Lecture 26.

| | |
|---|---|
| Question/Answering | Similarity Measures, Cosine similarity, Jaccard similarity |

A **Question Answering (QnA) system** is an information retrieval system that aims to find precise answers to natural language questions posed by users, rather than just returning a list of relevant documents. Instead of searching for keywords in documents, a QnA system tries to understand the intent of the question and extract or generate the most appropriate answer from a knowledge base.

One of the simplest methods to build a QnA system, especially for tasks involving finding the *most similar document* to a query, is by using **Jaccard Similarity**.

## What is Jaccard Similarity?

The **Jaccard Similarity Coefficient** (also known as the Jaccard Index or Jaccard coefficient) is a statistic used for gauging the similarity and diversity of sample sets. It measures the similarity between two finite sets, and is defined as the size of the intersection divided by the size of the union of the sample sets.

**Formula:**

Given two sets, A and B, the Jaccard Similarity is calculated as:

$J(A,B)=|A \cup B|\,|A \cap B|$

Where:

- $|A \cap B|$ is the number of elements common to both sets (the size of their intersection).
- $|A \cup B|$ is the total number of unique elements in both sets combined (the size of their union).

The Jaccard Similarity ranges from 0 to 1, where:

- **1** means the two sets are identical.
- **0** means the two sets have no elements in common.

**Simple Example:**

Let Set A = {apple, banana, orange} Let Set B = {banana, grape, orange}

- Intersection ($A \cap B$): {banana, orange} $\Rightarrow |A \cap B|=2$
- Union ($A \cup B$): {apple, banana, orange, grape} $\Rightarrow |A \cup B|=4$

Jaccard Similarity J(A,B)=42=0.5

## QnA System using Jaccard Similarity: Detailed Explanation

When applying Jaccard Similarity to a QnA system, the core idea is to treat the user's question and each potential answer (or the document containing the answer) as a "set of words" (or tokens). The system then calculates the Jaccard Similarity between the question's word set and the word set of each candidate answer/document. The answer/document with the highest Jaccard Similarity score is considered the most relevant.

This approach is primarily a **retrieval-based QnA system**, meaning it finds the most relevant pre-existing answer rather than generating a new one.

**Steps Involved:**

1. **Knowledge Base Preparation (Indexing):**
    - **Documents/Answers:** The QnA system needs a collection of documents or pre-defined answers.
    - **Preprocessing:** Each document/answer in the knowledge base undergoes preprocessing:
        - **Tokenization:** Breaking down text into individual words or meaningful units (tokens).
        - **Normalization:** Converting all text to lowercase, removing punctuation, and potentially removing stopwords (common words like "the", "a", "is" that don't carry much meaning for similarity). Stemming or lemmatization (reducing words to their root form) can also be applied.
        - **Set Creation:** Each preprocessed document/answer is converted into a set of unique tokens.
2. **Query Processing:**
    - **User Question:** When a user asks a question, it undergoes the exact same preprocessing steps (tokenization, normalization, set creation) as the documents in the knowledge base. This ensures consistency.
3. **Similarity Calculation:**
    - The Jaccard Similarity is calculated between the set of tokens from the user's question and the set of tokens from *each* document/answer in the knowledge base.
4. **Ranking and Retrieval:**
    - All calculated Jaccard Similarity scores are ranked in descending order.
    - The document/answer with the highest Jaccard Similarity score is retrieved and presented as the answer to the user's question. A threshold might be set, below which no answer is returned.

## Detailed Example

Let's assume our QnA system has a small knowledge base of three documents, and a user asks a question.

**Knowledge Base Documents:**

- **Doc 1:** "The capital of France is Paris."
- **Doc 2:** "Paris is a beautiful city."
- **Doc 3:** "The Eiffel Tower is in Paris, France."

**User Question (Query):**

- "What is the capital of France?"

