Modified Learning of a Gaussian Process Prior and Song Metadata Selection in Automatic Music Playlist Generation

By

Raimi bin Karim

Supervisor:

Prof. Lim Chinghway

ST4199 Honours Project in Statistics

Department of Statistics and Applied Probability

National University of Singapore

2014/2015

Abstract

Of late, the interdisciplinary science of music information retrieval (MIR) has seen a gained interest in automatic playlist generation. Selecting the "suitable" songs that blend well with our mood and putting them in the "correct" order can be achieved by hand, but not always are we right in stringing them appropriately into a playlist. Automatically generating playlist replaces this by learning the song information and seeks to produce playlists consonant with one's mood.

This project builds on these endeavours by introducing Au2seed, an automatic playlist generator that uses a modified Gaussian process prior learned from albums, user playlists, similar songs and user listening history. Gaussian process regression is carried out to learn a user preference function over songs, which is influenced by a user's mood. The final covariance kernel takes song metadata as inputs. We unveiled 10 different combinations of song features to be used as inputs, leading us to 10 different covariance kernels, which gave rise to 10 different *versions* of Au2seed. An assessment of these playlist versions has shown that they perform better than mere random generation of songs. Concurrently, we selected a handful of song features that are important in song similarity, and identified those that are dispensable. Au2seed v3.6, which we deemed the best out of the 10 versions, has displayed reasonable capabilities qualitatively when we put it to human evaluation and experimented with it for ourselves.

Keywords: music information retrieval, gaussian process regression, playlist generation, meta-training

Acknowledgements

I would like to take this opportunity to express my sincerest gratitude to:

My supervisor in-charge Prof. Lim Chinghway who has patiently guided me throughout this stint and provided invaluable feedback every now and then. Your keen help and guidance has certainly made this learning journey a pleasant one.

My family Mum, Dad, my brother and my sister for your sacrifices and all your support these years

My schoolmates Serene Yeo, Wei Kuang and Kegan Ang who have helped me build a stronger foundation in statistics; Wei Qi, Benjamin and Zuhaili for the awesome free meals; and Anirudh, Nabilah and Farah for the occasional rendezvous

My friends Fahmi, Afrizal, Syazana, Jing Da, Azmi, Samir and Co. for the support and encouragement

My test subjects Serene Yeo, Azmi, Jing Da, Fahmi, Benjamin Khoo, Wei Qi, Annastasia, Farah, Syazana and Charles who have provided us with valuable user inputs and feedback

If music be the food of love, play on.

–William Shakespeare

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Music gives a soul to the universe, wings to the mind,
flight to the imagination and life to everything.
–Plato

1.1 Background

Back in the 1970s, people still found themselves exploring songs in the discography of some artist recommended by a friend by looking through audio cassettes and vinyl records, and — about a decade later — compact discs (CDs). Today, in this age of high digital technology, the digitalisation of audio like the audio coding formats MPEG-1 or MPEG-2 Audio Layer III (MP3) and Advanced Audio Coding (AAC) has changed the way people store and access music. Such accessibility means that anyone is just several mouse clicks away from music playback.

The Internet is where potentially millions of hours of songs reside. This change in scale of accessible music online raises an interest in accessing and discovering the data of these songs and best present the music to the user. This is essentially what

the field of music information retrieval (MIR) focuses on: research and development of computational systems to help us better make sense of this data. The International Society of Music Information Retrieval (ISMIR) is a non-profit organisation that heads the forum for research on the modelling, creation, searching, processing and use of musical information or data.

More often than not, music listeners do not listen to a single song in isolation but rather, listening sessions tend to continue over a series of songs [4]: a *playlist*. A playlist is an ordered list of music to be played that can be specified within, say, a personal music collection. There are several commercial services capable of automatically creating personalised playlists (mixes) containing tracks that are likely to correspond to our tastes. Spotify[1] is a well-known online audio player that provides a radio feature streaming automatic playlists. Many music listeners will also think of Last.fm[2] and Jango[3] when automatic playlists are mentioned.

Handling large-scale data is one part of MIR. The Million Song Dataset (MSD) is a large collection of metadata and audio features. Metadata is a set of data that describes and gives information about other data. This huge collection is compiled by Laboratory for The Recognition and Organization and Audio (LabROSA)[4] of Columbia University [2] using the services of The Echo Nest[5]. Its purposes are:

- To encourage research on algorithms that scale to commercial sizes
- To provide a reference dataset for evaluating research
- As a shortcut alternative to creating a large dataset with APIs (e.g. The Echo Nest API)

[1] http://www.spotify.com
[2] http://www.last.fm
[3] http://www.jango.com
[4] http://labrosa.ee.columbia.edu
[5] http://the.echonest.com

- To help new researchers get started in the MIR field

In this paper, the MSD is our primary source of information, to which we match other datasets like the Art of the Mix, Last.fm and Taste Profile Subset.

1.2 Motivation

Everyone listens to music in many different styles: some people listen to songs by using the shuffle feature, some go through an album from an artist, while others choose to create playlists manually by hand. However, conditional on our mood at the point of time while we listen to music, running through the shuffle feature and albums often requires user intervention i.e. skipping the track because we find that Nicki Minaj's Anaconda might not be so apt with our current state of melancholy, for example. Even within an album itself, the *mood* can vary significantly from song to song.

Constructing a playlist that matches a particular *mood* could be one way to solve this. However, such activity is tedious and takes some time. Constructing one in advance is also not simple as one might not anticipate all possible music moods or preferences in the future.

Ideally, we envisage a system that will not only *automatically* pick up a user's mood and play suitable music from an available collection of songs, but also able to respond to and learn from user feedback, with minimal user intervention. This motivates the design of an *automatic playlist generator*: Au2seed. For instance, an athlete who is about to perform calisthenics at home selects 2 tracks from his music library to listen to during his workout, possibly upbeat songs from Eminem to which he can keep up his reps. The Au2seed system generates a playlist of 'workout' songs and will improve this playlist when this athlete removes songs he does not deem fit to his exercise routine.

1.3 Outline

As our mood affects our songs preferences at different times, we will quantify 'mood' as the *user preference* function f over a certain song. The observed user preference, y_i takes a value of 1 if a user likes song i and 0 otherwise. Au2seed uses Gaussian process regression (GPR) to learn this user preference function, which takes song metadata (or song features) as inputs [1]. The Gaussian process is completely specified by the mean and the covariance kernel, which is first learned from different datasets in the modified kernel meta-training (KMT) phase (first introduced in [1]). The final covariance kernel is a sum of weighted Mercer kernels family, taking song metadata vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$. We have 10 different combinations of song features to be used as the inputs, leading us to 10 different covariance kernels. One can see these 10 different covariance kernels as 10 different versions of Au2seed. These versions were put to the test in the chapter on evaluation, where we picked the best Au2seed version. Concurrently, we also decided which song metadata are useful to us, and which ones are dispensable.

1.4 Goals

The goals of this project are highlighted as follows:

- To utilise both (numerical) audio features from the Echo Nest and textual metadata as the source of data
- To select 7 best metadata or audio features that capture the notion of playlist generation
- To improve the Gaussian prior learning process in kernel meta-training

- To create an automatic playlist generator (as a simple R function) that requires very minimal user intervention and whose playlist suits the user's mood during a particular listening session, given a library of songs

1.5 Programming Software

As the statistical programming software we have used R and several R packages [3]. The R codes can be found in the accompanying CD.

1.6 Nature of Project

This project involves a lot of work in data collection and pre-processing, experimentation with different priors and possible combinations of metadata, and evaluation of these models.

Chapter 2

Related Work and Literature Review

To the best of our knowledge, this is the first work involving the use of categorised timbre values derived from the MSD together with the KMT (which was introduced in [1]).

The bulk of this project is built upon the framework of the automatic playlist generator AutoDJ created by Platt et al. [1], where most of the playlist generation literature and setups are taken from and, for certain parts, modified. AutoDJ uses Gaussian Process Regression to learn a user preference function over songs. A large set of albums was used to learn a Gaussian process prior for playlist generation. This learned kernel was demonstrated to be more effective at predicting users' playlist than a hand-designed kernel. Though the dataset used was large, the features that they used were semantic in nature (as opposed to numerical). Moreover, there was no mention of using timbre (the quality of a musical sound that distinguishes different types of sound production, such as voices and musical instruments), which we consider a very important element in playlist generation. This work suggests several improvements that can be done and has captured our attention with its stimulating idea and literature, it

being relevant to real-life scenarios. Our concept of playlist generation by means of a Gaussian process regression is developed from this idea.

"Learning how to learn", or meta-training in this paper, is also called multitask learning [5]. Such a task learns a problem together with other related problems at the same time, offering an elegant way to express prior knowledge as a series of example tasks. [5] demonstrates how "learning how to learn" can be done by first partitioning the data, then learning a model for the partitions, and finally applying this model to new partitions. This interpretation of meta-training seems very promising, and would thus be applicable and relevant in the field of MIR.

Previous work in regression with Gaussian processes [6] made use of Gaussian processes for regression problems, employing the Bayesian machinery, used in this paper. Williams and Rasmussen showed how the hyperparameters, which ultimately control the final form of the Gaussian process, can be estimated from the data itself using a maximum likelihood or Bayesian approach. The results from some machine-learning tasks indicated that the Gaussian process approach is giving very similar performance to two well-respected techniques MacKay and Neal. However, unlike [1], they did not make use of meta-training data — the parameters are only learned from the training set. No useful similarity metric can be derived from one training example, so like [1], we used meta-training to learn the kernel.

[7] developed a flexible, scalable, and efficient class of generative playlist models based upon hypergraph random walks. The playlist models were tuned to specific categories or 'dialects' of playlists (e.g. Hip Hop, Electronic Music), each of which may exhibit its own particular structure. McFee et al. demonstrated that playlist model performance can be improved by treating specific categories of playlists individually. The use of hypergraph models as playlists is interesting but is not in our scope as the

playlists are of different categories.

McFee et al. also undertook a very interesting approach: the playlist as a natural language [4]. This is where (in statistical natural language processing) in place of vocabulary words, we have songs; rather than sentences, we have playlists. He prescribed a generative evaluation perspective when modelling playlist composition rather than attempting to objectively score playlists as good or bad. This work proposed an efficient algorithm to learn an optimal combination of simple playlist algorithms. Experiments on naturally occurring playlists demonstrate the efficacy of the evaluation procedure and learning algorithm. The idea of playlist as a natural language is indeed a fascinating perspective to us but is not in our context unfortunately.

[8] presented an extensive set of experiments in collaborative filtering and recommender systems, and used 2 main classes of evaluation metrics: accuracy over a set of individual predictions and utility of a ranked list of suggested items. The latter has been used in [1] and will be used in this paper as our basis of comparison for different playlist versions. The ranked scoring metric is defined for collaborative filtering applications with an ordered list of recommended items. The nature of our data — categorical — and of our playlist — ranked list — allows this evaluation method to be used befittingly in this thesis.

Initial Project Proposal: Improved Collaborative Filtering Recommender System to Bypass the New-Item Problem

Besides playlist generation, we had been exploring recommender systems in the MIR at the start of this project. We had also proposed a thesis topic in this subfield, but later switched to playlist generation as our area of research upon the discovery of the work produced by Platt et al.

Our main idea for this project proposal was the following: some recommender systems employ collaborative filtering to recommend music to other users in a particular platform. However, the collaborative method does not recommend music that has not yet been rated. This is known as the new-item problem. As a result, a user would miss out on the music that he might like. Also, the collaborative filtering algorithm, though commonly used, suffers from poor accuracy and high running time.

In recommender systems, [9] introduced a recommender system that can integrate both collaborative data (rating scores of users) and content-based data (acoustic features of audio signals). The result is that the system boasts a high degree of recommendation accuracy, a large variety of artists and a capability for recommending non-rated music. [10] proposed a switching hybrid recommendation approach by combing item-based collaborative filtering with a Naïve Bayes classifier, which is based on the Bayes' theorem. The authors showed that the approach empirically outperforms others in terms of accuracy and the coverage is more scalable. The proposal for the recommendation approach in [11] employs user clustering and Minkowski distance, and a method to guess no-vote ratings based on the k-nearest neighbours algorithm. The results showed that their approach is more accurate, but is more time-consuming than collaborative filtering.

