FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Towards Gathering and Mining Last.fm User-Generated Data

João Norberto Fernandes da Costa Lima

Report of Dissertation

Master in Informatics and Computing Engineering

Supervisor: Rui Camacho (Professor Associado)

Second Supervisor: Fabien Gouyon (Professor)

28^{th} July, 2009

Towards Gathering and Mining Last.fm User-Generated Data

João Norberto Fernandes da Costa Lima

Report of Dissertation

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Luís Paulo Reis (Professor Auxiliar)

External Examiner: Miguel Rocha (Professor Auxiliar)

Internal Examiner: Rui Camacho (Professor Associado)

31st July, 2009

Abstract

Online musical social communities are becoming more and more popular as time goes by. Nowadays, these social communities give folksonomy systems to their users, allowing them to play an active role by dictating musical tastes, popularity of artists and hit songs like it never happened before. Since it is the people, and especially the fans, that really determine how famous an artist can be or how a song can impact their musical tastes, these Web social communities are the ultimate form of capturing the perspective of the fans. Although the widespread musical communities give us more information about music, there is still too much to learn about it. We are trying to bring results to this information and find relations, patterns and knowledge that influence the music culture. Our studies focused on the popular music social community Last.fm, where we have access to millions of entries concerning music and artists. Our goal in this project is to collect Last.fm musical data and learn interesting patterns and knowledge about music from the past choices of fans, being the prediction of the popularity of artists our most ambitious goal.

In order to fulfill our goals we proposed the development of a Last.fm data gathering system, that had a crawling component (responsible for retrieving artists and their musical data from Last.fm) and a data mining component (responsible for processing the previously gathered Last.fm data into visual graphs and datasets for data mining tools).

The development of this system was based on the Knowledge Discovery in Data and the Cross Industry Standard Process for Data Mining major steps, which provided guidelines for the data retrieval from Last.fm and the processing of that same data in order to be analyzed by data mining algorithms. The data mining algorithms applied were the association apriori and the clustering k-means, and we discovered interesting musical association rules and grouping of music data into clusters, respectively. With more temporal snapshots of databases we could analyze data in order to find more interesting patterns and knowledge than the ones we found during this project.

We can say that the development of this system was a success as it allowed us to have a better understanding of Last.fm and gave us the necessary means to extract more musical data from Last.fm, for better understanding musical information and musical popularity measure in the future.

Resumo

As comunidades sociais de musica online estão a tornar-se cada vez mais populares à medida que o tempo passa. Hoje em dia, estas comunidades sociais oferecem sistemas de folksonomy para os seus utilizadores, permitindo que estes desempenhem um papel activo na imposição de gostos musicais, popularidade de artistas e músicas de sucesso como nunca antes tinha acontecido. Visto que são as pessoas, especialmente os fãs, que verdadeiramente determinam o quão famoso um artista pode ser ou qual o impacto que uma canção tem nos seus gostos musicais, estas comunidades sociais na Web são a derradeira forma de captar a perspectiva dos fãs.

Apesar das muito difundidas comunidades musicais nos darem mais informação sobre música, ainda há muito para aprender sobre ela. Estamos a tentar trazer resultados para esta informação e encontrar relações, padrões e conhecimento que influenciem a cultura musical. Os nossos estudos incidiram sobre a comunidade social de música do Last.fm, onde temos acesso a milhões de entradas sobre música e artistas. O nosso objectivo neste projecto é a recolha de dados musicais do Last.fm e a aprendizagem de padrões e conhecimento interessante sobre música através das escolhas antigas dos fãs, sendo o nosso objectivo mais ambicioso a previsão da popularidade de artistas.

De forma a cumprir os nossos objectivos, propusemos o desenvolvimento de um sistema de recolha de dados do Last.fm, que possuísse uma componente de crawling (responsável pela recolha de artistas e os seus dados musicais da Last.fm) e uma componente de data mining (responsável pelo processamento dos dados anteriormente recolhidos do Last.fm em gráficos e conjuntos de dados para as ferramentas de data mining).

O desenvolvimento deste sistema foi baseado nos passos essenciais de Knowledge Discovery Data mining e Cross Industry Standard Process for Data Mining, que forneceram linhas orientadoras para a extracção de dados do Last.fm e o processamento desses dados para serem analisados por algoritmos de data mining. Os algoritmos de data mining aplicados foram o de associação apriori e o de clustering k-means, e descobrimos regras de associação musical interessantes e o agrupamento de dados musicais em clusters, respectivamente. Com mais bases de dados temporais poderemos analisar os dados de forma a descobrir mais padrões interessantes e conhecimento do que aqueles que foram encontrados durante este projecto.

Podemos dizer que o desenvolvimento deste sistema foi um sucesso, visto que nos permitiu uma melhor compreensão dos dados do Last.fm e nos deu os meios necessários para extrair mais dados musicais do Last.fm, para melhor compreender a informação musical e medidas de popularidade musical no futuro.

Acknowledgements

I would like to thank my FEUP supervisor, Prof. Rui Camacho, and my INESC supervisors, Prof. Fabien Gouyon and Eng. Luís Sarmento, who have always challenged me to do a better job, for all the support given to me and to the project.

I would also like to thank Cláudio Costa, Hector Dantas, Hugo Gomes, Luís Barbosa, Miguel Lima and Tiago Teixeira for their support and their ideas that helped me do a better job at this project.

Finally, I would like to thank Talita, Raquel and the rest of my family for the support and the encouragement they gave me, especially in the final steps of this project.

João Norberto Fernandes da Costa Lima

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Problem definition . . . 5
- 1.3 Motivation and Goals . . . 6
- 1.4 Thesis Structure . . . 7

2 Analyzing the Last.fm data **8**
- 2.1 Similar studies and applications . . . 8
- 2.2 Knowledge Discovery in Databases . . . 9
  - 2.2.1 Overview . . . 9
  - 2.2.2 The Importance of KDD . . . 14
- 2.3 Conclusions . . . 18

3 Specifications **20**
- 3.1 Last.fm crawling component . . . 20
- 3.2 Last.fm data mining component . . . 22
- 3.3 General Implications . . . 22
- 3.4 System Architecture . . . 23

4 Implementation **27**
- 4.1 Technologies and Development Environment . . . 27
  - 4.1.1 Perl . . . 27
  - 4.1.2 MySQL . . . 28
  - 4.1.3 Eclipse IDE . . . 28
  - 4.1.4 Personal Review of technologies . . . 29
- 4.2 Implementation details . . . 29
  - 4.2.1 Last.fm API methods . . . 29
  - 4.2.2 Database creation . . . 33
  - 4.2.3 Crawler system . . . 34
  - 4.2.4 Artist Filter . . . 37
- 4.3 Conclusion . . . 40

5 Prospective Analysis of Data **41**
- 5.1 Methodology . . . 41
- 5.2 Visual Analysis . . . 42
  - 5.2.1 Artist Universe . . . 42
  - 5.2.2 Artist Graphs . . . 44

5.2.3 Popularity Graphs . . . 51
5.2.4 Tags Graphs . . . 55
5.3 Data Mining based prospective analysis . . . 56
5.4 Clustering tests . . . 57
5.4.1 Test 1: Grouping artists by tags . . . 57
5.4.2 Test 2: Artist evolution . . . 58
5.5 Association tests . . . 59
5.5.1 Test 3.1: Occurence of tags . . . 59
5.5.2 Test 3.2: Occurence of tags . . . 60
5.5.3 Test 4.1: Occurence of similar tags . . . 62
5.5.4 Test 4.2: Occurence of similar tags . . . 63
5.5.5 Test 5: Global relations with similar artists . . . 64

6 Conclusions **66**

7 Future work **68**

References **70**

A List of artists from the study of Musicbrainz relevance **71**
A.1 Japanese Artists . . . 71
A.2 Chinese Artists . . . 71
A.3 Russian Artists . . . 72
A.4 Portuguese Artists . . . 72
A.5 Indian Artists . . . 73
A.6 DJ Artists . . . 73

B Database Statistics **74**

List of Figures

1.1 Last.fm logo . . . 2
1.2 Context Diagram . . . 6

2.1 Process of the KDD . . . 11
2.2 CRISP-DM data mining project lifecycle . . . 12
2.3 Basic Architecture of the KDD process . . . 13

3.1 System Architecture . . . 24

4.1 *artist.getInfo* method attributes details . . . 30
4.2 *artist.getShouts* method attributes details . . . 31
4.3 *artist.getSimilar* method attributes details . . . 31
4.4 *artist.getTopTags* method attributes details . . . 32
4.5 *artist.getTopAlbums* method attributes details . . . 33
4.6 System Architecture with Artist Filter module . . . 39

5.1 Overview of the Artist Universe . . . 43
5.2 Histogram of Number of characters in artists names of the global universe 45
5.3 Histogram of Number of characters in artists names of the MusicBrainz universe . . . 46
5.4 Histogram of Number of artists by number of albums . . . 47
5.5 Histogram of Number of artists by number of tags . . . 48
5.6 Histogram of Number of artists by number of listeners . . . 49
5.7 Histogram of Number of artists by total playcount . . . 50
5.8 Artists by Listeners/Playcount . . . 52
5.9 Histogram of Artists by Listeners/Tags . . . 53
5.10 Histogram of Artists by Listeners/Albums . . . 54
5.11 Histogram of Top Tags . . . 55
5.12 Number of artists by tags . . . 56

List of Tables

B.1 Databases Crawl Statistics 74

Abbreviations

API	Application Programming Interface
CRISP-DM	Cross Industry Standard Process for Data Mining
GUI	Graphical User Interface
IDE	Integrated Development Environment
KDD	Knowledge Discovery in Databases
MIR	Music Information Retrieval
URL	Uniform Resource Locator
WEKA	Waikato Environment for Knowledge Analysis
XML	eXtensible Markup Language

Chapter 1

Introduction

1.1 Context

This project is inserted in the Multimedia and Telecommunications Unit of INESC Porto (private non-profit association for scientific research and technological development, based in Porto, Portugal).

This project is mainly inserted in two areas that interact with each other:

- Music Information Retrieval;
- Knowledge Discovery in Databases.

According to [Fin04], the Music Information Retrieval area is an area that studies music data by bringing together different disciplines and paradigms regarding music, music repositories and information retrieval.

The KDD data analysis approach transforms raw data from data repositories into valuable and interesting information that can help business, investigation and other sciences.

Altough the Music Information Retrieval embraces multiple areas, we will only focus on the development of information retrieval software, and the discovery of music knowledge using data mining programs and algorithms.

We will focus our study in the musical data repository of the Last.fm website[1].

Last.fm

Created in 2002, Last.fm is an Internet radio and music community website, where users are able to listen to music on demand or download it, and to create personal and customized playlists and radio stations from any song available in the music library of

[1] www.lastfm.com

Last.fm [tfeb]. The site offers numerous social networking features and can recommend and play artists similar to the users' favourites. This online community exists thanks to a music recommender system called "Audioscrobbler", which will be presented later.

Figure 1.1: Last.fm logo

Nowadays there are millions of songs in Last.fm's music library from artists from all over the world, with a huge variety of musical genres, and an extensive repertoire of songs of the most famous artists in the world, which is updated on a regular basis. This music library is constantly updated by the Last.fm staff, and by them only, so users don't have the rights to upload or download copyrighted songs, and therefore there is no piracy and other illegal actions concerning music in Last.fm website. Although Last.fm has an extensive amount of popular artists' songs, this website also encourages, independent and unknown artists to spread their musical work, in order to become more and more popular when users already listen to similar artists.

Audioscrobbler System

As mentioned before, Audioscrobbler is a music recommender system that produces detailed information of each user's musical taste by recording details of all the songs the user listens to via radio stations, personal computers and portable music devices. Then, all the details are "scrobbled" and stored in Last.fm databases via a plugin installed into the users' music players [tfeb]. This is an effective system to collect artist data from all over the world. We now present a list of features, according to [tfeb] that make the Last.fm website what it is today: an important music web resource and a community of users and artists.

Artist profile pages

When one user ”scrobbles” a song of a given artist, Last.fm automatically creates a profile page for that artist, which displays detailed information and statistics of similar artists, listeners, tags, albums, songs, and fans; it provides a shoutbox so that users can leave messages for the artists; it also includes a player, where users can listen to small portions of the artists' songs. Last.fm gives users the power to edit and update the information of an artist in its profile page, with a moderation to avoid misleading information about all the artists present in Last.fm.

Tag system

Last.fm has a tag system that enables users to label artists, albums, and songs using keywords that define them, in order to create a folksonomy (which is: *”the result of personal free tagging of information and objects (anything with a URL) for one's own retrieval. The tagging is done in a social environment (shared and open to others). The act of tagging is done by the person consuming the information”*) of music. This tag system gives users the freedom to create all sorts of defining keywords for all the elements of Last.fm, which can be any form of music genre, any defining characteristic of the artists, or any other form of music and artists classification. The most important benefit of this system is that it allows users to browse and listen to music and artists via tags in order to discover new artists similar to the ones they listen to and praise, which in the same way also allows unknown artists to be known and discovered by the same users, and in this way raise their popularity.

However, since tagging is not moderated, the manipulation by the site's users most often results in disagreements about music genres, inaccuracies in artist classification, and input of junk data.

Last.fm radio

Last.fm offers free radio stations consisting of songs selected from the music files in the Last.fm music library. This selection can be based on the users' personal music tastes and profiles, or be based on the group of friends that a user has. Last.fm also offers the tag radio option, which consists of a radio with a huge compilation of songs that share the same tag. Just like an actual radio station, Last.fm doesn't allow users to interact in the selection of the music that is played. Nowadays, the free radio stations are available for a trial period only to users outside the United Kingdom, Germany and the United States of America, that once expired requires for the users to pay a monthly fee, so that they can continue listening to those radio stations.

Audioscrobbler plugin

Last.fm offers the possibility of downloading the Audioscrobbler plugin, which is a user interface for the previously mentioned Audioscrobbler system, and can create an artist profile directly from the music played on the computer of a user. As long as the song is longer than thirty seconds and has music and song metadata, the song is considered valid for submission.

Chart system

Last.fm has a charting system that automatically generates and archives charts of top artists and songs listened to by all Last.fm users by using the information of the Audioscrobbler plugin or the Last.fm radios. This generation is generally based on the information compiled from the metadata from audio files ”scrobbled” from the users' computers. However, this information can have been incorrectly typed by users, therefore causing inaccuracies in the listings. The rest of the charts are generated by using the artists' listeners' metric, which consists of the number of users who play a certain artist or track rather than the total number of times that a certain artist or track are played. This prevents users from artificially boosting the popularity of an artist by listening to that artist's songs several times. This also prevents users from raising their own fan ranking in Last.fm. For all that was mentioned above, Last.fm musical charts are different from any other commercial music charts, which are based in radio plays and sales. The information in Last.fm is less volatile, because an artist that is considered a music legend (even if it is not recent) can be in the Last.fm charts for many months or years. Last.fm top charts reflect not only the information about artists and their music, but also the tags that have been inserted.

Social interaction

Last.fm allows users to interact with each other by giving the possibility of meeting people and adding friends via social networking features. One interesting aspect of this interaction is that Last.fm displays information about the musical compatibility between users, which is an important measurement when meeting new people and getting to know new musical tastes by listening to the friends' praised artists. Another social feature in Last.fm is one that allows users to leave messages or shouts in the artist profile pages. Last.fm also provides information about artist concerts or live acts that will happen in the geographical vicinity of the users, or the concerts or live acts which their friends or other users will attend to, creating a whole new perspective in attending to musical events.

Last.fm API

For development purposes of applications or plugins, Last.fm provides an API which allows developers to call methods that respond in REST style XML, which enable a much easier access to all Last.fm data. The API method calls range from artist information to artist music, tags, fans, albums and shouts. Last.fm encourages users to use its API and its applications in order to maximize and improve the Last.fm experience.

1.2 Problem definition

Nowadays - with the wide proliferation of the music industry over the Internet, the intensive use of MP3 content and the Apple's Ipods [New09] - more and more people listen to music from all over the world, and can ditacte musical tastes, popularity of artists and hit songs like it never happened before. The explosion of musical social networks, like Last.fm[2] and Myspace Music[3], and its intensive use from a wide community all over the world have provided information to the music industry about musical preferences and tastes. The folksonomy systems, provided by the musical social networks, are responsible for the generation and management of this musical information, due to the collaborative creation of tags that identify and classify artists, music and other musical data. The use of tags is useful for the creation and classification of music and artists, but also for simplifying the search of that same music and artists.

Although the widespread musical communities give us more and more information about music, there is still too much to learn about it, such as artist popularity and the hit song science. We are trying to bring results to these areas, and find relations, patterns and knowledge that influence the music culture. But before we can discover interesting knowledge about music, we need to have access to a valid and complete musical repository just like Last.fm, and the means to access it.

The need to analyze a musical repository in order to study interesting knowledge about music led to the creation of this project.

Based on the definition of the MIR area (which is: "*an emerging research area that focuses on the content-based retrieval of musical documents against musical queries, where both documents and queries may be in acoustic or notated form*" [otUoP]), we proposed the creation of an infrastructure for musical information retrieval, in order to acquire the data needed for our studies. We chose Last.fm to be the information repository from which we would extract musical information, because of the very large community around the world that it embraces, and its extensive database with millions of entries regarding music and artists.

Figure 1.2 shows a basic diagram of our system.

