Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Identifying music and inferring similarity

Bachelor's Thesis

Tobias Schlüter

schltobi@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Samuel Welten
Prof. Dr. Roger Wattenhofer

December 24, 2012

Acknowledgements

I would like to say thank you to Samuel Welten for not only offering this interesting and challenging project but also for supporting me actively and passionately. Additionally, I would like to say thank you to all proofreaders for their useful comments as well as to the people behind coursera for providing online lectures about machine learning, neural networks and scientific writing.

Abstract

Downloading and storing a large amount of music has become easier due to faster internet connections and cheaper storage. Finding desired music in a big music collection still poses a challenge that intelligent music players try to solve: Instead of organizing music in a hierarchical folder structure music is presented in a more intuitive way using similarity relations between artists and tracks. We provide the foundation that intelligent music players can use to organize the user's music collection. We infer the similarity relations between artists using collected data about users' music taste. Our system unambiguously identifies artists and transforms the collected music taste data into an intermediate representation that we use to embed artists into an euclidean space where similar artists are nearby. In an experimental study we evaluate our embedding by comparing it with the similarity notion of LastFM. We briefly show two possibilities of visualizing artists on a map.

Keywords: Jukefox, music identification, music similarity

Contents

Chapter 1

Introduction

1.1 Motivation

More and more people have big music collections since it has become easier to download and store a large amount of music. Having a lot of music, people find it more difficult to find the music they like the most. Intelligent music players help them to find music they want to listen to by relying on the similarity relations between artists and songs. They can create playlists with songs similar to one selected song. They can display songs on a map where similar music is nearby. They can analyze what kind of music the user does not want to listen to by observing which music a user skips. They can guide the user through his music collection by providing similar artist lists. All of these features are only possible if one knows something about the similarities of artists or tracks.

1.2 Similarity

A good similarity measure is the foundation for music exploration applications that help users to find music. There are different approaches to come up with a similarity measure:

- One could analyze the sound of all songs. This is a relatively objective approach but it's not so clear how to incorporate many different aspects of a song like rhythm, harmony and dynamic in a similarity measure that feels reasonable to the user.
- A music expert could characterize each artist in a way that allows us to compute a similarity measure. This works well but is very time consuming if we have a huge amount of artists.
- One could compare the music taste of many different users to derive a similarity measure of artists.

We use the last approach to derive a similarity measure: We use collected data about the music taste of users to build an euclidean similarity space.

1.3 Goal

We want to embed artists into an euclidean space so that two artists are nearby if they are similar. Every artist is represented as a vector - we prefer a compact vector so that we can use our embedding on mobile phones that do not have much computing power. The similarity of two artists is high if the distance between the corresponding two vectors is small. The embedding algorithm has to be scalable and efficient so that the embedding can be automatically recomputed every day (we want to provide up-to-date similarity information).

Chapter 2

Data

Over the last three years, we have gathered (in an anonymous way) data about the music taste of Jukefox[1] users, which tells us which user has which music. The collected data was not in a representation we could use immediately to infer similarity relations: We collected the artist and track name but did not unambiguously identify the actual artist and track. Identifying the actual artist and track might seem trivial at the first glance but more subtle after considering the challenges that make the mapping from (noisy) names to identifiers hard. We explain a matching system that tackles these challenges and transforms the collected data to a useful intermediate representation (which we use in the next chapter to embed artists into an euclidean space).

2.1 Collected data

We have collected more than 290 million request logs over the last three years that contain information about the users' music taste. We have one request log for every track a user has. Every original request log contains the following:

- Artist **name**
- Track **name**
- Timestamp
- Hash (which is user-specific)

Using our matching system we changed this representation to the following representation, which is more useful for computing an embedding:

- Artist **id**

[1]Jukefox is an Android music player that is based on similarity relations between artists and tracks.

- Track **id**
- Timestamp
- Listener **id**

Instead of storing the artist name and the track name, we only store the corresponding **id**entifiers. Besides we only store a week number and a listener id. The resulting request log is smaller and facilitates it to compute an embedding.

2.1.1 Analysis

We analyze the request log to get some insights: Most people have popular music. Roughly 4% of all collected data (10 million request logs) are about the 10 most popular artists (Table 2.1). Figure 2.1 shows that most of the request log only contains information about popular artists. Over 80% of the request logs refer to 1% of the most popular artists. In other words, over 80% of the music that Jukefox users have refer to less than 7000 artists[2]. We do not have any request log for the majority of artists: We only have data about 43% of artists that we have in our database! These numbers are in line with research about selling revenues[3].

#	**Artist**	**Requests**
1	The Beatles	1.931.491
2	Eminem	1.245.051
3	Lil Wayne	1.021.618
4	Linkin Park	1.017.222
5	Pink Floyd	985.475
6	Metallica	976.737
7	Red Hot Chili Peppers	856.931
8	Queen	753.991
9	Michael Jackson	739.279
10	AC/DC	705.955

Table 2.1: TOP 10. This table shows the 10 most popular artists of the request log.

Our collected data belongs to the category of implicit feedback data: We implicitly assume that a user likes an artist if he has the artist. Implicit feedback provides us only with positive feedback and not with negative feedback, i.e. we only know that a user might like an artist but we don't know whether he dislikes an artist. Using implicit data our embedding cannot reach the quality that we would get if we would have explicit data.

[2]Our database has over 680.000 artists.
[3]http://www.guardian.co.uk/music/2008/dec/23/music-sell-sales

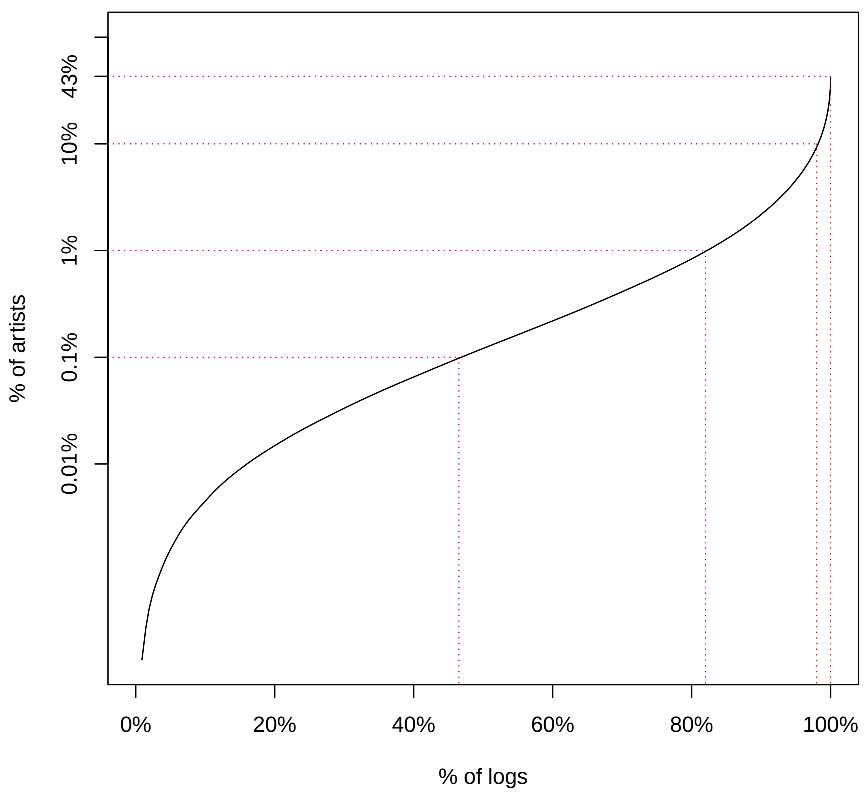

Figure 2.1: Request log distribution (Artists are ordered by popularity). This figure shows how many request logs are due to the first 0.1%/1%/10%/43% of the most popular artists. Over 80% of the request logs give information about 1% of the most popular artists.

