NLP Project

Lyrics Retrieval System for English Songs

By:

GEBRAN Zeina

ROQAI CHAOUI Ghalia

SAMAKE Habibata

ALOUDA Lidao

Table of Contents

1. Introduction

1.1. Overview

The **Lyrics Retrieval System for English Songs** is a project aimed at exploring how users discover and explore music based on lyrical content. With the ever-growing amount of music available online, traditional methods of song discovery, such as searching by title or artist, can be limiting when users only recall a fragment of the lyrics. This system addresses this challenge by providing a solution that enables users to input a snippet of lyrics as a query and receive a ranked list of the top 10 most relevant songs from a large database. The system is particularly valuable for users seeking to identify a song whose title or artist is unknown but whose lyrics they remember partially.

At its core, the project leverages natural language processing (NLP) techniques and similarity measures to ensure precise and meaningful matches, even when the wording or structure of the input query differs slightly from the lyrics stored in the database. Unlike simple keyword-based searches, the system considers the semantic relationships between words, capturing deeper contextual similarities to enhance the quality of the results. This is achieved through the integration of tools like Word2Vec and various similarity metrics, such as Cosine Similarity, Jaccard Distance, and Levenshtein Distance.

The project's scope spans multiple stages, including dataset preparation, query processing, and the development of a robust matching framework. It utilizes two datasets: the Jamendo dataset, which contains high-quality English lyrics and serves as the query source, and the Wasabi dataset, a comprehensive repository of over 2 million songs, filtered and preprocessed to focus on normalized English lyrics. Together, these datasets form the foundation for building a scalable and efficient lyrics retrieval system.

The overall goal of the system is to enhance music discovery, offering users an innovative way to explore songs based on lyrical content. This project not only showcases the potential of combining data science and NLP for practical applications but also highlights the importance of effective data cleaning, efficient search algorithms, and semantic understanding in creating user-centric solutions. By addressing common challenges in music retrieval, this system provides a meaningful contribution to the field of information retrieval and music technology.

1.2. Objectives

The primary objective of the project is to develop a robust and efficient **Lyrics Retrieval System** that enhances music discovery by identifying songs based on partial lyrics provided by users. This involves building a system capable of processing large datasets, understanding the semantic context of lyrics, and returning accurate and meaningful results. The specific objectives are as follows:

1. **Develop a Retrieval System**:
   - ➢ Create a system that can efficiently search a large database of songs and rank results based on their relevance to the input query.
   - ➢ Ensure scalability and robustness to handle extensive datasets like the Wasabi dataset, which contains millions of songs.
2. **Representation Analysis**:
   - ➢ Employ advanced natural language processing techniques, such as Word2Vec, to represent song lyrics as numerical vectors that capture semantic relationships between words.
   - ➢ Analyze the impact of different preprocessing methods and similarity measures (Cosine Similarity, Jaccard Distance, Levenshtein Distance) on the accuracy and relevance of results.

3. **System Evaluation**:
   - ➢ Establish evaluation metrics to assess the performance of the system in retrieving accurate matches.
   - ➢ Test the system using both baseline and real-world datasets (Jamendo and Wasabi) to validate its effectiveness in matching query lyrics to songs.
4. **Enhance User Experience**:
   - ➢ Optimize the retrieval process to ensure fast and accurate responses, improving the overall usability of the system.

By achieving these objectives, the project aims to provide an innovative solution for users seeking to identify or explore music based on lyrical content, bridging the gap between text-based queries and musical discovery.

2. Dataset Overview

The success of the **Lyrics Retrieval System** heavily relies on the quality and preparation of the datasets used for querying and searching. This project utilizes two distinct datasets: the **Jamendo Dataset** and the **Wasabi Dataset**, each serving a specific purpose in the retrieval process.

2.1 Jamendo Dataset

The **Jamendo Dataset** is a curated collection of English songs used primarily as the **query source** for testing and validating the retrieval system. It is designed to provide a clean and consistent benchmark for evaluating the system's ability to match lyrics accurately. Key characteristics include:

- **Size and Scope**: Contains a small, focused set of 20 English songs, ensuring high-quality lyric content suitable for generating query snippets.
- **Purpose**: Used exclusively for generating queries to test the system's ability to retrieve relevant matches from the Wasabi Dataset.
- **Preprocessing**:
  - Minimal preprocessing to preserve the original quality of the lyrics.
  - Lyrics are filtered to include only English songs, ensuring linguistic consistency for retrieval tasks.
- **Advantages**: Provides a controlled environment to benchmark the system's performance against smaller, high-quality data.