[2] www.lastfm.com
[3] music.myspace.com

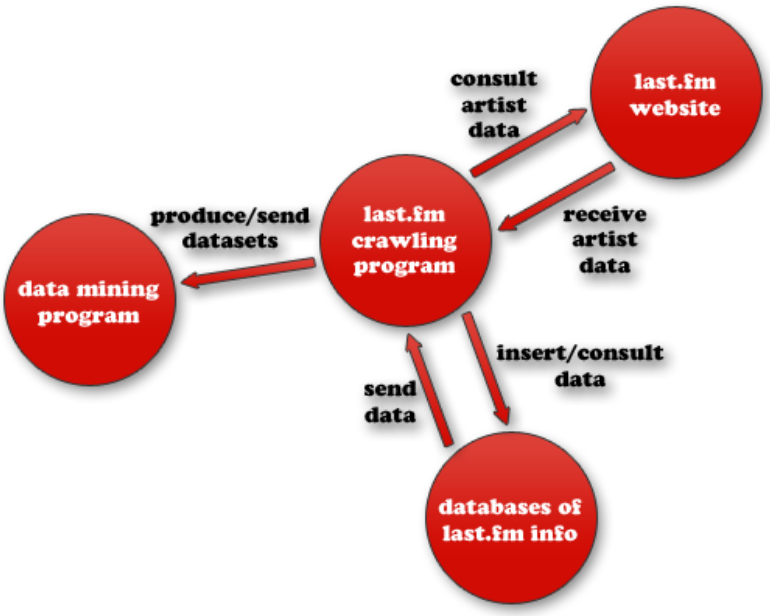

Figure 1.2: Context Diagram

As we can see, our proposal is to create a system that retrieves information from the Last.fm website, stores it, and makes it available to futher processing by data mining tools.

Although we considered Last.fm to be a good choice for the study of temporal tendencies of artists, songs, albums and tags, we were fully aware that it would be difficult to filter and select the most relevant data, due to the freedom a user has in inserting tags and the possibility that a user might ”scrobble” audio files that have incorrect music metadata. Another obstacle we would face would be the need to gather millions of artists (without knowing previously how many millions there would be) and their data in short intervals of time (for example, once a month or once every fortnight).

1.3 Motivation and Goals

The recent growth of online social music communities with folksonomy systems led to the empowerment of their users. In these communities they can actually take an active roll in the classification of music and artists. Since it is the people, and especially the fans, that really determine how famous an artist can be or how a song can impact their musical tastes, these Web social communities are the ultimate form of capturing the perspective of the fans.

That said, our main motivation was to identify interesting patterns and knowledge about music from the past choices of fans. We can also state that our most ambitious

motivation was the prediction of the popularity of artists, based on past information about similar artists.

To fulfill our intentions, the following main goals of this project were defined:

- create a music data gathering system that interacts with the Last.fm data repositories;
- apply visual analyses to the Last.fm data;
- apply data mining algorithms in order to discover musical knowledge from the Last.fm data.

1.4 Thesis Structure

This document is composed by the following six chapters.

In Chapter one, called ”Introduction”, we present the context where this project is inserted, the motivation that drove us to the creation of this project, and what we have done in order to fulfill the project objectives.

In Chapter two, called ”Analyzing the Last.fm data”, we present similar studies that were done in this area, an overview of KDD, the study of different data mining problems and algorithms, and the tools for fulfilling the data mining tasks.

In Chapter three, called ”Specifications”, we present the main features of our Last.fm data gathering system that need to be implemented in order to fulfill our goals.

In Chapter four, called ”Implementation”, based on the previous specification, we explain how the development process took place, and how the important modules of this system were implemented.

In Chapter five, called ”Prospective Analysis of Data”, we present the analysis of the retrieved Last.fm data during this project, the association and clustering algorithms applied over data and the interesting patterns and knowledge found about Last.fm data.

In Chapter six, called ”Conclusions”, the conclusions and a critical analysis are made regarding the developed work.

Finally, in Chapter seven, called ”Future work”, some notes are given about the future work that needs to be done, in order to continue the purpose of this project.

Chapter 2

Analyzing the Last.fm data

In this chapter we present similar studies and applications to our project, by giving examples of studies that used Last.fm, music information retrieval and data mining techniques. Then, we present an overview of KDD and the typical KDD architecture, which gave us guidelines for building our own infrastructure for data gathering. Then, we focus on a very important step of the KDD process - the data mining step - where we present the importance of that step, the different types of data mining problems, and algorithms that solve those problems. Then, we refer some tools for data mining which will be responsible for the data mining tasks over the data collected with our system.

2.1 Similar studies and applications

The following studies address problems similar to the ones we are tackling and provide approaches that could help us in the course of this project.

Hit Song Science

[PR08] mentions the problem of determining music title popularity by processing two sets of music title attributes. In this paper it is claimed that with well-known state of the art data mining algorithms it is not possible to determine if a music title will be a hit or not only by analyzing its audio features.

Artist Classification with Web-based Data

[KPW04] mentions the problem of classifying artists into musical genres by retrieving music information from the Internet. In this paper an approach of researching musical

information from the top 50 webpages retrieved from Google[1] or Yahoo![2] is used, in order to proceed to the artist classification into musical genres.

Automatic Detection of members and Instruments

[SWPS07] mentions the problem of automatically detect band members and instrumentation by retrieving music information from the Internet. In this paper an approach of researching musical information from the top 100 webpages retrieved from Google is used, in order to perform linguistic analysis to determine the actual role of an artist in a band.

Truth about tags

[GSK07] mentions the problem of verifying the consistency of tagging in the artist similarities, by analyzing tag musical data retrieved from the Last.fm website. In this paper it is claimed that the act of tagging for classification of artists and music is consistent for artist similarities, and that tags are descriptive enough for artist classification.

Automatically Generated Music Information System

[SKPW08] mentions the purpose of creating a music information system just like Last.fm, but instead of using the interaction of people to fill the data repositories, it uses an automatic method to retrieve the music information. The Google search engine is used in order to retrieve the most relevant data about music and artists.

2.2 Knowledge Discovery in Databases

2.2.1 Overview

According to [HK00], Knowledge Discovery in Databases is the software task responsible for the extraction of knowledge from large amounts of data stored in databases, data warehouses, or other information repositories. The term ”data mining” derives from the prehistoric activity of mining minerals, and the extraction of a small quantity of precious materials from large quantities of raw materials, which in this case is the extraction of knowledge from a large amount of data. Although the extraction of knowledge is the basis of the definition of data mining, data mining is only one step of the overall process of knowledge discovery. The KDD process is essentially composed by the iterative sequence of seven major steps:

[1] www.google.com
[2] www.yahoo.com

1. **Data cleaning** - the first step of the whole process that is responsible for the removal of noise and inconsistent data that is not relevant for the overall extraction of knowledge;

2. **Data integration** - the step when multiple retrieved and cleaned data sources are combined to produce only one data source to be analyzed on the remaining steps;

3. **Data selection** - the step when limited samples of relevant data are retrieved from the database, so that it can be analyzed on the data processing set of steps;

4. **Data transformation** - the step where data is prepared and transformed into a specific form so that the data mining processes can understand the previous form of data;

5. **Data mining** - the most important step in knowledge discovery, where intelligent mechanisms are employed to the previously prepared data in order to extract data patterns that can define the knowledge we are looking for in this whole process;

6. **Pattern evaluation** - the step when the previous data patterns are identified, based on intelligent measures, and only the truly interesting ones are selected to be the representative knowledge;

7. **Knowledge presentation** - the step where the knowledge found from the whole knowledge discovery process is visually and graphically presented to the user.

Figure 2.1 shows the process model of the KDD.

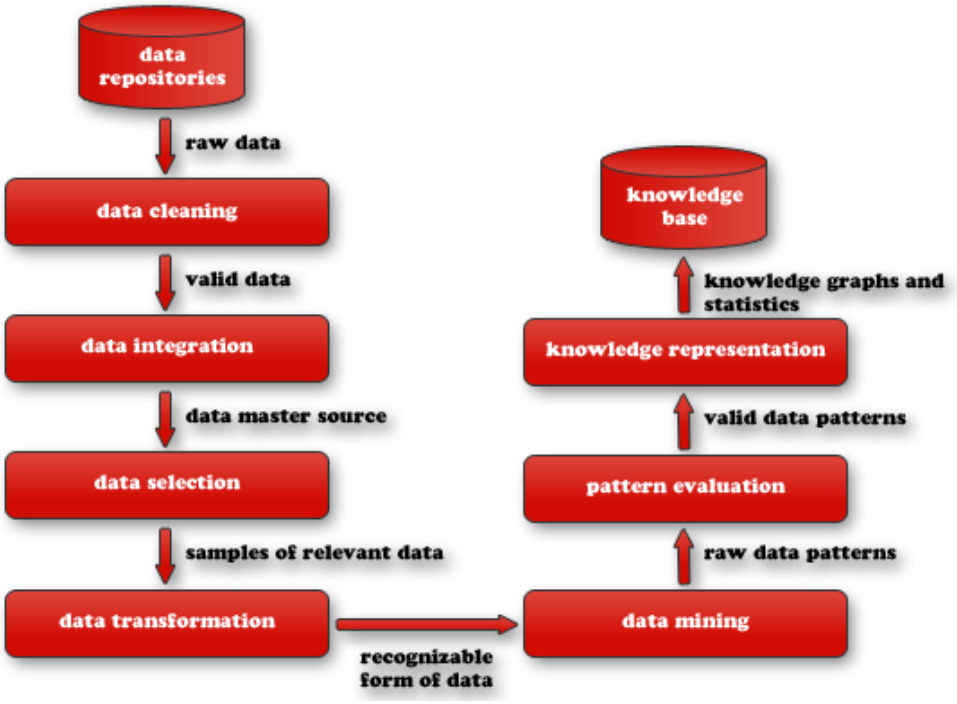

Figure 2.1: Process of the KDD

The CRISP-DM Methodology

One process model that emerged in the industry that is closely related to the KDD process and is considered a standardization of data mining language [HK00] is the Cross Industry Standard Process for Data Mining (CRISP-DM) reference model [CCK+00]. This process model aims for the facilitation of a systematic development of data mining solutions.

Figure 2.2 represents the CRISP-DM data mining project lifecycle.

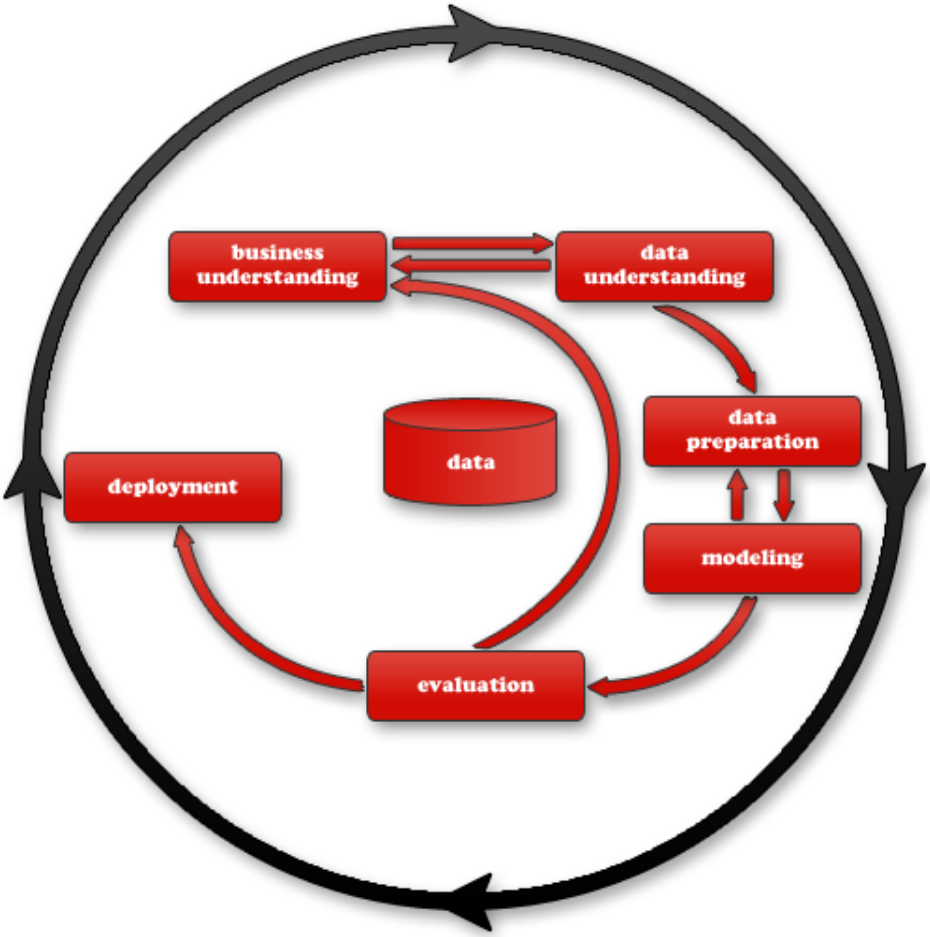

Figure 2.2: CRISP-DM data mining project lifecycle

CRISP-DM describes a typical data mining project lifecycle, which provides guidelines for tackling and solving data mining problems. The CRISP-DM lifecycle is essentially composed of six major phases [She00a]:

1. **Business understanding** - this phase is where the understanding of the project objectives and requirements should take place, from a business perspective. As soon as the objectives are well known and defined, one should define a data mining problem and an early plan, in order to meet the objectives and requirements initially defined;

2. **Data understanding** - this phase is where one should collect initial data and analyze it, in order to discover a global perception of the data. This perception enables further detection of subsets for the discovery of interesting patterns;

3. **Data preparation** - this phase consists of the compiling of raw data in order to construct a dataset ready to be analyzed by data mining tools or modules. This

phase is cyclical and can be performed numerous times, in order to find the best dataset possible to fufill the business objectives;

4. **Modelling** - this phase represents the Data Mining step of the KDD process, where intelligent modelling techniques are selected and applied. Sometimes, one needs to get back to the data preparation phase, due to the requirements of some modelling techniques;

5. **Evaluation** - this phase takes place after the building of the modelling phase in the project. But before proceeding to the final phase of this process, one needs to evaluate and review all previous phases that were executed to construct the model, in order to know with certainty that the business objectives were achieved. After doing that, one must decide what usage will be given to the data mining results;

6. **Deployment** - this phase is about using the knowledge gained in the previous phases and deploying it to the organization's decision making processes.

The basic architecture of a knowledge discovery system has to inherit the previous stated steps and integrate all the components in an effective way so that it can meet its objectives. The basic architecture of this kind of systems is represented in Figure 2.3:

Figure 2.3: Basic Architecture of the KDD process

To conclude, KDD is a fast growing interdisciplinary field that includes areas such as: database systems, data warehousing, statistics, machine learning, data visualization, information retrieval, pattern recognition, spatial data analysis, image databases and signal processing.

2.2.2 The Importance of KDD

Nowadays, there is an high and instant availability of huge amounts of data, but the information and knowledge that derives from that data is not easily accessible. In fact, it can be stated that we are data rich but information poor [HK00]. So, the urge to retrieve such useful information and knowledge from huge amounts of data has lead, over the years, to the necessity of creating the means to fulfill it. In order to extract knowledge from data, the first step that needs to be taken is the development and evolution of sophisticated and powerful database systems, which took place since the 1960s. This evolution led to an era where there were so many different sophisticated kinds of databases and information repositories, such as data warehouses with analytical and decision making tools, yet so few data detailed analysis tools, such as data classification, clustering, and the characterization of data. At that time, the abundant data that was stored on databases wasn't used as it should have been, by the lack of powerful data mining tools and the human incomprehension of that abundant data. Consequently, important decisions were often based on human intuition, and made the retrieval of data pointless [HK00]. So, the future relies on data mining tasks in order to retrieve knowledge from abundant amounts of data and uncover important data patterns, contributing greatly to business strategies, market analysis, fraud detection, customer retention, production control, science exploration and other knowledge bases [HK00].

As it is common in the literature, we will use the terms KDD and Data Mining interchangeably from now on.

Data Mining Problems

When performing a set of data mining tasks, it is important, in the knowledge discovery process, for the users to basically understand what type of data matters and what kinds of patterns in that data can be interesting. It is a very difficult task and it may be impossible to understand this immediately and intuitively, so multiple and parallel searches of different kinds of patterns must be applied. In order to accomplish the previously stated situation, one needs to be in possession of a data mining system that can mine different kinds of patterns for multiple situations, applications and expectations, and that is able to receive guidance from users to focus the search for the most relevant patterns in data. A data mining system must meet all these purposes by having two categories of data mining

tasks: descriptive, where the general properties of the data to be analyzed is characterized; and predictive, where inference on data to make predictions is performed [HK00]. In this thesis, we will only address the descriptive tasks of cluster analysis and association rules discovering, due to the context of this project.

Association rules

Before knowing what association rule mining really means, one must know the definition of frequent patterns. So, according to [HK00], frequent patterns, as it is suggested by the name, are patterns that occur frequently in data. An itemset is frequent when the set of items frequently appear together in a dataset. The importance of mining frequent patterns is that it leads to the discovery of interesting association rules derived from data. By analyzing frequent patterns, one can know several different association rules, with a support percentage and confidence percentage. For example, if we have the association rule ”rock” => ”indie”, with a 90% support and a 90% confidence, that means that an artist that plays rock music has a 90% chance or certainty of also playing indie music. The 90% support means that 90% of the data analyzed showed that an artist that plays rock music also plays indie music. Basically, if a minimum support threshold and a minimum confidence threshold is defined, some association rules may be considered uninteresting if they don't satisfy both minimum support and minimum confidence measures.

Cluster Analysis

A cluster is a group of data objects that are somehow similar to each other, and differ from objects of other groups. According to [HK00] clustering is the process of grouping the data objects into clusters of similar objects, and a form of data compression and segmentation.

Clustering has been an important process for numerous of areas, such as data mining, statistics, machine learning, spatial database technology, business, biology and many other areas that require pattern recognition, data analysis, and image processing. As a data mining function, clustering can also be used as a tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for in-depth analysis.

