CSE 4392 Special Topics
Natural Language Processing

Information Retrieval

2024 Spring

Definition of information retrieval

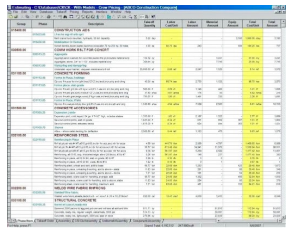
Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

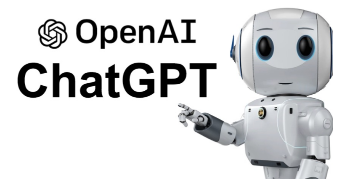
How good are the retrieved docs?

- *Precision* : Fraction of retrieved docs that are relevant to the user's information need
- *Recall* : Fraction of relevant docs in collection that are retrieved

  - More precise definitions and measurements to follow later

Evolution of Information Retrieval





Library Science

Before 1950's

Data Mining/
Info Retrieval

After 1950's

Web Search &
Mining

After 1995

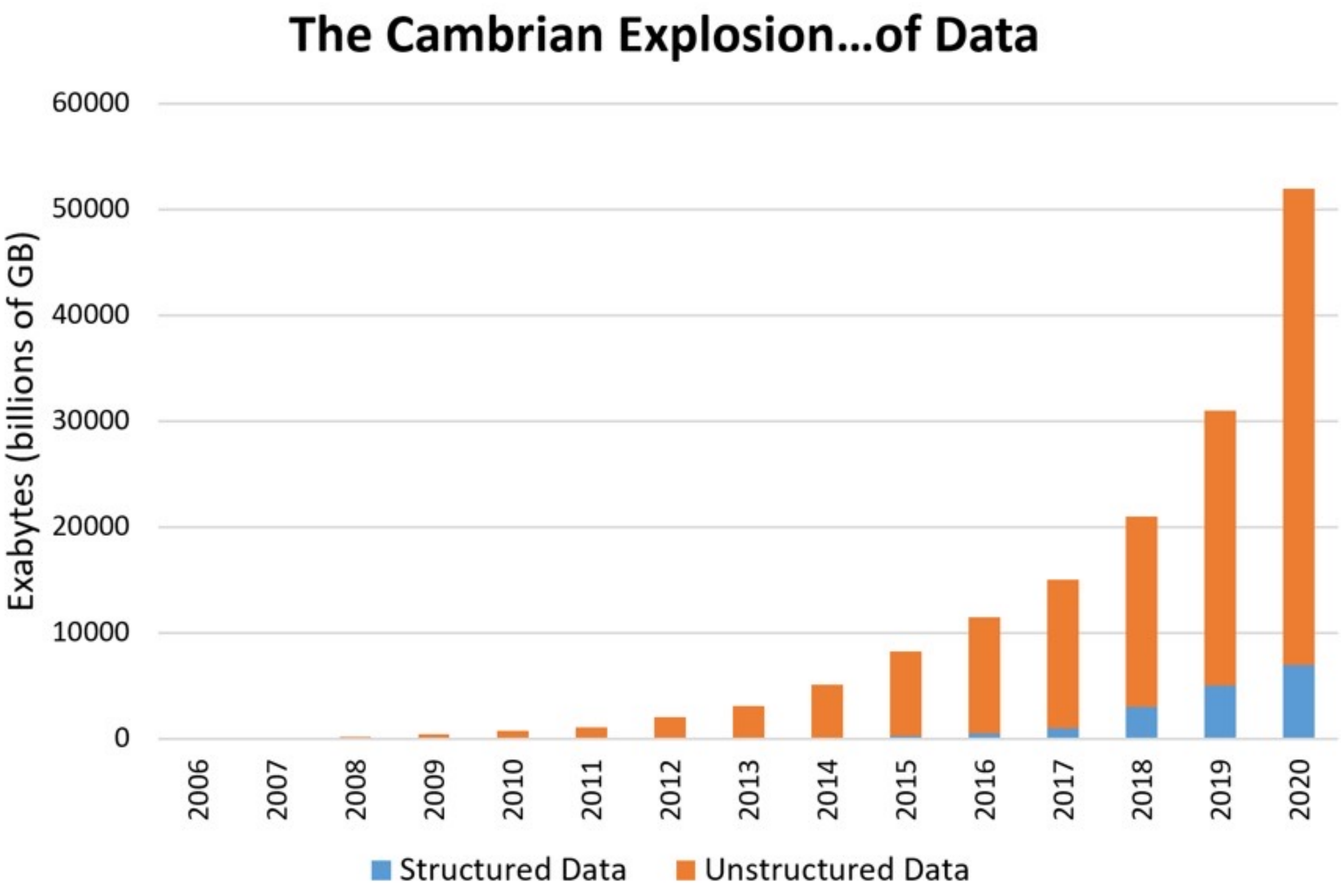
QA, Summary
& ChatBots

After 2015

Term &
Vector based

Large
Language
Model based

Web Data Exploded!



Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., Python AND Jobs
- The search engine returns all documents that satisfy the Boolean expression.

Unstructured data in 1650: Shakespeare

- Query: Which plays of Shakespeare contain the words BRUTUS and CAESAR, but not CALPURNIA?

Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	1	1	0	0	0	1
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0
CLEOPATRA	1	0	0	0	0	0
MERCY	1	0	1	1	1	1
WORSER	1	0	1	1	1	0
. . .						

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*. Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:
  - Take the vectors for BRUTUS, CAESAR AND NOT CALPURNIA
  - Complement the vector of CALPURNIA
  - Do a (bitwise) **and** on the three vectors
  - 110100 AND 110111 AND 101111 = 100100

0/1 vector for BRUTUS AND CAESAR AND NOT CALPURNIA:

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTHONY	1	1	0	0	0	1
BRUTUS	1	1	0	1	0	0
CAESAR	1	1	0	1	1	1
CALPURNIA	0	1	0	0	0	0
CLEOPATRA	1	0	0	0	0	0
MERCY	1	0	1	1	1	1
WORSER	1	0	1	1	1	0
. . .						
result:	1	0	0	1	0	0

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens
- $\Rightarrow$ total of 10^9 (1 billion) tokens
- On average 6 bytes per token, including spaces and punctuation
- $\Rightarrow$ size of document collection is about $6 \cdot 10^9 = 6$ GB
- Assume there are $M = 500{,}000$ distinct terms in the collection
- (Notice that we are making a term/token distinction.)

Can't build the incidence matrix

- $M = 500{,}000 \times 10^6$ = half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.
- What is a better representations?
  - We only record the 1s.

Inverted Index

For each term t, we store a list of all documents that contain t.

BRUTUS	$\longrightarrow$	1	2	4	11	31	45	173	174	
CAESAR	$\longrightarrow$	1	2	4	5	6	16	57	132	...
CALPURNIA	$\longrightarrow$	2	31	54	101					
⋮										

dictionary postings

INVERTED INDEX CONSTRUCTION

1. Collect the documents to be indexed:

   Friends, Romans, countrymen. | So let it be with Caesar | . . .

2. Tokenize the text, turning each document into a list of tokens:

   Friends | Romans | countrymen | So | . . .

3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: friend | roman | countryman | so | . . .

4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Generate posting

Doc 1. i did enact julius caesar i was killed i' the capitol brutus killed me
Doc 2. so let it be with caesar the noble brutus hath told you caesar was ambitious

$\Longrightarrow$

term	docID
i	1
did	1
enact	1
julius	1
caesar	1
i	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Sort postings

term	docID
i	1
did	1
enact	1
julius	1
caesar	1
i	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

$\Longrightarrow$

term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Create postings lists, determine document frequency

term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

$\Longrightarrow$

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate CALPURNIA in the dictionary
  4. Retrieve its postings list from the postings file
  5. Intersect the two postings lists
  6. Return intersection to user

Intersecting two posting lists

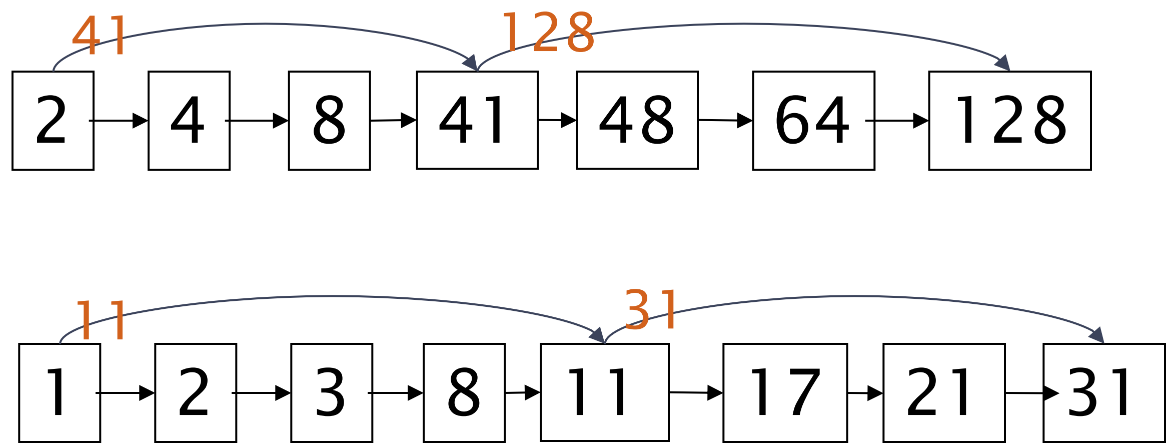
BRUTUS $\longrightarrow$ $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

CALPURNIA $\longrightarrow$ $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection $\Longrightarrow$ $\boxed{2} \rightarrow \boxed{31}$

- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

AUGMENT POSTINGS WITH SKIP POINTERS (AT INDEXING TIME)

