CS2V14 – TEXT AND SPEECH ANALYSIS (PE) 2024-2025 ODD SEM
UNIT 1 – BASICS OF NATURAL LANGUAGE PROCESSING

Foundations of natural language processing – Language Syntax and Structure- Text Preprocessing and Wrangling – Text tokenization – Stemming – Lemmatization – Removing stop-words – Feature Engineering for Text representation – Bag of Words model- Bag of N-Grams model – TF-IDF model

INTRODUCTION

Artificial intelligence (AI) integration has revolutionized various industries, and now it is transforming the realm of human behaviour research. This integration marks a significant milestone in the data collection and analysis endeavors, enabling users to unlock deeper insights from spoken language and empower researchers and analysts with enhanced capabilities for understanding and interpreting human communication. Human interactions are a critical part of many organizations. Many organizations analyze speech or text via natural language processing (NLP) and link them to insights and automation such as text categorization, text classification, information extraction, etc.

In business intelligence, speech and text analytics enable us to gain insights into customer-agent conversations through sentiment analysis, and topic trends. These insights highlight areas of improvement, recognition, and concern, to better understand and serve customers and employees. Speech and text analytics features provide automated speech and text analytics capabilities on 100% of interactions to provide deep insight into customer-agent conversations. Speech and text analytics is a set of features that uses natural language processing (NLP) to provide an automated analysis of an interaction's content and insight into customer-agent conversations. Speech and text analytics includes transcribing voice interactions, analysis for customer sentiment and topic spotting, and creating meaning from otherwise unstructured data.

FOUNDATIONS OF NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) is the process of producing meaningful phrases and sentences in the form of natural language. Natural Language Processing precludes Natural Language Understanding (NLU) and Natural Language

Generation (NLG). NLU takes the data input and maps it into natural language. NLG conducts information extraction and retrieval, sentiment analysis, and more. NLP can be thought of as an intersection of Linguistics, Computer Science and Artificial Intelligence that helps computers understand, interpret and manipulate human language.

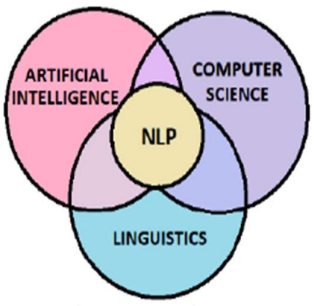


Fig. NLP Overview

Ever since then, there has been an immense amount of study and development in the field of Natural Language Processing.

Today NLP is one of the most in-demand and promising fields of Artificial Intelligence! There are two main parts to Natural Language Processing:

1. Data Preprocessing
2. Algorithm Development

Applications	Core technologies
Machine Translation	Language modelling
Information Retrieval	Part-of-speech tagging
Question Answering	Syntactic parsing
Dialogue Systems	Named-entity recognition
Information Extraction	Coreference resolution
Summarization	Word sense disambiguation
Sentiment Analysis	Semantic Role Labelling

In Natural Language Processing, machine learning training algorithms study millions of examples of text — words, sentences, and paragraphs — written by humans. By studying the samples, the training algorithms gain an understanding of the "context" of human speech, writing, and other modes of communication. This training helps NLP software to differentiate between the meanings of various texts. The five phases of NLP involve lexical (structure) analysis, parsing, semantic analysis, discourse integration, and pragmatic analysis. Some well-known application areas of NLP are Optical Character Recognition (OCR), Speech Recognition, Machine Translation, and Chatbots.

Fig: Five Phases of NLP

Phase I: Lexical or morphological analysis

The first phase of NLP is word structure analysis, which is referred to as lexical or morphological analysis. A lexicon is defined as a collection of words and phrases in a given language, with the analysis of this collection being the process of splitting the lexicon into components, based on what the user sets as parameters – paragraphs, phrases, words, or characters.

Similarly, morphological analysis is the process of identifying the morphemes of a word. A morpheme is a basic unit of English language construction, which is a small element of a word, that carries meaning. These can be either a free morpheme (e.g. walk) or a bound morpheme (e.g. -ing, -ed), with the difference between the two being that the latter cannot stand on it's own to produce a word with meaning, and should be assigned to a free morpheme to attach meaning.

In search engine optimization (SEO), lexical or morphological analysis helps guide web searching. For instance, when doing on-page analysis, you can perform lexical and morphological analysis to understand how often the target keywords are used in their core form (as free morphemes, or when in composition with bound morphemes). This type of analysis can ensure that you have an accurate understanding of the different variations of the morphemes that are used. Morphological analysis can also be applied in transcription and translation projects, so can be very useful in content repurposing projects, and international SEO and linguistic analysis.

Phase II: Syntax analysis (parsing)

Syntax Analysis is the second phase of natural language processing. Syntax analysis or parsing is the process of checking grammar, word arrangement, and overall – the identification of relationships between words and whether those make sense. The process involved examination of all words and phrases in a sentence, and the structures between them.

As part of the process, there's a visualisation built of semantic relationships referred to as a syntax tree (similar to a knowledge graph). This process ensures that the structure and order and grammar of sentences makes sense, when considering the words and phrases that make up those sentences. Syntax analysis also involves tagging words and phrases with POS tags. There are two common methods, and multiple approaches to construct the syntax tree – top-down and bottom-up, however, both are logical and check for sentence formation, or else they reject the input.

Syntax analysis can be beneficial for SEO in several ways:

• Programmatic SEO: Checking whether the produced content makes sense, especially when producing content at scale using an automated or semi-automated approach.

• Semantic analysis: Once you have a syntax analysis conducted, semantic analysis is easy, as well as uncovering the relationship between the different entities recognized in the content.

Phase III: Semantic analysis

Semantic analysis is the third stage in NLP, when an analysis is performed to understand the meaning in a statement. This type of analysis is focused on uncovering the definitions of words, phrases, and sentences and identifying whether the way words are organized in a sentence makes sense semantically.

This task is performed by mapping the syntactic structure, and checking for logic in the presented relationships between entities, words, phrases, and sentences in the text. There are a couple of important functions of semantic analysis, which allow for natural language understanding:

• To ensure that the data types are used in a way that's consistent with their definition.

• To ensure that the flow of the text is consistent.

• Identification of synonyms, antonyms, homonyms, and other lexical items.

• Overall word sense disambiguation.

• Relationship extraction from the different entities identified from the text.

There are several things you can utilise semantic analysis for in SEO. Here are some examples:

• Topic modeling and classification – sort your page content into topics (predefined or modelled by an algorithm). You can then use this for ML-enabled internal linking, where you link pages together on your website using the identified topics. Topic modeling can also be used for classifying first-party collected data such as customer service tickets, or feedback users left on your articles or videos in free form (i.e. comments).

• Entity analysis, sentiment analysis, and intent classification – You can use this type of analysis to perform sentiment analysis and identify intent expressed in the content analysed. Entity identification and sentiment analysis are separate tasks, and both can be done on things like keywords, titles, meta descriptions, page content, but works best when analysing data like comments, feedback forms, or customer service or social media interactions. Intent classification can be done on user queries (in keyword research or traffic analysis), but can also be done in analysis of customer service interactions.

Phase IV: Discourse integration

Discourse integration is the fourth phase in NLP, and simply means contextualisation. Discourse integration is the analysis and identification of the larger context for any smaller part of natural language structure (e.g. a phrase, word or sentence).

During this phase, it's important to ensure that each phrase, word, and entity mentioned are mentioned within the appropriate context. This analysis involves considering not only sentence structure and semantics, but also sentence combination and meaning of the text as a whole. Otherwise, when analyzing the structure of text, sentences are broken up and analyzed and also considered in the context of the sentences that precede and follow them, and the impact that they have on the structure of text. Some common tasks in this phase include: information extraction, conversation analysis, text summarisation, discourse analysis.

Here are some complexities of natural language understanding introduced during this phase:

• Understanding of the expressed motivations within the text, and its underlying meaning.

• Understanding of the relationships between entities and topics mentioned, thematic understanding, and interactions analysis.

• Understanding the social and historical context of entities mentioned.

Discourse integration and analysis can be used in SEO to ensure that appropriate tense is used, that the relationships expressed in the text make logical sense, and that there is overall coherency in the text analysed. This can be especially useful for programmatic SEO initiatives or text generation at scale. The analysis can also be used as part of international

SEO localization, translation, or transcription tasks on big corpuses of data.

There are some research efforts to incorporate discourse analysis into systems that detect hate speech (or in the SEO space for things like content and comment moderation), with this technology being aimed at uncovering intention behind text by aligning the expression with meaning, derived from other texts. This means that, theoretically, discourse analysis can also be used for modeling of user intent (e.g search intent or purchase intent) and detection of such notions in texts.

Phase V: Pragmatic analysis

Pragmatic analysis is the fifth and final phase of natural language processing. As the final stage, pragmatic analysis extrapolates and incorporates the learnings from all other, preceding phases of NLP. Pragmatic analysis involves the process of abstracting or extracting meaning from the use of language, and translating a text, using the gathered knowledge from all other NLP steps performed beforehand.

Here are some complexities that are introduced during this phase

• Information extraction, enabling an advanced text understanding functions such as question-answering.

• Meaning extraction, which allows for programs to break down definitions or documentation into a more accessible language.

• Understanding of the meaning of the words, and context, in which they are used, which enables conversational functions between machine and human (e.g. chatbots).

Pragmatic analysis has multiple applications in SEO. One of the most straightforward ones is programmatic SEO and automated content generation. This type of analysis can also be used for generating FAQ sections on your product, using textual analysis of product documentation, or even capitalizing on the 'People Also Ask' featured snippets by adding an automatically-generated FAQ section for each page you produce on your site.

LANGUAGE SYNTAX AND STRUCTURE

For any language, syntax and structure usually go hand in hand, where a set of specific rules, conventions, and principles govern the way words are combined into phrases; phrases get combines into clauses; and clauses get combined into sentences. We will be talking specifically about the English language syntax and structure in this section. In English, words usually combine together to form other constituent units. These constituents include words, phrases, clauses, and sentences. Considering a sentence, "The brown fox is quick and he is jumping over the lazy dog", it is made of a bunch of words and just looking at the words by themselves don't tell us much.

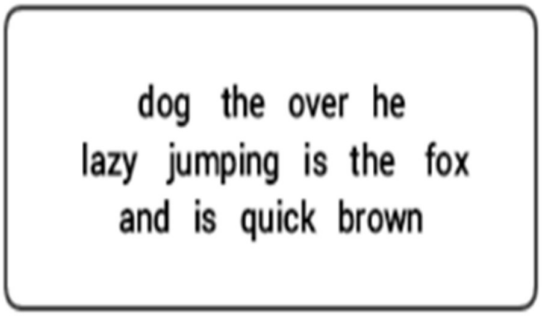


Fig. A bunch of unordered words don't convey much information

Knowledge about the structure and syntax of the language is helpful in many areas like text processing, annotation, and parsing for further operations such as text classification or summarization. Typical parsing techniques for understanding text syntax are mentioned below.

• Parts of Speech (POS) Tagging

• Shallow Parsing or Chunking

• Constituency Parsing

• Dependency Parsing

We will be looking at all of these techniques in subsequent sections. Considering the previous example sentence “The brown fox is quick and he is jumping over the lazy dog”, if we were to annotate it using basic POS tags, it would look like the following figure.

Fig. POS tagging for a sentence

Thus, a sentence typically follows a hierarchical structure consisting the following components,

sentence → clauses → phrases → words

Tagging Parts of Speech

Parts of speech (POS) are specific lexical categories to which words are assigned, based on their syntactic context and role.

Usually, words can fall into one of the following major categories.

• **N(oun):** This usually denotes words that depict some object or entity, which may be living or nonliving. Some

examples would be fox , dog , book , and so on. The POS tag symbol for nouns is N.

• **V(erb):** Verbs are words that are used to describe certain actions, states, or occurrences. There are a wide variety of further subcategories, such as auxiliary, reflexive, and transitive verbs (and many more). Some typical examples of verbs would be running , jumping , read , and write . The POS tag symbol for verbs is V.

• **Adj(ective):** Adjectives are words used to describe or qualify other words, typically nouns and noun phrases. The phrase beautiful flower has the noun (N) flower which is described or qualified using the adjective (ADJ) beautiful . The POS tag symbol for adjectives is ADJ .

• **Adv(erb):** Adverbs usually act as modifiers for other words including nouns, adjectives, verbs, or other adverbs. The phrase very beautiful flower has the adverb (ADV) very , which modifies the adjective (ADJ) beautiful , indicating the degree to which the flower is beautiful. The POS tag symbol for adverbs is ADV. Besides these four major categories of parts of speech , there are other categories that occur frequently in the English language. These include pronouns, prepositions, interjections, conjunctions, determiners, and many others. Furthermore,each POS tag like the noun (N) can be further subdivided into categories like singular nouns (NN), singular proper nouns(NNP), and plural nouns (NNS).

The process of classifying and labeling POS tags for words called parts of speech tagging or POS tagging . POS tags are used to annotate words and depict their POS, which is really helpful to perform specific analysis, such as narrowing down upon nouns and seeing which ones are the most prominent, word sense disambiguation, and grammar analysis.

Let us consider both nltk and spacy which usually use the Penn Treebank notation for POS tagging. NLTK and spaCy are two of the most popular Natural Language Processing (NLP) tools available in Python. You can build chatbots, automatic summarizers, and entity extraction engines with either of these libraries. While both can theoretically accomplish any NLP task, each one excels in certain scenarios. The Penn Treebank, or PTB for short, is a dataset maintained by the University of Pennsylvania.

```
# create a basic pre-processed corpus, don't lowercase to get POS context
corpus = normalize_corpus(news_df['full_text'], text_lower_case=False,
                          text_lemmatization=False, special_char_removal=False)
# demo for POS tagging for sample news headline
sentence = str(news_df.iloc[1].news_headline)
sentence_nlp = nlp(sentence)
# POS tagging with Spacy
spacy_pos_tagged = [(word, word.tag_, word.pos_) for word in sentence_nlp]
pd.DataFrame(spacy_pos_tagged, columns=['Word', 'POS tag', 'Tag type'])
# POS tagging with nltk
nltk_pos_tagged = nltk.pos_tag(sentence.split())
pd.DataFrame(nltk_pos_tagged, columns=['Word', 'POS tag'])
```

	Word	POS tag	Tag type
0	US	NNP	PROPN
1	unveils	VBZ	VERB
2	world	NN	NOUN
3	's	POS	PART
4	most	RBS	ADV
5	powerful	JJ	ADJ
6	supercomputer	NN	NOUN
7	,	,	PUNCT
8	beats	VBZ	VERB
9	China	NNP	PROPN

SpaCy POS tagging

	Word	POS tag
0	US	NNP
1	unveils	VBZ
2	world's	VBZ
3	most	RBS
4	powerful	JJ
5	supercomputer,	JJ
6	beats	NNS
7	China	NNP

NLTK POS tagging

Fig. Python code & Output of POS tagging a news headline

We can see that each of these libraries treat tokens in their own way and assign specific tags for them. Based on what we see, spacy seems to be doing slightly better than nltk.

Shallow Parsing or Chunking

Based on the hierarchy we depicted earlier, groups of words make up phrases. There are five major categories of phrases:

Noun phrase (NP): These are phrases where a noun acts as the head word. Noun phrases act as a subject or object to a verb.

Verb phrase (VP): These phrases are lexical units that have a verb acting as the head word. Usually, there are two forms of verb phrases. One form has the verb components as well as other entities such as nouns, adjectives, or adverbs as parts of the object.

Adjective phrase (ADJP): These are phrases with an adjective as the head word. Their main role is to describe or qualify nouns and pronouns in a sentence, and they will be either placed before or after the noun or pronoun.

Adverb phrase (ADVP): These phrases act like adverbs since the adverb acts as the head word in the phrase. Adverb phrases are used as modifiers for nouns, verbs, or adverbs themselves by providing further details that describe or qualify them.

Prepositional phrase (PP): These phrases usually contain a preposition as the head word and other lexical components like nouns, pronouns, and so on. These act like an adjective or adverb describing other words or phrases.

Shallow parsing, also known as light parsing or chunking, is a popular natural language processing technique of analyzing the structure of a sentence to break it down into its smallest constituents (which are tokens such as words) and group them together into higher-level phrases. This includes POS tags and phrases from a sentence

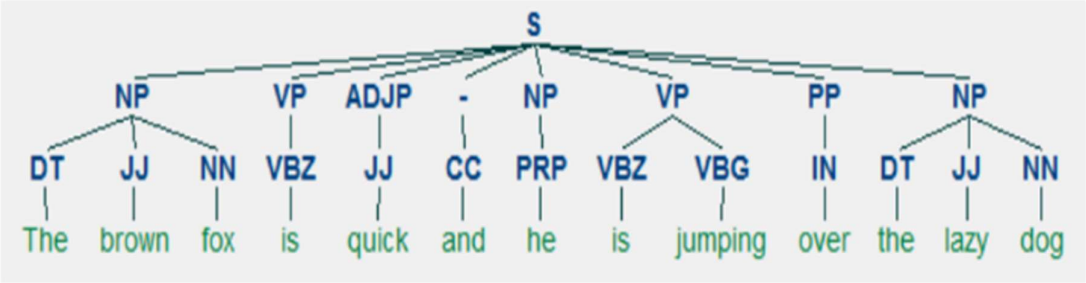


Fig. An example of shallow parsing depicting higher level phrase annotations

Constituency Parsing

Constituent-based grammars are used to analyze and determine the constituents of a sentence. These grammars can be used to model or represent the internal structure of sentences in terms of a hierarchically ordered structure of their constituents.

Each and every word usually belongs to a specific lexical category in the case and forms the head word of different phrases. These phrases are formed based on rules called phrase structure rules.

Phrase structure rules form the core of constituency grammars, because they talk about syntax and rules that govern the hierarchy and ordering of the various constituents in the sentences. These rules cater to two things primarily.

• They determine what words are used to construct the phrases or constituents.

• They determine how we need to order these constituents together.

The generic representation of a phrase structure rule is S → AB , which depicts that the structure S consists of constituents A and B , and the ordering is A followed by B .

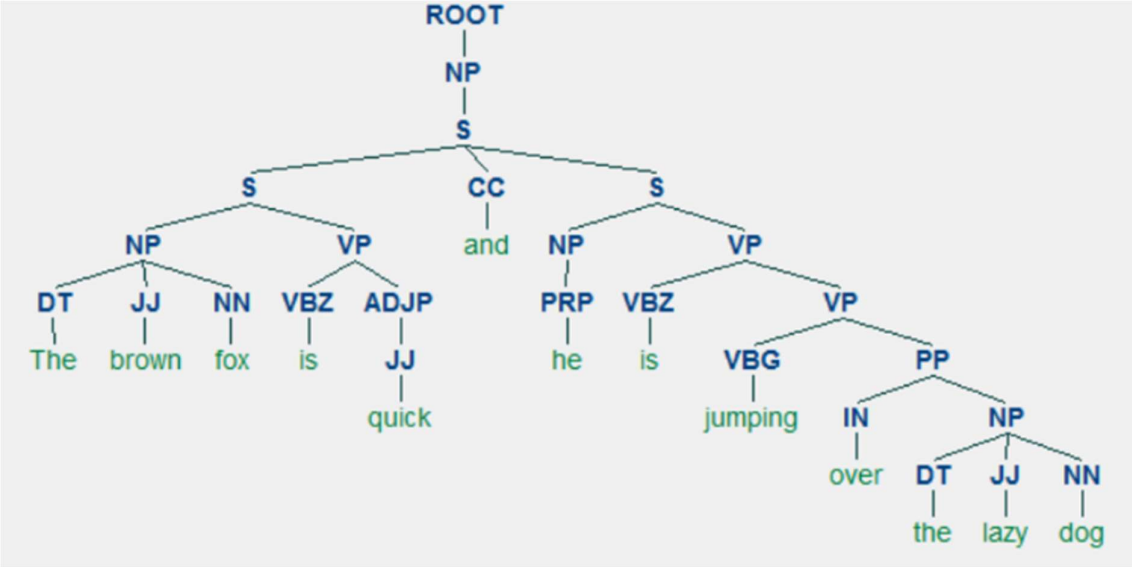


Fig. An example of constituency parsing showing a nested hierarchical structure

Dependency Parsing

In dependency parsing, we try to use dependency-based grammars to analyze and infer both structure and semantic dependencies and relationships between tokens in a sentence. The basic principle behind a dependency grammar is that in any sentence in the language, all words except one, have some relationship or dependency on other words in the sentence.

The word that has no dependency is called the root of the sentence. The verb is taken as the root of the sentence in most cases. All the other words are directly or indirectly linked to the root verb using links, which are the dependencies.

Considering the sentence “The brown fox is quick and he is jumping over the lazy dog”, if we wanted to draw the dependency syntax tree for this, we would have the structure.

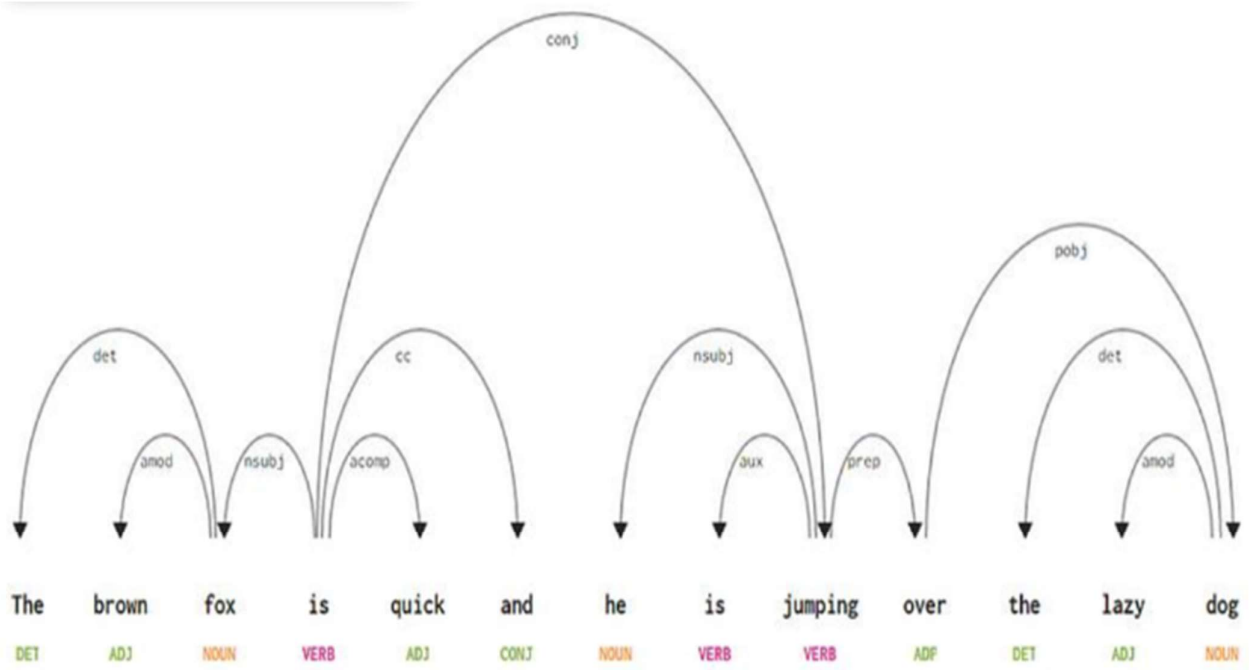


Fig. A dependency parse tree for a sentence

These dependency relationships each have their own meaning and are a part of a list of universal dependency types.

Some of the dependencies are as follows:

• The dependency tag det

is pretty intuitive—it denotes the determiner relationship between a nominal head and the determiner. Usually, the word with POS tag DET will also have the det dependency tag relation. Examples include fox → the and dog → the.

• The dependency tag amod stands for adjectival modifier and stands for any adjective that modifies the meaning of a noun. Examples include fox → brown and dog → lazy.

• The dependency tag nsubj stands for an entity that acts as a subject or agent in a clause. Examples include is → fox and jumping → he.

• The dependencies cc and conj have more to do with linkages related to words connected by coordinating conjunctions. Examples include is → and and is → jumping.

• The dependency tag aux indicates the auxiliary or secondary verb in the clause. Example: jumping → is.

• The dependency tag acomp stands for adjective complement and acts as the complement or object to a verb in the sentence. Example: is → quick

• The dependency tag prep denotes a prepositional modifier, which usually modifies the meaning of a noun, verb, adjective, or preposition. Usually, this representation is used for prepositions having a noun or noun phrase complement. Example: jumping → over.

• The dependency tag pobj is used to denote the object of a preposition. This is usually the head of a noun phrase following a preposition in the sentence. Example: over → dog.

TEXT PREPROCESSING OR WRANGLING

Text preprocessing or wrangling is a method to clean the text data and make it ready to feed data to the model. Text data contains noise in various forms like emotions, punctuation, text in a different case. When we talk about Human Language then, there are different ways to say the same thing, And this is only the main problem we have to deal with because machines will not understand words, they need numbers so we need to convert text to numbers in an efficient manner.

Techniques to perform text preprocessing or wrangling are as follows:

Contraction Mapping/ Expanding Contractions: Contractions are a shortened version of words or a group of words, quite common in both spoken and written language. In English, they are quite common, such as I will to I'll, I have to I've , do not to don't, etc. Mapping these contractions to their expanded form helps in text standardization.

Tokenization: Tokenization is the process of separating a piece of text into smaller units called tokens. Given a document, tokens can be sentences, words, subwords, or even characters depending on the application. Noise cleaning: Special characters and symbols contribute to extra noise in unstructured text. Using regular expressions to remove them or using tokenizers, which do the pre-processing step of removing punctuation marks and other special characters, is recommended.

Spell-checking: Documents in a corpus are prone to spelling errors; In order to make the text clean for the subsequent processing, it is a good practice to run a spell checker and fix the spelling errors before moving on to the next steps.

Stopwords Removal: Stop words are those words which are very common and often less significant. Hence, removing these is a pre-processing step as well. This can be done explicitly by retaining only those words in the document which are not in the list of stop words or by specifying the stop word list as an argument in CountVectorizer or TfidfVectorizer methods when getting Bag-of-Words(BoW)/TF-IDF scores for the corpus of text documents.

Stemming/Lemmatization: Both stemming and lemmatization are methods to reduce words to their base form. While stemming follows certain rules to truncate the words to their base form, often resulting in words that are not lexicographically correct, lemmatization always results in base forms that are lexicographically correct. However, stemming is a lot faster than lemmatization. Hence, to stem/lemmatize is dependent on whether the application needs quick pre-processing or requires more accurate base forms.

TOKENIZATION

Tokenization is a common task in Natural Language Processing (NLP). It's a fundamental step in both traditional NLP methods like Count Vectorizer and Advanced Deep Learning-based architectures like Transformers. As tokens are the building blocks of Natural Language, the most common way of processing the raw text happens at the token level.

Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. Hence, ***tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization.***

For example, consider the sentence: "Never give up".

The most common way of forming tokens is based on space. Assuming space as a delimiter, the tokenization of the sentence results in 3 tokens – Never-give-up. As each token is a word, it becomes an example of Word tokenization.

Similarly, tokens can be either characters or sub-words. For example, let us consider "smarter":

1. ***Character tokens***: s-m-a-r-t-e-r

2. ***Sub-word tokens***: smart-er

Here, Tokenization is performed on the corpus to obtain tokens. The following tokens are then used to prepare a vocabulary.

Vocabulary refers to the set of unique tokens in the corpus. Remember that vocabulary can be constructed by considering each unique token in the corpus or by considering the top K Frequently Occurring Words.

Creating Vocabulary is the ultimate goal of Tokenization.

Word Tokenization

Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed. Pretrained Word Embeddings such as Word2Vec and GloVe comes under word tokenization.

Drawbacks of Word Tokenization

One of the major issues with word tokens is dealing with Out Of Vocabulary (OOV) words. OOV words refer to the new words which are encountered at testing. These new words do not exist in the vocabulary. Hence, these methods fail in handling Out-of-Vocabolary (OOV) words.

• A small trick can rescue word tokenizers from OOV words. The trick is to form the vocabulary with the Top K Frequent Words and replace the rare words in training data with **unknown tokens (UNK)**. This helps the model to learn the representation of OOV words in terms of UNK tokens

• So, during test time, any word that is not present in the vocabulary will be mapped to a UNK token. This is how we can tackle the problem of OOV in word tokenizers.

• The problem with this approach is that the entire information of the word is lost as we are mapping OOV to UNK tokens. The structure of the word might be helpful in representing the word accurately. And another issue is that every OOV word gets the same representation

Another issue with word tokens is connected to the size of the vocabulary. Generally, pre-trained models are trained on a large volume of the text corpus. So, just imagine building the vocabulary with all the unique words in such a large corpus. This explodes the vocabulary!

Character Tokenization

Character Tokenization splits apiece of text into a set of characters. It overcomes the drawbacks we saw above about Word Tokenization.

• Character Tokenizers handles OOV words coherently by preserving the information of the word. It breaks down

the OOV word into characters and represents the word in terms of these characters

• It also limits the size of the vocabulary. Want to talk a guess on the size of the vocabulary? 26 since the vocabulary contains a unique set of characters

Drawbacks of Character Tokenization

Character tokens solve the OOV problem but the length of the input and output sentences increases rapidly as we are representing a sentence as a sequence of characters. As a result, it becomes challenging to learn the relationship between the characters to form meaningful words. This brings us to another tokenization known as Subword Tokenization which is in between a Word and Character tokenization.

Subword Tokenization

Subword Tokenization splits the piece of text into subwords (or n-gram characters). For example, words like lower can be segmented as low-er, smartest as smart-est, and so on. Transformed-based models – the SOTA in NLP – rely on Subword Tokenization algorithms for preparing vocabulary.

Byte Pair Encoding (BPE)

Byte Pair Encoding (BPE) is a widely used tokenization method among transformer-based models. BPE addresses the issues of Word and Character Tokenizers. BPE tackles OOV effectively. It segments OOV as subwords and represents the word in terms of these subwords

- The length of input and output sentences after BPE are shorter compared to character tokenization
- BPE is a word segmentation algorithm that merges the most frequently occurring character or character sequences iteratively.

BPE is a word segmentation algorithm that merges the most frequently occurring character or character sequences
iteratively. Here is a step by step guide to learn BPE.
1. Split the words in the corpus into characters after appending </w>
2. Initialize the vocabulary with unique characters in the corpus
3. Compute the frequency of a pair of characters or character sequences in corpus
4. Merge the most frequent pair in corpus
5. Save the best pair to the vocabulary
6. Repeat steps 3 to 5 for a certain number of iterations

STEMMING

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”. Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words.

How do we get these tokenized words? Well, tokenization involves breaking down the document into different words.

Stemming is a natural language processing technique that is used to reduce words to their base form, also known as the root form. The process of stemming is used to normalize text and make it easier to process. It is an important step in text pre-processing, and it is commonly used in information retrieval and text mining applications. There are several different algorithms for stemming as follows:

- Porter stemmer
- Snowball stemmer
- Lancaster stemmer.

The Porter stemmer is the most widely used algorithm, and it is based on a set of heuristics that are used to remove common suffixes from words. The Snowball stemmer is a more advanced algorithm that is based on the Porter stemmer, but it also supports several other languages in addition to English. The Lancaster stemmer is a more aggressive stemmer and it is less accurate than the Porter stemmer and Snowball stemmer. Stemming can be useful for several natural language processing tasks such as text classification, information retrieval, and text summarization. However, stemming can also have some negative effects such as reducing the readability of the text, and it may not always produce the correct root form of a word. It is important to note that stemming is different from Lemmatization. Lemmatization is the process of reducing a word to its base form, but unlike stemming, it takes into account the context of the word, and it produces a valid word, unlike stemming which can produce a non-word as the root form. Some more examples stemming from the root word "like" include:

->"likes"
->"liked"
->"likely"
->"liking"

Errors in Stemming:

There are mainly two errors in stemming –
- over-stemming
- under-stemming

Over-stemming occurs when two words are stemmed from the same root that are of different stems. Over-stemming can also be regarded as false-positives. Over-stemming is a problem that can occur when using stemming algorithms in natural language processing. It refers to the situation where a stemmer produces a root form that is not a valid word or is not the correct root form of a word. This can happen when the stemmer is too aggressive in removing suffixes or when it does not consider the context of the word.
Over-stemming can lead to a loss of meaning and make the text less readable. For example, the word “arguing” may be stemmed to “argu,” which is not a valid word and does not convey the same meaning as the original word. Similarly, the word “running” may be stemmed to “run,” which is the base form of the word but it does not convey the meaning of the original word.
To avoid over-stemming, it is important to use a stemmer that is appropriate for the task and language. It is also important to test the stemmer on a sample of text to ensure that it is producing valid root forms. In some cases, using a lemmatizer instead of a stemmer may be a better solution as it takes into account the context of the word, making it less prone to errors.
Another approach to this problem is to use techniques like semantic role labeling, sentiment analysis, context-based information, etc. that help to understand the context of the text and make the stemming process more precise.

Under-stemming occurs when two words are stemmed from the same root that are not of different stems. Under-stemming can be interpreted as false-negatives. Under-stemming is a problem that can occur when using stemming algorithms in natural language processing. It refers to the situation where a stemmer does not produce the correct root form of a word or does not reduce a word to its base form. This can happen when the stemmer is not aggressive enough in removing suffixes or when it is not designed for the specific task or language.
Under-stemming can lead to a loss of information and make it more difficult to analyze text. For example, the word “arguing” and “argument” may be stemmed to “argu,” which does not convey the meaning of the original words. Similarly, the word “running” and “runner” may be stemmed to “run,” which is the base form of the word but it does not convey the meaning of the original words.
To avoid under-stemming, it is important to use a stemmer that is appropriate for the task and language. It is also important to test the stemmer on a sample of text to ensure that it is producing

the correct root forms. In some cases, using a lemmatizer instead of a stemmer may be a better solution as it takes into account the context of the word, making it less prone to errors.
Another approach to this problem is to use techniques like semantic role labeling, sentiment analysis, context-based information, etc. that help to understand the context of the text and make the stemming process more precise.

Applications of stemming:
Stemming is used in information retrieval systems like search engines. It is used to determine domain vocabularies in domain analysis. To display search results by indexing while documents are evolving into numbers and to map documents to common subjects by stemming. Sentiment Analysis, which examines reviews and comments made by different users about anything, is frequently used for product analysis, such as for online retail stores. Before it is interpreted, stemming is accepted in the form of the text-preparation mean.
A method of group analysis used on textual materials is called document clustering (also known as text clustering). Important uses of it include subject extraction, automatic document structuring, and quick information retrieval.

Fun Fact: Google search adopted a word stemming in 2003. Previously a search for “fish” would not have returned “fishing”
or “fishes”.
Some Stemming algorithms are:

Porter’s Stemmer algorithm
It is one of the most popular stemming methods proposed in 1980. It is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes. This stemmer is known for its speed and simplicity. The main applications of Porter Stemmer include data mining and Information retrieval. However, its applications are only limited to English words. Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word. The algorithms are fairly lengthy in nature and are known to be the oldest stemmer.

Example: EED -> EE means “if the word has at least one vowel and consonant plus EED ending, change the ending to EE” as ‘agreed’ becomes ‘agree’.

Advantage: It produces the best output as compared to other stemmers and it has less error rate.
Limitation: Morphological variants produced are not always real words.

Lovins Stemmer It is proposed by Lovins in 1968, that removes the longest suffix from a word then the word is recorded to convert this stem into valid words.
Example: sitting -> sitt -> sit
Advantage: It is fast and handles irregular plurals like 'teeth' and 'tooth' etc.
Limitation: It is time consuming and frequently fails to form words from stem.

Dawson Stemmer
It is an extension of Lovins stemmer in which suffixes are stored in the reversed order indexed by their length and last letter.
Advantage: It is fast in execution and covers more suffices.
Limitation: It is very complex to implement.

Krovetz Stemmer
It was proposed in 1993 by Robert Krovetz. Following are the steps:
1) Convert the plural form of a word to its singular form.
2) Convert the past tense of a word to its present tense and remove the suffix ‘ing’.
Example: ‘children’ -> ‘child’

Advantage: It is light in nature and can be used as pre-stemmer for other stemmers.
Limitation: It is inefficient in case of large documents.

Xerox Stemmer
Example:
'children' -> 'child'
'understood' -> 'understand'
'whom' -> 'who'
'best' -> 'good'

N-Gram Stemmer
An n-gram is a set of n consecutive characters extracted from a word in which similar words will have a high proportion of n-grams in common.
Example: 'INTRODUCTIONS' for n=2 becomes : *I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S*
Advantage: It is based on string comparisons and it is language dependent.
Limitation: It requires space to create and index the n-grams and it is not time efficient.

Snowball Stemmer:
When compared to the Porter Stemmer, the Snowball Stemmer can map non-English words too. Since it supports other languages the Snowball Stemmers can be called a multi-lingual stemmer. The Snowball stemmers are also imported from the nltk package. This stemmer is based on a programming language called 'Snowball' that processes small strings and is the most widely used stemmer. The Snowball stemmer is way more aggressive than Porter Stemmer and is also referred to as Porter2 Stemmer. Because of the improvements added when compared to the Porter Stemmer, the Snowball stemmer is having greater computational speed.

Lancaster Stemmer:
The Lancaster stemmers are more aggressive and dynamic compared to the other two stemmers. The stemmer is really faster, but the algorithm is really confusing when dealing with small words. But they are not as efficient as Snowball Stemmers. The Lancaster stemmers save the rules externally and basically uses an iterative algorithm. Lancaster Stemmer is straightforward, although it often produces results with excessive stemming. Over-stemming renders stems non-linguistic or meaningless.

LEMMATIZATION

Lemmatization is a text pre-processing technique used in natural language processing (NLP) models to break a word down to its root meaning to identify similarities. For example, a lemmatization algorithm would reduce the word better to its root word, or lemme, good. In stemming, a part of the word is just chopped off at the tail end to arrive at the stem of the word.

There are different algorithms used to find out how many characters have to be chopped off, but the algorithms don't actually know the meaning of the word in the language it belongs to. In lemmatization, the algorithms do have this knowledge. In fact, you can even say that these algorithms refer to a dictionary to understand the meaning of the word before reducing it to its root word, or lemma. So, a lemmatization algorithm would know that the word better is derived from the word good, and hence, the lemme is good. But a stemming algorithm wouldn't be able to do the same. There could be over-stemming or under-stemming, and the word better could be reduced to either bet, or bett, or just retained as better. But there is no way in stemming that can reduce better to its root word good. This is the difference between stemming and lemmatization.

Original	Stemming	Lemmatization
New	New	New
York	York	York
is	is	be
the	the	the
most	most	most
densely	dens	densely
populated	popul	populated
city	citi	city
in	in	in
the	the	the
United	Unite	United
States	State	States

Stemming vs Lemmatization

Lemmatization gives more context to chatbot conversations as it recognizes words based on their exact and contextual meaning. On the other hand, lemmatization is a time-consuming and slow process. The obvious advantage of lemmatization is that it is more accurate than stemming. So, if you're dealing with an NLP application such as a chat bot or a virtual assistant, where understanding the meaning of the dialogue is crucial, lemmatization would be useful. But this accuracy comes at a cost. Because lemmatization involves deriving the meaning of a word from something like a dictionary, it's very time-consuming. So, most lemmatization algorithms are slower compared to their stemming counterparts.

REMOVING STOP-WORDS

The words which are generally filtered out before processing a natural language are called stop words. These are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and does not add much information to the text. Examples of a few stop words in English are "the", "a", "an", "so", "what". Stop words are available in abundance in any human language. By removing these words, we remove the low-level information from our text in order to give more focus to the important information. In order words, we can say that the removal of such words does not show any negative consequences on the model we train for our task.

Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training. We do not always remove the stop words. The removal of stop words is highly dependent on the task we are performing and the goal we want to achieve. For example, if we are training a model that can perform the sentiment analysis task, we might not remove the stop words.

Movie review: "The movie was not good at all."
Text after removal of stop words: "movie good"

We can clearly see that the review for the movie was negative. However, after the removal of stop words, the review became positive, which is not the reality. Thus, the removal of stop words can be problematic here. Tasks like text classification do not generally need stop words as the other words present in the dataset are more important and give the general idea of the text. So, we generally remove stop words in such tasks.

In a nutshell, NLP has a lot of tasks that cannot be accomplished properly after the removal of stop words. So, think before performing this step. The catch here is that no rule is universal and no stop words list is universal. A list not conveying any important information to one task can convey a lot of information to the other task.

Word of caution: Before removing stop words, research a bit about your task and the problem you are trying to solve, and then make your decision.

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

Next comes a very important question: why we should remove stop words from the text?. So, there are two main reasons for that:

1. They provide no meaningful information, especially if we are building a text classification model. Therefore, we have to remove stop words from our dataset.
2. As the frequency of stop words are too high, removing them from the corpus results in much smaller data in terms of size. Reduced size results in faster computations on text data and the text classification model need to deal with a lesser number of features resulting in a robust model.

