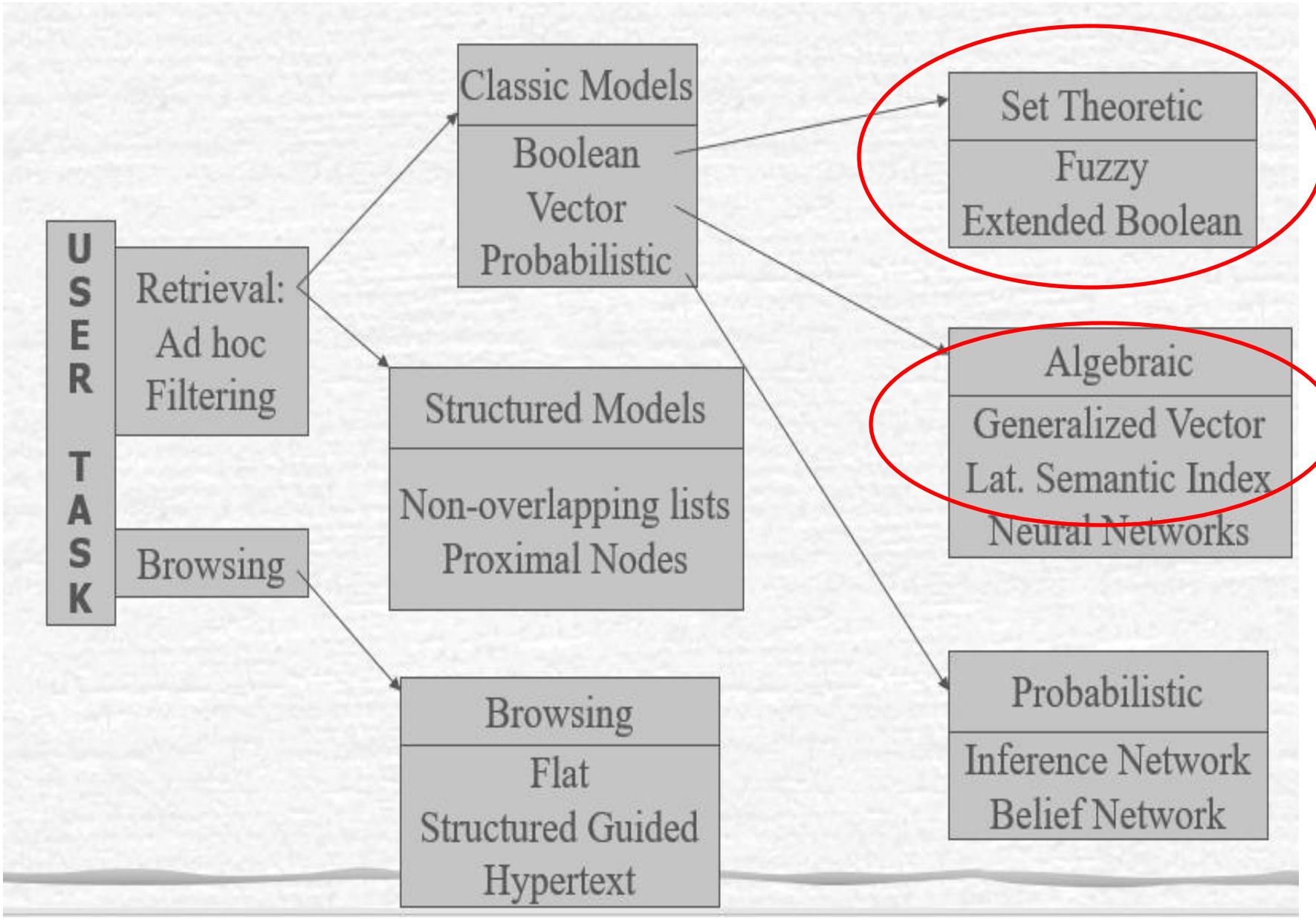
Information Retrieval

By
Dr Syed Khaldoon Khurshid


USER TASK
Retrieval: Ad hoc Filtering
Browsing
Classic Models
Boolean
Vector
Probabilistic
Structured Models
Non-overlapping lists
Proximal Nodes
Browsing
Flat
Structured Guided
Hypertext
Set Theoretic
Fuzzy
Extended Boolean
Algebraic
Generalized Vector
Lat. Semantic Index
Neural Networks
Probabilistic
Inference Network
Belief Network

Lectures of the week

- **Set Theoretic IR Models**
  - **Fuzzy**
  - **Boolean Extended**
- **Algebraic**
  - **Generalized Vector**
  - **Latent Semantic Index**

Extended Classical IR Models

- Set-theoretic Information Retrieval (IR) models are like **advanced versions of the classic models,** we use to search for information on the internet. Let's break them down into easy concepts:

1. Set-Theoretic IR Models:

- Think of these models as new and improved ways to find information. In classic search, it's like saying, "**Show me everything related to cats.**" But with set-theoretic models, we can say, "**Show me things about cats that are a little bit related to dogs, too**." It allows for more flexibility in our search.

Extended Classical IR Models

2. Algebraic IR Models:

- Algebraic models are like **math equations for searching**. In classic search, we use simple words and phrases. **Algebraic models use math to understand our search better.** So if you want to find "**apple**" plus "**orange**," **it can find things related to both fruits**.