Chapter 3

Au2seed: The Playlist Generator

"Au2seed" is a homonym of "auto-seed" — the playlist generator takes in *2 seed* songs (or starting songs) as inputs and *automatically* generates a playlist based on them.

3.1 Goal of Au2seed

The goal of Au2seed is to create an effective playlist for a user with minimal human intervention. An "effective playlist" is a playlist whose songs are pleasing to the listener, i.e. songs that are able to describe a listener's mood. Note that within a playlist, the ranking of songs is important. Effectively, the playlist starts with the selected seed songs, proceeds to the songs that are *similar* to the specified seed songs, and, towards the end of the playlist, gradually shifts away from the seed songs in terms of similarity.

3.2 A User's Perspective

The songs recommended by Au2seed are only limited to songs from a local library, i.e. a collection of music that a user has specified to be used as the music library. Except

for obtaining song metadata (*tempo*, *flatness* etc.) from the Echo Nest, Au2seed carries out playlist generation offline.

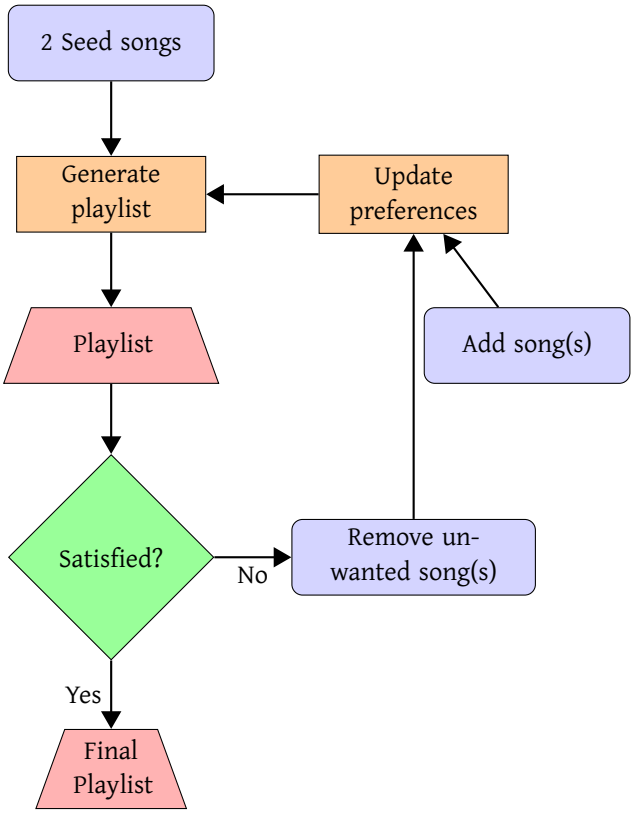

Figure 3.1: How Au2seed Works from a User's Perspective.

Its function is simple: generating a playlist of songs similar to the seed songs. Technically, it is possible to only start with 1 seed song but in this application, we would like a minimum of 2 songs as seed songs for more prediction accuracy and minimal user intervention. Until satisfied, the user is allowed to review the playlist by adding or removing tracks from it. By doing so, the system updates and tries to learn the user's song preferences, and replaces these songs with newer ones. Ideally, user intervention should not occur as often, as stated in one of the goals of Au2seed. Figure 3.1 provides an overview by means of a flowchart.

3.3 Building the System

This project is divided into 4 main phases, in chronological order:

1. Data collection and pre-processing (Chapter 4: Datasets)

2. Building different kernels using different sets of features (Chapter 5: Method)

3. Kernel model evaluation and selection, and song metadata recommendation (Section 6.1: Quantitative Evaluation)

4. Au2seed testing: User library data collection and pre-processing, and human evaluation (Section 6.2: Qualitative Evaluation)

The first phase is marked by building a database of songs and their *metadata* or *features*. A metadata is a set of a data that describes and gives information about other data. In this paper, both 'metadata' and 'features' are used synonymously. Examples of these include genre, *flatness* and *brightness*. These features were then pre-processed (by transforming into categorical data etc.) for them to be useful as inputs to the Mercer kernels. We also obtained other datasets to be used in KMT.

In Phase 2, we 'combined' the meta-training kernel and the Mercer kernels to obtain a Gaussian process covariance kernel. This process was repeated ten times for every combination of song features, since comparing between different combinations of song features is one of the aims of this project. Thus, in total we had 10 different covariance kernels. Note that the Gaussian process covariance kernels are like the machinery behind every playlist generator. Refer to Section 3.4 for these 10 versions.

Phase 3 evaluated the playlist generator across the different versions. In doing so, we selected the best features that can be used for playlist generation in the future.

Lastly, we obtained music libraries of our test subjects, obtained the respective song features from the Echo Nest, and computed a user covariance kernel for every subject. Human evaluation was carried out in this Phase 4.

Refer to Figure 3.2 for a brief flow chart of these phases.

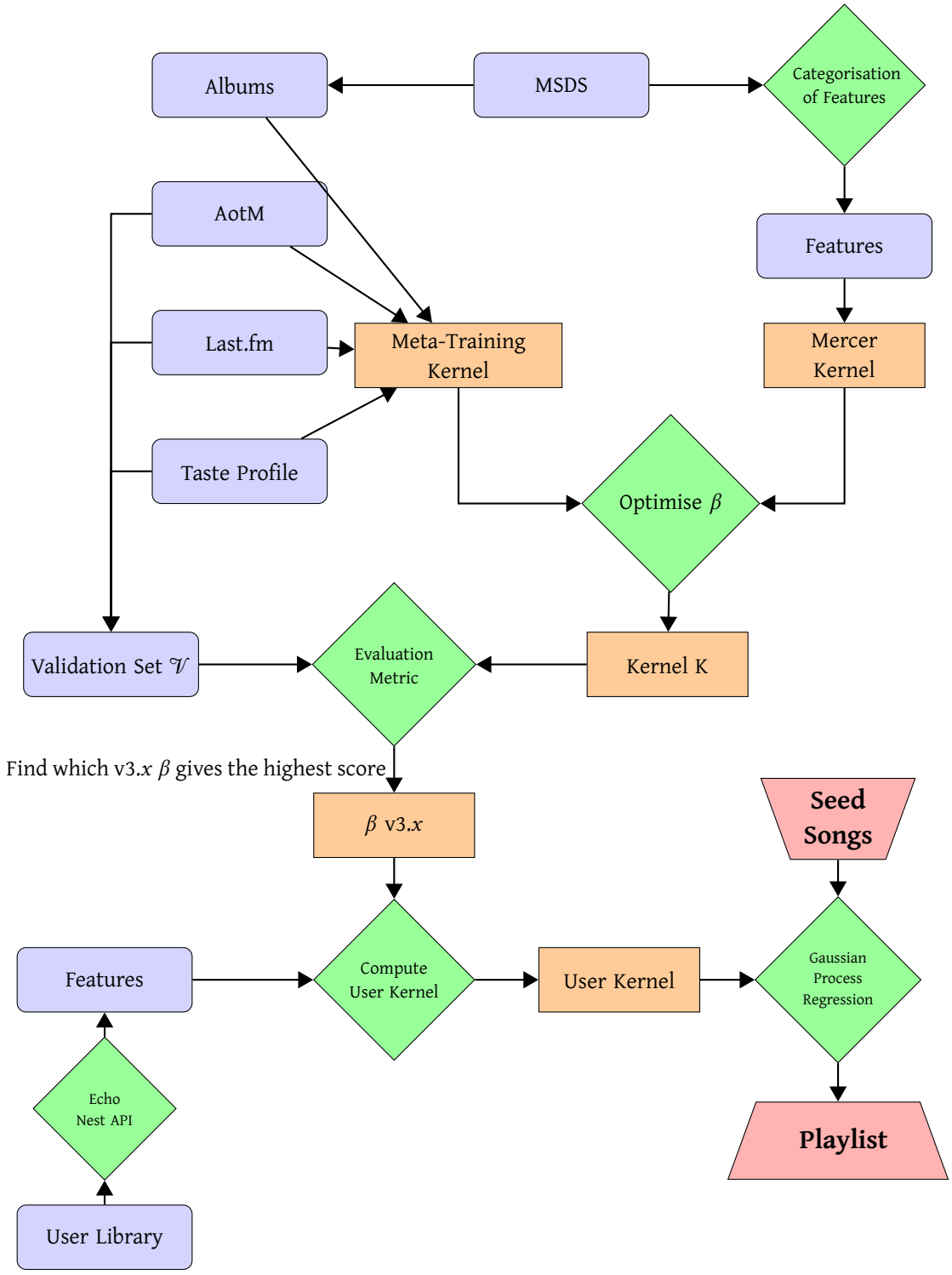

Figure 3.2: Building Au2seed. Blue indicates data, green are processes, orange are matrices or vectors, and red are inputs or outputs.

3.4 Au2seed Versions

From the 10 different sets of song features, we obtained 10 different Gaussian process covariance kernels. A covariance kernel gives the playlist generator and idea how songs should be correlated in terms of similarity. As such, one could think of a playlist generator as a covariance kernel itself.

During the course of this project, new datasets were occasionally discovered so we introduced certain new features. We also came across several papers that suggest several techniques for features used in playlist generation. These changes have been documented as different *versions*, where every version utilises a different combination of features, as seen in Table 3.1.

Metadata Field	v3.0	v3.1	v3.2	v3.3	v3.4	v3.5	v3.6	v3.7	v3.8	v3.9
Genre	✔	✔	✔							✔
Key	✔									
Mode	✔	✔			✔				✔	
Sections	✔									
Tempo	✔	✔	✔		✔	✔		✔		
Time signature	✔		✔		✔	✔	✔			
Segment_loudness	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Brightness		✔	✔	✔	✔	✔	✔	✔	✔	✔
Flatness		✔	✔	✔	✔	✔	✔	✔	✔	✔
Attack		✔	✔	✔	✔	✔	✔	✔	✔	✔
Timbre5				✔		✔	✔	✔	✔	✔
Timbre6				✔			✔	✔	✔	✔
Timbre7				✔						

Table 3.1: Summary of Features Used for Different Playlist Versions.
The lower part of the table are the timbre fields.

In our application, we maintained a maximum of 7 categorical features (discussed in Section 5.3.1: Mercer kernels in page 37). The timbre values are ordered by importance [12]. So, if 4 timbre values were to be chosen as features, then it will be the first 4 timbre values that will be used. More on the description and transformation of the

features from continuous to discrete in Chapter 4: Datasets on page 16.

3.4.1 Earlier Versions

The first few playlist versions made use of the following categorical features: genre, *tempo*, *key*, *time signature*, *mode*, *sections* and *year*. No timbre values were used at all. The genre tags were all machine-learned (from a particular dataset). The meta-training kernel was learned by using only the AotM dataset. These versions are not reflected in this paper as we consider them to be obsolete, deprecated and not worth for comparison, mainly due to the usage of *year*.

3.4.2 v3.0 – v3.9

These versions made use of timbre values. In order to ensure that every song in our database has a genre tag, we merged 2 sources of data together: machine-learned genre tagging and ground truth data. We also enhanced the kernel meta-training by combining the AotM, Last.fm and Taste Profile datasets. These are user playlist data, song-similarity data and user listening history data respectively. Such combination of data reduced the sparsity of the kernel meta-training matrix and, intuitively, provided us with a better Gaussian process prior. Section 5.3.3: Kernel Meta-Training on page 39 discusses this further.

Chapter 4

Datasets

We have discovered and explored a total of 6 collections of data in this project, summarised in Table 4.1 below.

