Assignment 3

Session: 2021 – 2025

Submitted by:

Ghulam Mustafa (2021-CS-39)

Supervised by:

Sir. Khaldoon Khurshid

Department of Computer Science

University of Engineering and Technology Lahore Pakistan

Description:

Implement to retrieve documents by Probabilistic, Non-Overlapped List and Proximal Nodes Models. For the Probabilistic retrieval the BIM is used here.

1. Probabilistic Retrieval Model

The **Binary Independence Model (BIM)** is a probabilistic approach to rank documents based on their likelihood of relevance to a user's query.

Steps

1.1 Preprocessing

Preprocessing is the foundation of efficient retrieval. The steps include:

- **Load Documents**:
  - Load `.txt` files from a directory, separating content into titles and body.
  - Maintain a mapping for quick access during querying.
- **Remove Empty Spaces**:
  - Eliminate extra spaces between words.
  - Example: `" Hello World "` → `"Hello World"`.
- **Remove Non-Nouns and Verbs**:
  - Filter words based on predefined stopword lists (`the`, `a`, etc.).
  - Exclude words ending in *-ing*, *-ed*, *-ly*.
  - Example:
    - Input: `"Running quickly, the brown fox jumps over lazy dogs."`
    - Output: `"brown fox jumps lazy dogs."`.
- **Filter Nouns**:
  - Identify nouns using suffixes like `-ness, -tion, -ity`.
  - Example: `"Happiness and ability lead to celebration."` → `"Happiness, ability, celebration."`.

1.2 Indexing

Create a **Linked_List_Dictionary** with the following structure:

- **Key**: Document title.
- **Value**: List of nouns extracted from the document content.
- **Sub-elements**: Nodes contain pointers to the next element for efficient traversal.

1.3 Query Representation

- Represent the user query as a binary vector.
    - If a term exists in the index, set the corresponding vector element to `1`; otherwise, set it to `0`.
    - Example Query: `"fox jumps"`
        - Document terms: `{fox, jumps, lazy, dogs}`
        - Query vector: `[1, 1, 0, 0]`.

1.4 Document Scoring

- Calculate similarity between the query vector and document vectors using coefficients such as:
    - **Jaccard Coefficient**: J(A,B)=|A∩B||A∪B|J(A, B) = \frac{|A \cap B|}{|A \cup B|}J(A,B)=|A∪B||A∩B|
    - **Dice Coefficient**: D(A,B)=2|A∩B||A|+|B|D(A, B) = \frac{2|A \cap B|}{|A| + |B|}D(A,B)=|A|+|B|2|A∩B|.

1.5 Ranking and Retrieval

- Rank documents based on their similarity scores.
- Retrieve and display the top-K documents as the most relevant.

---

2. Non-Overlapped List Model

This model retrieves and combines document lists for specified terms without overlapping results.

Steps

2.1 Identify Terms of Interest

- Specify terms or keywords (e.g., "machine learning," "data visualization").

2.2 Retrieve Document Lists

- Query the `Linked_List_Dictionary` to retrieve lists of documents containing each term.
    - Example:
        - Term 1: `"machine learning"` → `{D1, D2, D5}`.
        - Term 2: `"data visualization"` → `{D3, D4, D6}`.

2.3 Combine Lists (Non-Overlapping Results)

- Use set operations to compute the union of the document lists: DNonOverlap=Dmachine_learning∪Ddata_visualizationD_{\text{NonOverlap}} = D_{\text{machine_learning}} \cup D_{\text{data_visualization}}DNonOverlap =Dmachine_learning∪Ddata_visualization.
- Result: `{D1, D2, D3, D4, D5, D6}`.

2.4 Present Results

- Display documents from DNonOverlapD_{\text{NonOverlap}}DNonOverlap as a non-overlapping set.

---

3. Proximal Nodes Model

This model retrieves documents based on relationships in a network of interconnected terms/entities.

Steps

3.1 Define Proximal Nodes

- Identify key terms or entities related to the query.
  - Example Query: `"space exploration"`.
    - Proximal nodes: `{NASA, astronauts, space missions}`.

3.2 Explore Network Relationships

- Traverse a graph or node-based representation to find connections between terms and documents.
  - Example:
    - Node `"NASA"` connects to documents `{D1, D3}`.
    - Node `"space missions"` connects to `{D2, D4}`.

3.3 Retrieve Connected Documents

- Retrieve documents directly linked to proximal nodes.
  - Example: `{D1, D2, D3, D4}`.

3.4 Present Results

- Rank and display documents based on the strength of their connections to proximal nodes.

Integration with Indexer Implementation

1. **Workflow Overview (from Slides)**:
   - Load documents → Preprocess → Index nouns → Retrieve and rank documents.
   - Data flow diagrams (DFDs) ensure clarity in each stage.
2. **Advantages of Custom Linked_List_Dictionary**:
   - Efficiently handles term storage, retrieval, and node traversal.
   - Scalable for large datasets.
3. **Preprocessing Enhancements**:
   - Filtering non-relevant terms (verbs, stopwords) ensures clean and precise indexing.
4. **Real-World Application**:
   - Mimics search engine functionality by structuring queries and retrieving results based on relevance.