3. Probabilistic IR Models:

- These models use probabilities, which are like **guesses**, **to find what you're looking for**. In classic search, you get a big list of results. But **probabilistic models try to guess which results are most likely to be what you want. So if you're looking for "puppies," it will guess which websites are most likely to have cute puppy pictures**.

Set-theoretic Information Retrieval (IR) models

- In simple terms, **set-theoretic models make searching smarter**. **Fuzzy models understand that relevance is not just yes or no,** and **extended Boolean models let you use supercharged words to find exactly what you want.**
- Two key aspects of set-theoretic IR models are **Fuzzy Set Theory** and **Extended Boolean Set Theory.**

Set-theoretic Information Retrieval (IR) Models

Fuzzy Set Theory

Fuzzy Set Theory:

- Imagine regular search is like a light switch—it's either on (relevant) or off (not relevant). But with the fuzzy model, it's like a **dimmer switch**. It can be a **little relevant, somewhat relevant, or very relevant**. This helps when things are not just black and white. For example, if you're searching for "happy movies," it can find ones that make you really happy or just a little happy.
- Fuzzy set theory allows for **degrees of membership** rather than strict binary membership. In classical set theory, an element either belongs to a set or doesn't. However, in **fuzzy set theory, an element can have a partial or fuzzy membership in a set.**

Fuzzy Set Theory:

- In the context of IR, this means **that documents or queries can be associated with multiple degrees of relevance**. Instead of just being relevant or irrelevant, a document can be **partially relevant to a query**, capturing the idea that relevance is not always black and white.
- Fuzzy set-based IR models help address the ambiguity and vagueness inherent in many information retrieval tasks. They allow for a **more representation of relevance**.

Simple Example:

- Let's simplify Fuzzy Set Theory with a basic example related to searching and retrieving documents.

Scenario:

- Imagine you're searching for documents about cute animals. In classic search, it's like you're asking, "**Is this document about cute animals, yes or no**?“

Fuzzy Set Approach:

- With Fuzzy Set Theory, it's not just a simple yes or no. It's like saying, "**How much does this document talk about cute animals on a scale of 0 to 1**?“

Simple Example:

- **Document A:** Talks a lot about cute animals, so it gets a relevance score of 0.9.
- **Document B:** Talks a bit about cute animals, so it gets a relevance score of 0.5.
- **Document C:** Doesn't talk much about cute animals, so it gets a relevance score of 0.2.
- **Now, instead of just showing or hiding documents, you can rank them by their relevance scores:**
- 1. Document A (0.9) - Super cute animals!
- 2. Document B (0.5) - Kind of cute animals.
- 3. Document C (0.2) - Not so cute animals.
- **Fuzzy Set Theory** lets you express the "fuzziness" of relevance, making your search smarter and more hinted.

Mathematical Expression:

- The general mathematical expression for Fuzzy Set Theory:

- *Let **A** be a set, and **x** be an element in that set. The degree of membership (relevance) of element x in set A is represented by a membership function μA(x), which assigns a value between **0 and 1**:*

$$\mu A(x) = [0, 1]$$

- This equation indicates that μA(x) can take any value between 0 (completely not a member) and 1 (fully a member) to quantify the degree of membership of element x in set A.

- Let's denote a set of documents related to "**cute animals**" as A. In classic set theory, an element (a document) is either in the set (1) or not in the set (0).
- In Fuzzy Set Theory, we use a membership function μA(x) that assigns a degree of membership (a value between 0 and 1) to each document x in set A. It looks like this:
- **μA(x) = 0.9 (for a document highly related to cute animals)**
- **μA(x) = 0.5 (for a document moderately related to cute animals)**
- **μA(x) = 0.2 (for a document slightly related to cute animals)**
- **μA(x) = 0.0 (for a document not related to cute animals)**

Set-theoretic Information Retrieval (IR) Models

Extended Boolean Model

Extended Boolean Model:

- Regular search uses simple words like "AND," "OR," and "NOT" to find things. **The extended Boolean model is like using superpowers with these words.** You can be more specific. For example, if you want to find "**Fruits OR Vegetables BUT NOT Processed Food,**" it can do that easily.

Set-theoretic Information Retrieval (IR) models

- Let's simplify the Extended Boolean Model with a basic example.
- **Scenario:** Imagine you're searching for articles about animals, but you want to be specific. You want articles about "dogs" OR "cats" BUT NOT "birds."
- **Boolean Query:** In classic search, you might type "**dogs OR cats NOT birds**." It gives you some results, **but you still see articles about birds.**
- **Extended Boolean Model Approach:** With the Extended Boolean Model, you can be more specific:
- "dogs" OR "cats": This part is the same; it finds articles about dogs or cats.
- BUT NOT "birds": Here's where it gets powerful. **It filters out any articles about birds.** So, you get articles only about dogs and cats, and no more annoying articles about birds!
- In simple terms, the Extended Boolean Model gives you more control over your search by letting you include and exclude specific words or topics. It's **like telling the search engine exactly what you want and what you don't want.**

Another example

- Let's simplify the Extended Boolean Model with a basic mathematical example related to searching and retrieving documents.

Scenario:

- Imagine you're searching for documents in a library, and you have three types of documents: "Science," "Technology," and "Art." You want to find documents that are either about "Science" or "Technology" but not "Art."

Boolean Query:

- In classic search, you might ask the librarian for "Science OR Technology NOT Art." This gives you a bunch of documents, including some about "Art" that you don't want.