Name	Type	# Raw Records	Purpose
Million Song Dataset (MSD)	Metadata: audio features, song information	10,000	Mercer kernel
MSD Genre Tags	Genre tags	990,013	Mercer kernel
MSD Allmusic	Genre tags	406,427	Mercer kernel
Art of the Mix	User playlists	101,343	KMT & validation set
Last.fm	Similar songs	584,897	KMT & validation set
Taste Profile	User listening history	48,373,586	KMT & validation set
The Echo Nest API[1]	Metadata: audio features, song information	-	Evaluation

Table 4.1: Sources of Information Used in This Project

[1] `http://developer.echonest.com/docs/v4`
The Echo Nest's Application Programming Interface (API) allows us to call methods using an HTTP GET that responds in JSON. This API allows us to retrieve song information and audio features on an ad-hoc basis.

4.1 Song Features

Song features were used as Mercer kernel inputs in Section 5.3.1, page 37.

4.1.1 Million Song Dataset

Released by Columbia University's LabROSA, the Million Song Dataset (MSD)[2] is the main source of data for this project [2]. Thanks to The Echo Nest, a music intelligence platform company that provides music services to developers and media companies, this freely-available collection of data contains metadata and audio features for one million popular and contemporary Western songs in different genres, at different years and of different country origins. Metadata consists of author, album, title and so on. There are two major sets of audio features that are worth mentioning: Mel-Frequency Cepstral Coefficients (MFCC)-like and 'Chroma-like' (to be explained in the next few sections). The metadata explored for this paper is shown in Table 4.4 (all except for genre). For a full list of metadata available in the MSDS, refer to Appendix A.

Since it was too ambitious to work with the MSD itself due to its huge file size (300GB) and computational limitations, we settled for the smaller dataset: the MSD Subset[3] (MSDS), a subset of 1% of the MSD comprising 10,000 songs selected at random, which is only 2.7GB. The rich source of information within the MSDS serves the main purpose of the use of this dataset well: harness the large quantity of musical features extracted from one million songs to represent "all the songs in the true universe".

The file type used in the MSDS is HDF5[4]. We downloaded an R package `rhdf5` that can access this data of hierarchical nature. This package can reduce a 9939×9939 matrix of size 2GB to about 70MB.

[2] `http://labrosa.ee.columbia.edu/millionsong/`
[3] `http://labrosa.ee.columbia.edu/millionsong/pages/getting-dataset`
[4] `http://www.hdfgroup.org/HDF5`

Duplicates in the field of MIR is close to inevitable. We have removed 61 duplicates that were among the 10,000 songs, leaving us with 9939 songs. A list of the duplicates can be found on LabROSA's blog[5].

In this project we acknowledge the limitations of using the MSDS. A 1% random sampling of the MSD might have an impact on our results if, coincidentally, most of the sampled songs are, say, unpopular songs or from the same genre. Our results might be skewed and inaccurate due to this and our hopes for regarding the MSDS as "songs in the universe" might be dashed.

4.1.2 Genre Tags Data

Allmusic Dataset

Offering a wide range of music information including album reviews, artist biographies, as well as classification of albums according to genres, styles, moods and themes, the Allmusic website[6] is our source for ground truth genre tags. These tags were provided and curated by experts [13]. Schindler et. al have publicly made the MSD Allmusic Top Genre Dataset (Top-MAGD) available on their website[7] for benchmarking evaluations [13] against the MSD. Hence we were able to conveniently manipulate the data without much cleaning.

Note that the motivation behind obtaining this dataset is that almost every song in the MSDS has more than one tag (e.g. "pop rock", "american" and "female"). It is hard to mine genre tags among terms that describe the language of the song, the gender of the singer and so on. The Allmusic dataset presents itself as a better alternative as each song in its database is only tagged to one genre. Also, song genre is intuitively

[5] http://labrosa.ee.columbia.edu/millionsong/blog/11-3-15-921810-song-dataset-duplicates
[6] http://www.allmusic.com
[7] http://www.ifs.tuwien.ac.at/mir/msd/download.html

considered to be important in song recommendation, so genre tags should definitely be explored.

Genre	Blues	Country	Electronic	Folk	International	Jazz	Latin
No. of songs	126	174	224	48	147	173	257

Genre	New Age	Pop/Rock	R&B	Rap	Reggae	Vocal	(no genre)
No. of songs	95	1914	132	234	120	36	6259

Table 4.2: Allmusic Data

Each song in this dataset is tagged to one of the 13 genres used by Allmusic. After matching these songs to the MSDS, we only found 3680 matches. There were still 6259 songs whose genre tags were still not yet identified. The dataset in the next section will supplement this.

Million Song Dataset Genre Tags

Genre	Blues	Country	Electronic	Folk	International	Jazz	Latin
No. of songs	1469	731	92	-	36	638	-

Genre	New Age	Pop/Rock	R&B	Rap	Reggae	Vocal	(no genre)
No. of songs	-	5731	348	348	105	-	61

Table 4.3: MGT Data

This dataset was discovered prior to the Allmusic dataset. The Million Song Dataset Genre Tags[8] (MGT) is a product of genre classification research by Hu et. al where the genre tags were labeled by a machine-learning method using a combined classifier [14]. We chose this dataset because of 3 reasons: (i) the genre tags used are from ten Allmusic genres; (ii) the classification performance is encouraging; (iii) the author used

[8] `http://web.cs.miami.edu/home/yajiehu/resource/genre/`

the entire MSD (990,013 songs) for genre tagging. We were contented to find 99.38% matches with the songs in the MSDS; only 61 songs had no values.

Combining the Allmusic Dataset and MGT

Upon the discovery of the Allmusic dataset, we explored the accuracy of Hu's MGT with the ground truth Allmusic dataset. First, we matched all the 3680 songs from Allmusic with the songs in MGT that had genre tags. This amounted to 3654 songs for comparison. We found that there was a 50.36% accuracy of Hu's prediction in these 3654 songs. This proportion is very decent: Hu's genre tagging task is correct more than half of the time. Hence these machine-learned genre tags are quite reliable and useful to fill in the missing ground truth tags.

Before combining these two sets, notice from Table 4.3 that the MGT lacks 4 categories of genre tags compared to what Allmusic has: Folk, Latin, New Age and Vocal. We decided to combine them anyway as these genres look like they constitute only a small proportion of the other genres.

To merge these 2 datasets together, 6259 missing genre tags in the Allmusic dataset were filled in with Hu's data. Theoretically, we would expect only about $\frac{6259}{9939} \times 50.36 \times 100\% = 31.7\%$ of the combined genre dataset to be incorrect. Given the lack of benchmarking data online, this accuracy rate is tolerable in this paper.

Finally, after merging, there were only 35 songs that did not have any genre tags. This is promising as now the genre tags data is collectively substantial. We have since renamed this group of 35 no-genre songs to "Miscellaneous". The combined genre tags can be seen in Section 4.1.4.

4.1.3 Summary: Features Not Used

We refer the reader to Appendix A for a glossary of several musical terminologies used in this section. Some reasons why features were excluded are: (i) they were not considered to be relevant to song similarity; (ii) their potential applications were deemed too complex; and (iii) there was not enough information in the MSDS.

Pitch

Pitch content is given by a "chroma" vector, corresponding to the 12 pitch classes C, C#, D to B, with values ranging from 0 to 1. Every song has a $12 \times k$ vector of chroma-like vector, where k is the number of segments. The main reason for not using this is mere intuition — it is not useful enough given our limited knowledge of information retrieval. The data is too complex to be summarised in statistics; [15] only took the mean and variance of each of the 12 dimension. Chroma features showed inferior music classification performance compared to MFCC [16], also reproduced in [15].

Duration

As many commercial songs that we know of are about 3–5 minutes long, we strongly believe that this metadata would not be useful to us.

Sequenced Data

By intuition, we postulate that sequenced data like number of segments, tatums, beats and bars are not able to capture differences between two songs as we feel that this numerical information is too fundamental. However the number of sections, categorised under sequenced data, was used. See the next section for details.

Year

It is unfortunate that more than half of the songs — 5320 — had missing values in the *year* metadata. Otherwise, we could have exploited this valuable information. It is possible that music in a certain era, say, 70's might have certain common traits, alluding to similar songs.

4.1.4 Summary: Features Used

We refer the reader to Appendix A for a glossary of several musical terminologies used in this section. Except for genre tags, *mode* and *time signature*, the following features have been transformed from continuous data into categorical data. These are the features that have been used for the playlist generator. Categorisation of the features is necessary as the KMT and the Mercer kernels only accept categorical variables.

Metadata Field	Example Values	No. of Levels	Initial Type of Data
Genre	Electronic, R&B, Pop/Rock	14	●
Mode	0, 1	2	●
Sections	5, 7	4	△
Time signature	1, 3, 4, 5, 7	5	●
Tempo	1, 2, 3, 4, 5	5	△
Segment_loudness	11, 12, 13, 14	16	△
Brightness	11, 12, 13, 14	16	△
Flatness	21, 22, 23, 24	16	△
Attack	21, 22, 23, 24	16	△
Timbre5	31, 32, 33, 34	16	△
Timbre6	31, 32, 33, 34	16	△
Timbre7	41, 42, 43, 44	16	△

Table 4.4: Summary of Features Used. Except for genre, all metadata is from the MSDS. The features in the lower part of the table are timbre values.
● = discrete; △ = continuous.

Several reasons why we chose these features are: (i) features can be easily tagged to a specific group of songs, e.g. genre; (ii) features can numerically describe songs,

e.g. *tempo*; and (iii) we did not eliminate the possibility of a certain group of people liking songs of a certain attribute of a feature, e.g. songs in minor key.

Genre Tags

Genre tags were derived from combining the Allmusic and Hu's machine-learned genre tags, explained in Section 4.1.2. The 14 different genre tags for the 9939 songs are:

Genre	Blues	Country	Electronic	Folk	International	Jazz	Latin
No. of songs	1210	699	289	48	174	625	257

Genre	New Age	Pop/Rock	R&B	Rap	Reggae	Vocal	Miscellaneous
No. of songs	95	5210	347	726	188	36	35

Table 4.5: Combined Genre Tags

Key

This is a categorical variable with 12 levels: integers from 0 to 11, each representing C, C#, D, ... and B. There might be a possibility of people liking songs in a certain key, hence we introduced this in one of our playlist versions.

Mode

'0' indicates that a song is in a minor key (sounds sad) while '1' a major key. 69.1% of the songs are in major key (sounds happier), while the rest are in minor.

Sections

Sections are defined by large variations in rhythm or timbre, e.g. chorus, verse, bridge, guitar solo, etc. [12]. As the number of sections for a song can reach up to 78 (an outlier), we referenced the breakpoints by calculating the 25^{th}, 50^{th} and the 75^{th} quantiles.

No. of sections	6 or less	7–9	10–13	14 or more
No. of songs	1577	3402	3003	1957

Table 4.6: *Sections*

The respective quantiles are 7, 9 and 12. Quartiles measure spread of the observations. Since they report order statistics (rather than, say, the mean), they are appropriate for interval measurements. However, it was later decided that the number of sections would have lesser impact on music content similarity. Hence, only v3.0 includes *sections* as one of the features.

Tempo

Song tempo is measured in beats per minute (BPM). Initially, we tried k-means method to divide *tempo* into categories but the results did not reflect our intuition. Upon plotting a histogram of the tempi of all the songs, and with reference from [17], we heuristically determined 84, 112, 142 and 176 to be the dividers. In particular, we noticed a steep incline/decline at the first three breakpoints (Figure 4.1), possibly indicating different classes. Coincidentally, these numbers somewhat divide the groups of musical tempi together, as seen in Table 4.7, and Table 4.8 on page 24, where information was taken from the Classical Music City website[9].