**Step-by-Step Calculation:**

**1. Preprocessing (Tokenization, Lowercasing, Removing Punctuation, Removing Stopwords)**

Let's define our stopwords: `{'the', 'is', 'a', 'of', 'what', 'in'}`

- **Doc 1:** "The capital of France is Paris."
  - Tokens: `['the', 'capital', 'of', 'france', 'is', 'paris']`
  - Lowercase & Remove Punctuation: `['the', 'capital', 'of', 'france', 'is', 'paris']`
  - Remove Stopwords: `['capital', 'france', 'paris']`
  - **Set D1:** `{'capital', 'france', 'paris'}`
- **Doc 2:** "Paris is a beautiful city."
  - Tokens: `['paris', 'is', 'a', 'beautiful', 'city']`
  - Lowercase & Remove Punctuation: `['paris', 'is', 'a', 'beautiful', 'city']`
  - Remove Stopwords: `['paris', 'beautiful', 'city']`
  - **Set D2:** `{'paris', 'beautiful', 'city'}`
- **Doc 3:** "The Eiffel Tower is in Paris, France."
  - Tokens: `['the', 'eiffel', 'tower', 'is', 'in', 'paris', 'france']`
  - Lowercase & Remove Punctuation: `['the', 'eiffel', 'tower', 'is', 'in', 'paris', 'france']`
  - Remove Stopwords: `['eiffel', 'tower', 'paris', 'france']`
  - **Set D3:** `{'eiffel', 'tower', 'paris', 'france'}`
- **User Question (Query Q):** "What is the capital of France?"
  - Tokens: `['what', 'is', 'the', 'capital', 'of', 'france']`
  - Lowercase & Remove Punctuation: `['what', 'is', 'the', 'capital', 'of', 'france']`

    - Remove Stopwords: `['capital', 'france']`
    - **Set Q:** `{'capital', 'france'}`

**2. Similarity Calculation (Jaccard Similarity between Query Set Q and each Document Set)**

- **Query Q vs. Doc 1 (Set D1):**
    - Set Q: `{'capital', 'france'}`
    - Set D1: `{'capital', 'france', 'paris'}`
    - Intersection (Q∩D1): `{'capital', 'france'}`
        - ∣Q∩D1∣=2
    - Union (Q∪D1): `{'capital', 'france', 'paris'}`
        - ∣Q∪D1∣=3
    - **Jaccard Similarity** J(Q,D1)=32≈0.67
- **Query Q vs. Doc 2 (Set D2):**
    - Set Q: `{'capital', 'france'}`
    - Set D2: `{'paris', 'beautiful', 'city'}`
    - Intersection (Q∩D2): `{}` (empty set)
        - ∣Q∩D2∣=0
    - Union (Q∪D2): `{'capital', 'france', 'paris', 'beautiful', 'city'}`
        - ∣Q∪D2∣=5
    - **Jaccard Similarity** J(Q,D2)=50=0.0
- **Query Q vs. Doc 3 (Set D3):**
    - Set Q: `{'capital', 'france'}`
    - Set D3: `{'eiffel', 'tower', 'paris', 'france'}`
    - Intersection (Q∩D3): `{'france'}`
        - ∣Q∩D3∣=1
    - Union (Q∪D3): `{'capital', 'france', 'eiffel', 'tower', 'paris'}`
        - ∣Q∪D3∣=5
    - **Jaccard Similarity** J(Q,D3)=51=0.2

**3. Ranking and Retrieval:**

Based on the calculated Jaccard Similarities:

- Doc 1: 0.67
- Doc 2: 0.0
- Doc 3: 0.2

**Ranking (Highest to Lowest):**

1. Doc 1 (0.67)
2. Doc 3 (0.2)
3. Doc 2 (0.0)

**Result:** The system would retrieve **Doc 1: "The capital of France is Paris."** as the most relevant answer to the question "What is the capital of France?".