FEATURE ENGINEERING FOR TEXT REPRESENTATION

Feature engineering is one of the most important steps in machine learning. It is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Think machine learning algorithm as a learning child the more accurate information you provide the more they will be able to interpret the information well. Focusing first on our data will give us better results than focusing only on models. Feature engineering helps us to create better data which helps the model understand it well and provide reasonable results.

NLP is a subfield of artificial intelligence where we understand human interaction with machines using natural languages. To understand a natural language, you need to understand how we write a sentence, how we express our thoughts using different words, signs, special characters, etc basically we should understand the context of the sentence to interpret its meaning.

Extracting Features from Text

In this section, we will learn about common feature extraction techniques and methods. We'll also talk about when to use them and some challenges we might face implementing those techniques. ***Feature extraction methods can be divided into 3 major categories, basic, statistical, and advanced/vectorized.***

Basic Methods
These feature extraction methods are based on various concepts from NLP and linguistics. These are some of the oldest methods but still can be very reliable are used frequently in many areas.

- Parsing
- PoS Tagging
- Name Entity Recognition (NER)
- Bag of Words (BoW)

Statistical Methods
This is a bit more advanced feature extraction method and uses the concepts from statistics and probability to extract features
from text data.

- Term Frequency-Inverse Document Frequency (TF-IDF)

Advanced Methods
These methods can also be called vectorized methods as they aim to map a word, sentence, document to a fixed-length vector of real numbers. The goal of this method is to extract semantics from a piece of text, both lexical and distributional. Lexical semantics is just the meaning reflected by the words whereas distributional semantics refers to finding meaning based on various distributions in a corpus.

- Word2Vec
- GloVe: Global Vector for word representation

BAG OF WORDS MODEL

The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modelling, such as with machine learning algorithms.

The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents.
A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:
1. A vocabulary of known words.
2. A measure of the presence of known words.

It is called a “bag” of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document. The intuition is that documents are similar if they have similar content. Further, that from the content alone we can learn something about the meaning of the document. The bag-of-words can be as simple or complex as you like. The complexity comes both in deciding how to design the vocabulary of known words (or tokens) and how to score the presence of known words.

One of the biggest problems with text is that it is messy and unstructured, and machine learning algorithms prefer structured, well defined fixed-length inputs and by using the Bag-of-Words technique we can convert variable-length texts into a fixed-length vector.

Also, at a much granular level, the machine learning models work with numerical data rather than textual data. So to be more specific, by using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers.

Let us see an example of how the bag of words technique converts text into vectors

Example (1) without preprocessing:

Sentence 1: “Welcome to Great Learning, Now start learning”
Sentence 2: “Learning is a good practice”

Sentence	Welcome	to	Great	Learning	,	Now	start	learning	is	a	good	practice
Sentence1	1	1	1	1	1	1	1	1	0	0	0	0
Sentence2	0	0	0	0	0	0	0	1	1	1	1	1

But is this the best way to perform a bag of words. The above example was not the best example of how to use a bag of words. The words Learning and learning, although having the same meaning are taken twice. Also, a comma ’,’ which does not convey any information is also included in the vocabulary.

Example(2) with preprocessing:

Sentence 1: ”Welcome to Great Learning, Now start learning”
Sentence 2: “Learning is a good practice”

Step 1: Convert the above sentences in lower case as the case of the word does not hold any information.
Step 2: Remove special characters and stopwords from the text. Stopwords are the words that do not contain much information about text like ‘is’, ‘a’,’the and many more’.

After applying the above steps, the sentences are changed to

Sentence 1: ”welcome great learning now start learning”
Sentence 2: “learning good practice”

Step 3: Go through all the words in the above text and make a list of all of the words in our model vocabulary.

- welcome
- great
- learning
- now
- start
- good
- Practice

Now as the vocabulary has only 7 words, we can use a fixed-length document-representation of 7, with one position in the vector to score each word.The scoring method we use here is the same as used in the previous example. For sentence 1, the count of words is as follow:

Word	Frequency
welcome	1
great	1
learning	2
now	1
start	1
good	0
practice	0

Writing the above frequencies in the vector

Sentence 1 → [1,1,2,1,1,0,0]

Now for sentence 2, the scoring would be like

Word	Frequency
welcome	0
great	0
learning	1
now	0
start	0
good	1
practice	1

Similarly, writing the above frequencies in the vector form

Sentence 2 → [0,0,1,0,0,1,1]

Sentence	welcome	great	learning	now	start	good	practice
Sentence1	1	1	2	1	1	0	0
Sentence2	0	0	1	0	0	1	1

The approach used in example two is the one that is generally used in the Bag-of-Words technique, the reason being that the datasets used in Machine learning are tremendously large and can contain vocabulary of a few thousand or even millions of words. Hence, preprocessing the text before using bag-of-words is a better way to go.There are various preprocessing steps that can increase the performance of Bag-of-Words.

Limitations of Bag-of-Words

The bag-of-words model is very simple to understand and implement and offers a lot of flexibility for customization on your specific text data. It has been used with great success on prediction problems like language modeling and documentation classification.
Nevertheless, it suffers from some shortcomings, such as:

•Vocabulary: The vocabulary requires careful design, most specifically in order to manage the size, which impacts the sparsity of the document representations.
•Sparsity: Sparse representations are harder to model both for computational reasons (space and time complexity) and also for information reasons, where the challenge is for the models to harness so little information in such a large representational space.
•Meaning: Discarding word order ignores the context, and in turn meaning of words in the document (semantics). Context and meaning can offer a lot to the model, that if modeled could tell the difference between the same words differently arranged ("this is interesting" vs "is this interesting"), synonyms ("old bike" vs "used bike"), and much more.

What are N-Grams?

Again same questions, what are n-grams and why do we use them? Let us understand this with an example below-
Sentence 1: "This is a good job. I will not miss it for anything"
Sentence 2: "This is not good at all"
For this example, let us take the vocabulary of 5 words only. The five words being-

- good
- job
- miss
- not
- all

So, the respective vectors for these sentences are:
"This is a good job. I will not miss it for anything"=[1,1,1,1,0]
"This is not good at all"=[1,0,0,1,1]
Can you guess what is the problem here? Sentence 2 is a negative sentence and sentence 1 is a positive sentence. Does this reflect in any way in the vectors above? Not at all. So how can we solve this problem? Here come the N-grams to our rescue.

An N-gram is an N-token sequence of words: a 2-gram (more commonly called a bigram) is a two-word sequence of words like "really good", "not good", or "your homework", and a 3-gram (more commonly called a trigram) is a three-word sequence of words like "not at all", or "turn off light".For example, the bigrams in the first line of text in the previous section: "This is not good at all" are as follows:

- "This is"
- "is not"
- "not good"

- “good at”
- “at all”
-

Now if instead of using just words in the above example, we use bigrams (Bag-of-bigrams) as shown above. The model can differentiate between sentence 1 and sentence 2. So, using bi-grams makes tokens more understandable (for example, “HSR Layout”, in Bengaluru, is more informative than “HSR” and “layout”)So we can conclude that a bag-of-bigrams representation is much more powerful than bag-of-words, and in many cases proves very hard to beat.

N-grams:

N-grams are contiguous sequences of n items from a given sample of text or speech. In the context of natural language processing (NLP), these items are typically words, characters, or symbols. N-grams are used for various tasks in NLP, including language modeling, text generation, and feature extraction. The value of ’n’ in n-grams determines the number of consecutive elements considered. Here’s a breakdown of different types of n-grams:

1. Unigrams (1-grams): Unigrams are single words considered individually. For example, in the sentence “I love natural language processing,” the unigrams would be [‘I’, ‘love’, ‘natural’, ‘language’, ‘processing’].
2. Bigrams (2-grams): Bigrams consist of sequences of two adjacent words. For the same sentence, the bigrams would be [‘I love’, ‘love natural’, ‘natural language’, ‘language processing’].
3. Trigrams (3-grams): Trigrams are sequences of three adjacent words. Continuing with the example sentence, the trigrams would be [‘I love natural’, ‘love natural language’, ‘natural language processing’].
4. N-grams (N-grams): N-grams refer to sequences of N adjacent elements, which can be words, characters, or symbols. For instance, if ’N’ is 4, then we have 4-grams, also known as quadgrams.

N-grams are valuable because they capture more context than individual words, especially for tasks where word order is important. They can help in tasks like language modeling, where predicting the next word in a sentence relies on the preceding words. Additionally, n-grams can be used to identify common phrases or expressions in text, aiding in tasks such as sentiment analysis or document categorization.
N-grams can be used to generate text by predicting the next word or character based on the preceding n-1 words or characters. This approach is commonly used in chatbots, text summarization systems, and content generation tools.
However, the main challenge with using n-grams is the exponential increase in the number of unique combinations as ’n’ increases. This can lead to high dimensionality, increased computational complexity, and issues with sparsity, especially when dealing with large vocabularies or corpora.

TF-IDF MODEL

TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic used in natural language processing and information retrieval to evaluate the importance of a term (word) within a document relative to a collection of documents (corpus). TF-IDF is calculated based on two main components: term frequency (TF) and inverse document frequency (IDF).

Here’s a breakdown of how TF-IDF is calculated:

1. Term Frequency (TF): Term frequency measures how frequently a term appears in a document. It is calculated as the ratio of the number of times a term occurs in a document

to the total number of terms in the document. The idea is to give higher weight to terms that appear more frequently within a document.

The formula for TF is typically:

TF(t, d) = Number of times term t appears in document d /

Total number of terms in document d

2. **Inverse Document Frequency (IDF):** Inverse document frequency measures how unique or rare a term is across a collection of documents. It is calculated as the logarithm of the ratio of the total number of documents in the corpus to the number of documents containing the term. The formula for IDF is typically:

IDF(t, D) = log (Total number of documents in the corpus /D/ /

Number of documents containing term t)

3. **TF-IDF Calculation:** The TF-IDF score for a term 't' in a document 'd' is calculated by multiplying the term frequency (TF) of 't' in 'd' by the inverse document frequency (IDF) of 't' across the entire corpus. The formula for TF-IDF is:

TF-IDF(t,d,D) = TF(t,d) × IDF(t,D)

Here, D represents the entire corpus of documents.

TF-IDF assigns higher weights to terms that are frequent within a document (high TF) but rare across the entire corpus (high IDF), indicating their importance in distinguishing that document from others. Conversely, common terms that appear frequently across many documents are assigned lower weights.

TF-IDF is commonly used in various text processing tasks, including:

- Document retrieval: To rank documents based on their relevance to a query.
- Text classification: To extract features for training machine learning models.
- Keyword extraction: To identify important terms or phrases within a document.
- Information retrieval: To index and search text documents efficiently.

Overall, TF-IDF is a useful technique for representing and evaluating the importance of terms in text data, aiding in various NLP and information retrieval tasks.

Let's understand it using a simple example:
Sentence 1: good boy

Sentence 2: good girl
Sentence 3: boy girl good

First we calculate, words Frequency: *good: 3, boy: 2, girl: 2*

We will calculate TF:

	Sent1	Sent2	Sent3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

Then we will calculate IDF:

Words	IDF
good	log(3/3)=0
boy	log(3/2)
girl	log(3/2)

Now multiply last two tables:

	good	boy	girl
Sent 1	(1/2)*0	1/2* log(3/2)	0
Sent 2	0	0	1/2*log(3/2)
Sent 3	0	1/3*log(3/2)	1/3*log(3/2)

Hence we can see that, for Sentence 1, `boy` value is higher compared to other words. so there is some semantic meaning
And similarly, For Sentence 2 `girl` is given importance and for Sentence 3: `boy` and `girl`

Implementation of TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample corpus (collection of documents)
corpus = [
    'This is the first document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',
]

# Create an instance of TfidfVectorizer
vectorizer = TfidfVectorizer()

# Fit the vectorizer to the corpus and transform the corpus into a TF-IDF matrix
tfidf_matrix = vectorizer.fit_transform(corpus)
```

```
# Get the list of unique words (vocabulary)
vocab = vectorizer.get_feature_names_out()

# Convert the TF-IDF matrix to a dense numpy array for easier manipulation
tfidf_matrix_dense = tfidf_matrix.toarray()

# Print the TF-IDF matrix and the corresponding vocabulary
print("TF-IDF matrix:")
print(tfidf_matrix_dense)
print("\nVocabulary:")
print(vocab)
```

Output of above:

```
print("TF-IDF matrix:")
print(tfidf_matrix_dense)
print("\nVocabulary:")
print(vocab)
```

```
TF-IDF matrix:
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]
 [0.         0.6876236  0.         0.28108867 0.         0.53864762
  0.28108867 0.         0.28108867]
 [0.51184851 0.         0.         0.26710379 0.51184851 0.
  0.26710379 0.51184851 0.26710379]
 [0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]

