Enhanced Document Ranking System

Information Retrieval

Submitted to
Dr. Syed Khaldoon Khurshid

Submitted By

Shahzaib Irfan 2021-CS-07

University of Engineering and Technology
Lahore, Pakistan

Table of Contents

1 Introduction

1.1 Project Overview

This project implements a sophisticated document ranking system using Python and Streamlit, focusing on advanced text processing and information retrieval techniques. The system enables users to upload multiple text documents and perform search queries with multiple ranking methods, including keyword matching and TF-IDF scoring.

1.2 Objective

The primary goals of this document ranking system are to:

- Provide an interactive, user-friendly search interface
- Implement advanced text processing techniques
- Offer multiple document ranking strategies
- Enable efficient information retrieval from uploaded documents

2 System Design

2.1 System Architecture

The system comprises several key components:

1. **Document Preprocessing Module**
   - Handles text tokenization
   - Removes stopwords
   - Converts text to lowercase
   - Removes punctuation and non-alphanumeric characters
2. **Ranking Engine**
   - Supports two ranking methods:
     (a) Keyword Matching
     (b) TF-IDF (Term Frequency-Inverse Document Frequency) Scoring
3. **Streamlit Web Interface**
   - Allows document uploads
   - Provides search query input
   - Displays ranked search results

2.2 Text Processing Pipeline

2.2.1 Tokenization

The `preprocess_text()` function tokenizes input text using NLTK's `word_tokenize()`. It breaks text into individual words and applies several preprocessing steps:

- Convert text to lowercase
- Remove punctuation
- Remove stopwords

Example:

```
text = "The quick brown foxes are running fast."
tokens = preprocess_text(text)
# Result: ['quick', 'brown', 'foxes', 'running', 'fast']
```

2.2.2 Stopword Removal

Common English stopwords like "the", "is", "and" are removed to focus on meaningful content. This is achieved using NLTK's predefined stopwords list.

2.3 Ranking Mechanisms

2.3.1 Keyword Matching

The `keyword_matching()` function ranks documents based on the number of query keywords present in each document. It provides a simple, count-based ranking method.

2.3.2 TF-IDF Scoring

The TF-IDF ranking method calculates document relevance more sophisticatedly:

- **Term Frequency (TF)**: Measures how frequently a term appears in a document
- **Inverse Document Frequency (IDF)**: Measures how unique a term is across all documents

The TF-IDF score is calculated using the formula:

$$TF\text{-}IDF(\text{word}) = TF(\text{word}) \times \log\left(\frac{\text{Total Documents}}{\text{Documents Containing Word}}\right)$$

3 Implementation Details

3.1 Libraries and Tools

- **NLTK**: Natural Language Processing
- **Streamlit**: Web application framework
- **Python Standard Libraries**: `os`, `math`

3.2 Key Functions

3.2.1 `preprocess_text(text)`

- Tokenizes input text
- Converts to lowercase
- Removes stopwords and punctuation

3.2.2 `calculate_tf(word, document)`

- Calculates term frequency by dividing word count by total document words

3.2.3 `calculate_idf(word, documents)`

- Calculates inverse document frequency
- Measures term uniqueness across documents

3.2.4 `calculate_tf_idf(query, documents)`

- Computes TF-IDF scores for documents matching the query
- Ranks documents based on their relevance scores

4 User Interface

4.1 Features

- Document upload functionality
- Search query input
- Ranking method selection
- Result display with document name, score, and snippet

4.2 Ranking Method Options

1. **Keyword Matching**: Simple count-based ranking
2. **TF-IDF Scoring**: Advanced relevance-based ranking

5 Results and Evaluation

5.1 Performance Characteristics

- Supports multiple text documents
- Provides flexible search and ranking
- Efficient preprocessing and scoring mechanisms

5.2 Ranking Accuracy

- Keyword Matching: Quick, basic relevance determination
- TF-IDF Scoring: More nuanced, considers term importance

6 Conclusion

The Enhanced Document Ranking System demonstrates a robust approach to information retrieval. By combining text preprocessing, advanced ranking algorithms, and an intuitive user interface, the system provides an effective tool for searching and ranking text documents.

6.1 Future Improvements

- Implement semantic search
- Add support for more file types
- Enhance query expansion techniques
- Improve ranking algorithm sophistication