Carnegie Mellon University
Language
Technologies
Institute

# 11-411 Natural Language Processing

## Information Retrieval and Question Answering

David R. Mortensen

August 25, 2022

Language Technologies Institute

# Information Retrieval

# Information Retrieval

The task of information retrieval (IR):

- Given
  - A collection of documents
  - A query
- Return
  - A ranking of the set of documents

# Overview of IR

Here is a simple approach to IR:

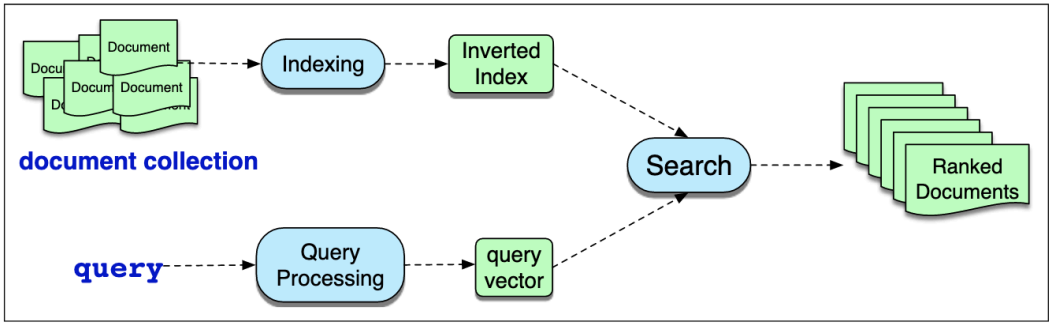



**Figure 23.1** The architecture of an ad hoc IR system.

# Documents Can Be Represented as Bags of Words

A BAG OF WORDS is a BAG or MULTISET of the words in a document.

- Like a set, except that identical elements can appear multiple times
- You could also think of it like a `Counter` object in Python

```
bow = {'and': 23502,
       'or': 12342',
       'the': 54939508,
       ...
       'hippopotamus': 1}
```

- If you take out sequencing information, any document can be viewed as a bag of words.

# Queries Can Also Be Represented as Bags of Words

- Natural language queries (like the query you type in when you're using a popular search engine like DuckDuckGo) are like little documents
- "What is the name of the current king of France?"

```
query = {'the': 2,
         'of', 2,
         'what': 1,
         'is': 1,
         'name', 1,
         'current': 1,
         'king': 1,
         'France': 1}
```

# Bags of Words Can Be Represented as Sparse Vectors

- So far, we've represented bags of words as dense MAPS, but sometimes it is useful to represent them as sparse vectors
- Led $V$ be the sent of all words in the document collection $D$. Then our BoW vectors for documents in $D$ will have $|V|$ dimensions.
- Each dimension corresponds to a word `type` and the value at this dimensions corresponds to the number of TOKENS of that type in the document that the vector is representing

# Computing the Similarity between BoW Vectors

- We represent documents and queries as vectors because it is easy to compute the similarity between them
- How do we compute the similarity between vectors?
  - DOT PRODUCT is the simplest similarity metric for vectors

$$a \cdot b = \sum_{i=1}^{n} a_i b_i \tag{1}$$

but it is sensitive to the magnitude of the vectors (how big the counts are) and we care about the directions only

  - COSINE SIMILARITY is a better metric for our purposes:

$$S_C(a,b) = \frac{a \cdot b}{\|a\|\|b\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{2}$$

It captures only the similarity in the **angle** or **direction** of the vector. But there's a problem...

# tf-idf Controls for Frequent but Uninformative Words

Some words are very common in a given document because they are common across all documents (e.g., *the*). They are not discriminative. `tf-idf` (product of term frequency and inverse document frequency) addresses this:

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t, d) + 1) \tag{3}$$

$$\text{idf}_f = \log_{10} \frac{N}{\text{df}_t} \tag{4}$$

$$\text{tf-idf}(t, d) = \text{tf}_{t,d} \cdot \text{idf}_t \tag{5}$$

When applied to computing the similarity between a query vector $\boldsymbol{q}$ and a document vector $\boldsymbol{d}$:

$$\text{score}(q, d) = \sum_{t \in q} \frac{\text{tf-idf}(t, q)}{\sqrt{\sum_{q_i \in q} \text{tf-idf}^2(\text{q}_\text{i}, \text{q})}} \cdot \frac{\text{tf-idf}(t, d)}{\sqrt{\sum_{d_i \in d} \text{tf-idf}^2(\text{d}_\text{i}, \text{d})}} \tag{6}$$

# Applying Simplifying Assuptions to tf-idf Scoring

Mathematically, `tf-idf` is rather ill-founded, but it works. We can make it even less well-founded but computationally cheaper (without hurting performance much) by simplifying the scoring formula:

$$\text{score}(q, d) = \sum_{t \in q} \frac{\text{tf-idf}(t, d)}{|d|} \tag{7}$$

This is what you would probably use in practice (when you are processing large numbers of documents).