Vocabulary:
['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']
```

Disadvantages of TF-IDF:

1. Semantic info is not stored, as order of words may not be same.
2. TF-IDF gives importance to uncommon words.
3. It doesn't consider the order of words within a document.

To overcome these problems of TF-IDF, we can use word2vec model.

One more example.

Let's say that the term "car" appears 25 times in a document that contains 1,000 words.
We'd calculate the term frequency (TF) as follows:

TF = 25/1,000 = 0.025

Next, let's say that a collection of related documents contains a total of 15,000 documents.
If 300 documents out of the 15,000 contain the term "car," we would calculate the inverse document frequency as follows:

IDF = log 15,000/300 = 1.69

Now, we can calculate the TF-IDF score by multiplying these two numbers:

TF-IDF = TF x IDF = 0.025 x 1.69 = 0.04225

CS2V14 – TEXT AND SPEECH ANALYSIS (PE)
2024-2025 -ODD SEM - QUESTION BANK
UNIT 1 – BASICS OF NATURAL LANGUAGE PROCESSING - TWO MARKS (PART A)

1. Define NLP along with two main parts of NLP.
   NLP for Natural Language Processing is a discipline that focuses on the understanding, manipulation and generation of natural language by machines. Thus, NLP is really at the interface between **computer science and linguistics**. It is about the ability of the machine to interact directly with humans.
   Two main parts of NLP are
   i. Data Preprocessing
   ii. Algorithm Development

2. What are the Five Phases of NLP?
   1. Lexical or morphological analysis
   2. Syntax analysis (Parsing)
   3. Semantic analysis
   4. Discourse integration
   5. Pragmatic analysis

3. What is Lexical/Morphological Analysis?
   The first phase of NLP is word structure analysis, which is referred to as lexical or morphological analysis. A lexicon is defined as a collection of words and phrases in a given language. Similarly, morphological analysis is the process of identifying the morphemes of a word. lexical or morphological analysis helps guide web searching. For instance, when doing on-page analysis, you can perform lexical and morphological analysis to understand how often the target keywords are used in their core form.

4. What is Morpheme? (Or) What are the Two types of Morphemes?
   A morpheme is a basic unit of English language construction, which is a small element of a word, that carries meaning. These can be either a free morpheme (e.g. walk) or a bound morpheme (e.g. -ing, -ed), with the difference between the two being that the latter cannot stand on its own to produce a word with meaning, and should be assigned to a free morpheme to attach meaning.

5. What is Syntax Analysis? (Or) Define Parsing.
   Syntax Analysis is the second phase of natural language processing. Syntax analysis or parsing is the process of checking grammar, word arrangement, and overall – the identification of relationships between words and whether those make sense. There are two common methods, and multiple approaches to construct the syntax tree – top-down and bottom-up.

6. Define Semantic Analysis.
   Semantic analysis is the third stage in NLP, when an analysis is performed to understand the meaning in a statement. This type of analysis is focused on uncovering the definitions of words, phrases, and sentences and identifying whether the way words are organized in a sentence makes sense semantically.

7. List the Functions of Semantic Analysis.
   There are a list of important functions of semantic analysis, which allow for natural language understanding:
   • To ensure that the data types are used in a way that's consistent with their definition.
   • To ensure that the flow of the text is consistent.
   • Identification of synonyms, antonyms, homonyms, and other lexical items.
   • Overall word sense disambiguation.
   • Relationship extraction from the different entities identified from the text.

8. Define Discourse Integration.
   Discourse integration is the fourth phase in NLP, and simply means contextualisation. Discourse integration is the analysis and identification of the larger context for any smaller part of natural language structure (e.g. a phrase, word or sentence).

9. Define Pragmatic Analysis.
   Pragmatic analysis is the fifth and final phase of natural language processing. As the final stage, pragmatic analysis extrapolates and incorporates the learnings from all other, preceding phases of NLP. Pragmatic analysis involves the process of abstracting or extracting meaning from the use of language, and translating a text, using the gathered knowledge from all other NLP steps performed beforehand.

10. List the steps involved in governing the language Syntax and Structure.
    - Parts of Speech (POS) Tagging
    - Shallow Parsing or Chunking
    - Constituency Parsing
    - Dependency Parsing

11. What do you mean by PoS?
    Parts of speech (POS) are specific lexical categories to which words are assigned, based on their syntactic context and role. Usually, words can fall into one of the following major categories like Noun, Verb, Adjective, Adverb etc...

12. List any two NLP tools available for PoS tagging.
    NLTK and spaCy.

13. List some examples for PoS Tags in Python.

	Word	POS tag	Tag type
0	US	NNP	PROPN
1	unveils	VBZ	VERB
2	world	NN	NOUN
3	's	POS	PART
4	most	RBS	ADV
5	powerful	JJ	ADJ
6	supercomputer	NN	NOUN
7	,	,	PUNCT
8	beats	VBZ	VERB
9	China	NNP	PROPN

SpaCy POS tagging

	Word	POS tag
0	US	NNP
1	unveils	VBZ
2	world's	VBZ
3	most	RBS
4	powerful	JJ
5	supercomputer,	JJ
6	beats	NNS
7	China	NNP

NLTK POS tagging

Fig. Python code & Output of POS tagging a news headline

14. Define **Shallow Parsing or Chunking .**

Shallow parsing, also known as light parsing or chunking, is a popular natural language processing technique of analyzing the structure of a sentence to break it down into its smallest constituents (which are tokens such as words) and group them together into higher-level phrases. This includes POS tags and phrases from a sentence

15. List the Phrases in **Shallow Parsing or Chunking.**

Noun phrase (NP): These are phrases where a noun acts as the head word.
Verb phrase (VP): These phrases are lexical units that have a verb acting as the head word.
Adjective phrase (ADJP): These are phrases with an adjective as the head word.
Adverb phrase (ADVP): These phrases act like adverbs since the adverb acts as the head word in the phrase. Adverb phrases are used as modifiers for nouns, verbs, or adverbs themselves by providing further details that describe or qualify them.
Prepositional phrase (PP): These phrases usually contain a preposition as the head word and other lexical components like nouns, pronouns, and so on. These act like an adjective or adverb describing other words or phrases.

16. Define Constituency Parsing.

Constituent-based grammars are used to analyze and determine the constituents of a sentence. These grammars can be used to model or represent the internal structure of sentences in terms of a hierarchically ordered structure of their constituents.

17. Define Dependency Parsing.

The basic principle behind a dependency grammar is that in any sentence in the language, all words except one, have some relationship or dependency on other words in the sentence. Dependency-based grammars to analyze and infer both structure and semantic dependencies and relationships between tokens in a sentence.

18. What is called as a root word?

The word that has no dependency is called the root of the sentence. The verb is taken as the root of the sentence in most cases. All the other words are directly or indirectly linked to the root verb using links, which are the dependencies.

19. List some of the dependency tags.

Det, amod, nsubj, cc, conj, acomp, pobj etc..,

20. What is Text Proprocessing/Wrangling?

Text preprocessing or wrangling is a method to clean the text data and make it ready to feed data to the model. Text data contains noise in various forms like emotions, punctuation, text in a different case.

21. List the techniques to perform text preprocessing or wrangling.

- Contraction Mapping/ Expanding Contractions.
- Tokenization
- Spell-checking
- Stopwords Removal
- Stemming/Lemmatization.

22. Define **Tokenization.**
    Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. Hence, *tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization.*

23. Word and Character Tokenization.
    Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed.
    Character Tokenization splits apiece of text into a set of characters. It overcomes the drawbacks we saw above about Word Tokenization.

24. Subword Tokenization.
    Subword Tokenization splits the piece of text into subwords (or n-gram characters). For example, words like lower can be segmented as low-er, smartest as smart-est, and so on.

25. Explain Stemming.
    Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”. Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words.

26. Name some different algorithms for stemming.
    - Porter stemmer
    - Snowball stemmer
    - Lancaster stemmer.

27. What are the two errors of stemming.
    a. Over- stemming: Over-stemming occurs when two words are stemmed from the same root that are of different stems. Over-stemming can also be regarded as false-positives. Over-stemming can lead to a loss of meaning and make the text less readable For example, the word “arguing” may be stemmed to “argu,” which is not a valid word and does not convey the same meaning as the original word.
    b. Under Stemming. occurs when two words are stemmed from the same root that are not of different stems. Under-stemming can be interpreted as false-negatives. It refers to the situation where a stemmer does not produce the correct root form of a word or does not reduce a word to its base form.

28. Applications of stemming.
    Stemming is used in
        information retrieval systems like search engines
        domain vocabularies in domain analysis
        to display search results by indexing,
        sentiment Analysis.

29. Discuss different stemmers with each other.
    The Porter stemmer is the most widely used algorithm, and it is based on a set of heuristics that are used to remove common suffixes from words. The Snowball stemmer is a more advanced algorithm that is based on the Porter stemmer, but it also supports several other languages in addition to English. The Lancaster stemmer is a more aggressive stemmer and it is less accurate than the Porter stemmer and Snowball stemmer.

30. Define Lemmatization.
Lemmatization is a text pre-processing technique used in natural language processing (NLP) models to break a word down to its root meaning to identify similarities. For example, a lemmatization algorithm would reduce the word better to its root word, or lemme, good. In stemming, a part of the word is just chopped off at the tail end to arrive at the stem of the word.

31. Perform Stemming and Lemmatization for the given sentence and observe the difference. List the same.

"New York is the most densely populated city in the united states "

Original	Stemming	Lemmatization
New	New	New
York	York	York
is	is	be
the	the	the
most	most	most
densely	dens	densely
populated	popul	populated
city	citi	city
in	in	in
the	the	the
United	Unite	United
States	State	States

32. What is meant by stop words removal / Define Removing stop-words.
The words which are generally filtered out before processing a natural language are called stop words. These are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and does not add much information to the text. Examples of a few stop words in English are "the", "a", "an", "so", "what". Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.

33. Perform Stop Word Removal for the below given sentence.
"The movie was not good at all."
Ans : Text after removal of stop words: "movie good"

34. Define Feature Engineering.
Feature engineering is one of the most important steps in machine learning. It is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Think machine learning algorithm as a learning child the more accurate information you provide the more they will be able to interpret the information well.

35. What are the feature extraction methods?

Feature extraction methods can be divided into 3 major categories,

- basic
- statistical
- advanced/vectorized.

36. What is bag of words model?

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

1. A vocabulary of known words.
2. A measure of the presence of known words.

It is called a "bag" of words, because any information about the order or structure of words in the document is discarded.

37. List the limitations of **bag-of-words.**

- Vocabulary
- Sparsity
- Meaning.

38. Define N-Grams.

N-grams are contiguous sequences of n items from a given sample of text or speech. In the context of natural language processing (NLP), these items are typically words, characters, or symbols. N-grams are used for various tasks in NLP, including language modeling, text generation, and feature extraction.

39. What are the types of N-Grams.

Unigrams (1-grams): Unigrams are single words considered individually. For example, in the sentence "I love natural language processing," the unigrams would be ['I', 'love', 'natural', 'language', 'processing'].

Bigrams (2-grams): Bigrams consist of sequences of two adjacent words. For the same sentence, the bigrams would be ['I love', 'love natural', 'natural language', 'language processing'].

Trigrams (3-grams): Trigrams are sequences of three adjacent words. Continuing with the example sentence, the trigrams would be ['I love natural', 'love natural language', 'natural language processing'].

N-grams (N-grams): N-grams refer to sequences of N adjacent elements, which can be words, characters, or symbols. For instance, if 'N' is 4, then we have 4-grams, also known as quadgrams.

40. Explain **TF-IDF MODEL .**

TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic used in natural language processing and information retrieval to evaluate the importance of a term (word) within a document relative to a collection of documents (corpus). TF-IDF is calculated based on two main components: term frequency (TF) and inverse document frequency (IDF).

$$TF(t, d) = \text{Number of times term t appears in document d} / \text{Total number of terms in document d}$$

$$IDF(t, D) = \log_e(\text{Total number of documents in the corpus } |D| / \text{Number of documents containing term t})$$

$$\text{TF-IDF}(t,d,D) = \text{TF}(t,d) \times \text{IDF}(t,D)$$

41. In a collection 15,000 documents, if 300 documents out of the 15,000 contain the term "car," Calculate the inverse document frequency. (TF of the car is calculated as 0.025). Also calculate the TF-IDF.

IDF = log (15,000/300) = 1.69.

TF-IDF = TF x IDF = 0.025 x 1.69 = 0.04225

42. List the applications of TF-IDF.

TF-IDF is commonly used in various text processing tasks, including:

- Document retrieval: To rank documents based on their relevance to a query.
- Text classification: To extract features for training machine learning models.
- Keyword extraction: To identify important terms or phrases within a document.
- Information retrieval: To index and search text documents efficiently.

CS2V14 – TEXT AND SPEECH ANALYSIS (PE)
2024-2025 - QUESTION BANK
UNIT 1 – INTRODUCTION TO NLP - PART B – QUESTIONS.

1. Explain Five Phases of NLP in detail
    - Lexical or morphological analysis
    - Syntax analysis (Parsing)
    - Semantic analysis
    - Discourse integration
    - Pragmatic analysis

2. Explain the parsing techniques to understand the language syntax and structure in detail.
    - Parts of Speech (POS) Tagging
    - Shallow Parsing or Chunking
    - Constituency Parsing
    - Dependency Parsing

3. Explain **Techniques to perform text preprocessing or wrangling in detail. (or) Explain Tokenization, Stemming / Lemmatization, Stop word Removal in detail.**

4. Explain Bag of Words Model and TF-IDF model with your own example.

5. For the Given sentence "The brown fox is quick and he is jumping over the lazy dog" perform parsing techniques and observe the language syntax and structure.

6. For the Given sentence "The brown fox is quick and he is jumping over the lazy dog" – apply the text preprocessing techniques. (NLTK can also be included).

7. For the below given sentences in a corpus calculate TF-IDF.

    S1: Monkey eats banana
    S2: Monkey plays ball
    S3: Child plays ball

8. For the below given sentences find the frequencies of the words for each sentence using Bag of Words model. Also represent the frequencies in the vector form for both the texts with and without preprocessing.
    - Sentence 1: "Welcome to Great Learning, Now start learning"
    - Sentence 2: "Learning is a good practice"

ACADEMIC YEAR	2024 – 2025 ODD
YEAR	III
SEM	V
SUBJECT CODE	CSV14
SUBJECT TITLE	TEXT AND SPEECH ANALYSIS
FACULTY INCHARGE	P PRIYADHARSHINI

UNIT II TEXT CLASSIFICATION 9

Basics of Sentiment Analysis – Lexicon based approaches: Words and Vectors – Word2Vec Model – Glove Model – FastText Model. Deep Learning Architecture for Sequence Processing: RNN, Transformers – Text Classification Applications.

Sentiment analysis is a process that involves analyzing textual data such as social media posts, product reviews, customer feedback, news articles, or any other form of text to classify the sentiment expressed in the text. The sentiment can be classified into three categories: Positive Sentiment Expressions indicate a favorable opinion or satisfaction; Negative Sentiment Expressions indicate dissatisfaction, criticism, or negative views; and Neutral Sentiment Text expresses no particular sentiment or is unclear.

Before analyzing the text, some preprocessing steps usually need to be performed. These include tokenization, breaking the text into smaller units like words or phrases, removing stop words such as common words like “and,” “the,” and so on, and stemming or lemmatization, which involves reducing words to their base or root form. At a minimum, the data must be cleaned to ensure the tokens are usable and trustworthy.

The strategies vary in complexity as well. In order of complexity

- **Lexicon-Based Methods:** Using dictionaries or lists of terms and their associated sentiment scores to determine overall sentiment. Consider a list of terms closely associated with positive sentiment within a domain and map those terms to a body of text to decide a final classification.
- **Machine Learning and Deep Learning:** One approach to classify sentiments is to use supervised learning algorithms or neural networks. These methods rely on pre-labeled data to accurately categorize different emotions or opinions.
- **Hybrid Approaches:** Combining multiple methods to improve accuracy, like machine learning models and lexicon-based analysis.

Sentiment analysis has multiple applications, including understanding customer opinions, analyzing public sentiment, identifying trends, assessing financial news, and analyzing feedback.

Challenges in sentiment analysis

It can be challenging for computers to understand human language completely. They struggle with interpreting sarcasm, idiomatic expressions, and implied sentiments. Despite these challenges, sentiment analysis is continually progressing with more advanced algorithms and models that can better capture the complexities of human sentiment in written text.

VECTOR SEMANTICS AND EMBEDDINGS

Vector semantics is the standard way to represent word meaning in NLP, helping vector semantics us model many of the aspects of word meaning. The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived from the distributions of embeddings word neighbours. Vectors for representing words are called embedding's (although the term is sometimes more strictly applied only to dense vectors like word2vec) Vector Semantics defines semantics & interprets word meaning to explain features such as word similarity. Its central idea is: Two words are similar if they have similar word contexts.

In its current form, the vector model inspires its working from the linguistic and philosophical work of the 1950s. Vector semantics represents a word in multi-dimensional vector space. Vector model is also called Embeddings, due to the fact that a word is embedded in a particular vector space. The vector model offers many advantages in NLP. For example, in sentimental analysis, sets up a boundary class and predicts if the sentiment is positive or negative (a binomial classification). Another key practical advantage vector semantics i of s that it can learn automatically from text without complex labelling or supervision. As a result of these advantages, vector semantics has become a de-facto standard for NLP applications such as Sentiment Analysis, Named Entity Recognition (NER), topic modelling, and so on.

WORD EMBEDDINGS It is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meaning to have a similar representation. They can also approximate meaning. A word vector with 50 values can represent 50 unique features. Features: Anything that relates words to one another. Eg: Age, Sports, Fitness, Employed etc. Each word vector has values corresponding to these features.

Goal of Word Embeddings

- To reduce dimensionality
- To use a word to predict the words around it
- Inter word semantics must be captured

How are Word Embeddings used?

- They are used as input to machine learning models.
- Take the words —-> Give their numeric representation —-> Use in training or inference
- To represent or visualize any underlying patterns of usage in the corpus that was used to train them.

Implementations of Word Embeddings:

Word Embeddings are a method of extracting features out of text so that we can input those features into a machine learning model to work with text data. They try to preserve syntactical and semantic information. The methods such as Bag of Words(BOW), Count Vectorizer and TFIDF rely on the word count in a sentence but do not save any syntactical or semantic information. In these algorithms, the size of the vector is the number of elements in the vocabulary. We can get a sparse matrix if most of the elements are zero. Large input vectors will mean a huge number of weights which will result in high computation required for training. Word Embeddings give a solution to these problems. Let's take an example to understand how word vector is generated by taking emoticons which are most frequently used in certain conditions and transform each emoji into a vector and the conditions will be our features.

Happy	????	????	????
Sad	????	????	????

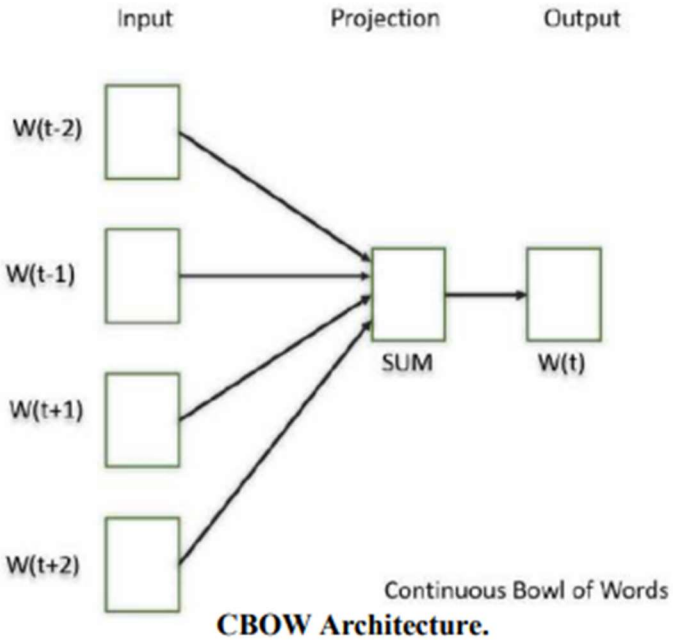
The emoji vectors for the emojis will be:

[happy, sad, excited, sick]
???? = [1,0,1,0]
???? = [0,1,0,1]
???? = [0,0,1,1] .....

In a similar way, we can create word vectors for different words as well on the basis of given features. The words with similar vectors are most likely to have the same meaning or are used to convey the same sentiment.

There are two different approaches for getting Word Embeddings:

1) Word2Vec: In Word2Vec every word is assigned a vector. We start with either a random vector or one-hot vector. One-Hot vector: A representation where only one bit in a vector is 1. If there are 500 words in the corpus then the vector length will be 500. After assigning vectors to each word we take a window size and iterate through the entire corpus. While we do this there are two neural embedding methods which are used: 1.1) Continuous Bowl of Words (CBOW) In this model what we do is we try to fit the neighbouring words in the window to the central word.

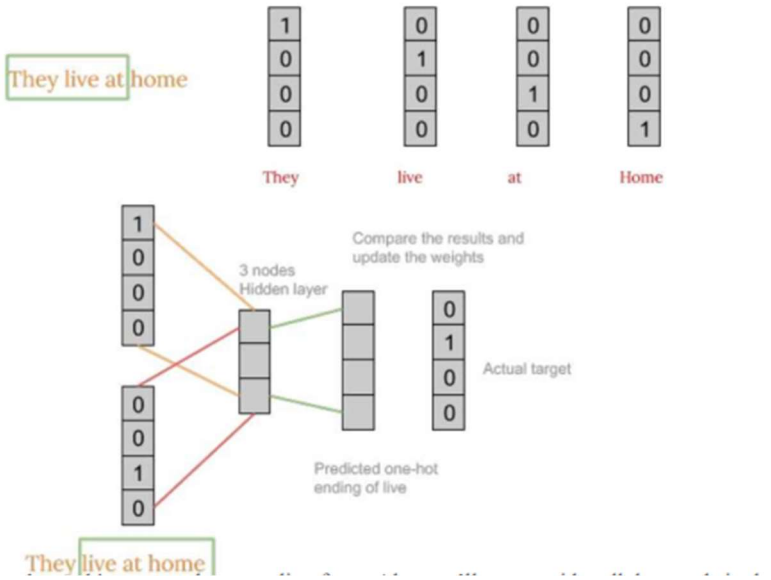


CBOW Architecture.

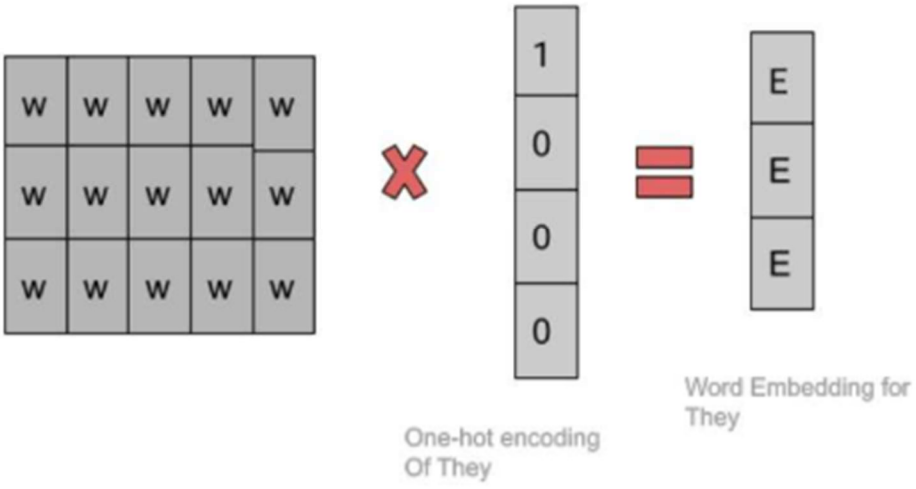
This architecture is very similar to a feed-forward neural network. This model architecture essentially tries to predict a target word from a list of context words.

The intuition behind this model is quite simple: given a phrase "Have a great day", we will choose our target word to be "a" and our context words to be ["have", "great", "day"]. What this model will do is take the distributed representations of the context words to try and predict the target word.

Consider a small example in which we have only four words i.e. live, home, they and at. For simplicity, we will consider that the corpus contains only one sentence, that being, 'They live at home'.



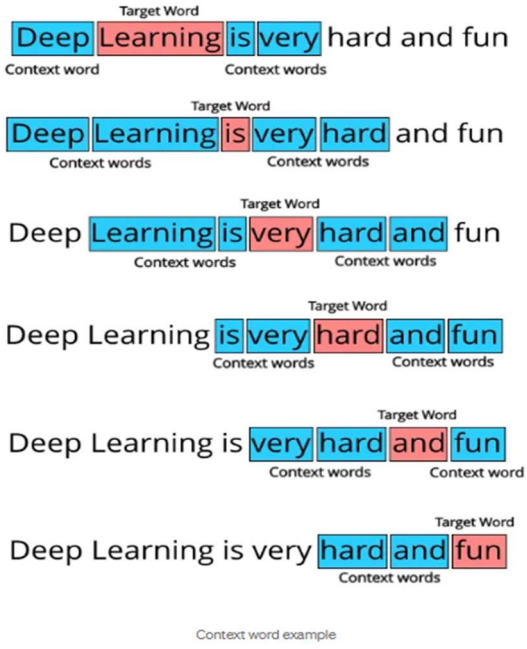
First, we convert each word into a one-hot encoding form. Also, we'll not consider all the words in the sentence but ll only take certain words that are in a window. For example, for a window size equal to three, we only consider three words in a sentence. The middle word is to be predicted and the surrounding two words are fed into the neural network as context. The window is then slid and the process is repeated again.
Finally, after training the network repeatedly by sliding the window a shown above, we get weights which we use to get the embeddings as shown below.



In this model, we try to make the central word closer to the neighbouring words. It is the complete opposite of the CBOW model. It is shown that this method produces more meaningful embeddings.

Example – 2

Consider, "Deep Learning is very hard and fun". We need to set something known as ***window size***. Let's say 2 in this case. What we do is iterate over all the words in the given data, which in this case is just one sentence, and **then consider a window of word which surrounds it**. Here since our window size is 2 we will consider 2 words behind the word and 2 words after the word, hence each word will get 4 words associated with it. We will do this for each and every word in the data and collect the word pairs.



Context word example

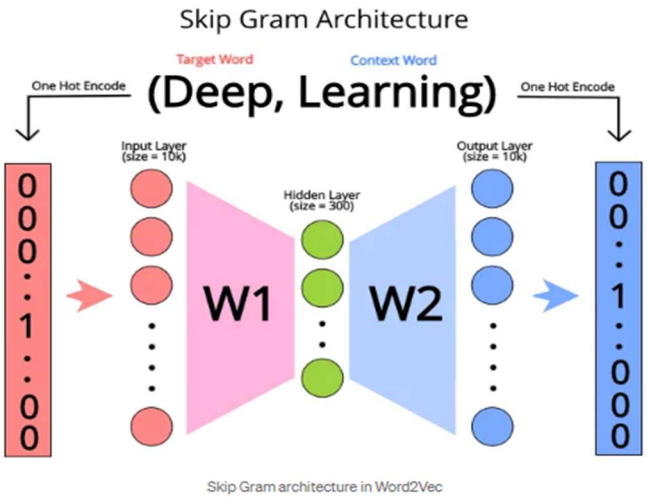
As we are ***passing the context window*** through the text data, we find all **pairs of target and context** words to form a dataset in the format of target word and context word. For the sentence above, it will look like this:
1st Window pairs: (Deep, Learning), (Deep, is)
2nd Window pairs: (Learning, Deep), (Learning, is), (Learning, very)
3rd Window pairs: (is, Deep), (is, Learning), (is, very), (is, hard)

And so on. At the end our **target word vs context word data set** is going to look like this:
(Deep, Learning), (Deep, is), (Learning, Deep), (Learning, is), (Learning, very), (is, Deep), (is, Learning), (is, very), (is, hard), (very, learning), (very, is), (very, hard), (very, and), (hard, is), (hard, very), (hard, and), (hard, fun), (and, very), (and, hard), (and, fun), (fun, hard), (fun, and)

This can be considered as our ***"training data"*** for word2vec.

In skipgram model, we try to predict each context word given a target word.

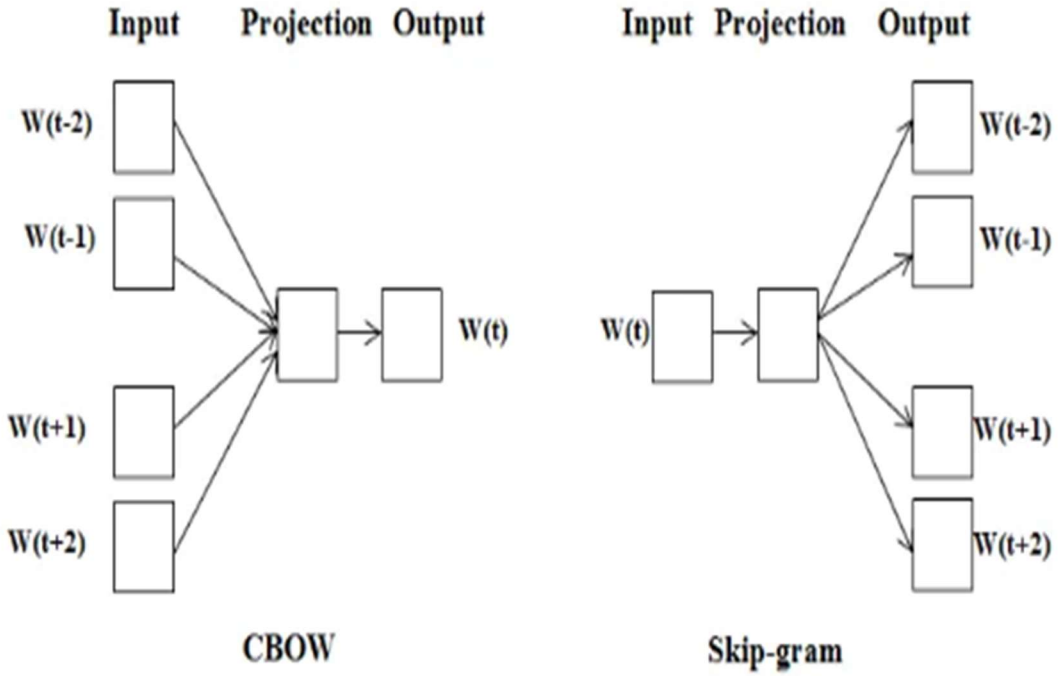
We use neural network for this prediction task. **The input to the neural network is the one hot encoded version of the context word**. Hence the size of the input and output layer is **V**(vocabulary count). This neural network has only one layer in the middle, the **size of the hidden layer determines the size of the word vectors we wish to have at the end.**



Since this neural network has a total of 3 layers, there will be only 2 weight matrices for the network, **W1** and **W2**. **W1** will have dimensions of 10000*300 and **W2** will have dimensions of 300*10000. These two weight matrices will play an important role in calculating word vectors.

For the entire dataset which we have collected from the original textual data, we will pass each pair into the neural network and train it. **Neural network here is trying to guess which context words can appear given a target word.** After training the neural network, if we input any target word into the neural network, it will give a vector output which represents the words which have a high probability of appearing near the given word.

For CBOW the only difference is that we try to predict the target word given the context words, essentially we just invert the skip gram model to get the CBOW model. It looks like this:



Here when we give a vector representation of a **group of context words**, we will get the **most appropriate target word which will be within the vicinity of those words**.

For example, if we give the sentence: Deep _____ is very hard, where ["Deep", "is", "very", "hard"] represents the context words, the neural network should hopefully give "Learning" as the **output target word**. This is the core task the neural network tries to train for in the case of CBOW.

Word vectors help represent semantics of the words — What does this mean?
It we means we could use vector reasoning for words one of the most famous example is from **Mikolov's paper**, where we see that if we use the word vectors and perform (here, we use **V(word)** to represent the vector representation of the word) V(King) -V(Man) + V(Woman), and the resulting vector is closest to V(Queen). It is easy to see why this is remarkable — our intuitive understanding of these words is reflected in the learned vector representations of the words.

This gives us the ability to add more of a punch in our text analysis pipelines-having an intuitive semantic representation of vectors will come in handy more than once.

Word2Vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique words in the corpus being assigned a corresponding vector in the space. Word vectors are positioned to vector space such that words that share common contexts in the corpus are located in close proximity to one other in the space.

Glove
Glove is based on **matrix factorization technique on word context matrix**. It first **constructs** a **large matrix** of (words x context) co-occurrence information ie. for each word, you count how frequently we see those word in some context in a large corpus.
In order to understand how GloVe works, we need to understand 2 main methods which GloVe was built on –

1. Global matrix factorization: In NLP, global matrix factorization is the process of using matrix factorization form linear algebra to reduce large term frequency matrices. These matrices usually represent the occurrences or the absence of words in the document.
2. Local context window: Local context window methods are CBOW and Skip-Gram, the one which were explained above.

Glove is a word vector representation method where training is performed on aggregated global word-word co-occurrence statistics from the corpus. This means that like word2vec it uses context to understand and create the word representations. The research paper describing the method is called *GloVe: Global Vectors for Word Representation* and is well worth a read as it describes some of the drawbacks of LSA and Word2Vec before describing their own method.

The author of the paper mention that instead of learning the raw occurrence probabilities, it was more useful to learn ratios of these co-occurence probabilities. The embeddings are optimized, so that the dot product of 2 vectors equals the log of number of times the 2 words will occur near each other.
For example, if 2 words "cat" and "dog" occur in the context of each other, say 20 times in 10-word window in the document corpus, then —
Vector(cat).Vector(dog) = log(10)
This forces the model to encode the frequency distribution of words that occur near them in a more global context.

FastText

FastText is a vector representation technique developed by facebook AI research. As its name suggests its fast and efficient method to perform same task and because of the nature of its training method, it ends up learning morphological details as well.

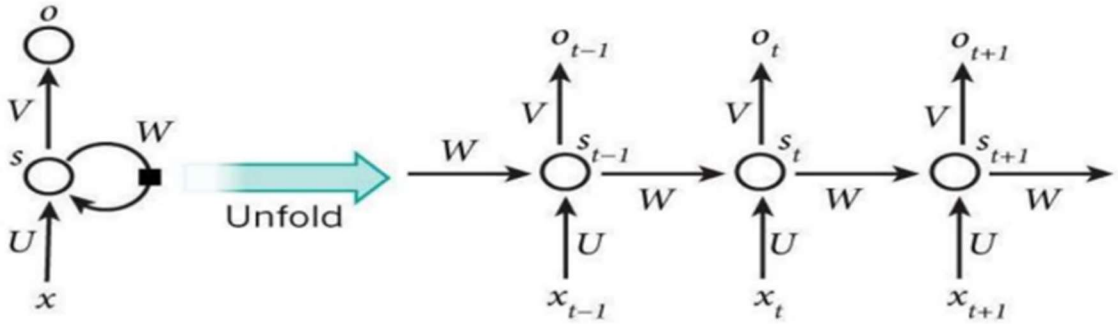
FastText is unique because it can derive word vectors for unknown words or out of vocabulary words — this is because by taking morphological characteristics of words into account, it can *create* the word vector for an unknown word. Since morphology refers to the structure or syntax of the words, FastText tends to perform better for such task, word2vec perform better for semantic task.
FastText works well with rare words. So even if a word wasn't seen during training, it can be broken down into n-grams to get its embeddings.
Word2vec and GloVe both fail to provide any vector representation for words that are not in the model dictionary. This is a huge advantage of this method.

Applications

- Analysing survey responses .
- Analysing verbatim comments.
- Music/Video recommendation system.

RNN - Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a type of Neural N etwork where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus, RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its Hidden state, which remembers some information about a sequence. The state is also referred to as Memory State since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.



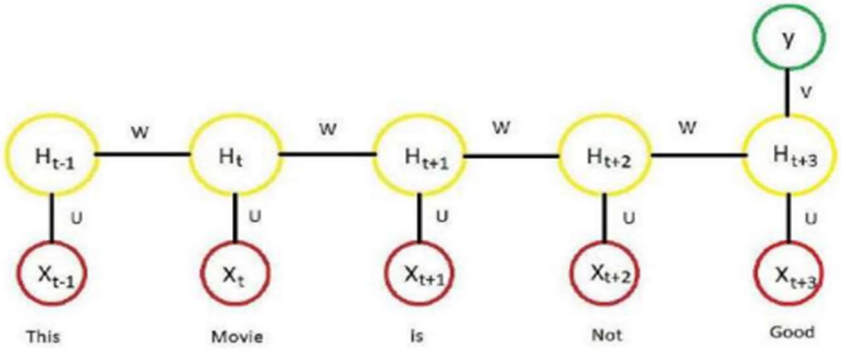
Recurrent neural network

The diagram above shows a detailed structure of an RNN architecture. The architecture described above is also called as a many to many architectures with (Tx = Ty) i.e. number of inputs = number of outputs. Such structure is quite useful in Sequence modelling.

Apart from the architecture mentioned above there are three other types of architectures of RNN which are commonly used.

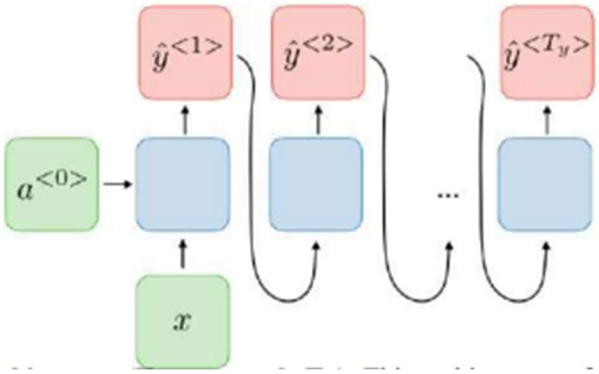
1.Many to One RNN: Many to one architecture refers to an RNN architecture where many inputs (Tx) are used to give one output (Ty). A suitable example for using such an architecture will be a classification task.

RNN are a very important variant of neural networks heavily used in Natural Language Processing. Conceptually they differ from a standard neural network as the standard input in a RNN is a word instead of the entire sample as in the case of a standard neural network. This gives the flexibility for the network to work with varying lengths of sentences, something which cannot be achieved in a standard neural network due to its fixed structure. It also provides an additional advantage of sharing features learned across different positions of text which cannot be obtained in a standard neural network.

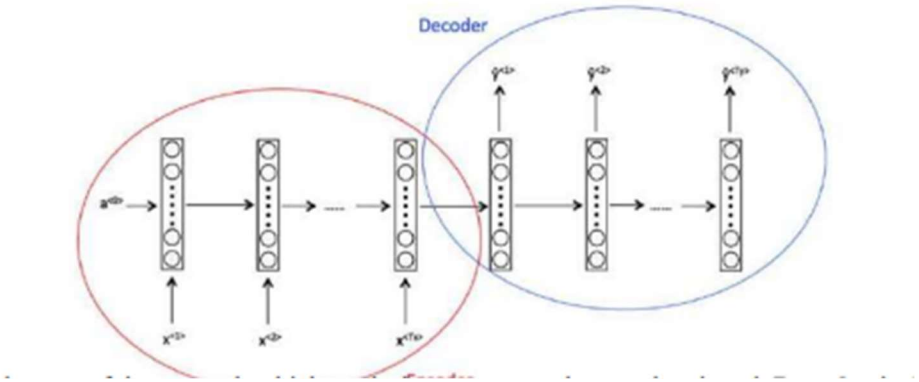


In the image above H represents the output of the activation function.

2. One to Many RNN: One to Many architecture refers to a situation where a RNN generates a series of output values based on a single input value. A prime example for using such an architecture will be a task, where a music generation input is a jounre or the first note.



3. Many to Many Architecture (Tx not equals Ty): This architecture refers to where many inputs are read to produce many outputs, where the length of inputs is not equal to the length of outputs. A prime example for using such an architecture is machine translation tasks.



Encoder refers to the part of the network which reads the sentence to be translated, and, is the part of the Decoder network which translates the sentence into desired language.

Limitations of RNN

Apart from all of its usefulness RNN does have certain limitations major of which are:

1. Examples of RNN architecture stated above can capture the dependencies in only one direction of the language. Basically, in the case of Natural Language Processing it assumes that the word coming after has no effect on the meaning of the word coming before. With our experience of languages, we know that it is certainly not true.
2. RNN are also not very good in capturing long term dependencies and the problem of vanishing gradients resurface in RNN.

NNs are ideal for solving problems where the sequence is more important than the individual items themselves. An RNNs is essentially a fully connected neural network that contains a refactoring of some of its layers into a loop. That loop is typically an iteration over the addition or concatenation of two inputs, a matrix multiplication and a non-linear function. Among the text usages, the following tasks are among those RNNs perform well at:

- Sequence labelling
- Natural Language Processing (NLP) text classification
- Natural Language Processing (NLP) text generation

Other tasks that RNNs are effective at solving are time series predictions or other sequence predictions that aren't image or tabular-based. There have been several highlighted and controversial reports in the media over the advances in text generation, Open AI's GPT-2 algorithm. In many cases the generated text is often indistinguishable from text written by humans.

RNNs effectively have an internal memory that allows the previous inputs to affect the subsequent predictions. It's much easier to predict the next word in a sentence with more accuracy, if you know what the previous words were. Often with tasks well suited to RNNs, the sequence of the items is as or more important than the previous item in the sequence.

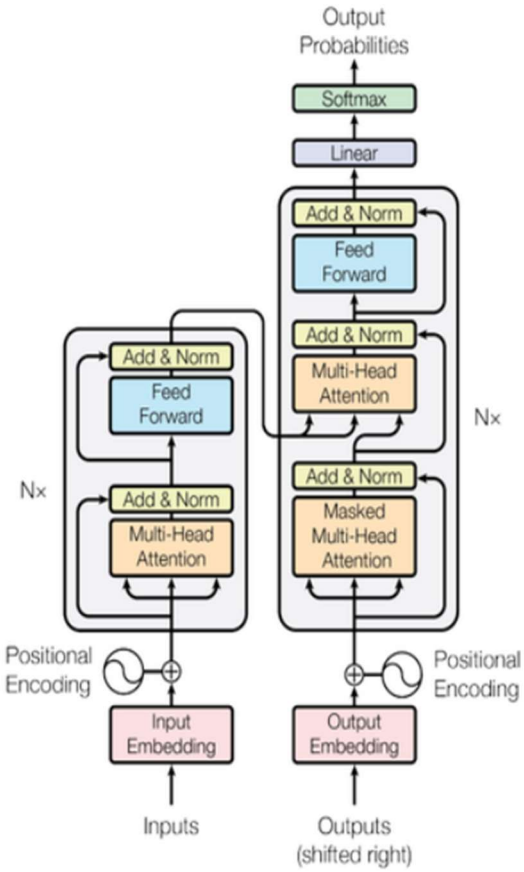
Transformer

"The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution."

Here, "transduction" means the conversion of input sequences into output sequences. The idea behind Transformer is to handle the dependencies between input and output with attention and recurrence completely.

Let's take a look at the architecture of the Transformer below. It might look intimidating but don't worry, we will break it down and understand it block by block.

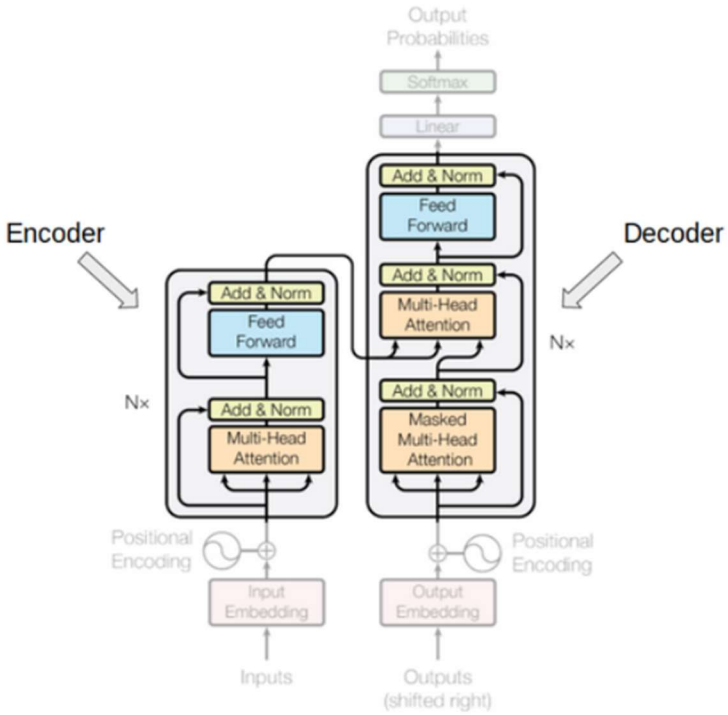
Transformer's Model Architecture



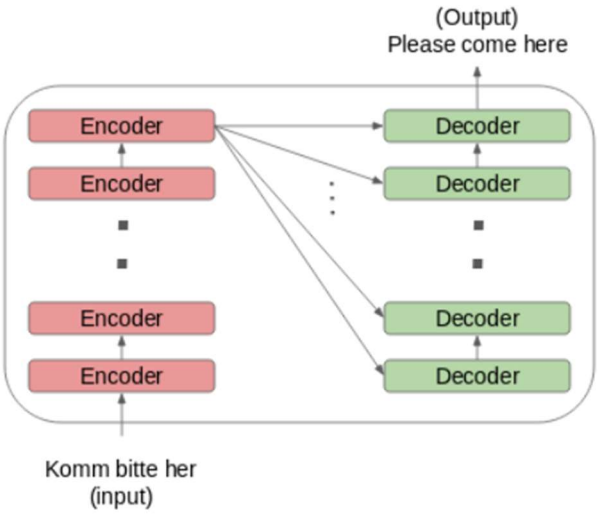
The Transformer – Model Architecture

The above image is a superb illustration of transformer in NLP architecture. Let's first focus on the Encoder and Decoder parts only.