Applications of clustering ask for a particular set of requirements in order to successfully find clusters that are coherent to reality, such as: **scalability**, in other words, the ability to process different sizes of data sets; **ability to deal with different types of attributes**, such as numerical, nominal, binary or a fusion of all; **discovery of clusters with arbitrary shape**; **ability to deal with noisy data**; **incremental clustering and insensitivity to the order of input records**; and **clustering based on constraints**.

So, this process is very important for this project, because with it, we can identify regions in object space and discover distribution patterns and interesting correlations among Last.fm data attributes. Just like it happened in our childhood when we made distinctions between objects and created subconscious clustering schemes, this automated clustering must be done in the same way, regarding the Last.fm data of this project.

Algorithms for Data Mining

Although there is a variety of algorithms to solve the previously stated problems, for the context of this project the most popular ones were chosen for each data mining problem. The following sections present a brief explanation of the Association Apriori and Clustering k-means Algorithms.

Association Apriori Algorithm

According to [HK00], Apriori is a classic algorithm proposed by R. Agrawal and R. Srikant in 1994 [SA97] for mining frequent itemsets in order to learn association rules from that. The algorithm uses prior knowledge of frequent itemset properties, mainly collected from databases of all sorts, and its name is based on that fact. In association mining problems, given a set of itemsets, this kind of algorithm attempts to find subsets which are common to at least a minimum number C of the itemsets. So, the algorithm uses a level-wise iterative search, where k-itemsets are used to explore (k+1)-itemsets, by the following sequence:

- Initially, the set of frequent 1-itemsets, denoted L_1, is found by scanning the dataset, in order to count each item, and collect the items that satisfy a minimum support;
- Then, the previously set L_1 is used to find the next set of frequent 2-itemsets, the L_2 set, that has counts of a pair of two items with a minimum support;
- The previous step is repeated in order to find L_3, L_4...,L_k, until there is no more frequent k-itemsets available to be found.

The previous process is considered the first step for this algorithm, and the second is the effective generation of association rules with all L_k itemsets with constraints of a minimal threshold support, in order to create interesting association rules.

The Apriori property present in this algorithm is used to improve the efficiency of the generation of frequent itemsets and reduce the search space, in other words, all nonempty subsets of a frequent itemset must also be frequent. So, if an itemset does not satisfy a minimum support threshold, then that itemset is not frequent, and also, if an item is added

to an itemset, then the resulting itemset cannot occur more frequently than the original one.

The following process is a brief explanation of how the previous property is used in the context of the algorithm, where C_k is a candidate itemset of size k, and Lk is a frequent itemset of size k. The first step is responsible for finding a frequent set L_{k-1}, then the second step, called join step, is responsible for generating C_k by joining L_{k-1} with itself, and finally, any (k-1)-itemset that is not frequent is removed, because it is not a subset of a frequent k-itemset. This process is repeated until there is nothing to be compared by this algorithm.

Clustering k-means Algorithm

According to [HK00] the clustering k-means algorithm is responsible for the division of a set of n objects into k clusters, in a way that the resulting intracluster similarity is high but the intercluster similarity is low. The measurement of this similarity is possible because of the mean value of the objects in a cluster, which is commonly named cluster's centroid. As said previously, the algorithm takes a parameter k from input, which will divide a set of objects into k clusters. Then, the algorithm chooses k objects to be the k initial clusters centroids, and each object is distributed to a cluster according to the similarity to the cluster center. After that, the centroid of each cluster is recalculated based on the current objects in the cluster, and the objects are distributed again, using the recalculated centroids. The process terminates when no redistribution of objects is possible, and the resulting clusters are provided to the user.

Tools for Data Mining

This section enumerates some data mining tools that can aid us in the discovery of data patterns and trends with the data collected from Last.fm. In the following sections, we describe three tools for data mining: Weka, Perl Association Data Mining Module and the RapidMiner programs.

WEKA

WEKA is free machine learning software implemented in Java and developed by the University of Waikato in New Zeland [oW].

WEKA is able to perform several standard data mining tasks, like clustering, classification, regression, associations and data pre-processing problems by analyzing data available on a single file, that contains a fixed number of attributes [Gar95].

WEKA provides its users the freedom to use not only numeric or nominal attributes, but any kind of attribute, on the assumption that the user as already defined the overall universe of elements to be considered. This tool also has very short processing times, when running the program by command-line terminals, which enables the processing of datasets with many attributes in short periods of time.

As it was said previously, WEKA expects a single file, but it is very restrictive. In most of the cases, data is stored in a database, which Weka cannot understand, due to the necessity of knowing type information about each attribute which cannot be automatically deduced from the attribute values. So, in order to make Weka analyze anything, one needs to convert data in the ARFF file format [ea08].

Although Weka provides simple access to SQL databases using Java Database Connectivity, it can only process results returned by a single database at a time [ea08], which is a problem when dealing with multiple databases, just as we intend to do.

The interfaces of Weka are presented to the users by interacting with a command-line terminal or by interacting with a GUI. Although the GUI is most of the times more simple and intuitive to the user, the command-line interface presents better processing response times and allows the execution of more complex data mining tasks.

RapidMiner

RapidMiner is an open-source environment for data mining experiments. [RI] states that Rapidminer is a data-mining solution that simplifies the construction of experiments and the evaluation of different data approaches. Rapidminer uses a GUI in order to provide an integrated development environment for data mining tasks.

Perl Association Data Mining Module

This module is not as complete as WEKA or RapidMiner, since it only covers association data mining problems [Fra]. The inability to cover any other data mining problem, such as classification or clustering, is well compensated by its simple use, and by not needing to receive data that is pre-processed too much, as it happens in WEKA. However, this module has the constraint of only supporting data sets with two atributtes, and its ability to process data is not as fast as the processing times of WEKA.

2.3 Conclusions

Analyzing the similar studies and applications to our project that we surveyed in this chapter, we can see that those studies focus on certain aspects of music only (such as tags, band members or hit songs), which differs from our general perspective of researching

multiple variables and measures for our experiments. For the discovery of interesting knowledge and patterns about music our approach is more global and it consists of the gathering of as much as music information we can possibly retrieve. Since Last.fm is a musical community where the music specialists, the fans, actually input data about it, we think that we don't need to look for other musical webpages in Google or Yahoo! to fulfill our objectives.

Although, as stated previously, data mining is an effective way of studying data trends, and of gathering and grouping data objects in order to detect interesting patterns and knowledge. We must have into consideration, however, that data mining is not "magic" in disguise to understand Last.fm as we want, and that there is the possibility that with the Last.fm data that we have in hand, data mining cannot produce interesting and new knowledge about artists and their data. Another aspect that we should take into consideration is that, although the data mining step in the knowledge discovery process is a very important one, one needs to spend a lot of time pre-processing data, removing the noisy values, and having a global understanding of how Last.fm data interacts and impacts the project.

In this part of the project, we didn't know how much data we needed to retrieve from Last.fm, but our suspicions pointed to millions of artists. So, for that amount of data we needed data mining tools that had faster data processing times. Since the GUI generally drops the processing times, we selected the WEKA and Perl Association data mining tools that provided a command-line interface where we could make our experiments and have high data processing times.

Chapter 3

Specifications

The specifications attend to all the considerations made in the beginning of the project, and are an important list of guidelines during the implementation of the Last.Fm crawling system. The features of the system were based on the KDD data analysis approach and the CRISP-DM methodology in order to fulfill our top priorities and business objectives.

Then, we present the system architecture, the details of its modules and how they interact with each other. The system architecture provides us guidelines for the implementation phase of this project, too.

During the preliminary analysis of the system, it was discussed that the Last.fm crawling system would have the purpose of fulfilling the data mining pre-processing tasks (Data cleaning, Data integration, Data selection and Data transformation) of the Last.fm database, and graph plotting tasks of that pre-processed data.

The Last.Fm crawling system would mainly have two components in order to meet its purpose:

- The Last.fm crawling component;
- The Last.fm data mining component.

3.1 Last.fm crawling component

This module has the capability of gathering all data from Last.fm, including artists, their albums, their songs, their fans, their tags, their shouts and their related artists. So, in order to meet its purpose, this module has the following main features:

- discovery of all Last.fm artists and their similarity to other ones;
- retrieval of artist tags;

- retrieval of artist albums;
- retrieval of artist shouts;
- retrieval of artist tracks or songs;
- retrieval of artist basic statistics;
- retrieval of artist fans;
- gathering of the previously retrieved information and artists' global statistics.

Each feature is not independent and has a specific order to be executed, for example, the retrieval of artist data (tags, albums, shouts) and its global statistics would be impossible if the discovery of Last.fm artists didn't take place before.

So, each feature should be executed in iterations, and until the previous one wasn't completed, the next couldn't be started.

The following sections describe in more detail the purposes and implications of each feature of the Last.fm crawler module.

Artist discovery

This module should be responsible for gathering all of Last.fm artists and their similarity with other ones, and build all of the artists' relations, which should represent the first step of all data retrieval from Last.fm. It was discussed that this module would firstly receive the most considered popular artist in Last.fm and begin the discovery of its similar artists. Once the similar artists to that popular artist were discovered, the process of discovery of each one of them too much, and so on, until all artists in Last.fm were discovered. The discovery of artists presupposed not only the retrieval of the artist name, but also the general information of that artist, like artist profile and image links, and the degree of relation with other artists, so that it could be differentiated which artist is more similar to other artists and at what level.

Retrieval of artist data

Assuming that the discovery of artists was successful, this module would be in charge of:

- consulting each artist and discover all data concerning that same artist, regarding tags, albums, fans, songs, shouts and other info;
- gathering the previously data for each artist, in order to be consulted later for other modules.

It was discussed that each type of data in Last.fm regarding tags, albums, fans, songs and shouts should have an unique and independent crawler for parallelization and faster crawling purposes.

Gathering of global statistics

Finally, the last step of the Last.fm crawler would be responsible for:

- collecting all data previously fetched from Last.fm;
- generating global statistics for each artist in Last.fm in order to be much easily analyzed by the user or other modules of the Last.fm crawling program.

3.2 Last.fm data mining component

The purpose of this module was defined, so that:

- it could access all data gathered from Last.fm;
- it could plot some line or bars graphs, so that data could be more easily comprehended and analyzed;
- it could create datasets with relevant information, in order to be analyzed by the data mining tools presented in the ”Analyzing the Last.fm Data” chapter.

3.3 General Implications

Finally, with all features clearly stated, some major implications that the Last.fm crawling system should also fulfill were discussed. The following list states those implications:

- the program should be able to fetch and gather all relevant information from Last.fm in a defined period of time, and should be cyclical. It was defined that the program should fetch all Last.fm data every fortnight;
- the program and its elements, such as crawlers or parsers, must be tolerant to flaws;
- the conception of the Last.fm crawling program should be based on an existent crawling prototype implemented by Luís Sarmento;
- it should be possible to launch multiple instances of each crawler in order to optimize the time of the crawling process;
- the system should be implemented in order to run on Linux and MacOSX operating systems.

3.4 System Architecture

Based on the Last.fm crawling program requirements discussed previously, the context diagram, and the basic architecture of the typical data mining system presented in the ”Analyzing the Last.fm Data” chapter, the system architecture of the Last.fm crawling program was sketched. Figure 3.1 represents the architecture of the system:

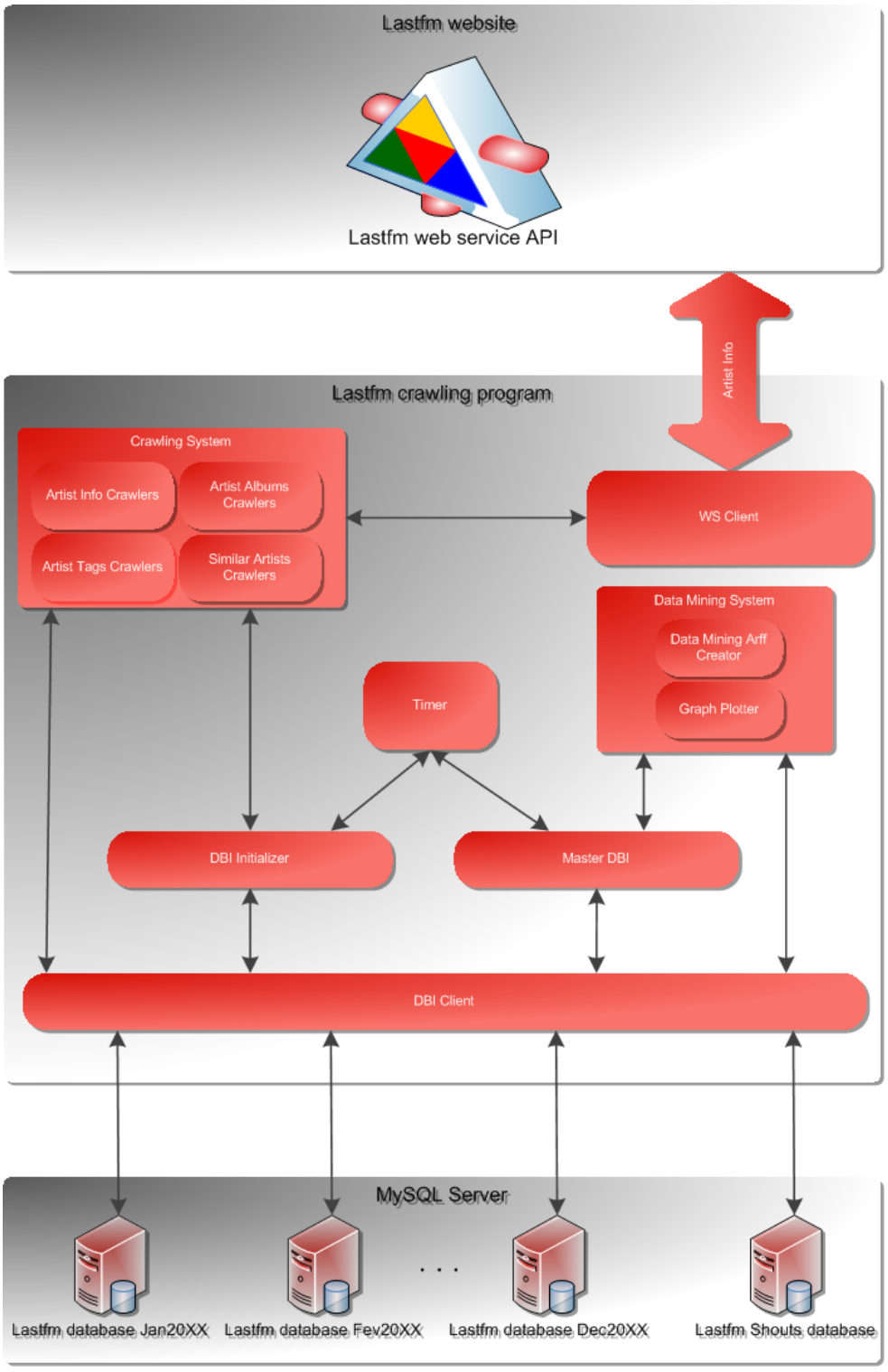

Figure 3.1: System Architecture

The diagram depicted in Figure 3.1 shows us that this system is comprised by: the Last.fm web service API, the Last.fm crawling system and the database server. The fol-

lowing sections will try to explain how each of these modules will interact with each other.

The Last.fm web service API

This web service is provided by Last.fm, which provides an API and enables our system to call specific methods about artists and their related information.

The Last.fm crawling program

This module is the core of all of our system, which gathers all artists and their relevant information from the Last.fm by using its API web service, and redirecting that same information to the specific databases present in the database server.

The main components that define this module are listed below:

- **The crawling system** - this system is responsible for launching all different kinds of crawlers, such as artist info, shouts, tags and albums crawlers, which communicate with the WSClient to request specific information, and communicate with the DBIClient in order to store the previously requested information in the databases. This system also communicates with the DBInitializer module, in order to create and prepare all databases to receive the crawled data. The crawling of shouts is enabled by the communication with the MasterDBI module, which updates the Shouts database;

- **The data mining system** - This system is responsible for communicating with all existent databases of Last.fm data in the database server, via MasterDBI or DBIClient modules, and for fetching all relevant Last.fm data it needs to create datasets. This system is also able to fetch all Last.fm data present in the database server, in order to produce graphs displaying basic statistics and other information;

- **The WSClient** - this module stands for Web service client, and is responsible for communicating directly with the Last.fm via API methods, in order to retrieve the data from the Last.fm website databases. As stated previously, the WSClient responds to all crawlers in this system with the data from Last.fm;

- **The DBIClient** - this module stands for Database interface client, and is responsible for communicating directly with the database server's Last.fm and Shouts databases. This client can perform basic operations of inserting and consulting the data fetched by the rest of the modules;

- **The DBInitializer** - this module is where all the process of crawling begins and stands for Database Initializer. It is responsible for the creation and the preparation of all databases, in order to be ready to store all data that comes from the Last.fm website. It receives temporal information of the month and the year from the Timer module, in order to fully classify each database by its date. When the crawler system has crawlers running, they communicate with this module in order to create the database and the tables that are ready to store the data fetched from Last.fm. Finally, this module communicates with the DBI Client module by giving it the basic operations for creating and preparing all databases and tables;
- **The MasterDBI** - this module stands for Master Database Interface, and has a global access to all existent databases. It is also responsible for fetching artists' names and updating the Shouts database via the Shouts crawler by receiving information of the current date from the Timer module. This module communicates with the data mining system by providing all necessary info from the databases, producing datasets and displaying graphs;
- **The Timer** - this module is responsible for giving temporal information of the current date to the overall system, which enables the creation of temporal databases and enables the crawling of shouts from a specific point of time.