***Tempo* Levels (BPM)**	**Musical Tempo**	**Meaning in English**	**No. of Songs**
0 – 84	Adagio, Largo, Grave	slow, very slow	925
85 – 112	Andante, Moderato	at a walking pace, moderately	3216
113 – 146	Allegro Moderato, Vivace, Allegro	moderately quick, fast	3441
147 – 168	Allegro	fast to very fast	1559
177 and above	Presto, Prestissimo	very fast, extremely fast	798

Table 4.7: Classification of *Tempo*

[9] `http://www.classicalmusiccity.com/search/article.php?vars=446/Basic-Tempo-Markings.html`

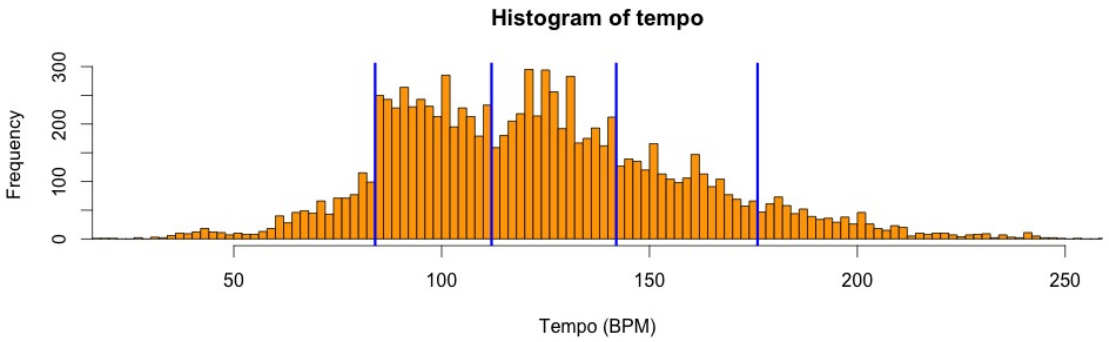

Figure 4.1: Histogram of *Tempo* Before Transformation. Breakpoints are indicated by the horizontal blue lines.

Musical Tempo	Meaning in English	Approximate BPM
Larghissimo	very, very slow	20 and below
Grave	slow and solemn	20 – 40
Lento	slowly	40 – 60
Largo	broadly	40 – 60
Larghetto	rather broadly	60 – 66
Adagio	slow and stately (literally, "at ease")	66 – 76
Adagietto	rather slow	70 – 80
Andante moderato	a bit slower than andante	
Andante	at a walking pace	76 – 108
Andantino	slightly faster than andante	
Moderato	moderately	108 – 120
Allegretto	moderately fast (but less so than allegro)	
Allegro moderato	moderately quick	112 – 124
Vivace	lively and fast (quicker than allegro)	≈140
Allegro	fast, quickly and bright	120 – 168
Allegro	fast, quickly and bright	120 – 168
Vivacissimo	very fast and lively	
Allegrissimo	very fast	
Presto	very fast	168 – 200
Prestissimo	extremely fast	more than 200

Table 4.8: Musical Tempi and *Tempo* Classification. The double horizontal lines roughly divide *tempo* into the 5 categories as used in this paper. Note that 'allegro' spans two categories.

Time Signature

Time signature takes values 3, 4, 5 and 7 indicating time signatures of $\frac{3}{4}$, $\frac{4}{4}$ etc. A value of 1 indicates a rather complex or changing time signature. There were initially 3 values whose *time signatures* were 0. We manually looked these songs up in the Echo Nest API using an API key and filled in the missing time signatures.

Time signature	1	3	4	5	7
No. of songs	1488	1271	6313	587	280

Table 4.9: *Time Signature*

Timbre

Also referred to as sound texture or tone quality, the quality of a musical note or sound that distinguishes different types of musical instruments, or voices is called timbre. The MSDS provides a vector of 12 values for every segment[10] in of a song, ordered by degree of importance. The first four are:

1. average loudness of a segment (*segment_loudness*)
2. *brightness*
3. closely correlated to the *flatness* of a sound
4. closely correlated to sounds with a stronger *attack*

Unfortunately, these are the only dimensions described in words in the Analyzer documentation [12]. Further interpretation given to timbre values as processed by the Echo nest is found in Appendix A. Since our Mercer kernels (Section 5.3.1) can only accept 7

[10] A segment is a set of sound entities (typically under a second) each relatively uniform in timbre and harmony. Segments are characterised by their perceptual onsets and duration in seconds, *segment_loudness*, pitch and timbral content [12].

variables at maximum, we took the 7 most important dimensions, which are the first 7 dimensions, according to The Echo Nest.

Categorising timbre values was a little tricky. With each track having about 700 to 900 segments, every song would have a large vector of about 900×7. In order to summarise these values, we took the mean and variance for every timbre dimension for each song. While Schindler and Rauber [15] suggested taking the statistical moments (mean, variance, median, skewness, kurtosis, min and max) the information provided by the MSDS, we argue that this would not be feasible in our application, as it will be too complicated to use 3 categorical variables. Nonetheless, [15] remarked that top results can already be obtained by calculating the average and variance of Echo Nest's timbre feature.

Firstly, for each song, we calculated the mean and log(variance) (to reduce the skewness of large values caused by taking variances) of every timbre feature (i.e. for *segment_loudness*, *brightness*, *flatness*, etc.). Then for every timbre feature, we took the 25^{th}, 50^{th} and the 75^{th} percentiles of all the 9939 means and variances and used them as the dividers for the categorisation levels. We took the percentiles as such a categorisation would be robust to outliers. Lastly, we grouped them into a 4×4 table. This 2-dimensional categorisation of continuous data can capture both the mean and variance at the same time, satisfying the suggestion by Schindler and Rauber.

This method of categorisation is able to classify songs according to the mean and variance of a timbre feature throughout the song, with reference to other songs in the MSDS. For example, a *brightness* of '41' means that the song is, compared to other songs in the MSDS, generally very 'bright' (high mean) and, stays 'bright' throughout the song (low variance). In another instance, if a song's *flatness* is '44', it is generally very flat (high mean) and deviates a lot in its *flatness* throughout (high variance).

		log(var)			
		less than 2.885	2.885 to 3.334	3.334 to 3.733	more than 3.733
mean	less than 38.89	11	12	13	14
	38.89 to 43.34	21	22	23	24
	43.34 to 46.89	31	32	33	34
	more than 46.89	41	42	43	44

Table 4.10: Categorisation of *Segment_loudness*

		log(var)			
		less than 7.233	7.233 to 7.624	7.624 to 7.996	more than 7.996
mean	less than -29.87	11	12	13	14
	-29.87 to 6.533	21	22	23	24
	6.533 to 35.16	31	32	33	34
	more than 35.16	41	42	43	44

Table 4.11: Categorisation of *Brightness*

		log(var)			
		less than 7.125	7.125 to 7.496	7.496 to 7.86	more than 7.86
mean	less than -10.11	11	12	13	14
	-10.11 to 12.78	21	22	23	24
	12.78 to 33.52	31	32	33	34
	more than 33.52	41	42	43	44

Table 4.12: Categorisation of *Flatness*

		log(var)			
		less than 6.834	6.834 to 7.245	7.245 to 7.649	more than 7.649
mean	less than -8.069	11	12	13	14
	-8.069 to 0.1181	21	22	23	24
	0.1181 to 10.01	31	32	33	34
	more than 10.01	41	42	43	44

Table 4.13: Categorisation of *Attack*

		log(var)			
		less than 6.473	6.473 to 6.757	6.757 to 7.046	more than 7.046
mean	less than -17.85	11	12	13	14
	-17.85 to -2.7	21	22	23	24
	-2.7 to 10.5	31	32	33	34
	more than 10.5	41	42	43	44

Table 4.14: Categorisation of *Timbre5*

		log(var)			
		less than 6.279	6.279 to 6.692	6.692 to 7.086	more than 7.086
mean	less than -16.2	11	12	13	14
	-16.2 to -7.73	21	22	23	24
	-7.73 to 2.193	31	32	33	34
	more than 2.193	41	42	43	44

Table 4.15: Categorisation of *Timbre6*

		log(var)			
		less than 6.064	6.064 to 6.392	6.392 to 6.683	more than 6.683
mean	less than -13.38	11	12	13	14
	-13.38 to -3.963	21	22	23	24
	-3.963 to 4.893	31	32	33	34
	more than 4.893	41	42	43	44

Table 4.16: Categorisation of *Timbre7*

4.2 Meta-Training Datasets

Meta-training datasets are used in the kernel meta-training (KMT) phase in Section 5.3.3, page 39.

4.2.1 Art of the Mix Dataset

The Art of the Mix (AotM) dataset contains over 101,000 user-contributed playlists. The processed form of this playlist data[11], collected by Berenzweig, et. al [18] was taken from the AotM website[12]. We chose this dataset because firstly, it is the largest publicly available playlist dataset to the best of our knowledge and secondly, each playlist was ostensibly generated by a user — not by a recommendation service or commercial radio DJ — so the corpus is an accurate sample of real playlists that occur in the wild [4]. We believe that this playlist data would form a very good prior for our kernel meta-training. Moreover, songs that occur in the same playlist are in some sense related, although it is not implied that they are the most similar.

The data provided to us was in JSON format. We parsed this file using an R package `jsonlite`. In the data-cleaning stage, we deleted all playlists that contained only one song — also a practise in [19]. Having only one song in the playlist would not capture the essence of playlist generation. Duplicate tracks within every playlist were subsequently removed as they might affect the KMT phase. We also deleted one of the playlists that had 21 songs, all of which were different renditions of the same song *Oops I Did It Again* by Britney Spears. Finally, after resolving songs to the MSDS, we were left with 815 unique playlists and 503 unique songs. The distribution of playlist lengths are summarised in Table 4.17.

[11] `http://www.ee.columbia.edu/~dpwe/research/musicsim/aotm.html`
[12] `http://www.artofthemix.org`

No. of songs in playlist	2	3	4
No. of such playlists	766	47	2

Table 4.17: AotM Playlist Data

The AotM data does not seem to reflect the true nature of playlists in reality: playlists are usually more than 2 or 3 songs long. This is why we sought another data source in the next section.

4.2.2 Last.fm Dataset

The Last.fm dataset[13] is the official song tags and song similarity collection for the MSD [2]. The collaboration between the MSD team and Last.fm provides us with the largest research collection of song-level tags and precomputed song-level similarity. We benefit from this accessibility and convenience as all the data is associated with MSD tracks, which makes it easy to link it to the audio features in the MSDS. The format of the data is in JSON. This dataset comprises about 584,000 tracks that have at least one similar track. The main reason why this dataset was acquired was to make up for the loss of substantial data in the AotM from the data cleaning.

Each song is associated with a ranked list[14] of other songs that are similar to it. We will call this list a *similarity list* for the purpose of this paper. These similarity lists are likened to the playlists in the AotM. Using these similar-songs data is intuitively promising as having each song related to a list of similar songs is almost equivalent to having a seed song related to its generated playlist. Another reason is that this dataset is quite huge: 5801 similarity lists (matched to the MSDS).

No. of songs in similarity list	2	3	4	5	6	7	8	9	10	11 or more
No. of such lists	1607	968	580	305	183	103	56	22	15	17

Table 4.18: Last.fm Similarity List Data

[13] `http://labrosa.ee.columbia.edu/millionsong/lastfm`
[14] No documentation found for the algorithm used to compute this similarity.

4.2.3 Taste Profile Subset

The Taste Profile Subset[15] has over 48 million triplets (user, song, count) describing the listening history of over 1 million users and 380,000 songs. The data-cleaning process (matching to MSDS, removing duplicates, deleting profiles with only one song) leaves us with 3499 unique songs and over 171,000 unique profiles. The most important contribution of this dataset to the project is that it considerably reduces the sparsity of the meta-training matrix in KMT by 10 times.

No. of songs in Taste Profile	2–5	6–10	11–20	21–30	31–40	41–50
No. of such profiles	158,132	11,553	1347	56	6	2

Table 4.19: Taste Profile Subset Data

[15] http://labrosa.ee.columbia.edu/millionsong/tasteprofile

Chapter 5

Method

The machinery behind playlist generation can be perceived as a machine learning problem: the nature of many such applications is such that information or knowledge about the true underlying mechanism behind the data generation process is limited [20]. Instead, one relies on generic 'smoothness' assumptions. For example, we might wish that for two inputs $\boldsymbol{x}$ and $\boldsymbol{x'}$ that are close, the corresponding outputs y and y' should be similar to each other. In statistics, to smooth a data is to create an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures. In the Gaussian process (GP) and Gaussian process regression (GPR) framework, the mathematical smoothness properties of the functions in this respect are well understood, giving confidence in the procedure.

