Leveraging User's Mood and Song Popularity for Music Recommendation

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

Master of Technology

by

Aritra Saha
Roll No.: Y8127125

under the guidance of
Dr. Arnab Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
May, 2014

CERTIFICATE

It is certified that the work contained in this thesis entitled "*Leveraging User's Mood and Song Popularity for Music Recommendation*", by *Aritra Saha (Roll No. Y8127125)*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Arnab Bhattacharya)
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur
Kanpur - 208016

May, 2014

Abstract

Music ethusiasts often prefer to listen to music as per their current mood. Also, a song's popularity plays an important role as a user might be interested in listening to a popular and trending song which might not get along with her musical taste or mood.

The proposed system tries to profile its users' musical tastes by paying attention to what they listen on their devices and is scrobbled by Last.FM. The mood of a user is profiled as well by analysing the recent few songs heard. Recommendations are made based on what other users with similar musical taste have heard right after when they had a similar musical mood in their history. Recommendations are made taking into account the song popularity as well as the user's current mood giving them appropriate weightages over each other.

Dedicated to my parents.

Acknowledgement

Since no one walks alone on the journey of life and no work can be accomplished without the contributions from various people around, where do you start to thank those that joined you, walked beside you, and helped you along the way and now when I stand at this stage of completion of my thesis work, I would like take this opportunity to express my sincere gratitude towards each and every one of them.

I would like to express my sincere gratitude towards my thesis supervisor, Dr. Arnab Bhattacharya, for his constant support and encouragement. I am grateful for his patient guidance and advice in giving a proper direction to my efforts. I thank the Department of Computer Science and Engineering, IIT Kanpur, for providing the necessary infrastructure and a congenial environment for research work.

A big thank you to all my department friends and my wing mates for making my last year of stay in campus a fun one. Last, but not the least, I thank my parents for always being there for me.

Aritra Saha

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recommendation systems is an extensive class of web applications that involve predicting user responses to options. It is a subclass of information filtering system. The main purpose of recommendation systems is to generate plausible options to users for items or products of their interest. These systems build up the user profile usually based on their past search history or ratings and compares it with some reference characteristics.

Recommendation systems are broadly categorized on the basis of how they analyze data sources to develop notions of affinity between the user and the items which can be used to identify well-matched pairs. They are:

- Content-based: The user's profile, created with their preferences and any available history, is compared with various candidate items and the best matching items are recommended.

- Collaborative-based: Large amounts of information on several users' preferences and activities are analyzed to obtain the similarity among users and thus predicting the future interests. Collaborative filtering is capable of recommending complex items without requiring its "understanding" accurately.

- Hybrid: This approach involves both content-based and collaborative-based filtering while recommending.

Recommendation systems are of great importance for the success of e-commerce and the IT industry and are gradually gaining popularity and becoming an active area for research. They enhance user experience by assisting them in finding information and reducing search and navigation time. In addition, recommendation systems increase productivity and creditability of a user. These systems have evolved to fulfill the natural dual need of buyers and sellers by automating the generation of recommendations based on data analysis.

Recommendation systems are gaining popularity in the field of music as well. One of the well known example is *Pandora Internet Radio*[1], which is an automated music recommendation service. Music recommendation systems ask for users ratings, like or dislike for particular artists, songs or albums, and based on these parameters it recommends choices closer to their taste.

1.1 Motivation

Most of the modern recommendation systems we come across use hybrid filtering, i.e., take into consideration a user's choices and cross-reference it against large amounts of user data finding similar interests. Even though this method works fine for some cases, there is a scope for improvement.

Recommending music to a user based on her pre-set choices may not be a success if she is in a totally different *mood*. We can determine the *mood* of a user by analyzing her recent history. The recent history considered to predict the mood is kept rolling forward, so as to accommodate the fact that a user's mood changes with time and music. This mood cross-referenced with content and collaborative filtering tend to give better recommendations.

[1] `http://www.pandora.com/`

1.2 The Problem

The mood of a music enthusiast can be determined by the songs she has heard recently. The mood usually changes gradually over time and can be modeled by the genre of the songs. Mood once modeled can be compared with those of other similar users and thus help conclude the preference of a song after a set of given songs (which determine the mood) as heard by other "similar" users.

1.3 Contributions of this Thesis

There are two main contributions of this thesis. The first is to determine the rolling current mood of a user based on her recently played tracks. The challenge here is to optimize the monitored rolling time duration, to come up with a method to compare two set of moods a set of weights for each of the contributing factor, i.e., mood, user's preferences and collaborative filtering to obtain a confidence score for each recommendation at any given instant. The second is to build a tool that not only shows the recommendations but also lets you customize the determining weights and parameters of the recommendation engine.

1.4 Organization of this Thesis

The rest of this thesis is organized as follows. Chapter 2 presents related work done. Chapter 3 discusses the various background work that one must be acquainted with in order to understand the work presented. Chapter 4 discusses the data sources and APIs used to set up a sample infrastructure of the recommendation system. Chapter 5 then discusses in detail the implemented methods and algorithms. Chapter 6 presents a summary of the results that was achieved and also talks about what can be done further.

Chapter 2

Related Work

There is a lot of interesting product and implementation related projects in the field of music recommendation. There are two primary ways to categorize and identify similar music, either by analyzing and mathematically formulating the audio signals or by mining from a plethora of available music related metadata. Some of the well known projects have been introduced below.