## Limitations of Jaccard Similarity in QnA Systems

While simple to implement, Jaccard Similarity has significant limitations for robust QnA:

1. **No Semantic Understanding:** It only considers word overlap. It doesn't understand synonyms (e.g., "car" vs. "automobile"), word embeddings, or the context/meaning of words.
2. **Order Insensitivity:** It treats text as a "bag of words," ignoring word order, which is crucial for understanding questions and answers (e.g., "dog bites man" vs. "man bites dog").
3. **Stopword Sensitivity (even with removal):** While stopwords are often removed, the choice of stopwords can impact results. Important words might be removed, or less important words might remain.
4. **Length Bias:** Longer documents tend to have more unique words, potentially diluting the similarity score if the relevant part is small.
5. **Lack of Term Weighting:** All words are treated equally. Important keywords (like "capital" or "France" in our example) are not given more weight than less important ones. More advanced methods like TF-IDF address this.
6. **Poor for Complex Questions:** It struggles with questions requiring inference, reasoning, or understanding of complex relationships between entities.

Due to these limitations, Jaccard Similarity is rarely used as the sole similarity metric in modern, high-performance QnA systems. More sophisticated techniques involving vector space models (like TF-IDF, Word2Vec, BERT embeddings), deep learning, and attention mechanisms are employed for better semantic understanding and answer extraction/generation. However, it serves as an excellent foundational concept for understanding text similarity.

## QnA System using Cosine Similarity

Another widely used method for measuring text similarity in QnA systems is **Cosine Similarity**. Unlike Jaccard Similarity which operates on sets of unique words, Cosine Similarity operates on **vector representations** of text. This allows it to incorporate term frequency and inverse document frequency (TF-IDF) weighting, giving more importance to rare but significant words.

## What is Cosine Similarity?

Cosine Similarity measures the cosine of the angle between two non-zero vectors in a multi-dimensional space. The closer the angle is to 0 degrees, the higher the cosine value (closer to 1), indicating greater similarity between the vectors. If the vectors are orthogonal (at 90 degrees), their cosine similarity is 0, meaning no similarity. If they are in opposite directions (180 degrees), the similarity is -1. In text analysis, vectors typically have non-negative components, so the similarity ranges from 0 to 1.

**Formula:**

Given two vectors, A and B, the Cosine Similarity $C(A, B)$ is calculated as:

$$C(A,B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Where:

- A · B is the dot product of vectors A and B.
- ||A|| and ||B|| are the Euclidean magnitudes (or lengths) of vectors A and B, respectively.
- Ai and Bi are the components of vectors A and B along dimension i.

**Simple Example:**

Let Vector A = [1, 1, 0] Let Vector B = [1, 0, 1]

Let Vector A = [1, 1, 0]

Let Vector B = [1, 0, 1]

- Dot Product ($A \cdot B$): $(1 \times 1) + (1 \times 0) + (0 \times 1) = 1 + 0 + 0 = 1$
- Magnitude $||A||$: $\sqrt{1^2 + 1^2 + 0^2} = \sqrt{1 + 1 + 0} = \sqrt{2} \approx 1.414$
- Magnitude $||B||$: $\sqrt{1^2 + 0^2 + 1^2} = \sqrt{1 + 0 + 1} = \sqrt{2} \approx 1.414$

Cosine Similarity $C(A, B) = \frac{1}{1.414 \times 1.414} = \frac{1}{2} = 0.5$

Cosine Similarity C(A,B)=1.414×1.4141=21=0.5

## QnA System using Cosine Similarity: Detailed Explanation

When applying Cosine Similarity to a QnA system, the core idea is to represent the user's question and each document/answer as numerical vectors in a shared multi-dimensional space. Each dimension in this space corresponds to a unique word (or term) in the vocabulary. The value in each dimension typically represents the importance of that word in the document, often using **TF-IDF (Term Frequency-Inverse Document Frequency)** weighting.