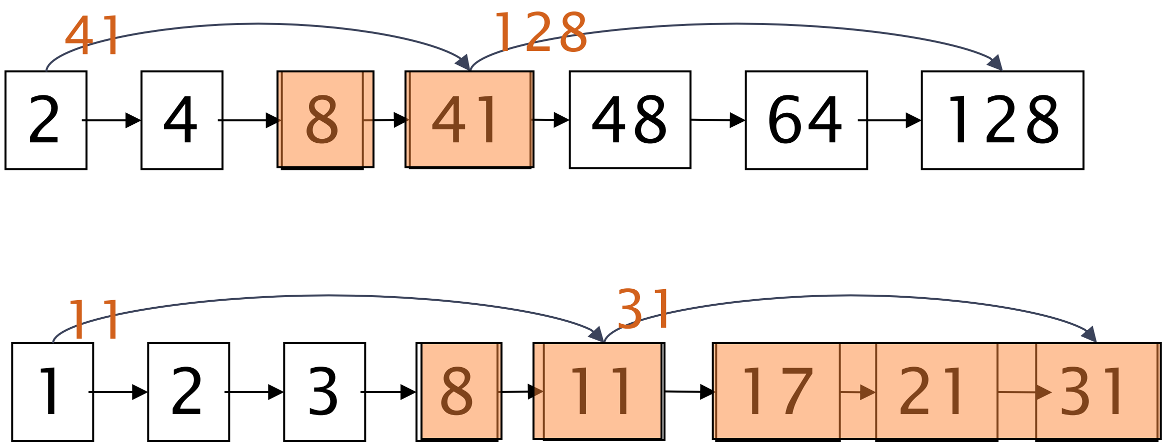


- Why?

  To skip postings that will not feature in the search results.

- How?
- Where do we place skip pointers?

Query processing with skip pointers



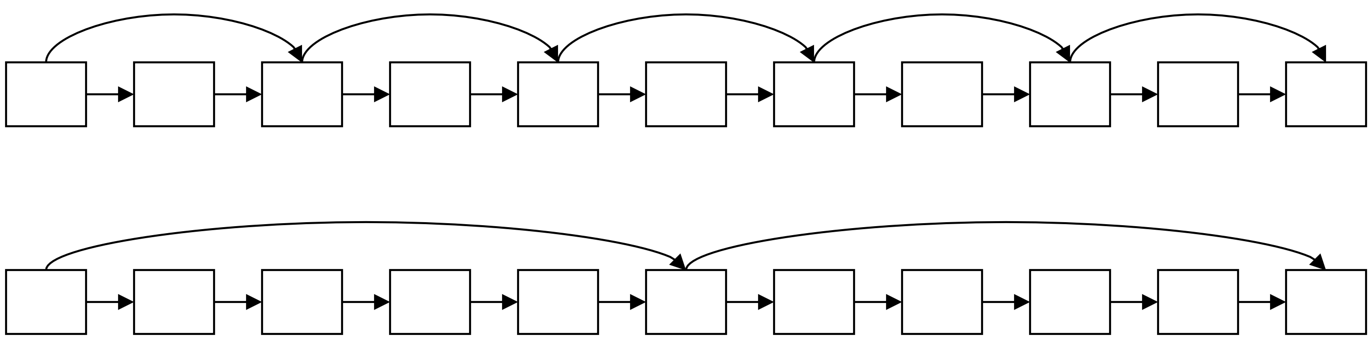
Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

We then have **41** and **11** on the lower. **11** is smaller.

But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

WHERE DO WE PLACE SKIPS?

- Tradeoff:
  - More skips $\rightarrow$ shorter skip spans $\Rightarrow$ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips $\rightarrow$ few pointer comparison, but then long skip spans $\Rightarrow$ few successful skips.



Where do we place skips? (cont)

- Simple heuristic: for postings list of length *P*, use $\sqrt{P}$ evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help a lot.
- With today's fast CPUs, they don't help that much anymore.

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and the collection.
  - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search.

FEAST OR FAMINE: NOT A PROBLEM IN RANKED RETRIEVAL

- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We just show the top k (≈ 10) results
  - We don't overwhelm the user
  - Premise: the ranking algorithm works
- Ranking by a score – say in [0, 1] – to each document
- This score measures how well document and query "**match**".

Google Result	Impressions	Percentage
1	2,834,806	34.35%
2	1,399,502	16.96%
3	942,706	11.42%
4	638,106	7.73%
5	[illegible]721	6.19%
6	416,887	5.05%
7	331,500	4.02%
8	286,118	3.47%
9	235,197	2.85%
10	223,320	2.71%
11	91,978	1.11%
12	69,778	0.85%
13	[illegible]	0.70%
14	46,822	0.57%
15	39,635	0.48%
16	32,168	0.39%
17	26,933	0.33%
18	23,131	0.28%
19	22,027	0.27%
20	23,953	0.29%

1st Page 94%

2nd Page 6%

QUERY-DOCUMENT MATCHING SCORES

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets A and B

  jaccard$(A,B) = |A \cap B| \;/\; |A \cup B|$

  jaccard$(A,A) = 1$

  jaccard$(A,B) = 0$ if $A \cap B = 0$

- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

Quiz: Jaccard coefficient

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?

Query: *ides of march*

Document 1: *caesar died in march*

Document 2: *the long march*

Take 2: Term Frequencies

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^v$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Term frequency tf

- The term frequency $\mathrm{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But ***not*** 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

NB: frequency = count in IR

LOG-FREQUENCY WEIGHTING

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

DOCUMENT FREQUENCY

- Rare terms are more informative than frequent terms
  - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*

→ We want a high weight for rare terms like *arachnocentric.*

IDF WEIGHT

- df_t is the <u>document</u> frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - $\mathrm{df}_t \leq N$ (total number of docs)
- We define the idf (inverse document frequency) of t by

$$\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$$

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

It turns out the base of the log is insignificant.