2.1 Pandora Radio

Pandora Radio[1], one of the most popular music recommendation and discovery services on the Internet today, bases its recommendations on data from the Music Genome Project (2.1.1) [6]. It uses *musicological analysis* [9] form of recommendation. Pandora has no concept of genre, user connections or ratings. When a user listens to a radio station on Pandora, it uses a pretty radical approach to delivering users personalized selections; having analyzed the musical structures present in the songs one likes, it plays other songs that possess similar musical traits.

An important aspect of Pandora is its feedback system. This allows users to like or dislike a presented song. Pandora makes efficient use of proximity measure algorithm [12] to recommend music from its database that matches users choice. Based on this, Pandora then recommends music and adapts its recommendations to

[1] `http://www.pandora.com/`

match the users taste.

2.1.1 Music Genome Project

The Music Genome Project[2] [3] assigns a vector of up to 400 'genes' (or attributes) to every song. These attributes capture the musical identity of a song and many other significant qualities that are relevant to understand the musical preferences of listeners. These 'genes' correspond to attributes of the track such as *gender of lead vocalist*, *type of background vocals*, *level of distortion*, etc. Each determined gene is given a score in the range of 0 to 5, with intervals of 0.5. Given the vector of one or more songs, a list of other similar songs is constructed using a distance function.

The project employs musical analysts who listen to music and rate songs based on those attributes. These analytics then gets imported into Pandora computer analytics system that is presented to the users for their feedback. Pandora takes that feedback and develops playlist metrics and recommends it to the users.

2.2 Modelling Internet Radio Streams

The radio can provide useful data regarding the popularity of a song and those that are trending. Radio usually plays music as per their listener's requests or based on prediction which will increase their user base. Either way, it is a fair source to determine a song's popularity. Internet Radio are no way behind in this regard, but they also happen to provide very structured information regarding the played songs and possibly the upcoming playlist.

Yahoo! attempts to mine this data obtained from several internet radio stations over a considerable period of time [1]. It can then be used to create all sorts of popularity and/or trends related charts for songs, artists, albums etc. Such kind of information often proves vital while recommending songs. A music enthusiast may like to hear a trending song, even if the songs does not match her preferred genre.

[2] https://www.pandora.com/about/mgp

Chapter 3

Background

The algorithms and data structures used in this thesis have been introduced and discussed below.

3.1 Levenshtein Distance

Levenshtein distance [8] or evolutionary distance is a concept from information retrieval. It is one of the most common variants of *edit distance* named after the Soviet Russian computer scientist Vladimir Levenshtein. Edit distance [14] describes the number of edits that has to be made in order to convert one string to another by performing minimum number of operations like insertions, deletions and substitutions. It is the most common measure to expose the dissimilarity between two strings; the greater the distance, the more different the strings are.

This measure allows to assess similarity between strings (or words) and has many applications that include spell-checking, examining correctness of pronunciation and affinities between dialects, analyzing the DNA structure or web mining. The Levenshtein distance can also be computed between two longer strings.

For example, the Levenshtein distance between "contest" and "context" is 1, since just one edit is required to convert one into the other, i.e conte**s**t → conte**x**t (substitution of s by x). Similarly, levenshtein distance between "incubate" and "incubus" is 3, since three edits are required to convert one into another and there is no way to do it with fewer than three edits:

1. incubat**e** → incubat (deletion of e)
2. incuba**t** → incuba**s** (substitution of t by s)
3. incub**a**s → incub**u**s (substitution of a by u)

3.1.1 Definition

Mathematically, the Levenshtein distance between two strings a and b is given by $lev_{a,b}(|a|,|b|)$, where

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and 1 otherwise.

3.1.2 How it works

Firstly, with the help of most common approach called dynamic programming, Levenshtein algorithm calculates the minimum number of operations that are required to convert one string to another. A matrix is initialized measuring in the (m,n)-cell the Levenshtein distance between the m-character prefix of one with the n-prefix of the other word. The matrix can be filled from the upper left to the lower right corner. Each horizontal jump corresponds to an insert and each vertical jump corresponds to a deletion. For each of the operation, the cost is normally set to 1. The diagonal jump can have either cost 0 or 1. If the cost is 0, it means that characters match and if it is 1, it means that characters do not match. The cost is always locally minimized by each cell. In this way the number in the lower right corner is the Levenshtein distance between both words. Figure 3.1 is an example that features the comparison of “meilenstein” and “levenshtein” (where ‘=:’ Match; ‘o : Substitution; ‘+ : Insertion; ‘− :’ Deletion).

The two possible paths through the matrix that produces the least cost solution is described in Figure 3.2 [1].

		m	e	i	l	e	n	s	t	e	i	n
	0	1	2	3	4	5	6	7	8	9	10	11
l	1	**1**	2	3	3	4	5	6	7	8	9	10
e	2	2	**1**	**2**	3	3	4	5	6	7	8	9
v	3	3	2	**2**	**3**	4	4	5	6	7	8	9
e	4	4	3	3	3	**3**	4	5	6	6	7	8
n	5	5	4	4	4	4	**3**	4	5	6	7	7
s	6	6	5	5	5	5	4	**3**	4	5	6	7
h	7	7	6	6	6	6	5	**4**	4	5	6	7
t	8	8	7	7	7	7	6	5	**4**	5	6	7
e	9	9	8	8	8	7	7	6	5	**4**	5	6
i	10	10	9	8	9	8	8	7	6	5	**4**	5
n	11	11	10	9	9	9	8	8	7	6	5	**4**