2.2 Identifying music

There are two different ways of identifying songs:

- A song can be identified using its sound: Assuming that we have a database that contains the sounds of songs[4], we can compare a given song with all known songs to identify the song.
- One can use metadata (commonly stored in ID3 tags) about a song to identify it.

We only use the last approach to identify songs, i.e. we use (noisy) metadata containing the artist, album and track[5] name to identify the song.

2.2.1 Challenges

It is far from trivial to match nosiy metadata belonging to a musical entity[6] to the correct identifier representing the intended entity. The following issues make a correct matching of requests to musical entities hard:

- Ambiguity: Different artists may have the same name, e.g. "John Williams" might refer to the popular soundtrack composer or to a classical guitar player or to a acoustic blues guitarist or to a jazz saxophonist. More than 10 different musicians with the name "John Williams" are listed on the MusicBrainz[7] website[8].
- Aliases: Some users use an alias to refer to an artist, e.g. people use the alias "Fab Four" to refer to "The Beatles".
- Noise: The users' requests are very noisy, e.g. over 90 different names in our original request log refer to "The Beatles". People use unnecessary symbols like "?,-_.[0-9];" or extend the artist name with additional information (like a particular year).
- Misspelling: Some users misspell the artist name, e.g. people use "The Beetles" although they want to refer to "The Beatles".

The ambiguity issue is not easy to solve: The famous music website Last.fm doesn't disambiguate different artists with the same name according to their

[4]In practice, one stores only fingerprints of the sounds of songs in the database.
[5]We use the term track to refer to a song of a particular album.
[6]We use the term musical entity to refer to an artist, album or track.
[7]MusicBrainz is an Open Music Encyclopedia.
[8]http://www.musicbrainz.org

FAQs[9]. Contrary, they even maintain different profiles for the same artist. According to their blog they work on improving disambiguation[10] (they cooperate with MusicBrainz). We tackle these challenges as follows:

- We use the MusicBrainz database[11] to disambiguate musical entities. MusicBrainz provides an "unambiguous form of music identification" (as stated on the MusicBrainz website).
- We develop a system that allows us to match musical entities properly.

[9]http://www.last.fm/help/faq?category=97

[10]http://blog.last.fm/2011/11/24/the-brainz-are-back-in-town

[11]We denormalize the MusicBrainz database for simplicity and performance.

2.3 Matching system

Our matching system converts the original request log to a more compact representation (Figure 2.2): Every original request log (containing the artist and track name) is transformed to a new representation of the request log (which just contains ids and the week).

id	artist_name	track_name	hash	timestamp
1	The Beatles	Héy-_Jude (track-1) _.?	...	...
2	Beatles Los	Submarine Yellow	...	...
3	**John Williams**	Selections From "West Side Story"	...	...
4	**John Williams**	Prelude to a Song	...	...

id	artist_id	album_id	recording_id	listener_id	week
1	303	996439	2803068	10	2104
2	303	953650	4260494	11	2104
3	**94**	1129941	13316089	12	2104
4	**238569**	1062097	12614706	13	2104

Figure 2.2: Matching from names to unique ids. We find the best possible match of every music track using the MusicBrainz data. Our matching system ignores some naming inconsistencies:

- Accents and special characters like _,- are ignored. Additional information like (1960) is ignored in some cases. [**id = 1**]
- We ignore the order of the words if we cannot match the names directly. [**id = 2**]
- Some artists have the same name: John Williams might refer to the famous music composer as well as to a guitar player. Our system is able to disambiguate these two different artists. [**id = 3,4**]

While designing the matching system we focused on two aspects: Quality and performance. The former helps us to compute a better embedding (a better matching results in more accurate input data, which improves the quality of our embedding). Using our matching system we were able to identify over 85% of the music that Jukefox users have. The latter is important because the number of requests that we get in one month increases (Figure 2.3). Nowadays we get nearly 20 million requests in one month.

The performance of our matching system was crucial in order to match over 290 million accumulated original request logs to corresponding identifiers. Using a Postgres[12] database index as well as an Lucene[13] index our matching system was not able to match more than 100 tracks per second to corresponding artist

[12]Postgres is an object-relational database management system.
[13]Lucene is a text search engine library.

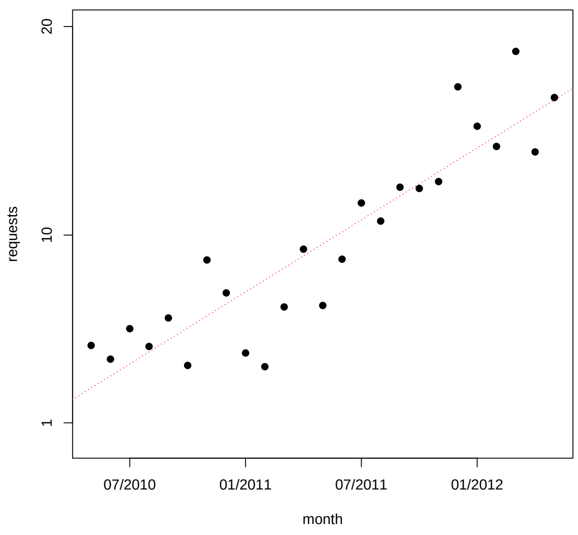

Figure 2.3: Requests in million per month. This figure shows how the requests per month, which the Jukefox server gets, increase over time. In 2010, we only had a few million requests per month whereas nowadays we nearly have 20 million requests per month.

and track ids. Therefore we would have needed more than one month to match the whole original request log. We were able to boost the performance to 1000 tracks per second by developing a self-made index using the key-value store Redis.

2.3.1 Matching names to ids

We explain how we match a user input referring to an artist to an artist id by giving an example: We show how our system indexes the artist “Wolfgang Amadeus Mozart” and how a user request for “Wolfgang Amadeus Mo**k**art (composer)” is matched to the intended artist (id). Before we explain indexing and searching we explain the data structure that we use for the index and the tokenizer needed for indexing and searching.

Data structure We use a (special) set to index musical entities that can be retrieved by key in O(1) (via hashing) and is sorted by score (this data structure is provided by the key value store Redis[14]). Each element in the set is associated with a score. The element with the highest score is at the beginning whereas the element with the smallest score is at the end.

[14]Redis is a fast in-memory key value data store.

Tokenizer Our tokenizer turns a string input into a set of strings consisting of:

- the exact input
- the normalized input
- the normalized input where we removed any information about word order
- the two longest normalized words sorted alphabetically

We refer with normalized input to the input where we removed characters that are likely to be not important for matching names to ids (e.g. whitespace, underscore, characters in parenthesis, ...).

Indexing We show how our system indexes the artist "Wolfgang Amadeus Mozart". We use the tokenizer to generate four keys:

1. Wolfgang Amadeus Mozart
2. wolfgangamadeusmozart
3. **a**madeus**m**ozart**w**olfgang
4. wolfgangamadeus

We add the id of the artist "Wolfgang Amadeus Mozart" to each sorted set that can be retrieved in O(1) via hashing with one of the above four keys.

Searching We show how our system handles the user query for "Wolfgang_Amadeus-Mo**k**art,**(composer)** 1791". We use the tokenizer to generate four keys:

1. Wolfgang_Amadeus-Mokart,(composer) 1791
2. wolfgangamadeusmokart
3. **a**madeus**m**okart**w**olfgang
4. wolfgangamadeus

We take a (special) union of all sorted sets that can be retrieved with the above four keys. The scores of artist ids that are in multiple sets are summed up.[15] In other words, we prefer the artist id that matches the user query best.

Because of a spelling error in the query "Wolfgang Amadeus Mo**k**art **(composer)**" the first three keys lead to three empty sorted sets. The last key leads to a non-empty sorted set that contains an artist id for "Wolfgang Amadeus Mozart". Our matching algorithm returns the found artist id.