In our application, our data can be represented as a sample from a multivariate Gaussian distribution. There are two statistics that fully describe a GP: the mean and the covariance. We outline below why Gaussian process can be used in playlist generation:

Data size GPR works well on small data [6]. This is relevant to us as our seed songs

are very small (1 to 5 songs).

Parameters The covariance can be specified in any way we want. Here, we would like it to be defined as how similar songs are to each other, acting as a prior. Also, the hyperparameters of a GP, which ultimately control the final form of the Gaussian process, can be estimated from the data itself using a maximum likelihood or Bayesian approach [6].

Regression Regression in GP is simple as it is closed-form and a mere matrix arithmetic — computationally fast for small data. It also automatically models uncertainty in the predictions. Predictions of songs can be ranked and used in a playlist, where order of songs matter.

5.1 Gaussian Process

A stochastic process is a collection of random variables $\{Y(\boldsymbol{x}_1), Y(\boldsymbol{x}_2), ...\}$ where $\boldsymbol{x}$ is the input space of the process with dimension d, the number of inputs. The stochastic process is defined by giving the *probability distribution* for every finite subset of variables $Y(\boldsymbol{x}_1), Y(\boldsymbol{x}_2), ..., Y(\boldsymbol{x}_k)$ in a consistent manner.

A Gaussian process is a type of stochastic process wherein if any finite N of $\boldsymbol{x}_i$'s ($\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_N$) is chosen as the input space, then the N corresponding samples, $y_1, y_2, ...y_N$, drawn from the Gaussian process are jointly Gaussian (normal). Like any Gaussian distribution, a Gaussian process is also specified by its mean $\mu(\boldsymbol{x}) = E[Y(\boldsymbol{x})]$ and covariance function $K(\boldsymbol{x}, \boldsymbol{x}')$.

In our application, $\boldsymbol{x}$ represents a *metadata vector*. Note that a metadata vector is a vector that comprises the 7 categorical variables (depending on which playlist version) as described in Table 3.1. A Gaussian process is placed over the user preference

functions. We will also assume that the mean of user preference function is zero everywhere. This means that at any particular time, a user does not want to listen to most of the songs in the world, leading to a mean preference close enough to zero to be approximated as zero. A zero mean implies that the covariance kernel $K(\boldsymbol{x}, \mathbf{x}')$ simply turns into a correlation over a distribution of functions g where $K(\boldsymbol{x}, \mathbf{x}') = < g(\boldsymbol{x})\, g(\boldsymbol{x}') >$.

5.2 Mood and User Preference Function

The basis for recommendation in this playlist system depends on a user's mood at the point where he or she wants to listen to music. As a person's mood differs from time to time, so would his or her preferences over certain songs. In this aspect, we have taken the literature from Platt et al. [1] to translate 'mood' to *user preference* over songs.

Let f_i be the true underlying user preference for song i. The expressed or observed user preference function for song i (achieved by selecting song i as a seed song) at a particular listening session is defined as

$$y_i = \begin{cases} 1 & \text{if a user likes song } i; \\ 0 & \text{otherwise,} \end{cases}$$

where y_i incorporates a noisy measurement

$$y_i = f_i + \xi_i, \tag{5.1}$$

with $\xi_i \sim \mathcal{N}(0, \sigma^2)$. By default, we assume that selected songs have target f values of 1, while removed songs have target f values of 0. A Gaussian process has been placed over f, where its mean is zero and covariance kernel defined in the next section.

According to [1], even though the y values are binary, Gaussian Process Classifica-

tion is not used, in order to maintain generality and because it requires an iterative procedure to estimate the posterior.

5.3 Covariance Kernel

To start off, we will learn a covariance kernel between 2 songs $Cov(f_i, f_j) \equiv \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ that takes 2 music metadata vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ from song i and song j respectively. We used the same covariance components model in [1] to define κ as follows:

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{n=1}^{N_\psi} \beta_n \psi_n(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sigma^2 \delta_{ij} \tag{5.2}$$

where ψ_n are the pre-defined Mercer kernels, N_ψ is the total number of such kernels (see Section 5.3.1 on Mercer kernels), $\beta_n \geq 0$ are the smoother coefficients and δ_{ij} is the Kronecker function

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j; \\ 0 & \text{if } i \neq j. \end{cases}$$

κ is then calculated among all pairwise combinations of the 9939 songs. Hence the final kernel covariance kernel is the following:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_{9939}) \\ \kappa(\boldsymbol{x}_2, \boldsymbol{x}_1) & \kappa(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & \kappa(\boldsymbol{x}_2, \boldsymbol{x}_{9939}) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{x}_{9939}, \boldsymbol{x}_1) & \kappa(\boldsymbol{x}_{9939}, \boldsymbol{x}_2) & \cdots & \kappa(\boldsymbol{x}_{9939}, \boldsymbol{x}_{9939}) \end{bmatrix} \tag{5.3}$$

As an example, before a user expresses his user preferences for, say, song 2, 7 and 9,

f_2, f_7, f_9 form a joint prior Gaussian:

$$\begin{pmatrix} f_2 \\ f_7 \\ f_9 \end{pmatrix} \sim \mathcal{N}\left[\mathbf{0}, \begin{pmatrix} \kappa(\boldsymbol{x}_2, \boldsymbol{x}_2) & \kappa(\boldsymbol{x}_2, \boldsymbol{x}_7) & \kappa(\boldsymbol{x}_2, \boldsymbol{x}_9) \\ \kappa(\boldsymbol{x}_7, \boldsymbol{x}_2) & \kappa(\boldsymbol{x}_7, \boldsymbol{x}_7) & \kappa(\boldsymbol{x}_7, \boldsymbol{x}_9) \\ \kappa(\boldsymbol{x}_9, \boldsymbol{x}_2) & \kappa(\boldsymbol{x}_9, \boldsymbol{x}_7) & \kappa(\boldsymbol{x}_9, \boldsymbol{x}_9) \end{pmatrix}\right]. \tag{5.4}$$

5.3.1 Mercer Kernels ψ

We reproduced the family of kernels that Platt et al. has designed: Mercer kernels [1]. The purpose of these kernels is to check the similarity between any two vectors whose elements are discrete in nature. As we had some categorical data (e.g. genre, *time signature* etc.) and transformed the continuous ones, this kernel usage would be convenient for us. We believe that checking all the possible combinations of vector positions for similarity is a primitive, unsophisticated yet reasonable procedure to check if 2 vectors are similar. Furthermore, it is computationally easy.

Let $\boldsymbol{a}_n$ be the 7-digit (7 is the length of the metadata vector) binary representation of the decimal number n (e.g. $\boldsymbol{a}_5 = 0000100$ and $\boldsymbol{a}_6 = 0000101$). Where h is the vector index or position within any vector, this family of Mercer kernels is defined as

$$\psi_n(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} 1 & \text{if } x_h = x'_h \text{ for every } h \text{ such that } a_{nh} = 1, \\ & \text{or if } a_{nh} = 0 \text{ for all } h; \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

In words, for all the indices h in the vector $\boldsymbol{a}_n$ that are 1, the h^{th} components of vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ must match in order for the output of the kernel to be 1. By way of illustration, $\psi_{90}(\boldsymbol{x}, \boldsymbol{x}')$, which corresponds to $\boldsymbol{a}_{90} = 1011001$, checks the 1^{st}, 3^{rd}, 4^{th} and 7^{th} positions

of both $\boldsymbol{x}$ and $\boldsymbol{x}'$. If, indeed, at these positions both vectors have the same values, then $\psi_{90}(\boldsymbol{x}, \boldsymbol{x}') = 1$, otherwise 0.

Where l is the length of any vector $\boldsymbol{y}$, note that these Mercer kernels were constructed such that there are 2^l number of ways to check between the positions between two vectors. Since the ψ kernels operate on music metadata vectors $\boldsymbol{x}$ of length 7, there are $2^7 = 128$ of such ways. So, h runs from 1 to 7 whereas n runs from 1 to $N_\psi = 128$, the total number of Mercer kernels.

The setup in [1] used 7 metadata fields. We also maintained the same number of metadata fields as Platt et al. has set a 'benchmark' by demonstrating that the learned kernel is shown to be more effective than a reasonable hand-designed kernel. We did not increase this number as using a small N_ψ forces a smoother, more general similarity metric [1].

5.3.2 Fitting β

As carried out by [1], to fit the β in (5.2), we fitted the covariance components model to an empirical covariance by applying a least-square distance function [1]:

$$\underset{\beta_n}{\operatorname{argmin}} \frac{1}{2} \sum_{i,j} \left[\mathcal{K}_{ij} - \sum_{n=1}^{N_\psi} \beta_n \psi_n(\mathbf{x}_i, \mathbf{x}_j) \right]^2 \tag{5.6}$$

where i and j index all of the samples in the meta-training dataset, and where

$$\mathcal{K}_{ij} = \frac{1}{M} \sum_{m=1}^{M} q_m(\mathbf{x}_i) \cdot q_m(\mathbf{x}_j) \tag{5.7}$$

is the empirical covariance. The empirical covariance is indeed our meta-training kernel (to be explained more in Section 5.3.3). The aim here is to fix the β coefficients

such that the final covariance kernel K will appear similar to $\mathcal{K}$. For this reason, the β are also called *smoother coefficients*. Note that since K is a correlation matrix, it is necessary for entries in $\mathcal{K}$ to be between 0 and 1, but not necessary for its diagonal entries to be the same value because ultimately, the diagonal entries in (5.3) will still have the same values.

The loss function in (5.6) is the square of the Frobenius norm[1] of the difference between $\mathcal{K}_{ij}$ and the covariance components model. The use of Frobenius norm has been proven to be consistent in [1]. An advantage of using Frobenius norm as Platt et al. has mentioned is that the loss function in (5.6) produces an easy-to-solve quadratic program.

5.3.3 Meta-Training Kernel $\mathcal{K}$

Kernel meta-training (KMT) was introduced by Platt et al. [1] to improve GPR by adding an additional phase of learning: meta-training. A *meta-training kernel* $\mathcal{K}$ is learned even before any training examples are available. This kernel is learned from a set of samples from meta-training functions that are drawn from the same function distribution, which will eventually generate the training function [1].

Platt et al. described albums as "professionally designed playlists", hence they used albums to generate a prior for playlist generation. For the meta-training function q_m, they used album indicator functions that are 1 for songs on an album m and 0 otherwise. Thus, KMT learns a similarity metric that professionals use when they assemble albums. Also, it was also claimed that this same similarity metric "empirically makes consonant playlists".

[1]The Frobenius norm of a matrix A is defined as $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|}$, i.e. taking the square root of the sum of squared elements in the matrix.

Since the album indicator functions are uniquely defined for songs, not for meta-data vectors, one cannot simply generate a kernel matrix according to (5.7). Rather, we generate a meta-training kernel using meta-training functions that depend on songs:

$$\mathcal{K}_{ij} = \frac{1}{A} \sum_{a=1}^{M} g_a(i) \cdot g_a(j) \tag{5.8}$$

where

$$g_a(i) = \begin{cases} 1 & \text{if song } i \text{ belongs to album } a; \\ 0 & \text{otherwise,} \end{cases}$$

and where A is the total number of unique albums in the MSDS.

However, using MSDS albums, we found that the 9939×9939 $\mathcal{K}$ matrix was too sparse: only 0.0165% of the entries were non-zeros. A severe implication is that the final covariance kernel for the GP might resemble that of the $\mathcal{K}$ matrix, suggesting that many songs are not very much similar to each other. There are 2 possible explanations why such sparsity occurs in $\mathcal{K}$ using MSDS albums. Firstly, the 10,000 songs chosen randomly from the 1 million songs coincidentally do not have many songs that share the same album. Secondly, all songs belong to only 1 album at most, and this leads to a similarity matrix that only has 0 or 1 as its entries. As a result, the final covariance kernel shows either a 'yes' or a 'no', i.e. each pair of songs is either similar or not similar at all. This is not a favourable outcome, as we want a range of values of similarity (from 0 to 1) such that we will be able to rank songs in decreasing order of f, the user preference function.