2.2 Wasabi Dataset

The **Wasabi Dataset** is an extensive dataset of over 2 million songs, serving as the **primary search database** for the retrieval system. It represents the real-world scenario of searching for lyrics within a vast collection of musical content. Key characteristics include:

- **Size and Scope**: A large-scale dataset containing millions of songs, filtered to include only English lyrics to maintain consistency with the Jamendo dataset.
- **Purpose**: Acts as the main database for identifying songs that match user queries.
- **Preprocessing**:Extensive cleaning to ensure data quality, including:
  - Filtering lyrics to retain English-only content.
  - Normalizing text by converting to lowercase and removing extraneous symbols or HTML tags.
  - Eliminating punctuation and standardizing line breaks.
- **Challenges**:
  - Managing the large dataset size while maintaining efficient search and retrieval times.
  - Ensuring the integrity and normalization of diverse lyric sources.
- **Advantages**: Offers a realistic and comprehensive dataset for testing the scalability and robustness of the system.

3. Methodology

3.1 Query-Based Search

The **Query-Based Search** methodology forms the backbone of the Lyrics Retrieval System, enabling users to input a lyric snippet and retrieve the most relevant songs from the database. This approach combines advanced natural language processing techniques with efficient search algorithms to deliver accurate and meaningful results.

Process Description

The query-based search process consists of the following steps:

1. **Input Query**:
   - ➢ Users provide a snippet of lyrics as input, typically a paragraph or set of paragraphs.
   - ➢ The system preprocesses the query to standardize its format, ensuring consistency with the stored data.

2. **Quick Search**:
   - ➢ The system performs a quick filtering step to identify potential matches in the database.
   - ➢ Heuristics, such as word counts or structural patterns, are applied to narrow the search space rapidly.

3. **Similarity Computation**:
   For each candidate returned from the quick search, the system computes similarity scores ,that indicate how closely the candidate lyrics match the query, using advanced metrics such as:

   - ➢ **Cosine Similarity**: Captures semantic alignment in a high-dimensional vector space.
   - ➢ **Jaccard Distance**: Measures shared words between the query and candidate.
   - ➢ **Levenshtein Distance**: Assesses similarity based on character-level transformations.

4. **Result Ranking**:
   - ➢ Songs are ranked based on their similarity scores.
   - ➢ The system returns the top 10 most relevant matches, providing a clear and concise output for the user.

Example Input and Output

To illustrate the functionality of the query-based search, consider the following example:

- **Input Query:**

  “i'm on the highway to hell

  on the highway to hell

  highway to hell

  i'm on the highway to hell”

- **Processing**:
  - ➢ The system preprocesses the query by removing unnecessary punctuation, and tokenizing the text.
  - ➢ A quick search identifies candidate songs with structural similarities to the input query.
  - ➢ Similarity metrics calculate scores for each candidate.

- **Output**: A ranked list of the top 10 most relevant songs is displayed, along with their respective similarity scores. For example:

```
Top 10 résultats pour Paragraphe 1:
Chanson : Highway To Hell - Artiste : AC/DC
Paragraphe de la chanson : "i'm on the highway to hell
on the highway to hell
highway to hell
i'm on the highway to hell"
Texte de la requête : "i'm on the highway to hell
on the highway to hell
highway to hell
i'm on the highway to hell"
Similarité : 1.0000
✨
Chanson : The Bum Bum Song (Lonely Swedish) - Artiste : Tom Green
Paragraphe de la chanson : "my bum is on the rail
bum is on the rail
look at me
```

The following results were generated using **Word2Vec**, trained on the Google News dataset, and the **cosine similarity** measure to evaluate the semantic alignment between the

query text and the songs in the Wasabi dataset. Here is a summary of the performance:

1. **"Highway to Hell" - AC/DC**: The song achieved a perfect similarity score of **1.0000**, demonstrating an exact match with the query.
2. **"The Bum Bum Song (Lonely Swedish)" - Tom Green**: A partial match was observed with a similarity score of **0.5928**, indicating moderate alignment.
3. **"My Heart, The Compass (Points West)" - Race The Sun**: Another partial match with a similarity score of **0.5888**, showing a weaker alignment with the query text.

These results illustrate the effectiveness of the **Word2Vec + cosine similarity** approach in capturing semantic relationships. However, they also highlight its limitations when no exact textual match is present.

3.2. Data Cleaning and Text Normalization

In any data project, cleaning and normalization are fundamental steps to ensure the quality and consistency of the information. These steps are essential for producing reliable and meaningful analyses. In this project, we worked with textual data, which required specific preprocessing steps tailored to the tasks at hand. The goal of this phase was to transform raw data into a standardized, noise-free dataset, ready for analysis.