Figure 3.1: Levenshtein distance example for ‘levenshtein‘ and ‘meilenstein‘

l	e		v	e	n	s	h	t	e	i	n		l	e	v		e	n	s	h	t	e	i	n
o	=	+	o	=	=	=	-	=	=	=	=	or	o	=	o	+	=	=	=	-	=	=	=	=
m	e	i	l	e	n	s		t	e	i	n		m	e	i	l	e	n	s		t	e	i	n

Figure 3.2: Least cost solution to compute Levenshtein distance

3.2 Hungarian Algorithm

The Hungarian method is an algorithm which finds an optimal assignment for a given cost matrix. It is also known as the *Kuhn-Munkres Algorithm*. The original algorithm [7] had a time-complexity of $O(n^4)$; however, it was later modified to obtain a running time complexity of $O(n^3)$ [10].

[1] `http://levenshtein.net/`

3.2.1 Definition

Assuming that numerical costs are available for each of n persons on each of n jobs, the *assignment problem* is the quest for an assignment of persons to jobs so that the sum of the n costs so obtained is as small as possible.

Let $c_{i,j}$ be the cost of assigning the i^{th} resource to the j^{th} task. We define the cost matrix to be the $n \times n$ matrix

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,n} \end{bmatrix}$$

An assignment is a set of n entry positions in the cost matrix, no two of which lie in the same row or column. The sum of the n entries of an assignment is its cost. An assignment with the smallest possible cost is called an *optimal assignment*.

3.2.2 Example

For example, say you have three workers: Jim, Steve and Alan. You need to have one of them clean the bathroom, another sweep the floors & the third wash the windows. Whats the best (minimum-cost) way to assign the jobs? First we need a matrix of the costs of the workers doing the jobs.

	Clean Bathroom	Sweep Floors	Wash Windows
Jim	$1	$3	$3
Steve	$3	$2	$3
Alan	$3	$3	$2

Table 3.1: Cost Matrix for the Hungarian Method example

Then the Hungarian method, when applied to the table below would give us the minimum cost it can be done with: Jim cleans the bathroom, Steve sweeps the floors and Alan washes the windows.

3.3 Trie

Trie [5] is an ordered multi-way tree data structure that is used to store strings over an alphabet. It is a tree data structure that allows string with similar character prefixes to use the same prefix data and store only the tails as separate data. One character of the string is stored at each level of the tree, with the first character of the string stored at the root. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree shows what key it is associated with. Each node contains an array of pointers, one pointer for each character in the alphabet and all the descendants of a node have a common prefix of the string associated with that node. The root is associated with the empty string and values are normally not associated with every node, only with leaves.

A trie can also be used to represent data types that are objects of any type, for example, strings of integers. Various operations such as searching, deletion, insertion etc. can be performed on a trie. One of the advantages of the trie data structure is that its tree depth depends on the amount of data stored in it. Each element of data is stored at the highest level of the tree that still allows a unique retrieval.

Applications of trie may include sotring a predictive text or an autocomplete dictionary like the one found on a telephone. It is also useful in implementating approximate matcing algorithms including those used in hyphenation and spell checking software.

The insertion to and serching in a *trie* has a time complexity of *O(key_length)*, each. The space complexity however, is of *O(alphabet_size * key_length)*

3.4 Cosine Similarity

Cosine Similarity [13] measures the similarity between two vectors of an inner product space by measuring the angle cosine between them. A cosine similarity of 1 indicates 0°, thus having the same orientation, however, a similarity of 0 indicates 90°. Even though the value of the cosine ranges from -1 to 1, the cosine similarity

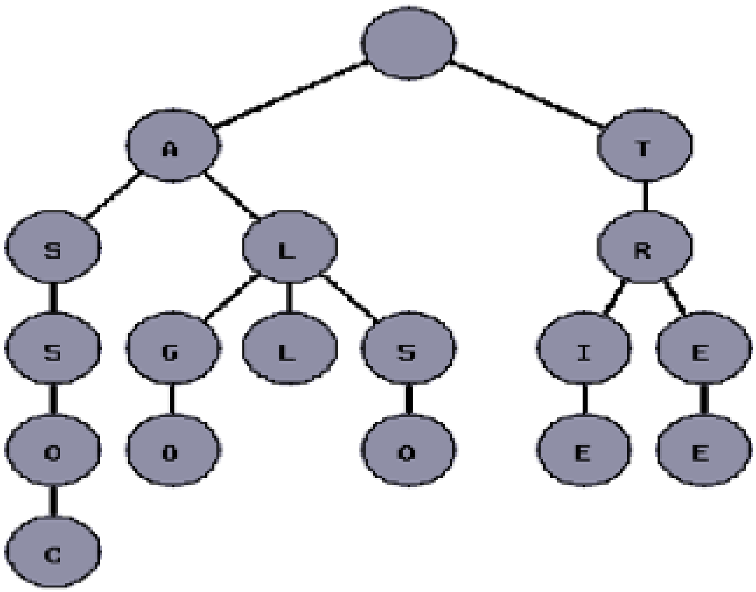

Figure 3.3: Trie with the words “tree”, “trie”, “algo”, “assoc”, “all”, and “also”

is particularly used in the positive space, i.e., $[0, 1]$.