Another example

Extended Boolean Model Approach:

With the Extended Boolean Model, you can be more precise with your query:

- "Science OR Technology": This part is the same; it finds documents about Science or Technology
- BUT NOT "Art": Here's where it makes a difference. It filters out any documents about Art. So, you get documents only about Science and Technology, and no more documents about Art.
- Mathematically, it's represented as:

(Documents related to Science) OR (Documents related to Technology) AND (NOT Documents related to Art)

- In simple terms, the Extended Boolean Model allows you to search for documents by including some topics and excluding others, making your search more accurate and efficient.

Mathematical Expression

The general mathematical expression for the Extended Boolean Model:

- Let A, B, C, and so on be sets of documents or terms. You can combine them using logical operators:

  OR: Union of sets

  AND: Intersection of sets

  NOT: Complement of a set

- The expression can take the form of complex combinations like:

  (A OR B) AND (NOT C)

- This allows you to create queries that include certain terms, exclude others, and combine them in various ways to retrieve documents that match your information needs.

- Set-theoretic IR models extend classical IR models, offer flexibility by allowing users to work with **degrees of relevance (fuzzy)** or to construct **precise queries (Extended Boolean),** depending on their information needs.
- In summary, set-theoretic IR models, including Fuzzy Set Theory and Extended Boolean Set Theory, extend classical IR models to offer more versatile and expressive approaches to information retrieval. Fuzzy models **accommodate the inherent uncertainty in relevance assessment**, while Extended Boolean models **provide a structured and precise way to retrieve information**. These models cater to different user preferences and the nature of the retrieval task.

Algebraic Information Retrieval (IR) Models

- Algebraic Information Retrieval models are extensions of classical IR models that use **mathematical operations** to represent and retrieve information.

Algebraic Information Retrieval (IR) Models

- Algebraic IR Models, like the **Generalized Vector Model**, **represent documents and queries as vectors in a mathematical space**, allowing for efficient retrieval and relevance ranking.
- **Latent Semantic Indexing (LSI)** is an algebraic model that uncovers latent semantic structures in text data, **improving search accuracy by capturing word associations.**
- **Neural networks** in IR leverage artificial intelligence to enhance information retrieval **by modeling complex patterns in text, offering more accurate results and recommendation systems.**

Algebraic Information Retrieval (IR) Models: Generalized Vector Model

Algebraic Information Retrieval (IR) Models

Generalized Vector Model:

- In this model, **documents and queries are represented as vectors in a multi-dimensional space. Each dimension corresponds to a unique term or feature. The model considers the relationship between terms and the importance of each term in documents and queries**.

- **Documents and queries** are like arrows in this multi-dimensional space.
- The **direction and length** of these arrows indicate the relevance and importance of terms.

Algebraic Information Retrieval (IR) Models

- **By comparing the direction and length of these arrows**, the model calculates the **similarity between queries and documents, ranking documents by their relevance** to the query.

- In simple terms, Algebraic IR models, including the Generalized Vector Model, **use mathematical vectors to measure how closely documents match a user's query.** This approach helps improve the accuracy of information retrieval and ranking.

A simple Example:

Scenario:

- You are searching for the articles about healthy eating, and you want to find the most relevant ones. Your query is: "**I'm interested in articles about 'fruits' and 'vegetables.'"**

Generalized Vector Model Approach:

1. **Create Vectors:** Each article is represented as a vector in a multi-dimensional space. For simplicity, let's say we still have two dimensions, one for "fruits" and one for "vegetables."

2. **Vector Direction:** The direction of each article's vector points toward "fruits" and "vegetables" based on how much these terms are mentioned in the article.

3. **Query Vector:** Your query, "fruits" and "vegetables," is also represented as a vector.

A simple Example:

4. **Measure Similarity:** The model calculates the similarity between your query vector and each article's vector. The closer the angle between the vectors, the more relevant the article is to your query.

5. **Rank Articles:** Articles with vectors that align closely with your query vector are ranked higher because they are more relevant to your interest in "fruits" and "vegetables."

In simple terms, the Generalized Vector Model now uses vectors to find articles that are most similar to your interest in "fruits" and "vegetables," considering how much these terms are mentioned in each article to rank them by relevance.

Mathematical expression of the Generalized Vector Model:

Let's translate the example into the mathematical expression of the Generalized Vector Model:

Mathematical Expression:

1. Vectors Creation:

- Each article is represented as a vector in a 2D space.
- The two dimensions are for "**fruits**" and "**vegetable**s.“

2. Vector Direction:

- The direction of each article's vector is determined by the number of times "fruits" and "vegetables" are mentioned in the article.
- For example, if an article mentions "fruits" five times and "vegetables" three times, its vector points in the direction of (5, 3).

3. Query Vector:

- Your query, "fruits" and "vegetables," is also represented as a vector: (1, 1).

4. Similarity Calculation:

- The model calculates the cosine similarity between the query vector and each article's vector.
- The cosine similarity is a measure of how close the angle is between the vectors.

5. Ranking:

- Articles with vectors that have a smaller angle with the query vector are ranked higher.
- This means articles that mention "fruits" and "vegetables" in a way similar to your query are considered more relevant.

- In simple terms, the Generalized Vector Model uses vectors to calculate the similarity between your query and articles. The closer the angle between the vectors, the more relevant the articles are to your query about "fruits" and "vegetables."

Mathematical expression of the Generalized Vector Model:

- General mathematical expression for the Generalized Vector Model:
- Let's consider a query Q with terms q_1, q_2, ..., q_n, and a document D with terms d_1, d_2, ..., d_m.

