

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



**FEUP**

**Towards the discovery of temporal  
patterns in music listening using  
Last.fm profiles**

**Mário João Teixeira Carneiro**

Master in Informatics and Computing Engineering

Supervisor: Fabien Gouyon (PhD)

Supervisor: Luís Sarmiento (PhD)

Supervisor: Eugénio Oliveira (PhD)

June 2011



# **Towards the discovery of temporal patterns in music listening using Last.fm profiles**

**Mário João Teixeira Carneiro**

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Luís Filipe Pinto de Almeida Teixeira (PhD)

External Examiner: Rui Pedro Pinto Carvalho Paiva (PhD)

Supervisor: Luís Sarmiento (PhD)

---

11<sup>st</sup> July, 2011



# Abstract

On-line social communities are becoming a part of everyone's daily life. Last.fm is a music-themed social network which has the particularity of recording the listening preferences of their users, and then using these same preferences to suggest new artists or events that match each user's taste. We developed a crawling application we then used to collect the profiles of some these users, in order to investigate some of their behaviors in terms of listening preferences, exploring the time component present in their profiles.

Research shows that there are users who exhibit seasonal and circadian preference for a particular genre, for example, by preferring to listen to dance music in the summer or folk music in the hours of the evening. Current research uses statistics-based methods to analyze these preferences. We are exploring this particular problem using a data mining-based methodology. We also explore the idea that users may also present a preference for often mixing certain genres together in the same music listening session.

Our approach is based on data mining, particularly, applying the association rules problem to these particular cases.



# Resumo

As comunidades sociais online estão cada vez mais a tornar-se parte da vida diária das pessoas. O Last.fm é uma rede social baseada em música, que tem a particularidade de guardar as preferências musicais dos seus utilizadores, e posteriormente utiliza-los para lhes sugerir novos artistas e eventos que vão de encontro aos seus gostos. Desenvolvemos uma aplicação de recolha de dados que posteriormente utilizámos para recolher os perfis de alguns destes utilizadores, com o objectivo de investigar alguns dos seus comportamentos em termos de preferências musicais, explorando a componente temporal presente nos seus perfis.

A investigação feita nesta área mostra que existem utilizadores que exibem preferências sazonais e circadianas por certos géneros. Esta investigação utiliza metodologias baseadas em estatística para analisar estas preferências. Este projecto investiga também estas componentes, mas com uma metodologia diferente, baseada em data mining. Exploramos também a ideia de que os utilizadores podem exibir uma preferência para misturar certos géneros quando ouvem música.

A nossa metodologia é baseada em data mining, nomeadamente, na aplicação do problema de geração de regras de associação a estes casos em particular.





# Acknowledgements

I would like to thank my supervisors, Prof. Fabien Gouyon and Prof. Luís Sarmiento for their valuable input and ideas during the time since I first started working in this project.

I would also like to thank all my friends who gave me ideas and encouragement, or simply listened to me, often blabbing on about my work. A special thank you goes to Rui Teixeira for all of his support.

Cheers! :)

Mário Carneiro



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	The Last.fm Social Platform . . . . .	2
1.1.1	Scrobbling and timestamps . . . . .	4
1.1.2	Social Tagging in Last.fm . . . . .	4
1.2	Objectives and Motivation . . . . .	6
1.3	Structure of this document . . . . .	7
<b>2</b>	<b>Related Work</b>	<b>9</b>
2.1	Towards time-aware contextual music recommendation: an exploration of temporal patterns of music listening using Circular Statistics . . . . .	9
2.2	Towards Time-Dependant Recommendation based on Implicit Feedback . . . . .	11
2.3	Statistical models of music-listening sessions in social media . . . . .	12
2.4	Temporal dynamics in music listening behavior: a case study of online music service . . . . .	13
2.5	Towards gathering and mining Last.fm user-generated data . . . . .	14
<b>3</b>	<b>Problem Definition</b>	<b>17</b>
3.1	The Last.fm data . . . . .	17
3.1.1	User profiles . . . . .	17
3.1.2	Artist profiles . . . . .	19
3.1.3	Mapping times to genres . . . . .	20
3.2	Deriving genre information from the Last.fm social tags . . . . .	20
3.3	Information hidden in the user's profiles . . . . .	22
3.3.1	When do people listen to music? . . . . .	23
3.3.2	The relation between time and the music people listen to . . . . .	24
3.3.3	Analyzing music listening sessions . . . . .	24
<b>4</b>	<b>Research Dataset</b>	<b>25</b>
4.1	The Crawling Application . . . . .	25
4.1.1	The Last.fm API . . . . .	27
4.1.2	Technologies . . . . .	29
4.1.3	Summary of features . . . . .	30
4.1.4	Compliance with the Last.fm API Terms of Use . . . . .	31
4.2	Methodology . . . . .	32
4.2.1	Database preparation . . . . .	33
4.2.2	Username discovery . . . . .	34
4.2.3	User information retrieval . . . . .	35

## CONTENTS

4.2.4	Filtering . . . . .	36
4.2.5	Random sample selection . . . . .	37
4.2.6	Listening history crawling . . . . .	37
4.2.7	Filtering, yet again . . . . .	37
4.3	Characteristics of the final dataset . . . . .	38
4.3.1	Demographic characteristics . . . . .	38
4.3.2	Characteristics of the users' profiles . . . . .	40
4.3.3	Characteristics related to the artists and genres . . . . .	42
<b>5</b>	<b>Data Analysis</b>	<b>47</b>
5.1	Background . . . . .	47
5.1.1	Knowledge Discovery in Databases . . . . .	47
5.1.2	Mining association rules . . . . .	49
5.1.3	Data mining platforms . . . . .	51
5.1.4	Generation of the test files . . . . .	51
5.2	Analysis of music listening sessions . . . . .	53
5.2.1	Analysis of individual users . . . . .	53
5.2.2	Analysis of the whole dataset . . . . .	54
5.3	Analysis of played items with respect to time . . . . .	55
5.3.1	Analysis of individual users . . . . .	55
5.3.2	Analysis of the whole dataset . . . . .	60
<b>6</b>	<b>Conclusions</b>	<b>63</b>
6.1	Future Work . . . . .	64
	<b>References</b>	<b>65</b>
<b>A</b>	<b>Description of the developed scripts</b>	<b>67</b>
A.1	Crawler Application . . . . .	67
A.2	Data Transformation scripts . . . . .	68
A.2.1	TagPool script . . . . .	68
A.2.2	Timeline script . . . . .	68
A.2.3	CircularTimeline script . . . . .	69
A.3	Chart generation scripts . . . . .	70

# List of Figures

1.1	Web presentation of a Last.fm User Profile . . . . .	3
1.2	Tags for Fiona Apple, a singer-songwriter in tag cloud form. A tag cloud is a representation of user-generated content. . . . .	5
1.3	Tag cloud for Pink Floyd. This tag cloud is representative mostly of the genres their career encompassed. . . . .	5
2.1	Two different Last.fm profiles analyzed using the circular statistics methodology described by Resa, concerning the hours of the day. The profile on the left shows a tendency to listen to music during the hours of the night while the profile on the right shows a preference for the hours of the afternoon. The red slice represents the user's centroid. . . . .	10
2.2	Two examples of a partition for a 24hour cyclic period. Illustration from [BAA09]. . . . .	12
3.1	Last.fm presents to the user his listening history in this form. . . . .	18
3.2	The tag cloud for Rebecca Black, teen pop phenomenon. . . . .	21
3.3	Timeline for two particular last.fm users. The horizontal axis represents days while the the vertical axis represents the time (hours). Each blue square marks a day hour where music has been listened to. The intensity of the blue represents the number of songs listened to. The images represent a profile sample taken from July to December 2010. . . . .	23
4.1	Overall view of the crawling application modules and interaction with the other systems involved in the process: the Last.fm API and the MySQL database server . . . . .	26
4.2	Flowchart of the dataset construction process . . . . .	33
4.3	Database diagram . . . . .	34
4.4	List of fields retrieved by the Last.fm Web API method <code>User.getInfo</code> . . . . .	35
4.5	Country distribution in the final dataset (for the top 40 countries) . . . . .	38
4.6	Gender distribution in the final dataset . . . . .	39
4.7	Age distribution in the final dataset . . . . .	39
4.8	Scatter plot of total played items and average played items per day . . . . .	40
4.9	Total played items distribution per user (nominal scale on the x axis) . . . . .	41
4.10	Total played artists distribution per user (nominal scale on the x axis) . . . . .	41
4.11	Total played tracks distribution per genre (nominal scale on the x axis) . . . . .	42
4.12	Total played tracks distribution per artist (nominal scale on the x axis) . . . . .	43
4.13	Top 30 artists represented in the dataset . . . . .	44
4.14	Top 30 genres represented in the dataset . . . . .	44

## LIST OF FIGURES

5.1	The iterative process of KDD as presented in [FPsS96]. . . . .	48
5.2	The architecture of a typical data mining system as presented by [HK00].	49
5.3	Overview of the system that allows the generation of the data files . . . . .	52
5.4	Total rules found distribution per number of users . . . . .	53
5.5	Tags Last.fm shows as being related to Rock artists . . . . .	54
5.6	Total rules found distribution per number of users . . . . .	56
5.7	20 most popular rules for this time partition . . . . .	57
5.8	Total rules found distribution per number of users . . . . .	58
5.9	20 most popular rules for this time partition . . . . .	59
5.10	Total rules found distribution per number of users . . . . .	60

# List of Tables

2.1	Aggregated vision of the objectives, methodology and data sources used by the related works . . . . .	16
3.1	A segment of a user listening profile . . . . .	19
3.2	Several partial artist tag profiles . . . . .	20
3.3	Composition of a user listening profile with the artist tag profiles of the artists in his listening history . . . . .	21
3.4	Some tags for the artist Rebecca Black and respective weights. . . . .	22
4.1	Summary of the <code>User.getFriends</code> API method . . . . .	28
4.2	Summary of the <code>User.getInfo</code> API method . . . . .	28
4.3	Summary of the <code>User.getInfo</code> API method . . . . .	29
A.1	Bash scripts included in the package . . . . .	67
A.2	Configuration files included in the package . . . . .	67
A.3	Runnable scripts included in the package . . . . .	67
A.4	Library scripts included in the package . . . . .	68
A.5	Summary of the <code>TagPool</code> script . . . . .	68
A.6	Summary of the methods available in the <i>timeline</i> script . . . . .	69
A.7	Summary of the class <code>CircularTimeline</code> . . . . .	70
A.8	Plotting scripts included in the package . . . . .	71

## LIST OF TABLES



# Chapter 1

## Introduction

A lot has changed in the past decade in the way people consume and explore music. We rely less on record stores and other traditional means to try new music, and we trust more on the Internet to help us achieve this. We no longer rely on shelves to keep and organize our physical personal collections, as we prefer to keep them as digital files within our computers and portable media devices. Moreover, we do not even need to own any kind of collection of music as we find a growing number of online services that stream audio to our computers without the need of us having to keep the music we are listening to.

Examples of Web-based services that offer music in digital form to consumers are online stores like the iTunes Store <sup>1</sup> and Internet radio services like *Pandora* <sup>2</sup>, *Spotify* <sup>3</sup> and *Last.fm* <sup>4</sup>. The latter is a case of particular interest as it offer services like those mentioned previously and integrates them seamlessly with several social networking features. Integral part of Last.fm's service is its recommendations system, as it is able to offer its users recommendations of artists, musical events and even other users that they may find interesting. However, the feature that differentiates Last.fm from its peers is a system that originated from a project called *audioscrobbler* — Last.fm builds a detailed profile of each user's musical taste and listening habits by recording details of the songs the user listens to. It's not hard to picture the amount of relevant data present in Last.fm profiles if we consider the 40 million active users Last.fm claims to have [Jon09], some with accounts that *scrobble* since 2005.

Music Information Retrieval, the main area this project is inserted in, is a cross-discipline that encompasses many different fields of study. These fields of study have one common objective — retrieving information within music and music-related content.

---

<sup>1</sup>Software-based online digital media store owned by Apple — <http://www.itunes.com>

<sup>2</sup>Automated music recommendation service and Internet radio. — <http://www.pandora.com/>

<sup>3</sup>Proprietary peer-to-peer music streaming service. — <http://www.spotify.com>

<sup>4</sup>The Last.fm service is of particular interest to this document and its features are described in detail further in this document. — <http://www.last.fm>

The increase in the availability of musical contents in digital form motivated the growth in this area. Topics that concern Music Information Retrieval range from feature extraction in songs (of rhythmic and melodic components) to the analysis of data concerning music with music, i.e., music metadata — like the data existing in copious amounts in Last.fm.

In fact, there is a growing number of works and publications concerned with the analysis and presentation of Last.fm data. There exists work developed here, at INESC Porto<sup>5</sup> with the objective of gathering artist information and data from Last.fm, with the objective of applying data mining techniques to extract relevant information, like relationships between artists and genres[dCL09]. With a similar theme, this work is directed towards the gathering of Last.fm user profile information, the application of data mining techniques to the data gathered and the extraction of conclusions similar to those taken by the previous work, but this time considering in as much detail as possible the element that was left out in [dCL09]: The users of Last.fm and their listening habits.

### 1.1 The Last.fm Social Platform

Last.fm is a web-based social music platform with an array of features that make it difficult to categorize. Last.fm combines traditional social network features like communication with and discovery of other users with radio stations, several recommendation services and other idiosyncratic features. It allows its users to listen to music, read information about artists (created by the users themselves in a collaborative *wiki*-like fashion) and discover artists that they do not know. In the Web 2.0 way, users are allowed to set up their very own profile. This facilitates automatic recommendation of artists, videos, other users and musical events in the user's local area.

User's profiles are constantly updated via a specific software. This software includes plugins for several popular media players and uploads statistics about the music a particular user listens to. As such, these records are analyzed by Last.fm and serve as the base for their music recommendation service.

---

<sup>5</sup>Private non-profit association for scientific research based on Porto, Portugal

## Introduction

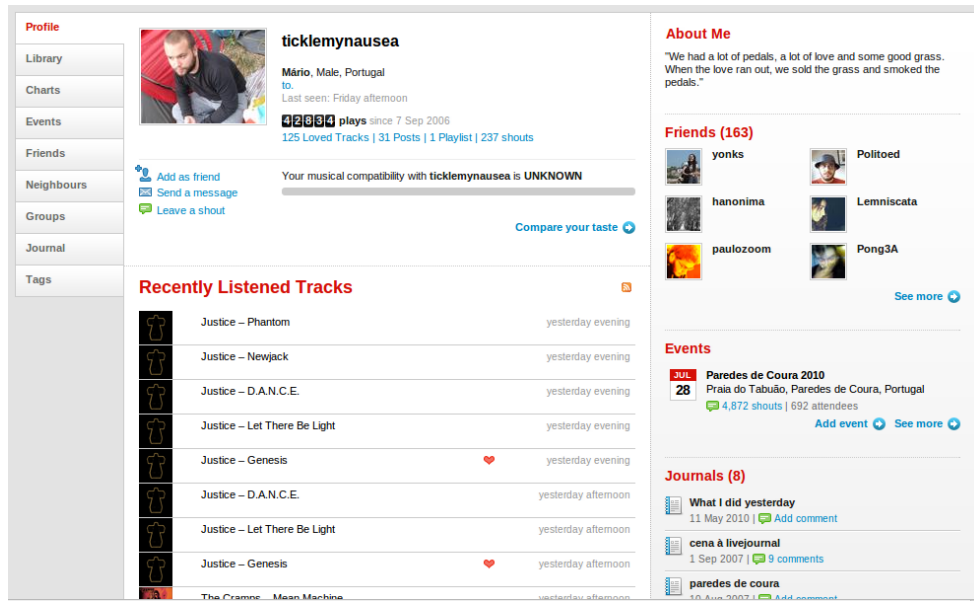


Figure 1.1: Web presentation of a Last.fm User Profile

The origin of Last.fm can be traced to a 2003 computer science project by Richard Jones, a University of Southampton student, called *Audioscrobbler*. The core idea of this project was the progressive construction of a user's personal profile detailing his taste in music. Through a specific software that monitors what a user listened to on his computer, the website would present the user with information regarding his musical listening preferences. Moreover, it would use this information as the basis for the delivery of personalized recommendations of other artists similar to what a user listens to and other users whose tastes are also similar to his.

In 2005 Audioscrobbler merged with Last.fm, an internet radio and music community website. The resulting website kept Last.fm's name, while Audioscrobbler's name and domain were used to refer to the backend that supports the construction of users' profiles and other features like the recommendation service.

Last.fm allows communication between users through two means: the more traditional private messages and shoutboxes. A shoutbox is a section of a page in Last.fm where a user can post a message (hopefully) regarding the content of the webpage. Shoutboxes are omnipresent throughout Last.fm, as they not only exist in user pages, but also on pages relating to pretty much everything in last.fm like tags, artists, albums and songs.

Last.fm also provides a Web API that allows anyone to build their own programs using Last.fm data. The data available ranges from user-generated data like tags, shouts and artist descriptions to a particular user's complete listening history<sup>6</sup>.

<sup>6</sup>Provided the user has not made his profile private via his user settings.

### 1.1.1 Scrobbling and timestamps

*Scrobbling* is the name given to the process of submitting information to Last.fm about each song the user listens to. This information is stored in the last.fm servers and used to build the users' profiles based on what they listen to. The submission process is achieved via specific software that can be installed on the user's media player of choice as a plugin. Scrobbling is not only possible in desktop applications but also on several musical web applications like The Hype Machine<sup>7</sup> and Grooveshark<sup>8</sup>.

Each submission sends to Last.fm details about the played track, like song name and duration, artist and album information. A timestamp is also sent. These informations are then stored in the Last.fm databases. However, the Scrobbling API details that the timestamp sent must be in the Unix time format<sup>9</sup>. Sending the time information in this fashion means purposely discarding information about the scrobbling user's time zone and daylight saving. Instead, Last.fm relies on an option in the website preferences, where each user can specify the time zone he's located at. Last.fm then uses this setting to render the correct time information from the viewer's perspective.

There is no way to obtain the value of this setting for each user, as this is considered private user information by Last.fm. This means that while the listening history of a user is easily obtainable via the Web API, we can only know when a user listened to a given song in a time frame relative to the UTC time standard. Therefore, we will have no notion of when the given user listened to a song in his timezone, and will not be able to tell if he did so during the morning or afternoon.

### 1.1.2 Social Tagging in Last.fm

A *tag* is a textual item associated with a piece of information, which it describes[Lam08]. Last.fm has a tag system that enables users to label artists, albums and songs using keywords that define them. They are a source of valuable information, as they may convey a lot of relevant information about an artist, such as genre, mood and instrumentation1.2[Cel08]. However, because of their free and unstructured nature, there is a lot of irrelevant information and noise in the tags.

---

<sup>7</sup>Mp3 blog aggregator, which allows the playing of songs via streaming on a web interface

<sup>8</sup>Online music search and streaming service

<sup>9</sup>Unix time is a system for describing points in time, defined as the number of seconds elapsed since midnight Coordinated Universal Time (UTC) of January 1, 1970

## Introduction

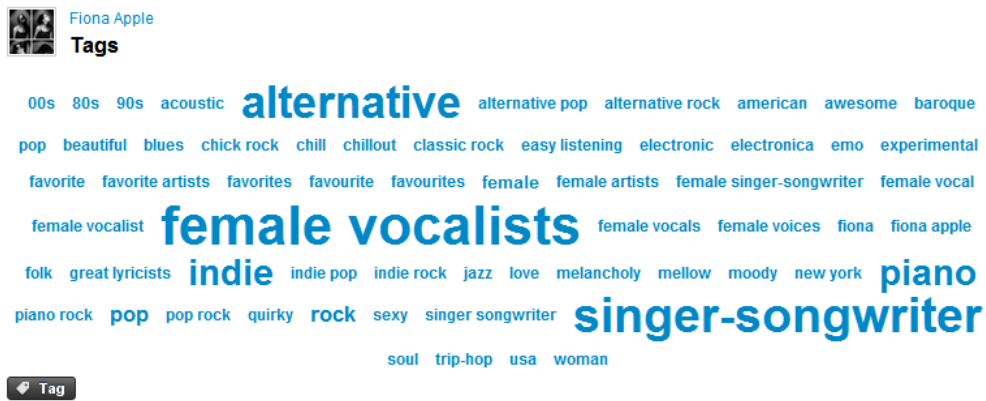


Figure 1.2: Tags for Fiona Apple, a singer-songwriter in tag cloud form. A tag cloud is a representation of user-generated content.