Now focus on the below image. The Encoder block has 1 layer of a Multi-Head Attention followed by another layer of Feed Forward Neural Network. The decoder, on the other hand, has an extra Masked Multi-Head Attention.

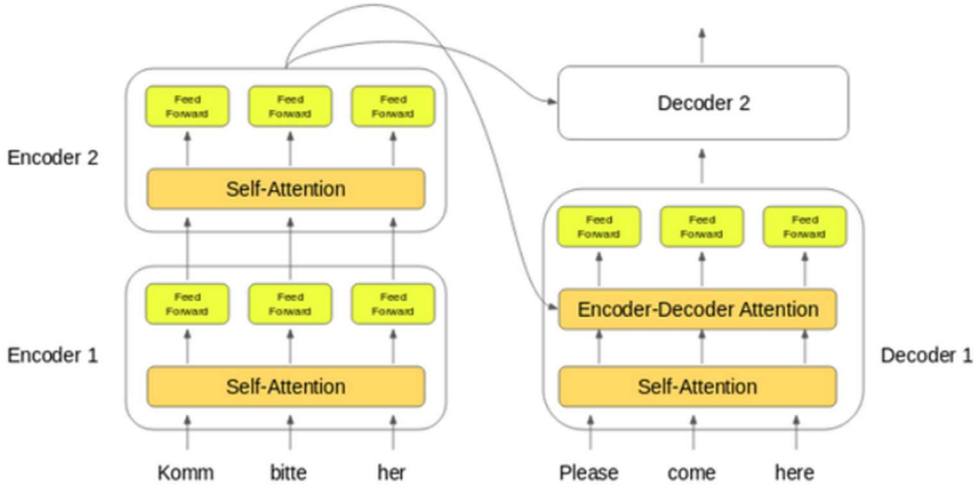


The encoder and decoder blocks are actually multiple identical encoders and decoders stacked on top of each other. Both the encoder stack and the decoder stack have the same number of units.The number of encoder and decoder units is a hyperparameter.



Let's see how this setup of the encoder and the decoder stack works:

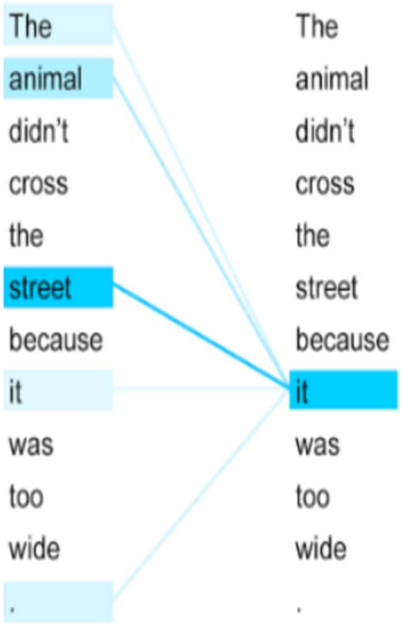
- The first encoder receives the word embeddings of the input sequence.
- It then transforms and propagates them to the next encoder.
- The output from the last encoder in the encoder-stack is passed to all the decoders in the decoder-stack, as depicted in the figure below:



An important thing to note here – in addition to the **self-attention** and feed-forward layers, the decoders also have one more layer of Encoder-Decoder Attention layer. This helps the decoder focus on the appropriate parts of the input sequence.

Getting a Hang of Self-Attention

"Self-attention, sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence."



Take a look at the above image. Can you figure out what the term **"it"** in this sentence refers to?

Is it referring to the street or to the animal? It's a simple question for us but not for an algorithm. When the model is processing the word "it", self-attention tries to associate "it" with **"animal"** in the same sentence.

Self-attention allows the model to look at the other words in the input sequence to get a better understanding of a certain word in the sequence. Now, let's see how we can calculate self-attention.

Calculating Self-Attention

First, we need to create three vectors from each of the encoder's input vectors:

- Query Vector
- Key Vector
- Value Vector.

During the training process, we train and update these vectors. We'll gain a deeper understanding of their roles once we complete this section.

Next, we will calculate self-attention for every word in the input sequence

Consider this phrase – "Action gets results". To calculate the self-attention for the first word "Action", we will calculate scores for all the words in the phrase with respect to "Action". This score determines the importance of other words when we are encoding a certain word in an input sequence

1. Taking the dot product of the Query vector (q1) with the keys vectors (k1, k2, k3) of all the words calculates the score for the first word:

Word	q vector	k vector	v vector	score
Action	q_1	k_1	v_1	$q_1 . k_1$
gets		k_2	v_2	$q_1 . k_2$
results		k_3	v_3	$q_1 . k_3$

2. Then, these scores are divided by 8 which is the square root of the dimension of the key vector:

Word	q vector	k vector	v vector	score	score / 8
Action	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$
gets		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$
results		k_3	v_3	$q_1 . k_3$	$q_1 . k_3 / 8$

3. Next, these scores are normalized using the softmax activation function:

Word	q vector	k vector	v vector	score	score / 8	Softmax
Action	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$	x_{11}
gets		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$	x_{12}
results		k_3	v_3	$q_1 . k_3$	$q_1 . k_3 / 8$	x_{13}

4. These normalized scores are then multiplied by the value vectors (v1, v2, v3) and sum up the resultant vectors to arrive at the final vector (z1).

This is the output of the self-attention layer. It is then passed on to the feed-forward network as input.

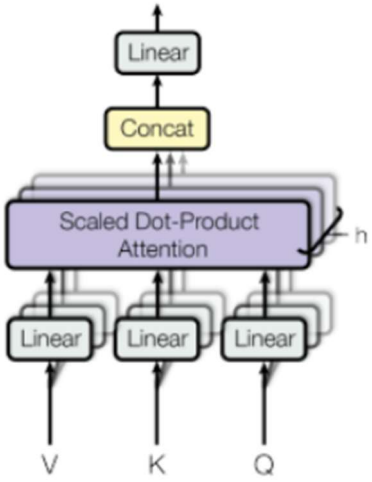
Word	q vector	k vector	v vector	score	score / 8	Softmax	Softmax * v	Sum
Action	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$	x_{11}	$x_{11} * v_1$	z_1
gets		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$	x_{12}	$x_{12} * v_2$	
results		k_3	v_3	$q_1 . k_3$	$q_1 . k_3 / 8$	x_{13}	$x_{13} * v_3$	

So, z1 is the self-attention vector for the first word of the input sequence "Action gets results". We can get the vectors for the rest of the words in the input sequence in the same fashion:

Word	q vector	k vector	v vector	score	score / 8	Softmax	Softmax * v	Sum
Action		k_1	v_1	$q_2 . k_1$	$q_2 . k_1 / 8$	x_{21}	$x_{21} * v_1$	
gets	q_2	k_2	v_2	$q_2 . k_2$	$q_2 . k_2 / 8$	x_{22}	$x_{22} * v_2$	z_2
results		k_3	v_3	$q_2 . k_3$	$q_2 . k_3 / 8$	x_{23}	$x_{23} * v_3$	

Word	q vector	k vector	v vector	score	score / 8	Softmax	Softmax * v	Sum
Action		k_1	v_1	$q_3 . k_1$	$q_3 . k_1 / 8$	x_{31}	$x_{31} * v_1$	
gets		k_2	v_2	$q_3 . k_2$	$q_3 . k_2 / 8$	x_{32}	$x_{32} * v_2$	
results	q_3	k_3	v_3	$q_3 . k_3$	$q_3 . k_3 / 8$	x_{33}	$x_{33} * v_3$	z_3

In the Transformer's architecture, self-attention is computed multiple times, in parallel and independently. This process is referred to as Multi-head Attention. The outputs concatenate and linearly transform, as depicted in the figure below:



Multi-Head Attention

Limitations of the Transformer

Transformer architecture NLP is undoubtedly a huge improvement over the RNN based seq2seq models. But it comes with its own share of limitations:

- Attention can only deal with fixed-length text strings. The text has to be split into a certain number of segments or chunks before being fed into the system as input
- This chunking of text causes **context fragmentation**. For example, if a sentence is split from the middle, then a significant amount of context is lost. In other words, the text is split without respecting the sentence or any other semantic boundary

CCS369- TEXT AND SPEECH ANALYSIS

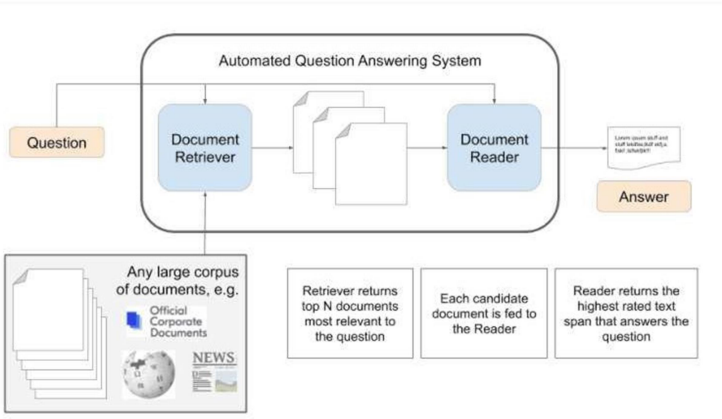
UNIT 3

INTRODUCTION

Question answering (QA) is a branch of artificial intelligence within the natural language processing and information retrieval fields; building systems that answer questions posed in a natural language by humans. QA is a computer science discipline within the fields of information retrieval and natural language processing (NLP) that is concerned with building systems that automatically answer questions that are posed by humans in a natural language.

Question-Answering System

This is a very adaptable design, and we have found that it can ask for a wide range of queries. Instead of having a list of options for each question, systems must choose the best answer from all potential spans in the passage, which means they must deal with a vast number of possibilities. Spans have the extra benefit of being simple to evaluate.



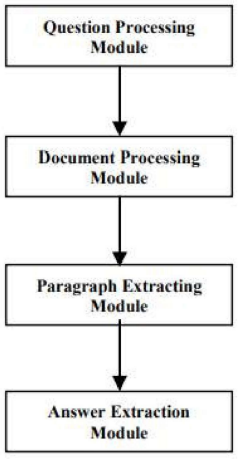


Fig 1: Framework of QA system

Question Processing Module

The question processing module converts natural language question queries for the document retriever. The process ranges simply returning the user's question as the query to employing question analysis to generate complex structured queries. This module also detects the expected answer type of a question e.g. the expected answer type of "When was Shivaji born?" is date this information helps guide the answer extraction process.

Document Processing Module

This module retrieves documents from the corpus that are likely to contain answers to the user's question. It consists of a query generation algorithm and text search engine. The query generation algorithm takes an input the user's question and creates a query containing terms likely to appear in documents containing an answer. This query is passed to the text search engine, which uses it to retrieve a set of documents.

Paragraph Extraction Module

Paragraph extraction algorithms take a document as a question and try to find passages from the document that contain an answer. Typical passage retrieval algorithms break the document into passages, compute a score for each passage and return the passage with the highest score. The system abstracts this mechanism so that passage tokenizing algorithms and passage scoring algorithms can be modularized as well the algorithm which cannot broken down can also support a large number of passage retrieval algorithms.

Answer extraction Module

The Modules takes as input a passage from the passage retrieval component and tries to retrieve an exact phrase to Question Processing Module - Document Processing Module - Paragraph Extracting Module - Answer Extraction, and return as an answer to achieve this required parsing and detailed question analysis by using of answer extraction algorithms. The identity answer extraction returns the center point of the passage, stripping words from either end until it fits within the specified answer window. There are many Approaches used in Question answering system based on different purpose namely **linguistic-based approach, statistical-based approach, and pattern matching approach.** The user needs precise and very specific answers. The large amount of data is continuously added in different scientific fields, in many disciplines. It becomes challenging for researchers and many users to cope up with data. Understands the way a specific approach is supporting for full fledge development of QA system.

- **Linguistic Approach**

The linguistic approach understands natural language text, linguistic & common knowledge Linguistic techniques such as tokenization, POS tagging, and parsing. These were implemented to user's question for formulating it into a precise query that merely extracts the respective response from the structured database

- **Statistical Approach**

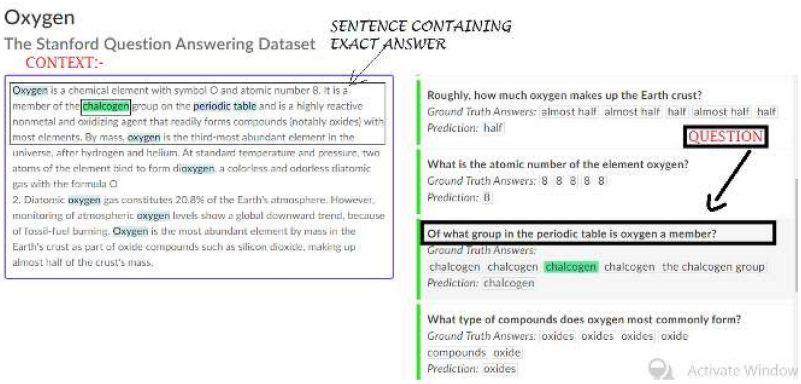
The availability of huge amounts of data on the internet increased the importance of statistical approaches. A statistical learning method gives better results than other approaches. Online text repositories and statistical approaches are independent of structured query languages and can formulate queries in natural language form. Mostly all Statistical Based QA systems applied a statistical technique in QA systems such as Support vector machine classifiers, Bayesian Classifiers, maximum entropy models

- **Pattern Matching Approach**

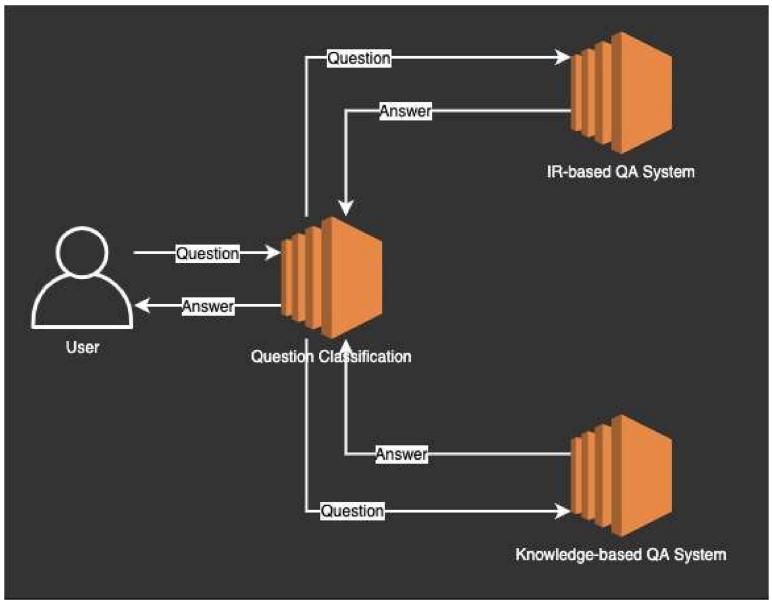
Pattern matching approach deals with the expressive power of text pattern, it replaces the sophisticated processing involved in other computing approaches. Most of the pattern-matching QA systems use the surface text pattern, while some of them also rely on templates for response generator

The QA setting, depending on the span is extremely natural. Open-domain QA systems can typically discover the right papers that hold the solution to many user questions sent into search engines. The task is to discover the shortest fragment of text in the passage or document that answers the query, which is the ultimate phase of "answer extraction."

Problem Description for Question-Answering System



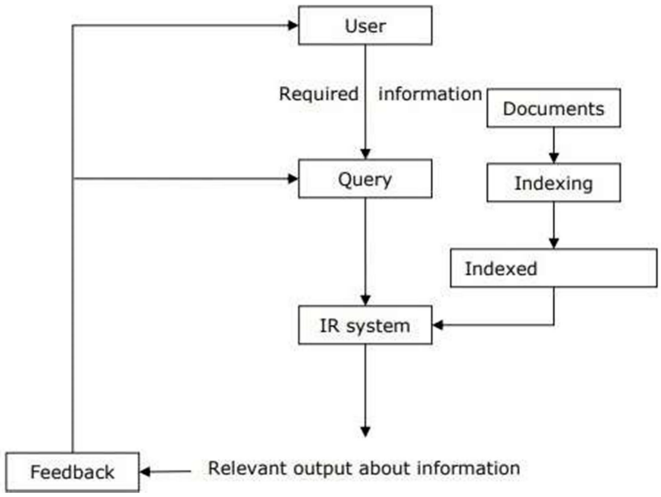
The purpose is to locate the text for any new question that has been addressed, as well as the context. This is a closed dataset, so the answer to a query is always a part of the context and that the context spans a continuous span. Here, divided the problem into two pieces as shown above.



INFORMATION RETRIEVAL

Information retrieval (IR) can be defined as a software program that deals with the organization, storage, retrieval, and evaluation of information from document repositories particularly textual information. The system assists users in finding the information they require but it does not explicitly return the answers to the questions. It informs the existence and location of documents that might consist of the required information. The documents that satisfy user's requirement are called relevant documents. A perfect IR system will retrieve only relevant documents.

With the help of the following diagram, we can understand the process of information retrieval (IR) –



It is clear from the above diagram that a user who needs information will have to formulate a request in the form of query in natural language. Then the IR system will respond by retrieving the relevant output, in the form of documents, about the required information.

Classical Problem in Information Retrieval (IR) System

The main goal of IR research is to develop a model for retrieving information from the repositories of documents. Here, we are going to discuss a classical problem, named ad-hoc retrieval problem, related to the IR system.

In ad-hoc retrieval, the user must enter a query in natural language that describes the required information. Then the IR system will return the required documents related to the desired information. For example, suppose we are searching something on the Internet and it gives some exact pages that are relevant as per our requirement but there can be some non-relevant pages too. This is due to the **ad-hoc retrieval problem.**

Aspects of Ad-hoc Retrieval

The information retrieval model needs to **provide the framework for the system** to work and define the many aspects of the retrieval procedure of the retrieval engines

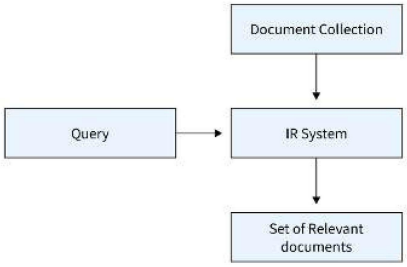
- The IR model has to provide a system for **how the documents in the collection** and user's queries are **transformed.**
- The IR model also needs to ingrain the functionality for **how the system identifies the relevancy** of the documents based on the query provided by the user.
- The system in the information retrieval model also needs to incorporate the **logic for ranking the retrieved documents** based on the relevancy.

Information Retrieval (IR) Model

Mathematically, models are used in many scientific areas having objective to understand some phenomenon in the real world. A model of information retrieval predicts and explains what a user will find in relevance to the given query. IR model is basically a pattern that defines the above-mentioned aspects of retrieval procedure and consists of the following –

- A model for documents.
- A model for queries.

- A matching function that compares queries to documents.



Mathematically, a retrieval model consists of –

D – Representation for documents.

R – Representation for queries.

F – The modeling framework for D, Q along with relationship between them.

R (q,di) – A similarity function which orders the documents with respect to the query. It is also called ranking.

Types of Information Retrieval (IR) Model

An information model (IR) model can be classified into the following three models –

Classical IR Model

It is the simplest and easy to implement IR model. This model is based on mathematical knowledge that was easily recognized and understood as well. The classical IR systems are designed **based on mathematical concepts** and are the most widely used, simplest, and easy-to-implement systems for information retrieval models. In this system, the retrieval of information depends on documents containing the **defined set of queries and there is no ranking or grading** of any kind.

- Classic Information Retrieval models can be **easily implemented** and updated accordingly.
- The different classical IR models are based on **mathematical knowledge** that was easily recognized and understood as well and take concepts like Document Representation, Query representation, and Retrieval / Matching function into account in their modeling.
- The term classic in the name of classical IR systems denotes that they **use foundational techniques** for text documents without extra information about the structure or content of a document.
- Examples of classical Information Retrieval models: Are boolean models, Vector space models, and Probabilistic IR models.

Non-Classical IR Model

It is completely opposite to classical IR model. Such kind of IR models are based on principles other than similarity, probability, Boolean operations. Information logic model, situation theory model and interaction models are the examples of non-classical IR model.Non-Classical Information Retrieval models are complete **opposite** to the classical IR models. They are based on principles other than similarity, probability, and Boolean operations.

- Non-classical IR models differ from classic models in that they are **built upon propositional logic**, a way to combine documents and queries in some representation and suitable logic.
    - Propositional logic (also known as sentential logic) is that branch of logic that studies ways of **combining** or altering statements or propositions to form more complicated statements or propositions.

- Examples of non-classical Information Retrieval models include Information Logic models, Situation Theory models, and Interaction models.

Alternative IR Model

It is the enhancement of classical IR model making use of some specific techniques from some other fields. Cluster model, fuzzy model, and latent semantic indexing (LSI) models are the example of alternative IR model.

Design features of Information retrieval (IR) systems

Let us now learn about the design features of IR systems –

Inverted Index

The primary data structure of most of the IR systems is in the form of inverted index. We can define an inverted index as a data structure that list, for every word, all documents that contain it and frequency of the occurrences in document. It makes it easy to search for 'hits' of a query word.

Stop Word Elimination

Stop words are those high frequency words that are deemed unlikely to be useful for searching. They have less semantic weights. All such kind of words are in a list called stop list. For example, articles "a", "an", "the" and prepositions like "in", "of", "for", "at" etc. are the examples of stop words. The size of the inverted index can be significantly reduced by stop list. As per Zipf's law, a stop list covering a few dozen words reduces the size of inverted index by almost half. On the other hand, sometimes the elimination of stop word may cause elimination of the term that is useful for searching. For example, if we eliminate the alphabet "A" from "Vitamin A" then it would have no significance.

Stemming

Stemming, the simplified form of morphological analysis, is the heuristic process of extracting the base form of words by chopping off the ends of words. For example, the words laughing, laughs, laughed would be stemmed to the root word laugh.

Some important and useful IR models.

The Boolean Model

It is the oldest information retrieval (IR) model. The model is based on set theory and the Boolean algebra, where documents are sets of terms and queries are Boolean expressions on terms.

The Boolean model in information retrieval is **based on the set theory and boolean algebra**. We can pose any query in the form of a Boolean expression of terms where the terms are logically combined using the **Boolean operators AND, OR, and NOT** in the Boolean retrieval model.

- Using the Boolean operators, the **terms** in the query and the concerned documents can be **combined** to form a whole new set of documents.
    - The Boolean **AND** of two logical statements x and y means that **both x AND y must be satisfied** and will be a set of documents that will **smaller or equal** to the document set
    - while the Boolean **OR** of these same two statements means that **at least one of these statements** must be satisfied and will fetch a set of documents that will be **greater or equal** to the document set otherwise.
    - **Any number** of logical statements can be **combined** using the three Boolean operators.
- The queries are designed as boolean expressions which have precise semantics and the retrieval strategy is based on **binary decision criterion**.
- The Boolean model can also be explained well by **mapping the terms in the query with a set of documents**.

The most famous web search engine in recent times Google also ranks the web page result set based on a two-stage system: In the **first** step, a **Simple Boolean Retrieval** model** returns matching documents** in no particular order, and in the next step **ranking** is done according to some **estimator of relevance**.

Aspects of Boolean Information Retrieval Model

Indexing: Indexing is one of the **core functionalities** of the information retrieval models and the **first step** in building an IR system assisting with the efficient retrieval of information.

- Indexing is majorly an **offline operation** that collects data about **which words occur in the text corpus** so that at search time we only have to access the **pre-compiled index** done beforehand.
- The boolean model builds the indices for the terms in the query considering that **index terms are present or absent** in a document.

Term-Document Incidence matrix: This is one of the basic **mathematical models to represent text data** and can be used to answer Boolean expression queries using the Boolean Retrieval Model. It can be **used to answer any query** as a Boolean expression.

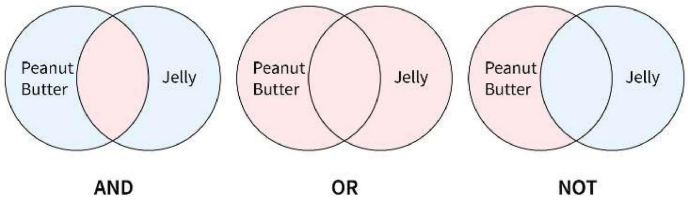
- It views the document as the **set of terms and creates the indexing** required for the Boolean retrieval model.
- The text data is represented in the form of a **matrix** where **rows** of the matrix represent the sentences and the **columns** of the matrix represent the word for the data which needs to be analyzed and retrieved and the **values** of the matrix represent the number of occurrences of the words.
- This model has **good precision** as the documents are retrieved if the condition is matched but, it **doesn't scale well** with the size of the corpus, and an inverted index can be used as a good alternative method.

Processing the data for Boolean retrieval model

- We should **strip** unwanted characters/markup like HTML tags, punctuation marks, numbers, etc. before breaking the corpus into tokens/keywords on whitespace.
- **Stemming** needs to be done and then common stopwords are to be removed depending on the application need
- The term document incidence matrix or **inverted index** (with the keyword a list of docs containing it) is built.
- Then the common queries/phrases may be detected using a **domain-specific dictionary** if needed.

Example of Information Retrieval in Boolean Model

- For example, the term **Peanut Butter individually** (or **Jelly** individually) defines all the documents with the term Peanut Butter (or Jelly) alone and indexes them.
- If the information needed is based on **Peanut Butter AND Jelly**, we will be giving a set of documents that contain **both** the words and so the query with the keywords Peanut Butter AND Jelly will be giving a set of **documents** that are having the **both the words** Peanut Butter AND Jelly.
- Using OR the search will return documents containing **either Peanut Butter or documents containing Jelly** or documents containing both Peanut Butter and Jelly.



Advantages of Boolean Model

- It is **easy** to implement and it is computationally **efficient**. Hence, it is the **standard** model for the current large-scale, operational retrieval systems and many of the major online information services use it.
- It enables users to **express structural and conceptual constraints** to describe important linguistic features. Users find that synonym specifications (reflected by OR-clauses) and phrases (represented by proximity relations) are useful in the formulation of queries
- The Boolean approach possesses a **great expressive power** and clarity. Boolean retrieval is **very effective** if a query requires an exhaustive and unambiguous selection.
- The Boolean method offers a **multitude of techniques to broaden or narrow down a query**.
- The Boolean approach can be especially **effective** in the **later stages of the search process**, because of the clarity and exactness with which relationships between concepts can be represented.

Shortcomings of Standard Boolean Approach

- Users find it **difficult to construct effective Boolean queries** for several reasons. Users are using the natural language terms AND, OR, or NOT that have different meanings when used in a query.
- Hence the users will **make errors** when they form a Boolean query because they resort to their own knowledge of English.

Advantages of the Boolean Mode

The advantages of the Boolean model are as follows –

- The simplest model, which is based on sets.
- Easy to understand and implement.
- It only retrieves exact matches
- It gives the user, a sense of control over the system.

Disadvantages of the Boolean Model

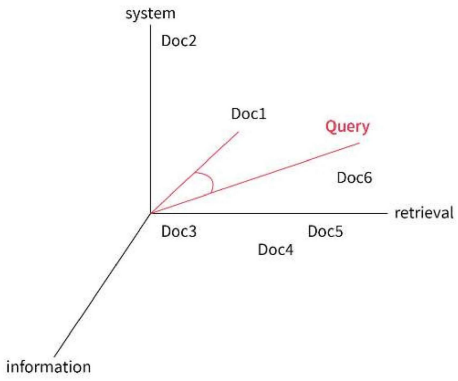
The disadvantages of the Boolean model are as follows –

- The model's similarity function is Boolean. Hence, there would be no partial matches. This can be annoying for the users.
- In this model, the Boolean operator usage has much more influence than a critical word.
- The query language is expressive, but it is complicated too.
- No ranking for retrieved documents.

Vector Space Model

Also called term vector models, the vector space model is an **algebraic model for representing text documents** (or also many kinds of multimedia objects in general) as vectors of identifiers such as index terms.

The vector space model is based on the **notion of similarity between the search document** and the representative query prepared by the user which should be like the documents needed for information retrieval.



Consider the following important points to understand more about the Vector Space Model –

- The index representations (documents) and the queries are considered as vectors embedded in a high-dimensional Euclidean space.
- The similarity measure of a document vector to a query vector is usually the cosine of the angle between them.

Cosine Similarity Measure Formula

Cosine is a normalized dot product, which can be calculated with the help of the following formula –

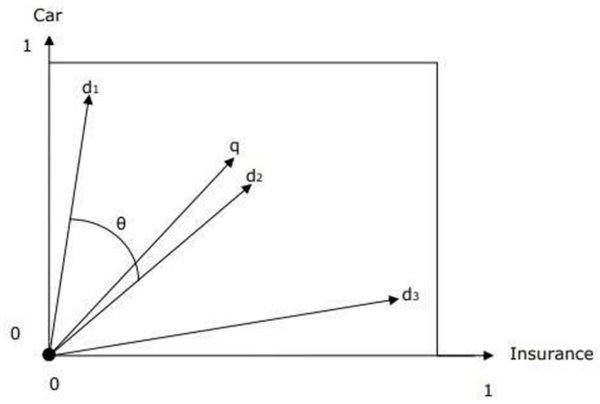
$$Score(\vec{d}\vec{q}) = \frac{\sum_{k=1}^{m} d_k \cdot q_k}{\sqrt{\sum_{k=1}^{m} (d_k)^2} \cdot \sqrt{\sum_{k=1}^{m} m(q_k)^2}}$$

$$Score(\vec{d}\vec{q}) = 1 \text{ when } d = q$$

$$Score(\vec{d}\vec{q}) = 0 \text{ when } d \text{ and } q \text{ share no items}$$

Vector Space Representation with Query and Document

The query and documents are represented by a two-dimensional vector space. The terms are *car* and *insurance*. There is one query and three documents in the vector space.



The top ranked document in response to the terms car and insurance will be the document d_2 because the angle between q and d_2 is the smallest. The reason behind this is that both the concepts car and insurance are salient in d_2 and hence have the high weights. On the other side, d_1 and d_3 also mention both the terms but in each case, one of them is not a centrally important term in the document.

Index Creation for Terms in the Vector Space Model

The **creation** of the indices for the vector space model **involves** lexical scanning, morphological analysis, and term value computation.

- **Lexical scanning** is the creation of individual word documents to identify the **significant terms and morphological analysis reduces** to reduce different word forms to common stems and then compute the values of terms on the basis of stemmed words.
- The terms of the query are also **weighted** to take into account their **importance**, and they are computed by using the **statistical distributions of the terms** in the collection and in the documents.
- The vector space model assigns a **high ranking score** to a document that contains only a few of the query terms if these terms occur **infrequently** in the collection of the original corpus but frequently in the document.

Assumptions of the Vector Space Model

- The **more similar** a document vector is to a query vector, the more likely it is that the document is **relevant** to that query.
- The **words** used to define the dimensions of the space are **orthogonal or independent**.
- The **similarity** assumption is an approximation and **realistic** whereas the assumption that words are pairwise **independent doesn't hold true** in realistic scenarios.

Disadvantages of Vector Space Model

- Long documents are **poorly represented** because they have poor similarity values due to a small scalar product and a **large dimensionality of the terms** in the model.
- Search keywords must be **precisely designed** to match document terms and the word substrings might result in a **false positive match**.
- **Semantic sensitivity:** Documents with similar context but different term vocabulary won't be associated resulting in **false negative matches**.
- The **order in which the terms appear** in the document is lost in the vector space representation.
- **Weighting** is intuitive but not represented **formally** in the model.

- **Issues with implementation:** Due to the need for the similarity metric calculation and in turn storage of all the values of all vector components, it is **problematic** in case of **incremental updates** of the index
    - Adding a **single new document** changes the document frequencies of terms that occur in the document, which **changes the vector lengths of every document** that contains one or more of these terms.

Probabilistic Model

Probabilistic models provide the **foundation for reasoning under uncertainty** in the realm of information retrieval.

Let us understand why there is **uncertainty** while retrieving documents and the basis for probability models in information retrieval.

Uncertainty in retrieval models: The probabilistic models in information retrieval are built on the idea that the process of retrieval is inherently uncertain from multiple standpoints:

- There is uncertainty in the **understanding of user's information needs** - We can not sure that the user mapped their needs into the query they have presented.
- Even if the query represents the need well, there is **uncertainty in the estimation of document relevance** for the query which stems from either the uncertainty from the selection of the document representation or the **uncertainty from matching the query and documents**.

Basis of probabilistic retrieval model: Probabilistic model is based on the **Probability Ranking Principle** which states that an information retrieval system is supposed to rank the documents based on their **probability of relevance to the query given** all the other pieces of evidence available.

- Probabilistic information retrieval models **estimate how likely it is that a document** is relevant for a query.
- There may be a variety of sources of evidence that are used by the probabilistic retrieval methods and the most common one is the **statistical distribution of the terms in both** the relevant and non-relevant documents.
- Probabilistic information models are also among the **oldest and best performing** and most widely used IR models.

Types of Probabilistic information retrieval models: The classic probabilistic models (BIM, Two Poisson, BM11, BM25), The Language models for information retrieval, and the Bayesian networks-based models for information retrieval.

User Query Improvement

The primary goal of any information retrieval system must be accuracy − to produce relevant documents as per the user's requirement. However, the question that arises here is how can we improve the output by improving user's query formation style. Certainly, the output of any IR system is dependent on the user's query and a well-formatted query will produce more accurate results. The user can improve his/her query with the help of *relevance feedback*, an important aspect of any IR model.

Relevance Feedback

Relevance feedback takes the output that is initially returned from the given query. This initial output can be used to gather user information and to know whether that output is relevant to perform a new query or not. The feedbacks can be classified as follows −

- **Explicit Feedback**

It may be defined as the feedback that is obtained from the assessors of relevance. These assessors will also indicate the relevance of a document retrieved from the query. In order to improve query retrieval performance, the relevance feedback information needs to be interpolated with the original query.

Assessors or other users of the system may indicate the relevance explicitly by using the following relevance systems –

- Binary relevance system − This relevance feedback system indicates that a document is either relevant (1) or irrelevant (0) for a given query.
- Graded relevance system − The graded relevance feedback system indicates the relevance of a document, for a given query, on the basis of grading by using numbers, letters or descriptions. The description can be like "not relevant", "somewhat relevant", "very relevant" or "relevant".

- **Implicit Feedback**

It is the feedback that is inferred from user behavior. The behavior includes the duration of time user spent viewing a document, which document is selected for viewing and which is not, page browsing and scrolling actions, etc. One of the best examples of implicit feedback is *dwell time*, which is a measure of how much time a user spends viewing the page linked to in a search result.

- **Pseudo Feedback**

It is also called Blind feedback. It provides a method for automatic local analysis. The manual part of relevance feedback is automated with the help of Pseudo relevance feedback so that the user gets improved retrieval performance without an extended interaction. The main advantage of this feedback system is that it does not require assessors like in an explicit relevance feedback system.

Consider the following steps to implement this feedback –

- Step 1 − First, the result returned by initial query must be taken as relevant result. The range of relevant result must be in top 10-50 results.
- Step 2 − Now, select the top 20-30 terms from the documents using for instance term frequency(tf)-inverse document frequency(idf) weight.
- Step 3 − Add these terms to the query and match the returned documents. Then return the most relevant documents.

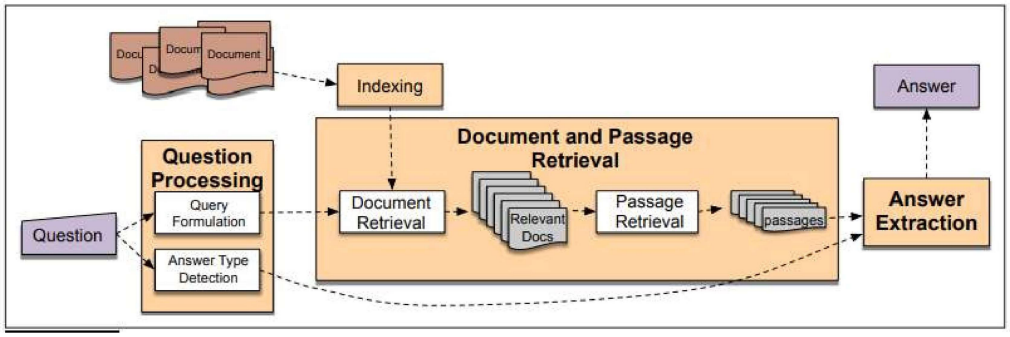
IR-BASED QUESTION ANSWERING

The goal of information retrieval-based question answering is to answer a user's question by finding short text segments on the web or some other collection of documents. The figure below shows some sample factoid questions and their answers.

Question	Answer
Where is the Louvre Museum located?	In Paris, France
What's the abbreviation for limited partnership?	L.P.
What are the names of Odin's ravens?	Huginn and Muninn
What currency is used in China?	The yuan
What kind of nuts are used in marzipan?	Almonds

IR-based Question Answering has three phases:

- Question processing
- Passage retrieval and ranking
- Answer extraction



IR-based QA System

Question Processing

The main goal of the question-processing phase is to extract the query: the keywords passed to the IR system to match potential documents. Some systems additionally extract further information such as:

- **answer type:** the entity type (person, location, time, etc.) of the answer.
- **focus:** the string of words in the question that is likely to be replaced by the answer in any answer string found.
- **question type:** is this a definition question, a math question, or a list question?

For example, for the question

"Which US state capital has the largest population?" the query processing might produce:

query: "US state capital has the largest population"

answer type: city

focus: state capital

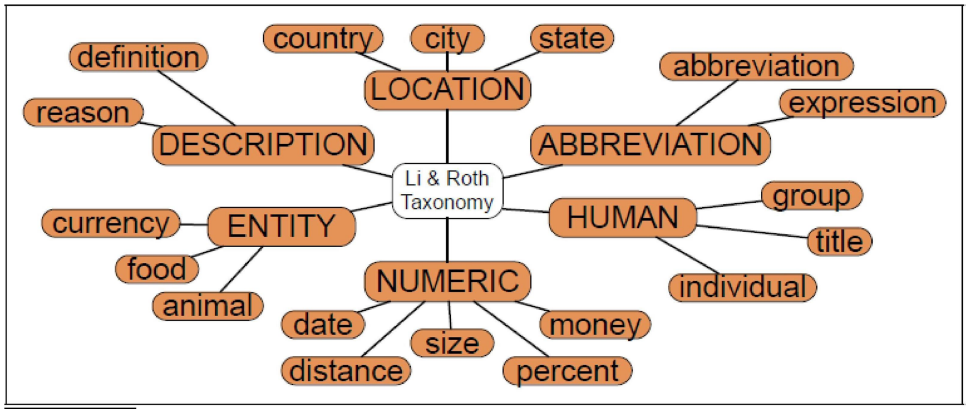
Query formulation is the task of creating a query—a list of tokens— to send to an information retrieval system to retrieve documents that might contain answer strings. For question answering from the web, we can simply pass the entire question
to the web search engine, at most perhaps leaving out the question word (where, when, etc.). When it comes to answering questions from smaller collections of documents, such as corporate information sites or Wikipedia, it is common practice to employ an information retrieval (IR) engine for the purpose of indexing and searching these articles. Typically, this involves utilizing the standard TF-IDF cosine matching technique. Query expansion is a necessary step in information retrieval as the diverse nature of web content often leads to several variations of an answer to a given question. While the likelihood of finding a matching response to a question is higher on the web due to its vastness, smaller document sets may have just a single occurrence of the desired answer. Query expansion methods involve the addition of query keywords with the aim of improving the likelihood of finding a relevant answer. This can be achieved by including morphological variations of the content words in the inquiry or using synonyms obtained from a dictionary.

Ex: The question *"When was the laser invented?"* might be reformulated as *"the laser was invented"*; the question *"Where is the Valley of the Kings?"* as "*the Valley of the Kings is located in*"

Question&Answer Type Detection: Some systems make use of question classification, the task of finding the answer classification answer type, and the named entity categorizing the answer. A question like *"Who founded Virgin Airlines?"* expects an answer of type PERSON. A question like *"What Canadian city has the largest population?"* expects an answer of type CITY. If we know thatthe answer type for a question is a person, we can avoid examining every sentence in the document collection, instead focusing on sentences mentioning people. We can also use a larger hierarchical answer

