

Information Retrieval – Assignment 3 (Creation of a Search Engine)

A presentation by Paridhi, Pranay, Saverro and Zayan



Table of contents

01

**Introduction to
the Search Engine**

02

Indexing Method

03

Retrieval Method

04

**Ranking
Mechanism**

05

**System
Demonstration**

06

Evaluation



01

Introduction to the Search Engine



The Dataset

Kaggle's Goodreads dataset offers an abundance of information about books, such as titles, authors, ratings, and categories. It provides insights on reading habits and patterns thanks to its millions of readers and extensive literary content. By analysing this data, researchers can better understand user preferences and create solutions that improve the reading experience.

Key Features

- **Book Metadata:** Title, author(s), publication date, publisher, ISBN, and description.
- **User Interactions:** Ratings, reviews, shelves, and reading statuses.
- **Genre Information:** Categorisation of books into various genres and sub-genres..

Potential Applications

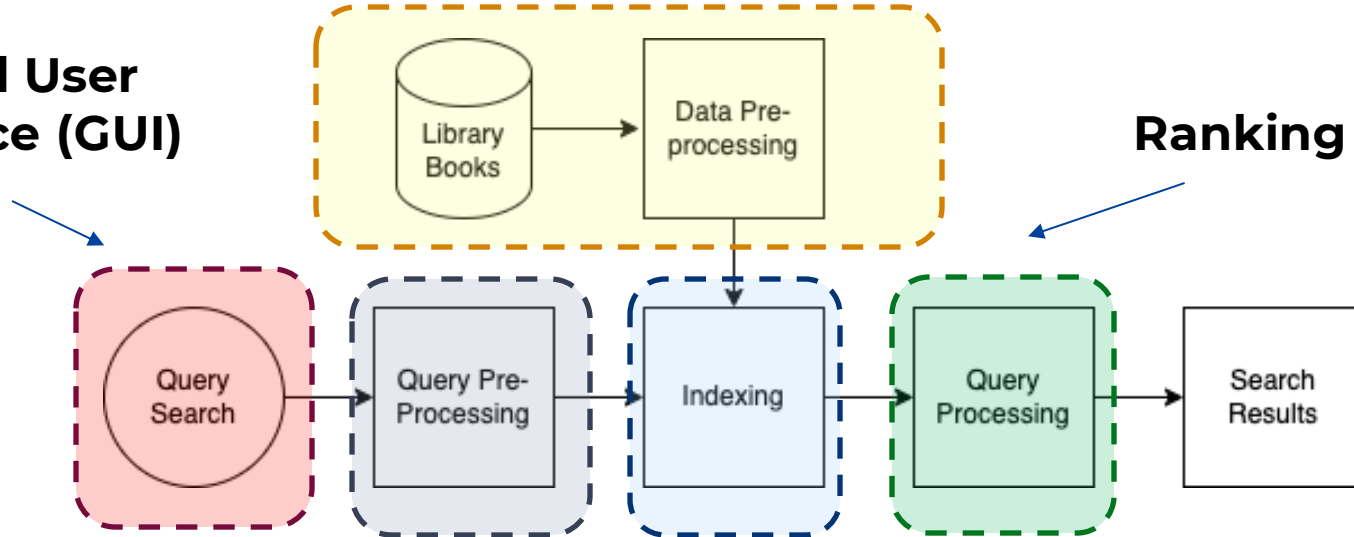
- **Recommender Systems:** Creating customised engines for book recommendations based on reading history and user preferences.
- **Sentiment analysis:** Examines opinions and evaluations from users to determine how people feel about particular books and writers.
- **User Analysis:** Investigating patterns and trends among various literary genres and subgenres is known as genre analysis.
- **Community Dynamics:** Examining the relationships, engagement, and dynamics of the Goodreads community on the platform.



Data Pre-Processing

General User Interface (GUI)

Ranking



Query Processing

Indexing with TF-IDF



02

Indexing Method



Data Pre-Processing

Data Cleaning

Text cleaning techniques are applied, such as removing symbols, punctuation, and special characters.

Data Transformation

Text is converted to lowercase to ensure uniformity in representation. Dates are also converted from MM/DD/YYYY format to DD/MM/YYYY.