[15]Redis provides this (special) union with the command zunionstore.

2.3.2 Remarks

Ambiguity Some artist names refer to multiple artists: "John Williams" can refer to more than ten different musicians. We use additional information like track title (if given by the user) or artist popularity to find the most probable artist.

Aliases We use aliases so that every artist can not only be found by the proper name but also by an alias: "Wolfgang Amadeus Mozart" can be found by its proper name or by one of more than 30 aliases (e.g. W. A. Mozart, W.A. Mozart, Volfgangs Amadejs Mocarts).

Matching tracks Tracks (as well as albums) can be matched in a similar fashion. First, we search for all possible artist ids, then for all possible album ids and finally for all possible track ids. We discard all ids that definitely are not meaningful and return the most probable track id.

Chapter 3

Embedding

We describe different ways of embedding high dimensional data into a lower dimensional space: We briefly present some popular approaches to reduce the dimensionality of our collected data. We preprocess our collected data and use the PCA to embed artists into an euclidean space. Finally, we show some possibilities to compute the similarity between two artists.

3.1 Survey of dimensionality reduction techniques

Obtaining a dimensionality reduction is actively researched. In recent years, many different approaches were proposed that embed high dimensional data into a lower dimensional space. In the following we describe the well-known PCA as well as other more recent proposals.

PCA The Principal Component Analysis (PCA) reduces the dimensionality of the high-dimensional input by projecting the data on the directions with most variance. We assume that the directions with most variance contain the most important information and that our input data lies on a linear manifold. We can obtain the PCA by using the SVD (Singular Value Decomposition).

Weighted-ALS Hu et al. describe in [1] how one can use a weighted Alternating Least Squares approach that is very similar to the above SVD approach but uses the input data in a more efficient way by introducing the notion of confidence, i.e. we use information about how confident we are that a user has a specific artist instead of discarding the confidence like in our SVD approach (which we will describe later).

LLE Locally Linear Embedding (LLE) is an “unsupervised learning algorithm that computes low-dimensional, neighborhood-preserving embeddings of high-dimensional inputs.” [2]. Every data point is reconstructed using k neighbors.

LLE scales quadratically in the number of artists (which we want to embed) if we use the common way of implementing this algorithm [3].

t-SNE t-SNE is an unsupervised dimensionality reduction techniques that is suitable for reducing the dimensionality to only a few dimensions [4, 5]. Therefore this technique is well suited to visualize artists on a map. Using t-sne we not only put similar artists but also similar clusters next to each other.

Neural network A non-linear generalization of the PCA can be computed by using a neural network [6] that can provide better results than the classical PCA (which we will use).

3.2 Our embedding using SVD

Many different methods reduce the dimensionality of our high-dimensional input data. We decided to use the SVD to obtain a PCA of our data because of the following reasons:

- Having an embedding of artists we can easily reduce the dimensionality of artists further by omitting dimensions of low variance. The client that uses our embedding can decide how many dimensions he wants to use.
- We can use the two most important dimensions to obtain a visualization of artists.
- We only have one parameter (the dimensionality d) that can be easily determined. We don't have to change parameters if we get more[1] data.
- The Netflix Prize[2] showed that matrix factorization models like the SVD (as well as Restricted Boltzmann machines) provide good[3] results in predicting ratings.
- The SVD is scalable and can be computed in a short amount of time.

We preprocess our collected data and embed artists into an euclidean space (so that similar artists are nearby).

[1]Other (complicated) methods require that you change some parameters of the model if you get more data.

[2]The Netflix Prize challenged the scientific community to find the best possible Collaborative Filtering methods.

[3]The winning team of the Netflix challenge used matrix factorization techniques in their final solution.

3.2.1 Preprocessing

We process our matched request log to obtain a matrix C where C_{ij} indicates how many tracks of artist j a user i has. Research about implicit[4] feedback data [1] shows that one can improve the quality of the embedding by not directly applying the SVD to C but to a sparse matrix X where $X_{ij} = 1$ if user i has artist j. We remove all columns corresponding to artists that are owned by less than 10 users so that we only provide similarity information about artists if we have enough data. Then, we normalize every column and obtain $\bar{X}$. Without normalization every popular artist would have a big distance in the final embedding to his similar artists (which would not make sense).

3.2.2 Embedding

We embed the preprocessed data $\bar{X}$ into an euclidean space. We apply[5] the SVD to the matrix $\bar{X}$ so that we get the following matrix decomposition that gives us a lower dimensional representation of artists:

$$\bar{X} = USV^T$$

Artist j is described by the row $V_{j,:} \in \mathbb{R}^d$ (and user i is described by row $U_{i,:} \in \mathbb{R}^d$). We refer to $V_{j,:}$ as the coordinates of artist j, which we use to compute the similarity between artists. We will determine the optimal number of latent factors d in the experimental study by using the singular values, which are stored in the diagonal of the matrix S.

3.3 Computing similarity

Different measures allow us to compute the distance D or similarity S between two artists j and k.

1. Euclidean distance: $D_{jk} = \sqrt{\sum_{l=1}^{d} (V_{j,l} - V_{k,l})^2}$
2. Manhattan distance: $D_{jk} = \sum_{l=1}^{d} |V_{j,l} - V_{k,l}|$
3. Cosine similarity: $S_{jk} = \frac{V_{j,:} V_{k,:}^T}{\|V_{j,:}\| \|V_{k,:}\|} \in [-1, 1]$

[4]Our request logs are implicit data since we implicitly assume that a user likes an artist if he has the artist.

[5]We used the svdlibc implementation written by Doug Rohde.

Chapter 4

Experimental study

We measure how much time we need to obtain our embedding, we show how we choose the number of latent factors (of each artist), and we evaluate the quality of our embedding.

4.1 Benchmark

We load over 250 million matched request logs from the database and preprocess them to obtain the binary matrix $\bar{X}$, which describes 100.000 artists characterized by the music taste of over 380.000 users. Finally, we compute the embedding. The whole process takes less than half an hour (Table 4.1).

What	**Time in minutes**
Loading from DB	11
Preprocessing	5
SVD	8
Total	**24**

Table 4.1: Overall time. This table shows the overall time we need to compute the embedding.

4.2 Model order selection

We determine the number of latent factors d of our embedding. The lower we choose d the simpler and computationally more efficient our model. Since we want to use artist coordinates on mobile phones, we prefer a small number of latent factors even if the quality of our embedding degrees slightly.

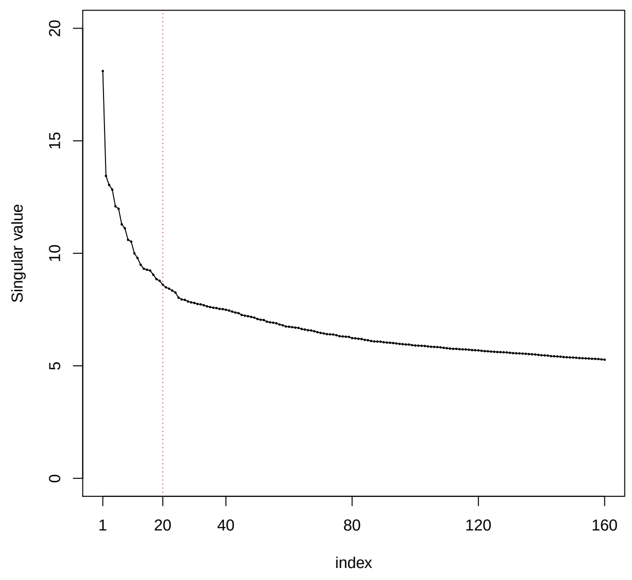

Figure 4.1: Magnitude of the singular values. Using 20 latent factors for each artist we preserve a good amount of similarity information.