# BM25 Is More Powerful than tf-idf

However, there is a solution to the same problem that is far more powerful: BM25.

$$\text{BM25}(d, q) = \sum_{t \in q} log\left(\frac{N}{\text{df}_t}\right) \frac{\text{tf}_{t,d}}{k(1 - b + b(\frac{|d|}{avgdl})) + \text{tf}_{t,d}} \tag{8}$$

Two hyperparameters:

- $k$ controls the weight of term frequency
- $0 \leq b \leq 1$ controls scaling by document length ($b = 0 \Rightarrow$ no length scaling)

# Evaluating IR Systems with Precision and Recall

NLP works because of metrics that allow us to compare the performance of different systems.

Three of the most important metrics are Precision, Recall, and (their harmonic mean) F1:

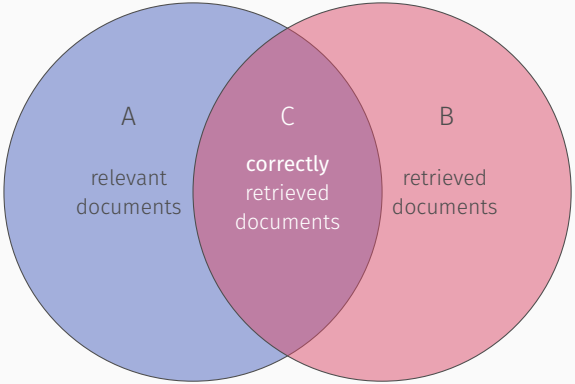
$$\text{precision} = \frac{TP}{TP + FP} \tag{9}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{10}$$

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{11}$$

where $TP$ = true positives, $FP$ = false positives, and $FN$ = false negatives

# Precision and Recall Illustrated




A Venn diagram can simplify precision and recall considerably:

- $A$ documents that are relevant to the query
- $B$ documents that are retrieved
- $C$ is the intersection between these two sets

$$\text{precison} = \frac{|C|}{|B|} \tag{12}$$

$$\text{recall} = \frac{|C|}{|A|} \tag{13}$$

Note that we are assuming that the IR system retrieves a set of documents rather than imposing a ranking over the documents in the collection

# Mean Average Precision (MAP) Can Be Used to Evaluate Ranked Retrieval

For ranked retrieval, we need a different metric. First, consider average precision (*AP*):

$$AP = \frac{1}{|R_r|} \sum_{d \in R_r} \text{precision}_r(d) \tag{14}$$

Where $R_r$ is the set of relevant documents at or above $r$. We can then compute Mean Average Precision (*MAP*):

$$MAP = \frac{1}{|Q|} \sum_{q \in Q} AP(q) \tag{15}$$

where $Q$ is the set of queries. This allows us to evaluate our rankings of the documents, not just whether the documents we have retrieved are relevant, by rewarding us for having more relevant documents at the top of the ranking

# IR with Dense Vectors

Sparse vectors may have been fine in 1980, but we can do better now **by representing queries and documents as dense vectors, using self-supervised contextual language models like BERT** (You will learn more about BERT in Module 4)

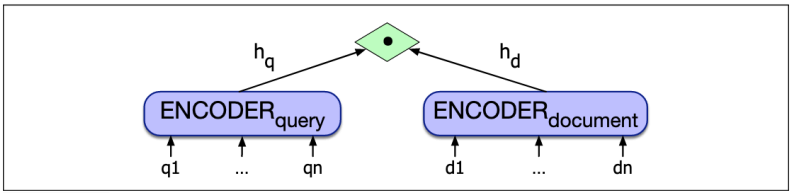



**Figure 23.8** BERT bi-encoder for computing relevance of a document to a query.

This solves the **vocabulary mismatch problem**: queries and documents about the same topic sometimes use different, synonymous words and words may have different meanings depending on their context. **More about this in a moment**.