IDF EXAMPLE, SUPPOSE $N = 1$ MILLION

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

$$\mathrm{idf}_t = \log_{10}\left(N/\mathrm{df}_t\right)$$

There is one idf value for each term t in a collection.

QUIZ: IDF

- Why is the idf of a term *in a document* always finite?

$$\mathrm{idf}_t = \log_{10}\left(N/\mathrm{df}_t\right)$$

TF-IDF WEIGHTING

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$\mathrm{w}_{t,d} = (1+\log_{10} \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

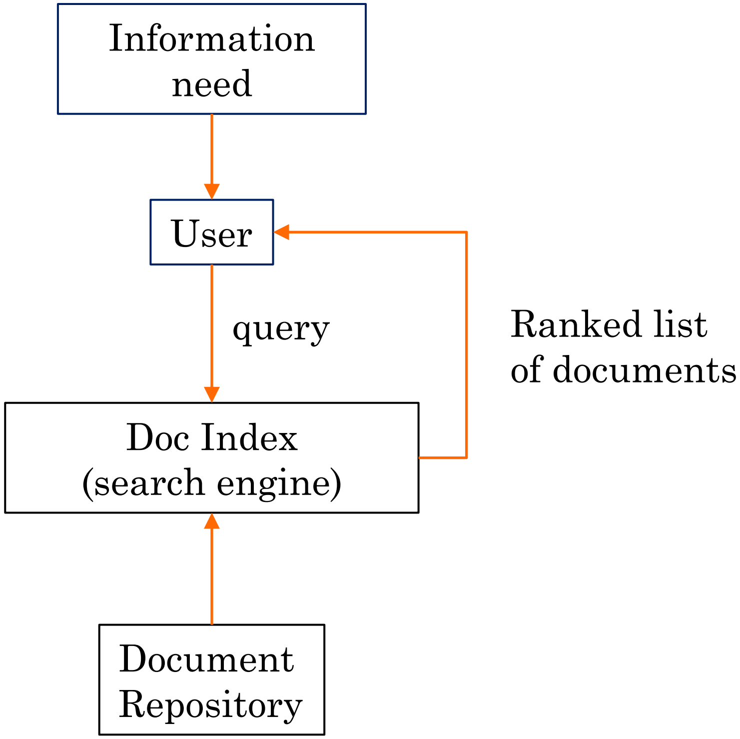
- Best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Score for a document given a query

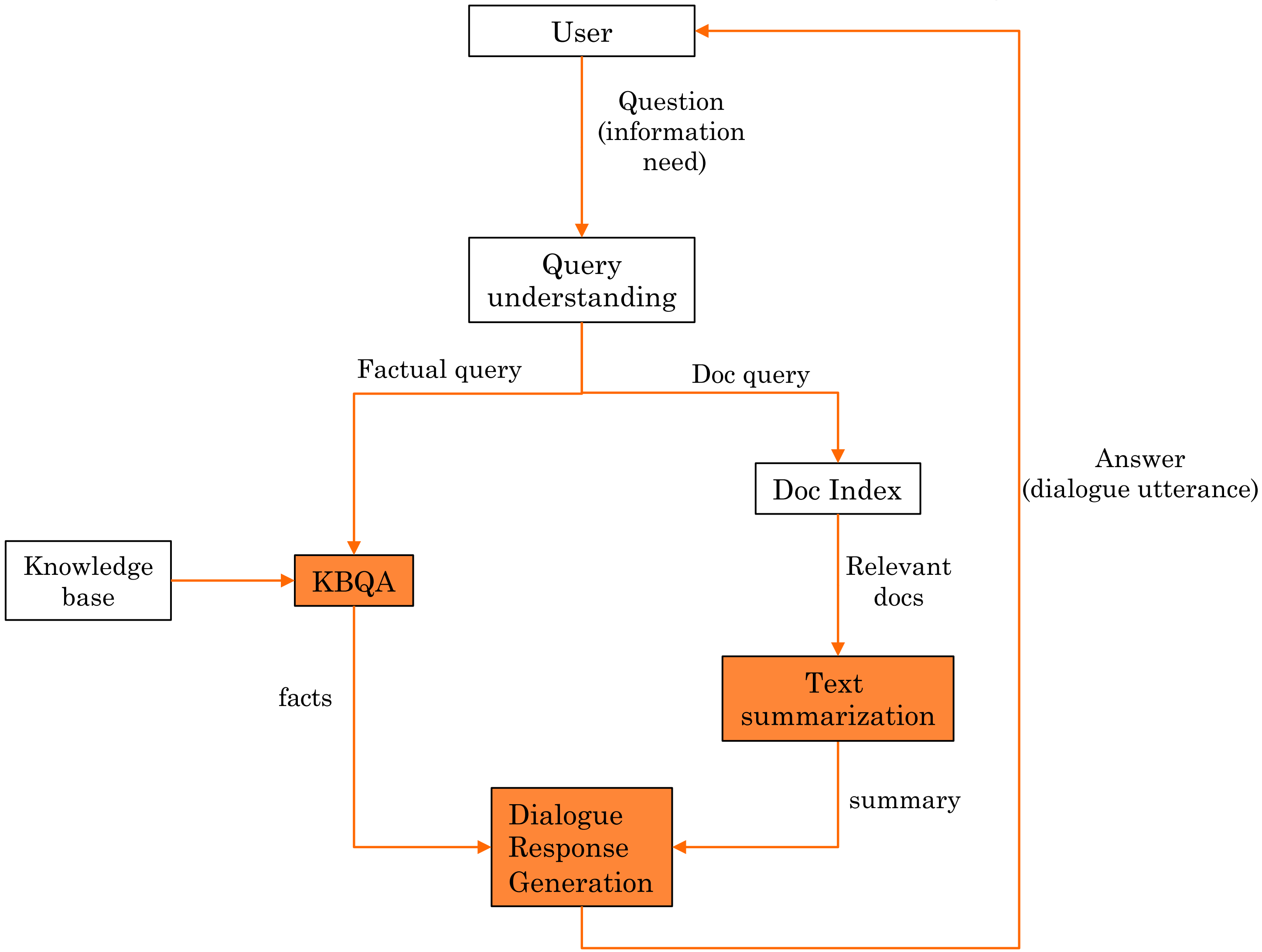
$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

