**LyricsIR - Vector Space IR System Documentation**

**Overview**

A Vector Space Information Retrieval (IR) engine designed to search for a collection of song lyrics by computing the similarity between a query and each document using TF-IDF weighting and cosine similarity. The system processes text from artist-named folders and individual lyric files and ranks documents by their relevance to the user's query.

**Folder Structure**

The lyrics are stored in the following format:

Lyrics/

├── Artist1/

│ ├── Song1.txt

│ ├── Song2.txt

├── Artist2/

│ ├── SongA.txt

│ ├── SongB.txt

Each .txt file contains the full lyrics of a song.

**Features**

* **TF-IDF Weighting**: Each document and query is represented as a TF-IDF vector.
* **Cosine Similarity Ranking**: Results are ranked by similarity scores.
* **Stopword Filtering**: Common stopwords like "the", "is", "of" are ignored to improve accuracy.
* **Text Normalization**: Input is lowercased and cleaned of punctuation for consistent comparison.

**How It Works**

**1. Preprocessing**

* Converts all text to lowercase.
* Removes non-alphanumeric characters.
* Split text into word tokens.
* Filters out predefined stopwords.

**2. Index Construction**

* Assigns a unique ID to each song (doc\_id).
* Computes **term frequencies** (TF) for each word in each document.
* Stores this data in a dictionary-like structure: term\_doc\_freq[term][doc\_id] = frequency.

**3. Vector Length Normalization**

* For each document, its vector length (L2 norm) is precomputed.
* This normalization is used during cosine similarity calculation.

**4. Query Handling**

* User enters a free-form keyword query.
* Query is preprocessed similarly to lyrics.
* A query vector is created using TF-IDF weights.

**5. Ranking**

* Cosine similarity is calculated between query vector and each document vector.
* Documents are ranked in descending order of similarity.
* Top results (e.g., top 10) are printed with scores.

**Stopword Filtering**

Common stopwords are excluded during indexing and querying:

a, an, the, is, are, was, were, and, or, in, on, of, for, to, etc.

**Running the System**

1. Place all lyrics in the Lyrics/ArtistName/SongTitle.txt format.
2. Run the Python program.
3. Input a query (e.g. love peace)
4. View the ranked list of results like:

Artist1/SongA (score: 0.6124)

Artist2/SongB (score: 0.4312)

**Example Queries and Results**

| **Query** | **Description** | **Matches (ranked)** |
| --- | --- | --- |
| love | Finds songs with "love", ranked by TF-IDF | Songs with frequent "love" |
| freedom peace | Finds songs with both terms | Songs weighted by both terms |
| summer night | Phrase-like query using keywords | Ranked by joint relevance |

**Limitations**

* No Boolean logic (AND/OR/NOT) supported in this model.
* Only free-text keyword matching with similarity-based ranking.
* Phrase search is not explicitly handled (no positional index).

**Summary**

The Vector Space IR model for lyrics enables ranked retrieval using cosine similarity and TF-IDF scoring. It preprocesses both lyrics and queries for consistency, builds term-frequency-based inverted structures, and returns results sorted by semantic relevance. It is simple, scalable, and ready for future extensions like synonym expansion or query expansion.