# Question Answering

# The Task of Question Answering

- Given:
  - A collection of documents
  - A question from a user
- Return:
  - A short passage containing the answer, or
  - A set of short passages ranked according to how likely they are to contain the answer

| Question | Answer |
|---|---|
| What is the second largest city in Pennsylvania? | Pittsburgh |
| After whom is Pittsburgh named? | Sir William Pitt |
| What is the highest-ranked university in Pittsburgh? | Carnegie Mellon University |
| When was Fort Pitt founded? | November 27, 1758 |

# Question Answering Can Be Seen as an IR Problem

Given a question and a corpus of documents

- Retrieve a set of documents relevant to the question
- Retrieve the set of passages from these documents relevant to the question
- Return answers extracted from these passages

# Traditional IR-based QA Pipeline

This is how we did it in the old days:

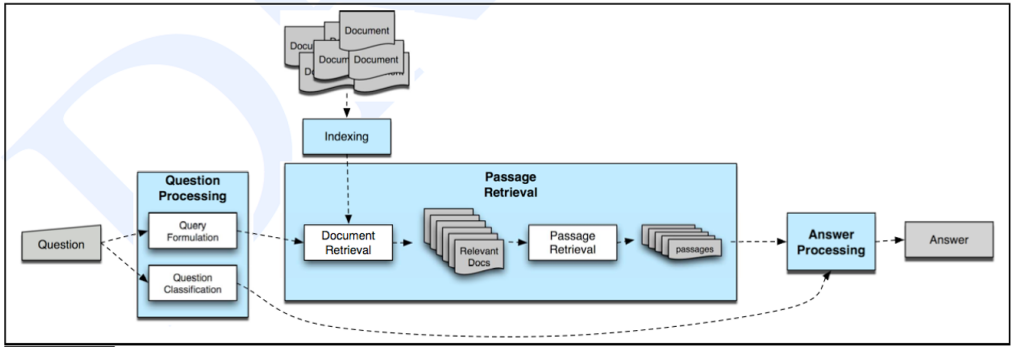


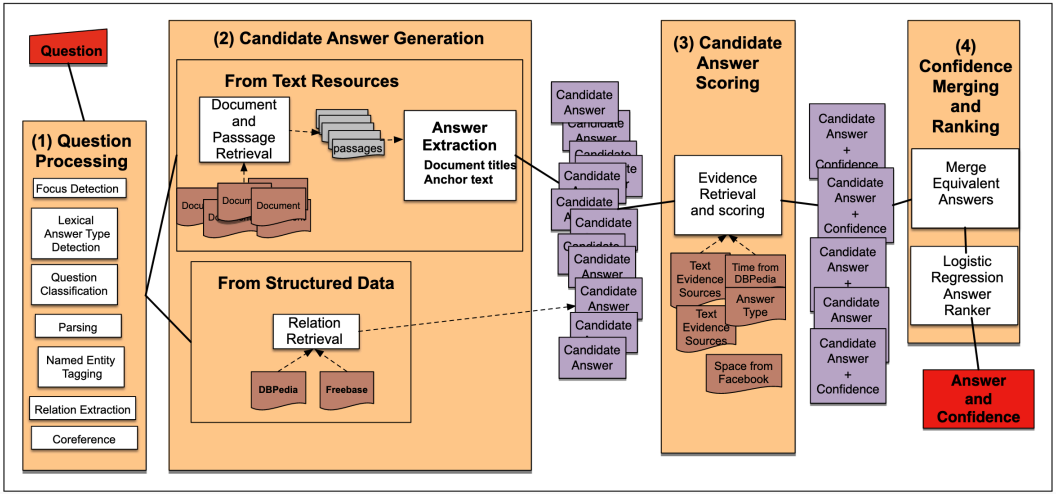

**Figure 23.8** The 3 stages of a generic question answering system: question processing, passage retrieval, and answer processing..

# Components of Traditional QA Pipeline (Baseline)

- Question Processing
  - Classify question according to type (for answer processing)
  - Convert question to vector
- Indexing
  - Represent documents as indexed vectors
- Passage Retrieval
  - Retrieve documents based on similarity to question (e.g. tf-idf)
  - Segment retrieved documents into passages
  - Rank passages according to similarity to question vector
- Answer Processing
  - Extract answers from passages, using question type information

**This is your primary baseline.** You must improve on this approach in some way.

# Watson QA


Question
(1) Question Processing
Focus Detection
Lexical Answer Type Detection
Question Classification
Parsing
Named Entity Tagging
Relation Extraction
Coreference
(2) Candidate Answer Generation
From Text Resources
Document and Passsage Retrieval
passages
Answer Extraction
Document titles
Anchor text
From Structured Data
Relation Retrieval
DBPedia
Freebase
Candidate Answer
(3) Candidate Answer Scoring
Evidence Retrieval and scoring
Text Evidence Sources
Time from DBPedia
Answer Type
Text Evidence Sources
Space from Facebook
Candidate Answer + Confidence
(4) Confidence Merging and Ranking
Merge Equivalent Answers
Logistic Regression Answer Ranker
Answer and Confidence