In this project, in order to reduce sparsity, we modified the KMT by adding three meta-training datasets to the meta-training kernel such that we have 4 in total: MSDS albums, AotM, Last.fm and Taste Profile. The modified meta-training kernel is now

interpreted as follows:

$$\mathcal{K}_{ij} = \frac{1}{A+B+C+D}\Big[\sum_{a=1}^{A} g_a(i)g_a(j) + \sum_{b=1}^{B} g_b(i)g_b(j) + \sum_{c=1}^{C} g_c(i)g_c(j) + \sum_{d=1}^{D} g_d(i)g_d(j)\Big], \tag{5.9}$$

where B, C and D are the total number of unique playlists in AotM, total number of unique similarity lists in Last.fm and total number of unique profiles in Taste Profile respectively and where

$$g_b(i) = \begin{cases} 1 & \text{if song } i \text{ belongs to AotM playlist } b; \\ 0 & \text{otherwise,} \end{cases}$$

$$g_c(i) = \begin{cases} 1 & \text{if song } i \text{ belongs to Last.fm similarity list } c; \\ 0 & \text{otherwise,} \end{cases}$$

$$g_d(i) = \begin{cases} 1 & \text{if song } i \text{ belongs to Taste Profile } d; \\ 0 & \text{otherwise.} \end{cases}$$

In words, the $\mathcal{K}_{ij}$ entry is simply the number of times both songs i and j appear together in the 4 meta-training data, divided by $(A+B+C+D)$. Such scaling is performed so that the entries in $\mathcal{K}$ are between 0 and 1, since $\mathcal{K}$ is set to mimic the covariance kernel K, which is a correlation matrix. Note that $\mathcal{K}$ itself is not a correlation matrix, thus the diagonal entries may not have to be the same.

From Table 5.1, we can see that the Last.fm data improved the sparsity by about 0.03% and the Taste Profile by about 0.43%. However, notice that the AotM decreased sparsity by only about 0.0015%. This AotM data seemed very insignificant for our use. However, we decided to keep this data instead of discarding it because about 700 pairs of songs had new entries due to the addition of this data. Furthermore, the AotM data

provides real naturally-occurring playlists — the best prior for playlist generation.

Type of $\mathcal{K}$	Meta-Training Data				% Sparsity of $\mathcal{K}$
	MSDS Albums	AotM	Last.fm	Taste Profile	
	✔				0.0165
	✔	✔			0.0180
	✔	✔	✔		0.0463
	✔	✔	✔	✔	0.442

Table 5.1: Sparsity of Meta-Training Kernel $\mathcal{K}$

It is hoped that by merging these 4 datasets together, a better prior can be formed for the covariance kernel K. We postulate that a huge advantage of using several (or at least 2) meta-training datasets is so that the KMT is not biased to a certain attribute. For instance, if we only used one dataset that only contains popular songs, eventually our playlist generator would mostly suggest popular songs. In this paper, the meta-training data is a combination of both popular (Taste Profile) and similar (MSDS album, Last.fm, Taste Profile) songs and real playlist data (AotM, Taste Profile).

5.3.4 Final Fitted Covariance Kernel K

Now that the meta-training kernel $\mathcal{K}$ and the Mercer kernels ψ are in place, our final step is back to fitting the β_n in (5.6). We first differentiate it with respect to β_m:

$$\frac{dE}{d\beta_m} = -\sum_{i,j}\left[\mathcal{K}_{ij} - \sum_{n=1}^{N_\psi}\beta_n\psi_n(\boldsymbol{x}_i,\boldsymbol{x}_j)\right]\psi_m(\boldsymbol{x}_i,\boldsymbol{x}_j) \tag{5.10a}$$

$$= -\sum_{i,j}\mathcal{K}_{ij}\psi_m(\boldsymbol{x}_i,\boldsymbol{x}_j) + \sum_{n=1}^{N_\psi}\beta_n\sum_{i,j}\psi_n(\boldsymbol{x}_i,\boldsymbol{x}_j) \tag{5.10b}$$

$$= -\frac{1}{M}\sum_{(i,j)\in Q}\psi_m(\boldsymbol{x}_i,\boldsymbol{x}_i) + \sum_{n=1}^{N_\psi}\beta_n\sum_{i,j}\psi_n(\boldsymbol{x}_i,\boldsymbol{x}_j)\psi_m(\boldsymbol{x}_i,\boldsymbol{x}_j), \tag{5.10c}$$

where Q is the set of all (i,j) pairs such that $\mathcal{K}_{ij} = 0$. We note that there is a typo-

graphical error in [1]: (5.10c), as seen above, is written without the coefficient $-\frac{1}{M}$ on the first term in the paper.

We reproduced the arithmetic simplification as in [1]: the first term in (5.10c) has been simplified to only take into account the (i, j) pairs in $\mathcal{K}$ that are non-zero, denoted by Q (due to the sparse $\mathcal{K}_{ij}$ matrix), the second term was estimated by sampling a random subset of (i, j) pairs i.e. generating 100 random j for every $i = 1, 2, ..., 9939$. A sum over all i and j is computationally impractical.

To compute (5.10c), we first expressed it in a matrix form and used a quadratic programming solver in R, `solve.QP()`, in the R package `quadprog` to compute the quadratic function. The matrix form is as follows:

$$-\mathbf{d}^T\mathbf{b} + \frac{1}{2}\mathbf{b}^T D\mathbf{b} \tag{5.11}$$

with the constraint $A'\mathbf{b} \geq \mathbf{b}_0$, where

$$\mathbf{d} = \frac{1}{M}\begin{pmatrix} \sum_Q \psi_1(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \sum_Q \psi_2(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \vdots \\ \sum_Q \psi_{N_\psi}(\boldsymbol{x}_i, \boldsymbol{x}_j) \end{pmatrix}, \tag{5.12a}$$

$$D = \begin{bmatrix} \sum_{i,j} \psi_1(\mathbf{x}_i, \mathbf{x}_j)\psi_1(\mathbf{x}_i, \mathbf{x}_j) & \cdots & \sum_{i,j} \psi_1(\mathbf{x}_i, \mathbf{x}_j)\psi_{N_\psi}(\mathbf{x}_i, \mathbf{x}_j) \\ \sum_{i,j} \psi_2(\mathbf{x}_i, \mathbf{x}_j)\psi_1(\mathbf{x}_i, \mathbf{x}_j) & \cdots & \sum_{i,j} \psi_2(\mathbf{x}_i, \mathbf{x}_j)\psi_{N_\psi}(\mathbf{x}_i, \mathbf{x}_j) \\ \vdots & \vdots & \vdots \\ \sum_{i,j} \psi_{N_\psi}(\mathbf{x}_i, \mathbf{x}_j)\psi_1(\mathbf{x}_i, \mathbf{x}_j) & \cdots & \sum_{i,j} \psi_{N_\psi}(\mathbf{x}_i, \mathbf{x}_j)\psi_{N_\psi}(\mathbf{x}_i, \mathbf{x}_j) \end{bmatrix}, \tag{5.12b}$$

$\mathbf{b} = (\beta_1, \beta_2, ..., \beta_{N_\psi})^T$, $\mathbf{b}_0 = \mathbf{0}_{N_\psi}$ and $A = \mathbf{I}_{N_\psi}$. In this project, we have rounded off machine epsilons to zero for β values that are less than 10^{-10}.

With these $\hat{\beta}$ optimised values, we fitted them to the covariance kernel in (5.3). This whole KMT procedure was carried out for different metadata inputs $\boldsymbol{x}$ (different sets of metadata fields) from v3.0 to v3.9.

Interpretation of $\hat{\beta}$

Currently, to the best of our knowledge, there is no literature for the analysis of $\hat{\beta}$ in the Mercer kernels defined by Platt et al. Even Platt et al. have not investigated them in [1]. We adopted a naïve but empirical approach to determine the weights of the 7 metadata fields in the input vector $\boldsymbol{x}$, i.e. the 'usefulness' or importance of every metadata field in $\boldsymbol{x}$. We define the *metadata weight* ω_h for the position h of the $\boldsymbol{x}$ vector as

$$\omega_h = \sum_{n=1}^{N_\psi} \hat{\beta}_n a_{nh} \tag{5.13}$$

where a_{nh} is the h^{th} entry on $\boldsymbol{a}_n$.

We first provide a basic interpretation of β. Recall that $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{n=1}^{N_\psi} \beta_n \psi_n(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Let us take β_6, which corresponds to $\boldsymbol{a}_6 = 0000101$ (defined in Section 5.3.1 Mercer Kernels). If $\hat{\beta}_6 \approx 0$ in the optimisation, it possibly indicates that 2 supposedly similar songs, defined in the meta-training $\mathcal{K}$, do not have common features at the 5^{th} and 7^{th} positions on the metadata vector.

Now for every $\boldsymbol{a}_n$ we multiply the corresponding $\hat{\beta}_n$ to it. We stack in rows all the $\hat{\beta}_n \boldsymbol{a}_n$ to obtain a matrix whose columns add up to ω_h. Suppose $\hat{\beta}_n$ only takes a value of 1 or 0. The value ω_h is the number of 1's at down the column h, signifying the number of times the metadata vector at position h is useful. We now extend this concept to $\hat{\beta}_n \geq 0$: ω_h is the weight of the metadata field at position h, to be used in comparison with the other values in other positions.

5.4 Gaussian Process Regression

At this stage, we are ready to perform GPR. Suppose now the user has already expressed his preferences for songs 3 and 7 (by selecting songs 3 and 7 as seed songs); i.e. $y_3 = 1$ and $y_7 = 1$. We want to predict his preference for song number $*$. Gaussian process tells us that (f_3, f_7, f_*) forms a multivariate normal distribution prior as follows:

$$\begin{pmatrix} f_3 \\ f_7 \\ f_* \end{pmatrix} \sim \mathcal{N}\left[\mathbf{0}, \left(\begin{array}{cc|c} \kappa(\boldsymbol{x}_3, \boldsymbol{x}_3) & \kappa(\boldsymbol{x}_3, \boldsymbol{x}_7) & \kappa(\boldsymbol{x}_3, \boldsymbol{x}_*) \\ \kappa(\boldsymbol{x}_7, \boldsymbol{x}_3) & \kappa(\boldsymbol{x}_7, \boldsymbol{x}_7) & \kappa(\boldsymbol{x}_7, \boldsymbol{x}_*) \\ \hline \kappa(\boldsymbol{x}_*, \boldsymbol{x}_3) & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_7) & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{array}\right)\right] \tag{5.14}$$

also written as

$$\begin{pmatrix} \boldsymbol{f} \\ f_* \end{pmatrix} \sim \mathcal{N}\left[\mathbf{0}, \begin{pmatrix} M & M_*^T \\ M_* & M_{**} \end{pmatrix}\right], \tag{5.15}$$

where

$$M = \begin{bmatrix} \kappa(\boldsymbol{x}_3, \boldsymbol{x}_3) & \kappa(\boldsymbol{x}_3, \boldsymbol{x}_7) \\ \kappa(\boldsymbol{x}_7, \boldsymbol{x}_3) & \kappa(\boldsymbol{x}_7, \boldsymbol{x}_7) \end{bmatrix}, \tag{5.16a}$$

$$M_* = \begin{bmatrix} \kappa(\boldsymbol{x}_*, \boldsymbol{x}_3) & \kappa(\boldsymbol{x}_*, \boldsymbol{x}_7) \end{bmatrix} \text{ and} \tag{5.16b}$$

$$M_{**} = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*). \tag{5.16c}$$

We are certainly interested in the conditional probability $p(f_*|\boldsymbol{f} = \boldsymbol{y} = (y_3, y_7))$: given the training examples $y_3 = 1$ and $y_7 = 1$, how likely is a certain prediction for f_*? Fortunately, there is a closed-form expression for this, as it is one of the justifications for the use of Gaussian process in this project. The probability $p(f_*|\boldsymbol{y})$ follows a

Gaussian distribution:

$$f_*|\boldsymbol{y} \sim \mathcal{N}\big(M_* M^{-1}\boldsymbol{y},\ \ M_{**} - M_* M^{-1} M_*^T\big) \tag{5.17}$$

whose posterior mean and variance are

$$\bar{f}_* = M_* M^{-1}\boldsymbol{y} \tag{5.18a}$$

$$Var(f_*) = M_{**} - M_* M^{-1} M_*^T. \tag{5.18b}$$

5.4.1 Noise

To obtain the posterior mean $\bar{f}_*$, we must first solve the free hyperparameter ξ. The hyperparameter is selected via maximum likelihood on the training data $\boldsymbol{y}$ [1]. This means maximising $p(\boldsymbol{y}|\xi)$, where $\boldsymbol{y}$ is our training data. As the data follows a multivariate normal distribution, we have

$$p(\boldsymbol{y}|\xi) = \frac{1}{(2\pi)^{N/2}\,|K|^{-1/2}} e^{-\frac{1}{2}\boldsymbol{y}^T K^{-1}\boldsymbol{y}} \tag{5.19a}$$

$$\log p(\boldsymbol{y}|\xi) = \frac{1}{2}\log|K| - \frac{1}{2}\boldsymbol{y}^T K^{-1}\boldsymbol{y} - \frac{N}{2}\log 2\pi, \tag{5.19b}$$

where N is the size of the training data. We note that there is a typographical error in [1] in the formula for the log-likelihood (a missing inverse sign on A of the first term in the log-likelihood formula). Every time a training data is entered, different values of σ are evaluated and the σ that generates the highest log-likelihood is used. We used the `optimx()` function in the `optimx` package, using the limited-memory Broyden-Fletcher-Goldfarb-Shanno-Bounds (L-BFGS-B) algorithm, since the ξ has a lower bound of 0.