Social tags are the result of the collaboration between users, as an individual may apply a short text annotation to an item in a collection to organize his personal content. In the context of Last.fm, users are free to apply any number of tags to any artist, album or song. These tags are then combined with those applied by other individuals to form a collective body[Lam08]. With a large enough group of tags and taggers, a very rich view of the tags emerges: a tag cloud. In this form of information visualization, tags are usually organized alphabetically, with the relevance of each word represented by the font weight. This allows for two ways of finding information[HK07]. The system of classification derived from this practice is known as a *folksonomy*, a portmanteu of *folk* and *taxonomy*.

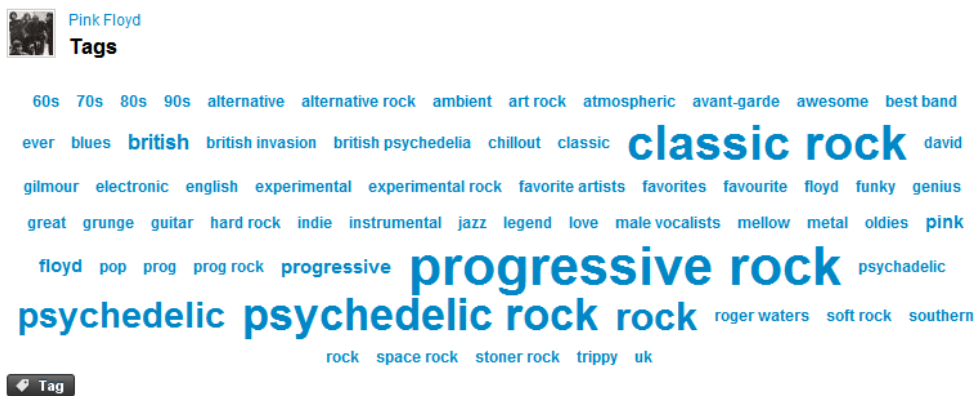


Figure 1.3: Tag cloud for Pink Floyd. This tag cloud is representative mostly of the genres their career encompassed.

Tags are unstructured and free-form in nature, meaning that there is usually no restriction on what a user may tag an item as. In the particular context of tagging music items in Last.fm, one can, for example, tag a band one loves as "*best band ever*" and another one loathes as "*utter crap*". These tags may convey information about many different facets

of the music, like genre, instrumentation, rhythm and vocalization. Tags may also include information about a user's relation to a particular band. A trending example is the "*seen live*" tag [Cel08].

Deriving genre information from the tags applied to artists is of great importance to this project. The work preceding this thesis by [dCL09] involved the crawling of this particular kind of user-generated data present in Last.fm. We began our work with a database that contains tag information for over 1,000,000 artists.

## 1.2 Objectives and Motivation

It is a fair assumption that many Last.fm users will listen to music almost exclusively through their computers and portable media devices. Considering this, their Last.fm profile will be essentially a detailed log of a significant part of their music listening experience for the duration of their last.fm membership. The ultimate objective of this work is to uncover some of the information latent in this data by describing the variables present in the Last.fm user listening profiles and discovering relationships among them.

For example, two variables present in these profiles are *time* — the instant in which a user listened to a particular song, and *genre* — the taxonomy given to the artist and song the user listened to. These two could be related in the sense that some users might prefer to listen to a particular genre at a particular point in time. More variables and relationships between them are described chapter chapter 3.

A possible application of the relationship given as an example above would be in recommendation systems. Present-day recommendation systems are able to analyze a user's profile in order to offer personalized recommendations of items the system predicts the user will be interested in[Ahm04]. However, the quality of a recommendation as perceived by the user depends not only on the item itself, but also of the time the item is delivered[BAA09]. A person that prefers listening to *folk rock* in the hours of the evening might appreciate even more the artist recommended if they listen to it during the aforementioned period of time. Also, as they grow, personal collections of music become increasingly hard to manage. Although modern-day media players provide library management mechanisms like searching and browsing through music metadata (such as ID3 tags), a linear search through the user's collection until a particular piece of music appeals to the user as what he wants to hear at the moment is the main method of music selection[J CJ04]. Knowing what a user listens to and when he likes to listen to it could improve current methods of automatic playlist generation.

In detail, this work has the following goals:

- To develop a crawling application capable of retrieving and storing Last.fm user profiles

- To isolate a data set composed of users whose profiles are representative of their music listening habits
- To develop a visual analysis of such data
- To develop a process capable of transforming the stored profiles into more adequate representations
- And to analyze these representations using data mining algorithms with the goal to uncover relationships and information already present there.

### **1.3 Structure of this document**

This document is divided into six chapters. This chapter, *Introduction*, gives a general context of this problem, as well as its particular objectives and motivations. It also delivers some background information about the Last.fm platform. The second chapter, *Related Work*, presents a review of several related works and scientific areas that are related to this work. The third chapter, *Problem Definition*, presents a detailed description of the problem and main goal of the research this thesis deals with. The fourth chapter, *Research Dataset* presents in detail the process that led to the final dataset we will analyze. The fifth chapter, *Data Analysis*, explains the data mining process we subjected the data we gathered to as well as the conclusions that resulted from that process. The sixth and final chapter, *Conclusions* presents a critical analysis of the developed work as well as the work that we leave to be continued.

## Introduction



## Chapter 2

# Related Work

This chapter presents a review of several works related with the analysis of the behavior of music listeners, in some way with respect to time. They all researched with similar datasets in form<sup>1</sup>. Resa et al. and Baltrunas et al. used data obtained from by Last.fm, while Zheleva et al. used data provided by Zune Social<sup>2</sup> and Park et al. used data provided by Bugs Music<sup>3</sup>.

### 2.1 Towards time-aware contextual music recommendation: an exploration of temporal patterns of music listening using Circular Statistics

The work developed by Resa et al.<sup>4</sup> had the premise that the music we listen to and love is a part of, and often, an extension of our personality. Since rhythm is an inevitable part of human life, be it the circadian rhythm or the weekly or annual repetition we find in our lives, we can come to the conclusion that these rhythms end up having an influence on our music selection.[Res10]

Resa et al. considered that for some music listeners, their music listening history could show patterns arising from a preference for certain genres and artists at certain times of the day or of the week. Resa used circular statistics to analyze the listening history of several Last.fm users, compiled in a publicly available dataset<sup>5</sup>. However, it neglected the nature of the timestamps kept in the Last.fm databases<sup>6</sup>.

---

<sup>1</sup>consisting mainly of  $\langle user, playeditem, timestamp \rangle$  tuples

<sup>2</sup>Music social network owned by Microsoft that has drawn comparisons to Last.fm

<sup>3</sup>Online music service popular in Korea

<sup>4</sup>[Res10] and [HRS]

<sup>5</sup>Accessible through <http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/index.html>

<sup>6</sup>Please see 1.1.1

## Related Work

The methodology used by Resa was based on the use of circular statistics. This sub-field of statistics is aimed at the analysis of circular data, i.e., data where angles have a meaning. It involved two approaches. A *user-artist* approach, where the data subjected to analysis was simply a list of  $\langle user, artist, timestamp \rangle$  tuples, as obtained from the dataset referred above. The other approach, a *user-genre* approach, involved the mapping of an artist to a genre, using the Last.fm *artist.getTopTags* API method. In order to extrapolate the desired artist-genre information, the obtained set of tags was then filtered by weight (which indicates the tag relevance), uniformized (to remove underscores, hyphenation and other characters that would introduce ambiguity, like between *hip-hop* and *hiphop*) and matched against a pre-compiled list of musical genres.

Their circular statistics methodology involved the mapping the points in time of the listening history of users to the value of an angle, to outline the repeating (rotationally invariant) nature of days and weeks. It then modeled the user listening behavior by characterizing the user's listening tendencies with statistical parameters such as mean direction and resultant mean vector length.

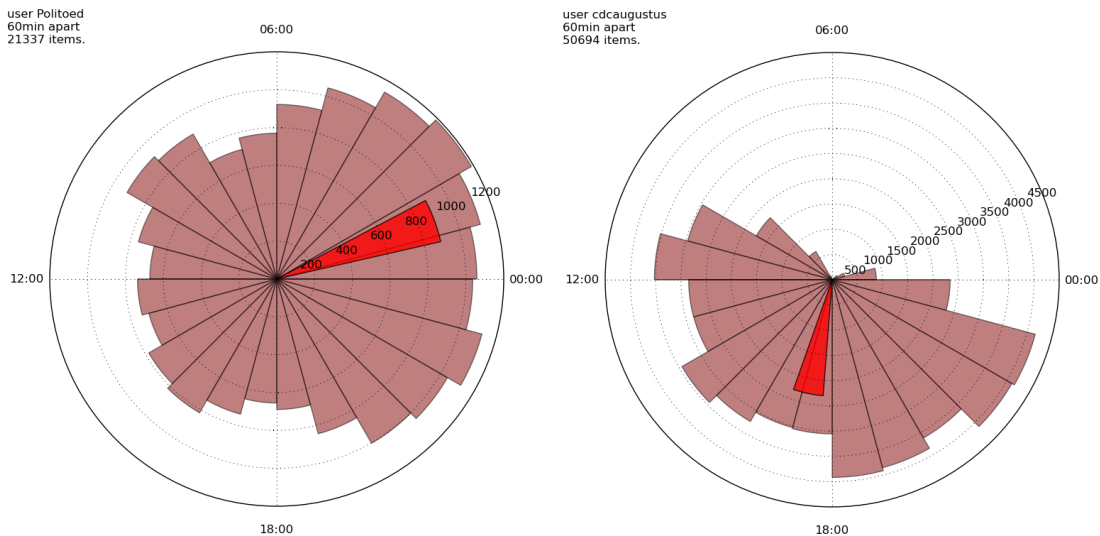


Figure 2.1: Two different Last.fm profiles analyzed using the circular statistics methodology described by Resa, concerning the hours of the day. The profile on the left shows a tendency to listen to music during the hours of the night while the profile on the right shows a preference for the hours of the afternoon. The red slice represents the user's centroid.

These are the descriptors of the listening tendencies of the users. They then showed that these trends are relevant as the distributions are different from uniform (as illustrated by 5.2)

Their *user-artist* and *user-genre* datasets were submitted to cleaning process, which aimed to remove *noise* from the user's profiles. Resa considered noise to be genres and

artists that only appeared in the user profiles sporadically and would not be a significant part of the user's listening history or source of information.

Following a content-based approach, it was then split into two datasets. the *prediction* dataset and the *validation* dataset. The *prediction* dataset contained the listening history of all users from one point in time backwards, and the *validation* dataset contained the listening history from the same point onwards. In this view, the user's past behavior is a reliable indicator of the user's future behavior, so any conclusions drawn from the first dataset could then be supported by the second dataset.

## 2.2 Towards Time-Dependant Recommendation based on Implicit Feedback

Linas Baltrunas and Xavier Amatriin<sup>7</sup> dealt with the improvement of user satisfaction (and the perceived quality of the recommendations) by proposing an approach called *micro-profiling* and with the long term goal of creating a time-aware recommendation system. This approach splits a profile into several sub-profiles, each one representing the user in a particular context. This approach assumes that the users' preferences can evolve over time, but exhibit some temporal repetition: listening to one genre while working and another before going to bed. The main idea behind Baltrunas and Amatriin's approach on recommendations was the partition of a large user profile into smaller *micro-profiles* and use of these multiple profiles on the recommendation instead of the single, larger one. This work also used data provided by Last.fm, so they were able use large amounts of time-enriched data. The data set included data collected during a two year period and the profiles of 338 random Spanish users. Also, this work did not explicitly acknowledge the nature of the timestamps kept by Last.fm<sup>8</sup>, although specifying a single country is a possible way of dealing with this problem<sup>9</sup>

This work was based on the use of *implicit feedback*, meaning that data only gave positive feedback about itself, i.e., feedback about the tracks the user listened to the most and when the user prefers to listen to the artist, leaving behind any kind of negative feedback about any artists or time preferences, in contrast with other datasets that use explicit feedback (i.e., the user himself rates the items, positively or negatively). Also, each rating is given to an artist, and not to a genre or genres the artist belongs to.

---

<sup>7</sup>[BAA09]

<sup>8</sup>Please refer to 1.1.1

<sup>9</sup>We have to take into account the several timezones that may span across a country and its dependent territories, which in the particular case of Spain are only GMT+00 and GMT+01.

## Related Work

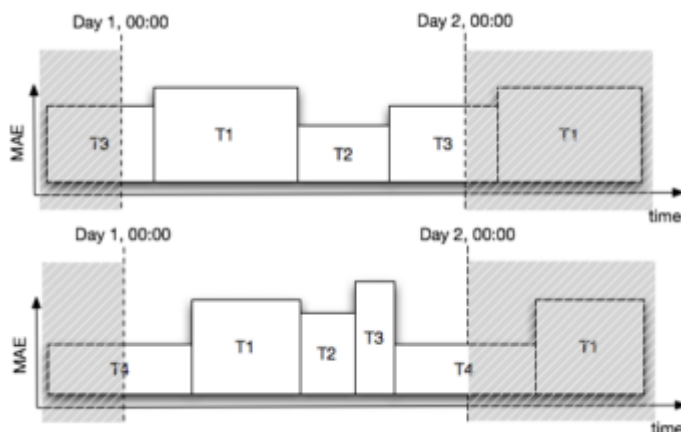


Figure 2.2: Two examples of a partition for a 24hour cyclic period. Illustration from [BAA09].

The main idea behind micro-profiling was the partition of a user profile  $u$  into several micro-profiles  $u_1, u_2, \dots, u_n$ , each one representing the user's taste in a particular time span — for example, representing the user in the morning, evening, weekend, summer, winter, etc. In this context, the challenge presented was the discovery of meaningful partitions based on time cycles, such that each partition represents a periodic instance of time (e.g., weekends) where the user has cyclic behaviour. This comes from the fact that for most users the definitions of morning, afternoon, work hours and personal time will vary, so using the same time partition for everyone would not work globally. This work also dealt with the fact that continuous variables (like time in this case) do not appear frequently in the recommendations problem.

The first main problem this work dealt with was to find which kind of partitioning would be the most meaningful for a given user. The approach involved the creation of a measure  $E$  of a partition  $T = T_1, T_2, \dots, T_n$  that is defined to be the weighted average of all the errors (of the recommendation predictions made) on the partitions  $T_i$  of the time domain. The best partition  $T$  would be the one that minimizes the value of  $E$ . The article suggested several partitions for consideration.  $T_{week}$  divided the week into working days and weekends.  $T_{day}$  divided the day into morning and evening periods. It also included a partition of even and odd hours to test the system's behavior with a meaningless time partition.

### 2.3 Statistical models of music-listening sessions in social media

Zheleva et al<sup>10</sup> presented several definitions of statistical models that describe patterns of song listening in an online music community (Zune Social). It dealt with the need to "han-

<sup>10</sup>[ZGMMF10]

dealing with "large-scale data logs" and "produce effective representations of media consumption" to enable efficient processing. Considering the large numbers of users and songs present in these types of communities, they stated that there is a need to create highly compact data representations of these logs. It also presented a heuristic for the grouping of songs in music listening sessions in the song listening logs we are dealing with, where the exact length of a played item is, quite often, unknown. In fact, we can only say for sure that a user was, in fact, listening to a particular item at a given time, while we can affirm nothing about when he started and stopped listening to the played item. According to Zeleva et al., a session  $S$  of media items  $(m_1, m_2, \dots, m_n)$  is a sequence of  $|S|$  songs that the user has listened to such that the difference between the timestamps in two consecutive played items is below a certain threshold  $\alpha$ :

$$t_{n+1} - t_n < \alpha$$

The *playlist*, as used by Zeleva et al. to refer to the full listening log of a given user, consisting of a single sequence  $U = (m_1, m_2, m_3)$  could then be divided into sessions as  $U = (S_1, S_2, \dots, S_n)$  accordingly to the definition given above, given the error introduced by the estimation of the  $\alpha$  parameter. This error brings a simple implication: if a user listens to a media item with duration above the  $\alpha$  parameter, what would in reality be a single music listening session would appear divided in two to the system.

## 2.4 Temporal dynamics in music listening behavior: a case study of online music service

Park et al.<sup>11</sup> were motivated by the lack of incorporation of the temporal context in traditional information filtering systems. It considered that the time variable stood out among all the other contextual variables (like location) in the sense that it is both continuous and periodic, therefore, the preference of the users on some information items might itself be periodic. Park et al. focused on the temporal dynamics present in the usage logs they obtained from Bugs Music. The usage logs Park et al. used in their research had the form  $\langle user, playeditem, timestamp, service\_length \rangle$  form, meaning that they had access to one more variable all the other works presented here had not: the length of the listening event. They also had immediate access to the genre of each played item (via one of several metadata tables). Since Bugs Music is an online music streaming service, Park et al. also had access to the exact lengths of each played item. The original log table contained billions of records. Since the goal of their paper was to approximate the distribution of popularity, they sampled their original log down to 1% of its volume.[PK10]

---

<sup>11</sup>[PK10]

They then cleaned down the records which have noisy data. Having the length of the listening event helped clean down the dataset because this way they could remove tracks that were skipped by the user. They removed from their dataset every listening event in which the duration of the event was less than half of the complete song length. Park et al. performed several analyses on their dataset, like the change of a certain item's popularity over time but the ones relevant to this work were the periodicity analyses. First they analyzed the number of listening events that belonged to each day of the year. It showed a periodic decrease and increase in the usage of the system during the weekends, a period of time in the Bugs Music website has less traffic. It then analyzed the number of played items versus the days of the week and the hours of the day, showing that users listen to a lot less music on weekends and on the hours of the night than on the weekdays and on hours ranging from 10AM to 10PM.[PK10]

Park et al. also had immediate access to the genres of each played item. The next step in their analysis was to determine how the time of day affects the users' listening preferences. They picked 10 genres to show the difference between them, and did so with normalized values (since some genres may be preferred over others by the population represented in their dataset). They performed a seasonal (months of the year) and a time of day analysis (hours of the day). The genres considered were *Ballad*, *Rock ballad*, *Dance Pop*, *Pop Rock*, *RNB*, *Adult contemporary*, *Indie rock*, *Hip-Hop*, *Solo Instrumental* and *Kids*.

Considering their time of day analysis, they saw that minor genres like *Kids* and *Solo Instrumental* showed a different pattern than all others, increasing during the morning and having a peak at around noon and rapidly decreasing with the afternoon, while the other genres showed a slower increase through the morning and afternoon, peaking around the evening. This analysis also showed, for instance, that people prefer to listen to rhythm and blues to dance pop at night[PK10]. Their seasonal analysis showed that people prefer to listen to ballads over dance-pop songs, and that this difference became bigger during the winter. They also showed that Christmas carol songs are highly unpopular (less than 0.05% of all listening events) during the months that precede the holiday seasons, becoming much more popular during the holidays (up to 4%).