# IR-Based QA with Dense Vectors

BERT-like models make much simpler pipelines possible:

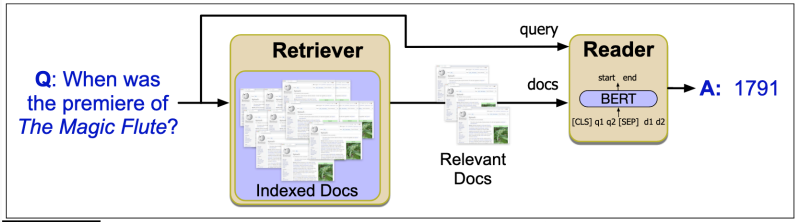



**Figure 23.10** IR-based factoid question answering has two stages: **retrieval**, which returns relevant documents from the collection, and **reading**, in which a neural reading comprehension system extracts answer spans.

- A BERT-based retriever (see dense vector IR above) retrieves and ranks the relevant documents
- A BERT-based "reader" extracts the answers from the documents

# BERT-Based Reader for QA

In extractive QA, the model identifies a span of a passage that corresponds to the answer. In modern QA systems, this is done with a Reader:

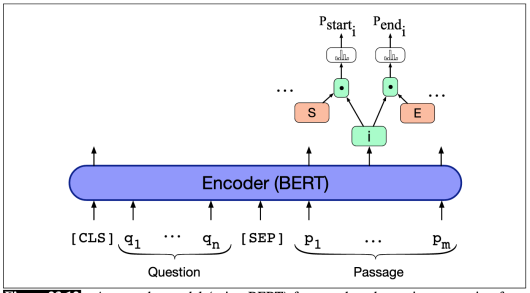



**Figure 23.12** An encoder model (using BERT) for span-based question answering from reading-comprehension-based question answering tasks.

It consists of an encoder (e.g., BERT) with a linear layer added. The linear layer has been fine-tuned to identify the start and end tokens of an answer span.

# SQuAD

- Stanford Question Answering Dataset
- Questions from Wikipedia
- Answers are spans of Wikipedia articles
- In SQuAD 2.0, some questions are unanswerable
- About 150,000 questions

# HotpotQA

- HotpotQA
- Questions are more difficult to answer than SQuAD
- Crowd workers were shown multiple context documents
- Asked to write question-answer pairs that required an understanding of all of the documents
- Suffers from SQuAD problem: annotators are likely to use words from the documents in the questions

# TriviaQA

- TriviaQA tries to address this problem by using questions that were not developed with the provided documents in mind
- Questions were developed by trivia enthusiasts
- Supporting documents were identified after the fact

# Natural Questions Dataset

- Natural Questions
- Based on queries to the Google search engine
- Annotators are presented with a query and a Wikipedia page from the top search results
- They provide a paragraph-length answer and a short answer

# TyDi QA

- TyDi QA
- The aforementioned datasets are all in English
- TyDi QA has 204,000 questions from a diverse set of languages:
  - Arabic
  - Bengali
  - Kiswahili
  - Russian
  - Thai
  - etc.

# Entity Linking

ENTITY LINKING is the task of associating mentions in text with the representations of real-world entities in an ontology. This ontology is often Wikipedia, in which case the term WIKIFICATION is sometimes used.

You will hear more about entity linking in Module 9

# Knowledge-Based QA Is Useful for Structured Data

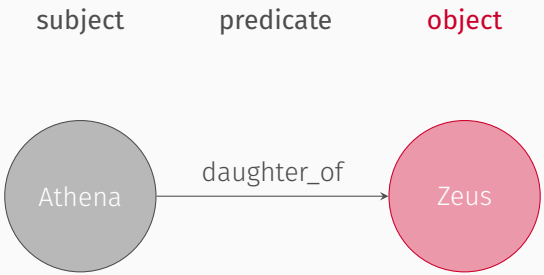
- For queries purely over natural language data, the IR-based methods described so far work well
- However, what if one also, or exclusively, wants to answer questions about structured databases (e.g., tabular data or knowledge graphs)?
- For these cases, knowledge-based QA is appropriate

# Knowledge-Based QA Is not Extractive

- In IR-based QA, one typically tries to identify one or more spans in documents that contain answers
- It is **extractive**
- In KB-QA, answers are derived from structured knowledge (relational databases, triple stores, other kinds of graphs) and converted to natural language
- It is **abstractive** in a sense