1. Create Vectors:

- Represent each document and the query as vectors in a multi-dimensional space.
- Each dimension corresponds to a unique term.

2. Vector Representation:

- For each document D, the vector's components represent the importance or frequency of terms in that document. **For example, D = [w_1, w_2, ..., w_m], where w_i represents the weight of term d_i in document D.**
- For the query Q, the vector represents the presence or absence of terms. For example, **Q = [1, 0, 1, ..., 1], where 1 indicates the presence of term q_i in the query.**

Mathematical expression of the Generalized Vector Model:

3. Similarity Calculation:

- **Calculate the cosine similarity between the query vector Q and each document vector D.** The cosine similarity measures the angle between the vectors and is used **to determine the relevance.**

4. Ranking:

- Rank the documents based on their cosine similarity with the query. **Documents with smaller angles and higher cosine similarity values are ranked higher, indicating higher relevance to the query.**

Mathematical expression of the Generalized Vector Model:

The mathematical formula for cosine similarity between two vectors A and B is given by:

Cosine Similarity(Q, D) = (Q • D) / (||Q|| * ||D||)

- **Q • D** represents the dot product of the query and document vectors.
- **||Q|| and ||D||** represent the **magnitudes (or lengths)** of the query and document vectors.
- In summary, the Generalized Vector Model uses vector representations and cosine similarity to rank documents by their relevance to a given query. **The formula quantifies this relevance based on the angles between the vectors.**

The Vector Space Model (VSM) Vs the Generalized Vector Space Model (GVSM)

- The Vector Space Model (VSM) and the Generalized Vector Space Model (GVSM) are both techniques used in information retrieval to represent and measure the relevance of documents to a query. Here's a simple comparison of the two:

Vector Space Model (VSM):

1. **Representation:** In VSM, documents and queries are represented as vectors in a multi-dimensional space.
2. **Term Importance:** Each dimension of the vector corresponds to a unique term, and the value in each dimension represents the importance or frequency of that term in the document or query.

Vector Space Model (VSM)

3. **Term Frequency:** VSM typically uses term frequency-inverse document frequency (TF-IDF) to weigh the importance of terms.
4. **Binary Representation:** In its simplest form, VSM can use binary values (1 or 0) to indicate the presence or absence of terms in the document or query.
5. **Cosine Similarity:** VSM measures the similarity between the document and query vectors using cosine similarity.

Generalized Vector Space Model (GVSM)

1. **Extension of VSM:** GVSM is an extension or a generalization of the VSM.
2. **Representation:** It still represents documents and queries as vectors in a multi-dimensional space, similar to VSM.
3. **Term Importance:** GVSM allows for more flexibility in determining the importance or relevance of terms within documents and queries. It can use various methods to weigh terms, such as Boolean, probabilistic, or fuzzy logic.

Generalized Vector Space Model (GVSM)

4. **Flexible Term Weights:** GVSM can use a broader range of term weightings, including binary (presence or absence), probabilistic (based on probabilities), or fuzzy (gradual membership).
5. **Adaptability:** GVSM can adapt to different information retrieval scenarios and can incorporate various models beyond TF-IDF.
6. **Customization:** GVSM allows for customizing the model to suit specific requirements and information retrieval goals.

VSM Vs GVSM

In summary, **the primary difference is that GVSM is a more flexible and generalized approach that builds upon the foundations of the VSM. While VSM primarily uses TF-IDF and cosine similarity, GVSM provides greater customization and adaptability by accommodating different term weighting schemes and retrieval models.** It can be tailored to suit specific information retrieval needs, making it a more versatile choice in some cases.

Algebraic Information Retrieval (IR) models

- Algebraic Information Retrieval (IR) models, particularly the **Latent Semantic Indexing (LSI)** model, are extensions of classical IR models with a focus on understanding the **underlying meaning of words in documents**. Here's a simple explanation:

Classical IR Models: Demerit of VSM

- Classical IR models like the Vector Space Model (VSM) represent documents and queries as vectors based on term frequency.
- They match documents to queries using simple word matching, **but they don't capture the deeper meaning of words.**

Algebraic Information Retrieval (IR) models: Latent Semantic Indexing (LSI)

- Algebraic IR models, such as LSI, go beyond simple word matching. **They aim to understand the semantic relationships between words and documents**.
- LSI uses mathematical techniques like **singular value decomposition (SVD)** to analyze the relationships between terms in a large corpus of text.
- **By identifying patterns and associations between terms, LSI can discover hidden semantic structures.**
- This allows LSI to retrieve documents that are **conceptually related to the query**, even if they don't share exact words.

Algebraic Information Retrieval (IR) models: Latent Semantic Indexing (LSI)

- In essence, LSI extends classical IR models by introducing a **layer of semantic understanding**. It can find documents that are conceptually similar to a query, making it a powerful tool for information retrieval when you want to capture the **meaning behind words, not just their presence**.

A simple Example: Movie Recommendations

Let's use a real-life situation to illustrate **Latent Semantic Indexing (LSI)** in a simple way:

Real-Life Scenario: Movie Recommendations

- Imagine you have a movie streaming service like Netflix. You want to **recommend movies to users based on their preferences**. Here's how LSI could work in this context:

1. **User Movie Ratings:**

- Users rate movies they have watched (e.g., "Star Wars" 5 stars, "Harry Potter" 4 stars).