type set of answer types called an answer type taxonomy. Such taxonomics can be built taxonomy automatically, from resources like WordNet, they can be designed by hand. In this hierarchicaltagset, each question can be labeled with a coarse-grained tag like HUMAN or a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on.

The HUMAN:DESCRIPTION type is often called a BIOGRAPHY question because theanswer is required to give a brief biography of the person rather than just a name.Question classifiers can be built by hand-writing rules like the following rulefor detecting the answer type BIOGRAPHY:*who (was / are / were):* PERSON.



Different Answer Types

Feature-based methods rely on words in the questions and their embeddings, the part-of-speech of each word, and named entities in the questions. Often, a single word in the question gives extra information about the answer type, and its identity is used as a feature. This word is sometimes called the answer type word or question headword, and may be defined as the headword of the first NP after the question's wh-word; headwords are indicated in boldface in the following examples:

- *Which **city** in China has the largest number of foreign financial companies?*
- *What is the state **flower** of California?*

Document and Passage Retrieval

The IR query produced from the question processing stage is sent to an IR engine,resulting in a **set of documents** ranked by their relevance to the query. Becausemost answer-extraction methods are designed to apply to smaller regions such aspassages and paragraphs, QA systems next divide the top n documents into smaller passages such as sections, paragraphs, or sentences. These might be already segmented in the source document or we might need to run a paragraph segmentation algorithm. The simplest form of **passage retrieval** is then to simply pass along every stage to the answer extraction stage. A more sophisticated variant is to filter thepassages by running a named entity or answer type classification on the retrievedpassages, discarding passages that don't contain the answer type of the question. It'salso possible to use supervised learning to fully rank the remaining passages, usingfeatures like:

- The number of named entities of the right type in the passage
- The number of question keywords in the passage
- The longest exact sequence of question keywords that occurs in the passage
- The rank of the document from which the passage was extracted
- The proximity of the keywords from the original query to each other
- The number of n-grams that overlap between the passage and the question

For question answering from the web, we can instead take **snippets** from a Websearch engine as the passages.

Answer Extraction

The final stage of question answering is to extract a specific answer from the passage, for example responding 29,029 feet to a question like *"How tall is Mt. Everest?"*. This task is commonly modeled by span labeling: given a passage, identifying the **span** of text which constitutes an answer. A simple baseline algorithm for answer extraction is to run a

named entity tagger on the candidate passage and return whatever span in the passage is the correct answer type. Thus, in the following examples, the underlined named entities would be extracted from the passages as the answer to the HUMAN and DISTANCE-QUANTITY questions:

"Who is the prime minister of India?"
Manmohan Singh, Prime Minister of India, had told left leaders that the deal would not be renegotiated.

"How tall is Mt. Everest?"
The official height of Mount Everest is 29029 feet

Unfortunately, the answers to many questions, such as DEFINITION questions, don't tend to be of a particular named entity type. For this reason modern work on answer extraction uses more sophisticated algorithms, generally based on supervised
learning.

Feature-based Answer Extraction

Supervised learning approaches to answer extraction train classifiers to decide if aspan or a sentence contains an answer. One obviously useful feature is the answer type feature of the above baseline algorithm.features in such classifiers include:

- **Answer type match:** True if the candidate answer contains a phrase with the correct answer type.
- **Pattern match:** The identity of a pattern that matches the candidate answer.
- **Number of matched question keywords:** How many question keywords are contained in the candidate answer.
- **Keyword distance:** The distance between the candidate answer and query keywords.
- **Novelty factor:** True if at least one word in the candidate answer is novel, that is, not in the query.
- **Apposition features:** True if the candidate answer is an appositive to a phrase containingmany question terms. Can be approximated by the number of questionterms separated from the candidate answer through at most three words andone comma
- **Punctuation location:** True if the candidate answer is immediately followed by a comma, period, quotation marks, semicolon, or exclamation mark.
- **Sequences of question terms:** The length of the longest sequence of question terms that occurs in the candidate answer.

KNOWLEDGE-BASED QUESTION ANSWERING

Knowledge based question answering (KBQA) is a complex task for natural language understanding. KBQA is the task of finding answers to questions by processing a structured knowledge base.Like the textbasedparadigm for question answering, this approach dates back to the earliest daysof natural language processing, with systems like BASEBALL that answered questions from a structured database of baseball games and stats.Systems for mapping from a text string to any logical form are called *semantic parsers.* Semantic parsers for question answering usually map either to someversion of predicate calculus or a query language like SQL or SPARQL.

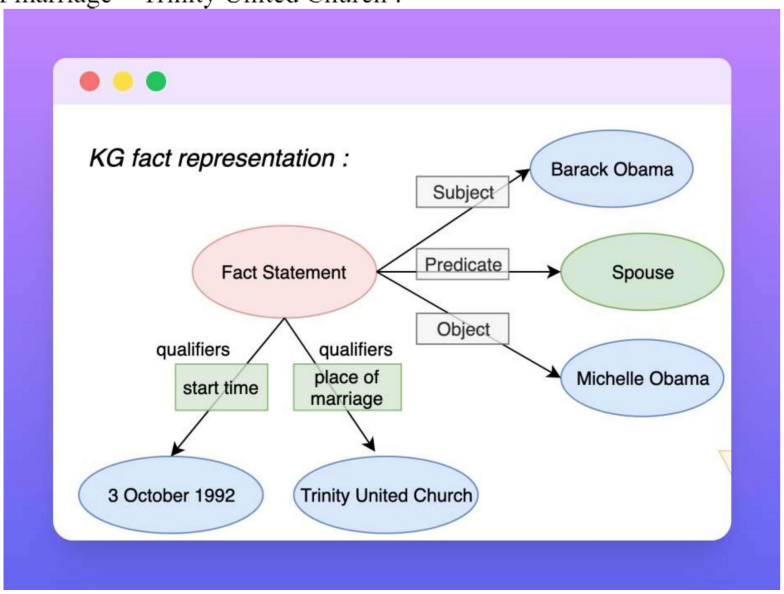
The knowledge base:

- is a comprehensive repository of information about a given domain or a number of domains,
- reflects the ways we model knowledge about a given subject or subjects, in terms of concepts, entities, properties, and relationships,
- enables us to use this structured knowledge where appropriate, e.g., answering factoid questions

The question answerer:

- validates questions against a preconfigured list of question templates, disambiguates entities using Entity Linking, and answers questions asked in natural language,
- can be used with knowledge base like Wikidata (English, Russian) and DBPedia (Russian).

A knowledge base (KB) is a structured database that contains a collection of facts in the form <*subject, relation, object*> , where each fact can have properties attached called *qualifiers*.
For example, the sentence *"Barack Obama got married to Michelle Obama on 3 October 1992 at Trinity United Church"* can be represented by the tuple <*Barack Obama, Spouse, Michelle Obama*> , with the qualifiers start time = 3 October 1992 and place of marriage = Trinity United Church .



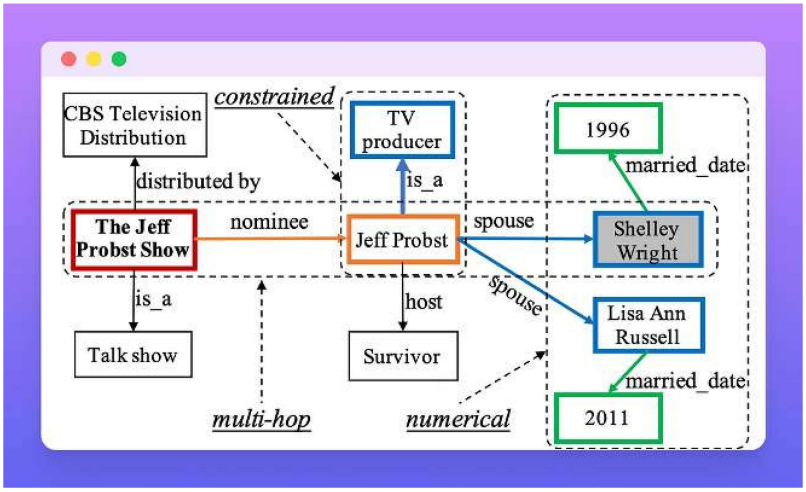
Representation of a fact with its qualifiers.

Simple questions vs complex questions
Early works on KBQA focused on simple question answering, where there's only a single fact involved. For example, *"Where was JK Rowling born?"* is a simple question that can be answered using just the fact <J.K. Rowling, birthplace, United Kingdom>.

Recently, attention shifted to answering complex questions. Generally, complex questions involve multi-hop reasoning over the KB, constrained relations, numerical operations, or some combination of the above.

Let's see an example of complex KBQA with the question *"Who is the first wife of the TV producer that was nominated for The Jeff Probst Show?"*. This question requires:

- **Constrained relations**: We are looking for the TV producer that was nominated for The Jeff Probst Show, thus we are looking for an entity with a nominee link to the The Jeff Probst Show and that is a TV producer.
- **Multi-hop reasoning**: Once we find the TV producer, we need to find his wives.
- **Numerical operations**: Once we find the TV producer's wives, we are looking for the first wife, thus we need to compare numbers and generate a ranking.



An example of complex KBQA for the question "Who is the first wife of the TV producer that was nominated for The Jeff Probst Show?". Multi-hop reasoning, constrained relations, and numerical operation are highlighted in black dotted boxes.

KBQA approaches

There are two mainstream approaches for complex KBQA. Both these two approaches start by recognizing the subject in the question and linking it to an entity in the KB, which will be called *topic entity*. Then, they derive the answers within the KB neighborhood of the topic entity:

- By executing a parsed logic form, typical of **semantic parsing-based methods (SP-based methods).** It follows a **parse-then-execute paradigm.**
- By reasoning in a question-specific graph extracted from the KB and ranking all the entities in the extracted graph based on their relevance to the question, typical of **information retrieval-based methods (IR-based methods).** It follows a **retrieval-and-rank paradigm.**

Semantic Parsing-based Methods

This category of methods aims at parsing a natural language utterance into logic forms. They predict answers via the following steps:

1. Parse the natural language question into an uninstantiated logic form (e.g. a SPARQL query template), which is a syntactic representation of the question without the grounding of entities and relations.
2. The logic form is then instantiated and validated by conducting some semantic alignments to structured KBs via KB grounding (obtaining, for example, an executable SPARQL query).
3. The parsed logic form is executed against KBs to generate predicted answers.

Information Retrieval-based Methods

IR-based methods directly retrieve and rank answers from the KBs considering the information conveyed in the questions. They consist of the following steps:

1. Starting from the topic entity, the system first extracts a question-specific graph from KBs, ideally including all question-related entities and relations as nodes and edges.
2. Next, the system encodes input questions into vectors representing reasoning instructions.
3. A graph-based reasoning module conducts semantic matching via vector-based computation to propagate and then aggregate the information along the neighboring entities within the graph.
4. An answer ranking module is utilized to rank the entities in the graph according to the reasoning status at the end of the reasoning phase. The top-ranked entities are predicted as the answers to the question.

Pros and Cons of the two KBQA approaches

Semantic parsing-based methods produce a more interpretable reasoning process thanks to logic forms. However, their heavy reliance on the design of logic forms and parsing modules is usually a bottleneck for performance improvements.IR-based methods naturally fit into popular end-to-end training, making them easy to train. However, the black-box style of the reasoning model makes the intermediate reasoning less interpretable.

LANGUAGE MODELS FOR QA

Language Models for Information Retrieval

Language modeling is a **formal probabilistic retrieval framework** based on concepts in speech recognition and natural language processing. However, language models have been used for **speech recognition** and natural language processing tasks (machine translation) for more than **thirty years**. They have been used in the IR domain since the 2000s with great success in **recent times**.

- The language models for information retrieval are based on the **idea** that a query is **considered generated from an ideal document that satisfies the information need** and the IR system's job is then to estimate the **likelihood** of each document in the **collection being the ideal document** and rank the documents in ascending or decreasing order.
    - Each document is viewed as a **language sample**, each query as a **generation process**. The retrieved documents are ranked based on the probabilities of producing a query from the corresponding language models of these documents.
    - Many research studies have also confirmed that **language modeling techniques are preferred over tf-idf** (term frequency-inverse document frequency) weights because of empirical performance as well as the probabilistic meaning that can be formally derived from a language modeling framework.

The majority of language modeling approaches to information retrieval can be categorized into one of four groups:

- The **generative query likelihood approach**, which ranks based on the likelihood of a document language model generating the query
- The **generative document likelihood approach**, which ranks based on the likelihood of a query language model generating a document
- The **comparative approach**, which ranks based on the similarity between the query language model and document language model
- The **translation models**, which rank based on the likelihood of the query being viewed as a translation of a document
- The cluster-based language models.

Challenges with Language Modeling

- Formal languages (like a programming language) are precisely defined. All the words and their usage is predefined in the system. Anyone who knows a specific programming language can understand what's written without any formal specification.
- Natural language, on the other hand, isn't designed; it evolves according to the convenience and learning of an individual. There are several terms in natural language that can be used in a number of ways. This introduces ambiguity but can still be understood by humans.
- Machines only understand the language of numbers. For creating language models, it is necessary to convert all the words into a sequence of numbers. For the modellers, this is known as encodings.
- Encodings can be simple or complex. Generally, a number is assigned to every word and this is called label-encoding. In the sentence "I love to play cricket on weekends", every word is assigned a number [1, 2, 3, 4, 5, 6]. This is an example of how encoding is done (one-hot encoding).

How does Language Model Works?

- Language Models determine the probability of the next word by analyzing the text in data. These models interpret the data by feeding it through algorithms.

- The algorithms are responsible for creating rules for the context in natural language. The models are prepared for the prediction of words by learning the features and characteristics of a language. With this learning, the model prepares itself for understanding phrases and predicting the next words in sentences.
- For training a language model, a number of probabilistic approaches are used. These approaches vary on the basis of the purpose for which a language model is created. The amount of text data to be analyzed and the math applied for analysis makes a difference in the approach followed for creating and training a language model.
- For example, a language model used for predicting the next word in a search query will be absolutely different from those used in predicting the next word in a long document (such as Google Docs). The approach followed to train the model would be unique in both cases.

Types of Language Models:
There are primarily two types of language models:

1. Statistical Language Models
Statistical models include the development of probabilistic models that are able to predict the next word in the sequence, given the words that precede it. A number of statistical language models are in use already. Let's take a look at some of those popular models:

N-Gram: This is one of the simplest approaches to language modelling. Here, a probability distribution for a sequence of 'n' is created, where 'n' can be any number and defines the size of the gram (or sequence of words being assigned a probability). If n=4, a gram may look like: "can you help me". Basically, 'n' is the amount of context that the model is trained to consider. There are different types of N-Gram models such as unigrams, bigrams, trigrams, etc.

Unigram: The unigram is the simplest type of language model. It doesn't look at any conditioning context in its calculations. It evaluates each word or term independently. Unigram models commonly handle language processing tasks such as information retrieval. The unigram is the foundation of a more specific model variant called the query likelihood model, which uses information retrieval to examine a pool of documents and match the most relevant one to a specific query.

Bidirectional: Unlike n-gram models, which analyze text in one direction (backwards), bidirectional models analyze text in both directions, backwards and forwards. These models can predict any word in a sentence or body of text by using every other word in the text. Examining text bidirectionally increases result accuracy. This type is often utilized in machine learning and speech generation applications. For example, Google uses a bidirectional model to process search queries.

Exponential: This type of statistical model evaluates text by using an equation which is a combination of n-grams and feature functions. Here the features and parameters of the desired results are already specified. The model is based on the principle of entropy, which states that probability distribution with the most entropy is the best choice. Exponential models have fewer statistical assumptions which mean the chances of having accurate results are more.

Continuous Space: In this type of statistical model, words are arranged as a non-linear combination of weights in a neural network. The process of assigning weight to a word is known as word embedding. This type of model proves helpful in scenarios where the data set of words continues to become large and include unique words. In cases where the data set is large and consists of rarely used or unique words, linear models such as n-gram do not work. This is because, with increasing words, the possible word sequences increase, and thus the patterns predicting the next word become weaker.

2. Neural Language Models
These language models are based on neural networks and are often considered as an advanced approach to execute NLP tasks. Neural language models overcome the shortcomings of classical models such as n-gram and are used for complex tasks such as speech recognition or machine translation.

Language is significantly complex and keeps on evolving. Therefore, the more complex the language model is, the better it would be at performing NLP tasks. Compared to the n-gram model, an exponential or continuous space model proves to be a better option for NLP tasks because they are designed to handle ambiguity and language variation.Meanwhile, language models should be able to manage dependencies. For example, a model should be able to understand words derived from different languages.

Language models like GPT-3 can be used to build powerful question answering systems. These systems take a question in natural language as input and generate a relevant and coherent answer. Here's a general approach to building a question answering system using language models:

1. **Data Collection and Preprocessing:*
- Gather a large dataset of question-answer pairs relevant to the domain you're targeting.
- Preprocess the data by cleaning the text, tokenizing sentences, and converting text to a suitable format for model input.

2. **Selecting a Language Model:*
- Choose a suitable language model for your task. GPT-3 is a popular choice, but you can also consider other models like BERT, T5, or RoBERTa.

3. **Fine-Tuning (Optional):*
- If you have domain-specific data, you might fine-tune the chosen language model on your dataset. Fine-tuning can help the model specialize in the specific domain and improve performance.

4. **Input Representation:*
- Tokenize the input question and format it according to the language model's requirements (e.g., adding special tokens like [CLS] and [SEP] for BERT-based models).

5. **Model Inference:*
- Pass the tokenized input through the language model to generate the answer. For GPT-3, you would send a prompt including the question and any additional context if needed.

6. **Post-processing:*
- Extract and process the generated answer from the model's output. This might involve removing unnecessary tokens, ensuring coherence, and improving readability.

7. **Answer Ranking (Optional):*
- If your system generates multiple potential answers, you can implement a ranking mechanism to select the most relevant and accurate answer. This could involve scoring based on context, confidence scores from the model, or other criteria.

8. **User Interaction:*
- Design an interface or integration where users can input their questions and receive answers. This could be a web application, chatbot, or any other platform suitable for your use case.

9. **Evaluation and Iteration:*
- Regularly evaluate the performance of your question answering system using human evaluators or automated metrics. Gather feedback and make improvements to the system as needed.

10. **Scaling and Deployment:*
- Once you're satisfied with the system's performance, deploy it to your desired platform. Ensure that the deployment is scalable and can handle user traffic effectively.

Remember that building an effective question-answering system involves not only technical aspects but also careful consideration of user experience and the specific requirements of your application. Additionally, be aware of ethical considerations and potential biases in the language model's responses.

Classic QA Models

Classic QA models are more structured and rule-based compared to the more flexible and language-driven approaches of modern neural language models. They can be effective for specific domains or applications where structured data is available and where users have well-defined queries. However, they may struggle with handling ambiguous or complex natural language queries that don't adhere to predefined patterns.

It's important to note that classic QA models require a significant amount of manual engineering and domain expertise to design effective rules and patterns for question analysis, answer extraction, and ranking. Modern neural language models like GPT-3 have the advantage of being able to learn from data and handle a wider range of language patterns, which can make them more versatile for general question-answering tasks.

GPT-QA

GPT-QA refers to a "Generative Pre-trained Transformer for Question Answering." It is a variant or adaptation of the GPT (Generative Pre-trained Transformer) model specifically designed for the task of question answering.

GPT (Generative Pre-trained Transformer):

GPT is a class of language models developed by OpenAI. It's based on the Transformer architecture, which is designed to process sequences of data, making it particularly well-suited for natural language understanding and generation tasks. GPT models are pre-trained on a vast amount of text data from the internet, which allows them to learn grammar, syntax, semantics, and other language patterns.

QA (Question Answering):
Question answering is a task in natural language processing where a machine is given a question in natural language and is expected to provide a relevant and accurate answer. QA models typically analyze the question and a given context (such as a passage of text) to generate an answer that addresses the question.

Combining GPT and QA:
To create a "GPT QA" system, you would take advantage of the GPT model's generative capabilities and adapt it for question-answering tasks. Here's how this could work:

1. **Pre-training**: The GPT model undergoes its initial pre-training process on a large dataset of text. During this phase, the model learns language patterns and general knowledge from the diverse text it's exposed to.
2. **Fine-tuning for QA**: After pre-training, the model can be fine-tuned on a dataset specifically focused on question answering. This dataset would include pairs of questions and their corresponding answers. The model learns to generate answers that are contextually relevant and accurate based on the input questions.
3. **Inference**: During inference, the "GPT QA" model takes a question as input. It processes the question and any associated context (such as a passage of text) and generates a response that serves as the answer to the question.
4. **Response Generation**: The model generates the answer by leveraging the knowledge it gained during pre-training and the fine-tuning process. It considers the context provided and generates a coherent and contextually appropriate response.

The result is a system that can generate human-like answers to questions based on its understanding of language and the information it has been trained on. This "GPT QA" system can be applied to various tasks, including chatbots, customer support, information retrieval, and more.

BERT
BERT, which stands for "Bidirectional Encoder Representations from Transformers," is a groundbreaking natural language processing (NLP) model introduced by researchers at Google in 2018. BERT is designed to understand and represent the context of words in a sentence by considering both the left and right context, unlike previous models that only looked at the left or right context.

Here's a detailed explanation of BERT and its key components:

1. **Bidirectional Encoding**: BERT's main innovation is its bidirectional approach to language modeling. Traditional models like the ones based on the Transformer architecture (such as GPT) typically read text in one direction (left-to-right or right-to-left). BERT, on the other hand, reads text in both directions simultaneously. This means it considers all the words in a sentence at once to capture richer context and relationships.
2. **Transformer Architecture**: BERT is built upon the Transformer architecture. The Transformer uses self-attention mechanisms to weigh the importance of different words in a sentence relative to each other. This allows BERT to capture long-range dependencies and understand the relationships between words.
3. **Pre-training**: BERT undergoes a two-step training process. In the pre-training phase, it is trained on a massive amount of text data. During this phase, the model learns to predict missing words in sentences (masked language model pre-training) and also learns to predict whether sentences come in a continuous order (next sentence prediction). The pre-training process helps BERT learn the contextual relationships between words.
4. **Fine-tuning**: After pre-training, BERT can be fine-tuned on specific NLP tasks, such as sentiment analysis, named entity recognition, question answering, and more. During fine-tuning, the model is trained on task-specific data to adapt its representations and predictions for the specific task at hand.
5. **Tokenization**: BERT tokenizes input text into subword units, such as words and subwords. Each token is associated with an embedding vector that captures its meaning and context. BERT can handle variable-length input sequences, and it uses special tokens to indicate the start and end of sentences.
6. **Layers and Attention**: BERT consists of multiple layers, each containing self-attention mechanisms and feedforward neural networks. The self-attention mechanism allows BERT to weigh the importance of words based

on their relationships within a sentence. The outputs from all layers are combined to create contextualized word representations.

7. **Contextualized Embeddings**: BERT produces contextualized word embeddings, which means the embeddings are different for the same word depending on its context in a sentence. This enables BERT to capture nuances and polysemy (multiple meanings) in language.
8. **Applications**: BERT's bidirectional nature and contextual embeddings make it highly effective for a wide range of NLP tasks, including question answering, sentiment analysis, text classification, text generation, and more. By fine-tuning BERT on specific tasks, it can achieve state-of-the-art performance on various benchmarks.

BERT has significantly advanced the field of NLP and has paved the way for many subsequent models and research efforts. Its ability to capture bidirectional context has led to improved language understanding and generation capabilities in a variety of applications.

BERT ARCHITECTURE

The BERT (Bidirectional Encoder Representations from Transformers) architecture is based on the Transformer architecture. BERT builds upon this architecture with specific modifications to enable bidirectional context modeling. Here's a detailed overview of the BERT architecture:

1. **Input Encoding and Tokenization:**

- BERT takes variable-length text input, which is tokenized into subword units like words and subwords using WordPiece tokenization.

- Special tokens are added to mark the start and end of sentences, as well as to distinguish between different sentences in a pair.

2. **Embedding Layer:**

- Each token is associated with an embedding vector that combines a word embedding, a positional embedding (to capture token position), and segment embeddings (to distinguish between sentence pairs).

3. **Transformer Encoder Stack:**

- BERT consists of multiple identical layers, each containing a self-attention mechanism and feedforward neural networks.

- Each layer processes the token embeddings sequentially.

4. **Self-Attention Mechanism:**

- Self-attention allows each token to consider the other tokens in the input sequence while calculating its representation.

- BERT uses multi-head self-attention, where the model learns multiple sets of attention weights to capture different types of relationships between words.

5. **Position-wise Feedforward Networks:**

- After self-attention, each token's representation passes through a position-wise feedforward neural network, which includes two fully connected layers.

- The feedforward network introduces non-linearity and further contextualizes token representations.

6. **Layer Normalization and Residual Connections:**

- Layer normalization is applied after each sub-layer (self-attention and feedforward) to stabilize training.

- Residual connections (skip connections) are used to ensure that original token embeddings are preserved and facilitate gradient flow during training.

7. **Output Pooling:**

- For certain tasks (e.g., sentence classification), BERT employs a pooling layer to aggregate token representations into a fixed-size representation for the entire sequence.

- Common pooling strategies include max-pooling and mean-pooling.

8. **Task-Specific Heads:**

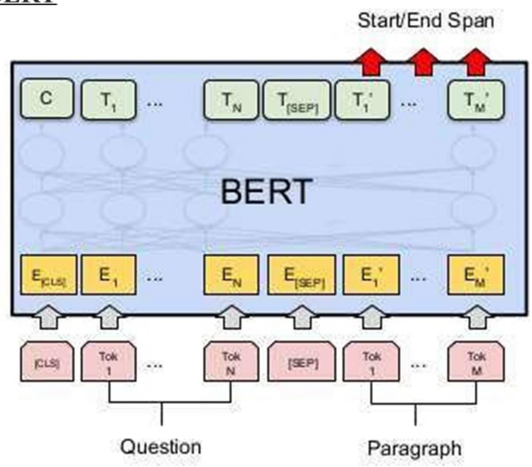
- BERT can be fine-tuned for various NLP tasks by adding task-specific layers on top of the BERT encoder.

- For example, for text classification tasks, a linear layer and softmax activation can be added to predict class labels.

The key innovation of BERT is its bidirectional approach, which allows it to capture contextual information from both the left and right contexts of a token. This contrasts with traditional models that only consider either the left or right context. The bidirectional encoding enables BERT to better understand language nuances, relationships between words, and the broader context within sentences.BERT's architecture has served as a foundation for subsequent advancements in NLP, and its pre-trained representations have proven highly effective for a wide range of downstream tasks through fine-tuning.BERT reads the whole input text sequence altogether unlike other directional models which do the same task from one direction such as left to right or right to left.

BERT can better understand long-term queries and as a result surface more appropriate results. **BERT** models are applied to both organic search results and featured snippets. While you can optimize for those queries, you cannot "optimize for **BERT**."
To simplify: *BERT helps the search engine understand the significance of transformer words like 'to' and 'for' in the keywords used.*

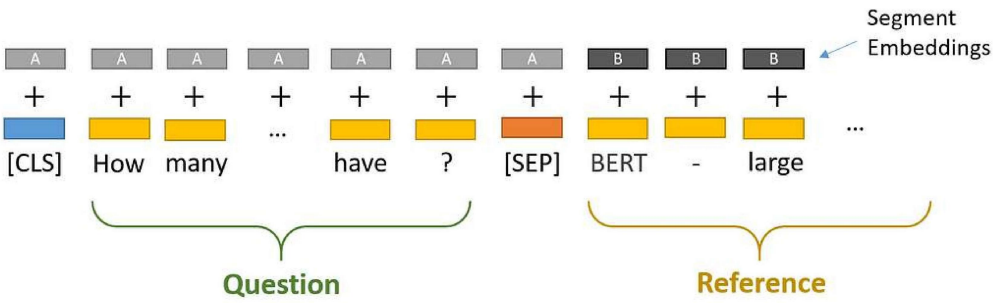
Question Answering System using BERT



Building a Question Answering System with BERT

For the Question Answering System, BERT takes two parameters, the input question, and passage as a single packed sequence. The input embeddings are the sum of the token embeddings and the segment embeddings.

1. **Token embeddings:** A [CLS] token is added to the input word tokens at the beginning of the question and a [SEP] token is inserted at the end of both the question and the paragraph.
2. **Segment embeddings:** A marker indicating Sentence A or Sentence B is added to each token. This allows the model to distinguish between sentences. In the below example, all tokens marked as A belong to the question, and those marked as B belong to the paragraph.



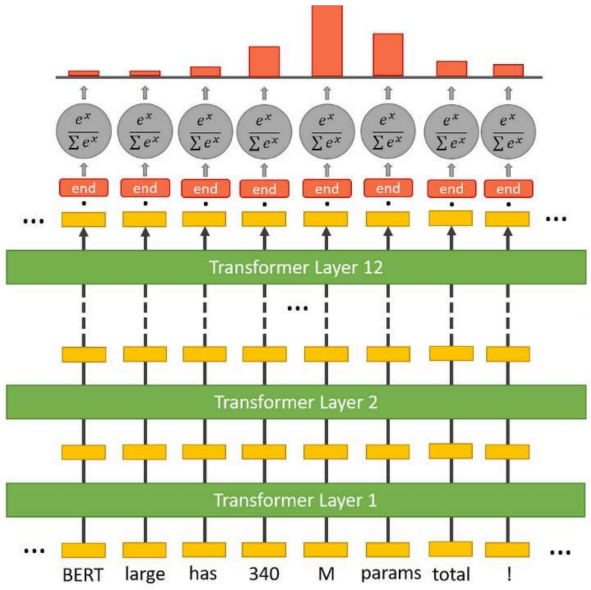
Question: How many parameters does BERT-large have?

Reference Text: BERT-large is really big... it has 24 layers and an embedding size of 1,024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

The two pieces of text are separated by the special [SEP] token.

BERT uses "Segment Embeddings" to differentiate the question from the reference text. These are simply two embeddings (for segments "A" and "B") that BERT learned, and which it adds to the token embeddings before feeding them into the input layer.

Start & End Token Classifiers



Transformer Architecture of Layers to find start-word and end-word

For every token in the text, we feed its final embedding into the start token classifier. The start token classifier only has a single set of weights which applies to every word.After taking the dot product between the output embeddings and the 'start' weights, we apply the softmax activation to produce a probability distribution over all of the words. Whichever word has the highest probability of being the start token is the one that we pick.

T5 (Text-to-Text Transfer Transformer)

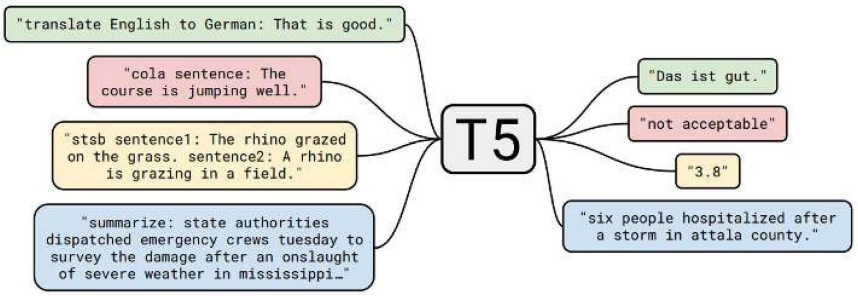
T5 (Text-to-Text Transfer Transformer) is a versatile and powerful natural language processing model developed by Google Research. T5 is designed to frame most NLP tasks as a text-to-text problem, where both the input and output are treated as text sequences. This approach allows T5 to handle a wide range of NLP tasks in a unified manner.

Here's a detailed explanation of the T5 model and its key components:

1. **Text-to-Text Framework**:
    - T5 introduces a unified framework where all NLP tasks are cast as a text generation task. This means that both the input and output are treated as text sequences, which enables T5 to handle tasks like classification, translation, summarization, question answering, and more.
    - The input text includes a prefix indicating the specific task, and the model learns to generate the appropriate output text.
2. **Transformer Architecture**:
    - T5 is built upon the Transformer architecture, which includes self-attention mechanisms and feedforward neural networks.
    - The architecture allows T5 to capture contextual relationships between words and generate coherent and contextually relevant output text.
3. **Pre-training**:
    - T5 undergoes a pre-training phase where it is trained on a large corpus of text data using a denoising autoencoder objective. It learns to reconstruct masked-out tokens in corrupted sentences.
    - The pre-training process helps T5 learn rich representations of language.
4. **Fine-tuning**:
    - After pre-training, T5 is fine-tuned on specific NLP tasks using task-specific datasets.
    - During fine-tuning, the model learns to generate the appropriate output for each task while conditioning on the provided input.
5. **Task-Specific Prompts**:
    - For each task, T5 is provided with a specific prompt that guides it to generate the desired output text.
    - The prompts include task-specific instructions to guide the model's behavior.
6. **Versatility**:
    - T5's text-to-text framework makes it highly versatile. It can be fine-tuned for a wide range of tasks, including text classification, translation, summarization, question answering, sentiment analysis, and more.
    - By using a consistent text generation approach across tasks, T5 simplifies the process of adapting the model to new tasks.
7. **Evaluation and Benchmarks**:
    - T5 has achieved state-of-the-art performance on several NLP benchmarks and competitions.

- It has demonstrated strong performance even when fine-tuned on tasks for which it was not explicitly trained, showcasing its ability to generalize across tasks.

T5's innovative text-to-text approach has demonstrated the potential for a unified framework that can handle diverse NLP tasks. It offers a streamlined way to apply a single model to various tasks by framing them as text generation problems.



Text-Text framework:

T5 uses the same model for all various tasks by the way we tell the model which task to perform by prepending the task prefix which is also a text.

As shown in the above picture if we want to use **T5 for the classification task** of predicting sentence grammatically correct or not, adding the prefix "Cola sentence: "will take care of it and return two texts as output **'acceptable'** or **'not acceptable'**

Interestingly T5 also perform **two sentences similarity regression task** in Text- text framework. They posed this as a classification problem with 21 classes (from 1–5 with 0.2 increments eg:'1.0','1.2','1.4'……'5.0') and asked the model to predict a string and T5 gave SOTA results to this task too.