The database server

This server meets the purpose of storing temporal snapshots of Last.fm data in databases that can be identified by the date they were created. It also stores the Shouts database, that contains all shouts from the most popular artists in Last.fm since the beginning of this project.

Chapter 4

Implementation

The development phase of the Last.fm crawling is described in this chapter of the thesis. In the first section, the technologies involved in the development process are briefly described and some conclusions about their applications are presented. Then, we describe the implementation process of this system. Firstly, we describe the selection of artist attributes of each Last.fm API method, which would be the same as the ones present in our databases. Then, we describe the algorithm that supports each crawler of the system. Finally, we present the artist filter functionality, which was discovered in the middle of the implementation process, and proved to be very useful in the extraction of artists in useful time.

4.1 Technologies and Development Environment

This section presents a brief description about all the technologies used during the development of the Last.fm crawling system. The program was mainly developed in Perl and the editor employed was Eclipse IDE. The combination of these programming tools proved to be very useful and practical due to the inbuilt support for development, the easily available documentation and the help of the Internet for some pertinent situations.

4.1.1 Perl

Perl is a high-level dynamic programming language that was originally developed by Larry Wall in 1987 [She00b]. Perl derives from various programming features and has similarities with other programming languages such as C and shell scripting, but claims to be much more simple to program [WCO00].

This programming language became the chosen one for the development of the Last.fm crawling program, due to its efficient text processing capabilities, easy access to database

systems and web pages on the Internet. Another characteristic is its integration with the CPAN module search engine[1], that provides useful and easy to install modules for Perl [WCO00].

4.1.2 MySQL

MySQL, created by MySQL AB in Sweden, is a relational database management system that runs as a server providing multiple access to a number of databases [Ull06].

Accessing MySQL databases is possible in all major programming languages by using specific APIs, libraries and plugins. MySQL works on several system platforms, such as FreeBSD, Linux, Mac OS X, Microsoft Windows, and includes several features, such as: openness, application support (like Java, Perl, C/C++ and PHP), cross-database joins, outer join support, several different character sets support and cross-platform support [KRY02].

4.1.3 Eclipse IDE

Eclipse is a software development platform that embodies an IDE and a plug-in system to extend it. Although this platform is designed for developing Java applications, it can also be used to develop applications with other programming languages such as C, C++, Cobol, Python, Perl and PHP, with the installation of several plugins in order to enable the integration of all these programming languages [tfea].

Although Eclipse presents various features and functionalities, some of them would become essential during the development stage of the Last.fm crawler:

- **Perl EPIC plug-in** - open source Perl IDE that is integrated in the Eclipse platform, compatible with Windows, Linux and Mac OS X, which is mainly considered to be the most complete, rich and extensible free Perl IDE available today, due to its integration with Eclipse. Perl EPIC major features include: syntax highlighting, real-time syntax check, Perl auto-completion and content assistance, and Perldoc support [Int];
- **Subclipse's Subversion (SVN) plug-in** - version control system created by CollabNet Inc in 2000, with the purpose of maintaining current and historical versions of applications' source code, web pages, and other documentation [tfec]. This system integrates with Eclipse platform by giving an easy access to current version repositories.

[1] www.cpan.org

4.1.4 Personal Review of technologies

The Perl language and the Eclipse IDE chosen provided a powerful development environment with enough freedom to experiment new situations. The Perl CPAN module search engine provided powerful modules that helped in the development process of this system. MySQL proved to be a good database management system when storing our databases; however, its ability to recover crashed databases took more time to complete than it was previously expected.

4.2 Implementation details

In this section, we will present implementation details of this Last.fm crawling program. First, we describe in detail the Last.fm API methods that were used and selected in order to fetch from Last.fm only the necessary data to meet the purposes of this project. Then we present the implementation details of the most important module of the program: the crawling system. Finally, new modules are presented, due to the massive amount of data that was fetched.

4.2.1 Last.fm API methods

In order to gather the relevant Last.fm data of artists and their associated information, we have selected the most relevant Last.fm API methods regarding that information. Based on the detailed information contained on the Last.fm API page[2], the following methods were selected:

- *artist.getInfo* - this method retrieves general information about an artist, including its statistics in Last.fm and its biography;
- *artist.getShouts* - this method retrieves all shouts regarding an artist;
- *artist.getSimilar* - this method retrieves information about similar artists of a given artist;
- *artist.getTopAlbums* - this method retrieves information about the top albums of a given artist;
- *artist.getTopTags* - this method retrieves information about the top tags of a given artist;
- *artist.getTopTracks* - this method retrieves information about the top tracks of a given artist;

[2] www.lastfm.com/api

- *artist.getTopFans* - this method retrieves information about the top fans of a given artist.

Although all previous methods were included in the application, we will only address the details of the methods considered more important to fulfill the purposes of this project. The following sections will present tables with details of the most relevant attributes that were selected for each method, and the reason why they were selected is presented. Obviously, the same attributes selected from each method would be the same as the ones stored in our databases.

***artist.getInfo* method**

Attribute	Description	Reason to include
listeners	The number of listeners that an artist has in Last.fm	Potential measure of popularity
playcount	The number of times that an artist was played in Last.fm	Potential measure of popularity

Figure 4.1: *artist.getInfo* method attributes details

***artist.getShouts* method**

Attribute	Description	Reason to include
body	The body of a shout written by a user for a given artist	Potential measure of popularity and other tendencies
author	The name of the user that wrote a shout	Keep track of information about a shout
date	The actual date a shout was posted in the Last.fm website	Way of distinguishing shouts in certain periods of time

Figure 4.2: *artist.getShouts* method attributes details

artist.getSimilar **method**

Attribute	Description	Reason to include
name	The name of a similar artist	Identification of a similar artist
mbid	The MusicBrainz id of a similar artist	Keep track of information about a similar artist
match	The similarity measure of an artist with a similar one	Way of selecting the most important similar artists of an artist
url	The URL of a similar artist Last.fm profile page	Keep track of information about a similar artist
image	The photograph of a similar artist	Keep track of information about a similar artist

Figure 4.3: *artist.getSimilar* method attributes details

***artist.getTopTags* method**

Attribute	Description	Reason to include
name	The name of the tag of an artist	Identification of a tag
count	The normalized count of the tag that a given artist has	Way of selecting the most important tags of an artist

Figure 4.4: *artist.getTopTags* method attributes details

***artist.getTopAlbums* method**

Attribute	Description	Reason to include
album rank	The ranking of an album of an artist	Way of selecting the most important albums of an artist
name	The name of the album of an artist	Identification of an album
mbid	The MusicBrainz id of an album	Keep track of information about an album
playcount	The number of times that an album was played in Last.fm	Potential measure of popularity
image	The photograph of an album	Keep track of information about an album

Figure 4.5: *artist.getTopAlbums* method attributes details

4.2.2 Database creation

As said previously, this program would have the ability to create Last.fm databases in a defined period of time. By using the temporal information from the Timer module, the system has its particular way to classify each database created.

The implementation of this mechanism was defined to be structured as the following procedure:

1. Receive information about the actual month and the year;
2. Creation of a database with the name: LastFm_<MONTH><YEAR>;
3. Storage of the name and the basic information of one of the most popular artists in Last.fm.

The storage of the name and the basic information of one of the most popular artists in Last.fm is an essential step in order to prepare the discovery of Last.fm artists.

4.2.3 Crawler system

This module represents the most important module on this system, where the actual gathering of Last.fm data takes place. Using a time scheduler system, this system is able to launch the crawlers automatically and in a certain period of time, which is from month to month. The system uses the following crawlers:

- artist discovery crawler;
- album crawler;
- artist Info crawler;
- tag crawler;
- track crawler;
- fan crawler;
- shout crawler.

Although all of the previous crawlers were implemented in this system, only the more important will be addressed in the following sections:

Artist discovery crawler

This crawler is very important, and it is where the beginning of the gathering of Last.fm data takes place. It is responsible for gathering all artists from the Last.fm website, starting from a given artist that is assumed to be in the database, and iteratively finding its related artists, which will in turn be crawled so that their similar artists can be found. The implementation of this crawler was defined to be structured as the following procedure:

1. Start by finding the similar artists of a given artist, by giving the name of the artist to the WSClient;
2. The WSClient, using the Last.fm *artist.getSimilar* method, retrieves the information of similar artists of a given artist;
3. The information is parsed and only the relevant one about the similar artists is stored in the database, using the DBIClient;
4. The status that defines if an artist was searched is updated, depending on the success in the retrieving similar artists or not;
5. If there are no more artists to be discovered, the procedure ends;

6. Else, there are artists to be discovered and the whole procedure starts again.

The process crawls artists stored in the databases randomly, which permits multiple crawlers to be executed simulteanously. The update of the status of each artist enables the continuity of the crawling process, if the process is interrupted.

After all of the names of the artists and their similar ones were fetched from the Last.fm, the preparation for all the rest of the crawlers is possible.

The implementation of this mechanism was defined to be structured as the following procedure:

1. The artist names previously fetched by the artist discovery crawler are used in the discovery of the information of all crawlers;
2. The crawled status of each artist is updated;
3. Each crawler artist status is reset, so that all information can be retrieved for the previously fetched artists.

As soon as this process ends, each crawler can crawl all artist information regarding tags, albums, shouts, and other statistics.

Album crawler

This crawler checks the databases for the previously discovered artists from the Artist discovery crawler. It is responsible for gathering information of all albums of a given artist. The implementation of this crawler was defined to be structured as the following procedure:

1. Start by finding the albums of a given artist, by giving the name of the artist to the WSClient;
2. The WSClient, using the Last.fm *artist.getTopAlbums* method, retrieves the information of albums of a given artist;
3. The information is parsed and only the relevant one about the albums is stored in the database, using the DBIClient;
4. The crawl status of the artist is updated, depending on the success of retrieving albums or not;
5. If there are no more artists to be crawled, procedure ends;
6. Else, the whole procedure starts again.

The process crawls artists stored in databases randomly, which permits multiple crawlers to execute simulteanously. The update of the status of each artist enables the continuity of the crawling process, if the process is interrupted.

Artist Info crawler

This crawler checks the databases for the previously discovered artists from the Artist discovery crawler. It is responsible for gathering information of all basic statistics of a given artist. The implementation of this crawler was defined to be structured as the following procedure:

1. Start by finding the basic statistics of a given artist, by giving the name of the artist to the WSClient;
2. The WSClient, using the Last.fm *artist.getInfo* method, retrieves the basic statistics of a given artist;
3. The information is parsed and only the relevant one about the albums is stored in the database, using the DBIClient;
4. The crawl status of the artist is updated, depending on the success of retrieving statistics or not;
5. If there are no more artists to be crawled, procedure ends;
6. Else, the whole procedure starts again.

The process crawls artists stored in databases randomly, which permits multiple crawlers to execute simulteanously. The update of the status of each artist enables the continuity of the crawling process, if the process is interrupted.

Tag crawler

This crawler checks the databases for the previously discovered artists from the Artist discovery crawler. It is responsible for gathering information of all tags of a given artist. The implementation of this crawler was defined to be structured as the following procedure:

1. Start by finding the tags of a given artist, by giving the name of the artist to the WSClient;
2. The WSClient, using the Last.fm *artist.getTopTags* method, retrieves the information about tag of a given artist;
3. The information is parsed and only the relevant one about the tags is stored in the database, using the DBIClient;
4. The crawl status of the artist is updated, depending on the success of retrieving tags or not;

5. If there are no more artists to be crawled, procedure ends;

6. Else, the whole procedure starts again.

The process crawls artists stored in databases randomly, which permits multiple crawlers to execute simulteanously. The update of the status of each artist enables the continuity of the crawling process, if the process is interrupted.

Shout crawler

This crawler checks the databases for the previously discovered artists from the Artist discovery crawler. It is responsible for gathering information of all shouts of a given artist. The implementation of this crawler was defined to be structured as the following procedure:

1. Start by finding the shouts of a given artist, by giving the name of the artist to the WSClient;

2. The WSClient, using the Last.fm *artist.getShouts* method, retrieves the shouts of a given artist;

3. The information is parsed and only the relevant one about the shouts is stored in the database, using the DBIClient;

4. The crawl status of the artist is updated, depending on the success of retrieving shouts or not;

5. If there are no more artists to be crawled, procedure ends;

6. Else, the whole procedure starts again.

The process crawls artists stored in databases randomly, which permits multiple crawlers to execute simulteanously. The update of the status of each artist enables the continuity of the crawling process, if the process is interrupted.

4.2.4 Artist Filter

When the system started to crawl Last.fm data, the need to filter artists arose in order to meet the requirement of fetching Last.fm databases every month. Last.fm allows its users to scrobble their songs in order to retrieve as much data of artists, tracks, albums...etc as possible, while they enjoy their songs on a player in their computers. Although its a very efficient way of getting data of artists, it also leads to what we call ”artist junk” due to the inaccuracy of artist tags of some users’ songs, which can be seen as multiple forms of strings, such as URLs and other dispensable info. Extracting the most relevant info

from Last.fm in the crawling process requires the implementation of some sort of filter of information in order to prevent the later processing of the above ”artist junk”.

The importance of the MusicBrainz website

The MusicBrainz website is a user-maintained community music metadatabase that gathers information about artists, their music and albums in a similar way as Last.fm does, but instead of the Last.fm file-oriented scrobbling, it scrobbles data in an album-oriented way, which can limit the inaccuracy of the information and the proliferation of unwanted ”artist junk” [Mus]. Although MusicBrainz is a very important web resource to use in this project, it is not perfect because it can also gather ”artist junk” and, by itself, it is not sufficient to filter all ”artist junk” available in Last.fm.

The Relevance of the MusicBrainz website

In order to prove the relevance of the MusicBrainz website, and that its metadatabase doesn't consist of United States of America artists only, 10 random non-US artists (Japanese, Chinese, Russian, Portuguese, Indian and DJ artists) were researched. This study proved that the MusicBrainz's web resource is a valid asset for the filtering of artists from the whole universe of artists of the Last.fm website.

The list of all researched artists of this study is presented in the Appendix section.

The need to conceive an artist filter for this system altered the initial system architecture and the communication between the crawlers and the WSClient. The redefined system architecture is depicted in Figure 4.6 below:

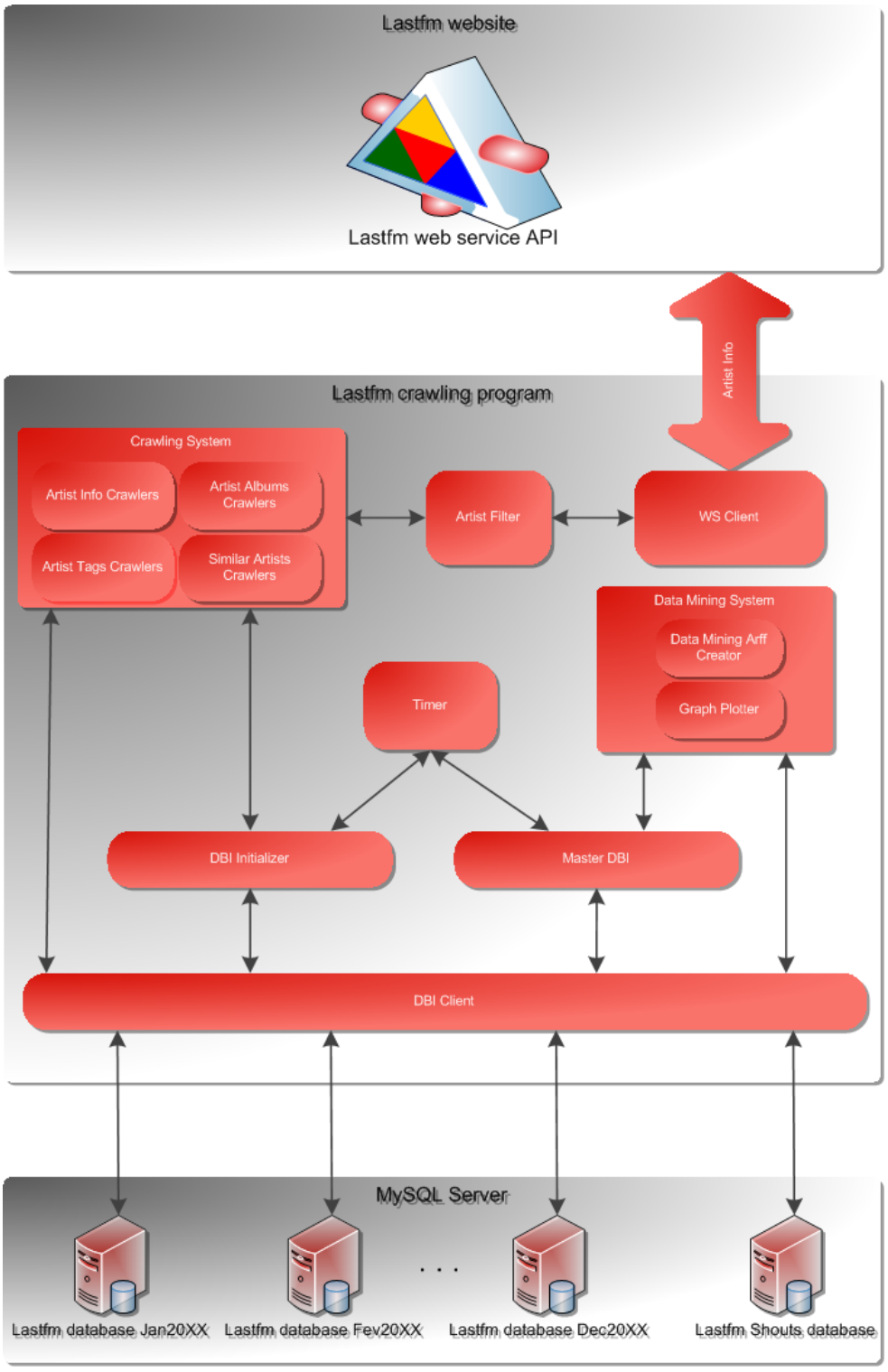

Figure 4.6: System Architecture with Artist Filter module

Artist Filter Implementation

So basically, when the artist filter receives each name from the WSClient, it has to discard names of artists by using two possible approaches: by resource of the Musicbrainz web resource, or by analyzing the name of the artist. By using the Musicbrainz web resource, this module has to discard all artist names that don't have a proper id in the MusicBrainz website (if it's not there it's not considered a real artist or a well-known one). By analyzing the name of the artist, this module has to discard an artist name that:

- contains any kind of URL, website or any other Internet location on its name;
- contains the characters: .,!,?,',<space>,&,, on the beginning of its name;
- is, in fact, an artist featuring another artist, or an artist battling another artist;
- contains the string MP3 or MP4 on its name;
- contains the string various artists on its name, which is not an artist but a group pf artists;
- contains the string Track, OST, Original Soundtrack on its name, which is not an artist but a song or a collection of songs.