2. **Movies and Keywords:**

- Each movie has **keywords associated with it** (e.g., "Star Wars" has keywords like "space," "adventure," "aliens," and "hero").
- These **keywords describe the content of the movies**.

A simple Example: Movie Recommendations

3. **Building a Term-Document Matrix:**

- Create a **term-document matrix** where rows represent keywords and columns represent movies.
- Fill in the matrix with **1** if a keyword is associated with a movie and **0** if it's not.

4. **Singular Value Decomposition (SVD):**

- **Apply SVD to the matrix** to uncover **latent patterns**.
- SVD reveals that **keywords** like "space" and "aliens" **are related and often appear together in the user ratings.**

Singular Value Decomposition (SVD)

- **Singular Value Decomposition (SVD)** is a mathematical technique used in various fields, including information retrieval, **to analyze and represent data**. It decomposes **a matrix into three other matrices, which helps extract important information from the original data**.

Understanding SVD:

- Imagine you have a matrix that represents user ratings for movies, where **rows represent users**, **columns represent movies, and the values are ratings**

Singular Value Decomposition (SVD)

User	Movie 1	Movie 2	Movie 3
User 1	5	4	0
User 2	0	3	4
User 3	4	0	5

You can perform SVD on this matrix to find underlying patterns or **latent factors** that explain the ratings. SVD decomposes this matrix into three matrices:

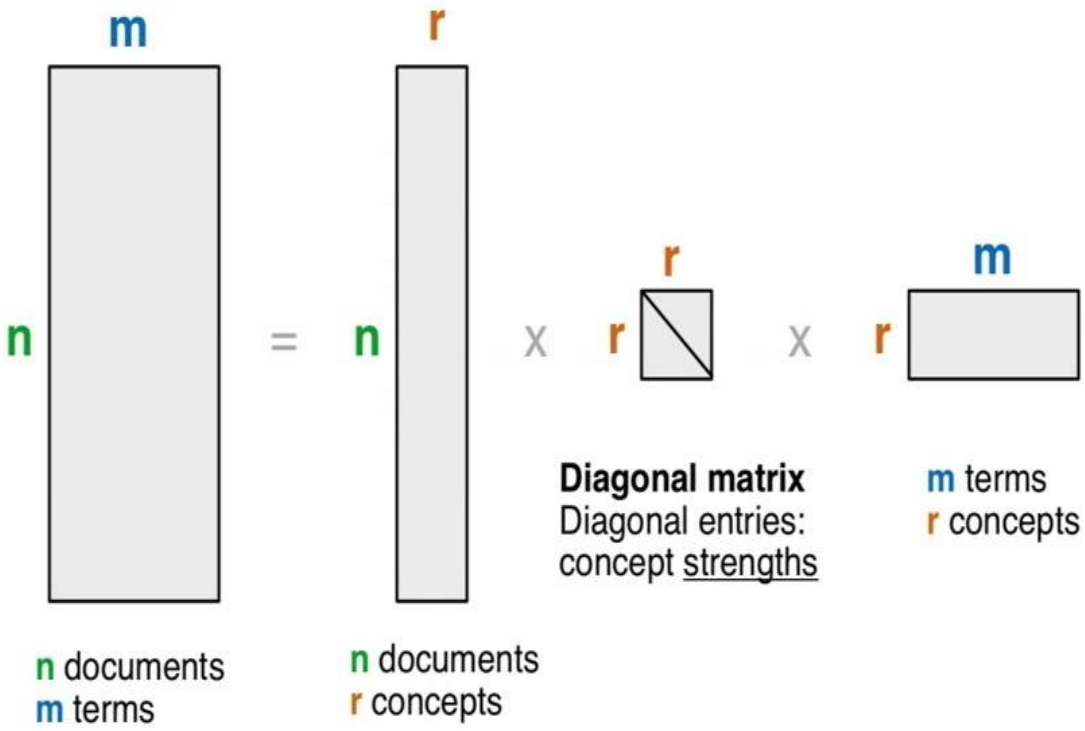
i. **U (Left Singular Vectors):** Describes how users relate to latent factors.

ii. **Σ or Λ (Singular Values):** A diagonal matrix that tells you the importance of each latent factor.

iii. **V^T (Right Singular Vectors):** Describes how movies relate to latent factors.

SVD Definition (pictorially)

$$A_{[n \times m]} = U_{[n \times r]} \Lambda_{[r \times r]} (V_{[m \times r]})^T$$



GTx

How to Calculate latent factors of LSI Model

To understand how **Latent Factor 1** in the **U matrix** is calculated, we need to walk through the main steps of **Singular Value Decomposition (SVD)**. The latent factors are essentially patterns or features that are discovered in the matrix. Let's break it down in a simple way:

Step-by-Step Explanation with Example

Let's start by considering a simple example matrix, where each row represents **users** and each column represents **movies**. The values are the **ratings** given by users to movies.

Example Matrix: User-Movie Ratings

$$M = \begin{bmatrix} 5 & 0 & 3 \\ 4 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 3 & 5 \end{bmatrix}$$

This matrix M has 4 users (rows) and 3 movies (columns). SVD will help decompose this into **latent factors** that explain patterns in how users rate movies.