5.4.2 Posterior $\bar{f}$

Finally, we are ready to generate a playlist. Again, we followed the method used in [1] to generate playlists by computing $\bar{f}$ for every song that the user owns. The songs are then ranked in descending order of $\bar{f}$ such that the first few songs the user listens to have high values of $\bar{f}$ while songs later in the playlist have lower values of $\bar{f}$, probably those which the user might not enjoy listening.

In this project, because of the sparsity of the data, several repeated $\bar{f}$ values within playlists were observed upon testing. The order of the songs that have such values will be randomly permuted within. Au2seed will also cut off when the last few songs in the list have the same $\bar{f}$.

Chapter 6

Evaluation

The evaluation of a playlist generation system presents a number of challenges due to the details of the task. Playlist evaluation can be broadly grouped into three categories: sequence prediction, human evaluation and semantic cohesion [4]. We provide the first two in the form of quantitative and qualitative analyses respectively.

6.1 Quantitative Evaluation

In this section, we evaluate playlist generation by sequence prediction. Given a user's preference (or generally, some contextual query), the algorithm must predict which song to play next. The algorithm is then evaluated on the grounds of its prediction, under some notion of correctness [4]. For this section, we will use the word 'playlist' to also refer to a similarity list from Last.fm or a Taste Profile.

6.1.1 Validation Set

We acquired a validation set $\mathcal{V}$ from the meta-training datasets for which we could use as a basis of comparison among the different playlist versions. Note that $\mathcal{V}$ is a substitute for hand-designed playlists in [1] because it would not be very practical to invite someone to look through the 9939 songs in the MSDS database and create a playlist out of it.

We derived $\mathcal{V}$ from the AotM, Last.fm and Taste Profile. The MSDS albums, though used in the KMT, were not used because we prefer real playlist data to validate with. Platt et al. had 60 playlists designed from users in Microsoft Research. Although the lengths of the playlists are unknown, we believe they were roughly at least 10 songs in each playlist. Since the lengths of our playlists are small, we decided to use more than 60 playlists for $\mathcal{V}$.

The number of samples taken from each of the 3 datasets for $\mathcal{V}$ is almost proportional to the number of playlists each data has contributed to the KMT. In total, there were 112 playlists with an average of 11 songs per playlist:

AotM 3 playlists; all with 3 songs each

Last.fm 5 similarity lists; number of songs range from 5-9

Taste Profile 104 Taste Profiles; number of songs range from 10-18

Except for the AotM dataset, we chose 'playlists' with more songs. Longer playlists (about 7 or more) would be more representative of the real world. Note that a random stratified sampling was used to obtain the 'playlist' from the last 2 datasets. For the AotM dataset, we picked 3 playlists from those with 3 songs because picking from those 2 songs is not so practical and picking from those with 4 songs meant that we would lose data for KMT (the pool of playlists with 4 songs is very small).

6.1.2 Experiment

We repeated the same experiment in [1] but with 5 experiments, instead of 7, each with a different number of seed songs. Let the number of seed songs be S, where $S = \{1, 2, ..., 5\}$. By intuition, we chose 5 as the maximum number of seed songs to test because the average number of songs in $\mathcal{V}$ is 11. There was no guide to the maximum number of seed songs that should be used but we took reference from Platt et al.'s work. Assuming the playlists that they have for the validation set have a mean of 15 songs, it means they have halved the number of it to obtain about 7 songs to used as the maximum number of seed songs for experiments in evaluation.

In each experiment there were 1000 trials. Each trial j chose one playlist j (out of all the playlists with at least S+1 songs) at random from $\mathcal{V}$. Then, S songs were chosen at random to be the *training set* from this playlist j. This whole process emulates a user choosing seed songs.

The remaining songs in playlist j, plus one set of 150 random songs[1] from the MSDS, formed one *test set*. For every trial j, we have 5 different sets of 150 random songs, each named as η. This is done to ensure any bias in the randomised song libraries can be cancelled out by taking the average. This test set thus emulates all possible songs available to the playlist generator.

6.1.3 Standard Collaborative Filtering Metric

To score the playlist generators, we used the same evaluation metric used in [1] and in [8], which is essentially a ranked scoring system. The reason for the choice of this metric is that this ranked scoring is suitable as our votes (in this paper: observed

[1] We tried with the whole MSDS initially but it gave us very poor results. This is expected as it would be very hard and unfair for the playlist generator to pick a playlist from a pool of about 9939 songs to form a playlist.

user preferences) are binary (either 0 or 1) and our songs are ranked (top songs in the playlist would be preferred by the user). This ranked scoring estimates the expected utility of a particular ranked list to a user, using an estimate of the probability that a user will see a recommendation in an ordered list. In our application, this ranked list is the playlist itself, whose songs are ranked according to how the user might like them.

The rank score of playlist for trial j is defined as

$$R_j = \frac{1}{5} \sum_{\eta=1}^{5} R_{\eta j} \tag{6.1}$$

where $R_{\eta j}$ is the rank score of playlist using the song library η

$$R_{\eta j} = \sum_{i=1}^{N_{\eta j}} \frac{t_{ij}}{2^{(i-1)/(\alpha-1)}} \tag{6.2}$$

and where t_{ij} is the user preference of the ith element of the jth playlist (1 if ith element is on playlist j, 0 otherwise), α is the half-life of user interest in the playlist, and $N_{\eta j}$ is the number of test songs in song library η for playlist j. Half-life is the number of songs on a playlist such that there is a 50-50 % chance the user will review that item (for these experiments, we used a half-life of 10 songs to ensure our baseline results match with those in [1]). This score is summed over all 1000 trials, and normalised:

$$R = 100 \sum_{j=1}^{1000} R_j \;\; / \;\; \sum_{j=1}^{1000} R_j^{max} \tag{6.3}$$

where R_j^{max} is the maximum score attainable in (6.1) (i.e., if all of the true songs in the playlist were at the top of the playlist.) Thus, an R score of 100 indicates a perfect prediction. The normalisation above allows us to consider results independent of the size of the test set and number of songs predicted in a given experiment.

6.1.4 Results

Version	No. of Seed Songs				
	1	2	3	4	5
v3.0	**22.26**	**22.45**	**21.21**	**20.75**	**19.50**
v3.1	**23.20**	**22.75**	**20.79**	**20.80**	**19.91**
v3.2	21.38	21.49	20.28	20.19	19.25
v3.3	19.03	19.08	18.96	18.93	18.49
v3.4	19.19	18.89	18.04	17.98	17.38
v3.5	18.72	18.74	18.34	18.51	17.88
v3.6	**23.54**	**22.23**	**21.19**	**20.90**	**20.39**
v3.7	19.57	19.07	18.66	18.51	18.05
v3.8	18.09	18.71	18.58	18.57	18.16
v3.9	20.10	19.17	18.87	18.75	18.29
Random	9.29	8.95	7.88	6.61	6.05

Table 6.1: *R* Scores for Different Playlist Versions. The higher the score, the better it is. Boldface indicates best versions with statistical significance level less than 0.05.

Table 6.1 shows the results of the standard collaborative filtering metric evaluation for each version. We refer the reader to Table 3.1 on page 14 for the features used in each playlist version. To compare which version works the best among the 10, we carried out a pairwise Wilcoxon signed rank test. A boldface result shows the best versions (v3.0, v3.1 and v3.6) based on this test with a significance level of 0.05.

Note that we adjusted certain parameters like the half-life in order to reproduce the baseline random results in [1]. However, we remark that our results are almost incomparable to theirs, mainly due to the different data used. One can, nonetheless, casually observe that the (KMT + GPR) method in [1] is about 7 times better than their random playlist generator, while ours is 2 to 3.5 times better.

Playlist generators better than random

Here we would like to highlight that all the playlist versions perform much better than random. Hence, we can safely say that these playlist generators indeed capture some notion of playlist generation. This is probably due to the work that went into designing the metadata schema [1]. All playlist versions perform about 2 to 3.5 times better than a random song generator. One can see this statistic in another perspective too: for every time a playlist generator creates a sound playlist, a random playlist generator needs to re-create 2 more playlists to get the same effect. That being so, we cannot conclude that the Au2seed versions perform better or worse than AutoDJ.

More timbre values does not mean better results

Among all the versions, only v3.0 uses only 1 timbre value: *segment_loudness*. The fact that v3.0 performs better than most versions is unexpected: we had guessed that the more the timbre values used as metadata, the better it might be. Clearly, this idea is debunked on observing that v3.0 performs better than v3.3, whose metadata fields are all timbre values. A possible explanation for this is that there are certain elements that the timbre values cannot capture. Perhaps the textual metadata (genre) incorporates certain components of music similarity that timbre values cannot record.

Decreasing score trend

Across all playlist versions (and including the random playlist generator), as the number of seed songs increases, the R score decreases. One plausible explanation for this is due to the evaluation procedure itself: it is harder to score well if the playlist generator has to only pick one song to complete the playlist, as compared to having to pick 10 more songs. This trend could also imply 'saturation', as observed by Platt et al.

Their R scores made little increase nor decrease as the number of seed songs, noting the fact that exact playlists are hard to predict as there are so many appropriate songs that would be valid in a test playlist even if the user did not choose those songs [1].

Recommendation: v3.6

To compare v3.0, v3.1 and v3.6, we contrasted their metadata weights ω obtained from optimising β earlier (Section 5.3.4).

	v3.0	v3.1	v3.2	v3.3	v3.4	v3.5	v3.6	v3.7	v3.8	v3.9
Genre	●	●	●							★
Key	△									
Mode	△	△			△				△	
Sections										
Tempo		△	△		△	△		△		
Time signature	★		★		★	●	●			
Segment_loudness	●	●	●	●	●	●		●	●	●
Brightness		★	△	△	●	△			★	△
Flatness				△			●	●		●
Attack						△	△	△	△	
Timbre5				●		★	★	★	●	
TImbre6							△			△
Timbre7				★						

Table 6.2: Metadata Weights ω for Each Playlist Version. A shaded cell denotes the feature used for the particular version. For each version, ● correspond to the highest 2 weights, ★ denotes the third highest weight, and △ corresponds to the lowest 2 weights.

For a particular playlist version, we bound the rows of all $\boldsymbol{a}_n$ (defined in Section 5.3.1 Mercer Kernels), whose corresponding $\hat{\beta}_n$ were more than 10^{-10} (machine zeroes), into a matrix. Then each row n was multiplied by the corresponding $\hat{\beta}_n$ weight. Summing the weights for every column means adding the weights for every metadata position, in a superficial manner. Finally, these weights were ranked in decreasing order. Table 6.2, shows the metadata weights for every version.

