CCS369- TEXT AND SPEECH ANALYSIS

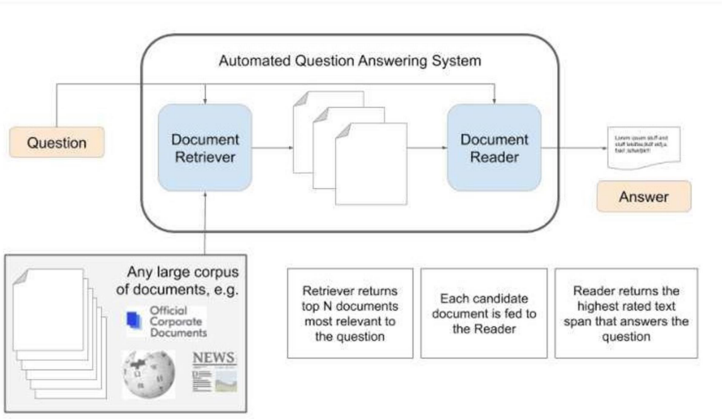
UNIT 3

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.



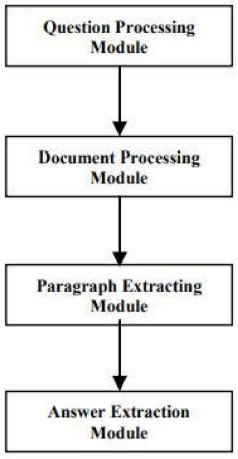


Fig 1: Framework of QA system

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**

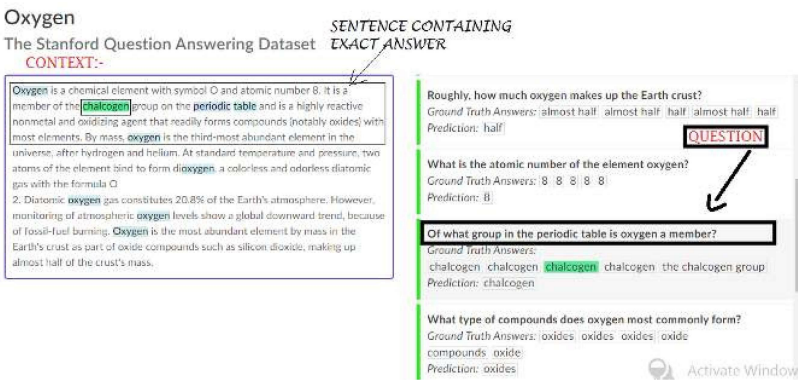
The availability of huge amounts of data on the internet increased the importance of statistical approaches. A statistical learning method gives better results than other approaches. Online text repositories and statistical approaches are independent of structured query languages and can formulate queries 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**

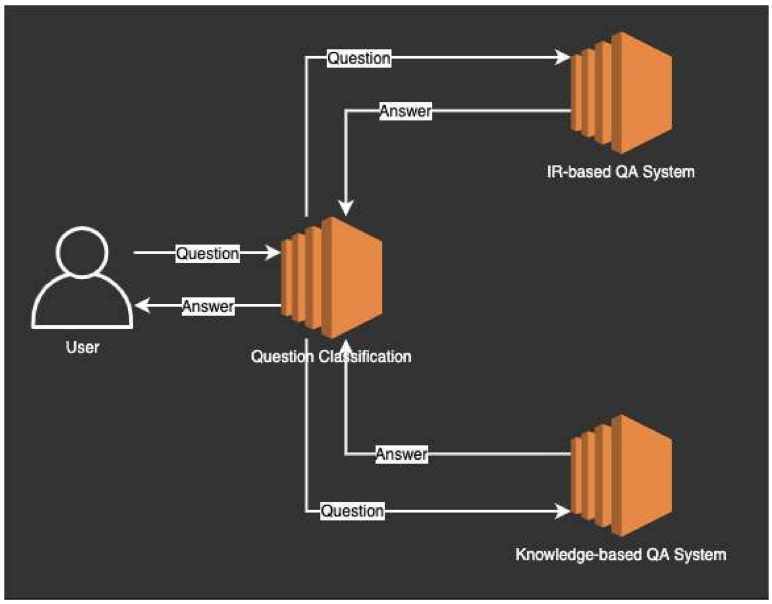
Pattern matching approach deals with the expressive power of text pattern, it replaces the sophisticated processing involved in other computing approaches. Most of the pattern-matching QA systems use the surface text pattern, while some of them also rely on templates for response generator