Step 1: Apply SVD

When we apply SVD to matrix M, we decompose it into three matrices:

$$M = U \cdot \Sigma \cdot V^T$$

Where:

- U: Describes how users relate to the **latent factors** (hidden patterns),
- Σ: Diagonal matrix showing the **importance** of each latent factor,
- V^T: Describes how movies relate to the **latent factors.**

Let's assume that after running SVD, the result looks like this (these values would typically be computed by software):

U Matrix (Users to Latent Factors)

$$U = \begin{bmatrix} 0.75 & 0.6 \\ 0.82 & -0.3 \\ -0.45 & 0.8 \\ -0.2 & -0.9 \end{bmatrix}$$

Each row corresponds to a user, and each column corresponds to a **latent factor** (hidden pattern). So, **Latent Factor 1** is the first column of this U matrix.

Step 2: Latent Factor 1 in the U Matrix

Let's focus on Latent Factor 1, the first column of the U matrix:

$$\text{Latent Factor 1} = \begin{bmatrix} 0.75 \\ 0.82 \\ -0.45 \\ -0.2 \end{bmatrix}$$

This column tells us how much each user is connected to Latent Factor 1.

- User 1 has a strong positive connection to Latent Factor 1 (0.75),
- User 2 has an even stronger connection (0.82),
- User 3 has a moderate negative connection (-0.45),
- User 4 has a weaker negative connection (-0.2).

Step 3: How Latent Factor 1 is Calculated

Now, let's get into how Latent Factor 1 is calculated during SVD.

1. **Step 3.1: Centering the Data** – SVD typically starts by **centering the matrix**, which means subtracting the average rating from each user or movie. This makes the data easier to work with.

   For example, if User 1 gave an average rating of 4, we would subtract 4 from each of User 1's ratings.

2. **Step 3.2: Identify Covariance** – Next, SVD looks at the covariance between users and movies to see which users tend to rate movies similarly. Essentially, it finds **patterns** in the data. For example:

   - Users who tend to rate **action** movies similarly may be grouped together by Latent Factor 1.

3. **Step 3.3: Eigenvectors and Eigenvalues** – SVD uses mathematical tools called **eigenvectors** and **eigenvalues** to identify the **directions** (or patterns) of greatest variation in the data. These eigenvectors are the **latent factors**.

4. **Step 3.4: Singular Values** – The singular values in the Σ matrix tell us how important each of these directions (latent factors) is. Latent Factor 1 typically captures the **biggest pattern** in the data.

5. **Step 3.5: Assigning Weights** – Finally, each user is assigned a **weight** (how strongly they relate to a latent factor) by projecting their original ratings onto the **direction** of Latent Factor 1.

This projection results in the values you see in **Latent Factor 1** of the U matrix. These values represent **how strongly each user is connected** to this hidden pattern.

Step 4: Interpreting Latent Factor 1

The numbers in **Latent Factor 1** (e.g., 0.75, 0.82, -0.45, -0.2) tell us how much each user is connected to this hidden pattern. In practice, **Latent Factor 1** might represent a general trend like:

- **"Users who prefer action movies"** – If Latent Factor 1 relates to action movies, then users with a positive number (like User 1 and User 2) would likely prefer action movies, while those with a negative number (like User 3) would not.

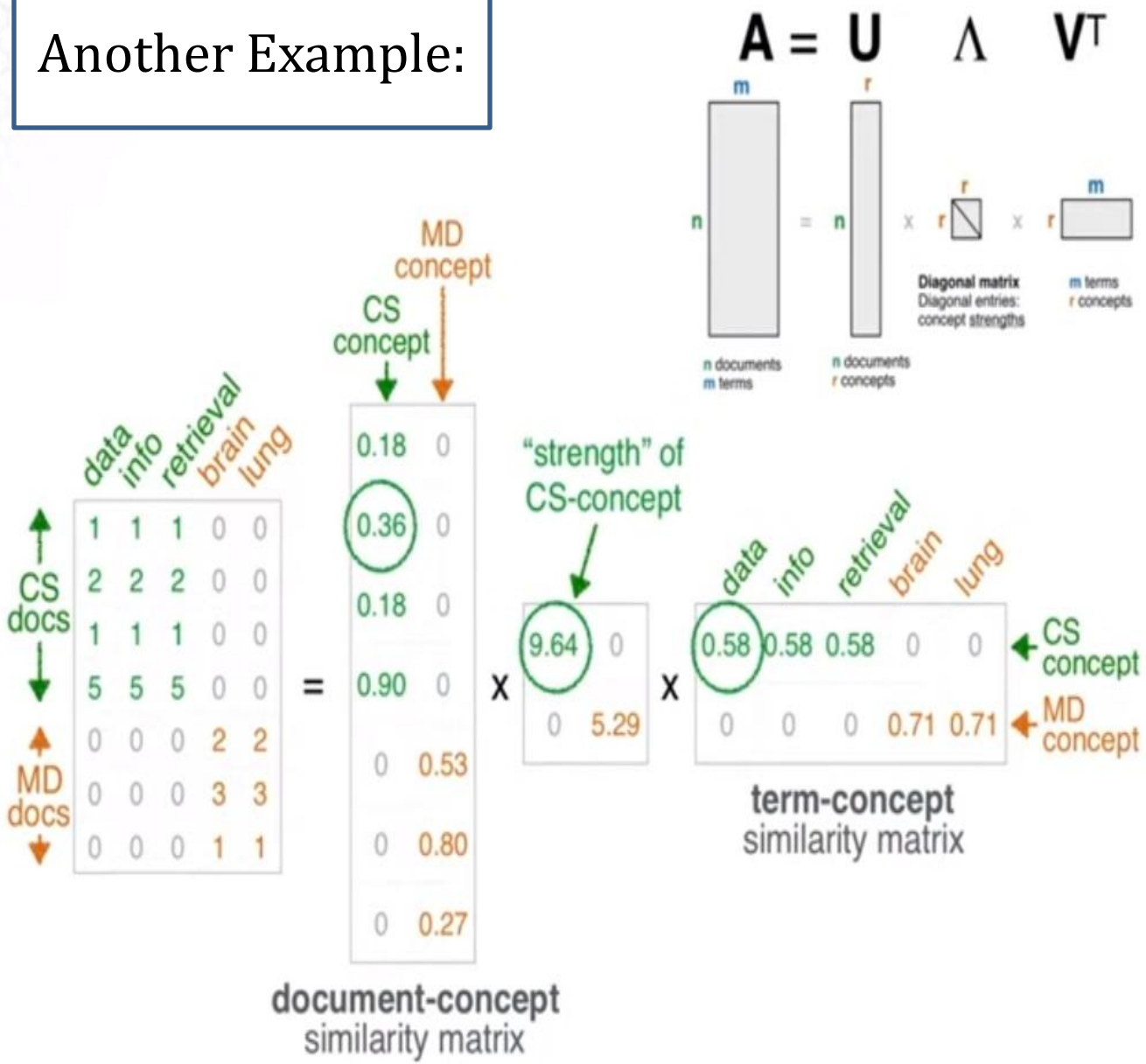
Step 5: Reconstructing the Original Ratings