3.2.1. Data Cleaning Steps

1. Conversion to Lowercase

This step involves standardizing the text by converting all characters to lowercase. This eliminates variations caused by case sensitivity, allowing uniform handling of words. For instance, "Data," "data," and "DATA" are all transformed into "data." This normalization is crucial for tasks such as term searches or word counting.

2. HTML Entity Conversion

The initial dataset contained HTML entities that, in some cases, correspond to readable special characters. For example, ' is equivalent to an apostrophe ('). To make the text more comprehensible and usable, we converted these entities into their standard characters. For example, a phrase like I'm becomes I'm.

3. Replacing
 with Line Breaks

To preserve the structure of paragraphs in the text, we replaced
 tags with actual line breaks. This ensures better readability and maintains the original structure of the text.

4. Removing Non-Relevant HTML Tags

Certain HTML tags, such as , <p>, or , added no value to our textual analysis and were removed to clean the data. However, useful structural tags like
 were retained and converted into line breaks.

5. Removing Unnecessary Punctuation

To prevent the introduction of noise into the data, most punctuation marks, such as ;, ., or !, were removed. This step helps homogenize the text while preserving its meaning, making tasks like identifying similar words or analyzing word frequencies more efficient.

3.2.2. Expected Outcome

At the end of these cleaning steps, we obtained a clean, standardized dataset, ready for the next phases of the project, such as exploratory analysis, modeling, or training natural language processing (NLP) algorithms. This phase ensures the consistency and reliability of the data, providing a solid foundation for future processing. Additionally, we chose to restrict our scope by selecting only English-language songs, focusing our analysis efforts on a specific linguistic corpus.

3.3 Quick Search

Processing a massive dataset such as Wasabi, containing millions of songs, presents significant challenges in terms of textual search. Performing an exhaustive comparison of every query against the entire dataset would not only be inefficient but also highly costly in terms of time and resources. To address these limitations, we developed an optimized search system named **Quick Search**, which significantly reduces processing time while maintaining relevant results.

Objective and Approach

The primary goal of **Quick Search** is to efficiently narrow down the search space by quickly identifying the most relevant candidate songs without performing exhaustive comparisons. This method aims to optimize performance while ensuring sufficient accuracy for large-scale structured analyses.

To achieve this objective, we implemented an approach based on rapid filtering heuristics, structured around three main techniques:

1. **Using the Number of Words per Line**
   By leveraging the length of lines in terms of word count, this metric allows us to group songs based on similar textual patterns. This simplifies processing by immediately excluding paragraphs that do not match the structure of the query. For instance, a query with short lines will automatically exclude songs composed primarily of long lines.
2. **Identifying Similar Textual Structures**
   This approach relies on analyzing the structural patterns of songs. The heuristics help isolate paragraphs with characteristics similar to those of the query, significantly reducing the search space.
3. **Building the Textual Structures of Songs**
   This task involved constructing, for each paragraph in the Wasabi dataset(for each song), the corresponding structure based on the defined heuristic (number of words per line). This step represented a major technical challenge, particularly in selecting the most suitable data structure. The primary goal was to minimize computational time and resource usage while ensuring rapid retrieval of the required information during the search process.

Data Structures

Efficient and rapid search in a large dataset such as Wasabi, which contains millions of songs, requires careful selection of data structures. Choosing the right structures was critical for reducing computational time and resource usage, while ensuring scalability for the dataset's volume.

To meet the project's needs, we evaluated and utilized the following data structures:

➔ **Dictionaries and Lists:**

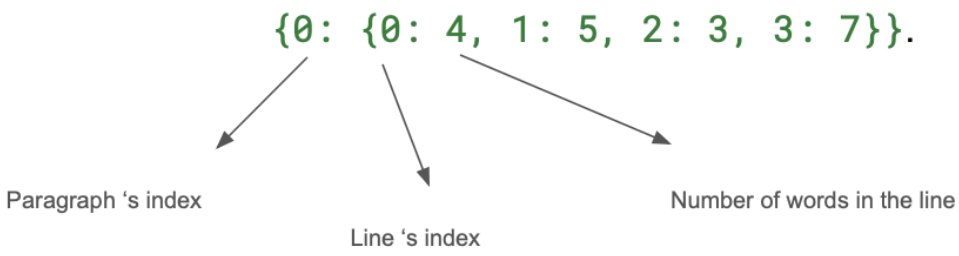
Description: A dictionary was employed to store song information based on the textual structure defined by the heuristic (number of words per line). For example, the following song snippet:

"through days of thunders

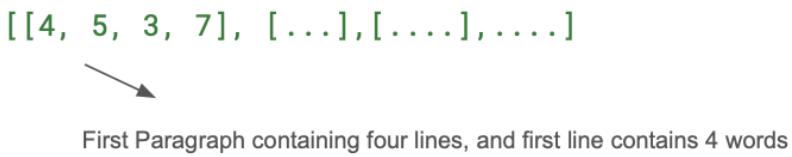
continuing stolid smiles to wear

we're going on

we nearly carried it to the end"



Songs were also represented as two-dimensional arrays, where each element corresponds to the word count of a specific line. For instance, the same snippet above could be represented as:



The main limitation of these two approaches lies in the need to construct a structure table for each song. Consequently, searching within these structures requires iterating through all elements, resulting in a worst-case complexity of O(N×M), where N represents the number of songs and M the maximum number of paragraphs. This approach can lead to query processing times of 1 to 2 minutes when dealing with large datasets.

➔ **Hash Map (Hash Table):**

To enhance the performance of our search algorithm, we explored the use of a hash table where the key corresponds specifically to the structure of a given paragraph. For instance, for the same paragraph above the hash table associates this structure with all paragraphs in the Wasabi dataset that share the same pattern (e.g., the number of words per line). By grouping paragraphs with identical structures under

a single key, the hash table enables efficient and rapid retrieval of relevant matches during the search process. This approach significantly reduces the computational complexity and improves query response times. The complexity of a research operation is then reduced to O(1) which is constant in time and with this approach we can have a response in just 3 seconds which represents a big improvement.

The Quick Search algorithm efficiently matches the structure of input paragraphs to precomputed structures in the Wasabi dataset using a hash map. For each paragraph from the query, the algorithm computes its structure (word count per line) using the structure_paragraphe function and retrieves matching entries directly from the hash map. It handles both DataFrame inputs, where song lyrics are processed paragraph by paragraph, and string inputs, splitting text into paragraphs. The hash map enables rapid retrieval with O(1) lookup time per structure, significantly reducing computational costs. Results are returned as a DataFrame, including metadata such as song titles and matching paragraphs, ensuring flexibility and scalability for large datasets.

Example of an output from the quick search:

	title	name	paragraphe	word_count_vect	Chanson	Paragraphe
0	Magic Flame	Almah	a decision can change all your ways\nyour beli...	[7 7 5 6 7]	undifined	people gonna hate let them do it\nshine like i...
1	I Think I'm Gonna Like It Here	Ann Reinking	no need to pick up any toys\nno finger will yo...	[7 7 5 6 7]	undifined	people gonna hate let them do it\nshine like i...
2	Mr. Superlove	Ass Ponys	the storm was blowing from the south\nthe bloo...	[7 7 5 6 7]	undifined	people gonna hate let them do it\nshine like i...
3	Yo Te Amo(I Still Love You)	Aventura	baby im sorry te juro im sorry \nmira me a los...	[7 7 5 6 7]	undifined	people gonna hate let them do it\nshine like i...
4	Yo Te Amo(I Still Love You)	Aventura	baby im sorry te juro im sorry\nmira me a los ...	[7 7 5 6 7]	undifined	people gonna hate let them do it\nshine like i...

3.4. Word2Vec: Transforming Lyrics into Semantic Vectors

The **Word2Vec** model plays a central role in this project by enabling the transformation of song lyrics into numerical vectors, thereby capturing the semantic relationships between words. Unlike traditional keyword-based approaches, Word2Vec allows us to analyze the contextual and semantic similarity between lyrics, which is crucial for retrieving relevant songs based on partial queries.

Objective

The primary goal of using Word2Vec is to map words from song lyrics into a high-dimensional vector space. In this space, words with similar meanings or contexts are positioned closer together. This enables the system to measure similarity between lyrics based not only on exact matches but also on contextual and semantic relationships, enhancing the accuracy of song retrieval.

Training

To train Word2Vec effectively and adapt it to the specific context of song lyrics, we employed two distinct datasets:

- **Wasabi Dataset**:
  - Focused on capturing context specific to musical lyrics.
  - Training Word2Vec on Wasabi helps the model understand word relationships unique to the music domain, such as rhymes, common expressions, and lyrical structures.
  - The hyperparameters we use for this task are the following:
    - vector_size=100: The size of the word embeddings.
    - window=5: The context window for considering neighboring words.

- min_count=1: Retain even rare words in the vocabulary.
- sg=1: Use the Skip-Gram algorithm, which focuses on predicting context words from a given target word.
- workers: Utilize all available CPU cores for parallel processing.
- compute_loss=True: Monitor training loss for better evaluation.