# What Is an RDF Triple?

- RDF triples consist of two entities and a relation that holds between them
- One such relation might be `daughter_of`
- Pairs that fall into such a relation:
  - ⟨Athena, Zeus⟩
  - ⟨Leia, Anakin⟩
  - ⟨Liza Minnelli, Judy Garland⟩
- The relation is stated as a PREDICATE
- The first entity is called the SUBJECT and the second entity is called the OBJECT
- A set of such triples forms a graph



# Extracting RDF Triples

RDF triples can easily be extracted from structured data:

- Wikidata
- Freebase
- Relational databases

They can also be extracted from text data using RELATION EXTRACTION, about which more will be said in the semantics module

# Using Triples to Answer Questions

Triples are useless unless their entities can be linked to the entities in questions. Thus, ENTITY LINKING is very important to KB-QA. Take the following question:

*Who is the father of Athena?*

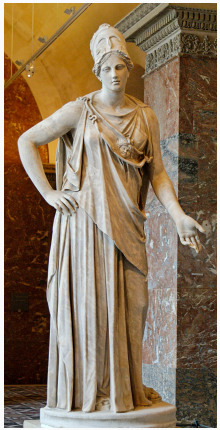
To which Athena, in the knowledge graph/triple store, does “Athena” refer?

- Athena Chu?
- Athena Gana?
- Athena Faris?

Knowing this, the question can be converted into

```
daughter_of(Athena, ?x)
```

Where `?x` is a variable. This query can be made to the database.



# Relation Detection

But how do you get from

*Who is the father of Athena?*

to

```
daughter_of(Athena, ?x)
```

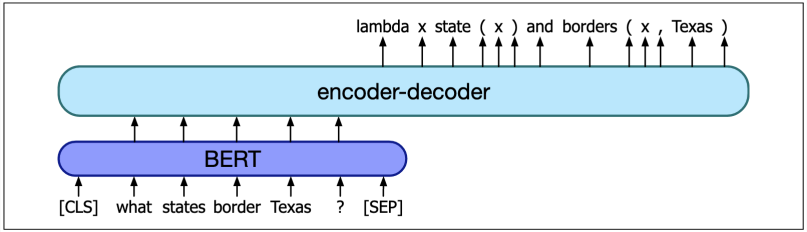
An important part of this is RELATION DETECTION. One common approach:

- You have an encoding of each possible relation (perhaps an embedding from a neural language model)
- You have an encoding of the question
- You compare this encodings, using the dot product, to find the best match

# Semantic Parsing and Question Answering

Another kind of KB-QA: use a SEMANTIC PARSER (Module 8) that takes a question and returns a logical form:

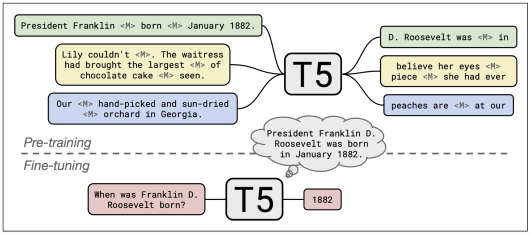
- First order logic (Module 8)
- SQL
- SPARQL




This logical form can then be used to query a structured database.

# Second Baseline: Using LMs to do QA

- Large language models like T5, BERT, or GPT-3 store the answers to many questions in their parameters
- It is easy to build a QA system based solely upon this stored information
- T5, for example, is pre-trained to “fill in” deleted tokes (indicated by <M>
- It can be fine-tuned to return an answer, given a question
- This type of model can be used as a baseline for your project but cannot be the basis of your QA system



# Mean Reciprocal Rank is a Metric for QA

The most common metric for evaluating QA is Mean Reciprocal Rank, or MRR. It assumes that QA systems output a series of questions ranked according to confidence.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{16}$$

where $Q$ is the set of questions in the test set and $rank$ is the first correct answer to the $i$th question in $Q$.

MRR is the sum of the reciprocal of the ranks of the first correct answers normalized by the number of questions in the test set.

# Metrics for Reading Comprehension

- **Exact match** For each answer, determine whether the HYPOTHESIZED answer is string-identical to the REFERENCE answer. This is a coarse-grained metric.
- **$F_1$** For each answer, identify the words that are present in the hypothesis, the words that are present in the reference, and the overlap between the two. Use these values to compute precision, recall, and $F_1$. There are various ways of averaging the results, which we will discuss at greater length in Module 3. This is a fine-grained metric.

# Questions?