-step process is written as code. Then we will be giving another input as we do in few shot.

*Reflexion:*

This involves the system's ability to analyse its own outputs, learn from interactions, and modify its approach to improve the quality and relevance of its responses.

*Example*:

How to reset my password?

How to reset my password? I didn’t receive password reset mail

*Multimodal CoT prompting:*

Multimodal CoT incorporates text and vision into a two-stage framework. The first step involves rationale generation based on multimodal information. This is followed by the second phase, answer inference, which leverages the informative generated rationales.