❖ **Google News Dataset**:
- ➢ Offers a broader linguistic context for comparison.
- ➢ By leveraging this pre-trained model, the system benefits from a general understanding of word relationships that extend beyond the music domain, complementing the domain-specific training on Wasabi.

Purpose

The Word2Vec model enhances the lyrics retrieval system by:

- **Capturing Semantic Relationships:** It recognizes word similarities and differences based on their usage in lyrics, enabling robust query matching.
- **Improving Retrieval Accuracy:** The embeddings allow for efficient computation of similarity scores (e.g., Cosine Similarity), ensuring the most relevant results are ranked higher.

3.5. Compute Match

The Compute Match component is the final step in the lyrics retrieval system. Its role is to refine the results obtained from the Quick Search by evaluating the similarity between the user query and candidate songs. This process ensures that the most relevant songs are ranked and displayed to the user

How It Works

The **Compute Match** process is a crucial stage in the lyrics retrieval pipeline. It refines the results obtained from the **Quick Search** step by evaluating the similarity between the user query and the candidate songs. This process involves three key steps:

a. Apply Distance Measures

For each candidate song returned by the Quick Search, a set of similarity or distance measures is applied between the query and the candidate. These metrics quantify how closely a song's lyrics align with the input query, ensuring the flexibility to handle diverse variations. The metrics employed are **Cosine Similarity**, **Jaccard Distance, Levenshtein Distance** which will be more detailed in the next section. These diverse metrics allow the system to handle both **semantic variations** (e.g., synonyms or rephrasing) and **structural differences** (e.g., spelling errors or word order changes) effectively.

b. Calculate Similarity Scores

The results of the distance measures are converted into normalized similarity scores for each candidate song. These scores reflect how well the lyrics of a song match the user's query:

- **Higher Scores**: Indicate stronger alignment with the query, signifying that the song is likely a relevant match.
- **Lower Scores**: Suggest a weaker connection to the query, pushing the candidate further down in the ranking.

c. Rank Songs

The computed similarity scores are used to rank the candidate songs. Songs with the highest scores are displayed at the top of the results, ensuring that the most relevant matches are presented first. This ranking simplifies the user experience by highlighting the best matches and reducing the effort required to find the desired song.

3.6. BagOfWords

In our project, the Bag of Words (BoW) model was initially used to calculate similarities between song lyrics and user queries.

BoW served as a foundational approach for numerically representing text data. This involved transforming song lyrics and user queries into structured formats, enabling efficient similarity calculations.

Training

The BoW model was implemented using the **CountVectorizer** from the scikit-learn library, which tokenized the text, created a vocabulary of unique words, and generated a word occurrence matrix. The lyrics and queries were preprocessed through cleaning and tokenization before being transformed into this structured numerical format.

Purpose

The primary purpose of using the BoW model was to represent text as numerical matrices that capture word occurrences or presence, allowing for efficient comparison of textual content. Specifically, we used the BoW representation to calculate **Jaccard Similarity**, which measures the overlap between sets of words in song lyrics and user queries. This helped identify the most relevant matches by quantifying the shared and unique words between the query and the lyrics.

Contributions of the Bag of Words Model

1. **Text Representation**: BoW converted our preprocessed text data (lyrics and queries) into a numerical matrix capturing word frequency or presence.
2. **Enabling Similarity Calculations**: The BoW representation facilitated the application of metrics like **Jaccard Distance**. This metric compared the BoW representations of the lyrics and queries, identifying the most relevant matches based on words.
3. **Preprocessing for Text Analysis**: BoW streamlined textual data into a consistent, analyzable format, which was particularly valuable for processing large datasets like the Wasabi dataset. This avoided the inefficiencies of analyzing raw text directly.
4. **Flexibility Across Similarity Metrics**: The BoW model provided a versatile framework for leveraging the Jaccard similarity metric which assessed the overlap of unique words between lyrics and queries.

The Bag of Words model was instrumental in enabling Jaccard similarity calculations by capturing the structure of texts. While it played a crucial role in the early stages of our system, it has since been replaced by more advanced approaches better suited to the project's evolving needs. This progression reflects the continuous improvement of our methods to achieve more nuanced and effective similarity computations.

4. Similarity Measures

In the development of the Lyrics Retrieval System, similarity measures play a critical role in assessing the relevance of stored song lyrics to a user's input query. Each measure evaluates the relationship between the query and the lyrics database using different approaches, ensuring a comprehensive analysis of potential matches.