Figure 4.1 shows the plot of the magnitudes of the first 160 singular values. Latent factors that correspond to singular values with a high magnitude usually convey the most useful information to distinguish artists. $d = 20$ seems to provide a compromise between space/time requirements and the quality of our embedding.

4.3 Evaluation

We give you some insight about the embedding and compare it with LastFM as well as with a similarity measure that directly uses the high-dimensional input data.

Insight We want to give you an opportunity to inspect the distances of some selected artists. Table 4.2 shows the euclidean distances of the 5 most popular artists followed by some selected artists that are similar in the embedding. High distances indicate no similarity while low distances indicate similarity.

	The Beatles	Eminem	Lil Wayne	Linkin Park	Pink Floyd	Beethoven	Mozart	John Williams	Clint Mansell	Hans Zimmer
The Beatles		0.42	0.37	0.36	0.15	0.19	0.2	0.2	0.21	0.21
Eminem	0.42		0.4	0.4	0.43	0.42	0.42	0.43	0.42	0.42
Lil Wayne	0.37	0.4		0.44	0.35	0.32	0.32	0.32	0.32	0.32
Linkin Park	0.36	0.4	0.44		0.36	0.3	0.31	0.3	0.3	0.29
Pink Floyd	0.15	0.43	0.35	0.36		0.17	0.17	0.17	0.18	0.18
Beethoven	0.19	0.42	0.32	0.3	0.17		**0.02**	**0.02**	**0.03**	**0.02**
Mozart	0.2	0.42	0.32	0.31	0.17	**0.02**		**0.01**	**0.02**	**0.02**
John Williams	0.2	0.43	0.32	0.3	0.17	**0.02**	**0.01**		**0.02**	**0.02**
Clint Mansell	0.21	0.42	0.32	0.3	0.18	**0.03**	**0.02**	**0.02**		**0.03**
Hans Zimmer	0.21	0.42	0.32	0.29	0.18	**0.02**	**0.02**	**0.02**	**0.03**	

Table 4.2: Euclidean distances of TOP 5 artists followed by 5 other artists that are similar to each other ($d = 20$). This table illustrates how one can compare artists using the euclidean distance. Small distances, which are highlighted, indicate similarity.

4.3.1 Comparison with LastFM

We compare our latent factor model (SVD) with the similarity notion of LastFM. We use the term "similar artist list of artist A" to refer to the list of the 100 most similar artists of artist A. We select 50 famous artists arbitrarily[1] and compare for each of these artists the "similar artist list" of LastFM with the "similar artist list" using our embedding: The Tables 4.3, 4.4 and 4.5 show for each artist the agreement in % of the "similar artist list" of LastFM with the "similar artist list" of our embedding depending on the number of latent factors d. We measure the agreement with LastFM as follows:

$$\text{agreement} = \frac{\text{number of artists in both lists}}{100} \in [0, 100\%]$$

Observations The euclidean distance provides the highest agreement with LastFM using the prior chosen parameter $d = 20$. Therefore we recommend to use the euclidean distance if one chooses to use 20 latent factors.

More common artists might be explainable with fewer dimensions than less common artists: The "similar artist lists" of Bryan Adams as well as the Rolling Stones already have a considerable agreement with LastFM using only one or two dimensions, whereas the "similar artist lists" of Ludwig van Beethoven have only a considerable agreement with much more dimensions.

Comparing the cosine similarity (Table 4.5) with the euclidean distance (Table 4.3) we notice that if we use more latent factors (i.e. if we increase d) we might benefit from using the cosine similarity instead of the euclidean distance.

Comparing the manhattan distance (Table 4.4) with the euclidean distance (Table 4.3) we notice that the difference between these too measures is negligible small.

4.3.2 Comparison using the jaccard similarity

We (also) compare our embedding with the jaccard similarity, which we can compute by directly using the high-dimensional binary input matrix X ($X_{ij} = 1$ if user i has artist j). The jaccard similarity coefficient allows us to measure the similarity between artists j and k. Let u_j/u_k refer to the set of indices corresponding to users that have artist j/k. Then the jaccard index can be computed as follows:

$$S_{jk} = \frac{u_j \cap u_k}{u_j \cup u_k} \in [0, 1]$$

[1]We did not choose to select random artists because typical users request popular artists and not random artists.

Observations Comparing the euclidean distance (Table 4.3) with the jaccard index (table 4.6) we notice that our latent factor model with 20 factors only provides a mean agreement of 24.76%, while the jaccard index provides a mean agreement of 33.56%. This implicates that our embedding does not preserve the high-dimensional input perfectly.

Figure 4.2 shows the mean agreement with LastFM depending on the dimensionality d and the chosen similarity measure. Our embedding has a lower mean agreement in comparison to the mean agreement we get by using the jaccard index directly on the high-dimensional input X.

d / **Artist**	**1**	**2**	**5**	**10**	**20**	**30**	**40**	**80**	**120**	**160**
Ludwig van Beethoven	**0**	**0**	3	5	15	19	20	28	35	42
Wolfgang Amadeus Mozart	1	2	5	9	21	23	25	30	38	42
Franz Schubert	0	0	13	24	29	32	34	37	48	51
John Williams	1	0	2	8	12	14	15	15	14	14
Clint Mansell	0	1	0	4	12	17	23	33	43	44
Hans Zimmer	2	0	3	7	13	13	17	15	16	20
Danny Elfman	0	1	0	6	14	15	18	17	21	27
Clueso	1	3	10	14	23	23	24	30	32	34
Yann Tiersen	2	0	1	5	13	14	14	12	10	9
Bon Jovi	13	24	25	34	40	47	52	42	10	11
Blues Brothers	1	1	1	2	3	5	4	5	5	5
Fats Domino	0	8	14	23	35	35	37	39	41	42
Bryan Adams	**11**	**32**	29	41	50	43	46	45	29	17
Mariah Carey	16	23	30	42	51	47	37	39	27	7
Billy Joel	6	19	29	46	56	55	50	47	46	39
Elton John	9	28	36	47	49	43	39	41	41	40
Atomic Kitten	4	1	5	6	15	26	32	37	37	37
John Denver	4	11	11	35	39	42	46	41	38	37
The Offspring	10	10	40	44	37	36	42	36	37	30
Jim Ladd	0	0	0	0	0	0	0	0	0	0
Wise Guys	1	0	3	9	9	12	17	18	16	15
Die Firma	1	1	3	9	11	13	18	25	24	28
Sportfreunde Stiller	1	6	15	24	26	27	29	33	36	37
Wir sind Helden	1	1	8	17	20	25	25	32	33	35
Christina Stürmer	2	0	17	32	42	35	33	36	42	34
Die Ärzte	1	2	12	14	17	20	27	28	38	40
Paul Kalkbrenner	0	0	1	1	3	4	6	14	14	14
Erste Allgemeine Verunsicherung	1	2	7	11	11	11	12	14	11	11
Curse	1	0	1	6	13	15	25	26	29	29
Massive Attack	1	4	8	18	26	32	33	38	41	41
Portishead	1	2	10	20	34	31	36	37	32	32
Ladytron	0	0	8	12	13	19	18	22	20	19
The Rolling Stones	**16**	**33**	42	52	58	54	54	28	24	18
Bob Dylan	4	14	26	41	41	42	41	39	31	34
ABBA	8	14	36	33	29	27	26	27	30	28
John Lennon	4	16	19	33	31	23	23	16	7	5
Little Richard	4	7	17	30	34	36	36	36	38	39
Jackie Wilson	3	7	12	20	37	44	44	41	44	44
Chuck Berry	3	12	15	25	35	30	30	29	29	24
Ray Charles	6	9	16	32	34	37	37	34	36	39
B.B. King	3	2	6	16	9	13	17	26	30	34
Claude Bolling	0	1	5	5	4	4	4	8	9	8
Emerson String Quartet	0	0	0	0	7	13	8	11	14	13
The Los Angeles Guitar Quartet	0	0	0	0	3	1	0	0	0	1
Tom Astor	1	1	11	20	22	21	22	32	31	32
Johnny Cash	5	13	14	27	30	35	32	34	38	39
Thelonious Monk	0	2	15	26	27	28	28	38	43	47
Django Reinhardt	0	0	13	20	26	26	25	35	36	35
The Dave Brubeck Quartet	0	2	7	20	28	33	32	39	47	49
Duke Ellington	1	5	20	25	31	33	32	33	36	38
Mean	3.0	6.4	12.48	20.0	**24.76**	25.86	26.9	28.36	28.54	28.2
Median	1.0	2.0	10.5	20.0	**26.0**	26.0	26.5	32.0	31.5	33.0
Max	16	33	42	52	58	55	54	47	48	51