**Steps Involved:**

1. **Knowledge Base Preparation (Indexing):**
    - **Documents/Answers:** A collection of documents or pre-defined answers.
    - **Preprocessing:** Similar to Jaccard, text undergoes:
        - **Tokenization:** Breaking text into words.
        - **Normalization:** Lowercasing, removing punctuation, etc.
        - **Stopword Removal & Stemming/Lemmatization:** (Optional but recommended for better results).
    - **Vocabulary Creation:** Create a unique list of all terms (words) from all preprocessed documents. This forms the dimensions of our vector space.
    - **Vectorization (TF-IDF):** Each document/answer is converted into a TF-IDF vector.
2. **Query Processing:**
    - **User Question:** The question undergoes the same preprocessing steps.
    - **Vectorization:** The preprocessed question is also converted into a TF-IDF vector using the ***same vocabulary and IDF values*** **calculated from the knowledge base.**
3. **Similarity Calculation:**
    - The Cosine Similarity is calculated between the TF-IDF vector of the user's question and the TF-IDF vector of *each* document/answer in the knowledge base.
4. **Ranking and Retrieval:**
    - All calculated Cosine Similarity scores are ranked in descending order.
    - The document/answer with the highest Cosine Similarity score is retrieved and presented as the answer.

## Detailed Example (using simplified Term Frequency for vectors)

Let's use the same knowledge base and query, but simplify the vector creation to just term counts for clarity, instead of full TF-IDF, as TF-IDF requires calculating IDF across the entire corpus.

**Knowledge Base Documents (after preprocessing and stopword removal):**

- **Set D1:** `{'capital', 'france', 'paris'}`
- **Set D2:** `{'paris', 'beautiful', 'city'}`

- **Set D3:** `{'eiffel', 'tower', 'paris', 'france'}`

**User Question (Query Q):**

- **Set Q:** `{'capital', 'france'}`

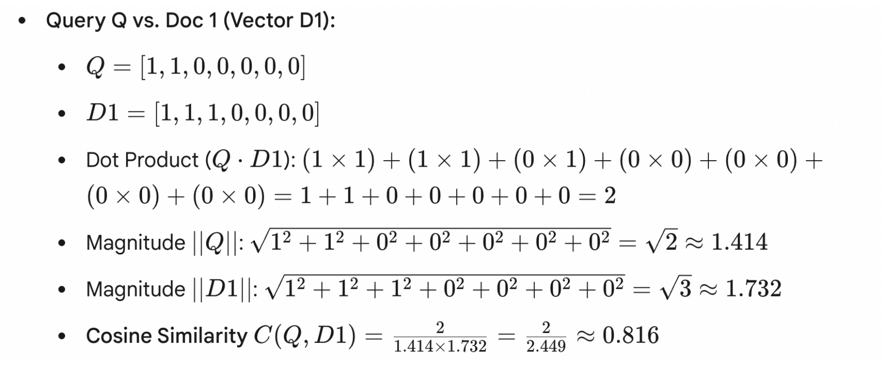
**1. Vocabulary Creation:** Unique terms across all documents and query: `{'capital', 'france', 'paris', 'beautiful', 'city', 'eiffel', 'tower'}`

**2. Vectorization (using simple term counts for illustration):** We'll create a vector for each document and the query, where each component corresponds to a term in our vocabulary. A '1' indicates presence, '0' indicates absence (similar to a Bag-of-Words model).