## 2.5 Towards gathering and mining Last.fm user-generated data

Lima, João<sup>12</sup> tried to discover patterns and knowledge about music among the copious amount of user-generated data present in Last.fm. Lima developed a crawling application capable of gathering artist information present in Last.fm, such as artist profiles, tag clouds and discographies. Tag clouds are built by users and artist profiles are edited in a

---

<sup>12</sup>[dCL09]

collaborative, *wiki*-like fashion. Lima performed several data mining tests with the gathered data.

Summarily, these tests were:

- Using the clustering  $k$ -means algorithm to discover the percentage of similar artists that share the same tags;
- Using the clustering  $k$ -means algorithm to cluster artists by evolution (in terms of listeners) over the period of a month;
- Using the association Apriori algorithm to discover association rules between tags in the artists' tag clouds;
- Using the association Apriori algorithm to discover association rules between tags of an artist and the tags of similar artists;
- Using the association Apriori algorithm to discover association rules between tags, albums and popularity measures between similar artists.

While Lima did discover some interesting rules (for example, if an artist has the "pop rock" tag, its most similar artist has the "rock tag, with 90% confidence)[dCL09], this work did not include the users of Last.fm and their profiles, as they were out of the work's scope.

In the dataset construction process, Lima found many artist names that appear are in fact created in Last.fm by users that scrobble tracks that have mis-formed mp3 (or other kind of) tags in their music files. Lima solved this problem by recurring to the MusicBrainz database<sup>13</sup>.

## Summary

In this chapter we reviewed several works and scientific articles that somehow relate to our problem. We conclude by showing an aggregated vision of all the objectives, methodologies and data sources used by all the related works in table 2.1.

---

<sup>13</sup>Open content music database, accessible at <http://musicbrainz.org>

## Related Work

Table 2.1: Aggregated vision of the objectives, methodology and data sources used by the related works

Work	Objectives	Methodology	Conclusions	Data Source
Towards time-aware contextual music recommendation <a href="#">2.1</a>	Study and prediction of musical preferences for certain artists or genres that exhibit a cyclical nature over time	Application of circular statistics	Discovery that a non-negligible amount of listeners do prefer to listen to certain artists and/or genres at some points in time that have a repetitive nature.	Last.fm <sup>a</sup>
Towards time-dependent recommendation based on implicit feedback <a href="#">2.2</a>	Improving music recommendations with the knowledge that user preferences do evolve over time but still exhibit some temporal repetition	Partitioning of user profiles into several micro-profiles that represent partitions of time of a repeating nature	Increase of the rating of the prediction of future listening of artists and genres	Last.fm <sup>a</sup>
Statistical models of music-listening sessions in social media <a href="#">2.3</a>	Creation of statistical models for the characterization of user preferences in social media	Partitioning of listening history logs into sessions for the creation and of statistical models and evaluation using baseline tests and perplexity comparisons with the LDA model	Creation of a session-based hierarchical graphical model that has a lower perplexity and shorter training time that the LDA model, used for baseline comparisons	Zune Social
Temporal Dynamics in Music Listening Behavior <a href="#">2.4</a>	Studying the temporal dynamics of user behavior in music listening	Comparison of several genres and artists in a temporal context	Showed that certain genres and artists are preferred either on a specific time of day or on a seasonal basis	Bugs Music
Towards gathering and mining Last.fm user-generated data <a href="#">2.5</a>	Gathering of and application of data mining techniques to Last.fm user generated data	Creation and cleaning of a dataset for analysis with several uses of the <i>Apriori</i> and <i>k</i> -means clustering algorithms	Creation of a data mining system that was used to produce association rules of artists/genres in several different contexts	Last.fm <sup>b</sup>

<sup>a</sup>Dataset compiled by Oscar Celma accessible via <http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/>

<sup>b</sup>Obtained with the crawling application developed in this work



## Chapter 3

# Problem Definition

This section will describe the relevant part of the data Last.fm makes available to us and that this work gathered. Then, it presents some of the information that might be hidden in this data and that we are interested in discovering. It also describes the problem of the derivation of musical genres from the information present in Last.fm and the solution we used to go past it.

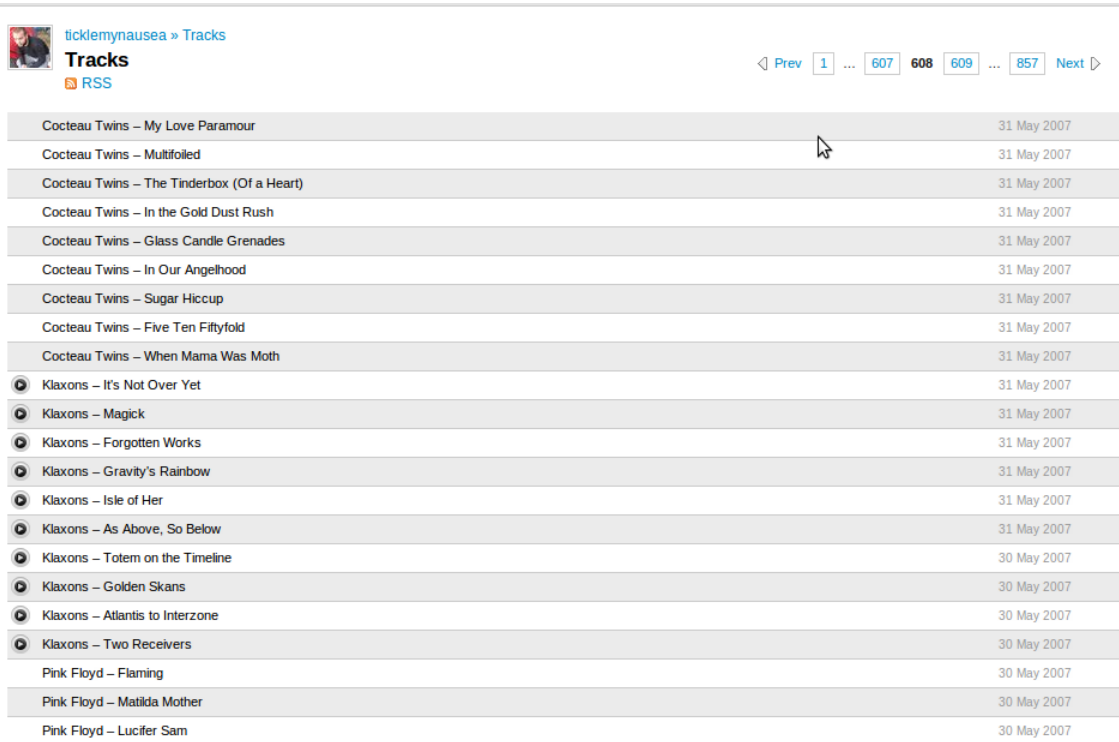
### 3.1 The Last.fm data

This section presents a formal definition of the data present in the user and artist profiles and of the methods used to manipulate them.

#### 3.1.1 User profiles

A Last.fm user listening profile contains a history of all the songs that the user has listened to and the point in time at which the listening event has occurred. It can be formally defined, for each user  $u$ , as a map  $p_u$  between points in time and the songs that  $u$  has listened to. Points in time are described using Unix time, defined as the number of seconds elapsed since 00:00 January 1, 1970, UTC. This point in time is called the *Unix epoch*.

## Problem Definition



The screenshot shows a Last.fm user profile for 'ticklemynausea' with a 'Tracks' section. The tracks are listed in a table with columns for track name and date. The tracks are: Cocteau Twins - My Love Paramour (31 May 2007), Cocteau Twins - Multifolied (31 May 2007), Cocteau Twins - The Tinderbox (Of a Heart) (31 May 2007), Cocteau Twins - In the Gold Dust Rush (31 May 2007), Cocteau Twins - Glass Candle Grenades (31 May 2007), Cocteau Twins - In Our Angelhood (31 May 2007), Cocteau Twins - Sugar Hiccup (31 May 2007), Cocteau Twins - Five Ten Fiftyfold (31 May 2007), Cocteau Twins - When Mama Was Moth (31 May 2007), Klaxons - It's Not Over Yet (31 May 2007), Klaxons - Magick (31 May 2007), Klaxons - Forgotten Works (31 May 2007), Klaxons - Gravity's Rainbow (31 May 2007), Klaxons - Isle of Her (31 May 2007), Klaxons - As Above, So Below (31 May 2007), Klaxons - Totem on the Timeline (30 May 2007), Klaxons - Golden Skans (30 May 2007), Klaxons - Atlantis to Interzone (30 May 2007), Klaxons - Two Receivers (30 May 2007), Pink Floyd - Flaming (30 May 2007), Pink Floyd - Matilda Mother (30 May 2007), and Pink Floyd - Lucifer Sam (30 May 2007). The page includes navigation links for 'Prev', '1', '607', '608', '609', '857', and 'Next'.

Track Name	Date
Cocteau Twins - My Love Paramour	31 May 2007
Cocteau Twins - Multifolied	31 May 2007
Cocteau Twins - The Tinderbox (Of a Heart)	31 May 2007
Cocteau Twins - In the Gold Dust Rush	31 May 2007
Cocteau Twins - Glass Candle Grenades	31 May 2007
Cocteau Twins - In Our Angelhood	31 May 2007
Cocteau Twins - Sugar Hiccup	31 May 2007
Cocteau Twins - Five Ten Fiftyfold	31 May 2007
Cocteau Twins - When Mama Was Moth	31 May 2007
Klaxons - It's Not Over Yet	31 May 2007
Klaxons - Magick	31 May 2007
Klaxons - Forgotten Works	31 May 2007
Klaxons - Gravity's Rainbow	31 May 2007
Klaxons - Isle of Her	31 May 2007
Klaxons - As Above, So Below	31 May 2007
Klaxons - Totem on the Timeline	30 May 2007
Klaxons - Golden Skans	30 May 2007
Klaxons - Atlantis to Interzone	30 May 2007
Klaxons - Two Receivers	30 May 2007
Pink Floyd - Flaming	30 May 2007
Pink Floyd - Matilda Mother	30 May 2007
Pink Floyd - Lucifer Sam	30 May 2007

Figure 3.1: Last.fm presents to the user his listening history in this form.

This makes our measure of points in time effectively a subset  $\mathbb{N}_u$  of the natural numbers, with a lower bound as the number of seconds elapsed since the *epoch event* up to the time of the user's first *scrobble*, and the upper bound set to the number of seconds elapsed since the epoch, up to the time of the user's latest *scrobble*. The set  $S_u$  contains all the songs that belong to the user's profile:

$$p_u : \mathbb{N}_u \rightarrow S_u \quad (3.1)$$

## Problem Definition

With the mapping defined as:

$$p_u(t) = s \quad (3.2)$$

such that song  $s$  was being listened by the user  $u$  at time  $t$ . If no song is registered by  $u$  at time  $t$ , we can say that a null element is mapped to  $t$ <sup>1</sup>. The set  $S_u$  contains artist-song tuples. What follows is an example of a partial Last.fm user listening profile:

$u$	$t \in \mathbb{N}_u$	$p_u(t) \in S_u$	
ticklemynausea	1284180564	Amusement Parks on Fire	Inside Out
ticklemynausea	1284180888	Amusement Parks on Fire	Raphael
ticklemynausea	1284181184	Slowdive	Here She Comes
ticklemynausea	1284181325	Slowdive	Souvlaki Space Station
ticklemynausea	1284181684	Slowdive	When The Sun Hits
ticklemynausea	1284188504	Slowdive	Crazy for You
ticklemynausea	1284188865	Slowdive	Miranda
ticklemynausea	1284193723	Slowdive	Souvlaki Space Station
ticklemynausea	1284194779	Neil Halstead	No Mercy For The Muse
ticklemynausea	1284216450	Neil Halstead	Sometimes The Wheels

Table 3.1: A segment of a user listening profile

For this profile,  $p_u(1284193723)$  maps to *Slowdive - Souvlaki Space Station* and  $p_u(1284193799)$  maps to the null element. This does not mean that the user was not listening to music at this point in time, it merely indicates that no song was registered by Last.fm as being played at that point in time.

### 3.1.2 Artist profiles

We define the map between the set of all artists and the set of all tags that belong to artists as:

$$t : A \rightarrow S \quad (3.3)$$

With the mapping defined as:

$$t(a) = S_a \quad (3.4)$$

$S_a$  is the set of all tags that belong to artist  $a$ .

<sup>1</sup>Although the user could still be listening to a song at that particular time. Last.fm doesn't keep track of the durations of the songs in the *scrobbling* process, so these timestamps represent the time a user was listening to a particular song

## Problem Definition

Also, we define a map between an artist/tag pair  $a, t$ , and the corresponding weight of the association between  $a$  and  $t$  as:

$$w : A \times T \rightarrow N \quad (3.5)$$

with the mapping:

$$w_a(t) = n \quad (3.6)$$

Such that the tag  $t$  applied to the artist  $a$  has the weight  $n$ .  $w$  has the range  $[0, 100]$ .

$a$	$tag$	$w_a(tag)$
Slowdive	shoegaze	100
Slowdive	dreampop	49
Slowdive	indie	23
Slowdive	ambient	20
Amusement Parks on Fire	shoegaze	100
Amusement Parks on Fire	post-rock	30
Amusement Parks on Fire	indie	28
Amusement Parks on Fire	alternative	15
Neil Halstead	singer-songwriter	100
Neil Halstead	folk	93
Neil Halstead	indie	72
Neil Halstead	alt-country	38

Table 3.2: Several partial artist tag profiles

For the universe of tags above,  $t(\text{Slowdive})$  maps to a set containing the tags *shoegaze*, *dreampop*, *indie* and *ambient*, while  $w_{\text{Slowdive}}(\text{shoegaze})$  maps to the natural number 100.

### 3.1.3 Mapping times to genres

Define the tag-time mapping function for a user  $u$  as

$$tt_u : N_u \rightarrow S_a \quad (3.7)$$

with the mapping:

$$tt_u(n) = \{t(s), \forall s \in p_u(n)\} \quad (3.8)$$

## 3.2 Deriving genre information from the Last.fm social tags

As we discussed in 1.1.2, not all tags convey genre information about a particular artist. Also, the social aspect of tags in Last.fm might pollute the tag clouds of various artist,

## Problem Definition

$u$	$n \in \mathbb{N}_u$	$tt_u(n)$
ticklemynausea	1284180564	shoegaze, post-rock, indie, alternative
ticklemynausea	1284180888	shoegaze, post-rock, indie, alternative
ticklemynausea	1284181184	shoegaze, dreampop, indie, ambient
ticklemynausea	1284181325	shoegaze, dreampop, indie, ambient
ticklemynausea	1284181684	shoegaze, dreampop, indie, ambient
ticklemynausea	1284188504	shoegaze, dreampop, indie, ambient
ticklemynausea	1284188865	shoegaze, dreampop, indie, ambient
ticklemynausea	1284193723	shoegaze, dreampop, indie, ambient
ticklemynausea	1284194779	singer-songwriter, folk, indie, alt-country
ticklemynausea	1284216450	singer-songwriter, folk, indie, alt-country

Table 3.3: Composition of a user listening profile with the artist tag profiles of the artists in his listening history

as some users will assign erroneous tags to particular artists. Let's consider a somewhat extreme example of an artist whose tag cloud is filled with garbage.

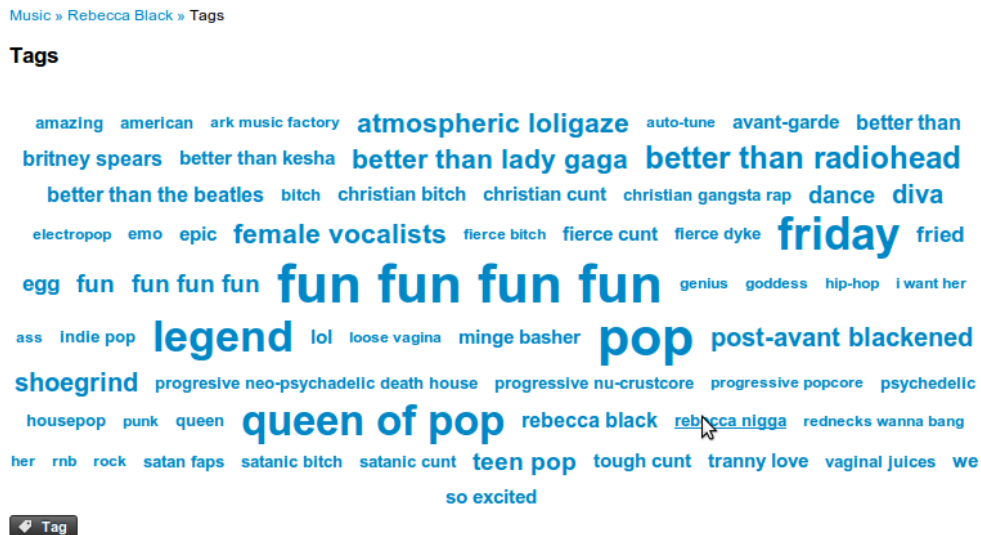


Figure 3.2: The tag cloud for Rebecca Black, teen pop phenomenon.

In fact, [Cel08] showed that users deliberately mistag items in order to provoke some (usually undesired) effect. Table 3.4 shows that in the top 10 tags for this artist, only two convey information about musical facets ("pop" and "female vocalists"), and of these, only one is a valid music genre. With lower weights on the tag cloud are three genre tags that certainly do not apply to Rebecca's music.

One way to deal with this is to set a high enough threshold so that only the tags that most users agree on are considered. The example above can't be completely solved by setting a high enough threshold, though it filters most garbage out. A way to work around this problem is to have a previously defined set of valid genres. We expect that the use

## Problem Definition

tag	weight
fun fun fun fun	100
pop	93
Friday	69
legend	67
queen of pop	60
better than radiohead	32
atmospheric loligaze	25
better than lady gaga	25
female vocalists	25
diva	23
...	
hip-hop	10
indie pop	9
punk	4

Table 3.4: Some tags for the artist Rebecca Black and respective weights.

of this set, together with a high enough threshold, will filter out the noise for most artists. The list of genres we will consider valid contains 1645 words we obtained from Wikipedia (removing hyphenation and spaces to cancel the ambiguity between tags such as *hip hop*, *hiphop* and *hip-hop*).

This process will benefit from the work done by Lima, João<sup>2</sup>. Thanks to it we have a fully developed process for crawling Last.fm artist information and several databases kept up to date with artist tag information. However, not all artists we will encounter when crawling user profiles will be present in our database. We will use fresh Last.fm information to "cover the holes" and complete the database in such cases, as described in 5.1.4.

### 3.3 Information hidden in the user's profiles

Our central belief is that there is a lot of meaningful information related to the users listening habits hiding among the lines and lines of data that make up their profiles. This section lists some aspects of music listening habits that can be looked for in a particular user profile (to discover details about that particular user's habits) or in a set of profiles (to discover details in a particular user, population or population sample).

---

<sup>2</sup>check [dCL09]

### 3.3.1 When do people listen to music?

The question we are asking is, *is there any relation between time and the music a given person listens to?* To answer this question, we must first define what we mean by *time* and what we mean by *the music listened to*.

In the section above we described in detail the data Last.fm provides us, which is a very simple mapping between points in time (described by a timestamp with the accuracy of a second) and the song that Last.fm registered as playing at that time.

If the limitation of the timestamps we described in 1.1.1 is not an issue (because we know the user or users' UTC offset), we can explore the circular (repeating) nature of particular hours or times of the day (morning, afternoon or night), weeks (work days or weekends) and months (which would reflect the impact of seasonal changes in the user's listening habits). It is worth noting that the timestamp limitation we identified becomes less of an issue when the granularity of time becomes larger than twelve hours (which is the maximum UTC offset).

What follows is a face to face comparison of two Last.fm profiles regarding the time-line aspect of the problem.