Cosine similarity is commonly used in high-dimensional positive spaces such as in text mining and information retrieval. The popularity of this similarity owes to the fact that it can evaluate very efficiently for sparse vectors as only the non-zero dimensions need to be considered.

3.4.1 Definition

The cosine of 2 vectors can be obtained by the use of the Euclidean Dot Product [13]:

$$a.b = \|a\|\|b\| cos\theta$$

The cosine similarity for 2 vectors A and B, $cos\theta$ is representated by

$$similarity = cos\theta = \frac{A.B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

Chapter 4

Data Acquisition

Several websites contain information about songs, artists and other relevant media descriptions, some of which also includes their signal analysis. In this section, the Million Song Dataset (Section 4.1) is briefly presented and few of its features are described.

Obtaining a considerable amount of user's song listening history is crucial to be able to recommend songs to other similar users. The publicly available Last.fm API (Section 4.2) used for this purpose has also been introduced and its features briefly described.

4.1 Million Song Dataset

The Million Song Dataset[1] is a freely-available collection of audio features and metadata for a million contemporary popular music tracks. The dataset started as a collaborative project between *LabROSA*[2] and *The Echo Nest*[3]. It includes data contributed by other similar communities doing similar work like *Last.fm*[4], *Musicbrainz*[5], *SecondHandSongs*[6], etc.

[1] http://labrosa.ee.columbia.edu/millionsong/
[2] http://labrosa.ee.columbia.edu/
[3] http://echonest.com/
[4] http://last.fm/
[5] http://musicbrainz.org/
[6] http://secondhandsongs.com

4.1.1 Data

It contains [2]:

- 280 GB of data
- 1,000,000 songs/files
- 44,745 unique artists
- 7,643 unique terms (*Echo Nest* tags)
- 2,321 unique *Musicbrainz* tags
- 43,943 artists with at least one term
- 2,201,916 asymmetric similarity relationships
- 515,576 dated tracks starting from 1922

The songs/files are stored in HDF5[7] format to be able to efficiently handle variety of audio features. It contains basic meta-data like *title*, *artists*, *year* of composition, *duration*, *IDs* mapped to other popular song databases (*Last.fm*, *Musicbrainz*, *The Echo Nest*), etc. as well as MFCC (4.1.2) features like *beats*, *danceability*, *energy*, *tempo*, *loudness*, and several other features.

4.1.2 Mel-Frequency Cepstrum Coefficients (MFCC)

The extraction and selection of the best parametric representation of acoustic signals are important tasks in the design of any speech recognition system. It significantly affects the recognition performance. A compact representation would be provided by a set of *Mel-Frequency Cepstrum Coefficients* [15], which are the results of a cosine transform of the real logarithm of the short-term energy spectrum expressed on a mel-frequency scale [11]. The MFCCs are proved more efficient [4].

[7] `http://www.hdfgroup.org/HDF5/`

4.2 Last.fm API

Last.fm scrobbles user music listening activity via plugins installed on the user's devices or directly from the music players. It uses these mined data to evaluate popularity of the songs, compare music history to find similar users and finally recommend songs. Last.fm uses collaborative filtering for recommendation. Last.fm has built an extensive database of songs and its related meta data like artists, albums, genres etc. It also provides several statistical data regarding songs, artists and albums.

Last.fm API[8] has been used to fetch user's song history and the genres of the songs obtained from the Million Song Dataset.

[8] http://last.fm/api

Chapter 5

Implementation

A workflow defined as a graphic summary of the following has been depicted in Figure 5.1.

5.1 Dictionary of Songs

The trie (Section 3.3) data structure is used to load the Million Song Dataset (Section 4.1). Trie not only enables faster searching but also is an ideal data structure for finding the Levenshtein Distances (Section 3.1) on the titles of the songs. The titles of the songs are used as an index for the *trie*. Each node of the *trie* stores with itself some data which is used frequently, for example, its unique track ID using which all other song information can be referenced from the MIllion Song Dataset and artist information. The artist‘s name for a song is stored because it along with the title uniquely identifies a song, and is also used for other API calls in the subsequent steps. The $999,999$ songs (1 song in the dataset was found to be corrupt and hence skipped) from the dataset occupies $6,510,645$ nodes.

5.2 User History

A set of 2614 users with a considerable amount ($\sim$1,000,000 before cleanup) of playback history has been compiled. These users' histories will serve as the basis for obtaining recommendations from similar users using *collaborative filtering.*

Each track from the users' histories is cross-referenced with those in the million song dataset by searching within the above discussed *trie.* Only those tracks which find a match are kept and the others are discarded; as song information would not be available unless a song is identified. Now, the track names scrobbled by Last.fm are often custom written by third party and/or the users themselves, in which case even a slight change in the name will deem it to be discarded. *Levenshtein Distance* has been used to match each of the fetched song. A threshold distance of 3 gives the optimal number of track filtering. Anything lesser would make the filter too strict thus losing too much of valuable user history; whereas keeping it higher will corrupt the data with too many incorrect matches.

For example, if the title of a song in the history was *"Everything All Of The Time"*, and say, the real name as per the million song dataset was *"'Everything All Of Time'*, with a missing *"The"*. Now, this would require a Levenshtein distance of at least 3 for this to be considered; keeping a lower threshold would have lost this track for analysis. Similarly, keeping a higher value of say 4, would lead to match some other irrelevant song but with a similar name, like *"Something All Of Time"*. The titles are striped of all special characters and white spaces before matching.