Once we have the **U**, **Σ**, and V^T matrices, we can multiply them back together to approximate the original ratings matrix. This can help predict how a user would rate a movie they haven't seen based on their relationship with the latent factors.

Key Takeaway:

- **Latent Factor 1** represents a hidden pattern that describes how users relate to movies.
- It is calculated by looking at patterns of variation in the data (like similar ratings behavior).
- The numbers in **Latent Factor 1** of the U matrix show how strongly each user is connected to this hidden pattern (e.g., a specific genre preference).

Another Example:



Singular Value Decomposition (SVD)

After performing SVD, you can represent the original matrix approximately by multiplying these three matrices:

$$\textbf{Original Matrix} \approx U * \Sigma * V^T$$

OR $$\textbf{Original Matrix} \approx U * \Lambda * V^T$$

The resulting matrices capture underlying **patterns in the data**. For example, the **first latent factor might represent a genre preference (e.g., action or drama).** By analyzing the values in the matrices, **you can understand which movies are associated with which genres and how much each user likes a particular genre.**

In information retrieval, SVD can help analyze document-term matrices, uncover hidden associations in text data, and improve various tasks, such as recommendation systems or text clustering.

A simple Example: Movie Recommendations

5. Recommendations:

- When a user watches a sci-fi movie (not rated yet), **LSI suggests movies with related keywords like "space," "aliens," and "adventure."**
- Even if the user hasn't rated movies with those exact keywords, **LSI captures the semantic relationships and makes better recommendations.**
- In this example, **LSI helps the movie service recommend films based on the semantic content and user preferences, even if the keywords used for recommendations aren't identical to the ones the user has rated.** It improves the user experience by considering latent semantics in movie preferences.

Another Example: Space Exploration

- Imagine you're searching for information about "space exploration." In a traditional Information Retrieval system, if a document contains the term "space exploration," it would be considered relevant.

Now, let's apply **Latent Semantic Indexing (LSI):**

1. Creating a Term-Document Matrix:

- You collect a set of documents and create a matrix where rows represent terms (words) and columns represent documents.
- You mark the presence (1) or absence (0) of terms in each document.

Another Example: Space Exploration

2. Singular Value Decomposition (SVD):

- LSI uses a mathematical technique called Singular Value Decomposition to identify underlying patterns in the term-document matrix.

3. Discovering Latent Semantics:

- LSI uncovers hidden relationships between terms and documents.
- It might find that "space exploration" is related to terms like "NASA," "moon landing," and "rocket technology.“

Another Example: Space Exploration

4. Enhancing Retrieval:

- When you search for "**space exploration**," LSI considers documents related not only to the exact term but also those associated with the discovered **latent semantics**.
- You might get documents that mention "**NASA missions**" or "**history of space travel**" even if they don't contain the exact phrase "space exploration.“

- In simple terms, LSI goes beyond matching keywords and understands the context and related concepts. **It helps you find documents that are semantically similar, even if they don't use the exact words you searched for.** This makes your search results more accurate and comprehensive.

Computation of the Latent Semantic Indexing (LSI)

- Here's the mathematical expression of the Latent Semantic Indexing (LSI) process in the context of the example:

Step 1: Creating a Term-Document Matrix

- Suppose you have a set of documents (D1, D2, D3) and a set of terms (T1, T2, T3). You create a term-document matrix (A) where each element A[i][j] represents the presence (1) or absence (0) of a term in a document. For example:

$$A = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

Computation of the Latent Semantic Indexing (LSI)

Step 2: Singular Value Decomposition (SVD)

- You perform SVD on the term-document matrix A to factor it into three matrices: U, Σ (Sigma), and V^T (V transpose).

Step 3: Discovering Latent Semantics

- The Σ matrix contains singular values that reveal latent semantics. For instance, if Σ has non-zero values in the first column, it suggests a latent semantic related to "space exploration."