Table 4.3: Agreement in % using the **euclidean distance**. Each entry in this table indicates how many artists are both in our similar artist list as well as in the LastFM similar artist list (given the artist and the number of latent factors *d*).

d	**1**	**2**	**5**	**10**	**20**	**30**	**40**	**80**	**120**	**160**
Artist										
Ludwig van Beethoven	**0**	**0**	2	4	14	15	17	25	33	40
Wolfgang Amadeus Mozart	1	2	4	6	19	21	24	28	34	40
Franz Schubert	0	0	11	22	25	28	33	35	46	50
John Williams	1	0	2	10	12	12	12	11	14	14
Clint Mansell	0	1	1	5	13	15	27	33	41	42
Hans Zimmer	2	0	3	7	9	12	16	17	15	20
Danny Elfman	0	0	0	3	11	12	17	17	20	24
Clueso	1	3	10	15	22	23	24	32	35	38
Yann Tiersen	2	0	1	4	12	13	12	9	10	9
Bon Jovi	13	23	22	32	45	45	47	44	31	28
Blues Brothers	1	1	1	3	3	6	4	6	6	5
Fats Domino	0	9	12	22	34	36	35	41	44	43
Bryan Adams	**11**	**30**	29	34	53	43	47	43	24	17
Mariah Carey	16	17	27	44	52	47	38	36	33	21
Billy Joel	6	18	29	40	55	59	51	44	42	40
Elton John	9	25	32	48	51	45	37	39	45	45
Atomic Kitten	4	1	5	5	14	25	30	38	37	40
John Denver	4	11	12	25	29	44	44	42	37	34
The Offspring	10	9	40	47	45	36	40	36	36	36
Jim Ladd	0	0	0	0	0	0	0	0	0	0
Wise Guys	1	0	4	8	11	13	16	16	16	16
Die Firma	1	1	6	10	12	13	19	26	25	28
Sportfreunde Stiller	1	5	15	22	24	26	30	34	37	35
Wir sind Helden	1	1	8	14	20	25	27	32	33	36
Christina Stürmer	2	0	16	30	44	37	30	39	41	37
Die Ärzte	1	2	10	14	17	21	26	34	36	39
Paul Kalkbrenner	0	0	1	1	3	3	3	13	12	12
Erste Allgemeine Verunsicherung	1	2	7	11	11	11	12	14	11	10
Curse	1	0	2	6	14	16	24	26	29	29
Massive Attack	1	5	8	16	28	29	35	38	37	38
Portishead	1	1	9	19	31	31	34	35	31	31
Ladytron	0	0	8	11	15	17	17	20	19	19
The Rolling Stones	**16**	**33**	44	54	57	45	48	28	24	20
Bob Dylan	4	12	26	40	36	41	42	41	29	29
ABBA	8	13	35	31	30	32	30	31	30	29
John Lennon	4	14	19	33	30	16	18	12	8	4
Little Richard	4	5	16	27	34	36	36	36	36	37
Jackie Wilson	3	4	12	22	35	42	43	41	44	45
Chuck Berry	3	11	15	25	35	30	30	29	29	24
Ray Charles	6	9	16	33	36	37	37	40	37	37
B.B. King	3	1	4	15	9	13	17	25	27	33
Claude Bolling	0	1	5	5	5	3	3	9	9	9
Emerson String Quartet	0	0	0	0	5	12	8	12	14	14
The Los Angeles Guitar Quartet	0	0	0	0	2	1	1	0	0	1
Tom Astor	1	1	11	20	21	23	24	30	30	32
Johnny Cash	5	13	16	25	28	32	31	37	39	40
Thelonious Monk	0	2	9	17	24	28	27	33	44	45
Django Reinhardt	0	0	9	14	21	24	27	35	36	34
The Dave Brubeck Quartet	0	2	7	18	26	28	28	37	42	49
Duke Ellington	1	5	17	24	27	29	33	36	36	37
Mean	3.0	5.86	11.96	18.82	**24.18**	25.02	26.22	28.3	28.48	28.7
Median	1.0	2.0	9.0	16.5	**23.0**	25.0	27.0	32.5	32.0	32.5
Max	16	33	44	54	57	59	51	44	46	50

Table 4.4: Agreement in % using the **manhattan distance**. Each entry in this table indicates how many artists are both in our similar artist list as well as in the LastFM similar artist list (given the artist and the number of latent factors *d*).

d / **Artist**	**1**	**2**	**5**	**10**	**20**	**30**	**40**	**80**	**120**	**160**
Ludwig van Beethoven	**0**	**0**	0	0	3	8	13	28	37	45
Wolfgang Amadeus Mozart	0	0	0	0	4	8	12	26	35	41
Franz Schubert	0	0	0	0	5	10	14	29	43	51
John Williams	0	0	0	0	4	6	7	12	14	14
Clint Mansell	0	0	0	2	5	4	11	16	22	23
Hans Zimmer	0	0	0	0	4	7	11	13	14	17
Danny Elfman	0	0	0	0	3	3	5	13	15	21
Clueso	0	0	7	18	21	22	25	31	34	37
Yann Tiersen	1	3	2	4	6	9	11	19	19	15
Bon Jovi	11	21	22	31	38	43	53	44	23	21
Blues Brothers	1	3	3	4	6	7	9	9	9	8
Fats Domino	0	1	2	6	26	32	32	35	39	44
Bryan Adams	**10**	**30**	34	42	50	51	54	59	55	39
Mariah Carey	15	18	26	45	53	51	44	44	26	22
Billy Joel	3	17	23	37	43	46	45	49	50	49
Elton John	8	28	35	47	41	38	36	45	53	51
Atomic Kitten	0	1	6	12	17	23	25	36	41	44
John Denver	0	5	7	24	28	36	47	55	56	56
The Offspring	13	13	39	49	52	43	48	47	49	37
Jim Ladd	0	0	0	0	0	0	0	0	0	0
Wise Guys	0	0	10	23	26	25	26	31	32	32
Die Firma	0	1	2	7	8	9	13	18	20	24
Sportfreunde Stiller	0	0	9	21	24	22	24	33	35	35
Wir sind Helden	0	0	8	22	26	24	27	33	38	39
Christina Stürmer	0	0	9	25	34	26	25	35	36	36
Die Ärzte	0	0	4	12	18	18	24	29	36	35
Paul Kalkbrenner	0	0	0	3	3	3	3	8	9	13
Erste Allgemeine Verunsicherung	0	0	1	6	10	12	13	13	13	12
Curse	0	0	1	8	10	11	14	20	23	27
Massive Attack	2	5	6	20	29	36	39	44	49	48
Portishead	2	5	9	21	31	39	44	47	42	47
Ladytron	0	2	2	10	14	11	11	16	21	21
The Rolling Stones	**13**	**32**	40	55	59	55	59	52	42	32
Bob Dylan	7	18	21	37	40	43	45	48	45	44
ABBA	3	14	33	39	40	41	39	41	41	39
John Lennon	11	23	24	39	44	38	41	42	36	32
Little Richard	1	2	8	16	38	39	42	45	44	43
Jackie Wilson	0	0	5	7	37	42	41	40	40	44
Chuck Berry	5	15	19	29	46	42	38	39	42	39
Ray Charles	3	8	16	29	43	45	47	49	50	52
B.B. King	3	7	9	16	16	17	19	18	20	30
Claude Bolling	0	0	0	0	0	1	1	3	3	4
Emerson String Quartet	0	0	0	0	2	2	2	2	4	4
The Los Angeles Guitar Quartet	0	0	0	0	0	0	0	0	0	0
Tom Astor	0	0	0	9	20	30	37	48	55	55
Johnny Cash	3	13	13	24	26	30	31	40	41	48
Thelonious Monk	0	0	0	1	2	12	14	24	32	40
Django Reinhardt	0	0	0	3	7	15	17	24	31	37
The Dave Brubeck Quartet	0	0	0	1	1	7	9	23	28	39
Duke Ellington	0	0	0	2	4	10	19	25	28	39
Mean	2.3	5.7	9.1	16.12	**21.34**	23.04	25.32	30.0	31.4	32.5
Median	0.0	0.0	4.5	11.0	**19.0**	22.0	24.5	31.0	35.0	37.0
Max	15	32	40	55	59	55	59	59	56	56