These histories are stored in files on a per user basis. The format is as follows.

$$139940268000 : TRZR!GS128G932C905$$

where, the first element is the time, in UTC format, at which the song was listened to; the second element is the *song id* as referenced by the million song dataset to get all other information.

A few other information available with the dataset such as *loudness*, *tempo*, *popularity* are also loaded into memory at this step. This is a part of pre-processing of the data and would considerably lower the runtime of generating the recommendations. These features has been described below in Section 5.4.

5.3 Determine Similar Users

A set of 127 commonly recognized genres (often referred to as track tags) has been compiled from the *topstags* as recognized by Last.FM. Each song in a user's history is classified in one or more of these genres and a vector is made out of the counts of each of these 127 genres. This goes forward to define the user's music preferences.

The similarity between users are determined by their music preferences using the vector of genre counts obtained from each of the songs as described in the previous section. The similarity between user X and user Y is calculated on their respective vectors using the *cosine similarity*.

$$similarity(X, Y) = cos(X, Y) = \frac{A \cdot B}{||A||||B||}$$

The top k similar users have been chosen by sorting on each of their respective similarities with the current user. These users' histories can be searched and used to recommend songs for any given user.

5.4 Recommend Songs

5.4.1 Defining Mood

Assuming an average song has a runtime duration of 3 minutes, the mood of a user is determined by the latest 5 songs from her history, i.e., $\sim$15 minutes. It is also assumed that the mood of a user changes gradually with time and the songs she listens to is considerably consistant with her mood over this period. A window of consecutive 5 songs would hereby define the mood of the user roughly for that period.

5.4.2 Collaborative Filtering

A mood as per the current user's recent music history is to be searched in the histories of other similar users (by music preferences as determined in the previous step). A rolling window denoting the mood at that particular point in time is compared to that of the fixed current mood again defined by a window of 5 songs.

The similarity between any 2 windows is determined by the use of the *Hungarian Algorithm* (Section 3.2). The cost matrix used by the algorithm is generated by calculating the similarities (Section 5.4.3) between each pair of songs, one from each of the 2 windows. The algorithm is applied on the thus formed 5×5 matrix.

Popularity of each song is a measure on a scale of 0 to 1. Popularity plays a significant role in the recommendation as users might want to hear new and top of the chart songs even if it does not flare well according to her mood. This information is however stale as of December, 2010.

The song heard right after a given window of 5 tracks is presented as a recommendation with a confidence value. This value is a weighted mean of the similarity score between the two windows and the popularity of the song. The weights for similarity and popularity are set to be 0.8 and 0.2 respectively. This way, each rolling window for each of the similar users are considered, sorted and the top 5 recommendations are presented.

5.4.3 Song Similarity

The following parameters define certain properties of a song and are provided in the million song dataset.

- Loudness: It is measured as the logarithm of the maximum power represented by the song. It is measured in decibels (dB). The loudness of a song in the dataset varies in the range of -100 to 100. A reference value of -60dB can be used to express the absolute loudness.

- Tempo: The average beats per minute of a song contributes to the tempo. This determines the pace of the song, for example, if the song is a slow one or a fast. The tempo of the songs in the dataset varies in the range of 0 to 500.

- Artist: Comparing the artist is crucial, as more often than not, users choose to remain loyal to a certain set of artists. The million song dataset happens to provide a vector of genres for each artist. The vector values are non-negative in nature and denote the count of a set of genres as defined by the dataset. Artist similarity is determined by the use of *cosine similarity* on this vector. A value of 1 indicated maximum similarity, whereas, a value of 0, would indicate that the 2 artists have nothing in common.

These 3 parameters contribute to the determination of the similarity between a pair of songs. A weight of 0.6 has been given to the aritst similarity and a weight of 0.2 each for the normalized absolute difference of both *loudness* and *tempo*.

5.5 Recommendation Engine

The recommendation system had been developed as a tool. Given a username, whose history is already available with the tool, the tool lists the top 5 song recommendations. A user may also modify a few parameters and/or weightages used in the recommendation engine so as to get more custom results. However, a default set of parameters have already been set which in most cases promises to give good results. The tool can be found at `http://aritra.cse.iitk.ac.in/amrs/`.

5.5.1 Tools/Libraries

- Eclipse: An Integrated Development Environment helped speed up the process of coding and its subsequent debugging.
- Java: Being a very common and widely used programming language, loads of documentation and third party libraries are avialable.
- Maven: A dependency resolver fetching all the required dependencies given the name and version of the required libraries.
- Jedis (Redis): A java implementation of the Redis DB server. Redis is an in-memory key-value pair DB.

- Apache Tomcat: An HTTP servlet implement in and for java execution environments.

5.5.2 System Requirements

The code and tests have been successfully run on the following configuration. Any system with a configuration equal or higher than this should be able to do the job faster.

- RAM: 16GB; fails to run on 8GB, due to the high amount of in memory data.

- CPU: Intel Core i7, 4th Generation; have used all 8 virtual cores with hyper-threading, with lower CPU, processes would take longer to complete.

- HDD: 280GB dedicated; Million Song Dataset is the only component using considerable persistent memory. Storing user history and codes consume very minimal data storage (in MBs).

- Internet Connection *(optional)*: A suitably fast internet connection to getch user history and a few song information. Slower internet speeds might slow down the entire workflow. This is required for pre-processing when all the required data is collected. Alternatively, the whole pre-processed data can be imported by any means.