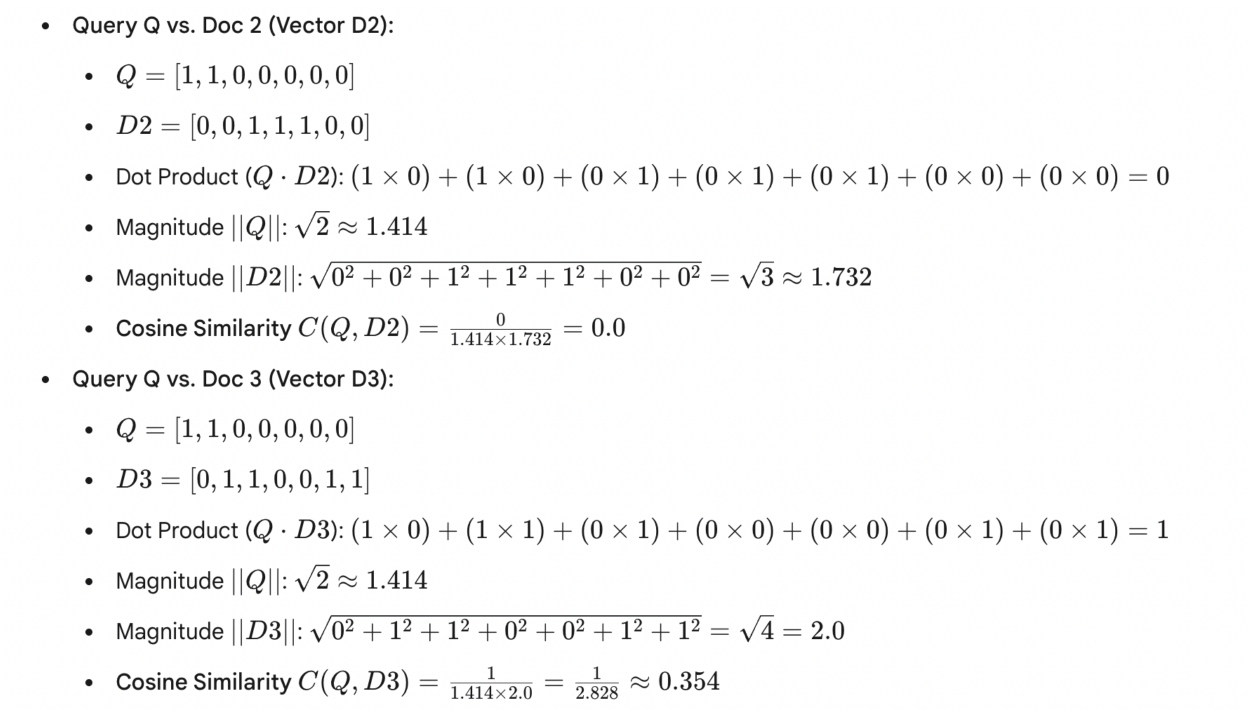
- **Vector Q (Query):**
  - capital: 1
  - france: 1
  - paris: 0
  - beautiful: 0
  - city: 0
  - eiffel: 0
  - tower: 0
  - **Q = [1, 1, 0, 0, 0, 0, 0]**
- **Vector D1 (Doc 1):**
  - capital: 1
  - france: 1
  - paris: 1
  - beautiful: 0
  - city: 0
  - eiffel: 0
  - tower: 0
  - **D1 = [1, 1, 1, 0, 0, 0, 0]**
- **Vector D2 (Doc 2):**
  - capital: 0
  - france: 0
  - paris: 1
  - beautiful: 1
  - city: 1
  - eiffel: 0
  - tower: 0
  - **D2 = [0, 0, 1, 1, 1, 0, 0]**
- **Vector D3 (Doc 3):**
  - capital: 0
  - france: 1
  - paris: 1

- beautiful: 0
- city: 0
- eiffel: 1
- tower: 1
- **D3 = [0, 1, 1, 0, 0, 1, 1]**

**3. Similarity Calculation (Cosine Similarity):**



- Query Q vs. Doc 1 (Vector D1):
  - $Q = [1, 1, 0, 0, 0, 0, 0]$
  - $D1 = [1, 1, 1, 0, 0, 0, 0]$
  - Dot Product ($Q \cdot D1$): $(1 \times 1) + (1 \times 1) + (0 \times 1) + (0 \times 0) + (0 \times 0) + (0 \times 0) + (0 \times 0) = 1 + 1 + 0 + 0 + 0 + 0 + 0 = 2$
  - Magnitude $||Q||$: $\sqrt{1^2 + 1^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2} = \sqrt{2} \approx 1.414$
  - Magnitude $||D1||$: $\sqrt{1^2 + 1^2 + 1^2 + 0^2 + 0^2 + 0^2 + 0^2} = \sqrt{3} \approx 1.732$
  - Cosine Similarity $C(Q, D1) = \frac{2}{1.414 \times 1.732} = \frac{2}{2.449} \approx 0.816$