Similarly for other tasks 'summarize:' which return a summary of the article and for NMT 'translate English to german:'

T5 Pretraining and Finetuning:

Q) What's new in T5?

Ans) Nothing

yeah its true, T5 Uses vanilla Transformer Architecture. Then how they got SOTA results? The main motivation behind T5 work is ….

Given the current landscape of transferlearning for NLP what works best and how far we can push the tools we have? Search Results — Colin Raffel

T5 base with Bert base sized encoder-decoder stacks with 220 million parameters experimented with an all wide variety of NLP techniques during pretraining and fine-tuning.

Summing up Best outcomes from T5 experiments :

1. **Used Large Dataset for Pre-training:** An important ingredient for transfer learning is the unlabeled dataset used for pre-training .T5 uses common crawl web extract text (C4) which results in 800 GB of data after cleaning and deduplication of data. The cleaning process involved deduplication, discarding incomplete sentences, and removing offensive or noisy content.

- **Architectures:** Experimented with encoder-decoder models and decoder-only language models similar toGPTand found encoder-decoder models did well
- **un-supervised objectives:** T5 Uses MLM-Masked Language Modeling as pertaining objective and it worked best for then they also experimented with Permutation Language modeling where XLNET uses this as **un-supervised objectives.**;

Finally,

Insights + Scale = State-of-the-Art

T5 further explores with scaling their models large, with dmodel = 1024, a 24 layer encoder and decoder, and dkv = 128. T5–3Billion variant uses dff = 16,384 and 32-headed attention, which results in around 2.8 billion parameters;

for T5 -11Billion has dff = 65,536 and 128-headed attention producing a model with about 11 billion parameters.

T5 the largest model had 11 billion parameters and achieved SOTA on the GLUE, SuperGLUE, SQuAD, and CNN/Daily Mail benchmarks. **One particularly exciting result was that T5 achieved a near-human score on the Superglue natural language understanding benchmark, which was specifically designed to be difficult for machine learning models but easy for humans.**

1.1. Unified Input & Output Format

- T5 means "**T**ext-**t**o-**T**ext **T**ransfer **T**ransformer": **Every task** considered — including translation, question answering, and classification — is cast as feeding the T5 model **text as input** and training it to **generate some target text.**
- **Translation**: Ask the model to translate the sentence "That is good." from English to German, the model would be **fed** the sequence **"translate English to German: That is good."** and would be trained to **output "Das ist gut."**
- **Text classification**: The model simply predicts a single word corresponding to the target label. For example, on the **MNLI** benchmark, the goal is to predict whether **a premise implies ("entailment"), contradicts ("contradiction"), or neither ("neutral") a hypothesis**. The **input** sequence becomes **"mnli premise: I hate pigeons. hypothesis: My feelings towards pigeons are filled with animosity."**. Only possible labels are "entailment", "neutral", or "contradiction", other outcomes are treated as wrong prediction.
- **Regression**: In STS-B, the goal is to predict a similarity score between 1 and 5. **Increments of 0.2 are used as text prediction**.
- (It is in details for each task how they unify the format, please feel free to read the paper directly if interested.)

1.2. Encoder-Decoder Transformer Model

- T5 uses encoder-decoder Transformer implementation which closely follows the original Transformer, with the exception of below differences:
- (Please feel free to Transformer if interested.)
- But **a simplified layer normalization** is used where the activations are only rescaled and **no additive bias** is applied. After layer normalization, a residual skip connection, originated from ResNet, adds each subcomponent's input to its output.
- Also, instead of using a fixed embedding for each position, **relative position embeddings (Shaw NAACL'18)** produce a different learned embedding **according to the offset (distance) between the "key" and "query"** being compared in the self-attention mechanism.

1.3. Training

- A combination of model and data parallelism are used to train models on "slices" of Cloud TPU Pods. **5 TPU pods** are are multi-rack ML supercomputers that contain **1,024 TPU v3 chips** connected via a high-speed 2D mesh interconnect with supporting CPU host machines.

CHATBOTS

Chatbots are a relatively recent concept and despite having a huge number of programs and NLP tools. An natural language processing chatbot is a software program that can understand and respond to human speech. Bots powered by NLP allow people to communicate with computers in a way that feels natural and human-like — mimicking person-to-person conversations. These clever chatbots have a wide range of applications in the customer support sphere.

NLP chatbots: The first generation of virtual agents

NLP-powered virtual agents are bots that rely on intent systems and pre-built dialogue flows — with different pathways depending on the details a user provides — to resolve customer issues. A chatbot using NLP will keep track of information throughout the conversation and learn as they go, becoming more accurate over time. Here are some of the most important elements of an NLP chatbot.

Key elements of NLP-powered bots

- **Dialogue management:** This tracks the state of the conversation. The core components of dialogue management in AI chatbots include a context — saving and sharing data exchanged in the conversation — and session — *one* conversation from start to finish
- **Human handoff:** This refers to the seamless communication and execution of a handoff from the AI chatbot to a human agent
- **Business logic integration:** It's important that your chatbot has been programmed with your company's unique business logic
- **Rapid iteration:** You want your bot to provide a seamless experience for customers and to be easily programmable. Rapid iteration refers to the fastest route to the right solution
- **Training and iteration:** To ensure your NLP-powered chatbot doesn't go awry, it's necessary to systematically train and send feedback to improve its understanding of customer intents using real-world conversation data being generated across channels
- **Simplicity:** To get the most out of your virtual agent, you'll want it to be set up as simply as possible, with all the functionality that you need — but no more than that. There is, of course, always the potential to upgrade or add new features as you need later on

Benefits of bots

- Bots allow you to communicate with your customers in a new way. Customers' interests can be piqued at the right time by using chatbots.

- With the help of chatbots, your organization can better understand consumers' problems and take steps to address those issues.

- A single operator can serve one customer at a time. On the other hand, a chatbot can answer thousands of inquiries.
- Chatbots are unique in that they operate inside predetermined frameworks and rely on a single source of truth within the command catalog to respond to questions they are asked, which reduces the risk of confusion and inconsistency in answers.

Types of Chatbots

With the help of this experience, we can understand that there are 2 types of chatbots around us: Script-bot and Smart-bot.

Script Bot	Smart Bot
1. Script bots are easy to make 2. Script bots work around a script that is programmed in them 3. Mostly they are free and are easy to integrate into a messaging platform 4. No or little language processing skills 5 Limited functionality 6. Example: The bots that are deployed in the customer care section of various companies	1. Smart bots are flexible and powerful 2. Smart bots work on bigger databases and other resources directly 3 Smart bots learn with more data 4. Coding is required to take this up on board 5 Wide functionality 6. Example: Google Assistant, Alexa, Cortana, Siri, etc.

CCS369 TEXT AND SPEECH ANALYSIS

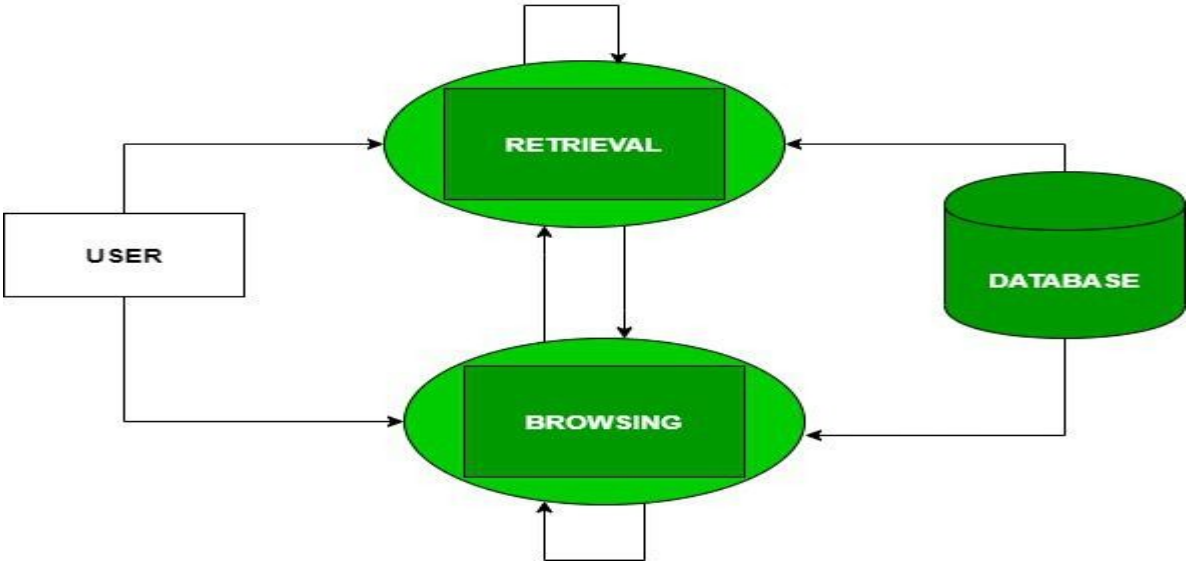
UNIT III QUESTION ANSWERING AND DIALOGUE SYSTEMS

Information retrieval:

What is text information retrieval?

• Text retrieval is to return relevant textual documents from a given collection, according to users' information needs as declared in a query.
• Main differences from database retrieval are concerned with: – Information.
• Unstructured text vs.

Information Retrieval (IR) can be defined as a software program that deals with the organization, storage, retrieval, and evaluation of information from document repositories, particularly textual information. Information Retrieval is the activity of obtaining material that can usually be documented on an unstructured nature **i.e.** usually text which satisfies an information need from within large collections which is stored on computers..

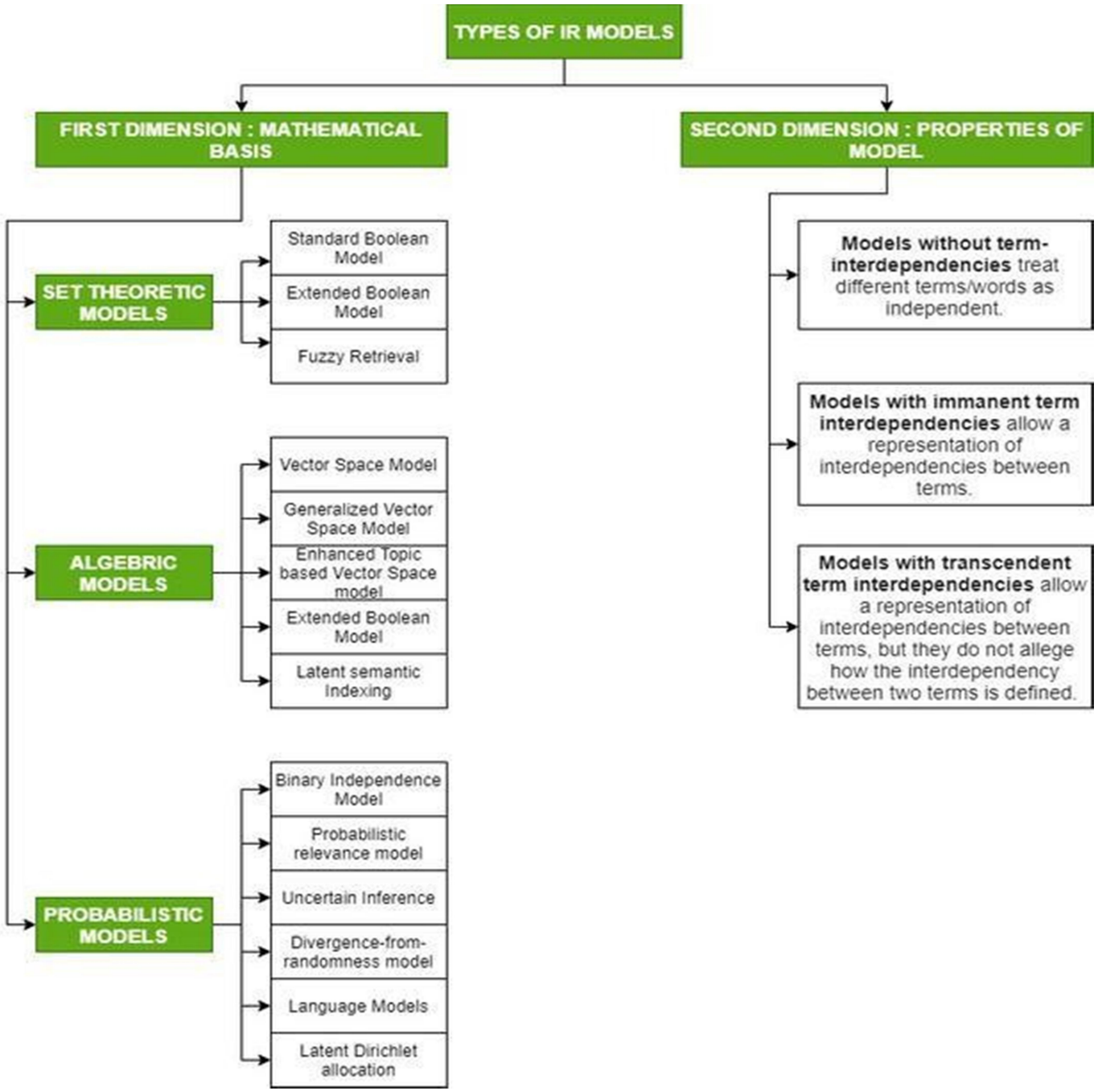


Examples:
Vector-space, Boolean and Probabilistic IR models. In this system, the retrieval of information depends on documents containing the defined set of queries.

What is an IR Model?

An Information Retrieval (IR) model selects and ranks the document that is required by the user or the user has asked for in the form of a query. The documents and the queries are represented in a similar manner, so that document selection and ranking can be formalized by a matching function that returns a **retrieval status value (RSV)** for each document in the collection. Many of the Information Retrieval systems represent document contents by a set of

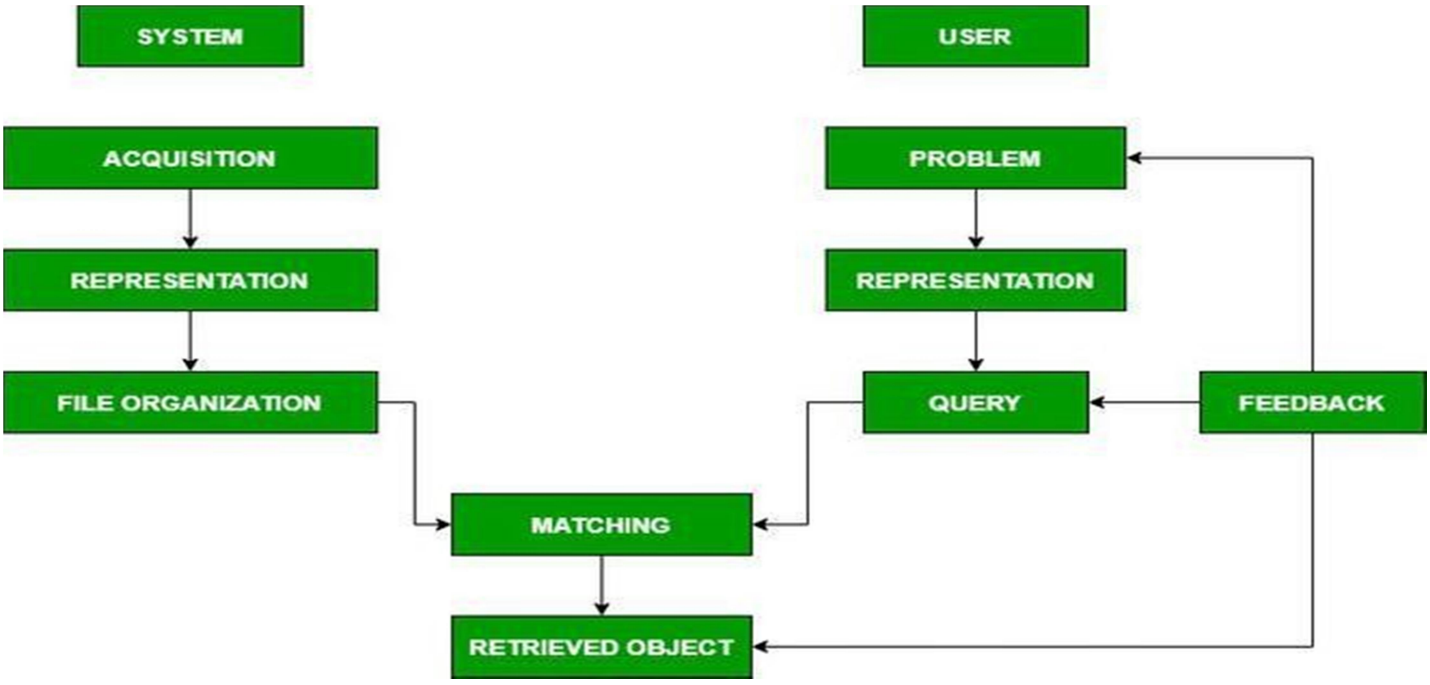
descriptors, called terms, belonging to a vocabulary V. An IR model determines the query-document matching function according to four main approaches:



Components of Information Retrieval/ IR Model

- **Acquisition:** In this step, the selection of documents and other objects from various web resources that consist of text-based documents takes place. The required data is collected by web crawlers and stored in the database.
  - **Representation:** It consists of indexing that contains free-text terms, controlled vocabulary, manual & automatic techniques as well. example: Abstracting contains

summarizing and Bibliographic description that contains author, title, sources, data, and metadata.



- **Representation:** It consists of indexing that contains free-text terms, controlled vocabulary, manual & automatic techniques as well. example: Abstracting contains summarizing and Bibliographic description that contains author, title, sources, data, and metadata.
- **File Organization:** There are two types of file organization methods. i.e. *Sequential*: It contains documents by document data. *Inverted*: It contains term by term, list of records under each term. *Combination* of both.
- **Query:** An IR process starts when a user enters a query into the system. Queries are formal statements of information needs, for example, search strings in web search engines. In information retrieval, a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy.

Difference Between Information Retrieval and Data Retrieval

Information Retrieval	Data Retrieval
The software program that deals with the organization, storage, retrieval, and evaluation of information from document repositories particularly textual information.	Data retrieval deals with obtaining data from a database management system such as ODBMS. It is A process of identifying and retrieving the data from the database, based on the query provided by user or application.
Retrieves information about a subject.	Determines the keywords in the user query and

Information Retrieval	Data Retrieval
	retrieves the data.
Small errors are likely to go unnoticed.	A single error object means total failure.
Not always well structured and is semantically ambiguous.	Has a well-defined structure and semantics.
Does not provide a solution to the user of the database system.	Provides solutions to the user of the database system.
The results obtained are approximate matches.	The results obtained are exact matches.
Results are ordered by relevance.	Results are unordered by relevance.
It is a probabilistic model.	It is a deterministic model.

The User Task: The information first is supposed to be translated into a query by the user. In the information retrieval system, there is a set of words that convey the semantics of the information that is required whereas, in a data retrieval system, a query expression is used to convey the constraints which are satisfied by the objects.

- **Logical View of the Documents:** A long time ago, documents were represented through a set of index terms or keywords. Nowadays, modern computers represent documents by a full set of words which reduces the set of representative keywords. This can be done by eliminating stopwords i.e. articles and connectives. These operations are text operations. These text operations reduce the complexity of the document representation from full text to set of index terms.

Past, Present, and Future of Information Retrieval

1. Early Developments: As there was an increase in the need for a lot of information, it became necessary to build data structures to get faster access. The index is the data structure for faster retrieval of information. Over centuries manual categorization of hierarchies was done for indexes.

2. Information Retrieval In Libraries: Libraries were the first to adopt IR systems for information retrieval. In first-generation, it consisted, automation of previous technologies, and the search was based on author name and title. In the second generation, it included searching by subject heading, keywords, etc. In the third generation, it consisted of graphical interfaces, electronic forms, hypertext features, etc.

3. The Web and Digital Libraries: It is cheaper than various sources of information, it provides greater access to networks due to digital communication and it gives free access to publish on a larger medium.

Advantages of Information Retrieval

1. Efficient Access: Information retrieval techniques make it possible for users to easily locate and retrieve vast amounts of data or information.

2. Personalization of Results: User profiling and personalization techniques are used in information retrieval models to tailor search results to individual preferences and behaviors.

3. Scalability: Information retrieval models are capable of handling increasing data volumes.

4. Precision: These systems can provide highly accurate and relevant search results, reducing the likelihood of irrelevant information appearing in search results.

Disadvantages of Information Retrieval

1. Information Overload: When a lot of information is available, users often face information overload, making it difficult to find the most useful and relevant material.

2. Lack of Context: Information retrieval systems may fail to understand the context of a user's query, potentially leading to inaccurate results.

3. Privacy and Security Concerns: As information retrieval systems often access sensitive user data, they can raise privacy and security concerns.

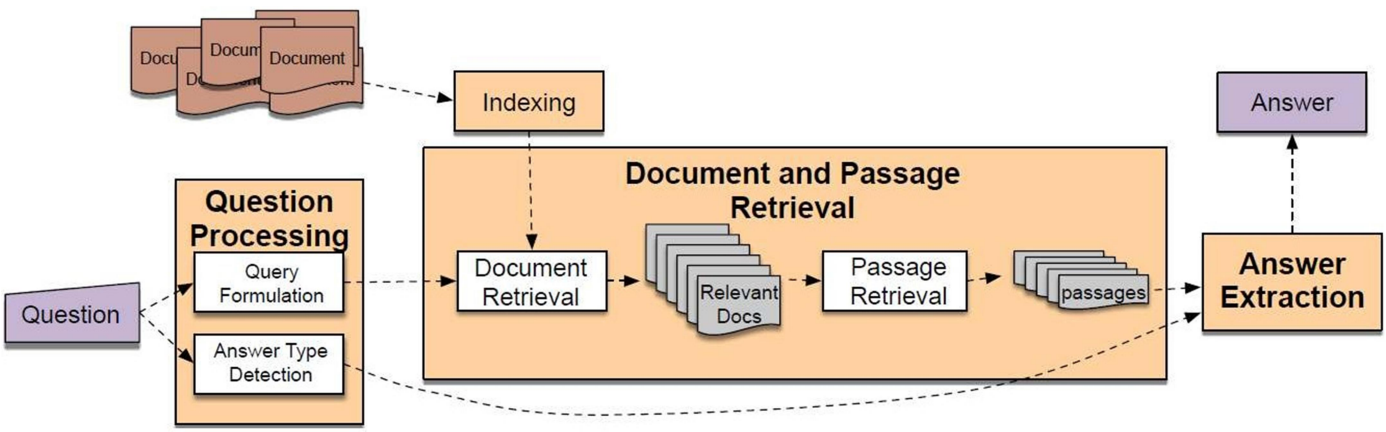
4. Maintenance Challenges: Keeping these systems up-to-date and effective requires ongoing efforts, including regular updates, data cleaning, and algorithm adjustments.

5. Bias and fairness: Ensuring that information retrieval systems do not exhibit biases and provide fair and unbiased results is a crucial challenge, especially in contexts like web search engines and recommendation systems.

IR-based question answering:

What is IR based question answering?

IR-based Factoid Question Answering. The goal of information retrieval based question answering is to answer a user's question by finding short text segments on the web or some other collection of doc- uments.



What is a question-answering System?

Question answering (QA) is a field of natural language processing (NLP) and artificial intelligence (AI) that aims to develop systems that can understand and answer questions posed in natural language.

How does a natural language question-answering system work?

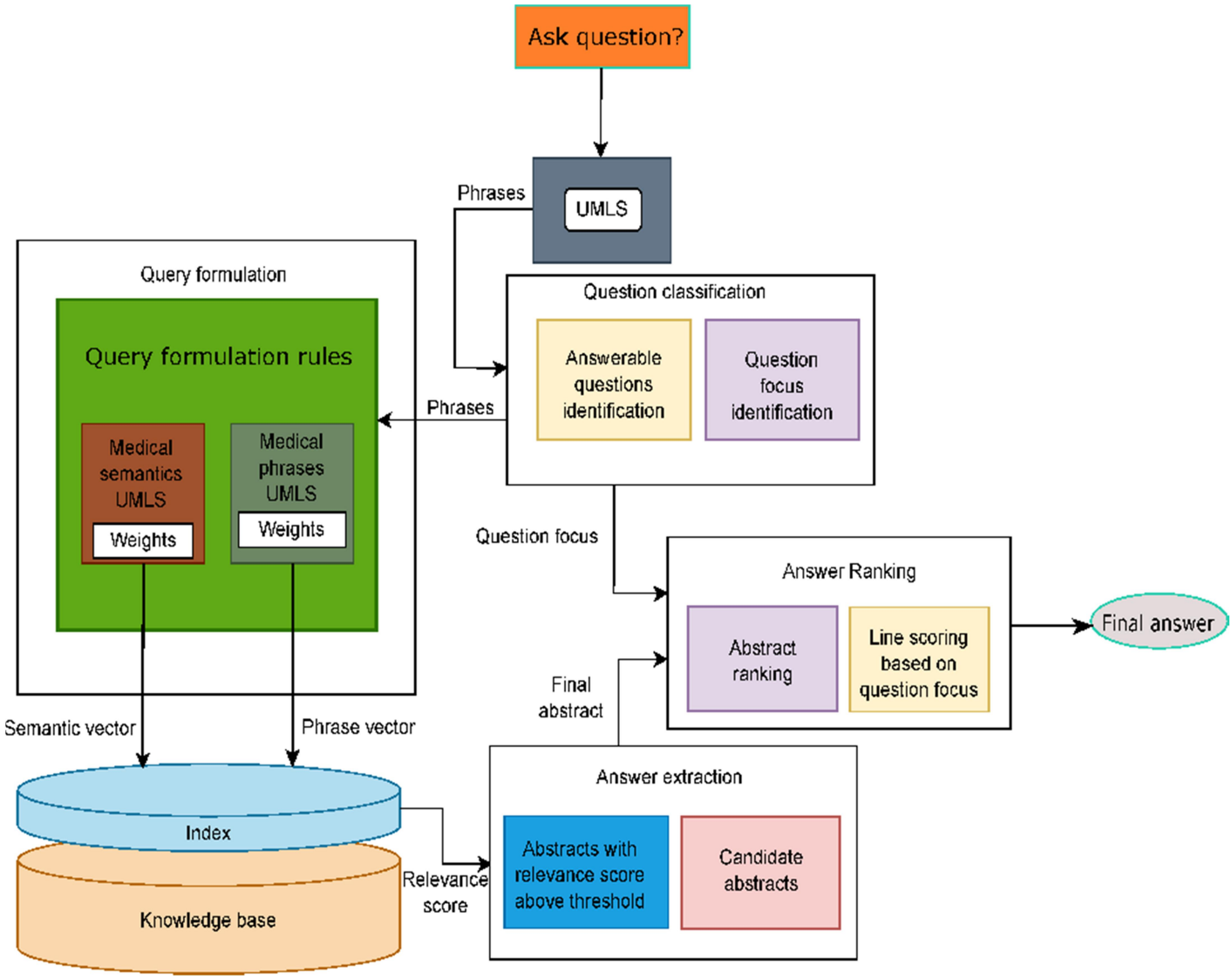
A natural language question-answering (QA) system is a computer program that automatically answers questions using NLP. The basic process of a natural language QA system includes the following steps:

1. **Text pre-processing**: The question is pre-processed to remove irrelevant information and standardise the text's format. This step includes tokenisation, lemmatisation, and stop-word removal, among others.
2. **Question understanding**: The pre-processed question is analysed to extract the relevant entities and concepts and to identify the type of question being asked. This step can be done using natural language processing (NLP) techniques such as named entity recognition, dependency parsing, and part-of-speech tagging.
3. **Information retrieval**: The question is used to search a database or corpus of text to retrieve the most relevant information. This can be done using information retrieval techniques such as keyword search or semantic search.
4. **Answer generation:** The retrieved information is analysed to extract the specific answer to the question. This can be done using various techniques, such as machine learning algorithms, rule-based systems, or a combination.
5. **Ranking**: The extracted answers are ranked based on relevance and confidence score.

Types of question answering system

1. Information retrieval-based QA

Information retrieval-based question answering (QA) is a method of automatically answering questions by searching for relevant documents or passages that contain the answer. This approach uses information retrieval techniques, such as keyword or semantic search, to identify the documents or passages most likely to hold the answer to a given question.



2. Knowledge-based QA

Knowledge-based question answering (QA) automatically answers questions using a knowledge base, such as a database or ontology, to retrieve the relevant information. This strategy's foundation is that searching for a structured knowledge base for a question can yield the answer.

Knowledge-based QA systems are generally more accurate and reliable than other QA approaches based on structured and well-curated knowledge.

3. Generative QA

Generative question answering (QA) automatically answers questions using a generative model, such as a neural network, to generate a natural language answer to a given question.

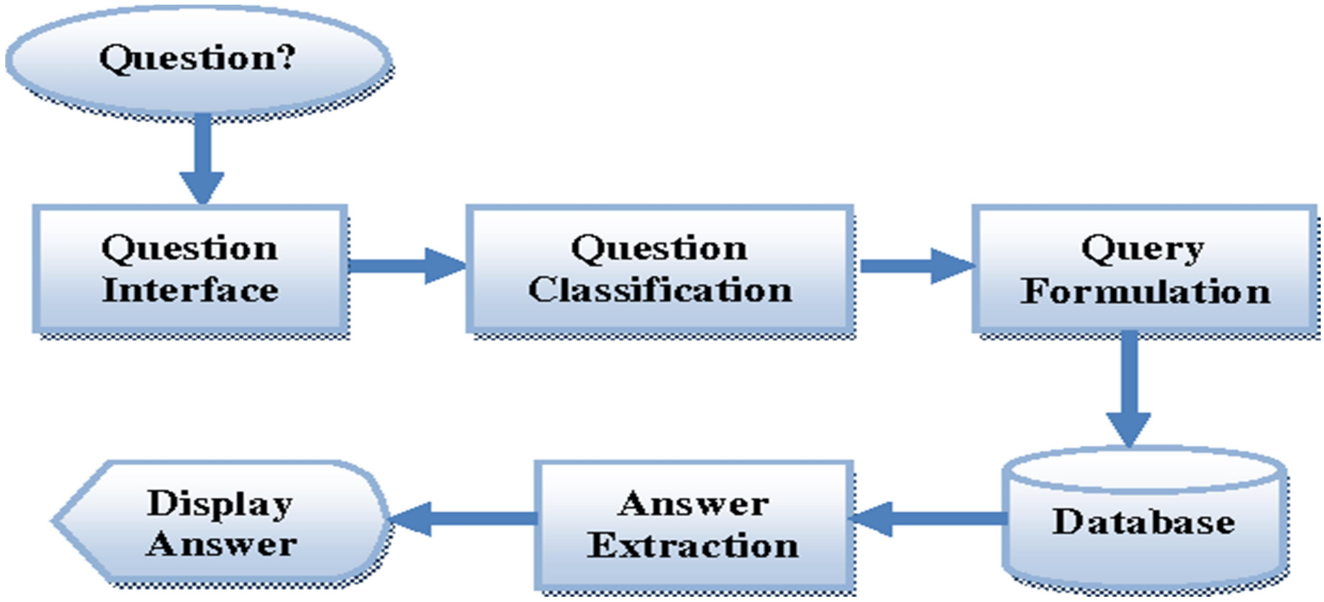
This method is based on the idea that a machine can be taught to understand and create text in natural language to provide a correct answer in terms of grammar and meaning.

4. Hybrid QA

Hybrid question answering (QA) automatically answers questions by combining multiple QA approaches, such as information retrieval-based, knowledge-based, and generative QA. This approach is based on the idea that different QA approaches have their strengths and weaknesses, and by combining them, the overall performance of the QA system can be improved.

5. Rule-based QA

Rule-based question answering (QA) automatically answers questions using a predefined set of rules based on keywords or patterns in the question. This approach is based on the idea that many questions can be answered by matching the question to a set of predefined rules or templates.



Applications:

- Customer Service
- Search engines
- Healthcare
- Education
- Finance
- In e-commerce
- Void assistants
- Chatbots
- Virtual assistants
- Business intelligence

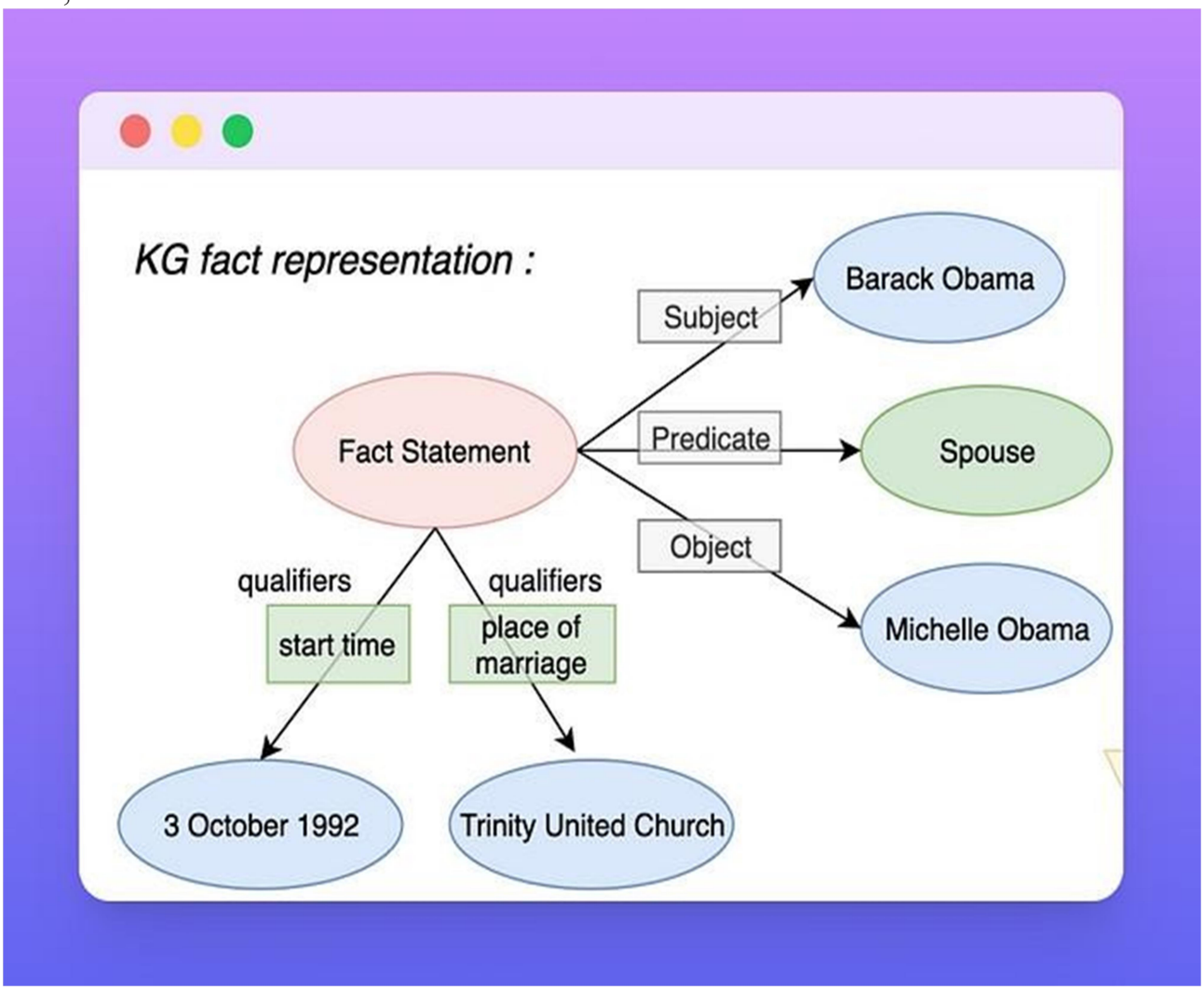
Tools:

TensorFlow
BERT
GPT-3
Hugging Face
SpaCy
NLTK
OpenNLP

knowledge-based question answering:

What is knowledge based question answering?

Knowledge-based question answering (KBQA) is the task of finding answers to questions by pro- cessing a structured knowledge base KB. A KB consists of a set of entities E, a set of relations R, and a set of literals S.



Knowledge-based question answering (KBQA) in text and speech analysis involves using structured knowledge bases or ontologies to answer questions posed in natural language. This approach contrasts with traditional information retrieval systems, which primarily match keywords or phrases to documents. Here's an overview of how KBQA works:

Knowledge Representation: KBQA systems rely on structured knowledge representations such as ontologies, knowledge graphs, or semantic networks. These representations capture entities, their attributes, relationships, and hierarchies in a formalized manner.

Natural Language Understanding: The system analyzes the natural language question to understand its meaning, including entity mentions, relationships, and constraints implied by the question. Techniques such as part-of-speech tagging, named entity recognition, dependency parsing, and semantic role labeling are often used.

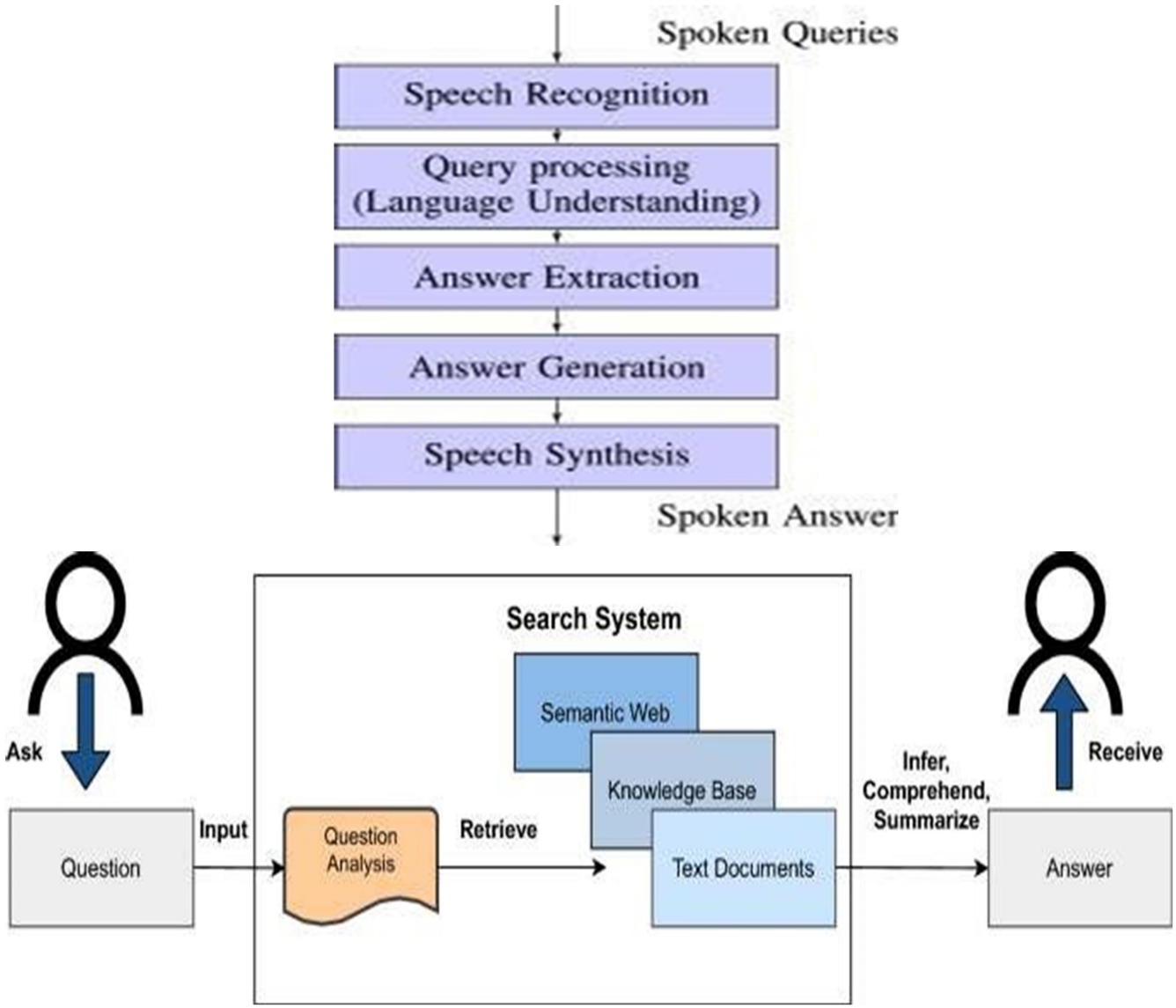
Query Formulation: Based on the understanding of the question, the system formulates a structured query that can be executed against the knowledge base. This query typically involves selecting relevant entities, properties, and relationships to retrieve the desired information.

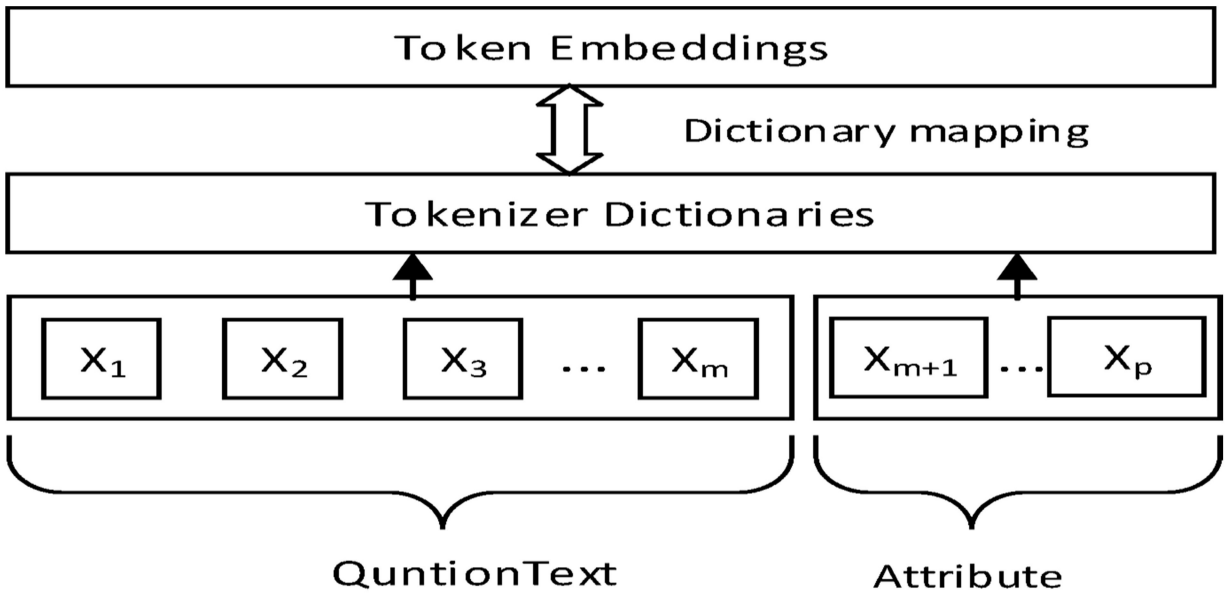
Knowledge Base Querying: The formulated query is executed against the knowledge base to retrieve relevant information. This process may involve querying a structured database, a knowledge graph, or accessing external sources such as linked data on the web.

Answer Generation: Once the relevant information is retrieved from the knowledge base, it is processed to generate a natural language answer that directly addresses the user's question. This may involve aggregating and summarizing information, as well as ensuring that the answer is fluent and grammatically correct.

Response Presentation: Finally, the generated answer is presented to the user through the appropriate interface, whether it's a text-based response in a chatbot interface or synthesized speech in a voice-based interaction.

KBQA systems can vary in complexity and sophistication, ranging from simple rule-based approaches to more advanced systems leveraging machine learning and natural language processing techniques.


Spoken Queries
Speech Recognition
Query processing
(Language Understanding)
Answer Extraction
Answer Generation
Speech Synthesis
Spoken Answer
Search System
Semantic Web
Knowledge Base
Text Documents
Ask
Question
Input
Question
Analysis
Retrieve
Infer,
Comprehend,
Summarize
Receive
Answer

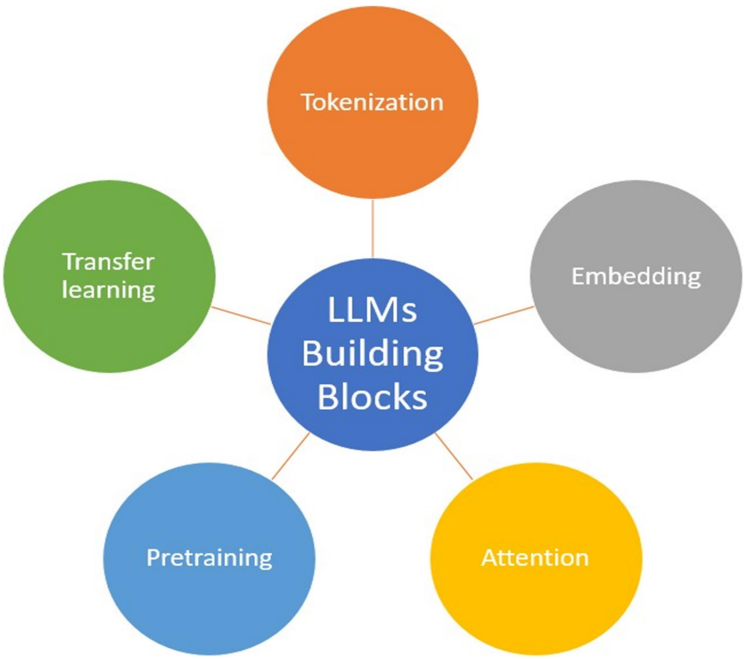

Token Embeddings
Dictionary mapping
Tokenizer Dictionaries
X_1
X_2
X_3
...
X_m
X_{m+1}
...
X_p
QuntionText
Attribute

language models for QA:

These models can predict any word in a sentence or body of text by using every other word in the text. Examining text bidirectionally increases result accuracy. This type is often used in machine learning models and speech generation applications.

What is language model in speech?

Language models rely on acoustic models to convert analog speech waves into digital and discrete phonemes that form the building blocks of words.



Large language Models

Challenges with Language Modeling?

Formal languages (like a programming language) are precisely defined. All the words and their usage is predefined in the system. Anyone who knows a specific programming language can understand what's written without any formal specification.

Machines only understand the language of numbers. For creating language models, it is necessary to convert all the words into a sequence of numbers. For the modellers, this is known as encodings.

How does Language Model Works?

Language Models determine the probability of the next word by analyzing the text in data. These models interpret the data by feeding it through algorithms.

The algorithms are responsible for creating rules for the context in natural language. The models are prepared for the prediction of words by learning the features and characteristics of a language. With this learning, the model prepares itself for understanding phrases and predicting the next words in sentences.

For training a language model, a number of probabilistic approaches are used. These approaches vary on the basis of the purpose for which a language model is created. The amount of text data to be analyzed and the math applied for analysis makes a difference in the approach followed for creating and training a language model.

For example, a language model used for predicting the next word in a search query will be absolutely different from those used in predicting the next word in a long document (such as Google Docs). The approach followed to train the model would be unique in both cases.

Types of Language Models:

There are primarily two types of language models:

1. Statistical Language Models

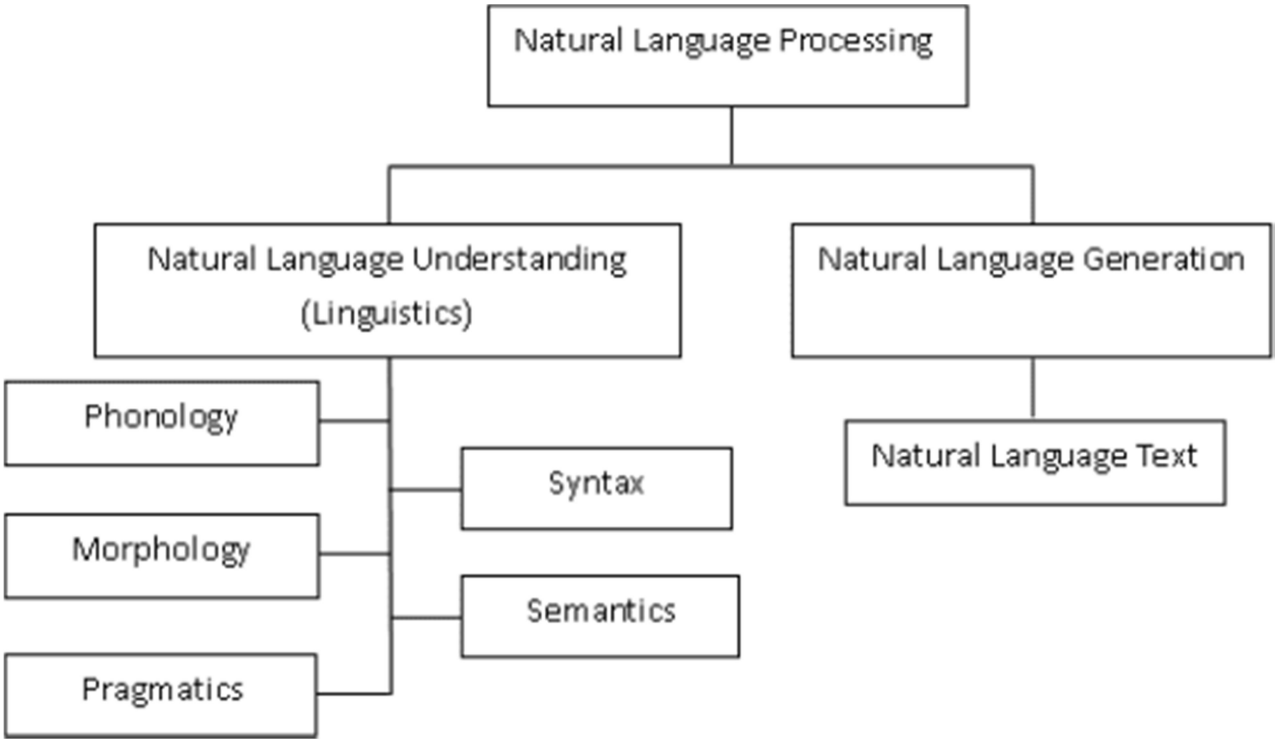
Statistical models include the development of probabilistic models that are able to predict the next word in the sequence, given the words that precede it. A number of statistical language models are in use already.

Let's take a look at some of those popular models:

- N-Gram
- Unigram
- Bidirectional
- Exponential
- Continuous Space

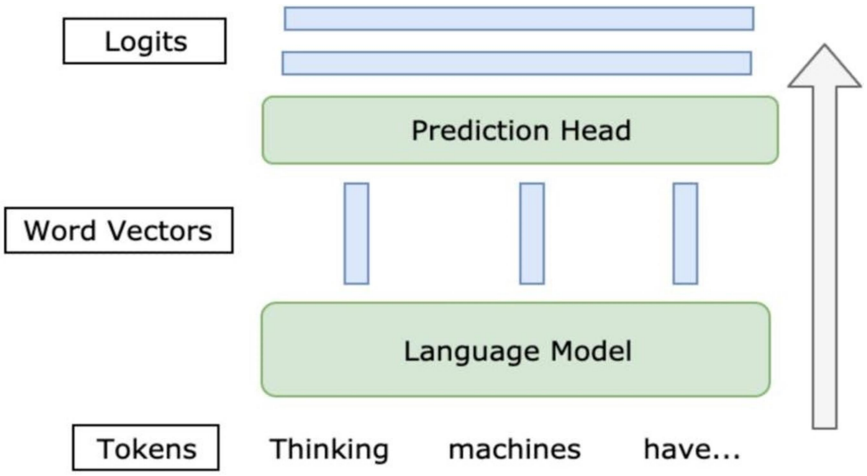
2. Neural Language Models

These language models are based on neural networks and are often considered as an advanced approach to execute NLP tasks. Neural language models overcome the shortcomings of classical models such as n-gram and are used for complex tasks such as speech recognition or machine translation.

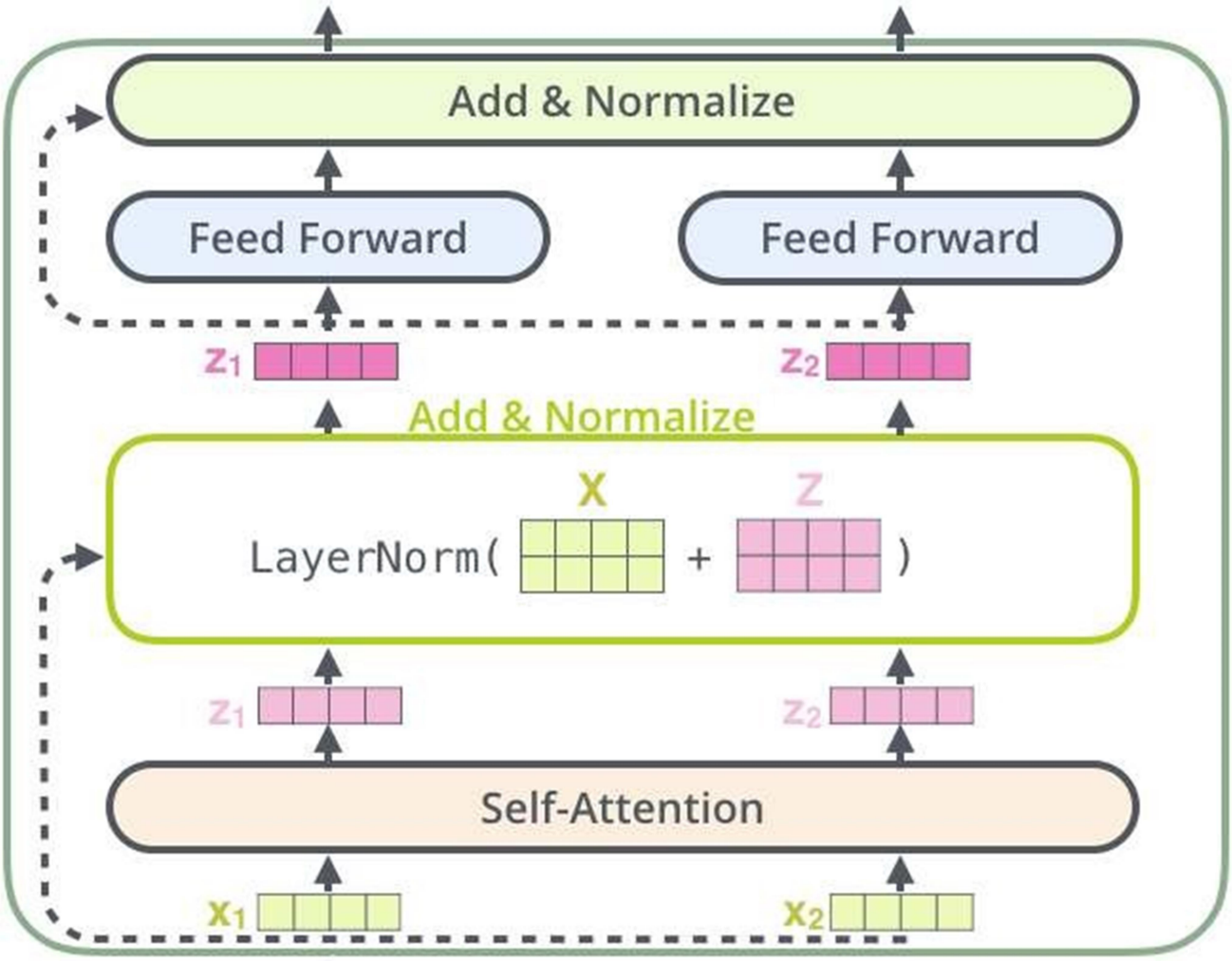


Some Common Examples of Language Models:

1. Speech Recognition
2. **Machine Translation**
3. Sentiment Analysis
4. Text Suggestions
5. Parsing Tools
6. Text Classification
7. Dialog Systems and Creative Writing
8. Text Summarization



Modern Questions Answering System



How to Train-A Question and Answering Machine Learning Models

Common Challenges in NLP Language Models:

1) Long-Term Dependency
2) Low-Resource Languages
3) Sarcasm and Irony
4) Handling Noisy Text
5) Contextual Ambiguity

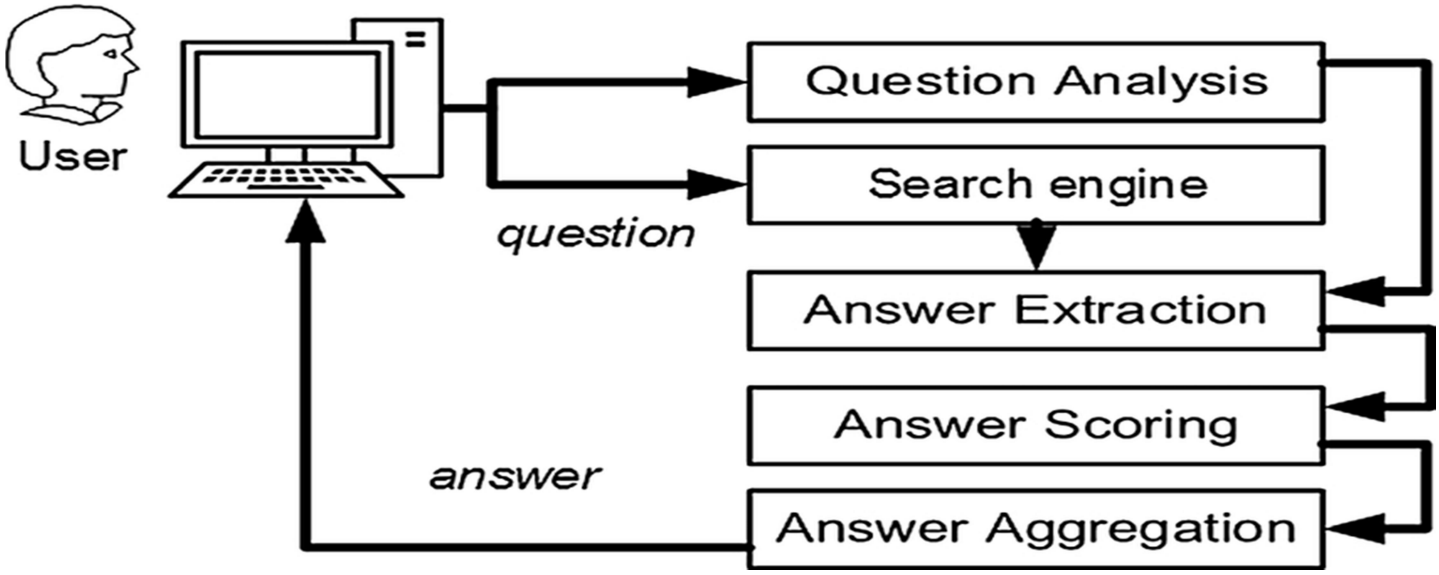
classic QA models:

Classic question-answering (QA) models in text and speech analysis have evolved over the years. Here are some of the classic models:

1. Information Retrieval (IR) Models: These models are based on retrieving relevant documents or passages from a collection in response to a query. Classic IR models include:

Vector Space Model (VSM): Represents documents and queries as vectors in a high-dimensional space and computes similarity scores between them.

Term Frequency-Inverse Document Frequency (TF-IDF): Measures the importance of a term in a document relative to a corpus.



Web based Questions and Answering Models

2.Rule-based QA Systems: These systems rely on handcrafted rules to parse questions and retrieve relevant information from structured or semi-structured data sources. Classic examples include:

ELIZA: A rule-based natural language processing program that simulates a conversation by following patterns and rules.

ALICE: Another early chatbot that uses pattern matching and predefined responses.

3.Statistical QA Models: These models utilize statistical techniques to analyze text and generate answers. Classic examples include:

IBM Watson: Utilizes a combination of statistical techniques, natural language processing, and machine learning to understand and answer questions.

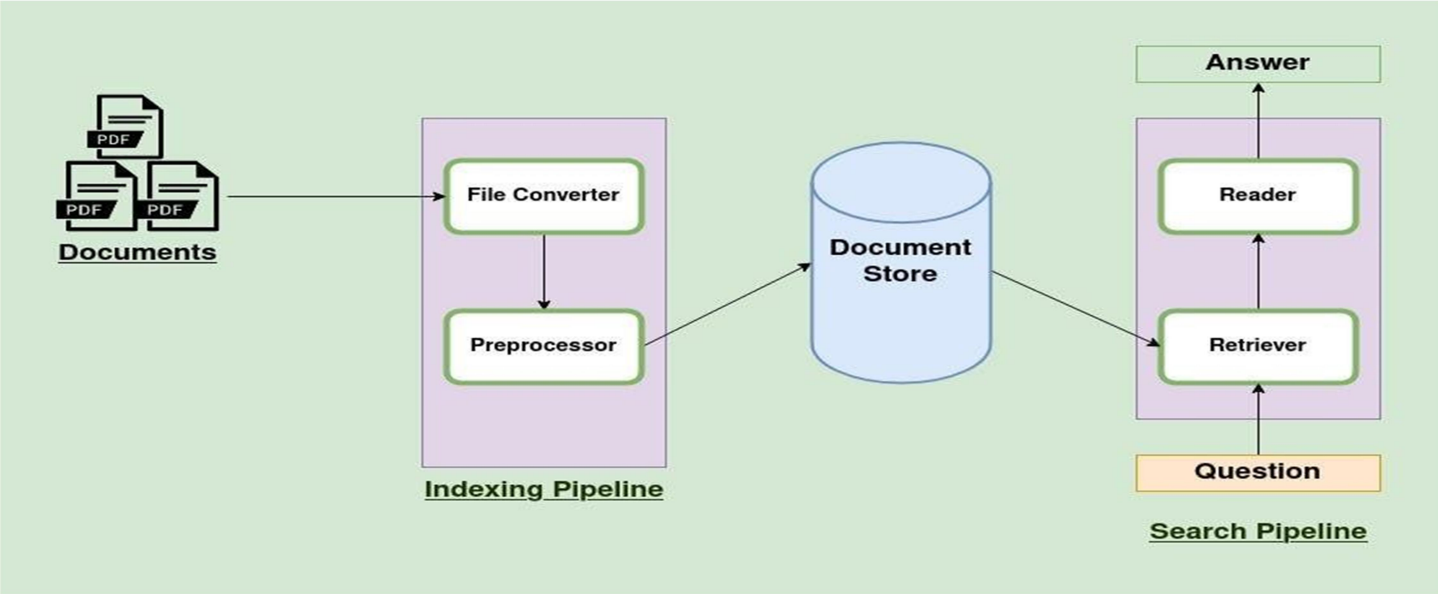
DeepQA: The architecture behind IBM Watson, which combines various algorithms and techniques for question answering.

4.Neural QA Models: These models leverage neural networks to understand and answer questions. Classic examples include:

Memory Networks: Models designed to store and retrieve information from memory, useful for tasks like question answering.

Attention Mechanisms: Mechanisms that allow neural networks to focus on relevant parts of the input, improving performance in QA tasks.

Transformer-based Models: Models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) have shown significant advancements in QA tasks.



Questions and Answering – NLP Projects

5. **Graph-based QA Models:** These models represent text or knowledge as graphs and perform reasoning over them to answer questions. Classic examples include:

Knowledge Graphs: Represent structured knowledge as graphs and perform graph-based reasoning to answer questions.

Graph Neural Networks (GNNs): Neural networks designed to operate on graph-structured data, which can be used for QA tasks involving graph representations.

Each of these classic models has its strengths and weaknesses, and modern QA systems often combine multiple approaches for improved performance.

What are the uses of question answering system?

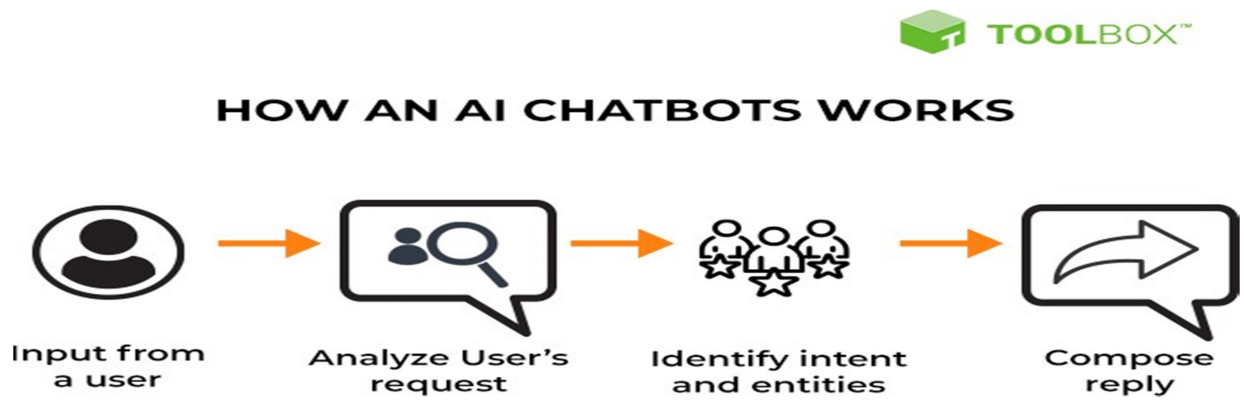
Question answering is commonly used to build conversational client applications, which include social media applications, chat bots, and speech-enabled desktop applications.

What are the 5 applications of NLP?

NLP business applications come in different forms and are so common these days. For example, spell checkers, online search, translators, voice assistants, spam filters, and autocorrect are all NLP applications.

Chatbots:

The role of chatbots in NLP lies in their ability to understand and respond to natural language input from users. This means that rather than relying on specific commands or keywords like traditional computer programs, chatbots can process human-like questions and responses.



What are the main types of chatbots?

Depending on their capabilities, chatbots can be simple, intelligent, and hybrid.

1. **Simple bots** are quite basic tools that rely on natural language processing. They can understand and respond to human queries with certain actions that are based on keywords and phrases. This type of bots has a defined rule-based decision tree (or RBDT), which helps users find needed information. FAQ chatbot is a perfect example of a simple bot.

2. **Intelligent chatbots**, which are also known as virtual assistants or virtual agents, are powered by artificial intelligence and are much more complicated than simple chatbots. They can understand human written and oral language and, which is more important, the context behind it.

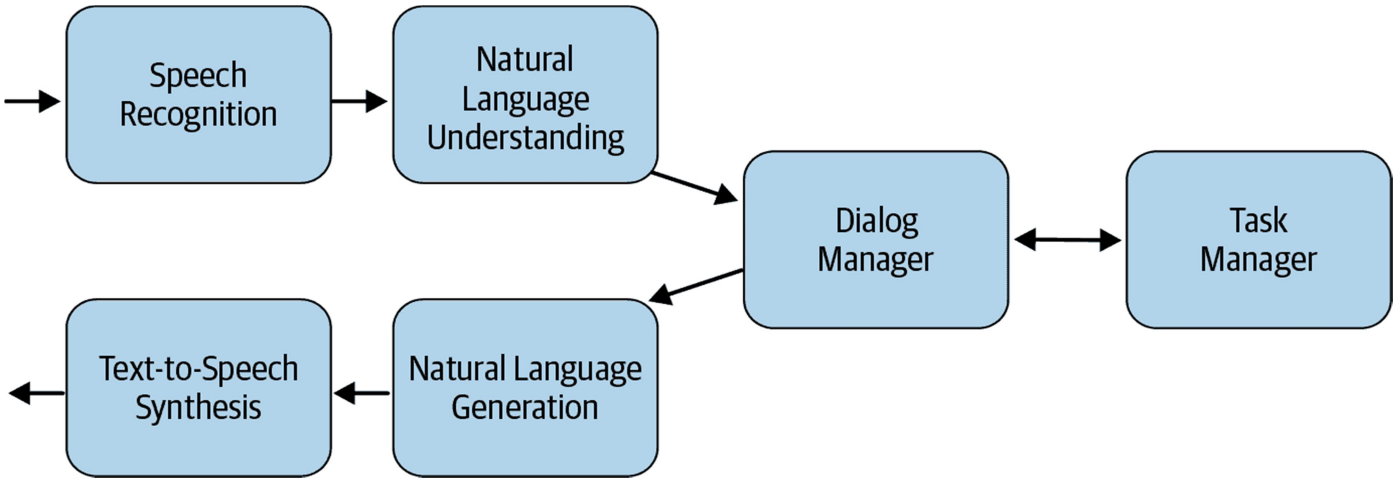
Hybrid chatbots are bots that are partially automated, meaning that they lead conversations until a human interaction is required. They might have the same functionality as simple bots, but a user can opt for a person when needed.

Chatbots can be powerful tools in text and speech analysis due to their ability to process large amounts of data quickly and efficiently. Here's how they're used:

1. **Text Analysis**: Chatbots can analyze text data to extract valuable insights such as sentiment analysis, topic modeling, keyword extraction, and named entity recognition. They can

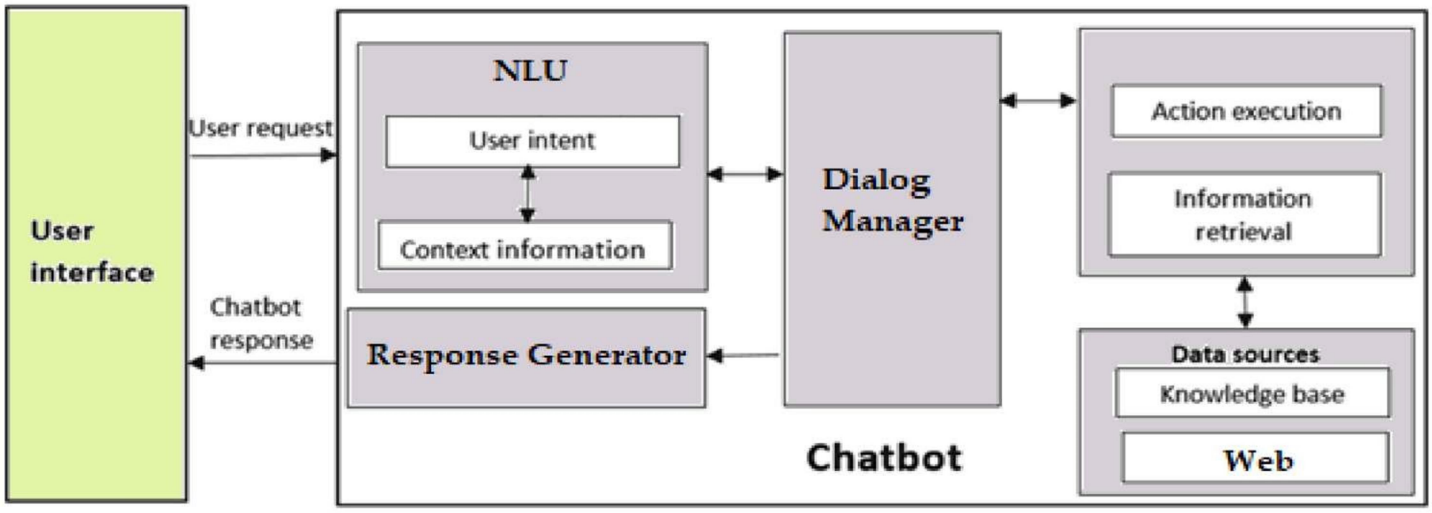
understand the context of the conversation and provide relevant responses or take appropriate actions based on the analysis.

2.Speech Recognition: With advancements in natural language processing (NLP) and speech recognition technology, chatbots can transcribe spoken language into text. This text data can then be further analyzed using text analysis techniques mentioned above.



3.Sentiment Analysis: Chatbots can analyze the sentiment expressed in text or speech, helping businesses gauge customer satisfaction, detect issues, or monitor public opinion about their products or services.

4.Customer Support: Chatbots are commonly used in customer support to analyze customer queries and provide appropriate responses. They can understand the intent behind the customer's message and either provide a solution or escalate the query to a human agent if necessary.



5. **Market Research:** Chatbots can be deployed to gather and analyze textual data from social media, forums, or surveys to understand consumer preferences, trends, and feedback on products or services.

6. **Language Translation:** Chatbots equipped with language translation capabilities can analyze and translate text or speech from one language to another, facilitating communication across linguistic barriers.

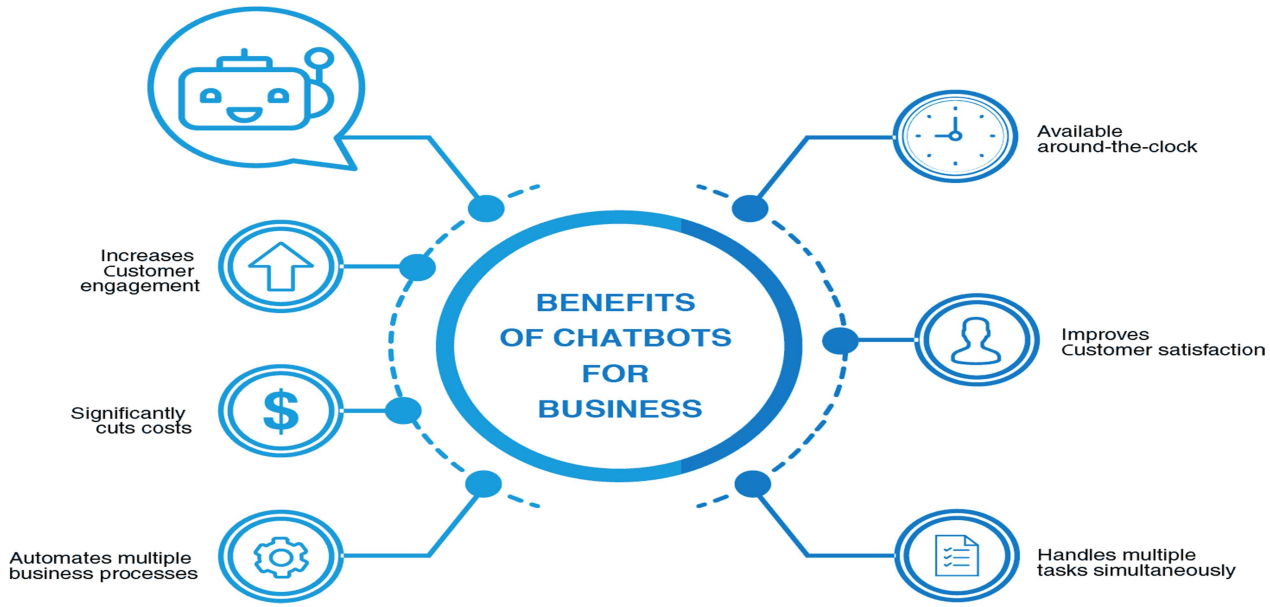
7. **Personalization:** By analyzing user interactions and preferences, chatbots can personalize responses and recommendations, improving user experience and engagement.

Chatbots Terminologies:

- Quick reply
- Hybrid Chat
- Intent
- Sentiment analysis
- Compulsory input
- Optional input
- Decision trees

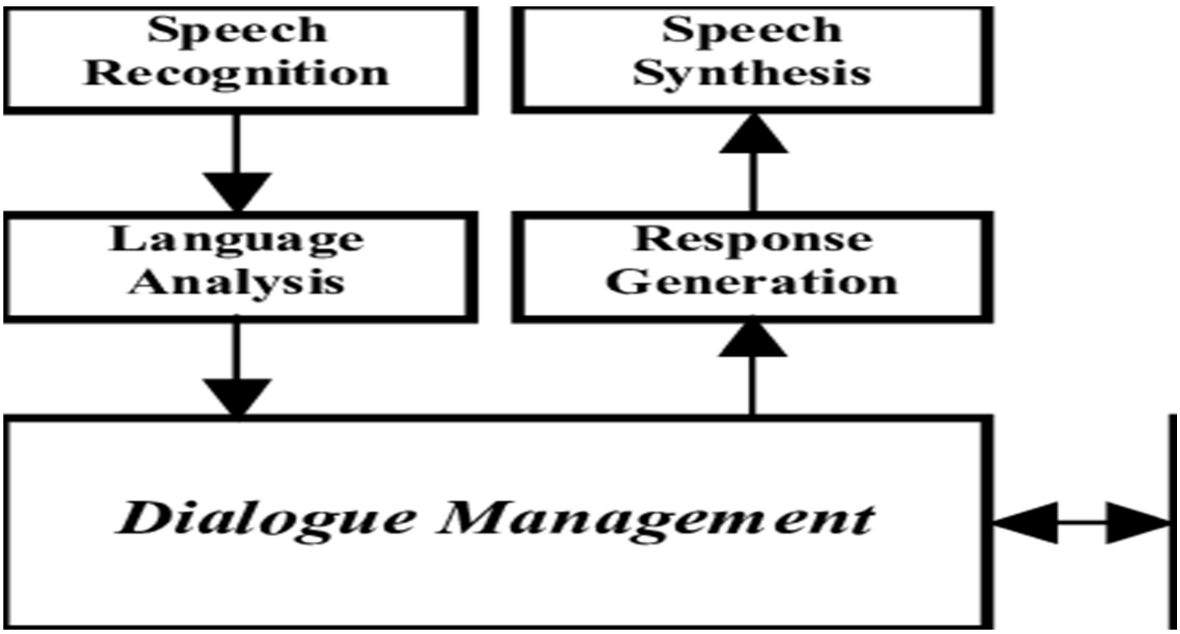
Benefits of commercial chatbots:

- Help customers find what they need much faster
- Can easily substitute a seller
- Always available at your customers' fingertips



Design of dialogue systems:

A Dialogue System is a system which interacts with human in natural language. At present many universities are developing the dialogue system in their regional language.Dialogue systems, also known as conversational agents or chatbots, are designed to interact with users in a natural and human-like manner. They can be implemented in various forms, including text-based interfaces like messaging apps or speech-based interfaces like virtual assistants.



Architecture of spoken dialogue systems

What is speech dialog system?

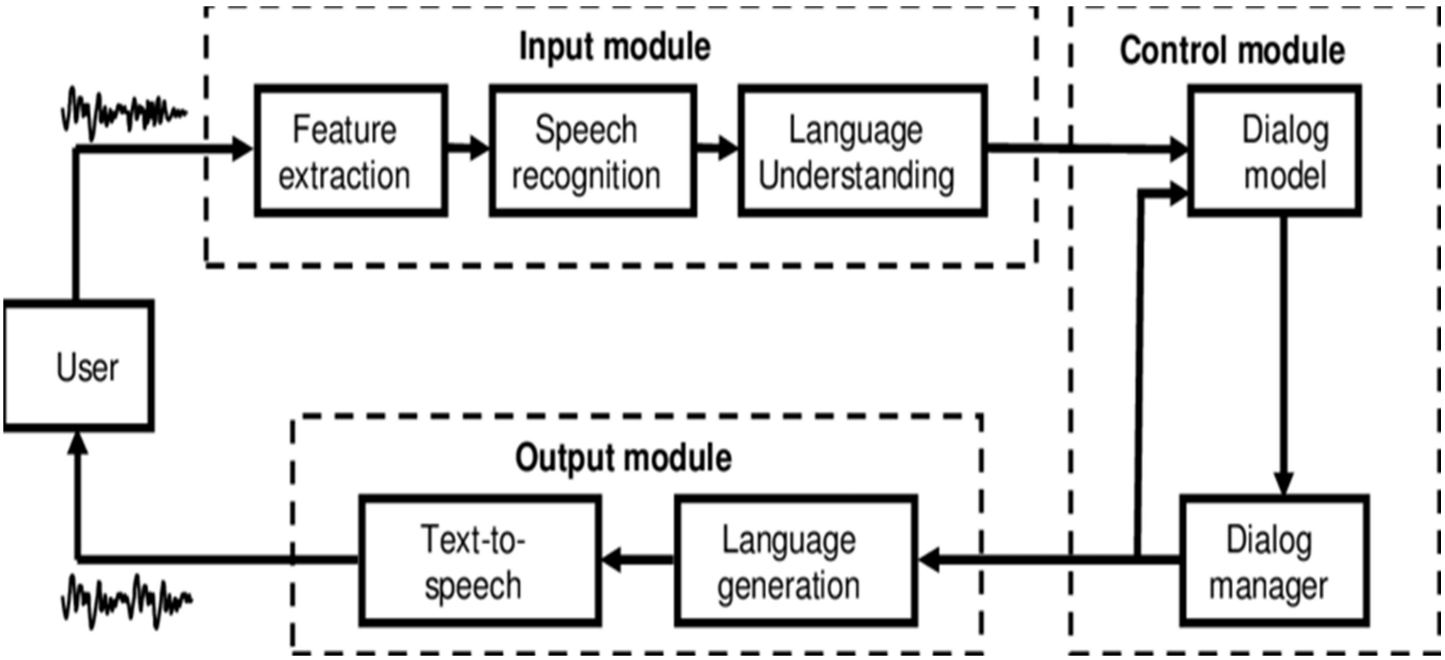
A spoken dialog system (SDS) is a computer system able to converse with a human with voice. It has two essential components that do not exist in a written text dialog system: a speech recognizer and a text-to-speech module (written text dialog systems usually use other input systems provided by an OS).

What is an example of a dialogue system?

Examples of dialogue systems in action include chatbots, food ordering apps, website AI assistants, automated customer support service, self-checkout systems, etc.

What are types of dialogue systems?

- Rule-based systems,
- Statistical systems,
- Neural networks



Components of Dialogue System:

A Dialogue system has mainly seven components. These components are following:

- Input Decoder
- Natural Language Understanding
- Dialogue Manager
- Domain Specific Component
- Response Generator
- Output Renderer

several key components:

1.Natural Language Understanding (NLU):

NLU is crucial for dialogue systems to comprehend user inputs accurately. It involves tasks such as intent classification, entity recognition, and sentiment analysis.

In text analysis, techniques like natural language processing (NLP) and machine learning models are used to parse and understand the meaning of user messages.

In speech analysis, automatic speech recognition (ASR) systems convert spoken language into text, which is then processed using NLU techniques.

2. Dialogue Management:

Dialogue management is responsible for determining the system's response based on the user's input and the current context of the conversation.

In text-based systems, dialogue management often involves maintaining a conversation state, tracking the dialogue history, and selecting appropriate responses using rule-based systems or machine learning algorithms.

In speech-based systems, dialogue management may also incorporate speech recognition results and handle interruptions or errors in speech input.

3. Response Generation:

Response generation involves creating human-like responses to user inputs. This can be achieved using templates, rule-based systems, or machine learning models like neural networks.

In text-based systems, response generation may involve generating text using language generation techniques such as neural language models (e.g., GPT).

In speech-based systems, text-to-speech (TTS) synthesis is used to convert textual responses into spoken language.

4. User Experience (UX) Design:

UX design focuses on creating a smooth and intuitive interaction between users and dialogue systems.

In text-based systems, UX design includes considerations such as message formatting, response timing, and error handling.

In speech-based systems, UX design involves designing voice prompts, handling interruptions gracefully, and providing feedback through speech.

5. Feedback and Adaptation:

Dialogue systems should be able to learn and adapt based on user feedback to improve their performance over time.

Techniques such as reinforcement learning can be used to optimize dialogue policies based on user interactions.

In text-based systems, sentiment analysis can be used to gauge user satisfaction and adjust system behavior accordingly.

In speech-based systems, user feedback can be collected through voice commands or post-interaction surveys.

6. Multi-Modality:

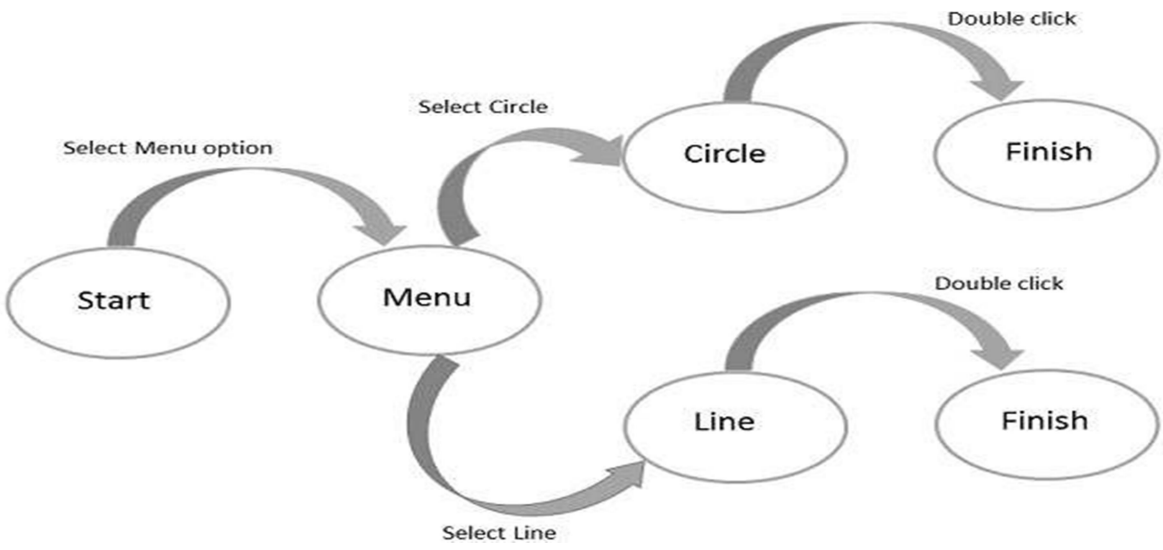
Some dialogue systems incorporate both text and speech modalities to provide a more versatile user experience.

Multi-modal systems must seamlessly integrate text and speech processing components while maintaining consistency across modalities.

7. Privacy and Security:

Dialogue systems often handle sensitive information, so ensuring privacy and security is paramount.

Techniques such as end-to-end encryption and secure data handling practices should be implemented to protect user data.



Dialog Design

Classification of Dialogue System:

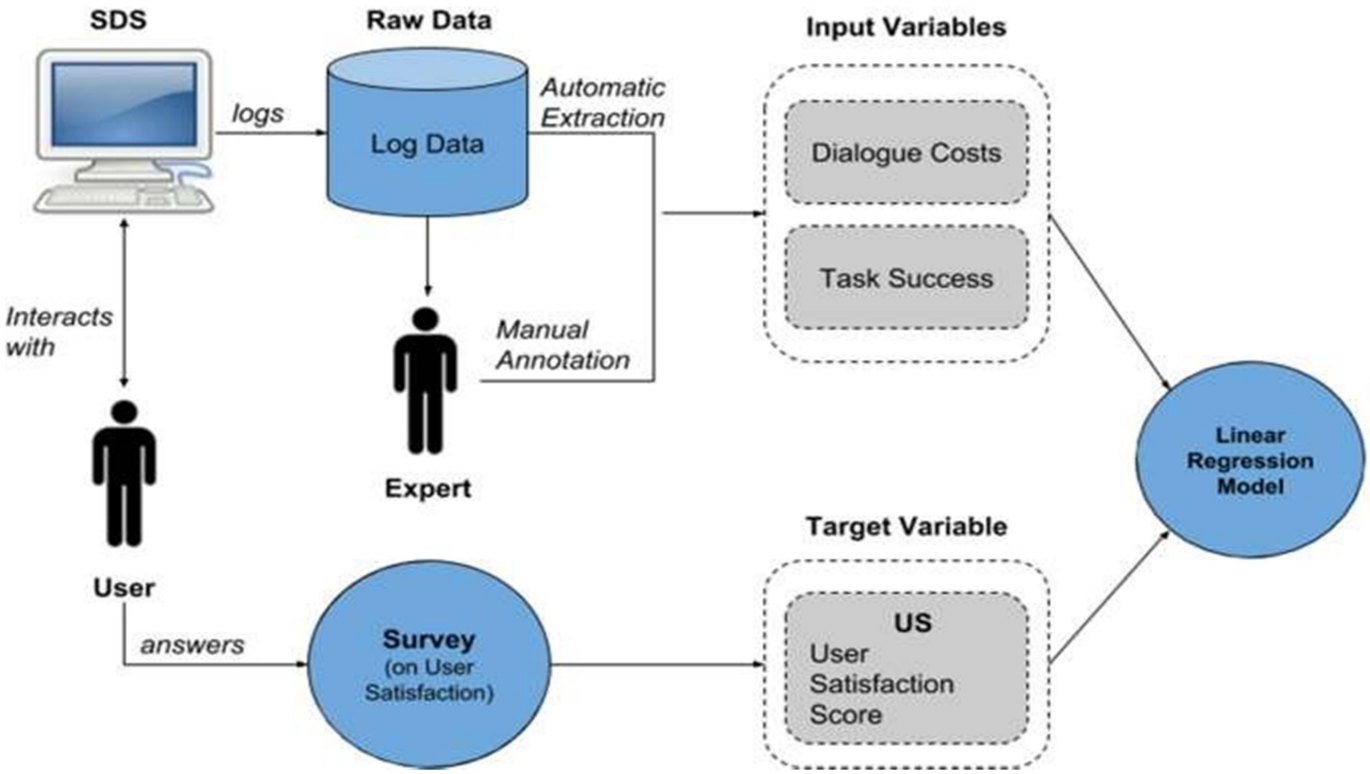
- Finite State (or graph) based systems