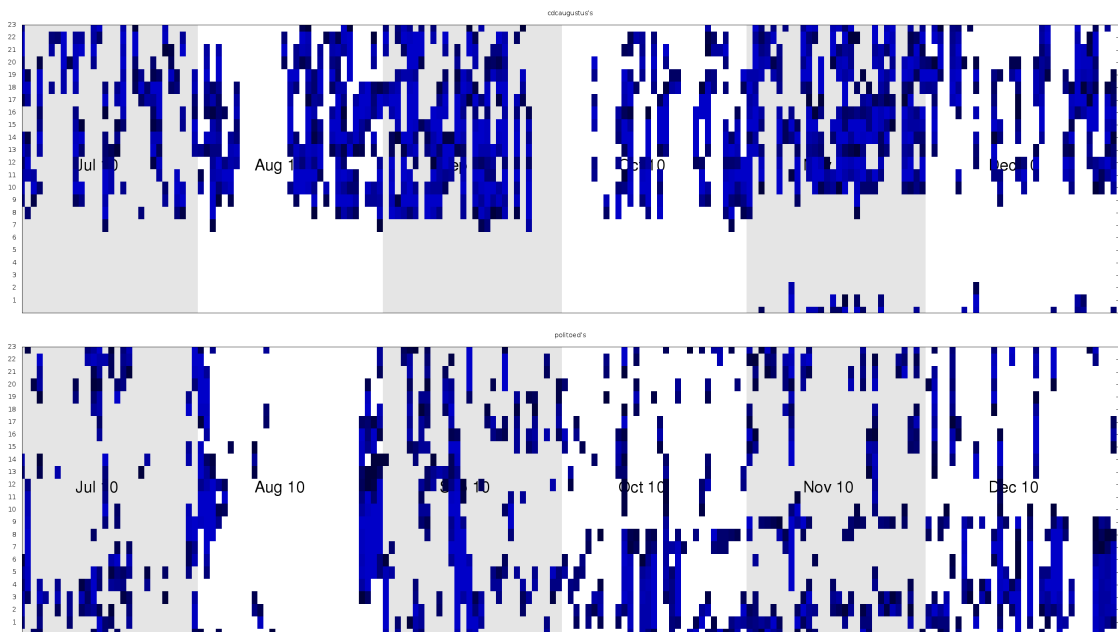


Figure 3.3: Timeline for two particular last.fm users. The horizontal axis represents days while the the vertical axis represents the time (hours). Each blue square marks a day hour where music has been listened to. The intensity of the blue represents the number of songs listened to. The images represent a profile sample taken from July to December 2010.

These images sum up some of the aspects referred to above. The regularity in the image above shows that the user seldom listens to music during sleep hours. The user

below shows no regularity of this kind.

### 3.3.2 The relation between time and the music people listen to

The time-related aspects referred to above can be extended to encompass the other part of the question left open in the beginning of this section: the music people listen to. In fact, the following questions come to mind:

- Is there any relationship between the aspects of time referred above and the genres that a user is familiar with?
- Which sequences (or sets) of genres usually appear together? Which genres do people mix together in the same music listening session?

These aspects can always be extended to include demographic information available to us via the Last.fm API, like **gender**, **age** and **country**. We have however to keep in mind that this information might be incorrect, as it is (obviously) subjected to no kind of verification.

### 3.3.3 Analyzing music listening sessions

A music listening session is a period of time in which a person listens to several songs one after the other. In we presented the idea that an entire user's profile can be divided into several sessions. As it was explained there, the division is based on the played media items' timestamps. It might be interesting to look for association rules<sup>5.1.2</sup> between musical genres in a user's or in a population of users' session data. This would enable us to see (for the user or users represented in the analyzed dataset) which genres are mixed together more often.

## Summary

In this chapter we explained how we approached the problem of extracting valid genre information from the artists' tags, which often are polluted with garbage tags using threshold values and a list of pre-compiled genres. This chapter also introduced the final goals of this project, which are to investigate if time does in fact influence people's choice of genre and if the genres themselves influence the choice of other genres.



## Chapter 4

# Research Dataset

This section details the process of the construction of a dataset containing the complete listening history of several hundreds of Last.fm users. This process involved several different uses of the Last.fm Web API, already introduced in [1.1](#). We used the Python programming language to create automated and scalable scripts that gather Last.fm profiles in a MySQL database. This section also details the filtering process applied to the retrieved data, in order to ensure that the profiles present in the dataset are a faithful representation of the user's listening habits. We conclude with a series of charts that illustrate some of the characteristics of the gathered dataset.

### 4.1 The Crawling Application

The crawling application was developed with Python 2.6.5, but it also proved compatible with Python 2.7. It uses MySQLdb, a python MySQL client, and a custom fork of PyLast for communication with the Last.fm API<sup>1</sup>.

---

<sup>1</sup>Some features were missing and some needed to be fixed. More in [4.1.2](#).

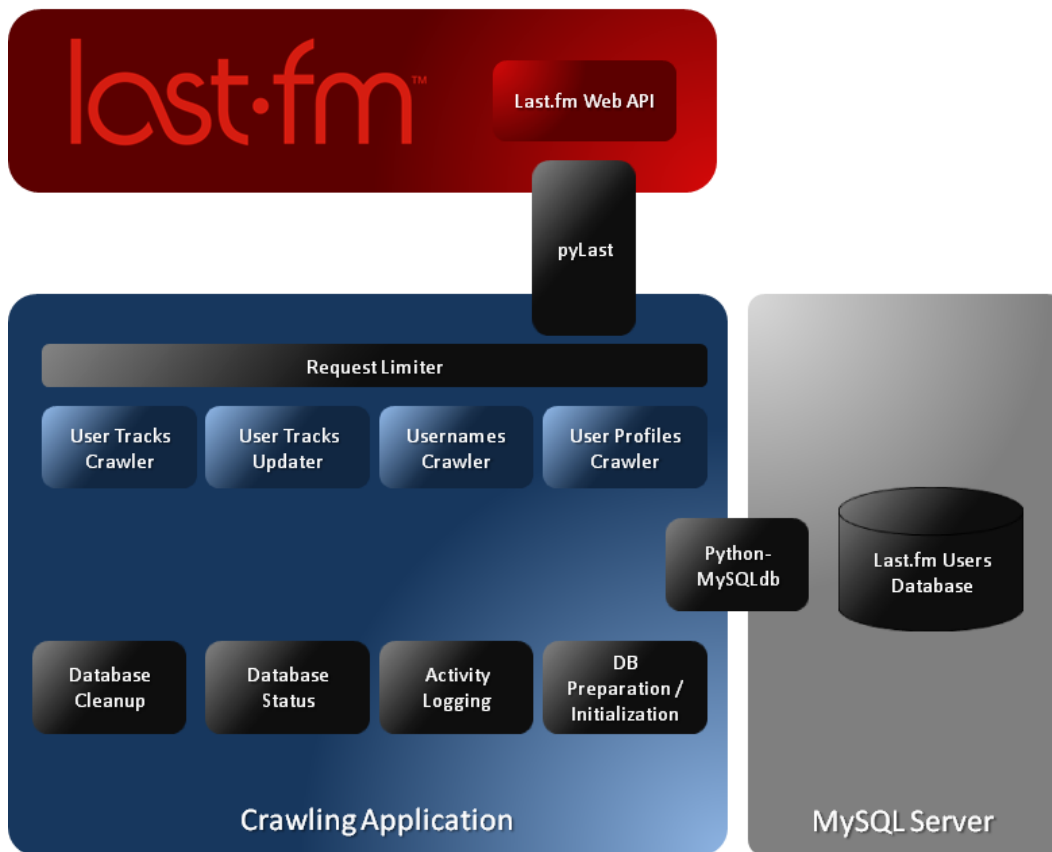


Figure 4.1: Overall view of the crawling application modules and interaction with the other systems involved in the process: the Last.fm API and the MySQL database server

Figure 4.1 shows the interaction between the crawling application modules and the other systems involved in this process. The crawling application is composed of nine modules. These are:

- **Usernames crawler**, which performs a crawling of Last.fm usernames;
- **User profiles crawler**, which performs a crawling of Last.fm profile data, like age, gender, country, etc;
- **User tracks crawler**, which performs a crawling of the entire (or part of) a user's profile up to his latest scrobble;
- **User tracks updater**, which updates the database with the scrobbles in his profile that weren't yet present when the user tracks crawler module retrieved his information;
- **Request Limiter**, responsible for limiting the number of requests each crawling script can make to the Last.fm API in a pre-determined window of time (for compliance with the API's terms of use)

- **Activity Logging**, responsible for logging the activity of all running scripts;
- **Database Status**, which shows the status of the crawling activity;
- **Database Initialization**, which is responsible for initializing the database for the crawling process (creation of tables and indexes and insertion of seed usernames);
- **Database Cleanup**, responsible for the cleanup of forcibly interrupted crawling activity.

The particular way these modules were used in the construction of the research dataset is detailed in [4.2](#).

#### 4.1.1 The Last.fm API

Last.fm's web services provide a public API that *allows anyone to build their own programs using Last.fm data, whether they're on the web, the desktop or mobile devices*. The Last.fm API allows programmers to call methods that respond in REST style XML or JSON. For example, an HTTP GET request to the following URL:

```
http://ws.audioscrobbler.com/2.0/?api_key=MY_API_KEY  
&method=user.getrecenttracks&user=ticklemynausea&page=1&limit=1
```

## Research Dataset

Would return this as an answer:

```
<?xml version="1.0" encoding="utf-8"?>
<lfm status="ok">
<recenttracks user="ticklemynausea" page="1" perPage="1" totalPages="45209">
<track>
  <artist mbid="a6bf1276-9150-40fc-a94e-6b14f377fe3d">Airiel</artist>
<name>Jeanette</name>
<streamable>1</streamable>
<mbid></mbid>
<album mbid="">Melted</album>
<url>http://www.last.fm/music/Airiel/_/Jeanette</url>
  <image size="small">http://userserve-ak.last.fm/serve/34s/8777317.jpg</image>
  <image size="medium">http://userserve-ak.last.fm/serve/64s/8777317.jpg</image>
  <image size="large">http://userserve-ak.last.fm/serve/126/8777317.jpg</image>
  <image size="extralarge">http://userserve-ak.last.fm/serve/300x300/8777317.jpg</image>
  <date uts="1289497088">11 Nov 2010, 17:38</date>
</track>
</recenttracks></lfm>
```

All calls to the Last.fm API are made through the PyLast library, which uses the object-oriented paradigm to generate the requests and encapsulate the responses, in order to allow the programmer to abstract himself from the parsing of the XML or JSON responses.

Here is a summary of all the methods and respective parameters that were relevant to this work.<sup>2</sup>

---

User.getFriends	
Description	Returns the usernames of the users that are friends with the given user
Relevance	Useful in the discovery of new usernames
<b>Parameters</b>	
username	A Last.fm username

---

Table 4.1: Summary of the User.getFriends API method

---

User.getInfo	
Description	Returns the profile information of the given user
Relevance	Useful for discovering the user's demographics, like age or country of origin
<b>Parameters</b>	
username	A Last.fm username

---

Table 4.2: Summary of the User.getInfo API method

---

<sup>2</sup>The API specification is available online at <http://www.last.fm/api/>

<code>User.getRecentTracks</code>	
Description	Returns a list of the recent tracks listened to by this user. "Recent" is actually a misnomer, as this extends to the whole user profile.
Relevance	Used in crawling the user's listening profile. Also returns the total number of pages/tracks present in the user profile.
Parameters	
<code>username</code>	A Last.fm username
<code>limit</code>	Number of tracks to be retrieved (on a per-page basis)
<code>page</code>	Number of the page to be retrieved
<code>from</code>	UTC timestamp. Only tracks scrobbled after this point in time are returned
<code>to</code>	UTC timestamp. Only tracks scrobbled before this point in time are returned

Table 4.3: Summary of the `User.getInfo` API method

#### 4.1.2 Technologies

The following is a summary of the technologies involved in this project.

- Python Programming Language** Python is an interpreted, high-level, general-purpose, object oriented programming language. It supports very different classes of problems and has a very simple syntax, as Python's design philosophy is centered around code readability. Perhaps Python's most known (and weird) feature is the use of code indentation as block delimiters. Python's standard library is extensive, and covers areas such as string processing (Unicode, regular expressions), native support for several Internet protocols, unit testing, profiling and operating system interfaces. The reference implementation of Python, CPython, is free software, and is licensed by the Python Software Foundation License. Python is cross-platform, and is available in all major operating systems. Python supports multiple programming paradigms, and has a dynamic typing system. Moreover, python's popularity has made it that there are several third party libraries with support for the interfaces needed for this work: The MySQL relational database system and the Last.fm Web API. The programmer's experience with Python was also determinant in the choice to use it as the main programming language, as it allowed for the fast development and debugging of all the modules. Moreover, python was the language of choice for all the tasks in this projected that to any extent involved programming.
- MySQL Database Server** MySQL is a cross-platform relational database management system that uses a client-server architecture and provides multi-user access to a number of databases. It proved reliable and efficient, however it was very slow when restoring databases from eventual database crashes we had due to power outages.

- **MySQLdb** is a thread-safe Python interface for the MySQL database server. MySQLdb provides support for Python versions 2.5 through to 2.7 and for MySQL versions 3.23 through 5.1. It implements the Python Database API specification 2.0.
- **PyLast** PyLast is an open source implementation of the Last.fm Web API for Python. It supports access to all the data accessible through the Last.fm Web API. A few quirks were detected that motivated further development on this library:
  - Fixed HTTP proxy support, which wasn't working reliably and was required at the time
  - Added support for the paged retrieval of user track profiles in the API method `User.getRecentTracks`
  - Added support for the *from* and *to* parameters of the API method `User.getRecentTracks`

#### 4.1.3 Summary of features

The following is a list of features implemented by the crawling application. A list of the files available in the installable package is available in [A.1](#).

- Discovery of Last.fm usernames through any number of *seed* users and through the `User.getFriends` API call. Username discovery is performed in a breadth-first fashion, i.e, first the seed's friends are gathered, then the seed's friends' friends are gathered, etc.
- Retrieval of Last.fm user profile data (name, age, homepage, etc.)
- Retrieval of Last.fm user listening data, either partially or in their entirety
- Updating of Last.fm user listening data, collecting all the new tracks that were scrobbed since the last crawl
- Designed with scalability in mind, as any number of instances of the same script may be running concurrently to speed up the crawling process
- All running scripts are aware of the limits imposed by the Last.fm Web API's terms of use, as no more than five requests per second should originate from the same IP address
- Configurable through a text file, with no hard-coded parameters
- Maintains consistency of data through a cleanup script and through a rollback routine that is executed on script interruption

#### 4.1.4 Compliance with the Last.fm API Terms of Use

The terms of use of the Last.fm API require that no client shall make more than five requests per second, averaged over a period of five minutes. This is respected by the crawling application without harming its scalability.

Every time a request is made, a single call is made to a function `tick()`. This function checks to see if the number of requests has exceeded the allowed five requests per second per five minutes. If this is the case, the function will halt for as many seconds as necessary until the remainder of the window of five minutes has passed.

This is achieved using access to a MySQL table as a data structure that can be shared between all concurrent processes. This table is stored using the InnoDB transactional storage engine, which supports the lock/unlock mechanism necessary to correctly support the concurrent access to this table. This table has a property-value structure, and the API compliance system requires the use of two properties known to all concurrent processes:

- `request_counter`, A counter of all requests made to the Last.fm Web API
  
- `request_clock`, The time of the next projected reset to the request counter

Before the crawling process begins, both fields are set to 0. This algorithm, used by `tick()`, is called after every call to the Last.fm API. The algorithm used in every call to `tick()` is explained in below:

---

**Algorithm 1** Request limiting algorithm

---

**function** *S*: A data structure shared between all processes that keeps two fields: a request counter, the current number of requests made and an expiration date, the time where the counter should be set to zero

**function** *get\_current\_time*(): returns the UNIX timestamp for the current time

**function** *request\_window\_duration*: as specified by the terms of use, is 5 minutes

**function** *max\_requests*: 1500 requests (a maximum of five requests per second in five minutes)

*request\_counter*  $\leftarrow$  *S.read\_request\_counter*() + 1

*expire\_time*  $\leftarrow$  *S.read\_request\_clock*()

*current\_time*  $\leftarrow$  *get\_current\_time*()

if the condition below is true, then the five minutes have already passed without the limit being hit. *S* should be updated in this case

**if** *current\_time* > *expire\_time* **then**

*projected\_next\_reset*  $\leftarrow$  *current\_time* + *request\_window\_duration*

*S.set\_request\_clock*(*projected\_next\_reset*)

*expire\_time* = *projected\_next\_reset*

**end if**

if we hit the limit, we set the request counter to zero, but not before sleeping. this way, other processes will also hit the limit and go to sleep

**if** *request\_counter*  $\geq$  *max\_requests* **then**

*sleep\_duration* = *expire\_time* - *current\_time*

*pause*(*sleep\_duration*)

*S.set\_request\_counter*(0)

**else**

*S.set\_request\_counter*(*request\_counter*)

**end if**

---

## 4.2 Methodology

The dataset construction process begins with the discovery Last.fm usernames using the `User.getFriends` API method and a given number of seed users. When enough usernames are collected, the first crawling phase begins: we use the `User.getInfo` API method to retrieve information about each known username that can then be used in a first filtering process.

After the first filtering process is concluded, an iterative process begins with the selection of random users (sampling), the retrieval of their entire listening profiles and afterwards, a second filtering process using the information retrieved previously. These three



processes repeat, until the number of accepted profiles reaches the desired amount. Figure 4.2 illustrates this process.

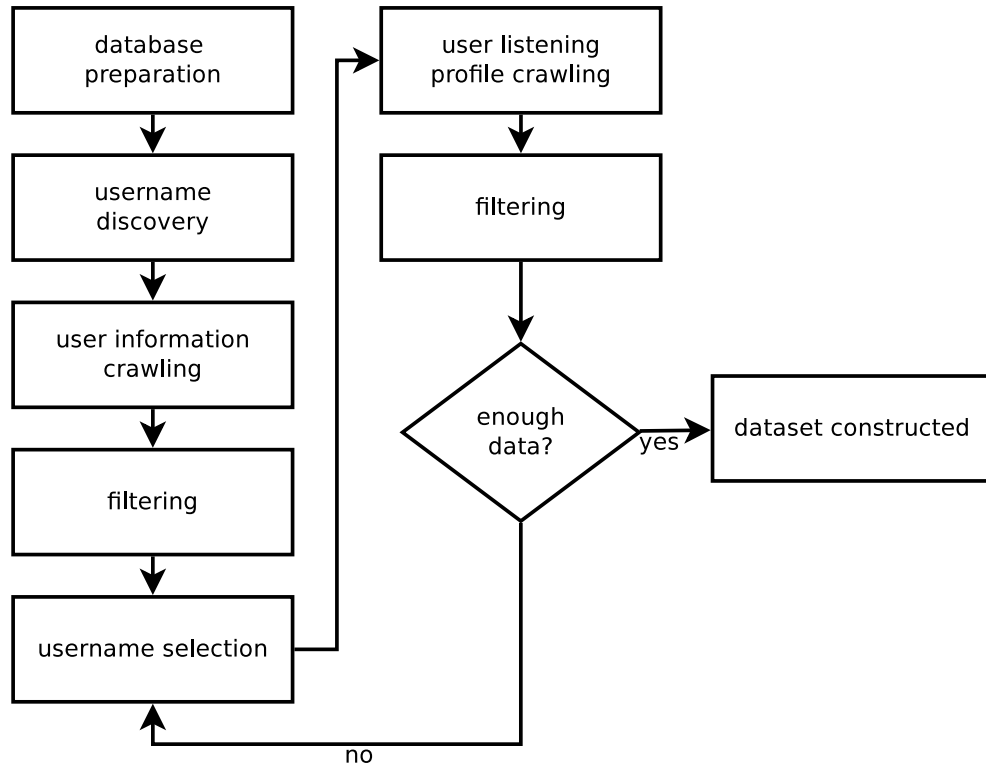


Figure 4.2: Flowchart of the dataset construction process

#### 4.2.1 Database preparation

The database preparation phase involves the creation of the tables, according to the specification illustrated in 4.3. It also involves the insertion of the *seed* usernames in the *known\_users* table, and the creation of the index structures that optimize the necessary time for data retrieval, at the expense of disk space and insertion time. In this case indexes are crucial, since these tables will grow to host several million lines. Excluding the index created in the case of a table's primary key (already shown in 4.3), the fields *username* and *artistname* in the *user\_tracks* table were also indexed, in order to allow a quicker retrieval of rows based on criteria that uses those two fields.