Table 4.5: Agreement in % using the **cosine similarity**. Each entry in this table indicates how many artists are both in our similar artist list as well as in the LastFM similar artist list (given the artist and the number of latent factors d).

Artist	**Agreement**
Ludwig van Beethoven	30
Wolfgang Amadeus Mozart	38
Franz Schubert	57
John Williams	12
Clint Mansell	35
Hans Zimmer	18
Danny Elfman	20
Clueso	33
Yann Tiersen	19
Bon Jovi	29
Blues Brothers	8
Fats Domino	40
Bryan Adams	45
Mariah Carey	35
Billy Joel	32
Elton John	39
Atomic Kitten	29
John Denver	40
The Offspring	38
Jim Ladd	0
Wise Guys	49
Die Firma	34
Sportfreunde Stiller	32
Wir sind Helden	39
Christina Stürmer	46
Die Ärzte	21
Paul Kalkbrenner	10
Erste Allgemeine Verunsicherung	17
Curse	47
Massive Attack	29
Portishead	40
Ladytron	31
The Rolling Stones	41
Bob Dylan	35
ABBA	32
John Lennon	35
Little Richard	45
Jackie Wilson	41
Chuck Berry	44
Ray Charles	39
B.B. King	23
Claude Bolling	16
Emerson String Quartet	19
The Los Angeles Guitar Quartet	9
Tom Astor	69
Johnny Cash	23
Thelonious Monk	56
Django Reinhardt	60
The Dave Brubeck Quartet	47
Duke Ellington	52
Mean	**33.56**
Median	**35.0**
Max	69

Table 4.6: Agreement in % using the **jaccard index**. Each entry in this table indicates how many artists are both in our similar artist list as well as in the LastFM similar artist list.

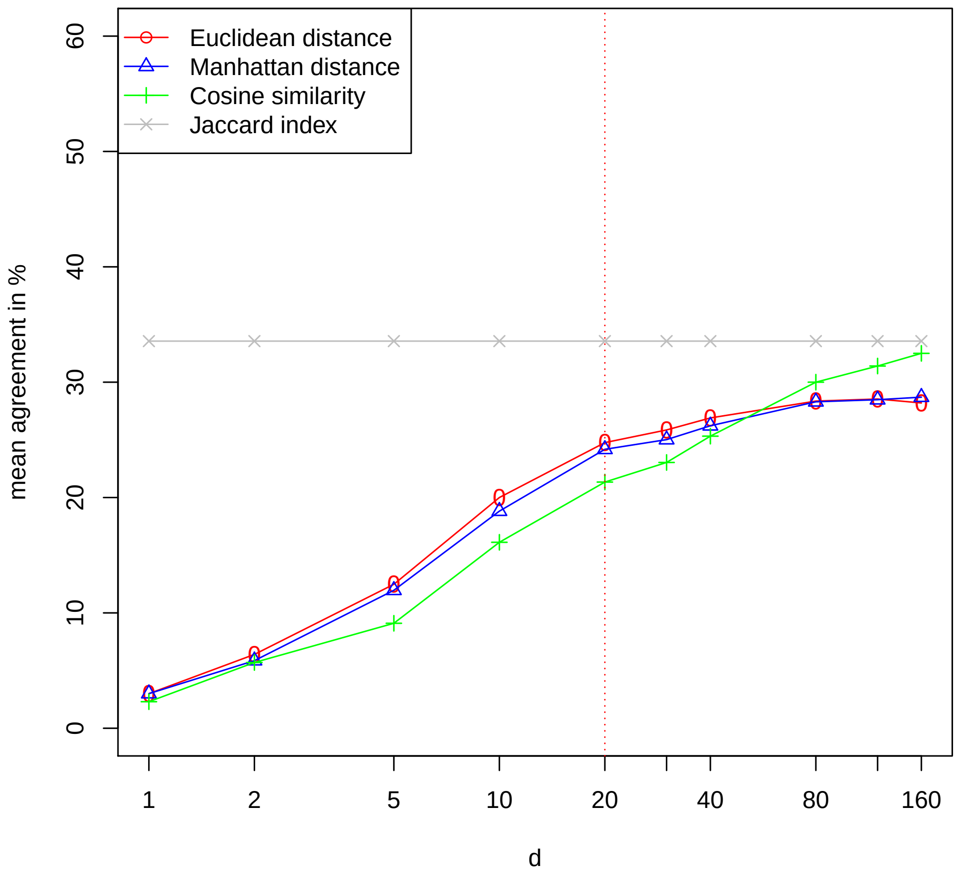

Figure 4.2: Mean agreement in % with LastFM depending on the similarity measure and the dimensionality *d*. Using our embedding with euclidean distance, manhattan distance or cosine similarity we get a lower mean agreement in comparison to using the jaccard index directly on the high-dimensional input data.

Chapter 5

Visualization

We visualize artists with two different approaches:

- We use the first two dimensions of our embedding directly.
- We reduce the dimensionality of the embedding to two dimensions using a nonlinear dimensionality reduction technique (LLE).

We connect every artist with a dashed line to his most similar artist so that we can visually assess the quality of the artist map.

5.1 PCA map

We plot the first two dimensions of our embedding that correspond to the directions of most variance. We assume that these two dimensions are the most informative dimensions of our embedding. Figure 5.1 shows the PCA map for the 30 most popular artists which gives us a global perspective of these artists. Most of the dashed lines are relatively large so that exploring the neighborhood of an artist might not be sensible.

5.2 LLE map

We use Locally Linear Embedding (LLE) to reduce the dimensionality of our embedding from 20 to 2 (described in Section 3.1). Figure 5.2 and 5.3 show us a visualization for the 30 most popular artists using 3 and 6 neighbors. Regarding the neighborhood of a chosen artist, these two maps look more reasonable than the PCA map: The dashed lines indicating the most similar artist are relatively short.