4.1. Cosine Similarity

Cosine Similarity measures the cosine of the angle between two non-zero vectors in a multi-dimensional space, capturing the textual similarity of documents. It is particularly useful for determining how closely related two sets of text are in terms of their word distribution, irrespective of their length.

Formula:

$$Cosine\ Distance = 1 - \frac{A.B}{||A|| * ||B||}$$

Cosine Similarity excels in handling high-dimensional data and is widely used in information retrieval due to its efficiency and effectiveness in capturing semantic similarity, even when the input query uses different word orders or phrasing compared to the stored lyrics.

Cosine similarity is widely used in text and document analysis for tasks like classification, clustering, and semantic search. It also plays a key role in recommendation systems, image and video recognition, and

user profiling. Beyond these, it finds applications in healthcare, social network analysis, and personalized e-commerce solutions.

4.2. Jaccard Distance

Jaccard Distance is a metric that measures the dissimilarity between two sets. Unlike Cosine Similarity, which considers word frequency and magnitude, Jaccard Distance focuses purely on the overlap of unique sets. It is calculated as one minus the Jaccard Index, which represents the ratio of the intersection of two sets to their union.

Formula:

$$Jaccard\ Index\ = \frac{|A \cap B|}{|A \cup B|} \qquad Jaccard\ Distance\ = 1\ -\ Jaccard\ Index$$

This metric is particularly effective for identifying shared content between short phrases, making it suitable for matching query lyrics when users provide only fragments. However, it is sensitive to minor variations in wording, as it does not account for semantic relationships.

Jaccard's distance is an intuitive way of measuring the similarity of texts based on shared words. It's simple to implement and useful for comparing sets of shared words.

The Jaccard similarity is primarily used in applications involving binary or categorical data. It is widely applied in recommendation systems for collaborative filtering, genomic analysis to measure similarity between genetic sequences, and data deduplication to identify and remove duplicate records.

4.3. Levenshtein Distance

Levenshtein Distance, also known as Edit Distance, quantifies the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. It provides a fine-grained comparison between strings, making it ideal for detecting close matches in the presence of typographical errors or slight variations in lyrics.

Example:

Transforming “hello” to “hallo” involves one substitution, resulting in a Levenshtein Distance of 1.

While Levenshtein Distance is computationally more intensive than Cosine Similarity or Jaccard Distance, its ability to capture character-level differences makes it invaluable for accommodating user queries with minor inaccuracies.

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	5	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

Matrix showing the functioning of Levenshtein Distance

Levenshtein distance measures the difference between two sequences by counting the minimum number of edit operations (insertions, deletions, or substitutions) required to transform one sequence into another. It is widely used in fuzzy search to find approximate matches in search engines, as well as in bioinformatics for comparing DNA or protein sequences. While effective for character-level comparisons, it does not account for semantic meaning and can be sensitive to string length or fail to handle transpositions well.

Each similarity measure contributes uniquely to the Lyrics Retrieval System:

- **Cosine Similarity** emphasizes semantic relationships and is well-suited for identifying contextually relevant matches.
- **Jaccard Distance** is ideal for assessing overlap between sets of words, effective for short and fragmented queries.
- **Levenshtein Distance** ensures robustness to spelling errors and textual variations, enhancing the system's flexibility and user-friendliness.

By leveraging these complementary techniques, the system achieves a balanced and accurate retrieval mechanism, optimizing the user experience in discovering songs.

5. Results and Analysis

5.1. Method Comparisons

Three similarity methods were compared in this project: **Jaccard Similarity**, **Levenshtein Distance**, and **Cosine Similarity**. Each metric brought unique capabilities to the task:

- **Jaccard Similarity** focuses on the overlap of unique words in two texts. It is effective for identifying lexical matches but fails to consider word order or semantic relationships.
- **Levenshtein Distance** measures the minimum number of edits (insertions, deletions, substitutions) required to transform one text into another. This method captures structural similarities, offering better results for text with shared sequences.
- **Cosine Similarity** utilizes vector representations to compare texts based on their semantic relationships. While computationally intensive, it captures context and meaning more effectively than the other methods.

These methods collectively illustrate the trade-off between speed, accuracy, and depth of analysis in NLP tasks.