- Query Q vs. Doc 2 (Vector D2):
  - $Q = [1, 1, 0, 0, 0, 0, 0]$
  - $D2 = [0, 0, 1, 1, 1, 0, 0]$
  - Dot Product ($Q \cdot D2$): $(1 \times 0) + (1 \times 0) + (0 \times 1) + (0 \times 1) + (0 \times 1) + (0 \times 0) + (0 \times 0) = 0$
  - Magnitude $||Q||$: $\sqrt{2} \approx 1.414$
  - Magnitude $||D2||$: $\sqrt{0^2 + 0^2 + 1^2 + 1^2 + 1^2 + 0^2 + 0^2} = \sqrt{3} \approx 1.732$
  - Cosine Similarity $C(Q, D2) = \frac{0}{1.414 \times 1.732} = 0.0$
- Query Q vs. Doc 3 (Vector D3):
  - $Q = [1, 1, 0, 0, 0, 0, 0]$
  - $D3 = [0, 1, 1, 0, 0, 1, 1]$
  - Dot Product ($Q \cdot D3$): $(1 \times 0) + (1 \times 1) + (0 \times 1) + (0 \times 0) + (0 \times 0) + (0 \times 1) + (0 \times 1) = 1$
  - Magnitude $||Q||$: $\sqrt{2} \approx 1.414$
  - Magnitude $||D3||$: $\sqrt{0^2 + 1^2 + 1^2 + 0^2 + 0^2 + 1^2 + 1^2} = \sqrt{4} = 2.0$
  - Cosine Similarity $C(Q, D3) = \frac{1}{1.414 \times 2.0} = \frac{1}{2.828} \approx 0.354$

**4. Ranking and Retrieval:**

Based on the calculated Cosine Similarities:

- Doc 1: 0.816
- Doc 2: 0.0
- Doc 3: 0.354

**Ranking (Highest to Lowest):**

1. Doc 1 (0.816)
2. Doc 3 (0.354)
3. Doc 2 (0.0)

**Result:** Similar to Jaccard, the system would retrieve **Doc 1: "The capital of France is Paris."** as the most relevant answer.

## Advantages of Cosine Similarity over Jaccard Similarity in QnA Systems

1. **Handles Term Frequency:** Cosine similarity, especially when used with TF-IDF vectors, accounts for how often a word appears in a document. A word appearing multiple times in a document is considered more important to that document's topic. Jaccard Similarity only considers presence/absence.
2. **Less Sensitive to Document Length:** By normalizing the vectors by their magnitude, Cosine Similarity is less affected by the absolute length of documents. It measures the *angle* between vectors, focusing on the orientation (i.e., the relative distribution of terms) rather than the absolute number of shared terms. This helps prevent longer documents from being unfairly penalized or rewarded.
3. **Incorporates Term Importance (with TF-IDF):** TF-IDF weighting gives higher scores to words that are rare in the overall corpus but frequent in a specific document, making them good indicators of document content. This improves relevance.

## Limitations of Cosine Similarity in QnA Systems

Despite its advantages over Jaccard, Cosine Similarity still has limitations:

1. **No Semantic Understanding (without advanced embeddings):** While better with TF-IDF, it still fundamentally operates on word overlap. It doesn't inherently understand synonyms, antonyms, or the deeper meaning of words. "Big" and "large" are treated as distinct words unless more advanced techniques (like word embeddings) are used to create the vectors.
2. **Order Insensitivity:** Like Jaccard, it treats text as a "bag of words." The order of words does not influence the similarity score, which can be critical for understanding nuances in questions and answers.
3. **Vocabulary Dependence:** The performance is limited by the vocabulary derived from the corpus. Out-of-vocabulary words are ignored.
4. **Computational Cost:** For very large vocabularies and many documents, creating and comparing high-dimensional vectors can be computationally intensive, though efficient sparse matrix operations mitigate this.

In modern QnA systems, both Jaccard and Cosine Similarity (especially with TF-IDF) serve as foundational concepts. However, they are often augmented or replaced by more advanced techniques that leverage deep learning models (e.g., Word2Vec, GloVe, BERT, Sentence

Transformers) to create dense, semantic vector representations of text, allowing for a much richer understanding of meaning and context.