4.3 Conclusion

The implementation process took more time than it was predicted in the beginning of the project. In the beginning of the project, we didn't have a single clue of how many artists there were in the Last.fm website. There were a few difficulties in implementing the majority of this system, but as soon as the exploration of an unknown universe of artists in Last.fm began, too much time was spent optimizing the performance of the crawlers and the overall procedure of crawling Last.fm data. Millions and millions of artists being fetched made it impossible to extract Last.fm data every fortnight.

Chapter 5

Prospective Analysis of Data

This chapter describes the analysis of all data retrieved from the Last.fm website. Since the actual analysis of this data was an important goal of this project, in this chapter we deepen the analysis as much as possible.

5.1 Methodology

In order to analyze and comprehend all data taken from the Last.fm website, we followed the KDD major steps. So, the Last.fm crawling system's crawlers implement the first four steps of KDD (data cleaning, data integration, data selection and data transformation), and the Last.fm crawling's plotters and the usage of Weka and the Perl Association rules module data mining capabilities were responsible for the remaining steps (Data mining, Pattern evaluation and Knowledge presentation).

At the beginning of the data mining studies over the Last.fm data retrieved, we followed the CRISP-DM step of data understanding, where one would have to have a global perspective of the data, such as the relation between artists and fans, tags, popularity and many others, and then apply the data mining studies. So the following Visual Analysis section gives that same perspective about that data by presenting the graphs produced by the plotters and other relevant diagrams.

The Data mining section presents the tests that were conducted with the data mining tools. For each test, we present the objective of the test, what was done to conduct that test, and then we report the results, trends and predictions that were originated by the use of data mining problems and algorithms over the data.

5.2 Visual Analysis

This section shows all graphs and visual representations of the Last.fm data we have gathered during this project.

5.2.1 Artist Universe

This section gives a perspective on the whole universe of Last.fm artists, and the intersection with the universe of Last.fm crawling and the Musicbrainz's universe.

In order to understand the artist universes of Last.fm, and both universes of our program using both approches of the artist filter, we performed the following tasks:

- Retrieval of Last.fm artists without the resource of the artist filter module;
- Retrieval of Last.fm artists with the resource of the artist filter module, using the MusicBrainz's web resource;
- Retrieval of Last.fm artists with the resource of the artist filter module, by analyzing the name of the artists.

Once we had the three universes we were able to intersect each universe, by manually searching for artists and querying the databases.

The Figure 5.1 shows the previously stated universes and how they intersect with each other. This figure represents the real intersections on the three artist universes but lacks precision, only giving an idea of how the different universes are related to each other.

Figure 5.1: Overview of the Artist Universe

As we can see, Last.fm is mainly composed by two kinds of artists, the valid artists and the junk artists. The valid artists are the ones that, as the name states, are valid and real with known popularity and information; the junk artists are the ones that, as the name suggests, are junk, not real and are actually user-created by scrobbled misformed tags from songs.

So, as described in the implementation chapter, the artist filter tries to fetch as much valid artists as possible, and as less junk artists as possible, but as it can be seen in the diagram, the Last.fm crawling retrieves some junk artists and doesn't fetch all valid artists, due to the difficult Last.fm artist filter's task of determining whether or not an artist is valid only by its name.

Finally, about a half of the MusicBrainz's artists are included in the Last.fm artist universe, and the Last.fm crawling system is able to get most of those artists.

This section presents some of the relations between Last.fm's attributes which are relevant for this project. The graphs included in this section show those relations. When necessary, and for a better comprehension of values and a better analysis of those values, graph axis may need to be adapted to logarithmic scales, due to the wide range of values.

Due to unexpected delays at the implementation phase of this project, it was not possible to display all information about the overall Last.fm crawling universe in graphs. In order to save time, the usage of a representative sample of all that universe was analyzed.

So, we chose a sample with all artists from Last.fm with a defined Musicbrainz id; as they had a defined Musicbrainz id we were sure that those artists were real and valid. By using valid artists only, we can identify the true characteristics of each artist and we can study and conclude about what characteristics of these artists differ from the characteristics of junk artists. The MusicBrainz id sample contains about 212000 artists, which in our perspective constitutes a good testing sample.

5.2.2 Artist Graphs

This section gives a global perspective about some relations with artists and their info. The following graphs show that same perspective, and for each one of them some conclusions about the information that each one addresses will be given.

Number of characters in artists' names

As it was previously said, one important module of this program is the Artist filter, which distinguishes artists that are valid from those that are junk. So, in the investigation for how the process of filtering artists would work, one characteristic that we considered was the filtering of artists by the number of characters in the artists' names.

By using the artist universe that the Last.fm crawling system retrieved, and the sample mentioned above, we can conclude about what information is inherent to a valid artist, and how we can use that same information to separate the valid artists from the junk ones.

Figure 5.2 displays that same information about the number of characters in artists' names for the whole universe of artists retrieved by the Last.fm crawling program, where each point of the x axis represents the number of characters of a name of an artist, and the y axis represents the total number of artists that have a specific number of characters:

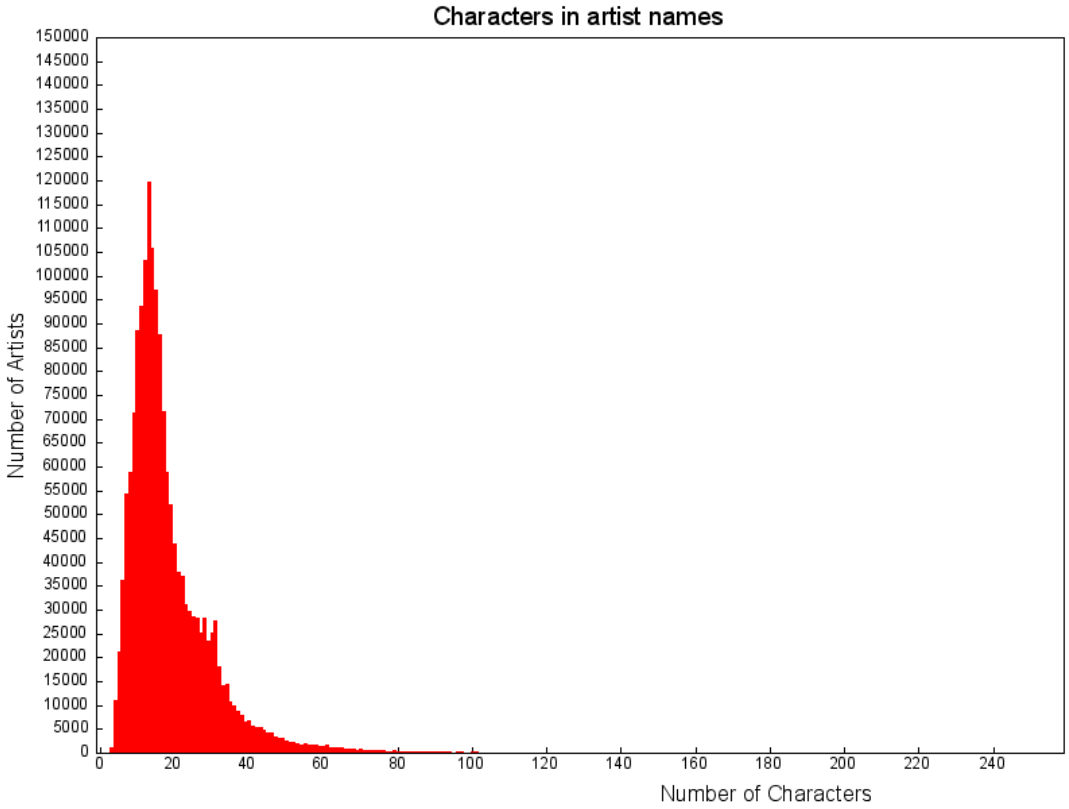

Figure 5.2: Histogram of Number of characters in artists names of the global universe

Figure 5.3 displays the previous information about the number of characters in artists' names, but this time only for the universe of artists with valid MusicBrainz id, fetched by the Last.fm crawling program, where each point of the x axis represents the number of characters of a name of an artist, and the y axis represents the total number of artists that have a specific number of characters.

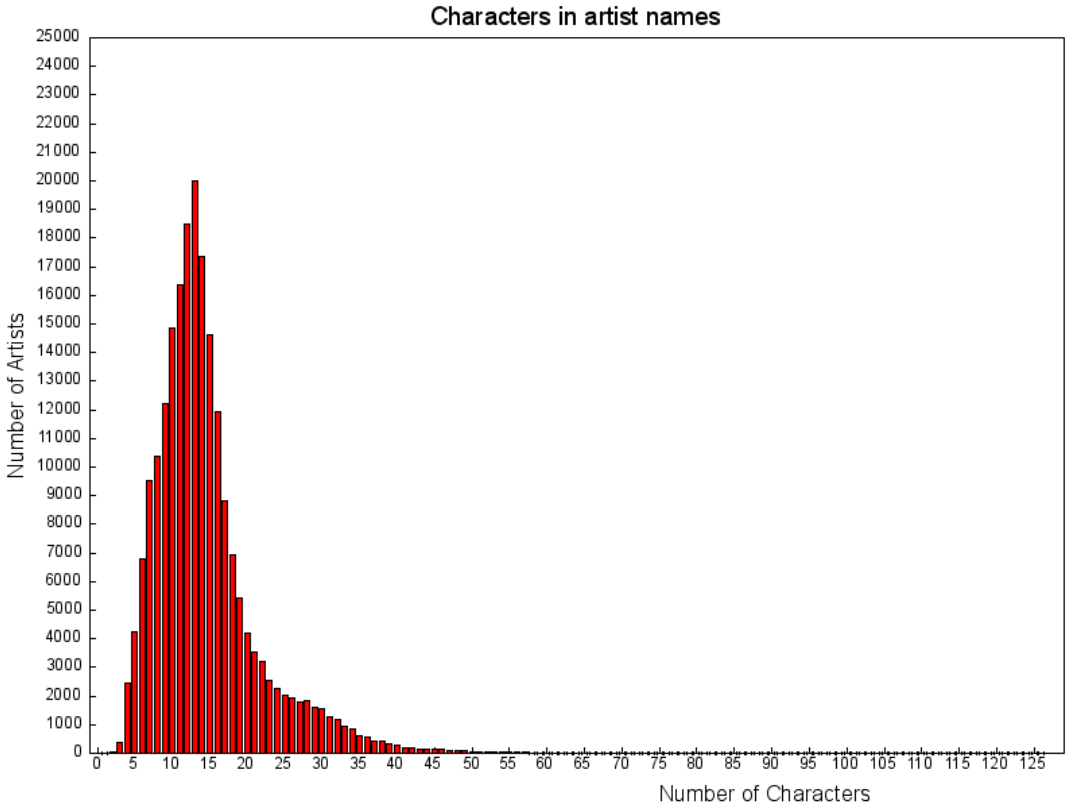

Figure 5.3: Histogram of Number of characters in artists names of the MusicBrainz universe

As we can see, the majority of artists' names has a number of characters that range from seven characters to eighteen characters. However, limiting the retrieval of artists by that range can be very restrictive, because by analyzing this graph we can see that other artists with a name that isn't included in the majority range are in fact valid artists as well, and we can't afford to lose information and discard those same artists, as that could make this project less credible.

By analyzing both previous graphs, some conclusions were made: the graph with the Musicbrainz id universe has a maximum limit of 125 characters; the other, with the whole universe of artists fetched, has a limit of 255 characters. Looking at this information we can infer that the maximum valid limit of characters in an artist's name is in fact 125 characters, and if a name is above that limit we can consider it a junk artist or more than one artist, and it needs to be discarded from the system. Other considerations on filtering artists could not be made, because they were too restrictive for the purpose of this project.

Number of artists by number of albums

Another characteristic that was discussed to be included in the artist filter module was to filter artists by the number of albums each one has or the albums their music was part of. As before, the study reflected the Last.fm artists with Musicbrainz id sample, which assures the validity of all artists. Figure 5.4 displays the information about albums and their artists where each point of the x axis represents the number of albums of an artist, or the number of albums where the music of that same artist is present, and the y axis represents the total number of artists that have a specific number of albums:

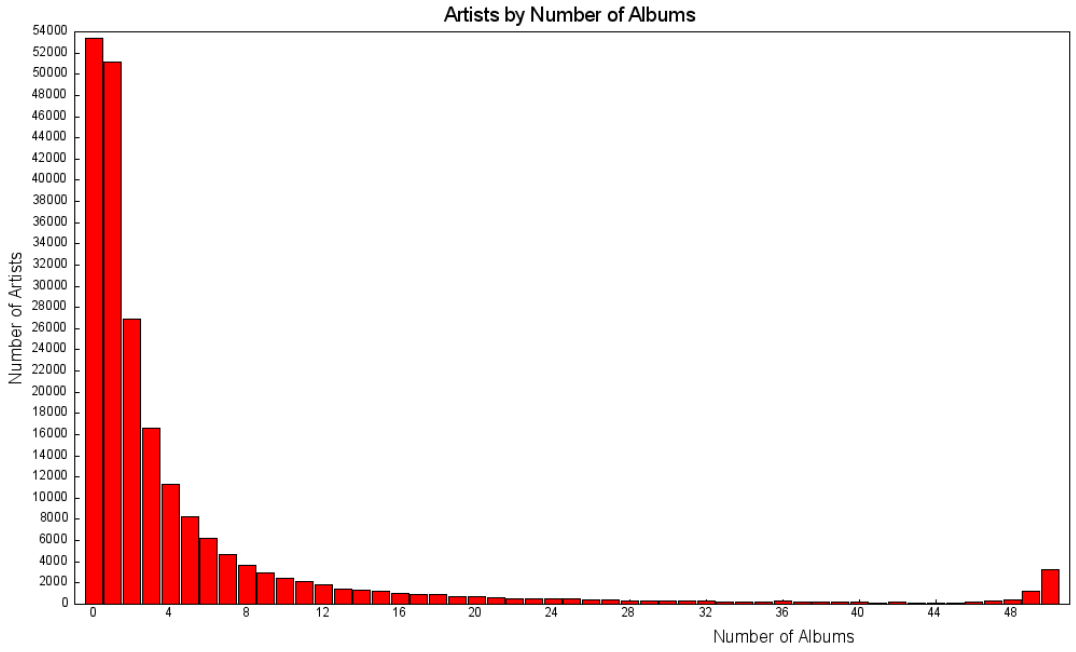

Figure 5.4: Histogram of Number of artists by number of albums

As we can see, there are plenty of artists with albums ranging from zero to three, and not so many for the rest of the number of albums. By analyzing the behavior of the graph, as the number of albums increases, we can see that there are less and less artists with that number of albums.

When the number of albums is 0, we can see that there are plenty of artists that don't have any albums. The reason behind it is that those artists only perform and produce single songs with no albums. When the number of albums is ranged from 48 to 50, we can see a relevant increase compared to the previous values. This increase can mean that not only the most popular artists have albums ranging from 48 to 50, but also the less popular ones. However this increase can also mean that the artists with more than 50 albums were also included here (because Last.fm has a limit of 50 albums per artist).

At first, we might think that the number of albums can measure the popularity of artists, because just like few artists are very popular, few have a great number of albums as well. However, we can't state with all certainty that the popularity of artists is influenced

by the number of albums, just by analyzing this graph. The popularity based on the number of albums can be seen in Figure 5.10.

In conclusion, we cannot filter artists based on the number of albums, because as it can be seen from the graph, various valid artists can have a different number of albums and are as valid and real as any other.