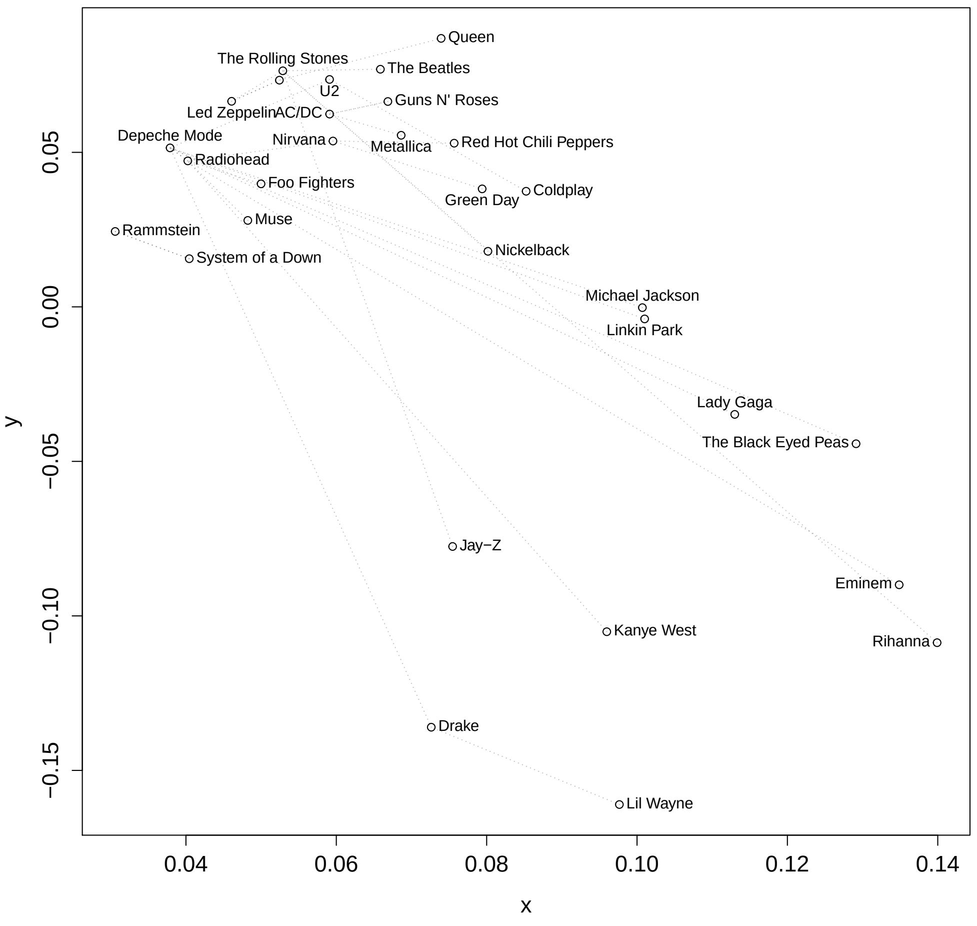

Figure 5.1: PCA Map. This figure shows the first two dimensions of TOP30 artists.

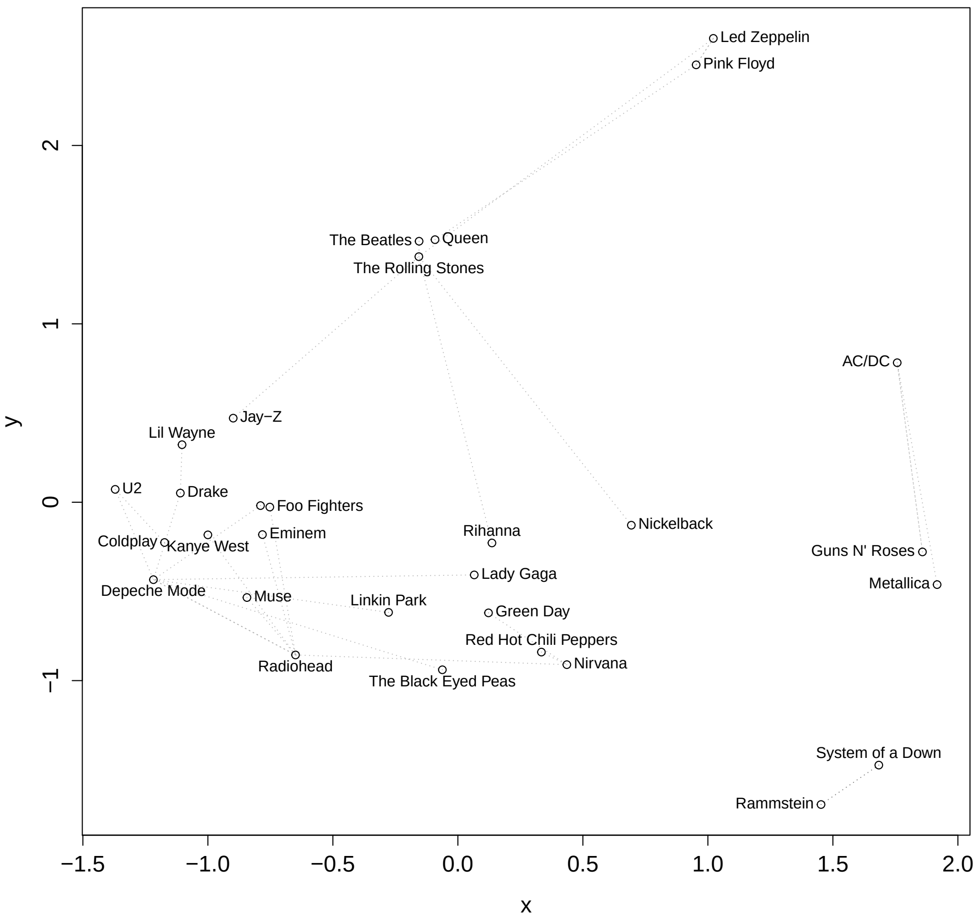

Figure 5.2: LLE Map ($k = 3$). This figure shows a nonlinear dimensionality reduction of TOP30 artists using $k = 3$ neighbors.

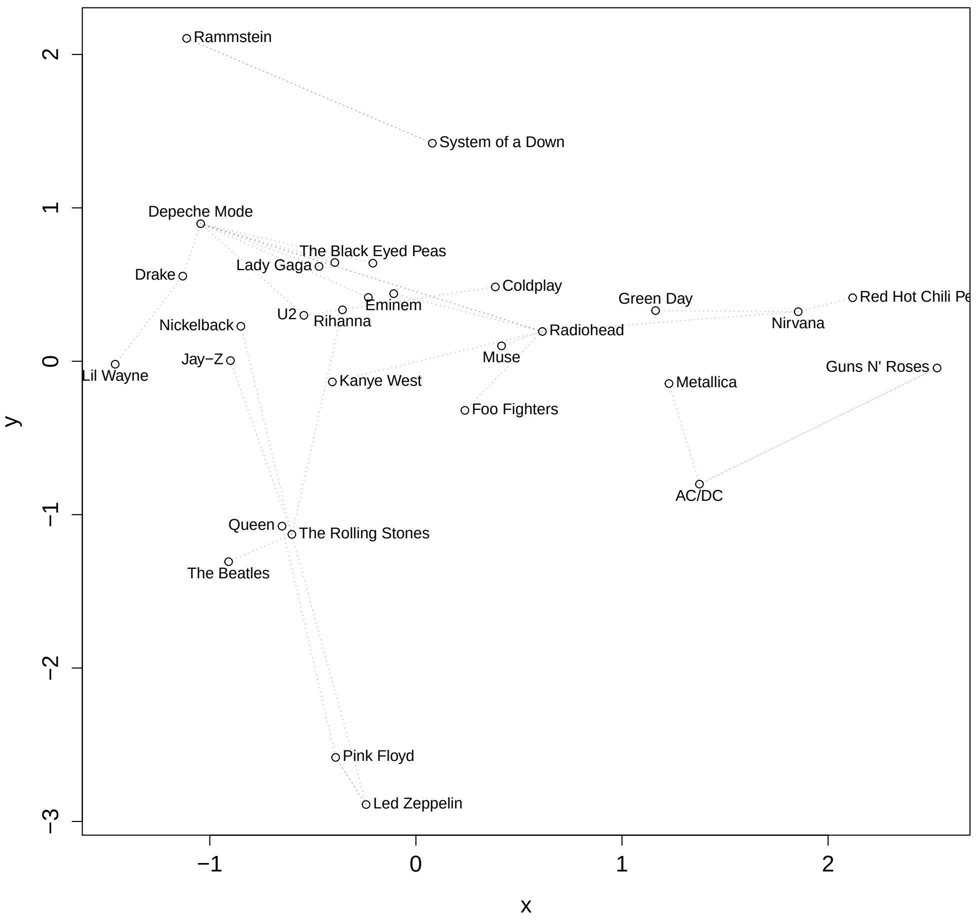

Figure 5.3: LLE Map ($k = 6$). This figure shows a nonlinear dimensionality reduction of TOP30 artists using $k = 6$ neighbors.