Step 4: Enhancing Retrieval

- When you search for "space exploration," LSI considers documents not only based on exact keyword matches but also those associated with the discovered latent semantics. It enhances retrieval by considering semantic similarities.

Computation of the Latent Semantic Indexing (LSI)

- In mathematical terms, LSI involves matrix operations and decomposition techniques, such as SVD, to uncover latent semantic structures within the term-document matrix. These structures are then used to improve information retrieval by expanding the search scope beyond exact keyword matches.

Computation of the Latent Semantic Indexing (LSI)

Let:

- **A** be the term-document matrix, where **A[i][j]** represents the weight of term **i** in document **j**.
- **U** be the left singular vectors matrix.
- **Σ** (Sigma) be the diagonal matrix of singular values.
- $\mathbf{V^T}$ (V transpose) be the right singular vectors matrix.
- In LSI, the formula for approximating the original term-document matrix A using a reduced rank k is given by:

$$\mathbf{A \approx U \Sigma_k V^T_k}$$

- **Here, the subscript k indicates that we consider only the top k singular values, which correspond to the most significant latent dimensions. This approximation allows LSI to reduce the dimensionality of the original data and uncover latent semantic relationships between terms and documents.**

Computation of the Latent Semantic Indexing (LSI)

- The actual application of LSI involves the specific calculations of U, Σ, and V^T through singular value decomposition (SVD) and then using the reduced matrices to enhance information retrieval by capturing semantic similarities.

Computed Example

- Let’s populate the formula with values from the example provided:
- Suppose you have a term-document matrix A and after performing singular value decomposition (SVD), you get the matrices U, Σ, and V^T. Here's a simplified representation with example values:
- Term-Document Matrix A:

$$A = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

Computed Example

- Let's assume we are interested in capturing the top 2 singular values (k = 2) for simplicity. The formula with example values becomes:

$$\mathbf{A \approx U\Sigma_k V^T_k}$$

Using example values, suppose U, Σ, and V^T are as follows:

U (Left Singular Vectors):

U = | 0.64 0.54 |

| 0.36 -0.80 |

| 0.68 0.25 |

Computed Example

- Σ (Singular Values - considering only the top 2):

$$\Sigma = \begin{vmatrix} 2.12 & 0 & 0 \\ 0 & 1.73 & 0 \end{vmatrix}$$

- V^T (Right Singular Vectors):
- $V^T = \begin{vmatrix} 0.57 & 0.57 & 0.57 \end{vmatrix}$
- With these values, the approximation becomes:

$$\mathbf{A \approx U \Sigma_k V^T_k}$$

$$A \approx \begin{matrix} (0.64 & 0.54) \end{matrix} * \begin{matrix} (2.12 & 0) \\ (0.36 & -0.80) \\ (0.68 & 0.25) \end{matrix} * \begin{matrix} (0.57) \\ (0) \\ (0) \end{matrix}$$

- This simplified example demonstrates how LSI reduces the dimensionality of the term-document matrix while capturing latent semantic relationships between terms and documents.

Working of the example

Let's simplify the working of the example:

1. **Term-Document Matrix (A):**

- We start with a matrix (A) representing the presence (1) or absence (0) of terms in documents.
- Each row corresponds to a term, and each column corresponds to a document.

2. **Singular Value Decomposition (SVD):**

- We apply SVD to matrix A to find hidden patterns in the data.
- SVD breaks down A into three matrices: U, Σ (Sigma), and V^T (V transpose).

3. **Choosing the Top 2 Singular Values (k = 2):**

- For simplicity, we decide to consider only the top 2 singular values.

Working of the example

4. **Approximation Formula:**
- We use the formula $\mathbf{A \approx U\Sigma_k V^T_k}$ to approximate the original matrix A.
- U contains information about terms, Σ holds the singular values, and V^T contains information about documents.
5. **Calculation:**
- We perform the calculations by multiplying these matrices with only the top 2 singular values.
6. **Result:**
- The result is an approximation of the original matrix A.
- This reduced-dimensional representation captures latent semantic relationships, allowing for more accurate information retrieval by considering semantic similarities.
- In simpler terms, Latent Semantic Indexing (LSI) helps us find hidden patterns in data, like words that often appear together. It reduces the data's complexity, making it easier to understand and retrieve relevant information, even if the exact words aren't used. This improves search accuracy and helps you find what you're looking for more effectively.

REFERENCES:

[1] “INFORMATION RETRIEVAL MODELS,” *Ebrary*, 2013. https://ebrary.net/201793/psychology/retrieval_models (accessed Feb. 22, 2023).

[2] “Ch_2 Information Retrieval Models,” *Rutgers.edu*, 2023. https://aspoerri.comminfo.rutgers.edu/InfoCrystal/Ch_2.html (accessed Feb. 22, 2023).

[3] “Fig. 2. Horizontal taxonomy.,” *ResearchGate*, 2021. https://www.researchgate.net/figure/Horizontal-taxonomy_fig2_47397195 (accessed Feb. 22, 2023).

[4] “4.1. IR MODELS – BASIC CONCEPTS – Wachemo University e-Learning Platform,” *Wachemo-elearning.net*, 2023.https://wachemo/elearning.net/courses/information-storage-and-retirievalitec3081/lessons/chapter-four-ir-model/topic/4-1-ir-models-basic-concepts/ (accessed Feb. 22, 2023).

[5] https://www.youtube.com/watch?v=M1duqgg8-IM (accessed Oct. 15, 2023).


Any
Questions