- q is a multi-term query.
- There are many variants
  - How "tf" is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - …

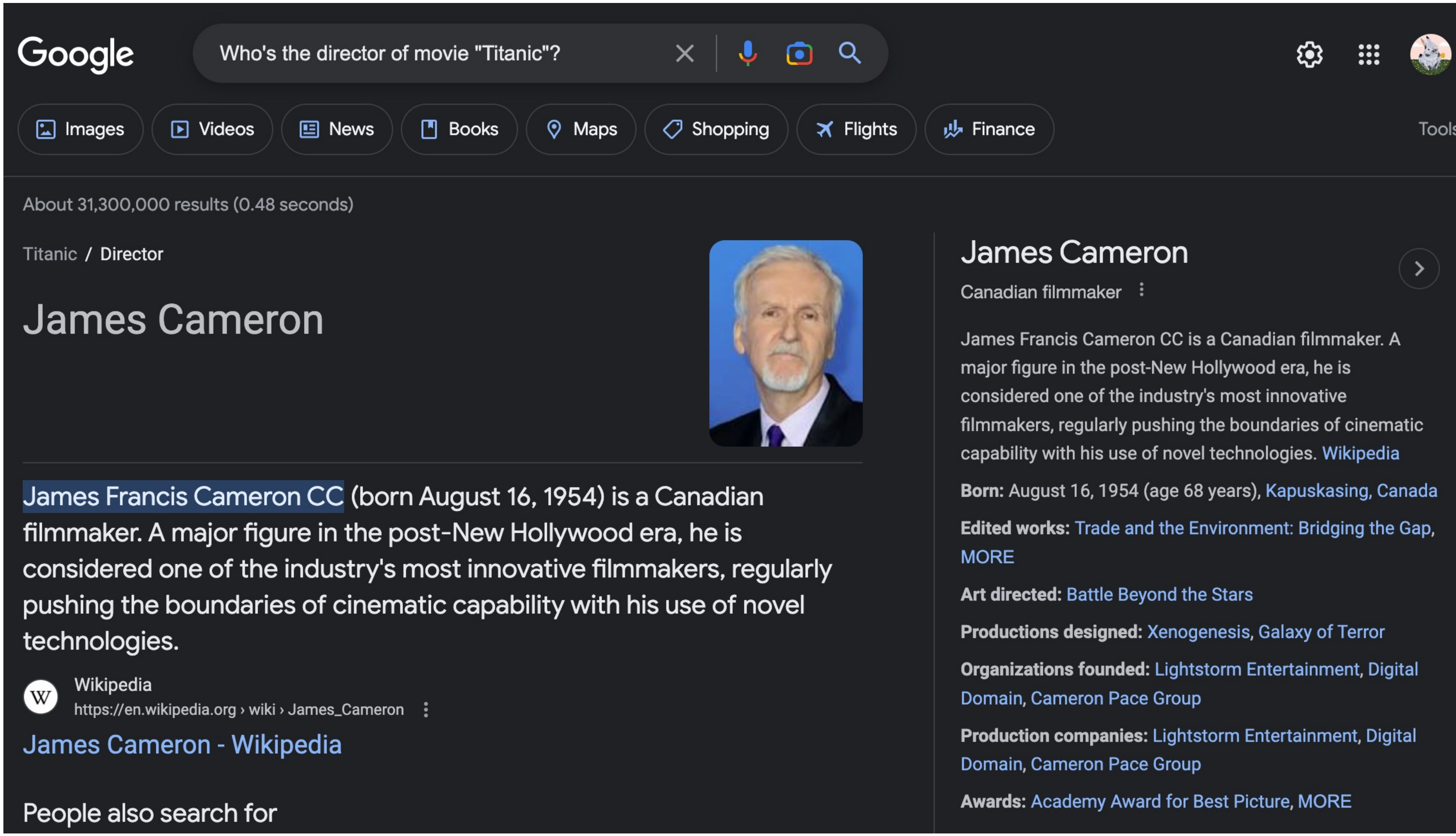
Classic IR Workflow



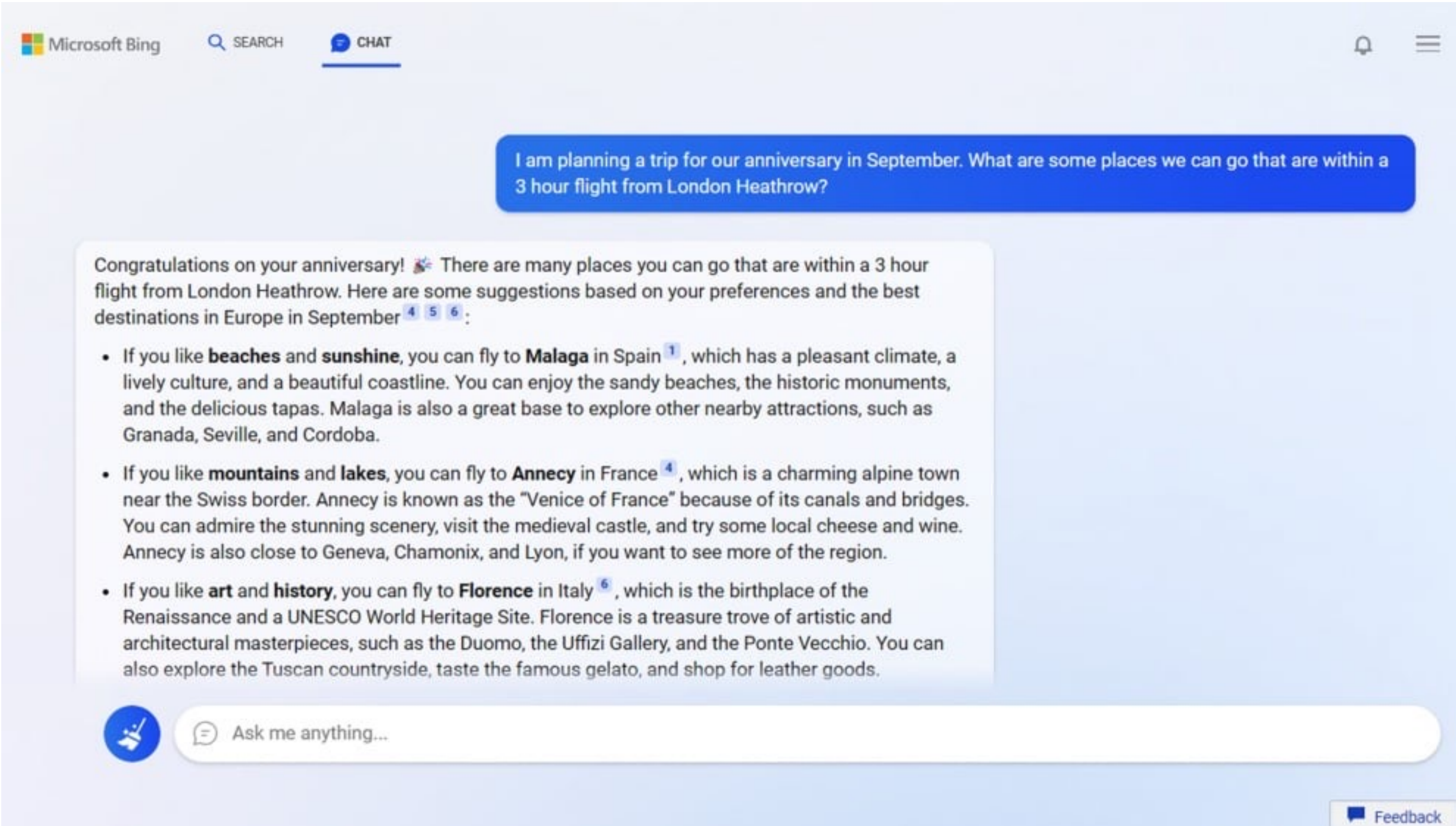
Next-gen IR (open-domain QA) Workflow



KBQA (Google Search)



Open-Domain QA (New Bing)



Text Summarization

- Text summarization is classified into two types — **Extractive** and **Abstractive Summarization**.
- **Extractive Summarization:** The extractive text summarization process extracts the main points of a text without any alteration to those points and rearranging the order of that points and the grammar to get the soul out of the summary.
- **Abstractive Summarization:** The Abstractive methods use advanced techniques to get a whole new summary. Some parts of this summary might not even appear within the original text.

Summarzation Dataset

- CNN/DailyMail (312k instances)