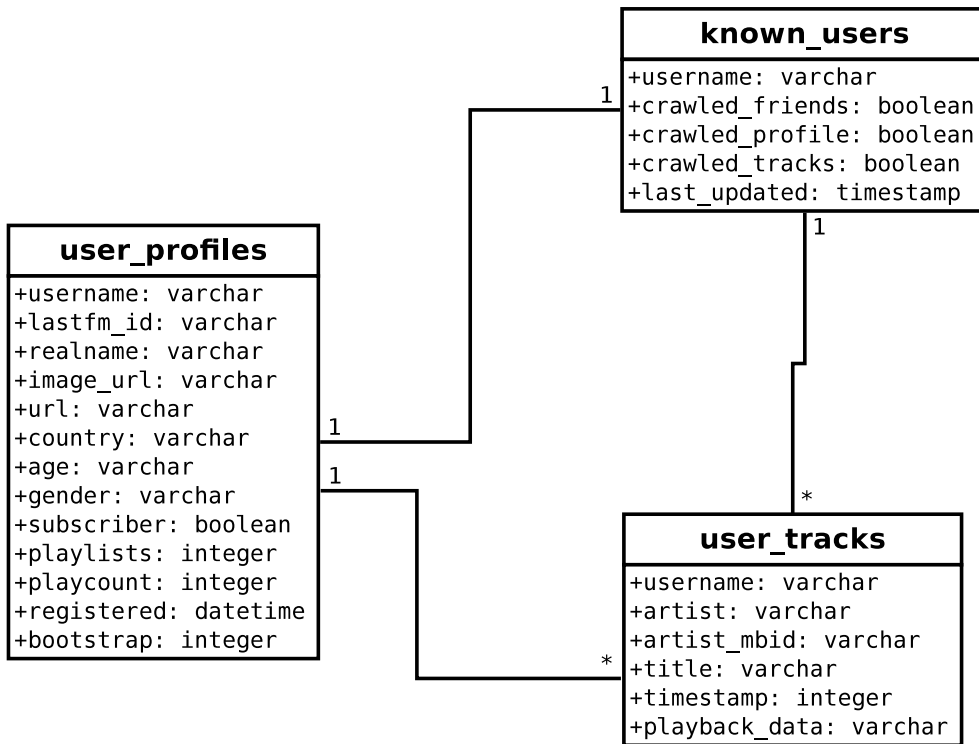


Figure 4.3: Database diagram

## 4.2.2 Username discovery

Username discovery is achieved through the `User.getFriends` Last.fm Web API method. This method returns the list of users that are friends with a given user. We can then make the same API call to the previously retrieved usernames, retrieving the friends of our friends.

This process is performed in a breadth-first fashion (Algorithm 2), i.e, first the seed’s friends are gathered, then the seed’s friends’ friends are gathered, etc. Results are stored in the `known_users` table<sup>4.3</sup> and this process is supported by the `crawl_usernames` script.

It is interesting to notice that a process like this would not stop until all the usernames that are part of the connected components (of the social network graph) the seed users belong to have been discovered. The crawling application gathered a total number of 1,667,314 usernames over approximately a week, and was manually terminated afterwards.

---

**Algorithm 2** Username discovery algorithm

---

```

function get_next_username() : Returns the next uncrawled username
function get_friends_of(username) : Returns the list of friends of username
function mark_as_crawled() : Marks the given user as examined

username_list  $\leftarrow \emptyset$ 
while there are unexamined friends do
  username  $\leftarrow$  get_next_username()
  friends  $\leftarrow$  get_friends_of(username)
  for friend in friends do
    if friend  $\notin$  username_list then
      username_list  $\leftarrow$  username_list + friend
      mark_as_crawled(friend)
    end if
  end for
end while

```

---

### 4.2.3 User information retrieval

The user information retrieval process retrieves personal information present in the Last.fm profiles that does not include the listening data. Specifically, this information consists of the fields listed in 4.4.

- The user's real name
- The user's profile image url
- The user's country of origin
- The user's age
- The user's gender
- The user's total played items
- The user's total number of playlists<sup>3</sup>
- The user's registration date
- The user is or isn't a Last.fm premium subscriber

Figure 4.4: List of fields retrieved by the Last.fm Web API method `User.getInfo`

The data is stored on the `user_profiles` table<sup>4.3</sup>, and the crawling process is supported by the `crawl_userprofiles` script. Of all the information fields that were retrieved, only the total played items and registration date are useful to the process that follows. Of the 1,667,314 usernames known, the crawling application queried the API for the information above for a total number of 1,135,615 users, effectively discarding the rest. This information is stored in the `user_profiles` table and is supported by the `crawl_userprofiles` script.

#### 4.2.4 Filtering

We decided that our dataset would contain no users that do not specify a country of origin, as using this information could be used to circumvent the timestamp issue we encountered<sup>1.1.1</sup> before. Using the country information is, in fact, a possible way to approximate the UTC timestamp in the Last.fm logs to the user's local time<sup>4</sup>.

We used a rough approximation of the number of tracks played per day (averaged using the registration date and total played items) to decide if the the profile contains enough data to be used in the process of retrieval of any significant information that we may find regarding our objectives. We also use the registration date to filter profiles that have not existed for long enough to contain enough data. Only profiles that are at least one year old were considered valid and survived this filtering step.

Summarily, the following sequence of steps were applied to the dataset, using the information retrieved by the `User.getInfo` Last.fm API method:

1. Excluding users that contained less than 20 and more than 160 tracks per day<sup>5</sup> yielded a set of 409,683 users.
2. Excluding users that weren't members for at least one year yielded a set of 335,956 users.
3. Excluding users that didn't specify a country of origin yielded a set of 302,897 users.

This process was still subject to some errors, as we encountered disparities between the data retrieved by the Last.fm Web API and what was actually present on the user's profiles. Namely, the total played items figure as retrieved by the API was different from what the user actually had on his profile. To ensure the quality of the dataset we produced in terms of our specified requirements, we had another filtering step that actually looks through the data present on the users' profiles.

---

<sup>4</sup>Provided that the user does not lie about his location

<sup>5</sup>Considering a rough approximation of 3 minutes per song, to users with less than 1 hour and more than 8 hours of music per day

#### 4.2.5 Random sample selection

After all profiles with undesired characteristics have been discarded, we randomly selected a smaller set of profiles that will be looked at in our final analyses. The two reasons behind this are that the retrieval of complete listening profiles is a time expensive process<sup>6</sup> and that users that are near each other in the Last.fm user network will be near each other (row wise) in the database. This could introduce bias in our dataset, since people are likely to be friends with people with similar tastes or listening habits. Of the 302,897 usernames that survived the filtering process, a random sample of 10,000 was selected for further crawling.

#### 4.2.6 Listening history crawling

Obtaining the entire listening history of a user is a time expensive process. Suppose a user has a listening history comprising 40,000 tracks. The Last.fm Web API can only retrieve 200 tracks at a time, so it would take 200 calls to the API to retrieve the entire history for this user. If we consider a reasonable little more than three seconds of waiting for a response, it's over 10 minutes for a profile of such size.

This process registers data in the `user_tracks` table and was performed using the `crawl_usertracks` script. The process gathered the complete listening history of all the 10,000 users and terminated afterwards.

#### 4.2.7 Filtering, yet again

We now possess a dataset with the complete listening history of 10,000 users. We used the information retrieved by the `User.getInfo` to determine preemptively which users would fit our established requirements. We have however verified that the information retrieved by this method may be incorrect, and as such, another filtering step is required, this time with an in-depth look through the complete listening history data. This step firstly involved the retrieval of the following information from the database:

- Total number of played tracks in the user's Last.fm profile
- The timestamps of the earliest and latest played item

And again, we applied the same filtering criteria as earlier. However, instead of using the Last.fm API's `User.getInfo`, we calculated the profile's length in time using the difference between the two retrieved timestamps and calculated the average number of played tracks using the real number of items played, as retrieved by the API method `User.getRecentTracks`.

---

<sup>6</sup>A single request to Last.fm takes about 2 seconds to complete (exarcebated if there are other crawling scripts running) and returns, at most, 200 played items. Each profile made of several thousands.

This process had to be performed iteratively until all users present in the final dataset stood by our requirements. A total number of 587 users were rejected. Our final, completely filtered research dataset has 10,000 users that comply with the requirements we specified earlier.

### 4.3 Characteristics of the final dataset

This section presents a series of charts that illustrate some of the characteristics of the dataset we gathered. These are demographic characteristics, characteristics of the user profiles (about the data we found in each profile) and artist/genre characteristics (relating to the artists and genres we found in each user profile).

#### 4.3.1 Demographic characteristics

We present an illustration of the characteristics of the dataset we gathered in terms of the demographic information available (gender, age and country).

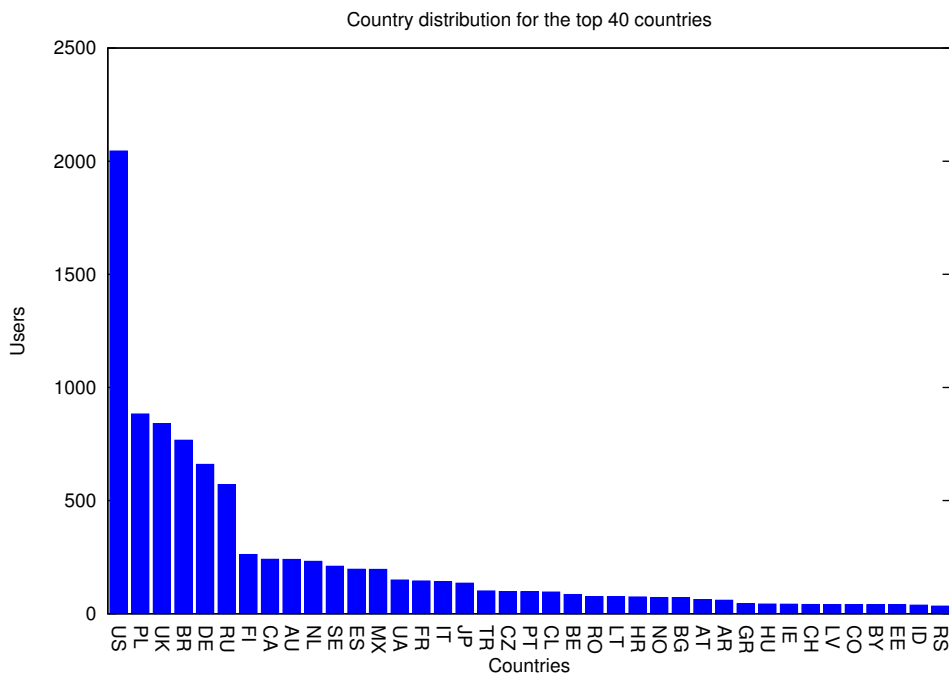


Figure 4.5: Country distribution in the final dataset (for the top 40 countries)

# Research Dataset

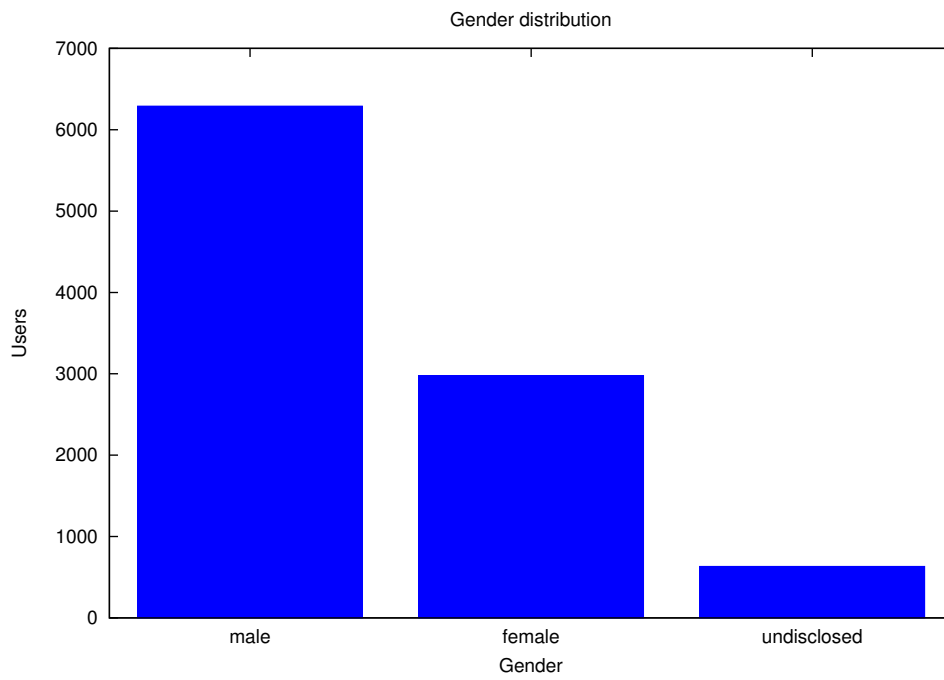


Figure 4.6: Gender distribution in the final dataset

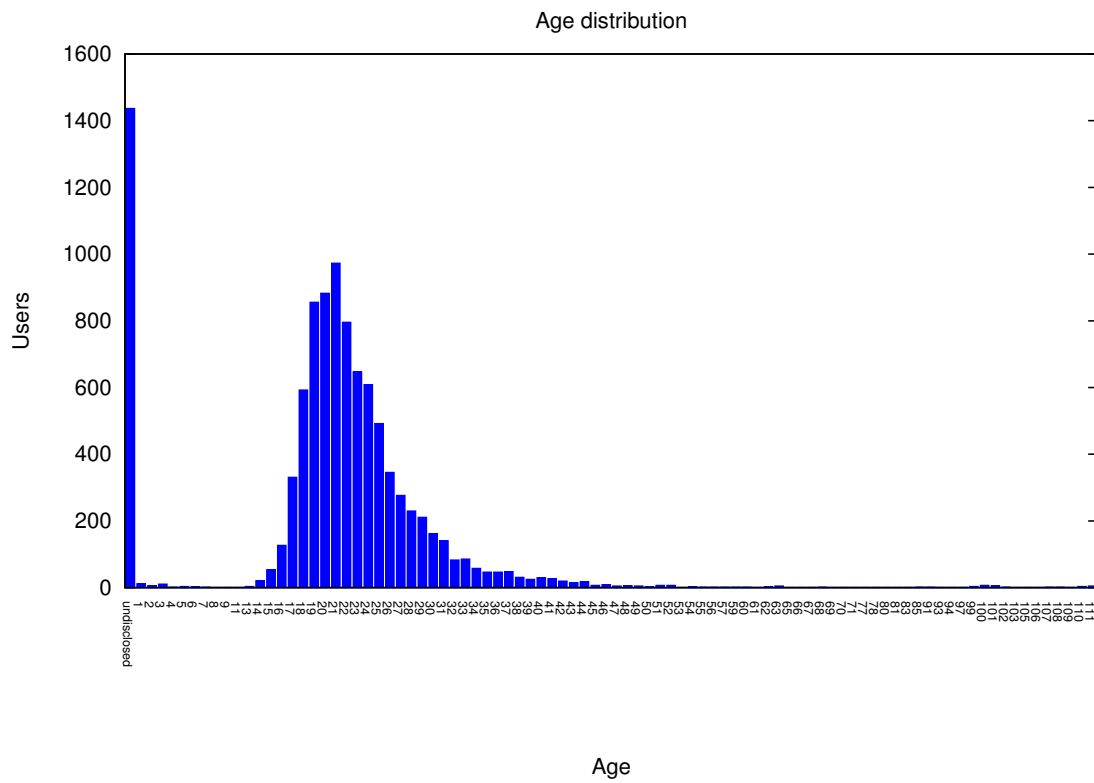


Figure 4.7: Age distribution in the final dataset

These charts summarize the demographic information retrieved for the 10,000 users in our dataset. 4.5 shows that about one fifth of the users in our dataset are from the United States. Interestingly, the second most represented country is Poland, with more users than the UK, in the third position. 4.6 shows that the majority of the users in our dataset say that they are male. 4.7 show that the ages of the users peak at 22 years old. It has to be reminded to the reader that this data is subjected to no kind of verification by Last.fm and many users may lie about their demographics<sup>7</sup>.

### 4.3.2 Characteristics of the users' profiles

We present three charts that summarize the amount of information (played items) present in each user profile.

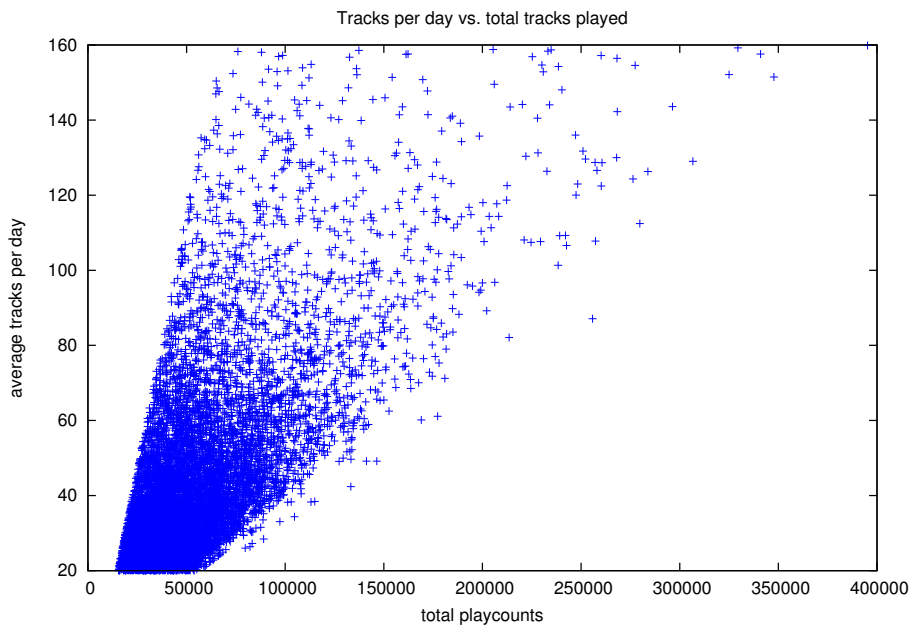


Figure 4.8: Scatter plot of total played items and average played items per day

<sup>7</sup>Also, we find hard to believe that our crawling process found so many Centenarians in the Last.fm community.