Indexing

The pre-processed data is then transformed into TF-IDF vectors, where each term in the document is represented by its TF-IDF weight, facilitating efficient indexing and retrieval.

```
def prepro(text):
    text = str(text)                                # Ensure text is a string
    text = re.sub(r'[#@]\w+', ' ', text)            # Delete hashtags and mentions
    text = re.sub(r'^a-zA-Z0-9\s', ' ', text)        # Keep Letters and numbers
    text = text.lower()                             # Change to Lower case
    text = re.sub(r' +', ' ', text).strip()         # Remove extra spaces
    text = re.sub(r'http\S+|www\S+', ' ', text)      # Remove URLs
    text = re.sub(r'[!?\']', '', text)              # Remove specific punctuation
    text = re.sub(r'</br>', ' ', text)              # Remove breaks
    text = re.sub(r'-', ' ', text)                  # Remove hypens
    return text

# Applying the preprocessing
df['title'] = df['title'].map(prepro)
df['authors'] = df['authors'].map(prepro)
df['publisher'] = df['publisher'].map(prepro)
```




TF – IDF Vectorisation

| Term Frequency-Inverse Document Frequency

Is a widely used technique in information retrieval for representing the importance of terms in documents within a collection.

Our Project |

It is used to transform textual data (titles, authors, publishers, ISBNs) into numerical vectors. This method assigns weights to terms based on their frequency in individual documents and their rarity across the entire document collection.



Why we used it against other methods?: The choice of TF-IDF vectorisation was made because it can improve search results by highlighting terms' importance throughout the collection as well as their frequency inside individual documents. It is also perfect for scaling across huge datasets because of its computational efficiency, which is essential for search engine operations to maintain fast response times.

TF-IDF Equation



The TF-IDF is calculated as:

- Here Term frequency, $tf(t,d)$, is the relative frequency of term t within document d .
- The inverse document frequency, $idf(t,D)$ is a measure of how much information the word provides, i.e., how common or rare it is across all documents.

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D)$$

Term Frequency

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

- Where $f_{t,d}$ is the raw count of a term in a document
- The denominator is simply the total number of terms in document d

Inverse Document Frequency

$$idf(t,D) = \log \frac{N}{|\{d: d \in D \text{ and } t \in d\}|}$$

- Where N : total number of documents in the corpus
- $|\{d \in D : t \in d\}|$: number of documents where the term t appears.

03

Retrieval Method



Vector Space Model (VSM)

- The **Vector Space Model (VSM)** is a widely used framework in information retrieval for representing documents and queries as vectors in a multidimensional space.
- In VSM, each document in the collection and each query is represented as a vector in a high-dimensional space.
- The dimensions of this space correspond to terms or features extracted from the documents and queries.

Advantages of VSM:

- **Flexibility:** VSM can handle documents of varying lengths and formats.
- **Scalability:** It is suitable for large document collections due to its efficient representation and retrieval process.
- **Intuitive Interpretation:** The geometric interpretation of vectors in a multidimensional space makes it easy to understand and visualise.

Limitations of VSM:

- **Sparse Representations:** VSM can result in high-dimensional and sparse representations, especially for large vocabularies.
- **Ignoring Semantic Relationships:** VSM does not capture semantic relationships between terms, which can limit its effectiveness in understanding the context of documents.