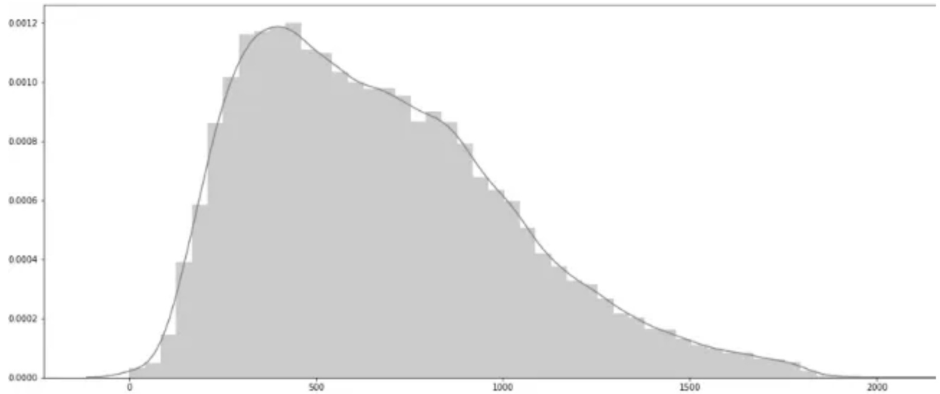
{'id': '0054d6d30dbcad772e20b22771153a2a9cbeaf62',
'article': '(CNN) -- An American woman died aboard a cruise ship that docked at Rio de Janeiro on Tuesday, the same ship on which 86 passengers previously fell ill, according to the state-run Brazilian news agency, Agencia Brasil. The American tourist died aboard the MS Veendam, owned by cruise operator Holland America. Federal Police told Agencia Brasil that forensic doctors were investigating her death. The ship's doctors told police that the woman was elderly and suffered from diabetes and hypertension, according the agency. The other passengers came down with diarrhea prior to her death during an earlier part of the trip, the ship's doctors said. The Veendam left New York 36 days ago for a South America tour.'
'highlights': 'The elderly woman suffered from diabetes and hypertension, ship's doctors say .\nPreviously, 86 passengers had fallen ill on the ship, Agencia Brasil says .'}

- Xsum (226k instances)

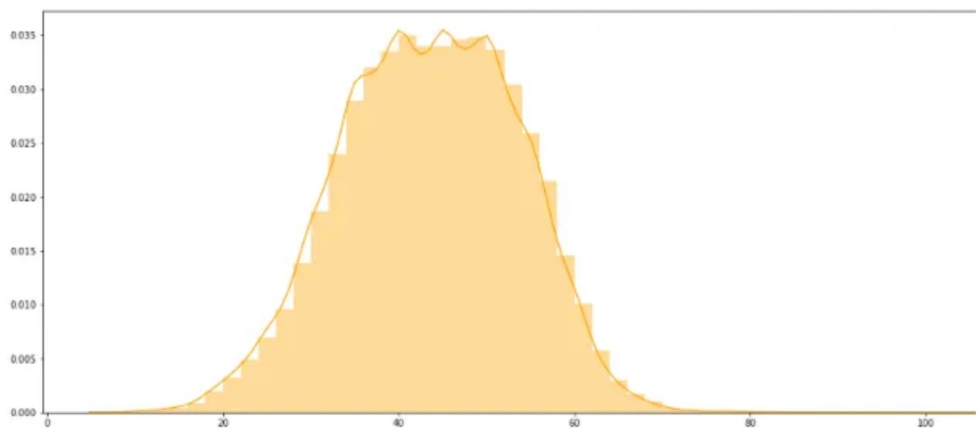
"Belgian cyclist Demoitie died after a collision with a motorbike during Belgium's Gent-Wevelgem race. The 25-year-old was hit by the motorbike after several riders came down in a crash as the race passed through northern France. "The main issues come when cars or motorbikes have to pass the peloton and pass riders," Team Sky's Rowe said. "That is the fundamental issue we're looking into. "There's a lot of motorbikes in and around the race whether it be cameras for TV, photographers or police motorbikes. "In total there's around 50 motorbikes that work on each race. "We've got a riders union and we're coming together to think of a few ideas, whether we cap a speed limit on how fast they can	"Welsh cyclist Luke Rowe says changes to the sport must be made following the death of Antoine Demoitie."	"35932467"

CNN/DM Dataset

- Article length:



- Summary length



Evaluation Metric

- **BLEU Score**
  - "bilingual evaluation understudy"
  - BLEU scores range from 0 and 1.
  - If predicted and original text is a similar score close to 1 and vice-versa.

- **ROUGE Score**
  - "Recall-Oriented Understudy for Gisting Evaluation"
  - ROUGE-1 refers to the overlap of unigram (each word) between the system and reference summaries.
  - ROUGE-2 refers to the overlap of bigrams between the system and reference summaries.
  - ROUGE-L: Longest Common Subsequence (LCS) based statistics. The longest common subsequence problem takes into account sentence-level structure similarity naturally and identifies the longest cooccurring in sequence n-grams automatically.