## Research Dataset

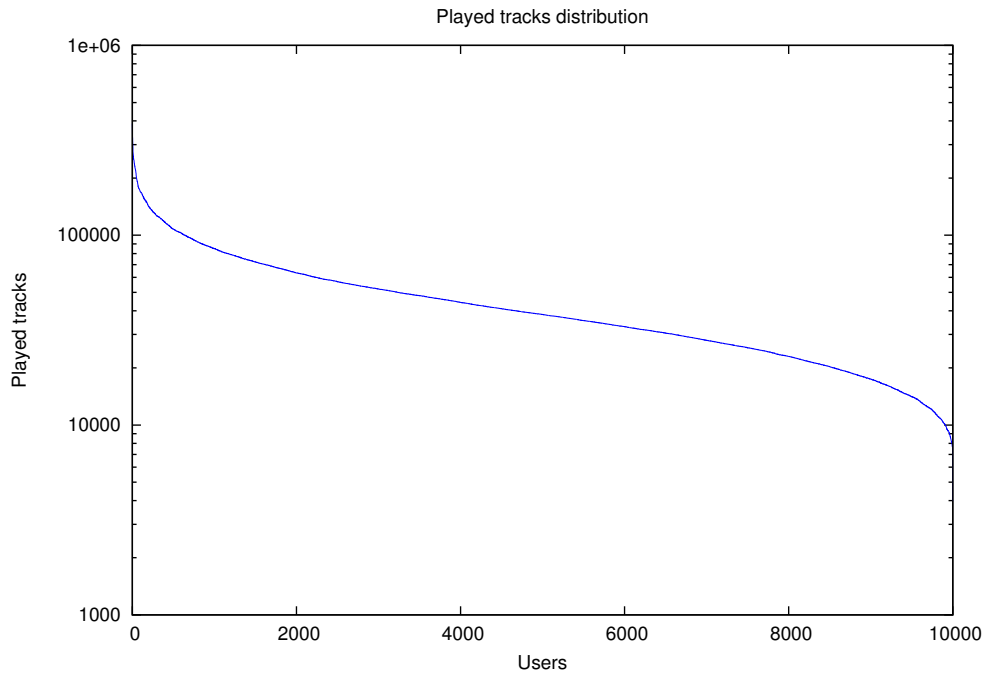


Figure 4.9: Total played items distribution per user (nominal scale on the x axis)

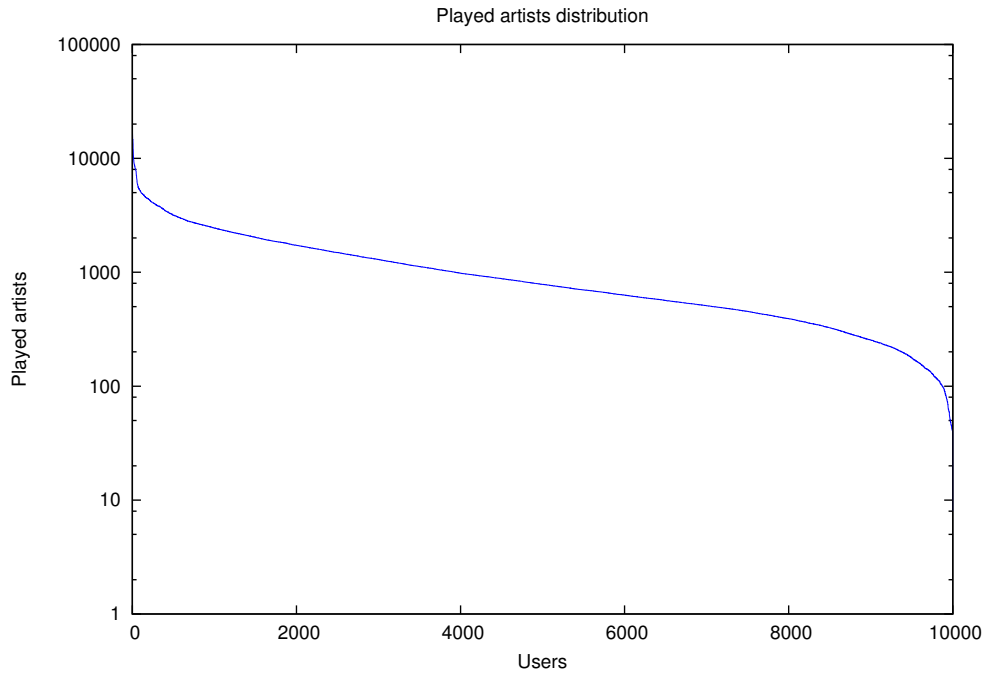


Figure 4.10: Total played artists distribution per user (nominal scale on the x axis)

4.8 shows where each user is in terms of total played items and the average of the items played daily. We can easily see that the large majority of the users are concentrated

below sixty average played tracks per day and have less than 100,000 tracks recorded in their profiles.

4.9 and 4.10 show how many played items and different artists each user listened to. These are represented using a logarithmic scale on the y axis due to the large range of values that need to be represented.

### 4.3.3 Characteristics related to the artists and genres

These charts reflect the popularity (in terms of number of played items) of the genres and artists we encountered.

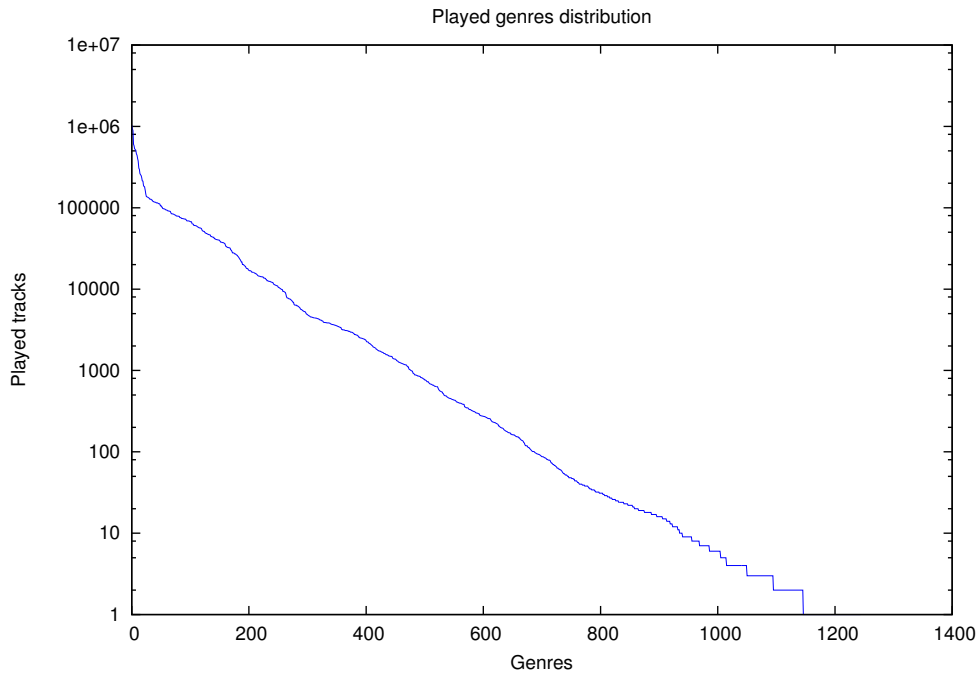


Figure 4.11: Total played tracks distribution per genre (nominal scale on the x axis)

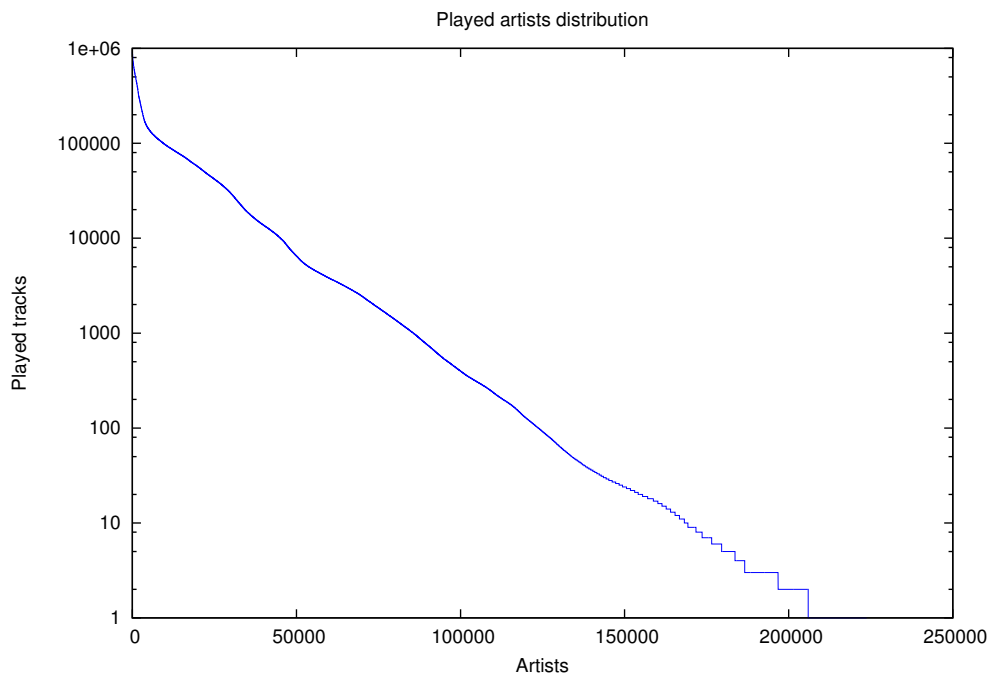


Figure 4.12: Total played tracks distribution per artist (nominal scale on the x axis)

Figures 4.11 and 4.12 present the distribution of played tracks over each genre and each artist that we’ve encountered in a logarithmic scale, due to the wide range of the values we want to represent. Both are clear examples of *long tail* distributions. In these kind of distributions, a small part of the population (in these case, artists and genres) holds the most popularity, while the largest part of the population rests within the tail where the less popular items are. Long tail distributions are often found in music consumption[[Cel08](#)]. In fact, nearly 70% percent of the artist names we found have only one played track.

There is some noise in figure 4.12, as we did not made any work towards filtering names of artists that contain typos or have any other kind of information next to the artist name in the mp3 ID3 tag<sup>8</sup>. This problem was also encountered in Lima’s work[[dCL09](#)].

---

<sup>8</sup>For example, one of these scrobbed artists is !!! (*Chk Chk Chk*), which refers to an artist whose real name is !!! but pronounced *chk chk chk*.

## Research Dataset

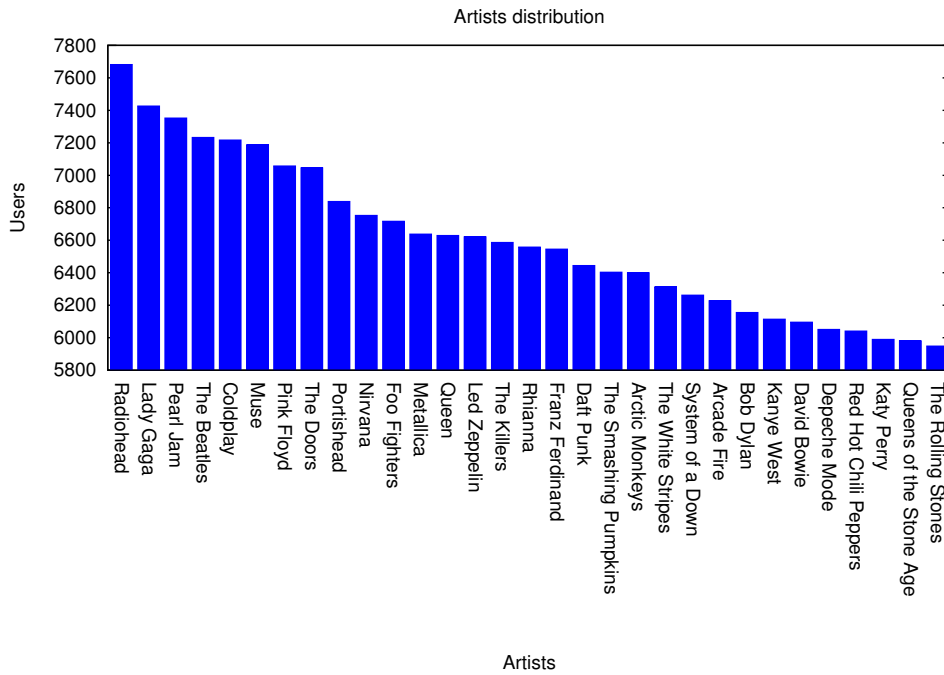


Figure 4.13: Top 30 artists represented in the dataset

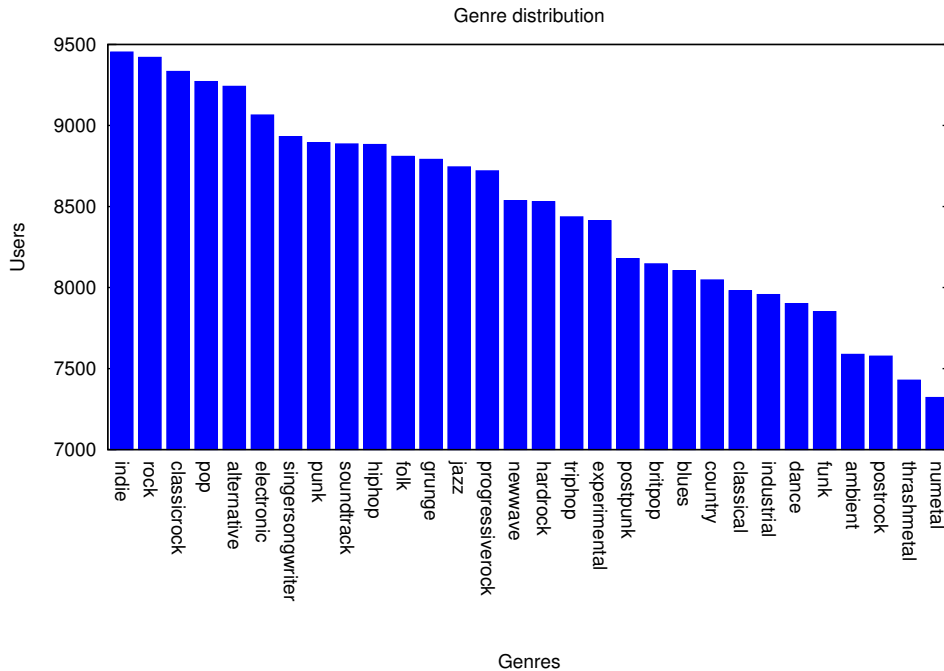


Figure 4.14: Top 30 genres represented in the dataset

4.14 and 4.13 show how many users listen to the top 30 different artists and genres represented in our dataset. These charts do not illustrate artist or genre preference, as they

do not differentiate between listening to an artist/genre one or a greater number of times and therefore indicating preference.

## Summary

In this chapter we presented some of the most relevant parts of the crawling application we developed, as well as those of the Last.fm API that the application is based upon. We showed in detail the construction process we used to build our dataset and used charts to describe some relevant features of the dataset we developed. We came to the conclusion that the distribution of played tracks of the artists and genres does in fact follow a long tail distribution. We also encountered several artist names that are misspelled, corrupted or do not represent a real artist, a problem that also Lima found (and solved) in his work[dCL09].

## Research Dataset

## Chapter 5

# Data Analysis

This chapter describes the step that followed the development of the crawler and the construction of the research dataset: extracting information from the retrieved data, the final goal of this project. It describes the methods used to process the gathered data into structures that are adequate for our final goal of knowledge discovery. Finally, it presents the two analyses we have made. We show an analysis of music listening sessions (as described in 3.3.3), where we try to determine if certain genres influence people to listen to other genres in the same session. We also show an analysis of genres with respect to time, where our goal was to find out how time influences people's choice of genre.

### 5.1 Background

#### 5.1.1 Knowledge Discovery in Databases

Knowledge Discovery in Databases (KDD) is the field concerned with the development of methods and techniques for “making sense of data”[FPsS96]. It deals with the process of mapping voluminous low-level data, which is in nature too hard to analyze into other, more compact, forms that are easier to understand. This may seem to clash with the definition of the more popular term *data mining*, as data mining itself is only one step of the larger, more encompassing process of knowledge discovery. The motivation for the development of KDD comes from the fact that copious amounts of data are more and more available nowadays. Just because we may think that information is inherently present in the data, such information is not easily accessible. KDD helps turning such data into useful information and concepts with a wide range of applications[HK00].

**The KDD process**

The KDD process is an interactive, iterative process that involves many different steps and at various points the user is required to take many decisions.[FPsS96]. The KDD process consists of an iterative sequence of seven sequential steps[HK00]:

1. **Data Cleaning** is the first step of the process. It has the responsibility of removing any noise and inconsistencies that are not relevant (and are potentially hazardous) for the overall extraction of the knowledge;
2. **Data Integration** accounts for the combination of data from multiple sources into a single source;
3. **Data Selection** retrieves the relevant data for the analysis;
4. **Data Transformation** transforms data into appropriate forms for the next process;
5. **Data Mining** consists of the processes where intelligent methods are applied to the data, in order to extract patterns;
6. **Pattern Evaluation** is the process of evaluating the patterns detected in the previous step, to determine those that contain truly valuable information;
7. **Knowledge Representation** consists of using visualization and knowledge representation techniques in order to present the extracted information.

[FPsS96] complements these seven steps with two more that precede them:

- The development of an understanding of the domain of the problem at hand and of all the relevant prior knowledge to the rest of the KDD process
- The creation of a target *data set* – the creation of the set of raw data in which the knowledge discovery process will be performed.

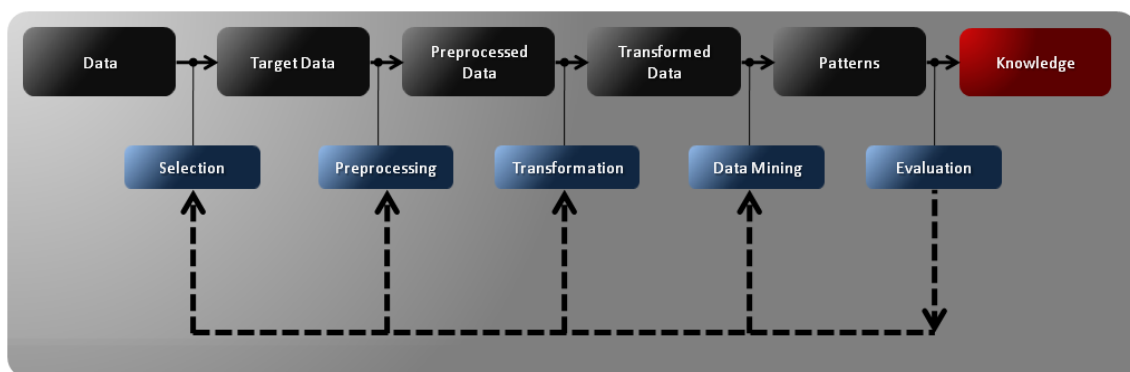


Figure 5.1: The iterative process of KDD as presented in [FPsS96].



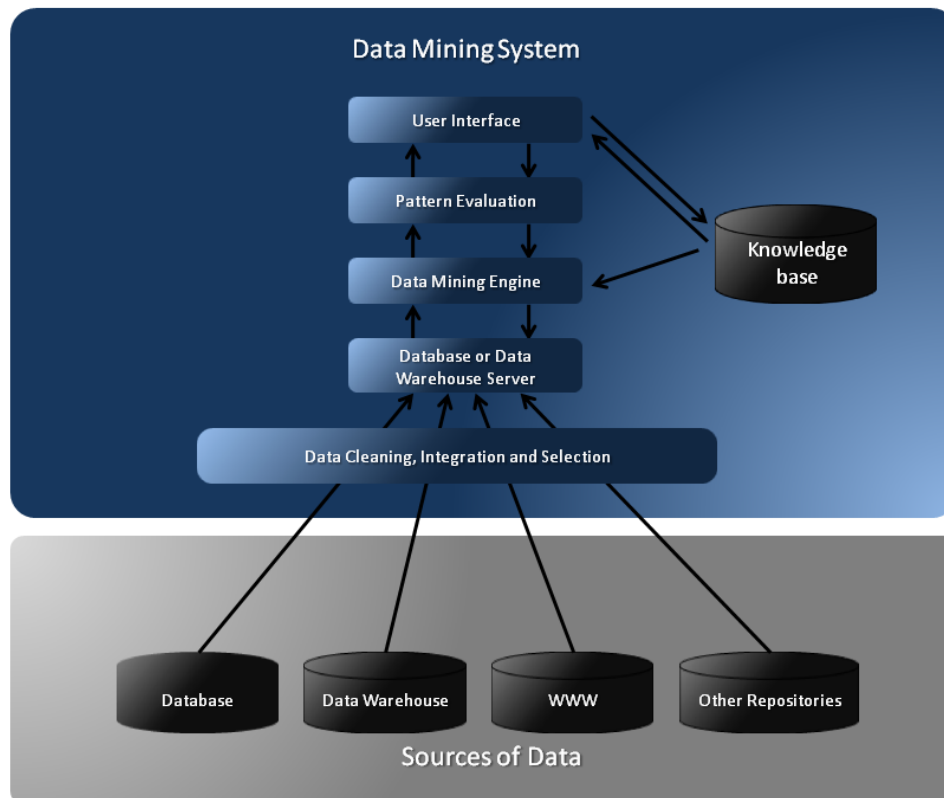


Figure 5.2: The architecture of a typical data mining system as presented by [HK00].