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



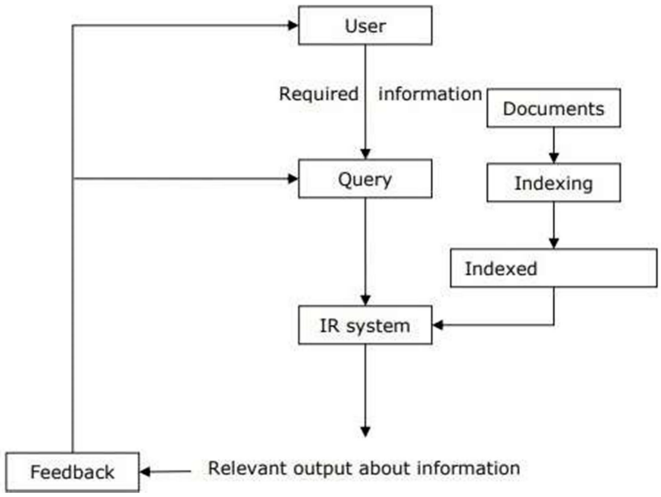
The purpose is to locate the text for any new question that has been addressed, as well as the context. This is a closed dataset, so the answer to a query is always a part of the context and that the context spans a continuous span. Here, divided the problem into two pieces as shown above.



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

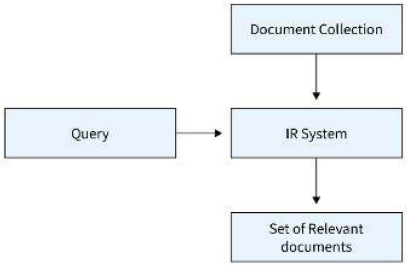
- The IR model has to provide a system for **how the documents in the collection** and user's queries are **transformed.**
- The IR model also needs to ingrain the functionality for **how the system identifies the relevancy** of the documents based on the query provided by the user.
- The system in the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

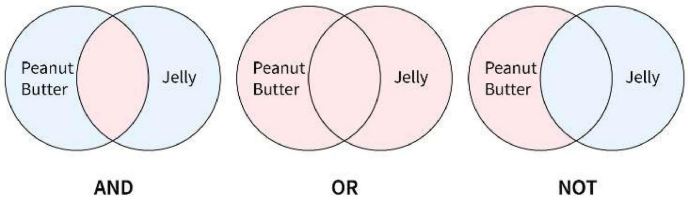
- It views the document as the **set of terms and creates the indexing** required for the Boolean retrieval model.
- The text data is represented in the form of a **matrix** where **rows** of the matrix represent the sentences and the **columns** of the matrix represent the word for the data which needs 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

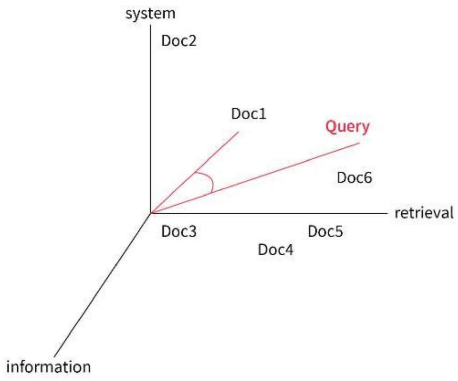
The disadvantages of the Boolean model are as follows –

- 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 –

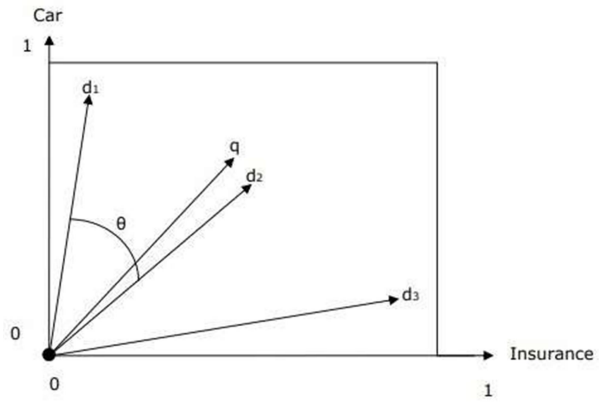
$$Score(\vec{d}\vec{q}) = \frac{\sum_{k=1}^{m} d_k \cdot q_k}{\sqrt{\sum_{k=1}^{m} (d_k)^2} \cdot \sqrt{\sum_{k=1}^{m} m(q_k)^2}}$$

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

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

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.