Number of artists by number of tags

Another characteristic that was discussed to be included in the artist filter module was the one that filters artists by the number of tags each one has. As before, the study reflected the Last.fm artists with Musicbrainz id sample, which assures the validity of all artists. Figure 5.5 displays the information about tags and their artists where each point of the x axis represents the number of tags of an artist, and the y axis represents the total number of artists that have that amount of tags:

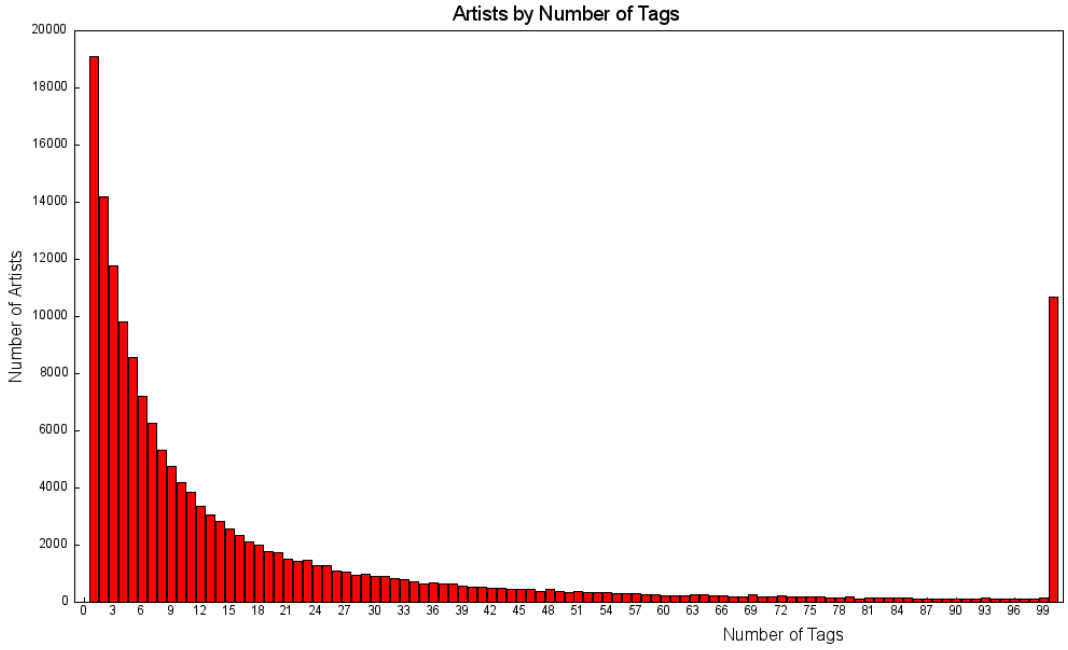

Figure 5.5: Histogram of Number of artists by number of tags

As we can see, the majority of artists has a number of tags that range from one to fifteen and one hundred tags. By analyzing the behavior of the graph, as the number of tags increases, we can see that there are less and less artists with that number of tags, except on the one hundred tags point, which has plenty of artists. This prabably happens because of the fact that users of Last.fm, not only tag popular artists, but also tag artists that are trendy during a certain period of time, but are not famous. The increase in the x

value of 100 can also mean that there are artists with more than 100 tags (because Last.fm has a limit of 100 tags per artist), because those values cover the artists with 100 tags or more. There is no conclusion that can be taken about the relation of popularity and the number of tags only by analyzing this graph, because the most famous artist in Last.fm only has 99 tags, and there are many artists that are not as popular and have one hundred tags. The popularity of artists should not be measured by the amount of tags the artists have, but by the tags themselves (when an artist is not very popular, the tags are less known or used). Filtering artists by the number of tags is out of the question, because as we can see by analyzing the graph, there are various artists that can have a varying number of tags (ranging from one to one hundred), and they are all valid and real.

Number of artists by number of listeners

Another characteristic that was discussed to be included in the artist filter module was the one that filters artists by the number of listeners each one has. As before, the study reflected the Last.fm artists with Musicbrainz id sample, which assures the validity of all artists. Figure 5.6 displays the information about listeners and their artists where each point of the x axis represents an interval of listeners of an artist, and the y axis represents the total number of artists that are present in that interval. Due to the wide range of values in the y axis, a logarithmic scale to the base 10 was used, in order to shrink the range of values and to better understand them and the results of the graph.

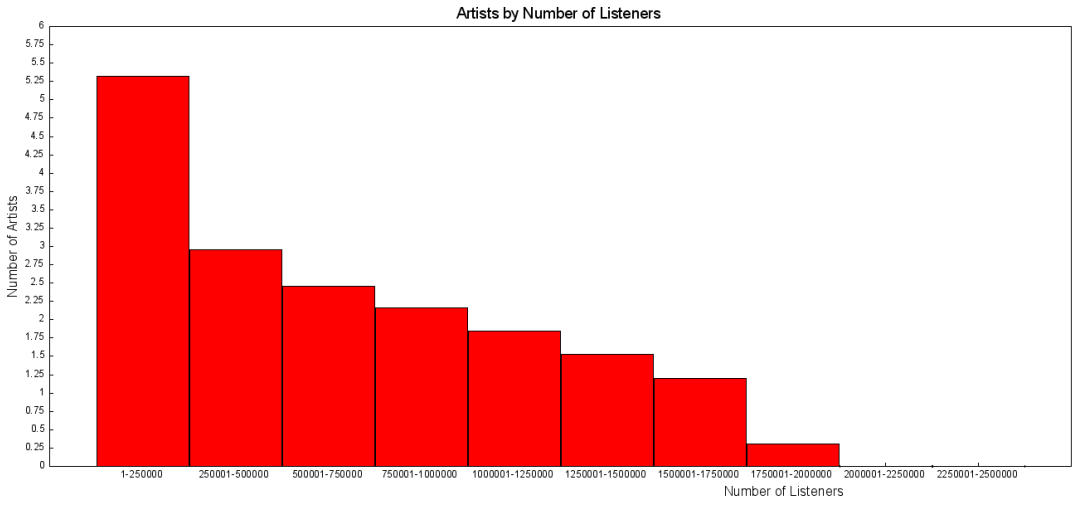

Figure 5.6: Histogram of Number of artists by number of listeners

As we can see, most of the artists in the sample has an amount of listeners ranging from one to 250000 listeners, but only one artist has an amount of listeners ranging from 2000001 to 2250000, and only an artist has an amount of listeners ranging from 2250001

to 2500000. Also, as the number of listeners increases, the number of artists decreases. Popularity is a trait that only few artists have, so, in a certain way, we can conclude that the number of listeners of an artist can really determine its popularity, based on the decreasing of the number of artists as the number of listeners increases.

Number of artists by total playcount

Another characteristic that was discussed to be included in the artist filter module, was the one that filters artists by the number of playcounts each one has. As before, the study reflected the Last.fm artists with Musicbrainz id sample, which assures the validity of all artists. Figure 5.7 graph displays the information about playcount and its artists where each point of the x axis represents an interval of playcount of an artist, and the y axis represents the total number of artists that are present in that same interval. Due to the wide range of values in the y axis, a logarithmic scale to the base 10 was used, in order to shrink the range of values and to better understand them and the results of the graph.

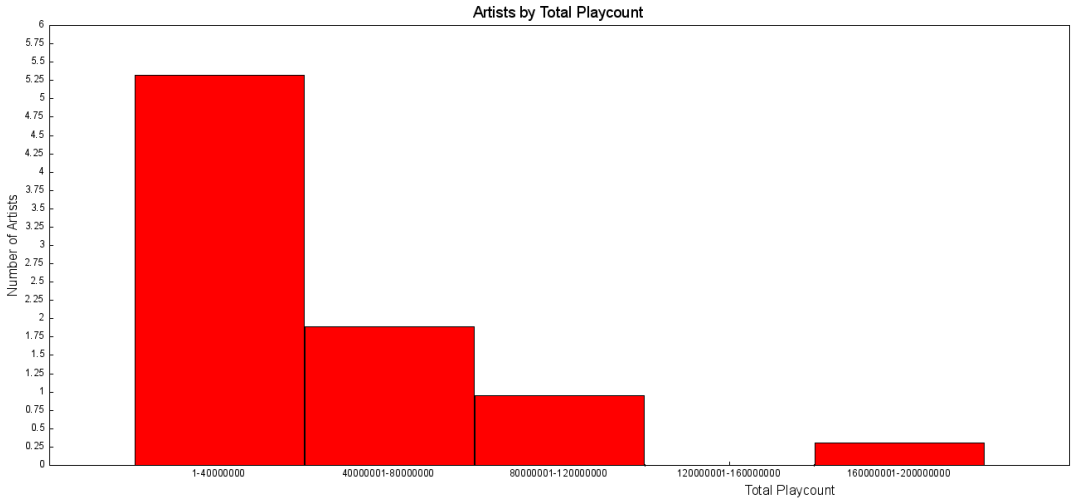

Figure 5.7: Histogram of Number of artists by total playcount

As we can see, most of the artists in the sample has a total playcount ranging from one to 40000000 playcounts, only one artist has a total playcount ranging from 120000001 to 160000000, and two artists ranging from 160000001 to 200000000. Also, as the number of playcount increases, the number of artists decreases, except for the two last ranges of playcounts. By comparing Figure 5.7 with Figure 5.6, we can see that the number of artists decreases with higher values of listeners, but the number of artists doesn't decrease with higher values of playcounts. So, we can conclude that the number of listeners of an artist can really determine the popularity of an artist. The number of playcounts can determine the popularity of artists in a way, but huge values of playcount can be produced

by many listeners and very few alike, which makes the playcount measure an unreliable popularity classifier.

5.2.3 Popularity Graphs

This section presents a global perspective on the relations of different potencial popularity measures of artists, and completes some of the conclusions taken from the Artist Graphs section. The following graphs show that same perspective, and for each one of them, some conclusions about the information that each one addresses will be given.

Artists by Listeners and Playcount

This section shows how the listeners and playcount attributes really determine the popularity of an artist. The graph on Figure 5.8 displays the information about listeners and playcounts of about 208000 artists of the MusicBrainz sample. Each point of the x axis represents the number of listeners of an artist, and the y axis represents the total number of playcounts of an artist. Due to the wide range of values on both axis, a logarithmic scale to the base 10 was used, in order to shrink the range of values and to better understand them and the results of the graph.

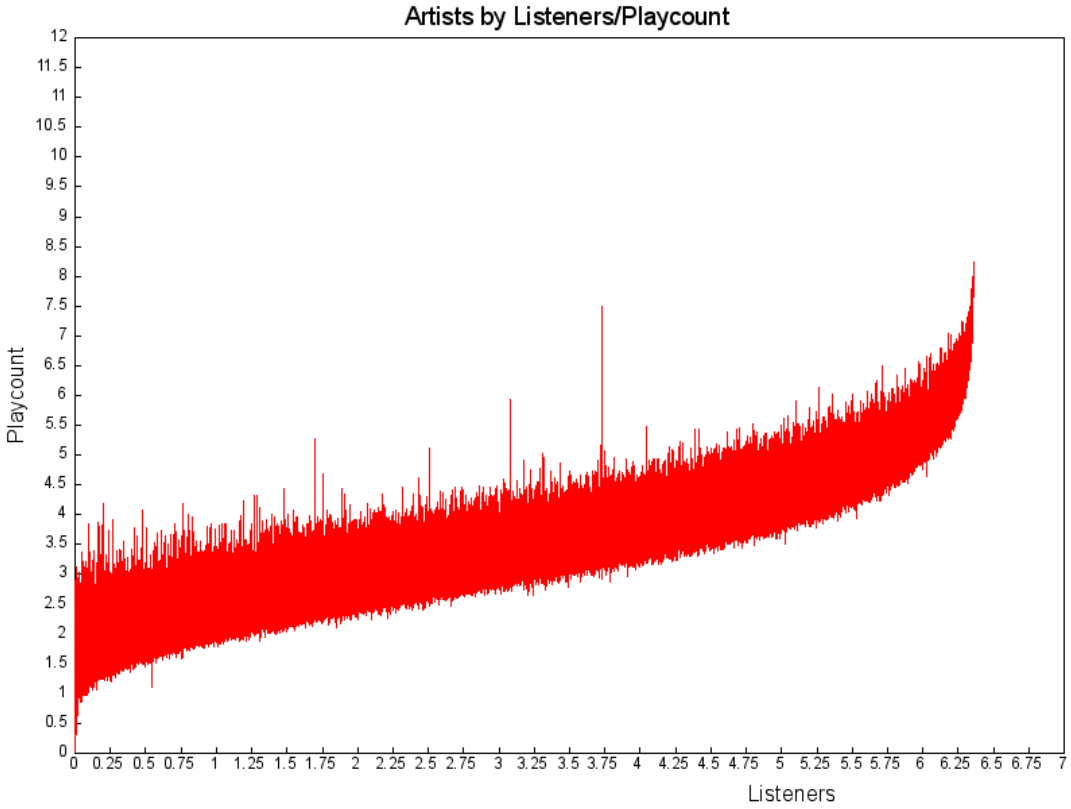

Figure 5.8: Artists by Listeners/Playcount

The results presented here corroborate what was concluded before about the fact that listeners and playcounts can determine the popularity of an artist. As we can see, the tendency of this graph is, as the number of listeners increases, the number of total playcounts increase as well. But there are many cases that contradict this tendency. For example, when the value of listeners is around 3.5 and 3.75 (which represents a range of listeners from 3162 to 5623 listeners) there is an artist with the playcount value of 7.5 (which represents 31622776 playcounts), but then there are some neighbors with a higher value of listeners (which means more artist popularity), but a lower value of playcounts. We can't state with all certainty that the popularity of artists is influenced by the playcounts, but if an artist has a high value of playcounts, there is a great probability that that artist is popular. But by analyzing the graph, we can say that if an artist has low values of playcount that artist is not popular.

Artists by Listeners and Tags

This section determines how the number of tags actually interferes with the popularity of artists. The graph on Figure 5.9 displays the information about the listeners and the

number of tags of artists. Each point of the x axis represents the number of listeners of an artist, and the y axis represents the total number of tags of an artist. Due to the wide range of values on the x axis, a logarithmic axis to the base 10 was used, in order to shrink the range of values and to better understand them and the results of the graph.

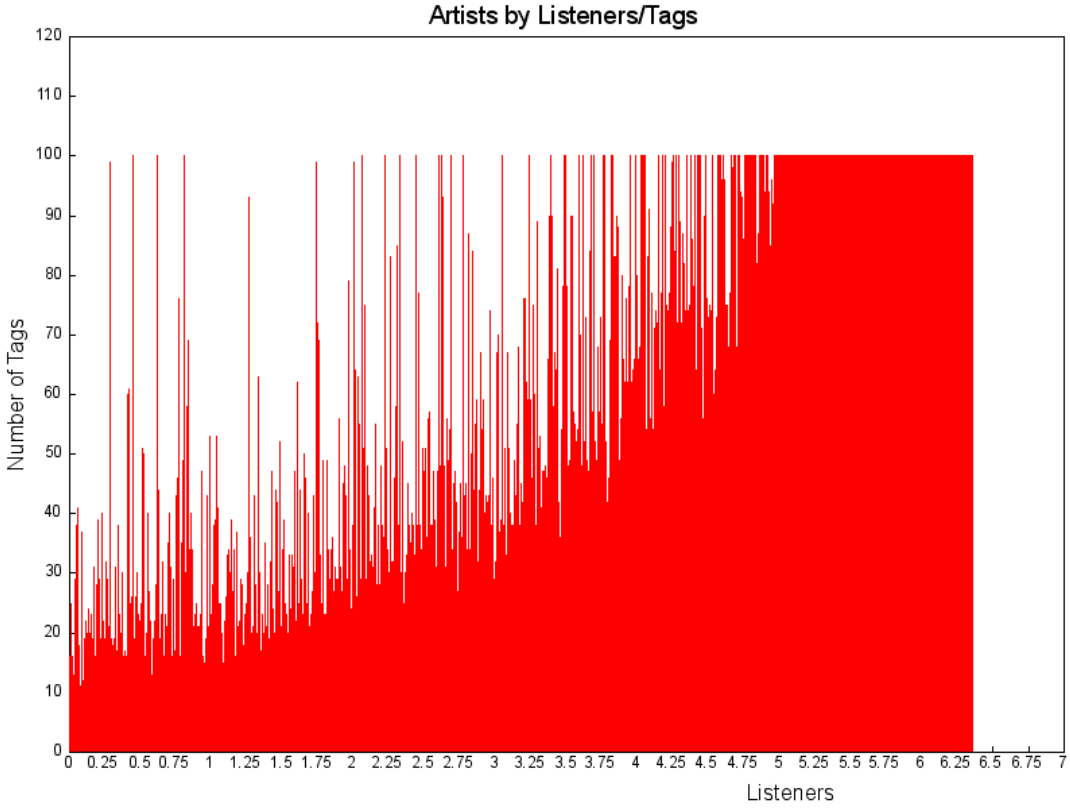

Figure 5.9: Histogram of Artists by Listeners/Tags

As we can see, the tendency of this graph is as the number of listeners increases the number of total tags increase as well. But there are many cases that contradict this tendency. For example, when the value of listeners is around 2.5 and 2.75 (which represents a range of listeners from 316 to 562 listeners) there are several artists with 100 tags, but then there are some neighbours with a value of listeners around 2.75 and 3 (which represents a range of listeners from 562 to 1000) that only have 75 tags. We can't state with all certainty that the popularity of artists is influenced by the number of tags, but if an artist has a high value of tags, there is a great probability that that artist is popular. As stated previously, the popularity of artists should not be measured by the amount of tags the artists have, but by the tags themselves (when an artist is not very popular, the tags are less known or used).

Artists by Listeners and Albums

This section determines how the number of albums actually interferes with the popularity of artists. The graph on Figure 5.10 displays the information about listeners and the number of albums of artists. Each point of the x axis represents the number of listeners of an artist, and the y axis represents the total number of albums of an artist. Due to the wide range of values on the x axis, a logarithmic scale to the base 10 was used, in order to shrink the range of values and to better understand them and the results of the graph.