5.5.3 Optimizations

- Parallelization: Since, gathering data for every user is fairly independent of each other, these tasks have been implemented in parallel. Also, during the recommendation, once the set of similar users have been collected, suggesting songs per user can again be done in parallel. 16 threads has been found to be the optimal for speed on a CPU with 8 virtual cores (2 hyperthreads per core).

- On-Demand: Most songs from the dataset would not be required for recommending, and might consume some valuable runtime. The songs are loaded in the memory at the first missed access. This not only ensures faster runtime but also a lower comsumption of physical memory.

- Load Minimal Data Efficiently: Only song data like *loundess*, *tempo* and *popularity* that is required for recommending is loaded into memory. This data is stored in a serialized JSON format. This lightens the overhead of high-level data structures.

- Prevent File Access: Each song's information in the million song dataset is stored away in a separate file for each. This heightens the overhead of several files being opened and closed at runtime. Also, there is a limit imposed by the OS to the number of files that can remain open at any given point of time. Thus, only the required information has been extracted, compressed into a serialized format and loaded in memory on an on-demand basis.

5.5.4 Complications

- Last.FM API: The API is not very robust when multiple calls are made in a short span of time. Several of the calls tend to get failed resulting in the obtained history being corrupted. This restricts the use of multi-threading to fetch user data, which would have improved the runtime significantly.

5.5.5 Improvements

- Distributed Systems: It gives advantages in terms of more number of CPUs and thus more threads at work, enabling to reduce the runtime considerably. Also, the Last.FM API finds itself in a bit of fix when several calls are being made at the same time from the same computer; this also then can be relieved. This would also enable load balancing the recommendation requests.

- NFS: Using an NFS over a local network could shared the dataset and user histories across all the involved machines.

5.6 Work Flow Summary

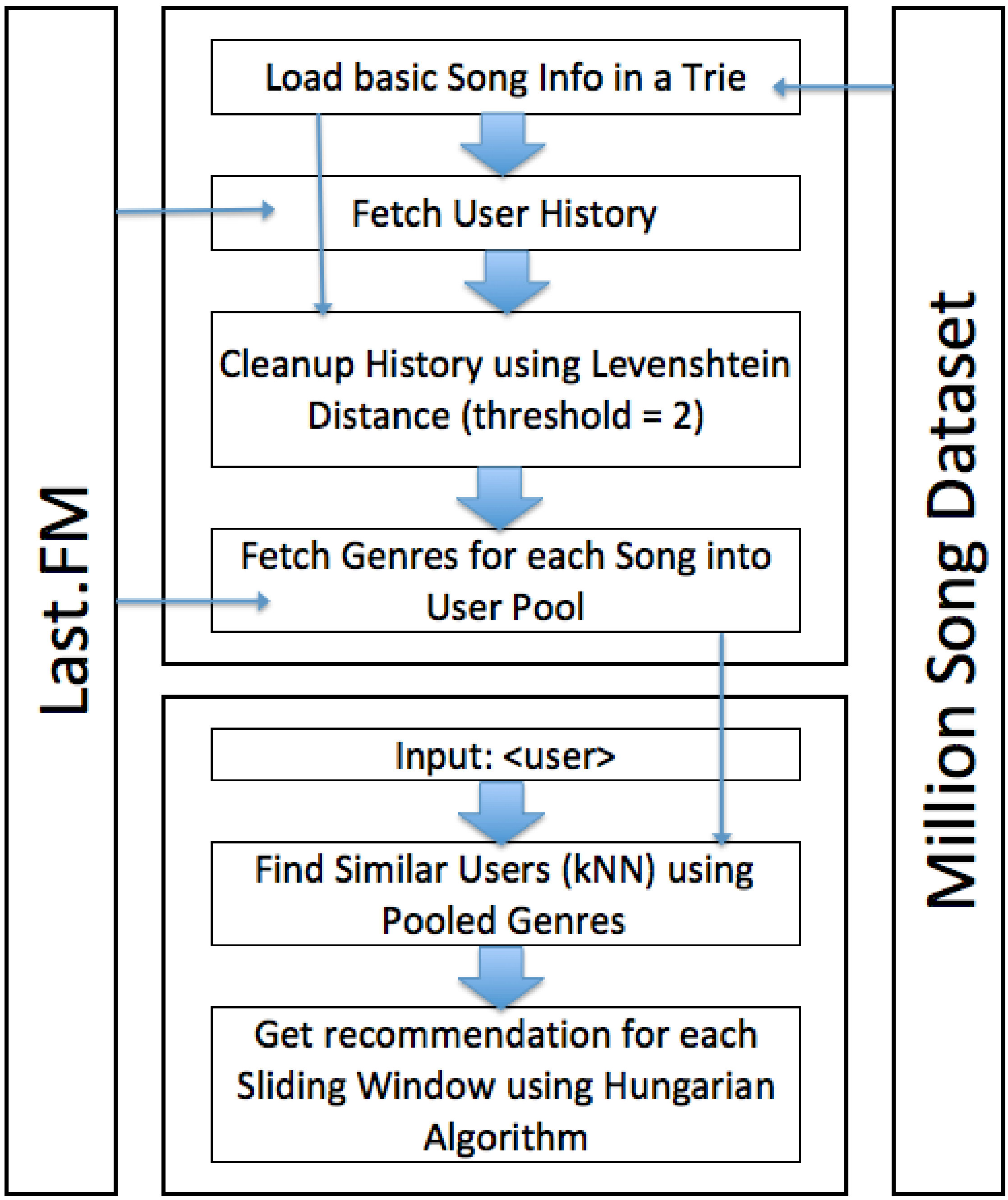

Figure 5.1: Work Flow

Chapter 6

Results

6.1 Tests

Tables 6.1, 6.2, 6.3, 6.4, 6.5 illustrates the tests have been performed on 5 Last.FM users' histories ("3en", "RJ", "eartle", "franhale", "massdosage") by varying the parameters such as the number of similar users selected for collaborative filtering, the number of songs to define a mood, and the set of weightages (artist, loudness, tempo) given to each property of the song to calculate its similarity with others.