- Frame based systems
- Agent based systems

Evaluating dialogue systems:

What is the dialogue system architecture?

While the architecture of Dialogue Systems can vary, they typically follow the same sequence of phases: Input Recognition, Natural Language Understanding, Dialogue Management, Response Generation, and Output Rendering.

Evaluating dialogue systems, whether they operate through text or speech analysis, involves assessing various aspects of their performance, including accuracy, effectiveness, user satisfaction, and scalability. Here are some common evaluation metrics and methodologies for both text and speech-based dialogue systems:



Survey on evaluation methods for Dialog Systems

Text-Based Dialogue Systems:

Accuracy Metrics:

Intent Classification Accuracy: Measures how accurately the system classifies user intents based on their input.

Entity Recognition F1 Score: Evaluates the system's ability to correctly identify and extract entities from user messages.

Response Generation Quality: Assess the coherence, relevance, and grammatical correctness of the generated responses using metrics like BLEU, ROUGE, or human judgment.

Effectiveness Metrics:

Task Completion Rate: Determines the percentage of user queries or tasks successfully completed by the system without errors.

Response Latency: Measures the time taken by the system to respond to user inputs, aiming for low latency to improve user experience.

User Satisfaction:

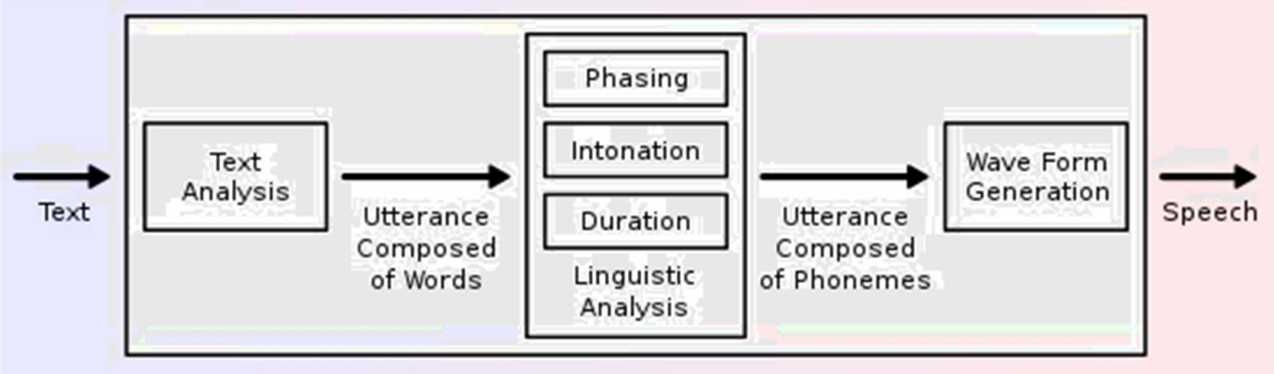
User Surveys: Collect feedback from users through surveys to assess their satisfaction with the dialogue system's performance, usability, and helpfulness.

User Ratings: Users can rate their interactions with the system on a scale, providing quantitative feedback on their satisfaction levels.

Error Analysis: Analyze common errors made by the system, such as misclassification of intents, incorrect entity recognition, or nonsensical responses, to identify areas for improvement.

Robustness and Adaptability:

Evaluate how well the system handles variations in user input, including typos, slang, or ambiguous language.



Speech Synthesis

<u>Speech-Based Dialogue Systems:</u>

Speech Recognition Accuracy:

Word Error Rate (WER): Measures the accuracy of the system's speech recognition component by comparing the transcribed text with the ground truth.

Phoneme Error Rate (PER): Evaluates the accuracy of phoneme-level transcription in speech recognition.

Naturalness of Speech Synthesis:

Mean Opinion Score (MOS): Collect subjective ratings from human listeners on the naturalness, intelligibility, and overall quality of synthesized speech.

Task Completion Rate:

Similar to text-based systems, assess the percentage of user queries or tasks successfully completed by the system through spoken interactions.

Speech Interaction Latency:

Measure the time taken by the system to process spoken input, recognize intents, generate responses, and synthesize speech, aiming for minimal latency.

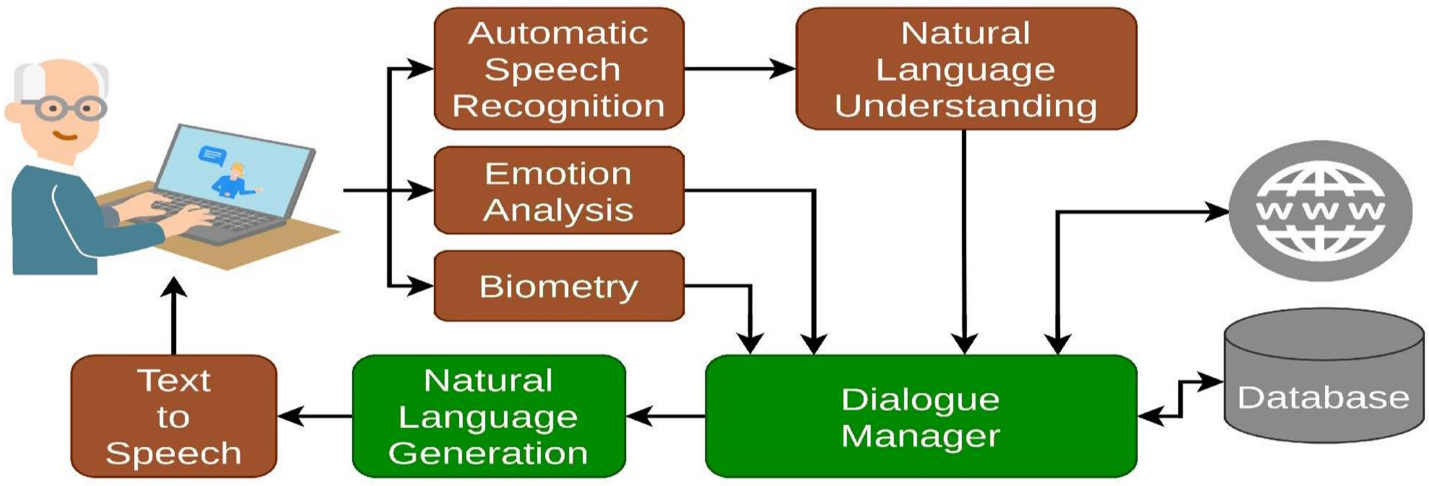
Noise Robustness:

Evaluate the system's performance in noisy environments by introducing background noise and assessing its impact on speech recognition accuracy and speech synthesis intelligibility.

User Experience in Speech-Based Interactions: Conduct user studies to gather feedback on the ease of use, naturalness, and effectiveness of speech-based interactions with the system.

Multimodal Integration:

Assess the effectiveness of integrating speech recognition and synthesis with other modalities, such as text-based input and output, to provide a seamless user experience across multiple channels.



Dialog Management and Language Generation

CCS369- TEXT AND SPEECH ANALYSIS

UNIT IV

OVERVIEW

At a high level, voice applications have three main components: speech recognition, speech profiling, and speech synthesis.

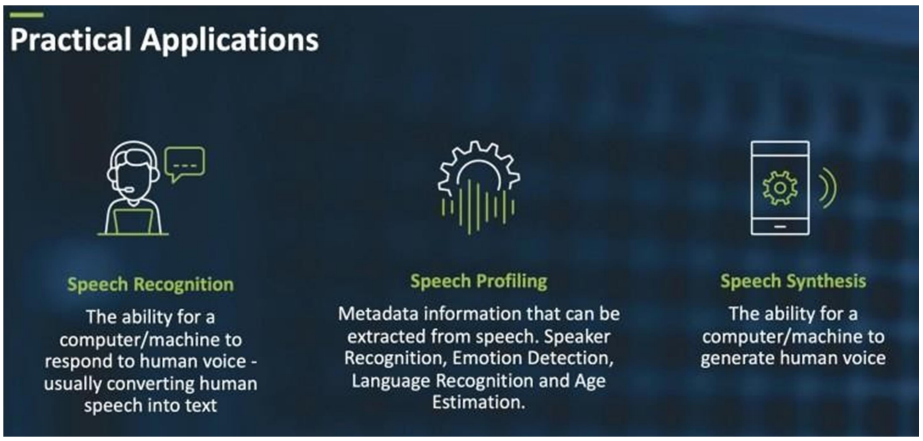


Figure 1: There are three main components of voice applications.

- Speech recognition is the translation of spoken language into text. It is also called automatic speech recognition, computer speech recognition, and speech to text. A major application of ASR is transcribing conversations. Other examples include "Hey, Siri" commands, such as "call home," and voice interfaces requiring simple data entry (e.g., entering credit card numbers or call routing—"press or say '1' for customer service").
- Speech profiling is the process of audio mining information from speech beyond recognition, including age, gender, emotion, the language spoken, speaker verification, etc. Applications include biometrics, sentiment analysis, and metadata extraction to improve customer intelligence initiatives.
- Speech synthesis is the artificial creation of human speech. In this post we'll occasionally use the term "speech synthesis" to refer to technologies that cut across TTS and speech synthesis. The more familiar term is "text-to-speech" but we're opting for "speech synthesis" because we expect input sources in the future to include a range of formats including text and audio. An example of a system that can take audio input is Tencent's PitchNet: a model that takes audio of one singing voice and converts it into audio of another voice singing the same content.

Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a speech synthesizer, and can be implemented in software or hardware products. A text-to-speech (TTS) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech. The reverse process is speech recognition. A TTS engine converts written text to a phonemic representation, then converts the phonemic representation to waveforms that can be output as sound.

Text-To-Speech Synthesis *is a machine learning task that involves converting written text into spoken words. The goal is to generate synthetic speech that sounds natural and resembles human speech as closely as possible.*

Use Cases

Text-to-Speech (TTS) models can be used in any speech-enabled application that requires converting text to speech imitating human voice.

Voice Assistants

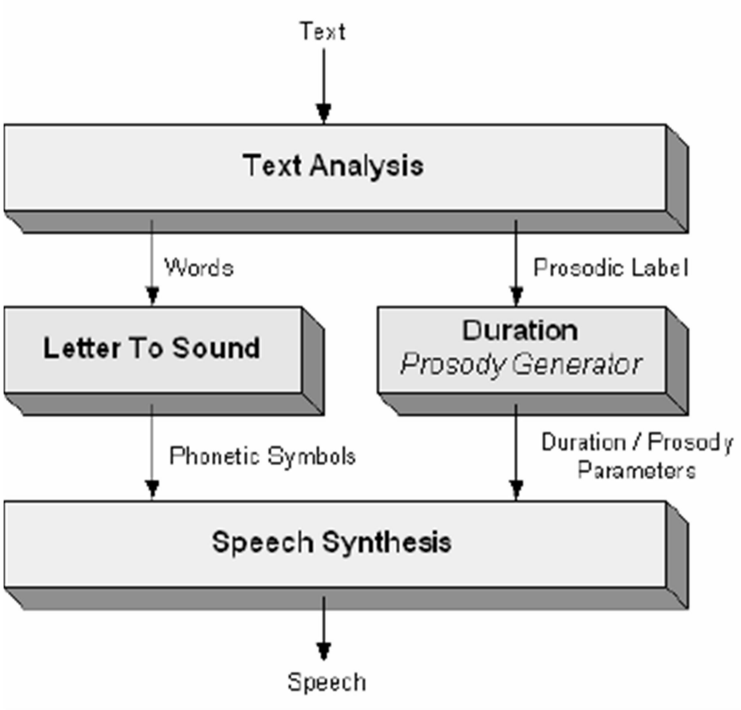
TTS models are used to create voice assistants on smart devices. These models are a better alternative compared to concatenative methods where the assistant is built by recording sounds and mapping them, since the outputs in TTS models contain elements in natural speech such as emphasis.

Announcement Systems

TTS models are widely used in airport and public transportation announcement systems to convert the announcement of a given text into speech

Some common applications:

- Conversational interactive voice response (IVR) systems, as in customer service call centers
- Voice commerce applications, such as shopping on an Amazon Alexa device
- Voice guidance and navigation tools, like GPS mapping apps
- Smart home devices and other voice-enabled Internet of Things (IoT) tools
- Independent virtual assistants like Apple's Siri, but for your own brand
- Experiential marketing and advertising solutions, like interactive voice ads on music streaming services or branded smart speaker apps
- Video game development, with dynamic runtime TTS for accessibility features, scene prototyping, and AI non-player characters
- Company training and marketing videos that allow creators to change voice-overs without tracking down original voice talent for ongoing recording sessions



TTS ARCHITECURE

A typical TTS system consists of two basic processing modules: **text analysis module and a speech synthesis module.**

Text analysis

The first module of a TTS pipeline, the text analysis module, is responsible for converting the incoming text into a linguistic representation that encodes the information of how the input text should be spoken.

As the first step, the module must process the text into a standardized format by taking care of any special characters or other inconsistencies. Then the text must be structurally analyzed to identify syntactical components of the sentences. Any inconsistencies between the written and spoken language (e.g., abbreviations and acronyms, proper names, numbers) must be detected and converted into a correct format (e.g., to map the string "100 sq ft" into "*one hundred square feet*", not "*one-zero-zero sq ft*") .

The second step consists of inferring the correct pronunciation of the words, also known as **grapheme-to-phoneme conversion.** Since the relationship between written and spoken language is not always straightforward, the text analysis module must convert strings of input letters into strings of phonemes based on the pronunciation conventions of the given language. This involves resolving many ambiguities prevalent in languages, such as how to deal with homographs, that is, words with the same written form but different pronunciations. Foreign language words and loan words have to be handled as well.

As the final stage, suprasegmental characteristics of the speech-to-be-produced must be introduced to the linguistic representation. Durations of the phonemes and intermediate pauses must be defined, even though the information does not exist in the text. In order to make the speech natural sounding and intelligible, the process also includes introduction of any potential rhythmic structure, stress patterns, and intonational cues to the linguistic representation. For this purpose, the text analysis module must interpret syntactic properties of the input text. For instance, the system must be able to differentiate different sentence types such as questions from statements and to infer which word should receive focus in the given sentential context.

Speech Synthesis

Speech synthesis systems aim at producing intelligible speech signals from some type of input language representation. Synthesis systems are also often referred to as text-to-speech (TTS) systems, as written text is a natural way to instruct what type of utterances should be produced. Speech Synthesis software are transforming the work culture of different industry sectors. A speech synthesizer is a computerized voice that turns a written text into a speech. It is an output where a computer reads out the word loud in a simulated voice; it is often called text-to-speech. It is not only to have machines talk simply but also to make a sound like humans of different ages and gender. With the rise of usage of digital services and the increase in dependency on voice recognition, the text-to-speech engine is gaining popularity.

There are 3 stages in which speech synthesis works; text to words, words to phonemes, and phonemes to sound.

Text to words –

The initial stage of speech synthesis, is generally called pre-processing or normalization, it is everything about reducing ambiguity: it's about narrowing down the many different ways a person could read a piece of text into the one that's the most appropriate. In Pre-processing it's about going through the text and then cleaning it up so the computer makes fewer mistakes when it reads the words aloud. Elements like numbers, dates, times, abbreviations, acronyms, and special characters need to be turned into words. This is however harder than it sounds. For example; the number 1953 might refer to several items, a year or a time, or a padlock combination; each of these is read out will sound slightly differently. While humans can figure out the pronunciation based on the way the text is written, computers generally don't have that ability to do that.

This is the reason they use statistical probability techniques or neural networks to arrive at the most likely pronunciation. If there were a decimal point before the numbers (".953"), then it would be read differently as "nine fifty-three." Pre-processing also handles homographs, these are the words pronounced in different ways but the meaning is different for each word. The word "sell" can be pronounced as "cell", so a sentence such as "I sell the flower" is problematic for a speech synthesizer. But if it can understand that the preceding text entirely has a different meaning, by recognizing the spelling ("I have a cell phone"), then it can make a reasonable guess that "I sell the pen" is likely correct.

Words to phonemes –

Once they figure out the words that need to be spoken, next the speech synthesizer has to generate the speech sounds that make up these words. Every computer needs is a huge alphabetical list of words and details of how to pronounce each word. For each word, they would need a list of the phonemes that make up its sound.

Theoretically, if a computer has a dictionary of words and phonemes, then all it needs to do is to read a word and look it up in the list, and then read out the corresponding phonemes. But practically, it's quite harder than it sounds.

The alternative approach involves breaking down the written words into their graphemes (written component units, typically made from the individual letters or syllables that make up a word) and then generate phonemes that correspond to them using a set of simple rules. The benefit of doing this is that the computer can make a reasonable attempt at reading any word.

Phonemes to sound –

At this point, the computer has converted the text into a list of phonemes. But how to find basic phonemes that the computer reads aloud when it's turning the text into speech. There are three different approaches to this.

- First to use recordings of humans saying the phonemes
- The second is for the computer to generate the phonemes itself by generating basic sound frequencies
- The last approach is to imitate the technique of the human voice.

Letter to Sound (LTS): Typically, there are **two possible solutions** to build a LTS module:

1) manually collect rules, with sufficient experience and language knowledge. This method is commonly referred to **rule-based method;**

2) acquire rules by machine learning, which is called **training-based method** generally.

Rule-based method of LTS Module

For a given language, if there exists a systematic relationship between a word format and its pronunciation, rule-based letter-to-sound can be efficient. During our research, we found Spanish, Italian, Portuguese and German are such kind of languages. The workflow of rule-based letter-to-sound is as follows. Firstly, we invite experienced linguists to create a set of pronunciation rules for a language. The programmer translates the rules that are expressed by natural language into a set of machine-readable rules. At the same time, we collect a number of pronunciation items for testing the rules. Secondly, the predicted items are compared with standard items to see what is the precision of the current rule set. If the precision is not high enough, we may modify the rule set and the codes. This interactive process is repeated until the precision reaches a predefined level within the acceptable code size.

Training-based method of LTS Module

The training-based method shows its advantages where there is no language knowledge available, or the work to write the pronunciation rule set systematically is too difficult. Many comparative experiments show that it can achieve higher accuracy than rule-based approach. The typical processes of training method include: *letter-to-phoneme alignment (or grapheme-to-phoneme alignment), model training, and testing.* Of course, for embedded applications, we always need to achieve tradeoff between model size and accuracy.

Letter-to-phoneme alignment

The letter-phoneme alignment task as the problem of inducing links between units that are related by pronunciation. Each link is an in- stance of a specific mapping between letters and phonemes.

TEXT NORMALIZATION

As part of a text-to-speech (TTS) system, the text normalization component is typically one of the first steps in the pipeline, converting raw text into a sequence of words, which can then be passed to later components of the system, including word pronunciation, prosody prediction, and ultimately waveform generation.

Text normalization converts text from written form into its verbalized form, and it is an essential preprocessing step before text-to-speech (TTS). It ensures that TTS can handle all input texts without skipping unknown symbols.

For example, "$123" is converted to "one hundred and twenty-three dollars."

Inverse text normalization is a part of the automatic speech recognition (ASR) post-processing pipeline. It converts ASR model output into its written form to improve text readability.

For example, the ITN module replaces "one hundred and twenty-three dollars" transcribed by an ASR model with "$123."

Inverse text normalization not only improves readability but also boosts the performance of downstream tasks such as neural machine translation or named entity recognition, as such tasks use written text during training.

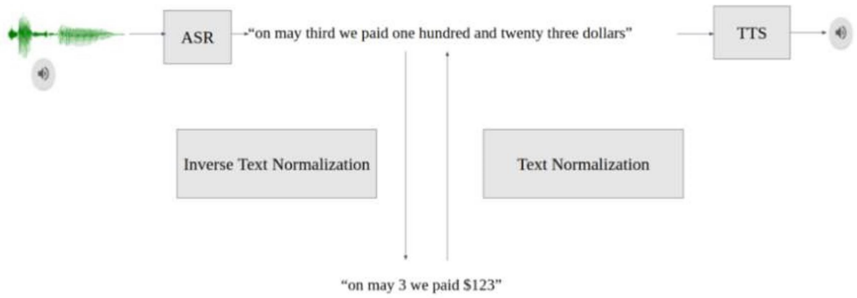


Figure 2: Text Normalization and Inverse Text Normalization

Text normalization and Inverse text normalization tasks face several challenges:

- Labeled data is scarce and difficult to collect.
- There is a low tolerance for unrecoverable errors, as normalization errors cascade down to subsequent models. Normalization errors that alter the input semantics are called unrecoverable.

Text normalization and Inverse text normalization systems support a wide variety of *semiotic classes*, that is, words or tokens where the spoken form differs from the written form, requiring normalization. Examples are dates, decimals, cardinals, measures, and so on.

Steps to Text Normalization

There are four main steps in text normalization:

- **Case normalization**
- *Tokenization and stop word removal*
- *Parts-of-Speech (POS) tagging*
- *Stemming.*

REFER UNIT 1 for the detailed explanation

Case normalization

Case normalization applies to languages that use uppercase and lowercase letters. All languages based on the Latin alphabet or the Cyrillic alphabet (Russian, Mongolian, and so on) use upper- and lowercase letters. Other languages that sometimes use this are Greek, Armenian, Cherokee, and Coptic. In the case normalization technique, all letters are converted to the same case. It is quite helpful in semantic use cases. However, in other cases, this may hinder performance. In the spam example, spam messages may have more words in all-caps compared to regular messages.

Case normalization provides case-normalized alternatives for words which, by their position in a sentence or because they occur in a title, may or may not appear with their inherent, meaningful capitalization.

Letter-to-sound

Letter to sound module plays a very important role in a TTS system. Letter-to-sound (LTS) conversion is a process used in linguistics, natural language processing, and speech synthesis to convert written text, typically in the form of letters or characters, into corresponding phonetic representations or sound sequences. The primary goal of letter-to-sound conversion is to enable text-to-speech (TTS) systems to produce spoken language from written text.The process of letter-to-sound conversion involves several key steps:

1. **Text Analysis:** The first step is to analyze the input text, typically at the level of individual words or sub-word units. This involves breaking down the text into its constituent letters or characters and determining how they are pronounced in the given language. Languages often have specific rules for how letters are pronounced in different contexts.

2. **Grapheme-to-Phoneme Rules:** Graphemes are the written representations of letters or characters, and phonemes are the basic units of sound in a language. Letter-to-sound conversion uses rules, often referred to as grapheme-to-phoneme rules, to map each grapheme to its corresponding phoneme. These rules can be language-specific and may account for various factors like letter combinations, word stress, and syllable structure.

3. **Contextual Considerations:** In many languages, the pronunciation of a letter or character can vary depending on the letters surrounding it. Letter-to-sound conversion systems may take into account the context of a letter within a word to generate the correct phonemic representation.

4. **Lexicon and Exception Handling:** Some words or names may not follow the regular grapheme-to-phoneme rules and require exceptions to be included in the system's lexicon. A lexicon is a dictionary that contains mappings of specific words or names to their corresponding phonetic representations.

5. **Prosody and Intonation:** In addition to converting individual letters to phonemes, letter-to-sound conversion may also consider aspects of prosody, such as word stress, intonation patterns, and pitch, to produce more natural-sounding speech.

Letter-to-sound conversion is crucial for the development of text-to-speech (TTS) systems, as it enables these systems to generate human-like speech from written text. TTS systems can be used in various applications, such as *voice assistants, audiobooks, accessibility tools for the visually impaired, and more*. Accurate letter-to-sound conversion is essential for the overall quality and naturalness of the synthesized speech.

It is difficult for a person to create a letter-to-sound module because it is hard to be a professional speaker of many languages. For a large TTS system, the orthodox method is to store a pronunciation lexicon with a reasonable number of items. And only apply letter to sound module to those not listed in this lexicon. But for embedded systems, it is not a smart way to use this method, because the lexicon will sharply increase the engine's size. The improved method is for the most frequently used words, apply letter-to-sound, for the words with high usage frequency but can NOT get the right prediction with a letter-to-sound module, add them to an exception list.

There are two types of Letter-to-sound modules:

- **Rule-based Letter-to-sound Module**
  This module manually collects rules, with sufficient experience and language knowledge. This method is commonly referred to as the rule-based method. For a given language, if there exists a systematic relationship between a word format and its pronunciation, rule-based letter-to-sound can be efficient. The workflow of rule-based letter-to-sound is as follows. Firstly, it invite experienced linguists to create a set of pronunciation rules for a language. The programmer translates the rules that are expressed by natural language into a set of machine-readable rules. At the same time, we collect a number of pronunciation items for testing the rules. Secondly, the predicted items are compared with standard items to see what is the precision of the current rule set. If the precision is not high enough, we can modify the rule set and the codes. This interactive process is repeated until the precision reaches a predefined level within the acceptable code size.
- **Training-based Letter-to-sound Module**
  This module acquires rules by machine learning, which is called a training-based method generally. A training-based letter-to-sound module learns letter-to-sound conversion rules and patterns from data during a training process. This module is designed to automatically generate phonetic representations (pronunciations) for words or text based on a dataset of text examples and their corresponding phonetic transcriptions.

PROSODY ANALYSIS

Prosody analysis is the examination and study of the acoustic, rhythmic, melodic, and intonational features of speech that go beyond the individual phonemes and words. It encompasses the analysis of suprasegmental elements, which are aspects of speech that span multiple phonemes or syllables. These elements are critical for understanding the rhythm, melody, and emotional content of spoken language. Prosody analysis involves the following key components:

1. **Pitch:** Pitch refers to the perceived highness or lowness of a speaker's voice. Prosody analysis involves measuring changes in pitch, known as intonation, which can indicate the rising or falling patterns in speech. Pitch variations are used to convey emphasis, question or statement intonation, and emotional expression.

2. **Duration:** Duration refers to the length of time that speech sounds, syllables, words, or phrases are produced. Prosody analysis considers variations in speech duration to determine the pacing, rhythm, and pauses in speech, all of which contribute to the overall expressiveness and meaning of spoken language.

3. **Intensity:** Intensity, also known as loudness, is a measure of the strength of the acoustic signal. Prosody analysis examines variations in intensity to determine how loud or soft specific parts of speech are. These variations can convey emotional expressiveness, emphasis, and sentence-level stress.

4. **Speech Rate:** Speech rate is the speed at which a speaker delivers speech. Prosody analysis involves measuring speech rate and tempo, which can indicate aspects like nervousness, excitement, and deliberate communication style.

5. **Rhythm and Timing:** Prosody analysis includes studying the rhythm of speech, which encompasses the regularity or irregularity of stressed and unstressed syllables. Rhythmic patterns in speech contribute to the cadence and flow of spoken language.

6. **Emotional Expression:** Prosody analysis is instrumental in identifying emotional content in speech. Variations in pitch, intensity, and rhythm can convey emotions such as happiness, sadness, anger, or surprise. This is particularly important in applications like sentiment analysis and emotion recognition.

7. **Pragmatic and Discourse Functions:** Prosody plays a crucial role in signaling pragmatic information in conversation, including turn-taking, distinguishing between questions and statements, conveying emphasis, and even indicating sarcasm.

Prosody analysis has a wide range of practical applications, including:

- **Speech Recognition:** It helps improve the accuracy of automatic speech recognition systems by considering prosodic cues in addition to phonetic content.

- **Text-to-Speech Synthesis:** In speech synthesis, prosody analysis is used to generate more natural and expressive synthesized speech by incorporating appropriate intonation, stress, and pacing.

- **Emotion Recognition:** Prosody analysis is a key component in recognizing emotional states from spoken language, which is valuable in applications such as virtual assistants and customer service.

- **Language Learning and Teaching:** Prosody analysis is used to teach and learn the prosodic features of a language, including its intonation patterns and speech rhythm.

- **Psycholinguistics and Cognitive Science:** It plays a role in studying how humans perceive and process prosody in speech, shedding light on the cognitive mechanisms involved in language understanding.

Overall, prosody analysis is essential for understanding the subtleties and nuances of spoken language, as it provides crucial cues beyond the mere content of words, contributing to effective communication, natural speech synthesis and recognition, and conveying emotional and pragmatic information.

PROSODIC EVALUATION

Prosody evaluation is the process of assessing and analyzing the prosodic features of speech. Prosody refers to the rhythm, melody, and intonation of speech, as well as other suprasegmental aspects, such as stress, pitch, tempo, and pausing. Prosody plays a crucial role in conveying the emotional and communicative aspects of spoken language and can greatly affect how a message is interpreted. Prosody evaluation aims to measure and understand these features for various purposes, including speech analysis, speech synthesis, language learning, and more. Here are some key aspects of prosody evaluation:

1. **Pitch and Intonation:** Pitch refers to the perceived frequency of a person's voice. Intonation involves the rise and fall of pitch in speech. Prosody evaluation assesses pitch patterns to determine how they contribute to the meaning and emotional expression in speech.

2. **Stress and Emphasis:** Prosody evaluation examines how speakers use stress and emphasis to highlight certain words or syllables, which can change the meaning or emphasis in a sentence.

3. **Tempo and Timing:** It assesses the tempo or speech rate, including factors like speech speed, speech rate variation, and timing of pauses. Prosody evaluation can reveal whether a speaker's tempo is appropriate for the context or if it conveys urgency, excitement, or other emotions.

4. **Rhythm and Syllable Structure:** The evaluation may also focus on the rhythm and syllable structure of speech, looking at how syllables are grouped and pronounced in connected speech.

5. **Emotional Expression:** Prosody plays a vital role in conveying emotions and attitudes in speech. Evaluation may assess how well a speaker conveys emotions such as happiness, sadness, anger, or sarcasm through their prosody.

6. **Speech Synthesis:** In the context of speech synthesis (text-to-speech systems), prosody evaluation is used to ensure that the generated speech sounds natural and conveys the intended meaning. This can involve evaluating the system's ability to produce varied intonation patterns, stress, and pacing.

7. **Language Learning and Teaching:** Prosody evaluation is used in language learning and teaching to help learners improve their pronunciation, stress patterns, and intonation in a foreign language. It can provide feedback to learners on how well they are reproducing the prosody of the target language.

8. **Speaker Identification:** Prosody evaluation can also be used in speaker identification or verification systems, where the unique prosodic features of a person's speech can serve as a biometric identifier.

Methods for prosody evaluation include perceptual evaluations, acoustic analysis, and machine learning techniques. Perceptual evaluation involves human listeners who rate or categorize speech based on prosodic features. Acoustic analysis uses software tools to extract prosodic parameters from speech recordings. Machine learning techniques can be employed to automatically analyze and classify prosodic features.

Overall, prosody evaluation is a multidisciplinary field that helps us better understand how the melody and rhythm of speech impact communication and can be applied in various domains to enhance the quality and effectiveness of spoken language.

SIGNAL PROCESSING

Signal processing plays a fundamental role in speech synthesis, which is the process of generating artificial speech from text or other forms of input. The main objective of signal processing in speech synthesis is to manipulate and transform digital representations of speech signals to produce natural-sounding and intelligible synthetic speech. Speech synthesis systems aim at producing intelligible speech signals from some type of input language representation. Synthesis systems are also often referred to as text-to-speech (TTS) systems, as written text is a natural way to instruct what type of utterances should be produced.

Synthesis algorithms

The second key module consists of the speech synthesizer module. Input to the synthesizer is the linguistic representation produced by the text analysis block while the output consists of acoustic waveforms of synthesized speech. There are several potential technical approaches to the creation of a speech waveform from linguistic instructions. Historically, methods such as formant synthesis or articulatory synthesis have been utilized (where the latter is still used in speech research). However, modern commercial speech synthesizers are based on one of the two alternative techniques: **concatenative synthesis** **or** **statistical parametric speech synthesis.** Both methods are described in more detail in their respective sub-sections.

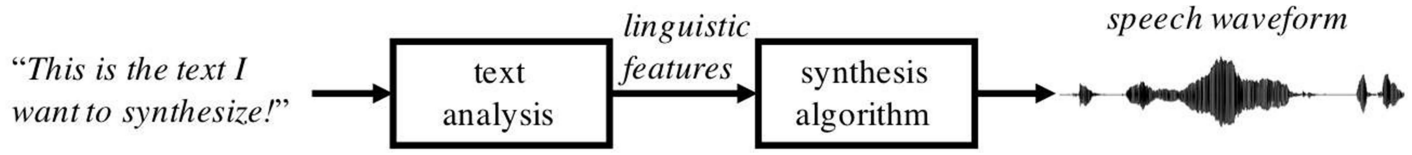


Figure 1: The basic structure of a speech synthesis system.

In general, synthesis algorithms aim at speech output that maximally resembles natural speech, is free of noise and artifacts, and has high intelligibility. Other characteristics may include possibility to use different speaker voices or to speaking styles to account for different use contexts and user preferences. In practical use, computational complexity of the system may also become a relevant design factor. This is especially true if the system must support real-time speech production and/or serve multiple users simultaneously. For instance, speech synthesis used on a standard mobile device or in a car entertainment system must operate with strict latency computational complexity constraints.

Speech quality and intelligibility a speech synthesizer is typically evaluated using subjective listening tests.

Concatenative synthesis is a technique for synthesising sounds by concatenating short samples of recorded sound (called units). The duration of the units is not strictly defined and may vary according to the implementation, roughly in the range of 10 milliseconds up to 1 second.

Statistical parametric synthesis might be most simply described as generating the average of some set of similarly sounding speech segments. This contrasts directly with the desire in unit selection to keep the natural unmodi- fied speech units, but using parametric models offers other benefits.

Concatenation synthesis

Concatenative synthesis is based on the concatenation (stringing together) of segments of recorded speech. Generally, concatenative synthesis produces the most natural-sounding synthesized speech. However, differences between natural variations in speech and the nature of the automated techniques for segmenting the waveforms sometimes result in audible glitches in the output. There are three main sub-types of concatenative synthesis.

Unit selection synthesis

Unit selection synthesis uses large databases of recorded speech. During database creation, each recorded utterance is segmented into some or all of the following: individual phones, diphones, half-phones, syllables, morphemes, words, phrases, and sentences. Typically, the division into segments is done using a specially modified speech recognizer set to a "forced alignment" mode with some manual correction afterward, using visual representations such as the waveform and spectrogram. An index of the units in the speech database is then created based on the segmentation and acoustic parameters like the fundamental frequency (pitch), duration, position in the syllable, and neighboring phones. At run time, the desired target utterance is created by determining the best chain of candidate units from the database (unit selection). This process is typically achieved using a specially weighted decision tree.

Unit selection provides the greatest naturalness because it applies only a small amount of digital signal processing (DSP) to the recorded speech. DSP often makes recorded speech sound less natural, although some systems use a small amount of signal processing at the point of concatenation to smooth the waveform. The output from the best unit-selection systems is often indistinguishable from real human voices, especially in contexts for which the TTS system has been tuned. However, maximum naturalness typically requires unit-selection speech databases to be very large, in some systems ranging into the gigabytes of recorded data, representing dozens of hours of speech. Also, unit selection algorithms have been known to select segments from a place that results in less than ideal synthesis (e.g. minor words become unclear) even when a better choice exists in the database. Recently, researchers have proposed various automated methods to detect unnatural segments in unit-selection speech synthesis systems.

Diphone synthesis

Diphone synthesis uses a minimal speech database containing all the diphones (sound-to-sound transitions) occurring in a language. The number of diphones depends on the phonotactics of the language: for example, Spanish has about 800

diphones, and German about 2500. In diphone synthesis, only one example of each diphone is contained in the speech database. At runtime, the target prosody of a sentence is superimposed on these minimal units by means of digital signal processing technique or more recent techniques such as pitch modification in the source domain using discrete cosine transform. Diphone synthesis suffers from the sonic glitches of concatenative synthesis and the robotic-sounding nature of formant synthesis, and has few of the advantages of either approach other than small size. As such, its use in commercial applications is declining, although it continues to be used in research because there are a number of freely available software implementations.

Domain-specific synthesis

Domain-specific synthesis concatenates prerecorded words and phrases to create complete utterances. It is used in applications where the variety of texts the system will output is limited to a particular domain, like transit schedule announcements or weather reports. The technology is very simple to implement, and has been in commercial use for a long time, in devices like talking clocks and calculators. The level of naturalness of these systems can be very high because the variety of sentence types is limited, and they closely match the prosody and intonation of the original recordings.