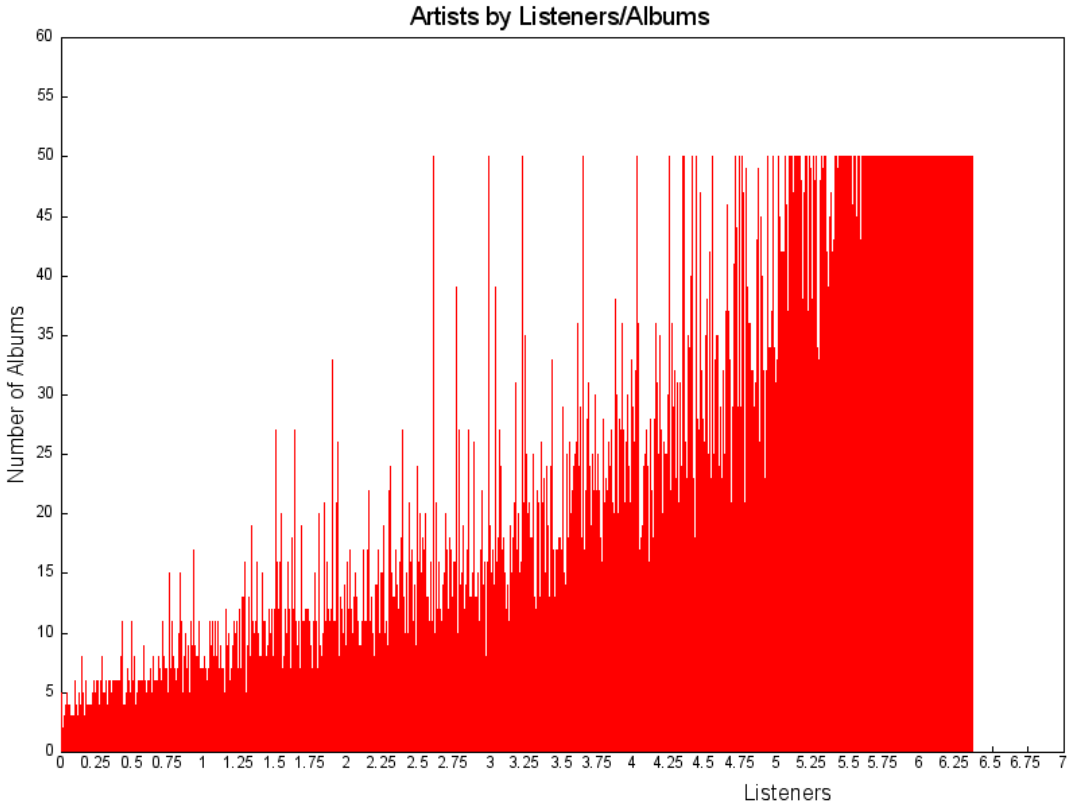

Figure 5.10: Histogram of Artists by Listeners/Albums

As we can see, the tendency of this graph is as the number of listeners increases the number of total albums increases as well. But there are many cases that contradict this tendency. For example, when the value of listeners is around 2.5 and 2.75 (which represents a range of listeners from 316 to 562 listeners) there is an artist with 50 albums, but then there is a neighbor with a value of listeners around 2.75 and 3 (which represents a range of listeners from 562 to 1000) with only 40 albums. We can't state with all certainty that the popularity of artists is influenced by the number of albums, but if an artist has a high value of albums, there is a great probability that that artist is popular.

5.2.4 Tags Graphs

This section gives a global perspective on the relations of artists and their tags, and completes some of the conclusions taken from the artist graphs section regarding tags. The following graphs show that same perspective, and for each one of them, some conclusions about the information that each one addresses will be given.

Top Ten Tags

The following graph is merely informative, showing the 10 most used tags in Last.fm.

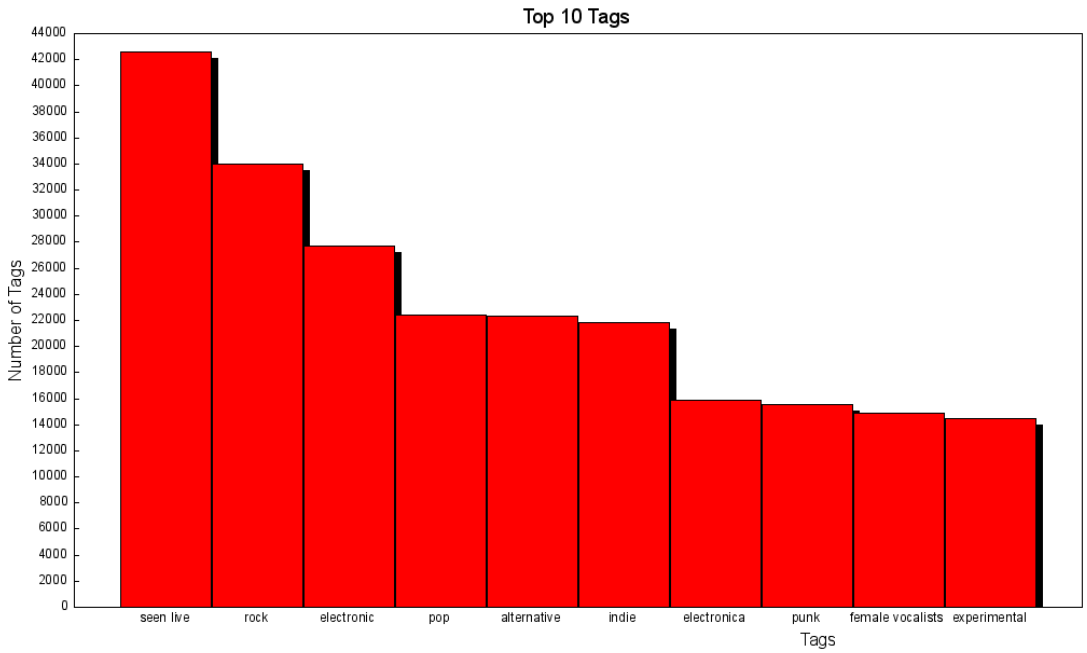

Figure 5.11: Histogram of Top Tags

As we can see, the 10 most used tags in Last.fm are: seen live, rock, electronic, pop, alternative, indie, electronica, punk, female vocalists and experimental. This information is very useful because it can give us guidelines, in the data mining phase in order to try to understand the relations that these tags have between them and between other tags, and how they reflect the tags of an artist.

Number of artists by tags

As it was described in the state of the art chapter, the Last.fm website receives user-input tags from all over the world, and we know that this kind of approach can insert lots of junk information in the Last.fm system. In order to understand how much valid data

regarding tags is present in the Last.fm system, the following graph gives a perspective on the overall usage of tags in Last.fm. Each point of the x axis of this graph represents a specific tag, and the y axis represents the total number of artists that have a specific tag. The graph was decreasingly ordered by usage of tags and due to the wide range of values in both axis, a logarithmic scale to the base 10 was used, in order to shrink the range of values and to better understand them and the results of the graph.

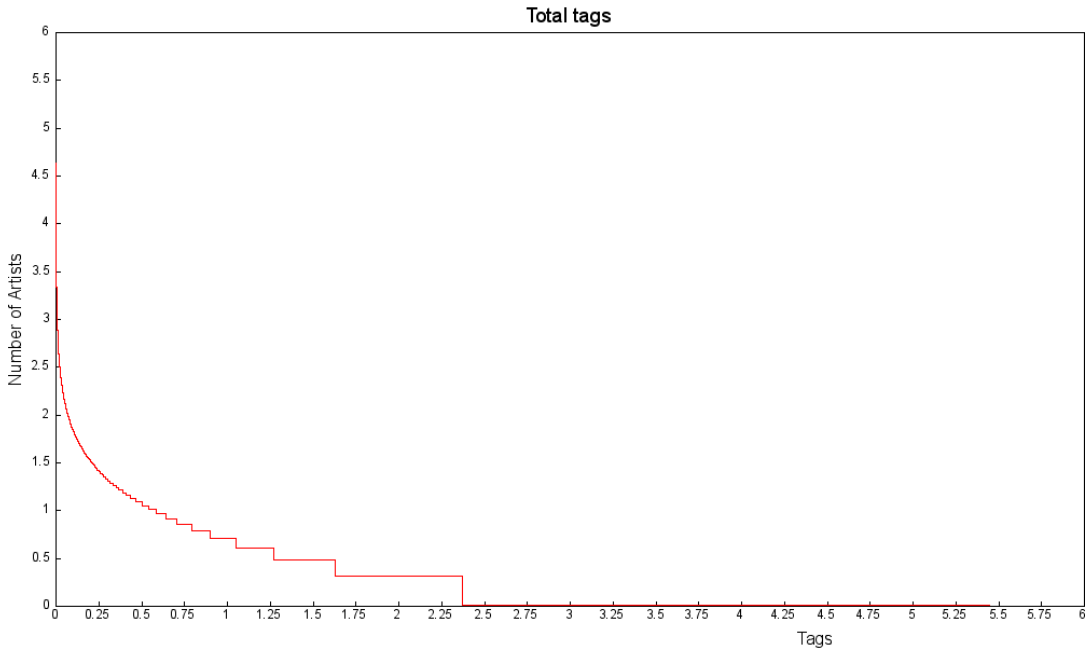

Figure 5.12: Number of artists by tags

This graph includes a universe of about 270000 tags, and as we can see from the curve, after the 2.375 value in the x axis (which is equal to the 237th tag), the tags are only used once. So, in about 270000 tags, only 237 are used more than once, which only represents 0.09% of the sample. Even though the tagging system of Last.fm is effective in artist classification, the system allows for a huge amount - about 99,9% - of junk input-data to be inserted.

5.3 Data Mining based prospective analysis

This section represents the second and most important part of the analysis of the results, and the culmination of all that was previously done in this project. The first part of this analysis was only a data representation of what was retrieved from the Last.fm website, which provided a global vision and a perspective, in order to successfully study data trends and patterns in Last.fm.

Unfortunately, due to the lack of time, it was not possible to create several temporal snapshots of the Last.fm database, in order to determine and predict important measures, such as studies about the evolution of an artist, or prediction of popularity of a group of artists. In addition, and also due to the lack of time, not all the possible data mining testing was performed. In the conclusions and future work chapters we will address in more detail what really needs to be done to perform these data mining tasks of analyzing and studying artist evolution.

So, although we cannot perform complex data mining tasks over a large number of databases with the Last.fm data, there were other data mining tasks that could be performed with only one or two databases. In order to ensure the validity of these studies, the selected databases were the ones with MusicBrainz id entries only. As stated previously, the majority of these artists are valid and real.

The following sections will explain the methodology used for the discovery of trends and patterns with few databases, the results of that same discovery, and the analysis of results of the most interesting patterns found with the Last.fm data. First, we will detail the data mining clustering problems and their tests. Then, we will detail the data mining association problems and their tests.

5.4 Clustering tests

5.4.1 Test 1: Grouping artists by tags

The purpose of this test was to determine how the tags of an artist really relate to the tags of its most similar artist. What we were trying to discover was the percentage of related artists that share the same tags.

Methodology used

In order to fulfill the purpose of this test, we focused on the artist tag data from the databases and selected the following attributes to include in our test files:

- Most relevant tag of an artist;
- Most relevant tag of the most similar artist to the previous artist.

With this pre-processing of data, we prepared it to be used with the Weka clustering algorithm. The data consisted of about 50000 artists, and 25000 pairs of tags, so that the clustering algorithm could analyze the relations between those tags. We thought that this sample would be representative for our studies, and the reason we didn't include more artist tag relations was to guarantee a good performance of the Weka program.

Applied algorithm

So, as soon as we had the test files, the applied algorithm was the clustering k-means. By using the information gathered on the Visual Analysis section about the tags, we defined a test with 10 clusters in order to map the top 10 tags found in the Last.fm system. For this test we used the Weka k-means algorithm because Weka was the only one, from our data mining tools, capable of running clustering algorithms.

Test Conclusions

The results produced by the algorithm proved that:

- 75% of the similar artists analyzed had different tags;
- 25% of the similar artists analyzed had the same tags.

These results actually make sense, because both artists don't necessarily need to have the same tags in order to be similar. Their actual tags can be different but represent similar music genres or similar information, which make them similar artists. In the case of both tags being similar, the artists are similar too.

5.4.2 Test 2: Artist evolution

The purpose of this test was to determine the artist's evolution in a month. What we were trying to discover was the percentage of artists that in a month verified:

- A quick evolution;
- A normal evolution;
- A not so relevant evolution.

Methodology used

In order to fulfill the purpose of this test, we focused on the artist popularity measures data from the databases and selected the following attributes to include in our test files:

- Subtraction of the listeners measure of an artist in two adjacent months;
- Subtraction of the playcount measure of an artist in two adjacent months;
- Subtraction of the album playcount measure of an artist in two adjacent months.

With this pre-processing of data, we prepared it to be used with the Weka clustering algorithms. The data consisted of about 45000 artists, so that the clustering algorithm could analyze the possible growth of an artist. We thought that this sample would be representative for our studies, and the reason we didn't include more artists was to guarantee a good performance of the Weka program.

Applied algorithm

So, as soon as we had the test files, the applied algorithm was the clustering k-means. To achieve the purpose of this test, we defined a test with 3 clusters in order to map the three situations we wanted to study. For this test we used the Weka k-means algorithm because Weka was the only one, from our data mining tools, capable of running clustering algorithms.

Test Conclusions

The algorithm produced its results with three centroids that were representative of the three situations we wanted to analyze. The results proved that:

- 8% of the artists had a quick evolution on both listeners and playcount measures;
- 54% of the artists had a normal evolution on both listeners and playcount measures;
- 39% of the artists had a not so relevant evolution on both listeners and playcount measures.

These results actually make sense based on the following facts:

- It is well known that only a small number of artists have a quick evolution in their musical career, and actually evolve in the popularity charts;
- Most of the artists already have a place in the music industry, and have a normal evolution in the popularity charts as time goes by;
- The rest of the artists can be artists that are being forgotten by their fans, or may be artists that are not well known by the users and were not discovered yet.

5.5 Association tests

5.5.1 Test 3.1: Occurence of tags

The purpose of this test was to determine the occurence of a group of tags. What we were trying to discover was the percentage of confidence in the occurence of a tag or a group

of tags, given a tag or a group of tags. For example, if an artist has a ”rock” tag maybe we can predict with a percentage of confidence that that same artist has the ”new rock” tag.

Methodology used

In order to fulfill the purpose of this test, we focused on the tag artist data from the databases and selected the ten most relevant tags of a group of artists to be the attributes to include in our test files.

The data consisted of about 150000 artists, so that the association algorithm could analyze the possible occurence of tags. We split the data in order to use 90% of it in this test. The other 10% would be used to validate the results this test would produce. We thought that this sample would be representative for our studies.

Applied algorithm

So, as soon as we had the test files, the applied algorithm was the association apriori. To achieve the purpose of this test, we defined a minimum confidence of 90% to be used in this test. With this level of confidence, we can guarantee a high level of probability for results to happen. For this test we used the Weka apriori algorithm only to see what kind of results it would produce.

Test Conclusions

Unfortunately, Weka produced no results for this problem. We think that the reason behind it is that with different possibilities of sets of 10 tags, Weka couldn't find frequent patterns only by knowing ten distinct tags of a given artist.

5.5.2 Test 3.2: Occurence of tags

The purpose of this test was to determine the occurence of a group of tags. What we were trying to discover was the percentage of confidence in the occurence of a tag or a group of tags, given a tag or a group of tags. For example, if an artist has a ”rock” tag maybe we can predict with a percentage of confidence that that same artist has the ”new rock” tag.

Methodology used

In order to fulfill the purpose of this test, we focused on the tag artist data from the databases and selected the ten most relevant tags of a group of artists to be the attributes to include in our test files.

The data consisted of about 150000 artists, so that the association algorithm could analyze the possible occurence of tags. We split the data in order to use 90% of it in this test. The other 10% would be used to validate the results this test would produce. We thought that this sample would be representative for our studies.

Applied algorithm

So, as soon as we had the test files, the applied algorithm was the association apriori. To achieve the purpose of this test, we defined a 90% minimum confidence to be used in this test. With this level of confidence, we can guarantee a high level of probability for results to happen. For this test we used the Perl Association module, due to the failed attempt of using Weka, and also because this module is capable of running the association apriori algorithm.

Test Conclusions

As soon as the algorithm produced its results, we compared the same results with the 10% data sample. The results proved that:

- with 92% of confidence, if an artist has the ”indie rock” and the ”rock” tags, it also has the ”indie” tag;
- with 90% of confidence, if an artist has the tag ”electronica”, it also has the ”electronic” tag;
- with 92% of confidence, if an artist has the ”alternative rock” tag and the ”indie” tag, it also has the ”rock” tag;
- with 91% of confidence, if an artist has the ”alternative rock” tag, it also has the ”rock” tag;
- with 94% of confidence, if an artist has the ”alternative” tag and the ”alternative rock” tag, it also has the ”rock” tag;
- with 93% of confidence, if an artist has the ”indie pop” tag, it also has the ”indie” tag;
- with 93% of confidence, if an artist has the ”alternative” tag and the ”indie rock” tag, it also has the ”indie” tag;

- with 95% of confidence, if an artist has the ”alternative” tag, ”alternative rock” tag and the ”indie” tag, it also has the ”rock” tag;
- with 91% of confidence, if an artist has the ”alternative rock” tag, ”indie” tag and the ”rock” tag, it also has the ”alternative” tag;
- with 90% of confidence, if an artist has the ”indie rock” tag, it also has the ”indie” tag;
- with 95% of confidence, if an artist has the ”hard rock” tag, it also has the ”rock” tag;
- with 93% of confidence, if an artist has the ”alternative” tag, the ”indie rock” tag, and the ”rock” tag, it also has the ”indie” tag.

The results actually make sense, because the tags are similar musical genres or similar information.

5.5.3 Test 4.1: Occurence of similar tags

The purpose of this test was to determine the occurence of artist tags with tags of similar artists. What we were trying to discover was the percentage of confidence in the occurence of an artist tag, given a similar artist tag. For example, if an artist has a ”rock” tag, maybe we can predict with a percentage of confidence that a similar artist has the ”pop rock” tag.