Evaluation Metrics

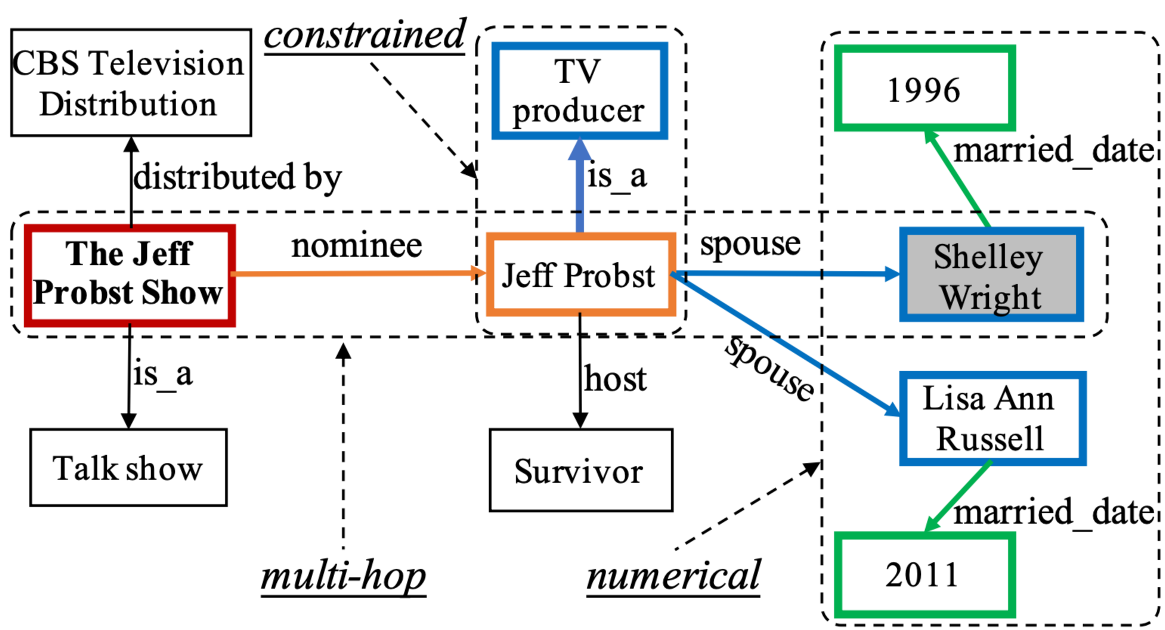
$$\text{ROUGE recall} = \frac{\text{number of overlapping words}}{\text{total words in reference summary}},$$

$$\text{ROUGE precision} = \frac{\text{number of overlapping words}}{\text{total words in system summary}},$$

- Normally we present ROUGE-F1 scores, which is calculated as we learned before.

Complex KBQA

- A knowledge base is a graph containing edges (*subject*, *relation*, *object*)
- A question such as:
  - "Who is the first wife of TV producer that was nominated for The Jeff Probst Show?"
  - Answer: Shelley Wright



Challenges of Complex KBQA

- Multi-hops
- Constrained relations
- Numerical operations
- Combinations of the above

Benchmark Datasets

Datasets	KB	Size	LF	NL
WebQuestions [Berant *et al.*, 2013]	Freebase	5,810	No	No
ComplexQuestions [Bao *et al.*, 2016]	Freebase	2,100	No	No
WebQuestionsSP [Yih *et al.*, 2016]	Freebase	4,737	Yes	Yes
ComplexWebQuestions [Talmor and Berant, 2018]	Freebase	34,689	Yes	Yes
QALD series [Lopez *et al.*, 2013]	DBpedia	-	Yes	Yes
LC-QuAD [Trivedi *et al.*, 2017]	DBpedia	5,000	Yes	Yes
LC-QuAD 2.0 [Dubey *et al.*, 2019]	DBpedia, Wikidata	30,000	Yes	Yes
MetaQA Vanilla [Zhang *et al.*, 2018]	WikiMovies	400k	No	No
CFQ [Keysers *et al.*, 2020]	Freebase	239,357	Yes	No
GrailQA [Gu *et al.*, 2020]	Freebase	64,331	Yes	Yes
KQA Pro [Shi *et al.*, 2020]	Wikidata	117,970	Yes	Yes

LF: Logical Forms NL: Rewrite LF in Natural Language

Evaluation Metrics

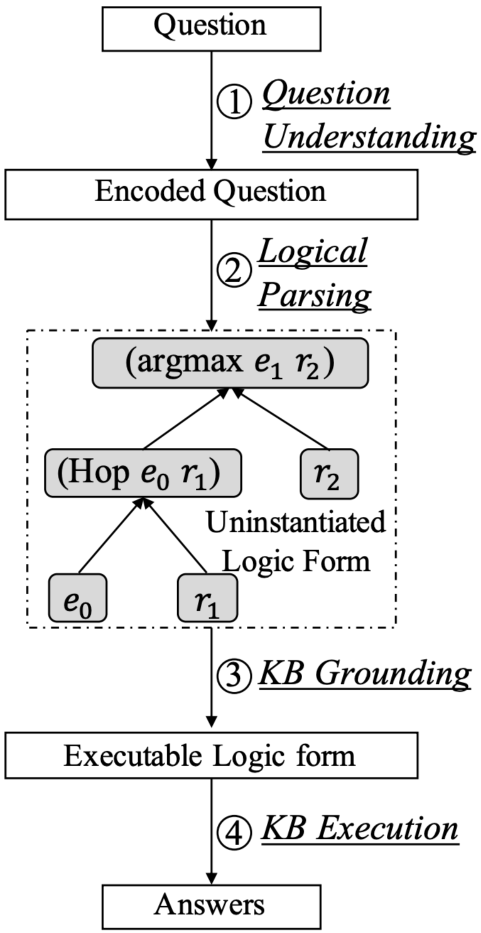
- Reliability
  - Precision, Recall and F1
  - Hits@1

$$\text{Hits@1} = \mathbb{I}(\tilde{a}_q \in \mathcal{A}_q),$$

where $\tilde{a}_q$ is the top 1 prediction in $\tilde{\mathcal{A}}_q$.

- Robustness
  - GrailQA dataset (Gu et al.)
  - three levels of generalization: *i.i.d.*, *compositional*, *zero-shot*
- System-user interaction

SEMANTIC PARSING APPROACH



SP 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.