The confidence is the measure of similarity of a "mood window", found in the history of another user having a very similar musical taste, which is most "similar" to that of the "mood window" considered for the current user.

The most recent t songs are used for testing purposes and the recommendations are based on the songs following the first t. The rank of a song in the recommendation that occurs in the set of test songs defined above is also noted down.

6.2 Inference

The higher the "confidence" value, the more similar is its corresponding "mood" to that of the current user's present mood and thus the chances of the current user liking the song heard by the other user right after the above window get higher.

The rank illustrates the success of the recommendation by cross-checking with

Similar Users	Mood Length	Weights	Confidence	Rank
50	5	$^1/_3, ^1/_3, ^1/_3$	48.45 %	3718
75	5	$^1/_3, ^1/_3, ^1/_3$	48.45 %	3879
100	5	$^1/_3, ^1/_3, ^1/_3$	48.45 %	84
150	5	$^1/_3, ^1/_3, ^1/_3$	51.01 %	135
200	5	$^1/_3, ^1/_3, ^1/_3$	52.63 %	211
50	10	$^1/_3, ^1/_3, ^1/_3$	45.06 %	3418
75	10	$^1/_3, ^1/_3, ^1/_3$	45.50 %	4751
100	10	$^1/_3, ^1/_3, ^1/_3$	46.93 %	1722
50	5	$^1/_5, ^2/_5, ^2/_5$	43.28 %	4033
50	5	$^3/_5, ^1/_5, ^1/_5$	60.53 %	3632
75	5	$^3/_5, ^1/_5, ^1/_5$	60.03 %	4367
100	5	$^3/_5, ^1/_5, ^1/_5$	62.10 %	78
150	5	$^3/_5, ^1/_5, ^1/_5$	64.73 %	120

Table 6.1: Test Results for Last.FM user: *3en*

Similar Users	Mood Length	Weights	Confidence	Rank
50	5	$^1/_3, ^1/_3, ^1/_3$	50.43 %	N/A
75	5	$^1/_3, ^1/_3, ^1/_3$	50.43 %	N/A
100	5	$^1/_3, ^1/_3, ^1/_3$	50.43 %	7608
150	5	$^1/_3, ^1/_3, ^1/_3$	50.43 %	8828
200	5	$^1/_3, ^1/_3, ^1/_3$	52.40 %	10018
50	10	$^1/_3, ^1/_3, ^1/_3$	48.91 %	N/A
75	10	$^1/_3, ^1/_3, ^1/_3$	48.91 %	N/A
100	10	$^1/_3, ^1/_3, ^1/_3$	48.91 %	N/A
50	5	$^1/_5, ^2/_5, ^2/_5$	43.28 %	N/A
50	5	$^3/_5, ^1/_5, ^1/_5$	66.15 %	N/A
75	5	$^3/_5, ^1/_5, ^1/_5$	66.15 %	N/A
100	5	$^3/_5, ^1/_5, ^1/_5$	66.15 %	7095
150	5	$^3/_5, ^1/_5, ^1/_5$	66.15 %	7767

Table 6.2: Test Results for Last.FM user: *RJ*

the songs the current user has actually heard next, i.e, the set of test songs chosen above. This however, determines the success of the tool in the current domain of the $999,999$ songs.

Based on the test results, it can inferred that a *mood length* of 5 based on the histories of 75 other users with similar musical taste gives a set of good results. The song's property weightage of $^3/_5, ^1/_5, ^1/_5$ is also seen to contribute well to the recommendations. Hence, these values are taken to be the default values of the tool. These can however be customised to give more weightage to a parameter of choice

Similar Users	Mood Length	Weights	Confidence	Rank
50	5	$^1/_3, ^1/_3, ^1/_3$	50.43 %	400
75	5	$^1/_3, ^1/_3, ^1/_3$	50.43 %	607
100	5	$^1/_3, ^1/_3, ^1/_3$	48.37 %	736
150	5	$^1/_3, ^1/_3, ^1/_3$	51.94 %	1095
200	5	$^1/_3, ^1/_3, ^1/_3$	51.94 %	1428
50	10	$^1/_3, ^1/_3, ^1/_3$	48.91 %	2632
75	10	$^1/_3, ^1/_3, ^1/_3$	48.91 %	3736
100	10	$^1/_3, ^1/_3, ^1/_3$	46.91 %	4304
50	5	$^1/_5, ^2/_5, ^2/_5$	43.28 %	563
50	5	$^3/_5, ^1/_5, ^1/_5$	66.15 %	363
75	5	$^3/_5, ^1/_5, ^1/_5$	66.15 %	555
100	5	$^3/_5, ^1/_5, ^1/_5$	63.88 %	650
150	5	$^3/_5, ^1/_5, ^1/_5$	64.59 %	970

