|  |
| --- |
| Music Recommendation Service |
|  |
| Implementation Report |

|  |
| --- |
| By Brian Davis  12-4-2020 |

Table of Contents

[Introduction 2](#_Toc56869691)

[Problem Analysis 2](#_Toc56869692)

[Solution Requirements 2](#_Toc56869693)

[Non-functional Requirements 2](#_Toc56869694)

[Functional Requirements 3](#_Toc56869695)

[Implementation of Solution 3](#_Toc56869696)

[Program Execution 3](#_Toc56869697)

[Program Structure Flowchart 3](#_Toc56869698)

[Reflection 3](#_Toc56869699)

[Dataset Loading 3](#_Toc56869700)

[Similarity Metrics 3](#_Toc56869701)

[Module Files Pseudocode 4](#_Toc56869702)

[Load dataset module 4](#_Toc56869703)

[Similarity module 4](#_Toc56869704)

[Appendix 5](#_Toc56869705)

[Functions not defined in the main paper 5](#_Toc56869706)

# Introduction

With the growing emergence of online content as a service to fulfil client needs, there is need for the implementation of a Music Recommendation Service, to allow potential customers and clients to be able to find music similar to their own taste, to help them in their discovery of new content to consume. The Music Recommendation Service Program outlined in this report aims to meet these goals and exceed client expectations with capability to enable the comparison of song features through a wide range of similarity metrics. The MRSP will do this through successful reading of the file structure, appropriate assignment through suitable data structures, alongside key comparison of suitable features collected from the provided data.

# Problem Analysis

To create the solution, the problem needs to be broken down into steps. The steps needed to solve the problem are.

1. Loading and parsing of the music dataset file
2. Computing similarity between artists
3. Computing similarity between music tracks

To accomplish step 1, it is required to first explore the file itself to understand the data within and consider any issues with the file/s before proceeding with any implementation. The first thing noticed here is that the file will require encoding at the utf-8 level due to some symbols found within such as ‘&’. Another issue found within the file is that the columns that contain names have the comma (,) symbol within their values. This is a potential problem due to the nature of the data file, which is **comma separated csv file** and thus the comma found within these values will result in incorrect splitting of the data. A suitable regex will help to resolve this problem through the detection of a string before a comma, which will then be replaced with a /.

Moving on to steps 2 and 3, these problems are similar, as the expected outcome of the data structure for this solution, dictionary, will contain similar features. The main difference here will be the inclusion/exclusion of the values of artist names. There is consideration to include some additional artist features for the artist features dictionary, but these have not been included to create a more concise program for the end user. A solution under consideration would be to include the ability for a user to find and compare a specific feature of two artists that they choose (search for) and retrieve the feature for every song that the artist has been part of, compiled into a single dictionary to allow for comparison using whatever metric the user chooses.

Another problem needed to solved is what if the user doesn’t know the name of the song or the name of the that they want to compare. This can be solved by allowing the user the ability to search for a song using a word matching service, which will find thew word they enter and show the user every song including the ID of each song that has at least 1 word they enter in it’s name. This will allow the user to find the song or artist they are looking for to provide comparison.

# Solution Requirements

This section outlines the characteristics of the solution, and how these characteristics enable the program to meet the needs of the stakeholders and the business.

## Non-functional Requirements

* Usability
  + Efficiency of use
  + Intuitiveness
  + Low perceived workload
* Security
* Reliability (lack of errors)
* Performance
* Availability
* Scalability

## Functional Requirements

* Use case specification diagram (textual)
* Design diagrams
  + Wireframes
  + Mock-ups
  + Design prototypes

# Implementation of Solution

# Program Execution

# Program Structure Flowchart

# Reflection

## Dataset Loading

## Similarity Metrics

# Module Files Pseudocode

This section outlines the pseudocode for each module created. These have been split into three sections, one for each module.

### Load dataset module

Function artist music / music features

With open the file with the name data.csv, in read mode, with encoding utf8

Create a new dictionary for the result

Create an index that starts at 1

Use the next keyword to skip the headers

For each line in the file

Using regex, substitute the comma if it is preceded by a string, to a /

Create a new variable assigned to the column for artist names

Remove the square brackets and escape characters from those names

Reassign it to the artist names variable

Create another empty dictionary d

Assign each column to a new key in the empty dictionary

Assign the d dictionary to the result dictionary, using index as the id

Increment the index number by 1

Return the result dictionary

Close the file

### Similarity module

Function similarity metric (take 3 positional arguments of dictionary, id1 and id2)

Take id numbers if not included in the parenthesis when the module is called

If the id numbers match, then stop the program and prompt the user

Else ask the user for a specific feature they want to compare, with a small message asking the user to enter Yes if they have created their own artist dictionaries (defined in the function search artist)

If the user enters the value of no, or leaves the response empty

Compare all features defined in the dictionary

Create two new lists to be able to compare each feature one by one

Create a list for the key names

Loop through the range of 0 to 9 excluding 9

For each value in the list of features in list 1 and list 2

Use the metric to compute the distance between the values

Print the result to the user, the program will terminate here

If the user enters yes, and the length of the dictionary is 2, this must be a created artist dictionary

Create empty lists for the x and y variables

For each value in dictionary id1 and id2

Append the lists x and y

Compute the distance metric and return/print the result

If the user entered a valid feature

Assign features to x and y

Compute the distance metric and return/print the result

# Appendix

### Functions not defined in the main paper

Function search artist (takes 1 positional argument of dictionary name)

Take the first name from the user as a string

Take the second name from the user as a string

Take the feature name from the user as a string

Create the empty dictionary, and an empty list

For increment I in the range of 1 to the overall length of the artist features dictionary

If the names entered are in the dictionary at the key for artist names

Print the result to the user

Append the feature from the song that was matched with the artist

If the length of the list result is empty

Return nothing

Else the dictionary takes the first name and surname initial as the new key and takes the results from the appended list

Return the dictionary

Function search song (takes 1 positional argument of dictionary name)

Take the input from the user of a word that they want to find from the song they are looking for

Strip away any whitespace at the end of the input

Split the values of the input by the space to create a list of words

If the length of the input equals 1, then we must have only one word as out input

Join the input word back together so it removes it from a list

For increment I in the range of 1 to the length of the dictionary being searched

Print the matching results to the user

Append this to a new list (not currently done, might not be needed)

Else there are more than 1 word in the entry

For increment I in the range of 1 to the length of the dictionary being searched

Capitalise each word in the list

Increment through and check each word in the list against the dictionary

Print the matching results

Append this to a new list (not currently done, might not be needed)