5.2. Case Study

To illustrate the behavior of these methods, we applied them to a verse from the song *Imagine* by John Lennon:

you may say i'm a dreamer
but i'm not the only one
i hope someday you'll join us
and the world will be as one
Imagine - John Lennon

	Jaccard	Levenshtein	Cosine
Similarities	**0.25**	**0.45**	**0.87**

Paragraph	oh i wanna be your doctor oh may not have a phd but you can come and sit and talk with the lover in me	don't cry for me my son because i'm not the only one where i go you can't follow this journey i will make all alone	There are so many lonely people Why must they ever be apart I hope someday you'll be together Singing these words with all your heart

- **Jaccard Similarity** identified verses with the highest word overlap, which included the original song and its covers. However, the method sometimes retrieved unrelated texts that shared a few common words but lacked thematic relevance.
- **Levenshtein Distance** performed better by preserving word order, retrieving results that aligned more closely with the structure of the original verse.
- **Cosine Similarity** delivered the most semantically relevant results, retrieving verses with similar themes such as unity and hope, even when lexical overlap was minimal.

This case study highlights how different similarity measures excel in distinct contexts, depending on the type of similarity sought.

5.3. Evaluation Metrics

The evaluation of similarity measures was based on the **top-k reference hit ratio**, which assesses how often the ground truth paragraph appears within the top-k ranked results. For this analysis, we focused on top-5 and top-10 rankings to determine the relevance and accuracy of the retrieval.

Key Observations:

- When the **ground truth paragraph** was present in the Wasabi dataset, it was consistently retrieved in the **top-1** position across all similarity measures, including **Cosine Similarity (Word2Vec)**, **Jaccard Similarity**, and **Levenshtein Distance**.
- Additionally, all **covers of the same song**, where the lyrics were unchanged, or slightly changed by different artists, were retrieved alongside the original song, appearing in the top-5 and top-10 results. This highlights the robustness of the methods in identifying exact matches and near-duplicates.

Evaluation Function:

The comparer_distances function was designed to compute and rank the **top-k most similar paragraphs** for each distance measure.

The output for each distance measure included:

- The **Title** and **Artist Name** from the dataset.
- The **Similarity Score** and corresponding text of the retrieved paragraphs.

This evaluation confirms the system's consistency and effectiveness in identifying the original ground truth and its duplicates, ensuring accurate rankings for cases with unchanged lyrics.

5.4. Observations

The results of the similarity measures revealed several key insights:

1. **Ground Truth Retrieval**:
    - When the ground truth paragraph was present in the Wasabi dataset, it was consistently ranked in the **top-1** position across all similarity measures.
    - Covers of the same song, where the lyrics were unchanged by different artists, were also accurately retrieved and ranked within the **top-5** and **top-10** results.
2. **Differences Between Metrics**:
    - **Jaccard Similarity** excelled at identifying paragraphs with high lexical overlap, focusing on the presence of shared unique words. However, this method does not account for word order or contextual meaning.
    - **Levenshtein Distance** performed well in capturing structural similarity by rewarding matches with minimal edits, such as similar word sequences, but struggled with semantic nuances.
    - **Cosine Similarity (Word2Vec)** demonstrated the best semantic alignment, retrieving paragraphs with similar meanings, even if the lexical overlap was minimal.
3. **Consistency of Results**:
    - For exact matches, all three methods reliably placed the ground truth in the **top-1** position, showing the robustness of the system for cases where lyrics were unchanged.
    - However, for paragraphs with rephrased or semantically similar content, **Cosine Similarity** often provided rankings that were difficult to interpret. In many cases, the semantic link identified by the metric was unclear, highlighting the challenge of explainability in vector-based similarity measures.

These observations underline the strengths and limitations of each similarity measure, emphasizing the need for improved semantic models to enhance explainability and accuracy, particularly when dealing with contextually complex content.

5.5. Results on Jamendo Database

We have conducted a comprehensive search to identify the top 5 similar paragraphs using three distinct distance measures for each English song from the Jamendo dataset. The findings are documented in the file named similarity_results_jamendo_top5.txt. It is important to note that none of the songs from the Jamendo dataset appear in the Wasabi dataset. This absence was confirmed through manual searches using the *WASABI Interactive Navigator*.

Given that no direct matches were found in the Wasabi dataset, we extended our search to include well-known songs such as “Imagine” to verify the accuracy and relevance of our program’s results. This step was crucial to ensure that our algorithm is functioning correctly and producing logically consistent outcomes.

6. Areas for Improvement

While the current system delivers promising results, there are several areas where improvements can be made to enhance both accuracy and efficiency:

a. Use of Sentence Embeddings

The current implementation uses Word2Vec to generate word-level embeddings, which are averaged to represent entire paragraphs. While this approach provides a basic semantic understanding, it has significant limitations:

- **Lack of Contextual Representation:**
  Word2Vec embeddings do not consider the context in which a word appears. For example, the word "light" in "lightweight algorithm" and "light bulb" would have the same vector representation, potentially leading to semantic mismatches.
- **Loss of Sentence Structure:**
  Averaging word embeddings disregards the order and syntactic relationships of words in a paragraph, which are often crucial for capturing deeper meaning.