### 5.1.2 Mining association rules

Mining association rules is closely related to the frequent item sets problems. As the name implies, an item set is a set of items that is called *frequent* when it appears frequently in some data. The importance of frequent itemset mining is that it allows the derivation of interesting association rules between items from the raw data[Goe]. For example, if we say that an user obeys the rule "played from 8pm to 10pm" => "indie rock" with 80% support and 90% confidence, we are saying that if such user is playing a song from 8pm to 10pm, we can assert with 90% confidence that the genre he is listening to is indie rock. The 90% support value shows that 80% of the data analyzed showed this particular rule. These two values, confidence and support are therefore a measure of the interestingness of a rule.

Association rule appears in many data mining tasks where the objective is the discovery of association or correlation relationships in a large set of data items. The identification of sets of items that occur together is itself, one of the most basic tasks in Data Mining. The original motivation for the definition of the frequent set problem came

from *market basket analysis* – the need to analyze supermarket transaction data, concretely, the analysis of the customers’ preferences in terms of the products they purchase together[Goe][HK00].

The *frequent set problem* operates over  $\iota$ , a set of items. A *transaction* over  $\iota$  is a tuple  $T = (tid, I)$ , where  $tid$  is an identifier unique to each transaction and  $I$  is a set,  $I \in \iota$ . A *database*  $D$  is defined as a set of transactions over  $I$ .

Our particular goal is to apply the association rules mining problem to our particular problem, namely:

- to see if there are genres that users frequently mix together in the same music listening session;
- to see if there are genres that users associate with a particular time of the day.

These two particular problems can be easily formulated as an association rules problem. The first one is trivial, as we let the set  $\iota$  be composed of all genres in our universe, and each transaction  $T$  be the set of all genres that belong in each session. Each session will correspond to a transaction.

For the second one, we let the set  $\iota$  be composed of all genres in our universe, united with the set of all possible time partitions of a day that we are considering. As each transaction corresponds to a played item in the user’s history, it will contain only two elements: the time partition the played item’s timestamp corresponds to and the genre the played item belongs to.

### 5.1.2.1 Apriori algorithm

The Apriori algorithm is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 [SA97] used in the association rules problem for Boolean association rules. It explores the *Apriori* (monotonicity) property: all nonempty subsets of a frequent itemset are themselves frequent. At the  $k$ th iteration, for  $k \geq 2$ , it forms frequent  $k$ -itemset candidates based on frequent  $(k - 1)$ -itemsets, and scans the database once to find the *complete* set of frequent  $k$ -itemsets,  $L_k$ . The Apriori algorithm solves the complete association rule mining problem, and there are several optimizations (involving transaction reduction and hashing) available that makes this procedure more efficient. Other variations include the partitioning of the data (mining each partition and combining the results) and sampling the data (mining a subset of the dataset)[Goe][HK00].

### 5.1.3 Data mining platforms

#### Weka

Weka (Waikato Environment for Knowledge Analysis)<sup>1</sup> is a suite of machine learning software written in Java. Originating from the University of Waikato, New Zealand, Weka is free software available under the GNU General Public License. Weka supports several data mining tasks, namely, data preprocessing, clustering, classification, regression, visualization and feature selection. Weka provides a GUI but its implemented algorithms can also be called from a command line. For Weka to process any kind of data, it must first be converted to the ARFF (attribute-relation) file format.

#### Rapidminer

Rapidminer<sup>2</sup>, formerly known as YALE (Yet Another Learning Environment) is a suite for machine learning, text mining, data mining, predictive analytics and business analytics. It is an open source application distributed under the AGPL license. Rapidminer is written in Java and a GUI mediates the creation of data mining experiments between the user and their internal XML representation. This representation can be used to launch the platform and run the experiments without the use of the GUI. It allows (via plug-ins) the integration of all the algorithms available in Weka and allows the execution of large scale experiments via a scripting language. Rapidminer can process data from various sources, including files in the ARFF format.

### 5.1.4 Generation of the test files

The Data Integration step of the KDD aims to merge data from various sources into a single dataset. A key part of this project is to be able to link two separate databases together, as we need to map an artist name to the music genres that describe it 3.1.3. Unfortunately MySQL does not support cross-database queries, so this had to be done one layer above, at application level. The `TagPool` script is able to query the artist database to retrieve the tags associated with the given artist. It uniformizes all tags to remove ambiguity and discards all tags that are below a given threshold, as described in 3.2.

When queried about an artist that is not present in the database, it fetches and saves the data using the Last.fm API. It also caches all results in memory so that they can be retrieved faster on subsequent queries. This script was used in the integration of the users and the artist's database, to produce a new database containing a table `user_genres` with `<username, timestamp, genre>` tuples.

<sup>1</sup><http://www.cs.waikato.ac.nz/ml/weka/>

<sup>2</sup><http://rapid-i.com/content/view/181/190/>

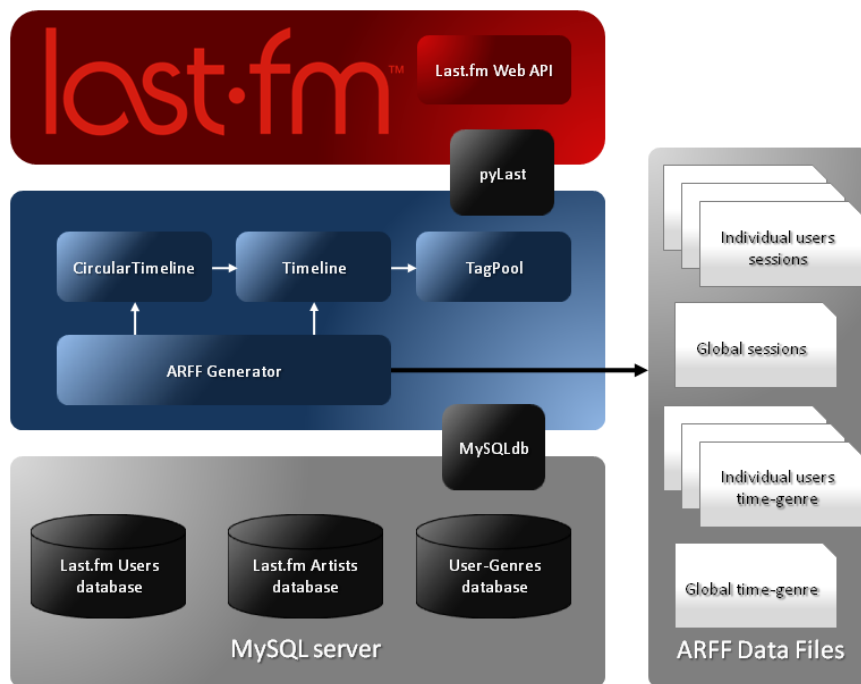


Figure 5.3: Overview of the system that allows the generation of the data files

This figure illustrates the interaction between the modules created for the purpose of transforming and generating the data, the Last.fm Web API and the MySQL database. Data is retrieved from the *users* database by the *Timeline* script. This script uses the *TagPool* script, whose chief function is to associate a set of genres with an artist. This is done using an artists' database, a file which contains the universe of genres, and in the event that an artist is not present in the database, retrieves fresh data from Last.fm.

The aspect of circularity in a timeline is handled by the *CircularTimeline* script, which uses the data structures created by the *Timeline* script to group them into *buckets* of circular nature, like the hours of the day or days of the week. These scripts facilitated the generation of the ARFF data files that we'll use in our analysis.

This system also filtered out the *junk* artists<sup>3</sup> from our database, as these won't be matched to any genre by the *TagPool* script. A more in-depth description of all the methods developed for each of these scripts is available in .

<sup>3</sup>Artists that appear in the Last.fm pages that are caused by a user *scrobbling* a track with a mis-formed artist name tag.

## 5.2 Analysis of music listening sessions

### 5.2.1 Analysis of individual users

With this analysis, we want to find out how many users exhibit these session patterns in their profiles. We have generated one individual data file for each one of the 10,000 users in our database and processed each one of them. Weka was configured to generate rules that have a minimum confidence of 30% and to generate, at most, 100 rules. The chart below summarizes the results that we have encountered.

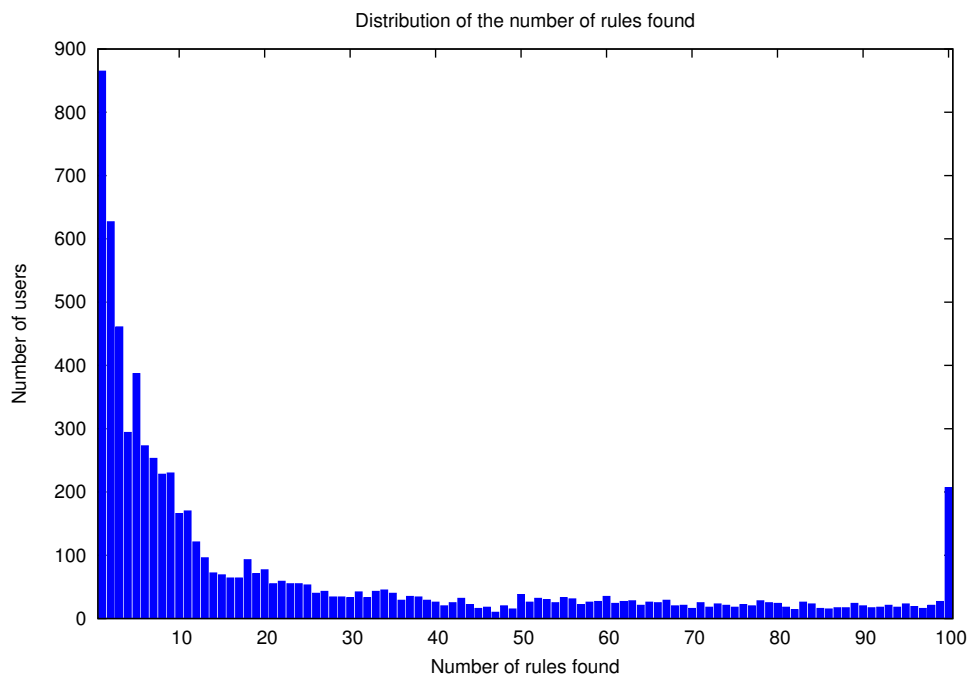


Figure 5.4: Total rules found distribution per number of users

First it's interesting to notice that for nearly 9% of the users in our dataset no rules were found. As we are considering rules with a relatively low confidence we believe that this means that these users do not mix the same genres often enough in the same session.

It's interesting to notice that for 2% of our users, we hit the maximum number of rules. A very high number of rules was obtained for a very large percentage of the population, as one can observe in 5.4. We looked into several of these results and found that these rules have very high confidence values and are made of permutations of the same genres. We believe that these users listen to shuffled playlists very often, and are not showing any preference for a particular set of genres when listening to others.

## 5.2.2 Analysis of the whole dataset

With this analysis, we want to discover which genres users prefer to listen to together in the same music listening session. In order to reduce the very high processing times and distortion in the results that might occur by the presence of genres that are less common in our dataset, we have restricted our analysis to the top 30 genres in our dataset<sup>4.14</sup>. We have only considered sessions with length greater than thirty minutes and we have also discarded sessions that are composed of a single genre. An implication of this is that sessions which are composed exclusively of played items of the same artist (which will occur, for example, when listening to a record) will not be a part of this analysis. The data file was partitioned into two, one for analysis with 70% of the data, and the other for verification, with 30%. The split was done by randomly selecting sessions, not sequentially.

- If a user listens to a song of the *indie* genre, he will listen to a song of the *rock* genre in the same session. (76% confidence)
- If a user listens to a song of the *progressive rock* genre, he will listen to a song of the *rock* genre in the same session. (69% confidence)
- If a user listens to a song of the *hard rock* genre, he will listen to a song of the *classic rock* genre in the same session. (76% confidence)
- If a user listens to a song of the *grunge* genre, he will listen to a song of the *rock* genre in the same session. (90% confidence)
- If a user listens to a song of the *alternative* genre, he will listen to a song of the *rock* genre in the same session. (76% confidence)
- If a user listens to a song of the *brit pop* genre, he will listen to a song of the *alternative* genre in the same session. (54% confidence)



Figure 5.5: Tags Last.fm shows as being related to Rock artists

It's interesting to notice that the significant rules Apriori has found and that we have validated revolve around sub-genres of Rock music. The genres presented in these are

actually not very distant. For example, Last.fm has The Rolling Stones tagged as *rock*, Queen tagged as *classic rock* and Pink Floyd tagged as *progressive rock*. People's taste varies, but mixing these and similar artists together in a session is a behavior that might be recurrent among many Last.fm users. In fact, figure 5.5 shows that Last.fm considers these tags related among them (in terms of the tagged artists' similarity).

### **5.3 Analysis of played items with respect to time**

Due to the issue with the timestamps Last.fm uses we identified and described in 1.1.1, we chose to analyze a smaller subset of our 10k users dataset, composed of only the users that live in fifteen European countries that belong to a single timezone. That way we can easily change the timestamps to each user's local time. Our dataset was thus reduced from 10,000 users to 4,423 users.

#### **5.3.1 Analysis of individual users**

We applied Weka's implementation of the Apriori algorithm to each data file for the 10,000 users in our database. We configured Weka to output only rules with a minimum confidence of 40%, and to generate, at most, 100 rules. The charts below summarize the distribution of rules for each user, in a logarithmic scale. This test makes no particular assessment to the quality of each rule produced.

## Dividing the day in 3 eight-hour periods

### Distribution of rules

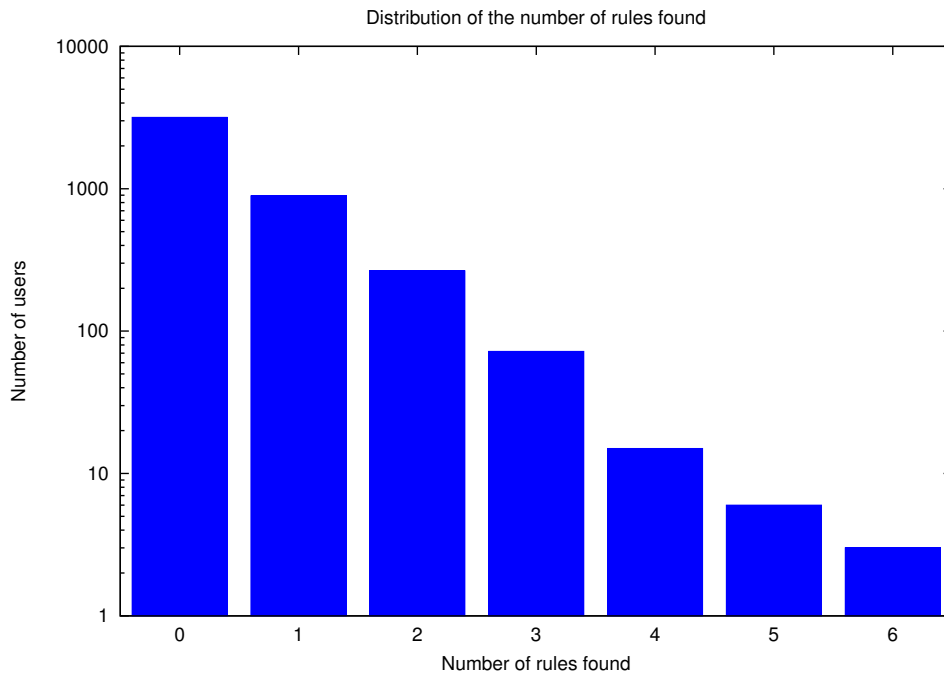


Figure 5.6: Total rules found distribution per number of users

Figure 5.6 shows that for over 67% of the users in our reduced dataset no rules were found. This means that these users exhibit no patterns in terms of a certain preference for a genre in the three time partitions this test considered. However, we have found at least one rule for the rest of the dataset, and a very small number of users (about 2%) have at least three rules indicating preference for a genre in one of these three time partitions.



## Data Analysis

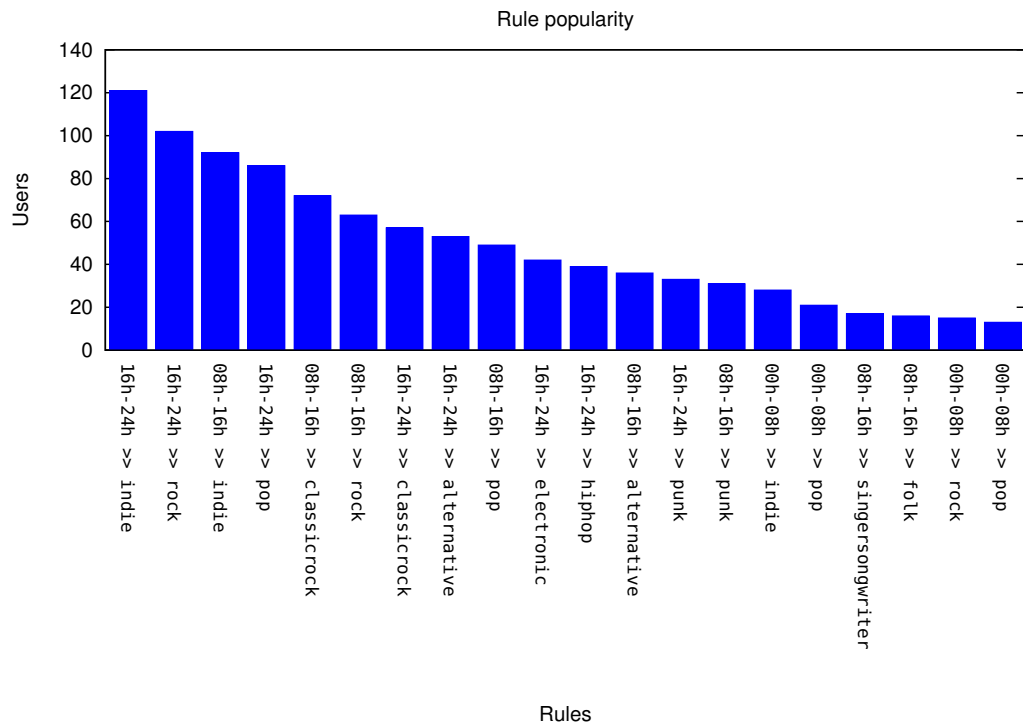


Figure 5.7: 20 most popular rules for this time partition

Figure 5.7 shows the distribution of the 20 most popular rules found in all users for this particular time partition.