Methodology used

In order to fulfill the purpose of this test, we focused on the artist tag data from the databases and selected the following attributes to include in our test files:

- most relevant tag of an artist;
- most relevant tag of the most similar artist to the previous artist.

The data consisted of about 100000 artists, so that the association algorithm could analyze the possible occurence of tags. We split the data in order to have 90% of it to be used in this test. The other 10% would be used to validate the results this test would produce. We thought that this sample would be representative for our studies.

Applied algorithm

So, as soon as we had the test files, the applied algorithm was the association apriori. To achieve the purpose of this test, we defined a minimum confidence of 90% to be used in this test. With this level of confidence, we can guarantee a high level of probability for results to happen. For this test we used the Weka apriori algorithm to see what kind of results it would produce.

Test Conclusions

Unfortunately, Weka produced no results for this problem. We think that the reason behind it is that with different possible pairs of tags, Weka couldn't find frequent patterns by knowing two tags only.

5.5.4 Test 4.2: Occurence of similar tags

The purpose of this test was to determine the occurence of artist tags with tags of similar artists. What we were trying to discover was the percentage of confidence in the occurence of an artist tag, given a similar artist tag. For example, if an artist has a "rock" tag, maybe we can predict with a percentage of confidence that a similar artist has the "90s" tag.

Methodology used

In order to fulfill the purpose of this test, we focused on the artist tag data from the databases and selected the following attributes to include in our test files:

- Most relevant tag of an artist;
- Most relevant tag of the most similar artist to the previous artist.

The data consisted of about 100000 artists, so that the association algorithm could analyze the possible occurence of tags. We split the data in order to have 90% of it to be used in this test. The other 10% would be used to validate the results this test would produce. We thought that this sample would be representative for our studies.

Applied algorithm

So, as soon as we had the test files, the applied algorithm was the association apriori. To achieve the purpose of this test, we defined a minimum confidence of 90% to be used in this test. With this level of confidence, we can guarantee a high level of probability for results to happen. For this test we used the Perl Association module, due to the failed attempt of using Weka for this test, and also because this module is capable of running the association apriori algorithm.

Test Conclusions

As soon as the algorithm produced its results, we compared the same results with the 10% data sample. The results proved that:

- with 90% of confidence, if an artist has the ”pop rock” tag, its most similar artist has the ”rock” tag;
- with 90% of confidence, if an artist has the ”acid house” tag, its most similar artist has the ”house” tag;
- with 90% of confidence, if an artist has the ”Bossa Nova” tag, its most similar artist has the ”jazz” tag;
- with 90% of confidence, if an artist has the ”country rock” tag, its most similar artist has the ”classic rock” tag;
- with 90% of confidence, if an artist has the ”rock n roll” tag, its most similar artist has the ”rock” tag.

The results actually make sense because the tags of both similar artists are similar musical genres. These results can also corroborate the results of Test 1, where 75% of the similar artists have different tags.

5.5.5 Test 5: Global relations with similar artists

The purpose of this test was to determine global relations with different similar artists. What we were trying to discover was association rules for tags, albums and different popularity measures between similar artists.

Methodology used

In order to fulfill the purpose of this test, we focused on the artist global data from the databases and selected the following attributes to include in our test files:

- information about the name and counts of the three most relevant tags of an artist;
- information about the name and counts of the three most relevant tags of the most similar artist;
- information about the name and counts of the three most relevant tags of the second most similar artist;
- information about the number of listeners of an artist;

- information about the number of listeners of the most similar artist;
- information about the number of listeners of the second most similar artist;
- information about the number of playcounts of an artist;
- information about the number of playcounts of the most similar artist;
- information about the number of playcounts of the second most similar artist;
- information about the number of albums of an artist;
- information about the number of albums of the most similar artist;
- information about the number of albums of the second most similar artist.

The data consisted of about 50000 artists, so that the association algorithm could determine global relations between similar artists. We split the data in order to have 90% of it to be used in this test. The other 10% would be used to validate the results this test would produce. We thought that this sample would be representative for our studies.

Applied algorithm

So, as soon as we had the test files, the applied algorithm was the association apriori. To achieve the purpose of this test, we defined a minimum confidence of 90% to be used in this test. With this level of confidence, we can guarantee a high level of probability for the results to happen. For this test we used the Weka apriori algorithm because Weka was the only one, from our data mining tools, capable of analyzing problems with more than two attributes.

Test Conclusions

Unfortunately, Weka produced no interesting results for this problem. We should think of new kinds of attributes that need to be included in our test files, or we should discard redundant attributes that remove the interesting aspect of the results.

Chapter 6

Conclusions

The development and deployment of the Last.fm system architecture based on the KDD data analysis approach and the CRISP-DM methodology guidelines was achieved successfully. The implementation of this system enabled us to better understand the Last.fm data and it gave us the necessary means to extract even more Last.fm musical data, in order to better understand music information and music popularity measures in the future.

During the development of this system, a lot of time was spent in optimizing its performance for data retrieval, due to the massive amount of data that needed to be retrieved. By exploring the previously unknown amount of the Last.fm data, we discovered that it was not possible to retrieve all of the relevant Last.fm data every fortnight. With millions of artists, our best achievement was to retrieve all relevant data every month.

With the gathering of these millions of artists a new problem arose during this project. We discovered that in the midst of this artist universe, there were many invalid artists, and we spent a lot of time figuring out how we could manage to retrieve as much valid artists as possible, and as less junk artists as possible. So, the need to implement an artist filter arose, and we found a way to filter those artists by analyzing their names. The results produced were satisfactory but not perfect. By analyzing an artist's name, it was certain that we could discard many junk artists - but we also discovered that not all junk artists were discarded, and that some valid artists were discarded as well.

The retrieval of Last.fm data during this project gave us some knowledge about artist similarity, artist evolution and tag relations, but we are fully aware that there is still too much to learn about music from this massive amount of Last.fm musical data. We think that, to better understand this musical data and to analyze it in order to find more complex and interesting knowledge about music, there is still the need to retrieve at least one year of Last.fm data, producing 12 successive Last.fm databases. By having one year of Last.fm data, we think that we could more effectively visualize artists and music evolution during a year, and we could infer and extract more interesting musical knowledge. Another interesting aspect we can find is the seasonal popularity of artists or the impact that a

song has during Christmastime or summertime, for example.

With the data gathered during the course of this project we were only able to do association and clustering data mining tests, because of the few databases we were able to fetch. We are certain that if we had at least one year of Last.fm databases we could try to use different data mining problems and conduct new experiments in order to unveil new musical knowledge.

The use of WEKA and the Perl association module using the command-line interface proved to give results with good processing times, but the learning curve for these programs was too short due to the lack of intuitiveness of these interfaces. Maybe we should reconsider Rapidminer and WEKA GUI for our future data mining experiments.

Chapter 7

Future work

Based on what was said in the conclusions chapter we are fully aware that there is still a lot of work to be done.

First of all, we need to improve the artist filter module so that we can get better filtering results. Maybe we should consider combining the use of the Last.fm website with other web resources, such as Google or Wikipedia[1], in order to produce those better filtering results we need for more accurate results about artists.

Then, we need to gather Last.fm data for at least one year in order to better understand music data and discover new knowledge about music.

After having this amount of data, there is still a need to investigate and study new data mining problems and algorithms so that we can find more interesting and complex musical knowledge.

For the pre-processing data and data mining tasks purposes, we should consider using the GUI of WEKA and Rapidminer for better intuitiveness in our studies and results, or we should integrate the WEKA libraries in our system in order to produce better results.

[1] en.wikipedia.org

References

[CCK+00] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. Crisp-dm 1.0 step-by-step data mining guide, 2000. CRISP-DM Consortium, available at http://www.crisp-dm.org/CRISPWP-0800.pdf.

[ea08] Frank Eibe et. al. Weka manual for version 3-6-0. http://sourceforge.net/project/downloading.php?groupname=weka&filename=WekaManual-3.6.0.pdf, 2008.

[Fin04] Michael Fingerhut. Music information retrieval, or how to search for (and maybe find) music and do away with incipits, 2004. http://articles.ircam.fr/textes/Fingerhut04b/.

[Fra] Dan Frankowski. Data-mining-associationrules-0.10. http://search.cpan.org/~dfrankow/Data-Mining-AssociationRules-0.10/.

[Gar95] Stephen R. Garner. Weka: The waikato environment for knowledge analysis. In *In Proc. of the New Zealand Computer Science Research Students Conference*, pages 57–64, 1995.

[GSK07] Gijs Geleijnse, Markus Schedl, and Peter Knees. The quest for ground truth in musical artist tagging in the social web era. In Simon Dixon, David Bainbridge, and Rainer Typke, editors, *Proceedings of the Eighth International Conference on Music Information Retrieval (ISMIR'07)*, pages 525 – 530, Vienna, Austria, September 2007.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, September 2000.

[Int] EPIC Eclipse Perl Integration. Epic - perl editor and ide for eclipse. http://www.epic-ide.org/.

[KPW04] Peter Knees, Elias Pampalk, and Gerhard Widmer. Artist classification with web-based data. In *Proceedings of 5th International Conference on Music Information Retrieval (ISMIR'04)*, pages 517–524, Barcelona, Spain, October 2004.

[KRY02] Tim King, George Reese, and Randy Jay Yarger. *Managing and using mysql, 2nd edition*. O'Reilly, 2002.

[Mus] MusicBrainz. Musicbrainz project. http://musicbrainz.org/.

[New09] CNN Money News. Sour note: Music sales down in 2008, 2009. http://money.cnn.com/2009/01/01/news/companies/music.reut/index.htm.

[otUoP] Information Management Systems Research Group of the University of Padua. Music information retrieval. http://ims.dei.unipd.it/websites/cms/research/music-information-retrieval.html.

[oW] University of Waikato. Weka. http://www.cs.waikato.ac.nz/ml/weka/.

[PR08] F. Pachet and P. Roy. Is hit song science a science?, 2008. Accepted to the International Symposium on Music Information Retrieval (ISMIR).

[RI] Rapid-I. Rapid miner. http://rapid-i.com/content/blogcategory/38/69/.

[SA97] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. *Future Generation Computer Systems*, 13(2–3):161–180, 1997.

[She00a] Colin Shearer. The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, pages 13–18, 2000.

[She00b] Doug Sheppard. Beginner's introduction to per, 2000. http://www.perl.com/pub/a/2000/10/begperl1.html.

[SKPW08] Markus Schedl, Peter Knees, Tim Pohle, and Gerhard Widmer. Towards an automatically generated music information system via web content mining. In *Proceedings of the 30th European Conference on Information Retrieval (ECIR'08)*, Glasgow, Scotland, 2008.

[SWPS07] Markus Schedl, Gerhard Widmer, Tim Pohle, and Klaus Seyerlehner. Web-based Detection of Music Band Members and Line-Up. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR'07)*, Vienna, Austria, September 2007.

[tfea] Wikipedia the free encyclopedia. Eclipse software. http://en.wikipedia.org/wiki/Eclipse_(software).

[tfeb] Wikipedia the free encyclopedia. Last.fm. http://en.wikipedia.org/wiki/Last.fm.

[tfec] Wikipedia the free encyclopedia. Subversion software. http://en.wikipedia.org/wiki/Subversion_(software).

[Ull06] Larry Ullman. *MySQL, Second Edition (Visual QuickStart Guide)*. Peachpit Press, Berkeley, CA, USA, 2006.

[WCO00] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl (3rd Edition)*. O'Reilly, July 2000.

Appendix A

List of artists from the study of Musicbrainz relevance

A.1 Japanese Artists

- - Mbid: e1410ddc-9745-4b5d-814d-77eb0cdfa6ad
- - Mbid: 078be324-de92-4c72-9371-65bdcf324154
- - Mbid: 4bd19b19-4ef4-42e0-ac5c-a21f709d248b
- - Mbid: 1e21cd5d-75e4-4585-9963-2d6c7d54fd79
- - Mbid: b215b03a-8e0b-48ca-b832-bbbd0db504f0
- - Mbid: 013509c5-7444-4982-88ce-e0f60c6a3fab
- - Mbid: ef8c4ade-2712-408a-bf49-aeeb59296f78
- - Mbid: 252c9f8a-be04-4146-b73e-a24696b2f5ed
- - Mbid: af3f7954-6b17-4973-9060-3c4526b2c12a
- - Mbid: 7d0780d9-d16b-4df4-9eef-71f9e23bff72

A.2 Chinese Artists

- - Mbid: b9a95482-d121-42c5-8a49-b735046e19cc
- - Mbid: d457f05e-8802-4888-83a6-b07d3df3bbf3
- - Mbid: 69528485-8804-4c14-8023-ad7c9834739f
- - Mbid: a2cab261-63cc-4ccf-8023-6b6e8588bb62
- - Mbid: a3dd7a70-697f-4747-816a-3209e2b8b97d
- - Mbid: ffd5961f-38bc-4e8f-9a7a-7e85d55a01d2
- - Mbid: 07234fa3-3edd-4658-9556-03cc1ee6eac5

- - Mbid: d780609b-eca5-4277-a0c2-801cc0404764
- - Mbid: 26be5daf-fb48-4eb0-9263-1d9b687288dd
- - Mbid: 692e367d-2846-442d-b13d-1177c3681c65

A.3 Russian Artists

- - Mbid: 1b67ca7b-3da9-4607-a5aa-93ed56034924
- - Mbid: 2619c907-4ae9-4c98-b464-31d7cb71f54f
- - Mbid: 46c83515-6df3-4afa-b597-cffd21aab719
- - Mbid: 2dce2a7b-ea6e-4adf-bb8c-e973dad5a605
- - Mbid: f7456b5b-e5a5-454e-b5db-047308210c8e
- - Mbid: d67b79c0-a026-43ee-9e5f-68141ef31018
- & Co - Mbid: bec3b214-0f07-4026-a859-07b7bbaa6ce0
- - Mbid: 97a6afa1-56c2-4088-9c3f-d0ee5dd587b5
- - Mbid: 7fc01d88-9c5a-447e-9045-6f29950cff7a
- - Mbid 1129b484-4007-491b-b646-ee3f869f53eb

A.4 Portuguese Artists

- Pólo Norte - Mbid: 22163ab3-4588-4391-b3b7-a1c329dfcc66
- Rádio Macau - Mbid: d7753fb3-5447-4ef2-bd1c-94805f57e407
- João Pedro Pais - Mbid: de3124db-87fa-4f62-8631-7c2dc7e02b11
- Paulo Gonzo - Mbid: 0bed69b8-e010-462f-8616-3dd1fa11f666
- Pedro Abrunhosa - Mbid: d80cfb78-51ee-446d-8b3d-578d50a7ce2e
- Xutos & Pontapés - Mbid: 12f2de9e-83d0-48ee-902b-c73d7a7cb6c9
- Rui Veloso - Mbid: 758416ad-4871-472a-ab82-fd6e1b7b6c67
- André Sardet - Mbid: e4a4d411-f4dd-4368-82ef-5a22ff2955a7
- Delfins - Mbid: 4e6c2aa8-4df0-4888-8ecc-e93ec474a9d7
- Mafalda Veiga - Mbid: b8d9aca9-ba8c-4822-ad2d-0b5a0e113fff

A.5 Indian Artists

- M. Balamuralikrishna - Mbid: d0e0e8cc-20e9-4909-927f-8be0ee71d094
- Dr. N. Ramani - Mbid: 74f1e36a-24c9-4929-b527-efd64d5a7208
- Lalgudi Jayaraman - Mbid: 9229f145-a8bd-442b-ae12-ca75e24ea178
- Nithyasree Mahadevan - Mbid: 243d59a5-f92b-4d51-a4a3-160827ed9a8f
- Kadri Gopalnath - Mbid: 3c785e5f-dc24-4a23-ab72-7c2931583ab2
- Raghunath Manet - Mbid: 5885e4b0-8ad0-4a8f-9b28-cac6d73b8c2e
- T.N. Krishnan - Mbid: 4d024ce6-f697-448e-be8a-c31caffdf068
- Kishori Amonkar - Mbid: d9bb0f7b-3f20-4951-8b20-3c6f5b3cab0d
- Shahid Parvez - Mbid: c2a765fd-3000-4ea8-953c-6ff87e074669
- Ram Narayan - Mbid: c94a221f-1708-4882-b035-37a1ed3d9cb4
- Budhaditya Mukherjee - Mbid: d29011df-6664-4d35-bfbf-8f704f8baf57

A.6 DJ Artists

- DJ Krush - Mbid: 38d16213-25ba-450d-8665-4e08548e62e3
- DJ Cam - Mbid: d2e123d0-c53e-4444-a52a-efeff44953c7
- DJ Vadim - Mbid: 66fce392-d89d-4eaa-a346-c50aa1021f00
- DJ Food - Mbid: 0019749d-ee29-4a5f-ab17-6bfa11deb969
- DJ QBert - Mbid: 47b0af85-4a35-4909-8ab5-7f9a5ed37a48
- DJ Spooky - Mbid: c8dbbadb-f5f1-449c-8827-43d00a73fe89
- DJ Format - Mbid: 65323ae4-1a49-497f-8f58-ae42b6c05eb6
- DJ Kentaro - Mbid: e5e9a7c1-9967-4385-b167-ff65a67779df
- DJ Yoda - Mbid: c266a7ab-7b9f-478f-a26d-91c5f8f04c7c
- DJ Z-Trip - Mbid: 03f93de6-6d62-4710-bcc7-9b3d7c3d95f5

Appendix B

Database Statistics

Database	Artists fetched	Artists processed	Crawl duration
Lastfm_Mar2009	484528	63678	10 days
Lastfm_Apr2009	1895684	439422	26 days
Lastfm_Apr2009_mb	216081	212605	5 days
Lastfm_May2009_mb	216081	212605	5 days

Table B.1: Databases Crawl Statistics