Table 6.3: Test Results for Last.FM user: *eartle*

Similar Users	Mood Length	Weights	Confidence	Rank
50	5	$^1/_3, ^1/_3, ^1/_3$	49.71 %	4744
75	5	$^1/_3, ^1/_3, ^1/_3$	49.83 %	629
100	5	$^1/_3, ^1/_3, ^1/_3$	49.71 %	674
150	5	$^1/_3, ^1/_3, ^1/_3$	51.10 %	4351
200	5	$^1/_3, ^1/_3, ^1/_3$	51.48 %	4363
50	10	$^1/_3, ^1/_3, ^1/_3$	47.49 %	3160
75	10	$^1/_3, ^1/_3, ^1/_3$	47.49 %	3135
100	10	$^1/_3, ^1/_3, ^1/_3$	47.54 %	3225
50	5	$^1/_5, ^2/_5, ^2/_5$	43.28 %	4422
50	5	$^3/_5, ^1/_5, ^1/_5$	64.43 %	4589
75	5	$^3/_5, ^1/_5, ^1/_5$	66.18 %	470
100	5	$^3/_5, ^1/_5, ^1/_5$	64.43 %	471
150	5	$^3/_5, ^1/_5, ^1/_5$	64.43 %	5227

Table 6.4: Test Results for Last.FM user: *franhale*

and thus get different sets of results.

6.3 Improvements

This recommendation system has been built more as a *proof of concept* than a competitive tool. A lot of things may enhance the performance the results of the recommendation and to make it even more accurate. Some of them have been discussed below.

Similar Users	Mood Length	Weights	Confidence	Rank
50	5	$^1/_3, ^1/_3, ^1/_3$	48.89 %	N/A
75	5	$^1/_3, ^1/_3, ^1/_3$	50.09 %	9857
100	5	$^1/_3, ^1/_3, ^1/_3$	50.09 %	11647
150	5	$^1/_3, ^1/_3, ^1/_3$	51.10 %	6587
200	5	$^1/_3, ^1/_3, ^1/_3$	51.48 %	8008
50	10	$^1/_3, ^1/_3, ^1/_3$	48.08 %	N/A
75	10	$^1/_3, ^1/_3, ^1/_3$	48.08 %	2584
100	10	$^1/_3, ^1/_3, ^1/_3$	48.08 %	2887
50	5	$^1/_5, ^2/_5, ^2/_5$	43.28 %	N/A
50	5	$^3/_5, ^1/_5, ^1/_5$	63.29 %	N/A
75	5	$^3/_5, ^1/_5, ^1/_5$	65.97 %	7819
100	5	$^3/_5, ^1/_5, ^1/_5$	65.97 %	9345
150	5	$^3/_5, ^1/_5, ^1/_5$	68.19 %	5749

Table 6.5: Test Results for Last.FM user: *massdosage*

- Million Song Dataset: The global dictionary of songs has been restricted to 1,000,000 and recommendations are made from this dataset only. Adding more songs, will improve the quality of recommendations.

- User History: There is a certain amount of loss while fetching the user history and cleaning it up. It corrupts the determined "mood" of the user. Using a cleaner method and a reliable source of history should be able to tackle this matter.

- Users: Currently, the set of users being used for collaborative filtering is not only limited but also very random and does not encompass the different moods and interests of music listeners.

- MFCC: There are several other parameters as obtained from the MFCC analysis of a song. These included appropriately in the computation would significantly bring out better results.

- Feedback: If a user skips a certain song, the recommendation engine should learn not to present her with the song anytime soon again. The weights for each parameters can be dynamically changed as per the user feedback using machine learning techniques.

6.4 Summary

- Mood: This plays a very significant role in the process of recommendation and has not been very widely tapped. The type of songs a user has been listening to tells a great deal about her current mood and proves to be prominent factor in determining the songs she might want to hear next.

- Popularity: This parameter has been gaining importance as new artists come up with various types of songs, which are hard to categorize and thus to be recommended. Including popularity gives a way to recommend any trending song even if it does not match the user's preferences.

Bibliography

[1] Natalie Aizenberg, Yehuda Koren, and Oren Somekh. Build your own music recommender by modeling internet radio streams. In *Proceedings of the 21st international conference on World Wide Web*, pages 1–10. ACM, 2012.

[2] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[3] Michael Castelluccio. The music genome project. *Strategic Finance*, 88(6):57–58, 2006.

[4] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(4):357–366, 1980.

[5] Ulrich Germann, Eric Joanis, and Samuel Larkin. Tightly packed tries: How to fit large models into memory, and make them load fast, too. In *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 31–39. Association for Computational Linguistics, 2009.

[6] Joyce John. Pandora and the music genome project. *Scientific Computing*, 23(10):40–41, 2006.

[7] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[8] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[9] Terence Magno and Carl Sable. A comparison of signal based music recommendation to genre labels, collaborative filtering, musicological analysis, human recommendation and random baseline. In *ISMIR*, pages 161–166, 2008.

[10] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial & Applied Mathematics*, 5(1):32–38, 1957.

[11] LCW Pols. Spectral analysis and identification of dutch vowels in monosyllabic words [dissertation]. *Free University, Amsterdam, The Netherlands*, 1966.

[12] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 158–167. ACM, 2000.

[13] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[14] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[15] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. Comparison of different implementations of mfcc. *Journal of Computer Science and Technology*, 16(6):582–589, 2001.