Because these systems are limited by the words and phrases in their databases, they are not general-purpose and can only synthesize the combinations of words and phrases with which they have been preprogrammed. The blending of words within naturally spoken language however can still cause problems unless the many variations are taken into account. For example, in non-rhotic dialects of English the *"r"* in words like *"clear"* /ˈklɪə/ is usually only pronounced when the following word has a vowel as its first letter (e.g. *"clear out"* is realized as /ˌklɪəɹˈʌʊt/). Likewise in French, many final consonants become no longer silent if followed by a word that begins with a vowel, an effect called liaison. This alternation cannot be reproduced by a simple word-concatenation system, which would require additional complexity to be context-sensitive.

Statistical Parametric Speech Synthesis (SPSS)

A complete SPSS system is generally composed of three modules: a text analysis module, a parameter prediction module which uses a statistical model to predict the acoustic feature parameters such as fundamental frequency (F0), spectral parameters and duration, and a speech synthesis module. The text analysis module mainly preprocesses the input text and transforms it into linguistic features used by the speech synthesis system, including text normalization , automatic word segmentation, and grapheme-to-phoneme conversion. These linguistic features usually include phoneme, syllable, word, phrase and sentence-level features. The purpose of the parameter prediction module is to predict the acoustic feature parameters of the target speech according to the output of the text analysis module. The speech synthesis module generates the waveform of the target speech according to the output of the parameter prediction module by using a particular synthesis algorithm. The SPSS is usually divided into two phases: the training phase and the synthesis phase. In the training phase, acoustic feature parameters such as F0 and spectral parameters are firstly extracted from the corpus, and then a statistical acoustic model is trained based on the linguistic features of the text analysis module as well as the extracted acoustic feature parameters. In the synthesis phase, the acoustic feature parameters are predicted using the trained acoustic model with the guidance of the linguistic features. Finally, the speech is synthesized based on the predicted acoustic feature parameters using a vocoder.

Formant synthesis

Formant synthesis does not use human speech samples at runtime. Instead, the synthesized speech output is created using additive synthesis and an acoustic model (physical modelling synthesis). Parameters such as fundamental frequency, voicing, and noise levels are varied over time to create a waveform of artificial speech. This method is sometimes called *rules-based synthesis*; however, many concatenative systems also have rules-based components. Many systems based on formant synthesis technology generate artificial, robotic-sounding speech that would never be mistaken for human speech. However, maximum naturalness is not always the goal of a speech synthesis system, and formant synthesis systems have advantages over concatenative systems. Formant-synthesized speech can be reliably intelligible, even at very high speeds, avoiding the acoustic glitches that commonly plague concatenative systems. High-speed synthesized speech is used by the visually impaired to quickly navigate computers using a screen reader. Formant synthesizers are usually smaller programs than concatenative systems because they do not have a database of speech samples. They can therefore be used in embedded systems, where memory and microprocessor power are especially limited. Because formant-based systems

have complete control of all aspects of the output speech, a wide variety of prosodies and intonations can be output, conveying not just questions and statements, but a variety of emotions and tones of voice.

Articulatory synthesis

Articulatory synthesis refers to computational techniques for synthesizing speech based on models of the human vocal tract and the articulation processes occurring there. The first articulatory synthesizer regularly used for laboratory experiments was developed at Haskins Laboratories in the mid-1970s by Philip Rubin, Tom Baer, and Paul Mermelstein. This synthesizer, known as ASY, was based on vocal tract models developed at Bell Laboratories in the 1960s and 1970s by Paul Mermelstein, Cecil Coker, and colleagues.

Until recently, articulatory synthesis models have not been incorporated into commercial speech synthesis systems. A notable exception is the NeXT-based system originally developed and marketed by Trillium Sound Research, a spin-off company of the University of Calgary, where much of the original research was conducted. Following the demise of the various incarnations of NeXT (started by Steve Jobs in the late 1980s and merged with Apple Computer in 1997), the Trillium software was published under the GNU General Public License, with work continuing as gnuspeech. The system, first marketed in 1994, provides full articulatory-based text-to-speech conversion using a waveguide or transmission-line analog of the human oral and nasal tracts controlled by Carré's "distinctive region model".

More recent synthesizers, developed by Jorge C. Lucero and colleagues, incorporate models of vocal fold biomechanics, glottal aerodynamics and acoustic wave propagation in the bronchi, trachea, nasal and oral cavities, and thus constitute full systems of physics-based speech simulation.

HMM-based synthesis

HMM-based synthesis is a synthesis method based on hidden Markov models, also called Statistical Parametric Synthesis. In this system, the frequency spectrum (vocal tract), fundamental frequency (voice source), and duration (prosody) of speech are modeled simultaneously by HMMs. Speech waveforms are generated from HMMs themselves based on the maximum likelihood criterion.

Sinewave synthesis

Sinewave synthesis is a technique for synthesizing speech by replacing the formants (main bands of energy) with pure tone whistles.

Deep learning-based synthesis

Deep learning speech synthesis uses Deep Neural Networks (DNN) to produce artificial speech from text (text-to-speech) or spectrum (vocoder). The deep neural networks are trained using a large amount of recorded speech and, in the case of a text-to-speech system, the associated labels and/or input text. It is known that the HMM-based speech synthesis method maps linguistic features into probability densities of speech parameters with various decision trees. Different from the HMM-based method, the DL-based method directly perform mapping from linguistic features to acoustic features with deep neural networks which have proven extraordinarily efficient at learning inherent features of data. In the long tradition of studies that adopt DL-based method for speech synthesis, people have proposed numerous models. a brief overview of the advantages and disadvantages in Table 1 and makes a detailed introduction in the following.

Table 1. The advantages and disadvantages of different speech synthesis methods, including hidden Markov model (HMM), restrictive Boltzmann machine (RBM), deep belief network (DBN), deep mixture density network (DMDN), deep bidirectional long short-term memory (DBLSTM), WaveNet, Tacotron and convolutional neural network (CNN).

Methods	Advantages	Disadvantages
HMM	Flexible with changing voice characteristics and the system is robust	The acoustic features are oversmoothed, making the generated speech sounds muffled
RBM	Can better describe the distribution of high-dimensional spectral envelopes to alleviate the over-smooth problem	Suffer from the fragementation problem of training data
DBN	Cannot suffer from the training data fragementation problem and reduce the over-smoothing problem	The quality of generated speech will be degraded
DMDN	Can solve the single modality problem	Can only leverage limited contexts and each frame is mapped independently
DBLSTM	Can fully leverage contextual information	Still needs a vocoder to synthesize waveform
WaveNet	Can produce high-quality speech waveforms	Too slow and the errors from the front-end will affect the synthesis effect
Tacotron	Fully end-to-end speech synthesis model and can produce high-quality speech waveforms	Quite costly to train the model
CNN	Fast to train the model	The speech quality might be degraded

4.1. Restrictive Boltzmann Machines for Speech Synthesis

In the field of speech synthesis, Boltzmann machine (RBM) is usually regarded as a density model for generating the spectral envelope of acoustic parameters. It is adopted to better describe the distribution of high-dimensional spectral envelopes to alleviate the over-smooth problem in HMM-based speech synthesis.

4.2. Multi-distribution Deep Belief Networks for Speech Synthesis

Multi-distribution deep belief network (DBN) [24] is a method of modeling the joint distribution of context information and acoustic features. It models the coutinuous spectral, discrete voiced/unvoiced (V/UV) parameters and the multi-space F0 simultaneously with three types of RBMs. In DBNs, the visible unit can obey different probability distributions, therefore, it is possible to characterize the supervectors that are composed of these features.

4.3. Deep Bidirectional LSTM-based Speech Synthesis

BLSTM-RNN is an extended architecture of bidirectional recurrent neural network (BRNN). It replaces units in the hidden layers of BRNN with LSTM memory blocks. With these memory blocks, BLSTM can store information for long and short time lags, and leverage relevant contextual dependencies from both forward and backward directions for machine learning tasks.

When using deep BLSTM-based (DBLSTM) model to predict acoustic parameters for speech synthesis, first we need to convert the input text prompt into a feature vector, and then use the DBLSTM model to map the input feature to acoustic parameters. Finally, the parameter generation algorithm is used to generate the acoustic parameters and a vocoder is utilized to synthesize the corresponding speech.

4.4. End-to-End Speech Synthesis

A TTS system typically consists of a text analysis front-end, an acoustic model and a speech synthesizer. Since these components are trained independently and rely on extensive domain expertise which are laborious, errors from each component may compound. To address these problems, end-to-end speech synthesis methods which combine those components into a unified framework have become the main stream of speech synthesis field. In the following we will give a brief introduction to the end-to-end speech synthesis methods.

4.4.1. Speech Synthesis Based on WaveNet

WaveNet is a complete probabilistic autoregressive model that predicts the probability distribution of the current audio sample based on all samples which have been generated before. As an important component of WaveNet, dilated causal convolutions are used to ensure that WaveNet can only use the sampling points from 0 to t - 1 while generating the tth sampling point. The original WaveNet model uses autoregressive connections to synthesize waveforms one sample at a time, with each new sample conditioned on the previous ones.

4.4.2. Speech Synthesis Based on Tacotron

Tacotron is a fully end-to-end speech synthesis model. It is capable of training speech synthesis model given <text, audio> pairs, thus alleviating the need for laborious feature engineering. And since it is based on character level, it can be applied in almost all kinds of languages including Chinese Mandarin. Tacotron uses seq2seq model with attention mechanism to map text to spectrogram which is a good representation of speech. Since spectrogram doesn't contain phase information, the system uses Griffin-Lim algorithm to reconstruct the audio by estimating the phase information from the spectrogram iteratively.

CCS369- TEXT AND SPEECH ANALYSIS

UNIT 5

SPEECH RECOGNITION

Speech recognition, also known as automatic speech recognition (ASR), computer speech recognition, or speech-to-text, is a capability which enables a program to process human speech into a written format. While it's commonly confused with voice recognition, speech recognition focuses on the translation of speech from a verbal format to a text one whereas voice recognition just seeks to identify an individual user's voice.

peech recognition, or speech-to-text, is the ability of a machine or program to identify words spoken aloud and convert them into readable text. Rudimentary speech recognition software has a limited vocabulary and may only identify words and phrases when spoken clearly. More sophisticated software can handle natural speech, different accents and various languages.

Speech recognition uses a broad array of research in computer science, linguistics and computer engineering. Many modern devices and text-focused programs have speech recognition functions in them to allow for easier or hands-free use of a device.

Speech recognition and voice recognition are two different technologies and should not be confused:

- **Speech recognition** is used to identify words in spoken language.
- **Voice recognition** is a biometric technology for identifying an individual's voice.

How does speech recognition work?

Speech recognition systems use computer algorithms to process and interpret spoken words and convert them into text. A software program turns the sound a microphone records into written language that computers and humans can understand, following these four steps:

1. analyze the audio;
2. break it into parts;
3. digitize it into a computer-readable format, and
4. use an algorithm to match it to the most suitable text representation.

Speech recognition software must adapt to the highly variable and context-specific nature of human speech. The software algorithms that process and organize audio into text are trained on different speech patterns, speaking styles, languages, dialects, accents and phrasings. The software also separates spoken audio from background noise that often accompanies the signal.

To meet these requirements, speech recognition systems use two types of models:

- **Acoustic models.** These represent the relationship between linguistic units of speech and audio signals.
- **Language models.** Here, sounds are matched with word sequences to distinguish between words that sound similar.

What applications is speech recognition used for?

Speech recognition systems have quite a few applications. Here is a sampling of them.

Mobile devices. Smartphones use voice commands for call routing, speech-to-text processing, voice dialing and voice search. Users can respond to a text without looking at their devices. On Apple iPhones, speech recognition powers the keyboard and Siri, the virtual assistant. Functionality is available in secondary

languages, too. Speech recognition can also be found in word processing applications like Microsoft Word, where users can dictate words to be turned into text.

Education. Speech recognition software is used in language instruction. The software hears the user's speech and offers help with pronunciation.

Customer service. Automated voice assistants listen to customer queries and provides helpful resources.

Healthcare applications. Doctors can use speech recognition software to transcribe notes in real time into healthcare records.

Disability assistance. Speech recognition software can translate spoken words into text using closed captions to enable a person with hearing loss to understand what others are saying. Speech recognition can also enable those with limited use of their hands to work with computers, using voice commands instead of typing.

Court reporting. Software can be used to transcribe courtroom proceedings, precluding the need for human transcribers.

Emotion recognition. This technology can analyze certain vocal characteristics to determine what emotion the speaker is feeling. Paired with sentiment analysis, this can reveal how someone feels about a product or service.

Hands-free communication. Drivers use voice control for hands-free communication, controlling phones, radios and global positioning systems, for instance.

What are the features of speech recognition systems?

Good speech recognition programs let users customize them to their needs. The features that enable this include:

- **Language weighting.** This feature tells the algorithm to give special attention to certain words, such as those spoken frequently or that are unique to the conversation or subject. For example, the software can be trained to listen for specific product references.
- **Acoustic training.** The software tunes out ambient noise that pollutes spoken audio. Software programs with acoustic training can distinguish speaking style, pace and volume amid the din of many people speaking in an office.
- **Speaker labeling.** This capability enables a program to label individual participants and identify their specific contributions to a conversation.
- **Profanity filtering.** Here, the software filters out undesirable words and language.

What are the different speech recognition algorithms?

The power behind speech recognition features comes from a set of algorithms and technologies. They include the following:

- **Hidden Markov model.** HMMs are used in autonomous systems where a state is partially observable or when all of the information necessary to make a decision is not immediately available to the sensor (in speech recognition's case, a microphone). An example of this is in acoustic modeling, where a program must match linguistic units to audio signals using statistical probability.
- **Natural language processing.** NLP eases and accelerates the speech recognition process.
- **N-grams.** This simple approach to language models creates a probability distribution for a sequence. An example would be an algorithm that looks at the last few words spoken, approximates the history of the sample of speech and uses that to determine the probability of the next word or phrase that will be spoken.

- **Artificial intelligence.** AI and machine learning methods like deep learning and neural networks are common in advanced speech recognition software. These systems use grammar, structure, syntax and composition of audio and voice signals to process speech. Machine learning systems gain knowledge with each use, making them well suited for nuances like accents.

What are the advantages of speech recognition?

There are several advantages to using speech recognition software, including the following:

- **Machine-to-human communication.** The technology enables electronic devices to communicate with humans in natural language or conversational speech.
- **Readily accessible.** This software is frequently installed in computers and mobile devices, making it accessible.
- **Easy to use.** Well-designed software is straightforward to operate and often runs in the background.
- **Continuous, automatic improvement.** Speech recognition systems that incorporate AI become more effective and easier to use over time. As systems complete speech recognition tasks, they generate more data about human speech and get better at what they do.

What are the disadvantages of speech recognition?

While convenient, speech recognition technology still has a few issues to work through. Limitations include:

- **Inconsistent performance.** The systems may be unable to capture words accurately because of variations in pronunciation, lack of support for some languages and inability to sort through background noise. Ambient noise can be especially challenging. Acoustic training can help filter it out, but these programs aren't perfect. Sometimes it's impossible to isolate the human voice.
- **Speed.** Some speech recognition programs take time to deploy and master. The speech processing may feel relatively slow**.**
- **Source file issues.** Speech recognition success depends on the recording equipment used, not just the software.

Acoustic Modelling

Acoustic modelling of speech typically refers to the process of establishing statistical representations for the feature vector sequences computed from the speech waveform. Hidden Markov Model (HMM) is one most common types of acoustic models.**Modern speech recognition systems use both an acoustic model and a language model to represent the statistical properties of speech.** The acoustic model models the relationship between the audio signal and the phonetic units in the language. The language model is responsible for modeling the word sequences in the language. These two models are combined to get the top-ranked word sequences corresponding to a given audio segment.

Speech audio characteristics

Audio can be encoded at different sampling rates (i.e. samples per second – the most common being: 8, 16, 32, 44.1, 48, and 96 kHz), and different bits per sample (the most common being: 8-bits, 16-bits, 24-bits or 32-bits). Speech recognition engines work best if the acoustic model they use was trained with speech audio which was recorded at the same sampling rate/bits per sample as the speech being recognized.

Telephony-based speech recognition

The limiting factor for telephony based speech recognition is the bandwidth at which speech can be transmitted. For example, a standard land-line telephone only has a bandwidth of 64 kbit/s at a sampling rate of 8 kHz and 8-bits per sample (8000 samples per second * 8-bits per sample = 64000 bit/s). Therefore, for telephony based speech recognition, acoustic models should be trained with 8 kHz/8-bit speech audio files.

In the case of Voice over IP, the codec determines the sampling rate/bits per sample of speech transmission. Codecs with a higher sampling rate/bits per sample for speech transmission (which improve the sound quality) necessitate acoustic models trained with audio data that matches that sampling rate/bits per sample.

Desktop-based speech recognition

For speech recognition on a standard desktop PC, the limiting factor is the sound card. Most sound cards today can record at sampling rates of between 16 kHz-48 kHz of audio, with bit rates of 8 to 16-bits per sample, and playback at up to 96 kHz.

As a general rule, a speech recognition engine works better with acoustic models trained with speech audio data recorded at higher sampling rates/bits per sample. But using audio with too high a sampling rate/bits per sample can slow the recognition engine down. A compromise is needed. Thus for desktop speech recognition, the current standard is acoustic models trained with speech audio data recorded at sampling rates of 16 kHz/16bits per sample.

Acoustic modeling is a crucial component in the field of automatic speech recognition (ASR) and various other applications involving spoken language processing. It is the process of creating a statistical representation of the relationship between acoustic features and phonemes, words, or other linguistic units in a spoken language. Acoustic models play a central role in converting spoken language into text and are a key part of the larger ASR system. Here's how acoustic modeling works:

1. **Feature Extraction:** The process starts with capturing audio input, which is typically sampled at a high rate. Feature extraction is performed to convert this raw audio into a more compact and informative representation. Common acoustic features include Mel-frequency cepstral coefficients (MFCCs) or filterbank energies. These features capture the spectral characteristics of the audio signal over time.

2. **Training Data:** Acoustic modeling requires a significant amount of training data, typically consisting of transcribed audio recordings. This data is used to establish statistical patterns between acoustic features and the corresponding linguistic units (e.g., phonemes, words).

3. **Phoneme or State Modeling:** In traditional Hidden Markov Models (HMMs), which have been widely used in ASR, the acoustic modeling process involves modeling phonemes or states. An HMM represents a sequence of states, each associated with a specific acoustic observation probability distribution. These states correspond to phonemes or sub-phonetic units.

4. **Building Gaussian Mixture Models (GMMs):** For each state or phoneme, a Gaussian Mixture Model (GMM) is constructed. GMMs are a set of Gaussian distributions that model the likelihood of observing specific acoustic features given a phoneme or state. These GMMs capture the variation in acoustic features associated with each phoneme.

5. **Training the Models:** During training, the GMM parameters are estimated to maximize the likelihood of the observed acoustic features given the transcribed training data. This training process adjusts the means and covariances of the Gaussian components to fit the observed acoustic data.

6. **Decoding:** When transcribing new, unseen audio, the acoustic model is used in combination with language and pronunciation models. The ASR system uses these models to search for the most likely sequence of phonemes or words that best matches the observed acoustic features. Decoding algorithms like the Viterbi algorithm are commonly used for this task.

7. **Integration:** The output of the acoustic model is combined with language and pronunciation models to generate a final transcription or understanding of the spoken input.

Modern ASR systems have evolved beyond HMM-based approaches, with deep learning techniques, such as deep neural networks (DNNs) and recurrent neural networks (RNNs), becoming more prevalent in acoustic modeling. Deep learning models can directly map acoustic features to phonemes or words, bypassing the need

for GMMs and HMMs. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are often used for this purpose. These deep learning models have significantly improved the accuracy of ASR systems, making them more robust to various accents, noise, and speaking styles.

In summary, acoustic modeling is a crucial step in automatic speech recognition, responsible for establishing the statistical relationship between acoustic features and linguistic units. This process enables the conversion of spoken language into text, and advances in deep learning techniques have greatly improved the accuracy and efficiency of acoustic models in ASR systems.

Feature Extraction

Feature extraction is a fundamental step in acoustic modeling for tasks like automatic speech recognition (ASR) and speaker identification. Its primary goal is to convert the raw audio signal into a more compact and informative representation that captures relevant acoustic characteristics. The choice of acoustic features greatly impacts the performance of the acoustic model. Here are some common techniques for feature extraction in acoustic modeling:

1. **Mel-Frequency Cepstral Coefficients (MFCCs):** MFCCs are one of the most widely used acoustic features in ASR. They mimic the human auditory system's sensitivity to different frequencies. The MFCC extraction process typically involves the following steps:

- Pre-emphasis: Boosts high-frequency components to compensate for the muffled low frequencies in speech.

- Framing: The audio signal is divided into short overlapping frames, often around 20-30 milliseconds in duration.

- Windowing: Each frame is multiplied by a windowing function (e.g., Hamming window) to reduce spectral leakage.

- Fast Fourier Transform (FFT): The power spectrum of each frame is computed using the FFT.

- Mel-filterbank: A set of triangular filters on the Mel-scale is applied to the power spectrum. The resulting filterbank energies capture the distribution of energy in different frequency bands.

- Logarithm: The logarithm of filterbank energies is taken to simulate the human perception of loudness.

- Discrete Cosine Transform (DCT): DCT is applied to decorrelate the log filterbank energies and produce a set of MFCC coefficients.

2. **FilterbankEnergies:** These are similar to the intermediate step of MFCC computation but without the logarithm and DCT steps. **Filterbank energies**are a set of values that represent the energy in different frequency bands over time. They are often used in conjunction with MFCCs or as a simpler alternative when the benefits of MFCCs are not required.

3. **Spectrogram:** The spectrogram is a visual representation of the spectrum of frequencies in the audio signal over time. It is often used as a feature for tasks that benefit from a time-frequency representation, such as music genre classification and environmental sound recognition.

4. **Pitch and Fundamental Frequency (F0):** Extracting pitch information can be important for certain applications. Pitch is the perceived frequency of a sound and is often associated with prosody and intonation in speech

5. **Linear Predictive Coding (LPC):** LPC analysis models the speech signal as the output of an all-pole filter and extracts coefficients that represent the vocal tract's resonances. LPC features are used in speech coding and sometimes ASR.

6. **Perceptual Linear Prediction (PLP) Cepstral Coefficients:** PLP is an alternative to MFCCs that incorporates psychoacoustic principles, modeling the human auditory system's response more closely.

7. **Deep Learning-Based Features:** In recent years, deep neural networks have been used to learn features directly from the raw waveform. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) can be used to capture high-level representations from audio data.

8. **Gammatone Filters:** These are designed to more closely mimic the response of the human auditory system to different frequencies.

The choice of feature extraction method depends on the specific task and the characteristics of the data. For ASR, MFCCs and filterbank energies are the most commonly used features. However, as deep learning techniques become more prevalent in acoustic modeling, end-to-end systems that operate directly on raw audio data are gaining popularity, and feature extraction is becoming integrated into the model architecture.

In signal processing, a filter bank (or filterbank) is an array of bandpass filters that separates the input signal into multiple components, each one carrying a single frequency sub-band of the original signal.

HMM

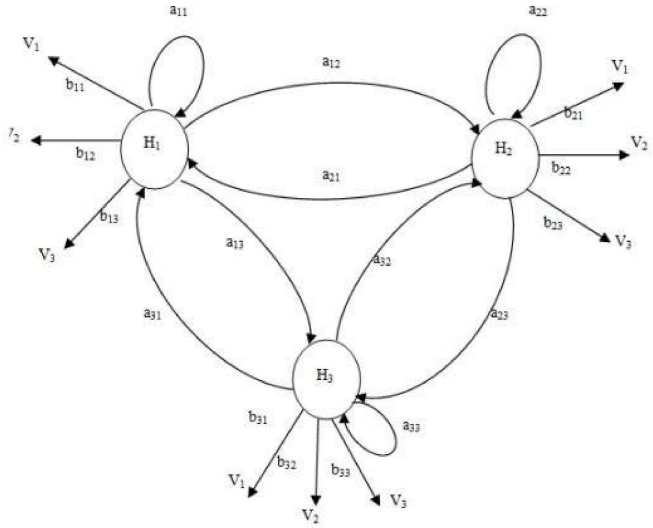


Figure 1. HMM Model

1. **Markov Process**:
   - A Markov process, also known as a Markov chain, is a stochastic model that describes a system's transitions from one state to another over discrete time steps.
   - In a simple Markov chain, the future state of the system depends only on the current state and is independent of all previous states. This property is called the Markov property.
2. **Hidden States**:
   - In an HMM, there are two sets of states: hidden states and observable states.
   - Hidden states represent the unobservable underlying structure of the system. They are responsible for generating the observable data.
3. **Observable Data**:

- Observable states are the data that can be directly measured or observed.
- For example, in speech recognition, the hidden states might represent phonemes, while the observable data are the audio signals.

4. **State Transitions**:
   - An HMM defines the probabilities of transitioning from one hidden state to another. These transition probabilities are often represented by a transition matrix.
   - Transition probabilities can be time-dependent (time-inhomogeneous) or time-independent (time-homogeneous).
5. **Emission Probabilities**:
   - Emission probabilities specify the likelihood of emitting observable data from a particular hidden state.
   - In the context of speech recognition, these probabilities represent the likelihood of generating a certain audio signal given the hidden state (e.g., a phoneme).
6. **Initialization Probabilities**:
   - An HMM typically includes initial probabilities, which represent the probability distribution over the initial hidden states at the start of the sequence.
7. **Observations and Inference**:
   - Given a sequence of observations (observable data), the goal is to infer the most likely sequence of hidden states.
   - This is typically done using algorithms like the Viterbi algorithm, which finds the most probable sequence of hidden states that generated the observations.
8. **Learning HMM Parameters**:
   - Training an HMM involves estimating its parameters, including transition probabilities, emission probabilities, and initial state probabilities.
   - This can be done using methods like the Baum-Welch algorithm, which is a variant of the Expectation-Maximization (EM) algorithm.
9. **Applications**:
   - HMMs have a wide range of applications, such as speech recognition, where they can model phonemes, natural language processing for part-of-speech tagging, bioinformatics for gene prediction, and more.
10. **Limitations**:
    - HMMs assume that the system is a first-order Markov process, which means it depends only on the current state. More complex dependencies might require more advanced models.
    - HMMs are also sensitive to their initial parameter estimates and might get stuck in local optima during training.

In summary, Hidden Markov Models are a powerful tool for modeling and analyzing sequential data with hidden structure. They are used in a variety of fields to uncover underlying patterns and make predictions based on observed data.

HMM-DNN

The hybrid HMM-DNN approach in speech recognition make use of the properties like the strong learning power of DNN and the sequential modelling activity of the HMM. As DNN accepts only fixed sized inputs it will be difficult to deal with speech signals as they are variable length time varying signal. So in this approach HMM deals with the dynamic characteristic of the speech signal and DNN is responsible for the observation probability. Given the acoustic observations, each output neuron of DNN is trained to estimate the posterior probability of continuous density HMM's state. DNN when trained in the usual traditional way through supervised manner does not produce good results and very difficult to get to an optimal point. When a set of

data is given as input, importance should be given to extract the variety of data rather than the quantity of data extracted because later on a good classification can be made from this data.

DNN-HMM systems, also known as Deep Neural Network-Hidden Markov Model systems, are a type of technology used in automatic speech recognition (ASR) and other sequential data modeling tasks. These systems combine deep neural networks (DNNs) with Hidden Markov Models (HMMs) to improve the accuracy and robustness of speech recognition and other related applications. Here's a detailed explanation of DNN-HMM systems:

1. **Hidden Markov Models (HMMs)**:

- As discussed earlier, HMMs are probabilistic models that describe the temporal evolution of a system. In ASR, they are used to model the sequence of phonemes or subword units that make up spoken language.

2. **Deep Neural Networks (DNNs)**:

- DNNs are a type of artificial neural network with multiple layers (hence "deep"). They have shown great success in various machine learning tasks, including image recognition, natural language processing, and speech processing.

- In the context of DNN-HMM systems, DNNs are used for acoustic modeling. They replace the traditional Gaussian Mixture Models (GMMs) that were used for modeling acoustic features in older HMM systems.

3. **Acoustic Modeling**:

- Acoustic modeling in ASR is the process of estimating the likelihood of observing a given acoustic feature (e.g., a frame of audio) given a particular state in the HMM.

- In DNN-HMM systems, DNNs are used to model this likelihood. They take acoustic features as input and produce the probability distribution over the set of states.

4. **Phoneme or SubwordModeling**:

- DNN-HMM systems typically model phonemes, context-dependent phonemes, or subword units (e.g., triphones) as the hidden states in HMMs.

- The DNNs are trained to predict which phoneme or subword unit corresponds to a given acoustic frame, given the surrounding context.

5. **Training**:

- DNN-HMM systems are trained using large datasets of transcribed speech. The DNNs are trained to minimize the error between their predicted state probabilities and the true state labels in the training data.

- DNNs can be trained using supervised learning techniques, and backpropagation is used to update the model's weights.

6. **Integration with HMMs**:

- The DNN-generated state probabilities are integrated into the HMM framework. This is often done by incorporating the DNN as an emission probability model in the HMM.

7. **Decoding**:

- During the decoding phase, DNN-HMM systems use algorithms like the Viterbi algorithm to find the most likely sequence of hidden states (phonemes or subword units) that best explain the observed acoustic features.

8. **Benefits**:

- DNN-HMM systems have significantly improved ASR accuracy, especially in challenging environments with background noise and variations in speech.

- They capture complex acoustic patterns and can model a wide range of speakers and accents effectively.

9. **Challenges**:

- Training deep neural networks requires large amounts of labeled data and significant computational resources.

- DNN-HMM systems can be complex to design and optimize, and there's a risk of overfitting the model to the training data.

DNN-HMM systems have been a major breakthrough in ASR technology and have significantly improved the accuracy of speech recognition systems, making them more practical for real-world applications, including voice assistants, transcription services, and more.