## Dividing the day in 8 four-hour periods

### Distribution of rules

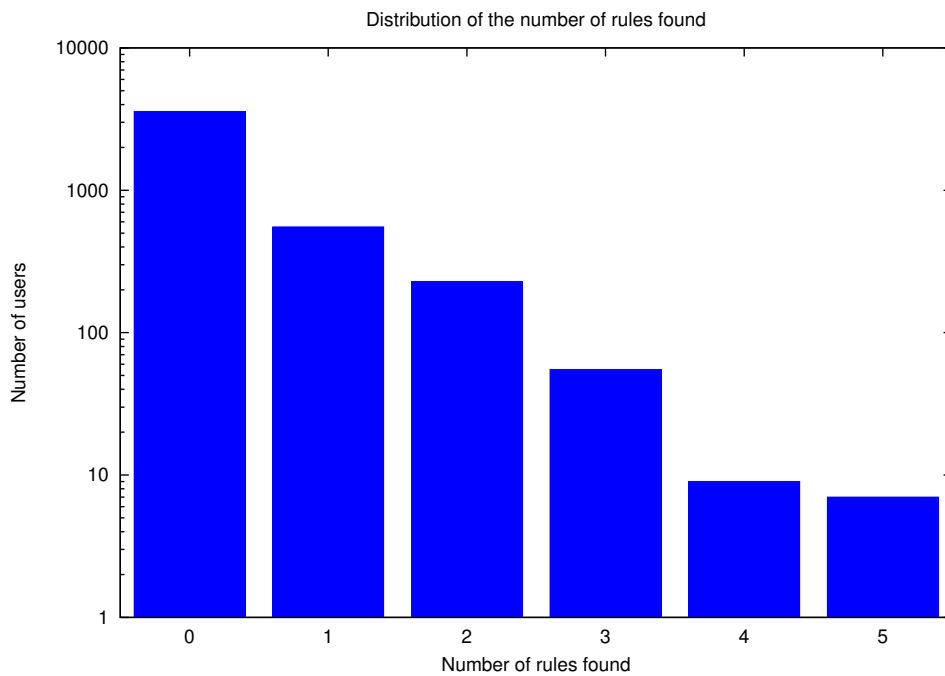


Figure 5.8: Total rules found distribution per number of users

This time the maximum number of rules we encountered was 5, but for less than 1% of our reduced dataset. With more time partitions, the number of users which exhibited no rules rose to 79%. Still, at least one or two rules were found for roughly 20% of the users in our dataset.

## Data Analysis

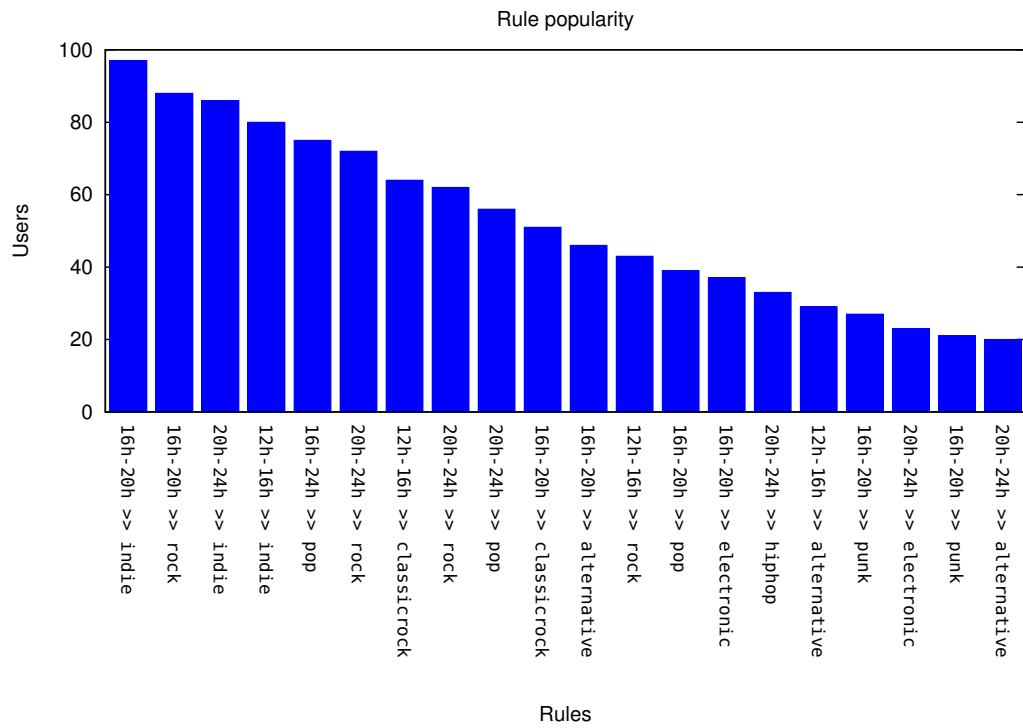


Figure 5.9: 20 most popular rules for this time partition

Figure 5.9 shows the distribution of the 20 most popular rules found in all users for this particular time partition.

## Dividing the day in 24 one-hour periods

### Distribution of rules

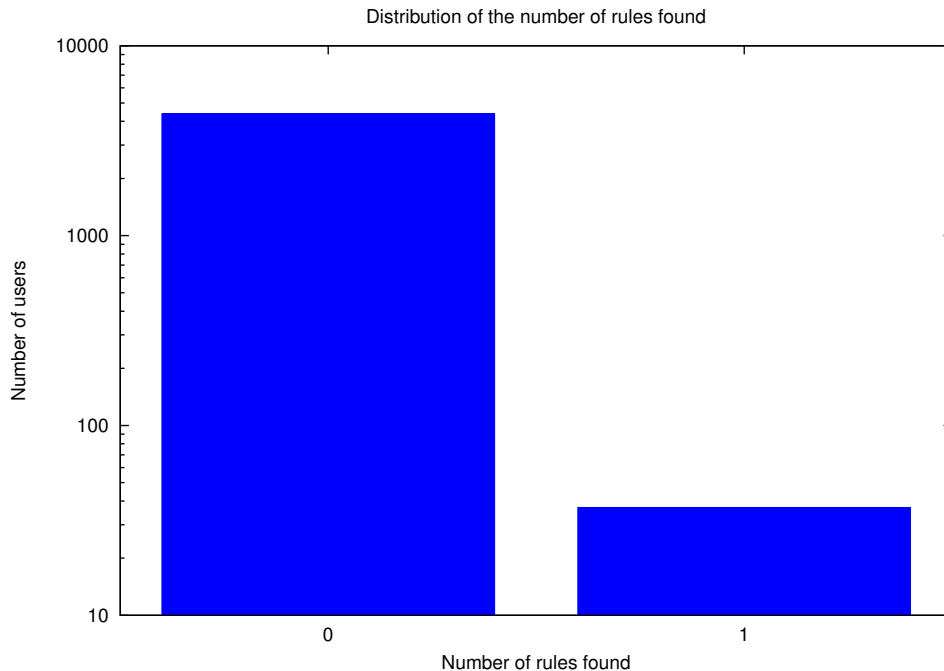


Figure 5.10: Total rules found distribution per number of users

This chart shows that when we consider time partitions for the day as small as an hour, we don't get positive results. This time, less than 0.5% of the users show that they listen to a particular genre often in a given hour of the day.

We believe that while there are users that exhibit genre preference in a particular frame of time, these users are not part of the majority. We are not assessing the quality of each rule individually, but rather taking the number of rules found as an indicator of genre preference over time for particular users.

### 5.3.2 Analysis of the whole dataset

Our methodology was the same as we used in 5.3.1. We divided our dataset in two, one for analysis with 70% of the data, and the other for verification, with 30%. Again, the split was done by random selection. We tried to search for rules for the same partitions of a day.

Unfortunately, our results for this test came empty as we were not able to validate any rules. We believe that the cause for this comes from the fact that the population of users in our dataset is very disperse in terms of the listening preferences and in the hours that they are active in Last.fm listening to music.

With this, our conclusion is that this work would benefit greatly if we worked towards clustering users based on the similarity of their listening preferences and of the hours that they listen to music, and only then should we try to make this kind of analysis on each cluster. We believe this would produce the results we sought to achieve.

### **Summary**

In this chapter we made a shallow revision of the Knowledge Discovery in Databases process and described the system we developed for the generation of the test files. We used the Weka implementation of the Apriori algorithm to run several tests, with the goal to see what genres people associate with certain times of the day and what genres people play together in the same music listening session. While we were able to extract and verify a set of rules for mixing genres in a session, we were unable to do so for the times of the day. We also did an individual analysis of each profile, and showed that a large number of users does in fact show a preference for at least one genre in a specific time of the day.

## Data Analysis

## Chapter 6

# Conclusions

We believe that of all the contributions this work presents, the one that had more success was the development of a crawling application and of a very large dataset containing information about a sample of Last.fm users. This and the data transformation scripts and data file generators we developed will surely prove useful to whoever uses this work as their state of the art.

We also took some interesting conclusions from the results of our tests. Our analysis of music listening sessions is groundbreaking work, as there is no research around this particular problem. We produced a small, yet concise set of rules that show how users like to mix genres in their listening sessions, and showed that a large number of users do show some preference for mixing together genres in the same session.

Our genre-time of day analysis did prove fruitful to some extent as we showed once more, but with a different methodology than what we found in the state of the art, that it exists a non-negligible amount of users that like to associate a particular genre with a particular time of day.

We came to the late conclusion that perhaps the conditions we established for the creation of our research dataset were not the best. Our dataset is composed of the entire listening history of 10,000 Last.fm users. This is perhaps more indicated towards the analysis of individual profiles, something we did but which was not our initial goal. Perhaps aiming to the extraction of a smaller period's worth of played tracks of a much larger number of users would best suit our goals.

While we did select 10,000 users at random from about a million of usernames we discovered, the randomness of the dataset can be criticized for two main reasons:

- The seed usernames we used were all based in Porto, Portugal. Choosing seed usernames from various places in the world would yield us a more varied dataset, in terms of artist choices and genre preferences.

- The username discovery process was performed in a breadth-first fashion. Changing this to a depth-first strategy would perhaps distance us even more from the seed usernames in the Last.fm social graph.

A possible effect of these two reasons can be observed in the country distribution of the dataset in 4.3.1, as there is no reason for Poland being the second most represented country in the dataset, with more users than the United Kingdom, where Last.fm originates from.

### 6.1 Future Work

We did not explore the time component of this problem to its maximum extent. There are more repeating patterns in time we did not analyze, like those in the cyclical nature of weeks and years. Research shows that these, like the days of the year, have an influence on the listener's choice. We leave for future work to apply this methodology to this particular component of the problem.

We came to the conclusion that while there are users that exhibit an influence of time in their music listening choices, we found out that the large majority does not. Perhaps working towards the clustering of the users either in terms of their genre preferences and/or the time that they are active (listening to music) would help us achieve the results we did not find.

We saw some results that did make sense in our session-genre analysis. It was, however, limited to a small number of genres. Again, applying a clustering of the users in terms of genre preference would help us extend this analysis to less popular genres and produce rules specific to certain classes of users.



# References

- [Ahn04] JoongHo Ahn. Collaborative filtering for recommender systems: a scalability perspective. *International Journal of Electronic Business*, 2(1):77, 2004.
- [BAA09] Linas Baltrunas, Xavier Amatriain, and Via Augusta. Towards Time-Dependant Recommendation based on Implicit Feedback. 2009.
- [Cel08] Oscar Celma. *Music Recommendation and Discovery in the Long Tail*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Spain, 2008.
- [dCL09] João Norberto Fernandes da Costa Lima. Towards gathering and mining last.fm user-generated data. Master’s thesis, Faculdade de Engenharia da Universidade do Porto, Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL, 2009.
- [FPsS96] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 17(3):37–54, 1996.
- [Goe] Maimon, Oded Goethals, Bart. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc. Secaucus, NJ, USA ©2005.
- [HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2000.
- [HK07] Martin J. Halvey and Mark T. Keane. An assessment of tag presentation techniques. *Proceedings of the 16th international conference on World Wide Web - WWW '07*, page 1313, 2007.
- [HRS] Perfecto Herrera, Zuriñe Resa, and Mohamed Sordo. Rocking around the clock eight days a week: an exploration of temporal patterns of music listening. *Information Retrieval*, pages 7–10.
- [JCJ04] Steve Jones, Sally Jo Cunningham, and Matt Jones. Organizing digital music for use: an examination of personal music collections. In *ISMIR*, 2004.
- [Jon09] Richard Jones. Last.fm Radion Announcement, March 2009. <http://blog.last.fm/2009/03/24/lastfm-radio-announcement>, last accessed on July 9, 2009.

## REFERENCES

- [Lam08] Paul Lamere. Social Tagging and Music Information Retrieval. *Journal of New Music Research*, 37(2):101–114, June 2008.
- [PK10] Chan Ho Park and Minsuk Kahng. Temporal Dynamics in Music Listening Behavior: A Case Study of Online Music Service. *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*, pages 573–578, August 2010.
- [Res10] Zuriñe Resa. Towards time-aware contextual music recommendation: an exploration of temporal patterns of music listening using Circular Statistics. 2010.
- [SA97] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. *Future Gener. Comput. Syst.*, 13:161–180, November 1997.
- [ZGMMF10] Elena Zheleva, John Guiver, Eduarda Mendes Rodrigues, and Nataša Milić-Frayling. Statistical models of music-listening sessions in social media. *Proceedings of the 19th international conference on World wide web - WWW '10*, page 1019, 2010.

# Appendix A

## Description of the developed scripts

### A.1 Crawler Application

The installable package of the crawler application contains the following files

---

<b>Bash scripts</b>	
<code>batch.sh</code>	Bash script for the execution of a <i>batch</i> crawling
<code>logger.sh</code>	Bash script that redirects a process's output to <i>processpid.log</i>
<code>stopjobs.sh</code>	Bash script that stops all running scripts

---

Table A.1: Bash scripts included in the package

---

<b>Configuration files</b>	
<code>configuration.py</code>	Python script that serves as a configuration file, defining several run-time variables

---

Table A.2: Configuration files included in the package

---

<b>Runnable scripts</b>	
<code>preparation.py</code>	Preparation and maintenance script
<code>status.py</code>	Presents a summary of the crawling process
<code>crawl_usernames.py</code>	Crawls Last.fm user names
<code>crawl_userprofiles.py</code>	Crawls Last.fm user profiles
<code>crawl_usertracks.py</code>	Crawls Last.fm user listening profiles
<code>update_usertracks.py</code>	Updates Last.fm user listening profiles

---

Table A.3: Runnable scripts included in the package

## Description of the developed scripts

---

<b>Libraries</b>	
<code>pylast.py</code>	PyLast Library
<code>connection.py</code>	Connection module. Supports connection to MySQL and to Last.fm
<code>timecontroller.py</code>	API request limit compliance module
<code>util.py</code>	Functions shared between scripts

---

Table A.4: Library scripts included in the package

## A.2 Data Transformation scripts

### A.2.1 TagPool script

The TagPool script is responsible for helping in the integration of the Users and the Artists database. When the genre of an artist that is not present in the database is requested, it fetches fresh data from Last.fm and stores it in the database.

---

<b>TagPool</b>
<code>TagPool(threshold)</code> Initializes an instance of TagPool with the given threshold as the default for validating tags
<code>get_genre(artist)</code> Returns a list of tags that were determined by the script to describe the given artist's genre facet

---

Table A.5: Summary of the TagPool script

### A.2.2 Timeline script

The Timeline script is responsible for retrieval of the data from the Users database.

## Description of the developed scripts

<b>Timeline</b>	
<code>get_listening_data(username)</code>	Returns a python list of python tuples containing each played item for the given username
<code>get_time_to_playeditem(username)</code>	Returns a python dictionary (map) that associates each timestamp with each item played by the given user
<code>get_time_to_playeditem_filtered_by_genre(username, genreset)</code>	Returns a python dictionary (map) that associates each timestamp with each item played by the given user, with the option to include only items that are associated with the genres in the given tag set
<code>get_time_to_playeditem_filtered_by_artists(username, artistset)</code>	Returns a python dictionary (map) that associates each timestamp with each item played by the given user, with the option to include only items that are associated with the artists in the given artist set
<code>get_time_to_genre(username)</code>	Returns a python dictionary (map) that associates each timestamp with the tag (genre) of each item played by the given user
<code>get_genre_to_time(username)</code>	Returns a python dictionary (map) that associates each tag (genre) in the given user's profile with the set of timestamps where they occurred
<code>get_session_data(username)</code>	Returns a python list of python tuples that contain the beginning and ending timestamps and the played items that fall in each music listening session in the given user's profile
<code>get_session_genre_data(username)</code>	Returns a python list of python tuples that contain the beginning and ending timestamps and the genres associated with the played items that fall in each music listening session in the given user's profile
<code>get_session_lengths(username)</code>	Returns a list of session lengths for each existing session in the given user's profile
<code>get_all_artists(username)</code>	Returns a list of all the artists in the given user's profile
<code>get_all_genres(username)</code>	Returns a list of all the genres in the user's profile

Table A.6: Summary of the methods available in the *timeline* script

### A.2.3 CircularTimeline script

The CircularTimeline script is responsible for retrieval of the data from the Users database and its transformation into a timeline of circular nature.

## Description of the developed scripts

---

<b>CircularTimeline</b>
<code>get_months_to_playeditems(username)</code> Returns a python dictionary (map) between months and items played by the given user
<code>get_months_to_playeditems_filtered_by_genres(username, genreset)</code> Returns a python dictionary (map) between months and items played by the given user, filtering out played items which are not associated with the genres in the given genre set
<code>get_months_to_playeditems_filtered_by_artists(username, artistset)</code> Returns a python dictionary (map) between months and items played by the given user, filtering out items which are not associated with the artists in the given artist set
<code>get_weekdays_to_playeditems(username)</code> Returns a python dictionary (map) between weekdays and items played by the given user
<code>get_weekdays_to_playeditems_filtered_by_genres(username, genreset)</code> Returns a python dictionary (map) between weekdays and items played by the given user, filtering out items which are not associated with the genres in the given tag set
<code>get_weekdays_to_playeditems_filtered_by_artists(username, artistsset)</code> Returns a python dictionary (map) between weekdays and items played by the given user, filtering out items which are not associated with the artists in the given artist set
<code>get_dayhours_to_playeditems(username)</code> Returns a python dictionary (map) between day hours and items played by the given user
<code>get_dayhours_to_playeditems_filtered_by_enres(username, genreset)</code> Returns a python dictionary (map) between day hours and items played by the given user, filtering out items which are not associated with the genreset in the given tag set
<code>get_dayhours_to_playeditems_filtered_by_artists(username, artistset)</code> Returns a python dictionary (map) between day hours and items played by the given user, filtering out items which are not associated with the artists in the given artist set
<code>get_dayhours_to_genres(username)</code> Returns a python dictionary (map) between day hours and the genres (genres) associated with the items played by the given user that fall on each day hour

---

Table A.7: Summary of the class CircularTimeline

### A.3 Chart generation scripts

This section describes the scripts that were developed in order to generate charts and plots that allow us to visualize the data for a given user.

## Description of the developed scripts

---

<b>Plotting scripts</b>	
<code>plot_days.py</code>	Creates a circular plot that shows how much music has been listened to in each of the 24 hours of the day. Created using <code>python-matplotlib</code> .
<code>plot_weekdays.py</code>	Creates a plot that shows how much music has been listened to in each of the 7 days of the week. Created using <code>python-matplotlib</code>
<code>plot_months.py</code>	Creates a plot that shows how much music has been listened to in each of the 12 months of the year. Created using <code>python-matplotlib</code>
<code>plot_timeline.py</code>	Creates a plot that represents graphically the user's profile timeline. Created using <code>gnuplot</code>
<code>plot_genre_timeline.py</code>	Creates a plot that represents graphically a particular genre's timeline. Created using <code>gnuplot</code>

---

Table A.8: Plotting scripts included in the package