Chapter 6

Future Work

Improve the matching algorithm We match every user's track request to the corresponding artist, album and track. In order to be able to match even more tracks we could use fuzzy matching so that more spelling mistakes are corrected. Additionally, one could identify songs by comparing the sound of a given song with sounds of known songs[1].

Integrate TOP100 data into the Jukefox Website We provide the TOP100 list of most popular artists in the JSON format (as explained in the appendix). Using this data one could extend the Jukefox website[2] to display the most popular artists of the current month.

Use the embedding in the Jukefox player The Jukefox player relies on the notion of similarity to recommend music to users. Using our embedding one could try to improve the quality of recommendations. Artist coordinates can be retrieved using the API described in the appendix.

Improve the quality of the embedding One could improve the quality of the embedding by using a better method or combining[3] different methods. It might be a challenge to find good algorithms that are efficient and scale well. Methods that are able to provide meaningful visualizations of the artists should be preferred.

[1]The open source acoustic fingerprint system AcoustID provides a way of identifying songs by sound.

[2]http://www.jukefox.org

[3]A combination of different methods can give better results than each individual method.

Chapter 7

Conclusion

We provide a system that can be used to identify music and deliver similarity information about artists. Our system automatically recomputes the embedding every day so that users get accurate similarity information about artists that become increasingly popular. The more music taste data we collect the better the quality of our embedding will be.

Processing a large amount of data is more difficult than processing a small amount of data: Every step in the transformation of the collected data to a measure of music similarity has to be efficient. Suitable programming languages and index data structures help in achieving a good performance.

We hope that our embedding will provide music applications like Jukefox with a new foundation that will help users to explore their personal music collection using the notion of similarity.

Bibliography

[1] Hu, Y.H.Y., Koren, Y., Volinsky, C.: Collaborative Filtering for Implicit Feedback Datasets (2008)

[2] Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science **290**(5500) (2000) 2323–2326

[3] Saul, L.K., Ave, P., Park, F., Roweis, S.T.: An Introduction to Locally Linear Embedding. Available from **I**(180) (2001) 1–13

[4] Maaten, L.V.D.: Learning a Parametric Embedding by Preserving Local Structure. Artificial Intelligence **5** (2006) 384–391

[5] Maaten, L.V.D., Hinton, G.: Visualizing Data using t-SNE. Journal of Machine Learning Research **9**(2579-2605) (2008) 2579–2605

[6] Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786) (2006) 504–7

Appendix A

API

Information about artists, releases and tracks can be obtained in JSON or XML format. One can request information about musical entities by specifying all names seperated by a comma (as shown in the examples). This information can be obtained using a HTTP GET or POST request. If one would like to obtain information about multiple musical entities, we recommend to use a HTTP POST request since a HTTP GET might not work because of a URL length limit. We provide some examples using the command line tool curl[1].

Remarks

- We use comma (i.e. ,) as a separator. Hence we remove any comma that might appear in a name that appears in the request.
- We use the MusicBrainz naming conventions (the term release is used instead of album!). Everything is explained at the MusicBrainz website[2].
- XML responses can be obtained by just replacing .json with .xml in the url.

[1]http://curl.haxx.se
[2]http://www.musicbrainz.org

A.1 Artist information

We can get artist information by name or id. The following four example requests produce the same response:

```
Example requests
curl http://jukefox.org:3000/artists.json?names=The+Beatles,John+Williams
curl http://jukefox.org:3000/artists.json -d "names=The Beatles,John Williams"
curl http://jukefox.org:3000/artists.json?ids=303,94
curl http://jukefox.org:3000/artists.json -d "ids=303,94"
```

Listing A.1: JSON Response containing two artists

```
[
  {
    "begin_date_year":1957,
    "comment":null,
    "coordinates":[],
    "country":"GB",
    "end_date_year":1970,
    "gender":null,
    "gid":"b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d",
    "id":303,
    "name":"The Beatles",
    "ref_count":25457,"type":"2"
  },
  {
    "begin_date_year":1932,
    "comment":"soundtrack composer & conductor",
    "coordinates":[],
    "country":"US",
    "end_date_year":null,
    "gender":"1",
    "gid":"53b106e7-0cc6-42cc-ac95-ed8d30a3a98e",
    "id":94,
    "name":"John Williams",
    "ref_count":12482,
    "type":"1"
  }
]
```

A.2 Release information

We can get release information by providing either artist and release name or just the id. The following four example requests produce the same response:

```
Example requests
curl http://jukefox.org:3000/releases.json?names=John+Williams,From+A+Bird
curl http://jukefox.org:3000/releases.json -d "names=John Williams,From A Bird"
curl http://jukefox.org:3000/releases.json?ids=1062097
curl http://jukefox.org:3000/releases.json -d "ids=1062097"
```

Listing A.2: JSON Response containing one release

```
[
  {
    "artist_id":238569,
    "barcode":null,
    "comment":null,
    "date_day":null,
    "date_month":null,
    "date_year":2008,
    "gid":"48a018a7-5168-45ab-971a-56cce792becb",
    "id":1062097,
    "name":"From a Bird"
  }
]
```

A.3 Track information

We can get track information by providing the track id or one of the following valid parameter for the attribute names:

- `names=<Artist1>,<Release1>,<Track1>,<Artist2>,<Release2>,<Track2>,...`
  `(If we have information about the release.)`
- `names=<Artist1>,,<Track1>,<Artist2>,,<Track2>,...`
  `(If we do not have any information about the release.)`

The following four example requests produce the same response:

Example requests

```
curl http://jukefox.org:3000/tracks.json
  ?names="John+Williams,From+A+Bird,Prelude+to+a+Song"?hash=<user_hash>
curl http://jukefox.org:3000/tracks.json
  -d "names=John Williams,From A Bird,Prelude to a Song" -d hash=<user_hash>
curl http://jukefox.org:3000/tracks.json?ids=12920594
curl http://jukefox.org:3000/tracks.json -d "ids=12920594"
```

(If one specifies a hash that identifies a user, we log the request to our database.)

Listing A.3: JSON Response containing one track

```
[
  {
    "artist_id":238569,
    "coordinates":[],
    "id":12920594,
    "name":"Prelude to a Song",
    "recording_gid":"f607d95f-87c6-4aab-8874-39907c1f4307",
    "recording_id":12614706,
    "release_id":1062097
  }
]
```

A.4 Artist TOP100

We can get the 100 most popular artists of a specific month as follows:

Example request

```
curl http://jukefox.org:3000/popular_artist_logs.json?year=2011&month=8
```

Listing A.4: JSON Response containing the TOP100 for the chosen month

```
[
  {
    "id":303,
    "name":"The Beatles",
    "log_count":71306
  },
  {
    "id":946,
    "name":"Eminem",
    "log_count":60334
  },
  {
    "id":57186,
    "name":"Lil Wayne",
    "log_count":56588
  },
  ...
]
```

The log count refers to the number of requests we got for the corresponding artist in the specified month.