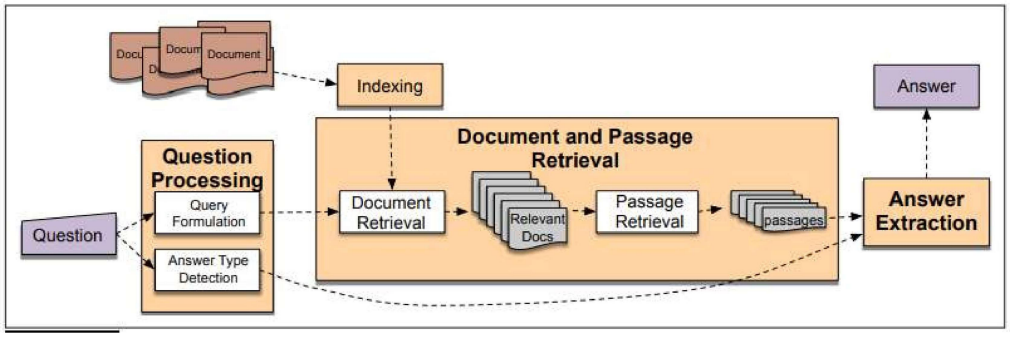
IR-BASED QUESTION ANSWERING

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

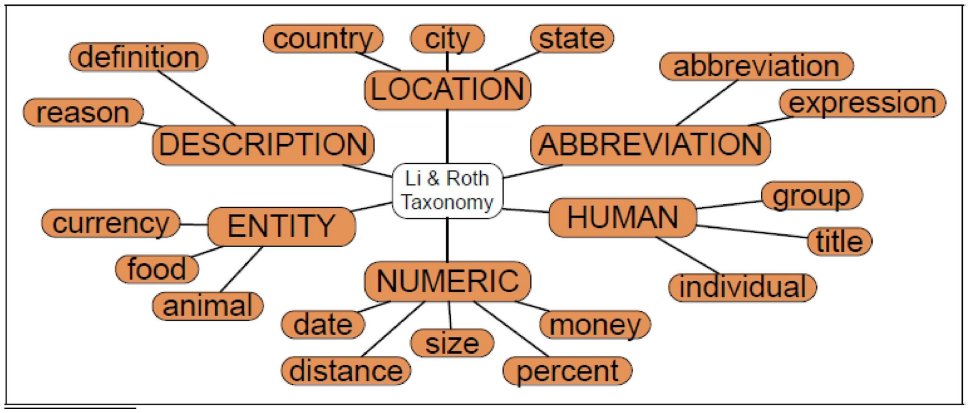
Query formulation is the task of creating a query—a list of tokens— to send to an information retrieval system to retrieve documents that might contain answer strings. For question answering from the web, we can simply pass the entire question
to the web search engine, at most perhaps leaving out the 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.

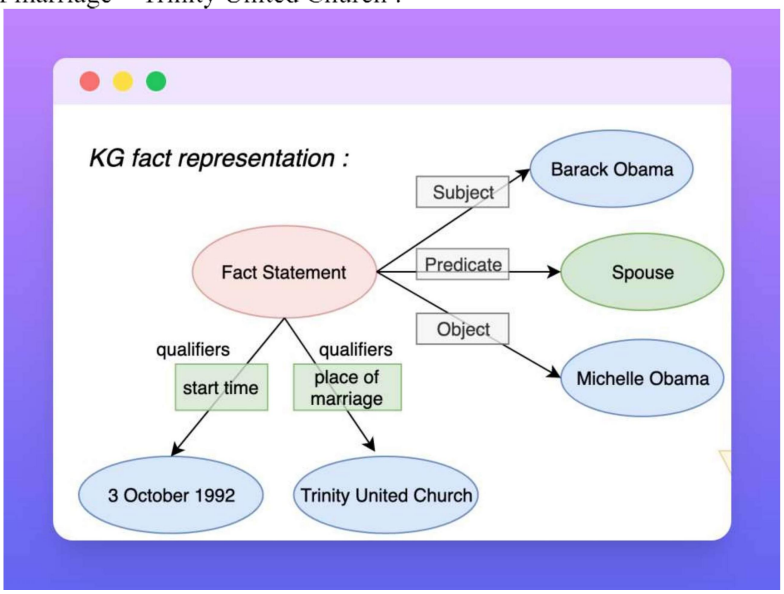
The knowledge base:

- 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 .



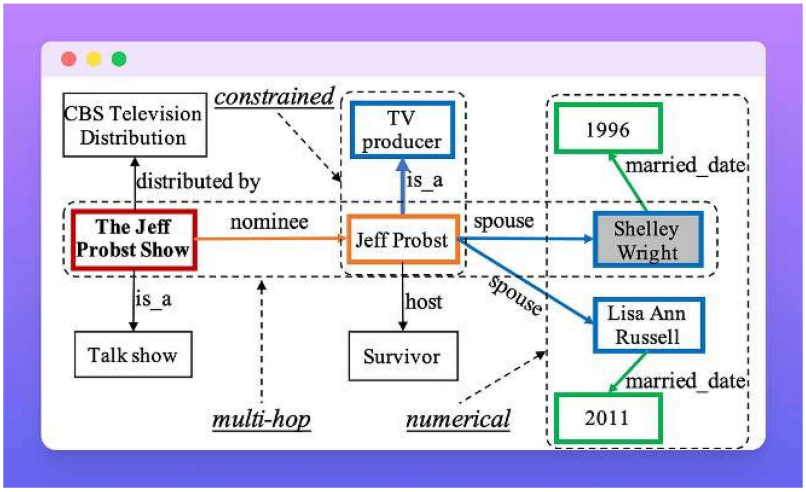
Representation of a fact with its qualifiers.

