**SOFTWARE DESIGN DOCUMENT**

**INTRODUCTION**

**Purpose:**

The purpose of this software design document is to describe, in depth, the software application’s architecture. This document can also be treated as a reference by the user who wishes to understand the different steps or modules that form the retrieval system.

**Scope:**

The application is a Boolean retrieval system that retrieves music ID’s from a dataset of songs (addressed by unique IDs), on inputting a query of n words. The application returns ten music IDs corresponding to the top ten songs most similar to the given list of words. The code for the application is written in Python 3, and the Natural Language Processing Toolkit package has been used for stemming words using Porter’s algorithm.

**DESIGN ARCHITECTURE**

The architecture of the retrieval system so designed can be broadly divided into the following modules.

1. **Stemming the query words:**

The first part of designing the application was to understand the dataset. The dataset (<https://labrosa.ee.columbia.edu/millionsong/musixmatch>) was found to be a bag of words where all words were stemmed using Porter’s algorithm and each document (each document corresponds to a song) was addressed by two unique identification numbers (document ID and music ID). Furthermore, the frequency of each term in all the documents that contain that term was explicitly specified next to its two IDs.

Understanding the algorithm was necessary to be able to apply the same on all the query tokens. The query words were then stemmed using the same algorithm.

1. **Making the Inverted Index:**

The next step was to make a dictionary in Python where keys corresponded to the stemmed tokens (now called terms) and each key was associated with a list of tuples. Each tuple corresponded to a document and was included in the postings list of the term that it contained. The tuple was of the form:

[ (Document ID, Music ID, term frequency) ].

**Data structures used:** Dictionary in Python, where keys are the terms and the value corresponding to each key is a list of lists (The innermost list is a tuple).

1. **Making lists of documents:**

The Inverted Index and the query tokens(as a list of stemmed words) are then inputted into a function that returns a list of lists, where the lists contain lists of document IDs that have all possible combinations of n words of the query in common, (n-1) words of the query in common, and so on to lists of documents that have one word of the query. The list of lists so returned are accepted into the ranking module, which ranks document IDs in the order of relevance.

**Data structures used:** A list of lists is returned.

1. **Ranking the list of documents:**

The documents are ordered in the order of ‘similarity’. Here, for a given query, one document is said to be more similar than the other if more number of query terms are contained in it. For two documents having the same of query terms, the document having a greater frequency corresponding to the lowest frequency term in each list is given greater priority. Consider two documents. One has w1, w2, w3 frequecies = [4,5,5] and the other has frequcncies corresponding to the words = [7,2,8]. Then document 1 is said to be more similar because its minimum (5) exceeds the minimum of document 2 (2).

1. **Returning the top ten documents:**

After ranking the documents in the order of similarity explained above, the top ten documents are outputted on the screen.