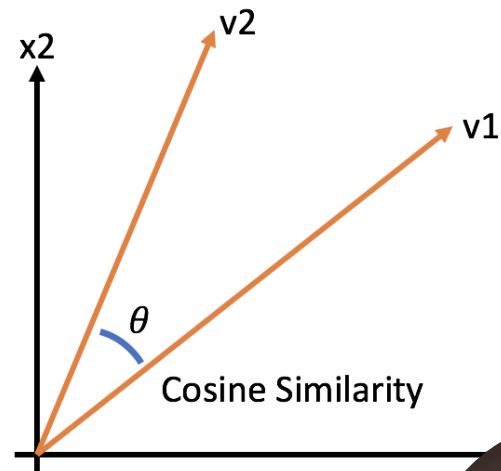
04

Ranking Mechanism



Cosine Similarity and Ranking

- **Definition:** Cosine similarity measures how similar two vectors are by calculating the cosine of the angle between them, which is particularly useful in high-dimensional spaces.
- **Application:** In search engines, it quantifies similarity by comparing the query vector with document vectors within the index.
- **Relevance Assessment:** A higher cosine similarity score suggests a document is more relevant to the given query.
- **Result Ranking:** Search results are ranked based on these scores, with higher scores indicating a closer match and thus a higher ranking.
- **User Experience:** By leveraging cosine similarity for ranking, users receive search results ordered by relevance, ensuring the most pertinent documents are highlighted first.



```

def search_engine(query, search_type='All'):
    preprocessed_query = prepro(query)
    if search_type != 'All' and search_type.lower() in df.columns:
        column_specific_df = df[df[search_type.lower()].str.contains(preprocessed_query,
                                                                       case=False, na=False)]

        if column_specific_df.empty:
            return None, None
        rankings = range(1, len(column_specific_df) + 1)
        return column_specific_df, rankings
    else:
        tfidf_vectorizer = TfidfVectorizer()
        combined_text = df['title'] + " " + df['authors'] + " " + df['publisher'] + " " + df['isbn']
        tfidf_matrix = tfidf_vectorizer.fit_transform(combined_text)
        query_vector = tfidf_vectorizer.transform([preprocessed_query])
        cosine_similarities = cosine_similarity(query_vector, tfidf_matrix).flatten()
        relevant_indices = cosine_similarities.argsort()[::-1]
        if len(relevant_indices) == 0:
            return None, None
        rankings = cosine_similarities[relevant_indices]
        return df.iloc[relevant_indices], rankings

```

05

System Demonstration



Information Retrival | Assingment 3 - Search Engine Implementation Using Vector Space Modelling

Group 16: Pranay Dhareshwar, Paridhi Gupta, Saverro Suseno, Zayan Adam Shareef

This Jupyter Notebook outlines the development of a search engine based on the Vector Space Model (VSM). We aim to showcase the implementation of it including indexing, querying, and ranking mechanisms within the VSM framework. This project is part of Group 16's assignment for the Information Retrieval course.

1.0 Data Preparation

In this section, we set up our environment and prepare our dataset for the search engine:

- **Library Imports:** Load the necessary Python libraries for data manipulation, regular expressions, and date handling.
- **Data Loading:** Read our dataset into a `pandas` DataFrame from a CSV file.
- **Null Value Replacement:** Substitute missing data with placeholder text to ensure data consistency.

In order for the search engine to produce accurate and effective search results, it needs clean, consistent, and meaningful data, which is why data preparation is so important. This stage is crucial in the context of the given search engine architecture since it determines the quality of data that will be matched against user queries and feeds into the indexing and query processing components.

```
[ ]: import pandas as pd
import re
from datetime import datetime
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext

# Load the CSV file into a pandas DataFrame
df = pd.read_csv('goodbooks_dataset.csv')

# Replacing the null values with the given placeholders
df['title'].fillna('[NO TITLE]', inplace=True)
df['authors'].fillna('[NO AUTHOR]', inplace=True)
df['publisher'].fillna('[NO PUBLISHER]', inplace=True)
df['isbn'].fillna('[NO ISBN]', inplace=True)
```



06

Overall Evaluation



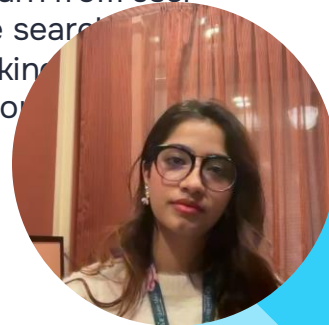
System Evaluation and Potential

Evaluation

- **Precision and Relevance:** The current use of TF-IDF and cosine similarity ensures that the system is good at identifying and ranking documents that are textually similar to the user query, but it may lack semantic understanding, which could affect the precision and relevance of the results.
- **User Interface Efficiency:** The simple and intuitive GUI allows for easy navigation and querying by the users, but the lack of advanced features such as query suggestion or autocorrect may limit the overall search experience.

Future Potential

- **Semantic Search Integration:** Future enhancements could include implementing natural language processing (NLP) techniques to interpret query context and semantics, potentially improving search accuracy beyond the current keyword matching approach.
- **Adaptive Learning Mechanisms:** Introducing machine learning algorithms that learn from user behaviour over time could allow the search engine to dynamically refine its ranking, creating a more responsive and tailored experience for individual users.



**Thank you for
listening!**