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:**

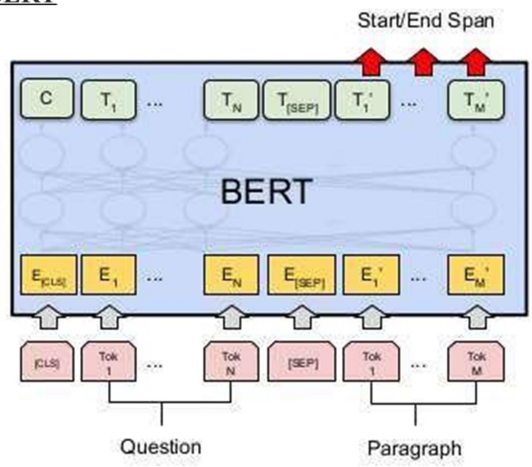
- BERT can be fine-tuned for various NLP tasks by adding task-specific layers on top of the BERT encoder.

- 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.*

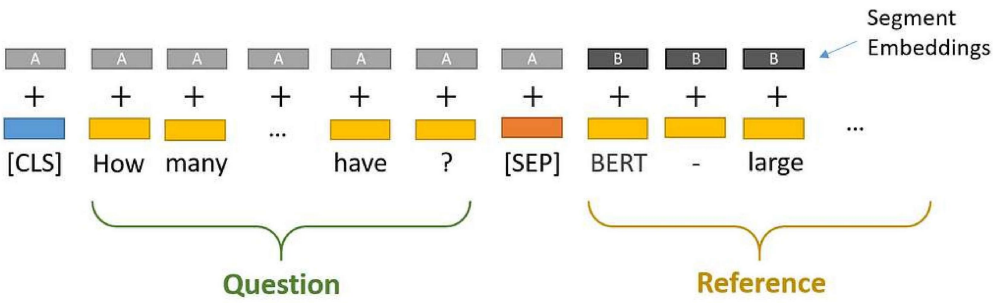
Question Answering System using BERT



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.



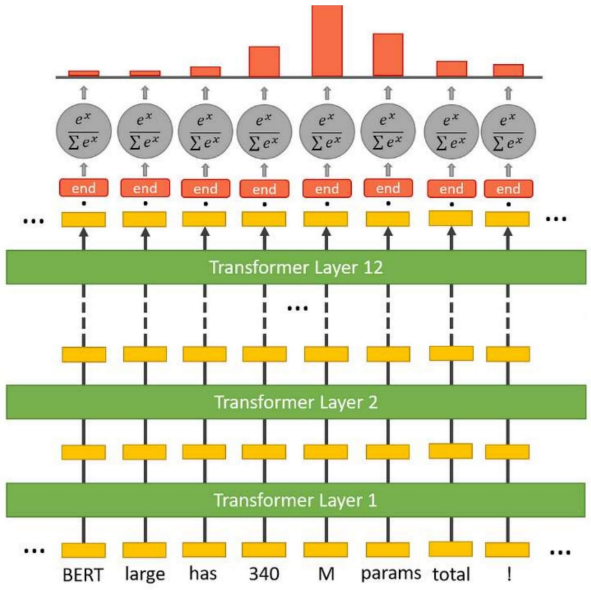
Question: How many parameters does BERT-large have?

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)

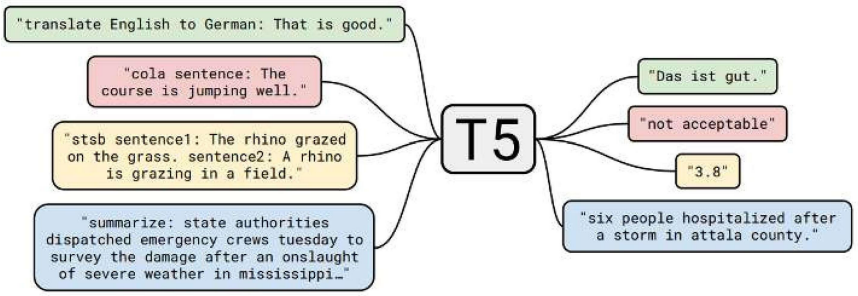
T5 (Text-to-Text Transfer Transformer) is a versatile and powerful natural language processing model developed by Google Research. T5 is designed to frame most NLP tasks as a text-to-text problem, where both the input and output are treated as text sequences. This approach allows T5 to handle a wide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