Proposed Solution:
Adopting sentence embeddings generated by advanced models like SBERT (Sentence-BERT) could address these issues. These embeddings capture both the context of each word and the overall meaning of the sentence or paragraph. They are designed specifically for tasks like semantic similarity and ranking.

Challenges:
However, integrating sentence embeddings such as BERT into our system comes with significant computational overhead:

- High Computational Cost:
  BERT-based models require processing every paragraph in the dataset to compute embeddings, which would involve recalculating embeddings for the entire hashmap. Given the size of our dataset, this preprocessing step would be computationally expensive and time-consuming.
- Storage Requirements:
  The resulting sentence embeddings for millions of paragraphs would require substantial storage capacity.

Balanced Approach:
Instead of replacing Word2Vec entirely, we could explore hybrid methods, such a exploring lighter alternatives like FastText to reduce computational load while improving contextual understanding.

b. Limitations of the Current Quick Search

The quick search mechanism, which is used to preselect candidate paragraphs for similarity comparison, has notable limitations:

- Strict Filtering Criteria:
  The current implementation requires exact alignment in word count or structure, which may exclude relevant candidates. For example, even a small difference in the number of words between a query paragraph and a candidate can prevent it from being considered.
- Reduced Recall:
  By relying heavily on heuristic-based rules, the system risks eliminating paragraphs that, while not identical, may still be semantically or contextually similar.

Proposed Solution:

- Allow small variations in word count or structure. Approximate matching can ensure that relevant paragraphs with minor differences are included.
- Adjust the filtering criteria based on the characteristics of the query paragraph, such as its length or lexical richness.

By addressing these limitations, the quick search can become more flexible and inclusive, ensuring that relevant paragraphs are not prematurely excluded while keeping computational costs manageable.

These improvements focus on balancing accuracy and efficiency. While adopting sentence embeddings like BERT can significantly improve semantic understanding, careful consideration of computational feasibility is necessary. Similarly, refining the quick search process ensures a better recall rate without overburdening the system.

7. Conclusion

In this project, we explored the use of different similarity measures: Cosine, Jaccard, and Levenshtein, to identify similar paragraphs within a large dataset. The results showed that these methods consistently retrieved the ground truth, which always appeared in the top-1 position when present in the database. However, their strengths and limitations became evident when dealing with more complex cases involving semantic or structural variations.

Cosine similarity, based on Word2Vec, performed best in capturing semantic links, although it occasionally produced results that were difficult to explain. Jaccard and Levenshtein distances proved effective for detecting lexical overlaps or structural similarities but did not account for the meaning of the paragraphs.

To improve the system, we identified two main areas: integrating sentence embeddings to better capture semantics and making the quick search phase less restrictive. Using models like BERT could enhance the results, but the computational and storage costs of precomputing all embeddings remain a challenge. Using lightweight embeddings for quick search and more advanced models for fine-tuning the results, could offer a good balance between precision and efficiency.

In conclusion, this project highlights the importance of tailoring similarity measures to the specific requirements of the task. With these improvements, the system could achieve greater accuracy and flexibility while handling even more complex datasets effectively.

8. References
IEEE FORMAT

- [1]OpenAI, “ChatGPT,” *OpenAI*, 2024. https://openai.com/chatgpt
- [2]R. Řehůřek, “gensim: topic modelling for humans,” *radimrehurek.com*. https://radimrehurek.com/gensim/
- [3] “langdetect,” PyPI. [En ligne]. Disponible : https://pypi.org/project/langdetect/
- [4]Scikit-Learn, “scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation,” Scikit-learn.org, 2019. https://scikit-learn.org/
- [5] WASABI, “WASABI,” WASABI - Music Analysis and Retrieval Research Group. [En ligne]. Disponible : https://wasabi.i3s.unice.fr/
- [6]f90, “GitHub - f90/jamendolyrics: Jamendo music dataset with time-aligned lyrics for lyrics alignment evaluation,” GitHub, Nov. 15, 2023. https://github.com/f90/jamendolyrics (accessed Nov. 25, 2024).
- [7]f90, “jamendolyrics/lyrics at master · f90/jamendolyrics,” GitHub, 2019. https://github.com/f90/jamendolyrics/tree/master/lyrics (accessed Nov. 25, 2024).
- [8]“Word2Vec Tutorial - The Skip-Gram Model · Chris McCormick,” mccormickml.com. https://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/