Across the versions, it is easy to see that metadata like *mode* and *tempo* have much lower weights compared to *segment_loudness*, *time signature*, genre and *timbre5*. As v3.6

includes the features *segment_loudness*, *time signature* and *timbre5*, we recommend this version over v3.0 and v3.1.

High performance for *segment_loudness*

We refer the reader to the interpretation of *segment_loudness* in the timbre subsection of Section 4.1.4, page 22. One conceivable reason why *segment_loudness* is performing well is that the general loudness of a song and its variability throughout the song can group songs together well. Take for example slow and 'sad' songs: we would find that its mean loudness for the particular song is low and remains low throughout. Certain upbeat pop songs would have generally high loudness but very varied loudness throughout. One possible example of such a song: certain verses with normal loudness, chorus having the greatest loudness and only one verse with very low loudness.

Poor performance for *tempo*

The poor performance of *tempo* was unexpected and quite counter-intuitive because one might think that 'fast' songs would be similar to each other. We reviewed the *tempo* data by selecting several songs with suspiciously very high BPM, listened to them on YouTube[2] and re-obtained the data (provided by The Echo Nest) from a website[3]. We also selected another 5 songs from the MSDS at random and checked their tempos.

From Table 6.3, we can see that the true tempo is more than the MSD tempo (tripled for some songs). Upon listening to the songs, we observed that some of these songs have triplets within a beat, which the audio processor might have falsely interpreted as individual beats. One can also observe that the tempo of Song 2 in Table 6.4 is incorrect. In conclusion, the *tempo* data in MSD is not accurate and should not be used.

[2] http://www.youtube.com
[3] https://songbpm.com

	Song Title	Artist	Tempo in MSDS	Songbpm.com
1	Been There All The Time	Dinosaur Jr.	253.357	164
2	Severe Severing	Klaus Badelt	246.593	84
3	Ego is the Drug/3am	The Frequency	246.5	83
4	I Only Have Eyes For You	John Stevens	244.36	80
5	Empress So Divine	Warrior King	241.877	81

Table 6.3: MSDS Songs with Very High Tempos

	Song Title	Artist	Tempo in MSDS	Songbpm.com
1	Still I Will	Lisa Lynne	106.009	106
2	Dans le Calme de la Nuit	Kali	120.406	90
3	La Nit de Reis	Gertrudis	111.953	112
4	So Say I	Mr Brown	140.252	140
5	La Vida Eres	Orquesta Sublime	134.923	135

Table 6.4: Tempos of 5 Songs from MSDS Selected at Random

Suggested metadata for playlist generation for future works

From the discussion above and based on observation, we determined the following 7 metadata to be the used for Mercer kernels (in decreasing order of importance): *segment_loudness*, *time signature*, genre, *timbre5*, *flatness*, *timbre7* and *brightness*. We also advise against the use of 2 MSD metadata, namely *mode* and *tempo* as the metadata weights of most versions are the least.

6.2 Qualitative Evaluation

6.2.1 Human Evaluation

Since the eventual goal of playlist algorithms is to improve user experience, the ideal method of testing Au2seed v3.6 is to directly measure human response [4]. We invited 9 test subjects to participate in this evaluation where we created a playlist for each person. This required their entire music libraries and the song features used in v3.6.

However, obtaining the features of every song would take too much time. Hence we settled for a smaller sample from their music library. We asked each subject to:

- pick about 20–50 songs from his/her music library where
  - the first 5–10 songs are created based on a certain mood
  - the rest of the songs is a random selection from the library
- label the playlist by mood (workout, sad, going to work, etc.)

We also cautioned them that the sample library should not have any duplicates, classical songs, and preferably of different genres.

If an individual has an iTunes library[4], a list of song titles and artist names can be retrieved quite elegantly by copying and pasting onto an Microsoft Excel sheet, facilitating the process of data collection.

For every song provided to us, we entered the song title and artist name as its parameters to look up its 7 song features in The Echo Nest API[5] using an API key. Using these vectors, for every individual, we calcu-

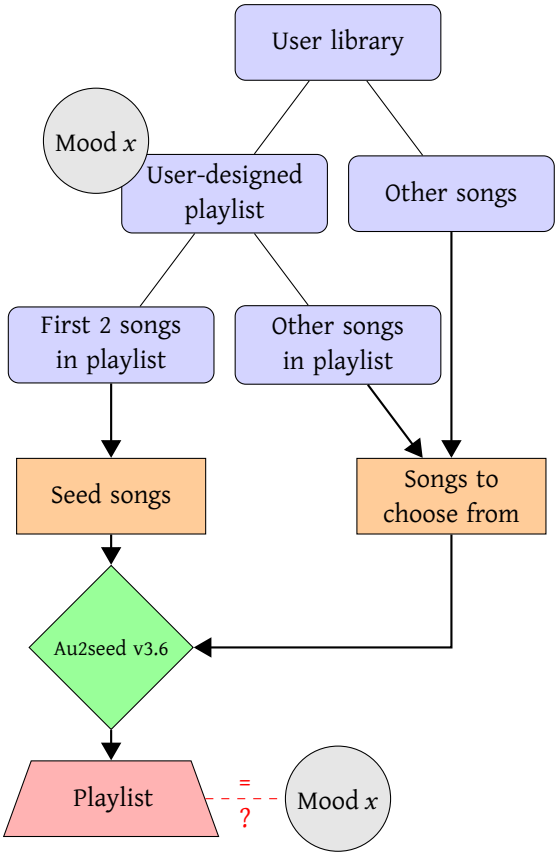

Figure 6.1: Au2seed v3.6 Evaluation

[4]https://www.apple.com/sg/itunes/

[5]For many popular songs, there were more than one result (different versions of the songs, different styles of naming songs etc.). We picked the first one for convenience.

lated the covariance kernel seen in (5.3) on page 36 using the $\hat{\beta}$ values for v3.6. The tracks that each individual has submitted to us form the (sampled) user music library. Note that we have made a necessary assumption for this playlist generator: the $\hat{\beta}$ in (5.2) are fixed for the entire universe.

For every subject i, we picked the first 2 songs from his or her playlist and fed them into Au2seed. Au2seed recommended songs from the user library of subject i. He or she was to indicate how many tracks in the generated playlist *do not fit* that playlist, according to the mood from which it was created. The playlist score is defined as

$$P = \frac{\text{\# songs that 'belong' in the playlist}}{\text{total \# songs generated}} \times 10. \tag{6.4}$$

If P was less than 7, we removed the songs that user deemed did not fit the playlist, generated another playlist, and obtained another score.

User Feedback

In general, we received positive feedback from our test subjects for the playlists we generated for them. We were also contented to know that the users were quite satisfied with the playlists generated. Appendix B shows the same table in Table 6.5 including the users' comments.

Subject	No. of Songs	Name of Playlist	Raw Score	P Score	Overall User Satisfaction (1-10)
Serene	16	emo	5/8 8/8	6.25 10	7
Azmi	30	gym	11/12	9.2	9
Jing Da	20	clubbing	6/9	6.7	7
Fahmi	50	workout	9/22 18/22	4.1 8.2	7
Ben	22	music for the soul	11/12	9.2	8.5
Wei Qi	22	emo	6/7	8.6	8
Farah	28	workout	9/12	7.5	9
Anna	25	workout music	5/11	4.5	7.5
Shaz	37	going to work	9/12	7.5	7

Table 6.5: User P Scores and Ratings

Limitations of Experiment

We acknowledge that the sample user library was small for some subjects. As a result, if the songs in the user library happened to be similar to the songs in the playlist, it would be easier to score P. The converse is also true. However, we remark that good results can be obtained with a library of about 30 songs or more (See Table 6.5).

6.2.2 Sample Playlists

	Song Title	Artist
Seed	Clarity	Zedd
Seed	Apollo	Hardwell
1	I Need Your Love	Ellie Goulding
2	Safe & Sound	Taylor Swift
3	Red	Taylor Swift
4	Love Is Gone	David Guetta

Table 6.6: Electronic Dance Music as Seed Songs

	Song Title	Artist
Seed	Lose Yourself	Eminem
Seed	Mockingbird	Eminem
1	Rap God	Eminem
2	Stereo Hearts ft. Adam Levine	Gym Class Heroes
3	When I'm Gone	Eminem
4	Safe And Sound	Capital Cities

Table 6.7: Seed Songs (Rap) from Eminem

We have tested Au2seed v3.6 for ourselves to get a first-hand experience on music playlists generated from this version, using our own music library of about 500 songs. The results were very encouraging. (Note that red font indicates that the song is not deemed to be fitting to the playlist.)

	Song Title	Artist
Seed	Sleep All Day	Jason Mraz
Seed	I'm Yours	Jason Mraz
1	You And I Both	Jason Mraz
2	Walks Like Rihanna	The Wanted
3	If It Kills Me	Jason Mraz
4	Pour Que Tu M'Aimes Encore	Celine Dion
5	Love Somebody	Maroon 5
6	Stereo Love	Edward Maya
7	Life Is Wonderful	Jason Mraz
8	The Remedy (I Won't Worry)	Jason Mraz

Table 6.8: Seed Songs (Pop) from Jason Mraz

In Table 6.7, we started with rap songs from Eminem and ended up with all rap songs (except for Song 4) from Eminem himself and Gym Class Heroes (rap rock, alternative hip-hop). We also chose 'relaxing' music from Jason Mraz as seed songs in

Table 6.8, leaving us with more Jason Mraz and several pop songs of the same mood (except for Song 2 and 5). However, one cannot expect Au2seed to be perfect: it suggested songs from Taylor Swift, whose songs are Pop/Rock, in the middle of electronic dance music in Table 6.6.

We also contrasted two broad types of songs sung by Taylor Swift: the slower and the more upbeat ones.

In Table 6.9, the slow seed songs gave rise to a playlist that has a 'slow' feel too (except for Song 6). Table 6.10 is a playlist generated using two more 'upbeat' songs Taylor Swift, giving rise to similarly 'upbeat' or fast songs (except for Song 7).

	Song Title	Artist
Seed A	Safe & Sound	Taylor Swift
Seed B	Begin Again	Taylor Swift
1	Tonight, Not Again	Jason Mraz
2	Sans Patience	Joyce Jonathan
3	Lady	Regina Spektor
4	Buildings	Regina Spektor
5	Field Below	Regina Spektor
6	Not So Usual	Jason Mraz

Table 6.9: Slow Songs from Taylor Swift as Seed Songs

	Song Title	Artist
Seed A	Shake It Off	Taylor Swift
Seed B	I Knew You Were Trouble	Taylor Swift
1	Blurred Lines	Robin Thicke
2	Wild Ones	Flo-Rida
3	Do What You Want	Lady GaGa
4	Lollipop	Mika
5	Loca People (What The ****)	Sak Noel
6	Shake	Jesse McCartney
7	After An Afternoon	Jason Mraz

Table 6.10: Upbeat Songs from Taylor Swift as Seed Songs

On the whole, our quantitative results were encouraging (see Section 6.1) and in the qualitative evaluation, we generally received positive feedback. Playlists generated using Au2seed v3.6 were shown to be able to reflect mood or genre and changes in tempo quite well, despite the absence of genre and *tempo* as the metadata for v3.6.

Chapter 7

Conclusion

Music expresses that which cannot be said
and on which it is impossible to be silent.
–Victor Hugo

While we have reproduced Platt et al.'s work in [1] for playlist generation using Gaussian process and the method of KMT, we have also modified certain setups to suit our circumstance. This includes using numerical and textual data (genre) as the metadata fields, modifying the KMT to acquire a better prior and making adjustments to the evaluation procedure by using different libraries as the test set. Our results cannot be compared side by side, due to the modification of the setup and different datasets.

Using a standard collaborative filtering metric, good performance was observed across the Au2seed versions, performing 2–3.5 times better than a random song generator. To single-out which version is the best for us, we compared the metadata weights ω between the best-performing versions v3.0, v3.1 and v3.6. Our choice of Au2seed v3.6 does not necessarily mean v3.6 is the best, rather because v3.6 includes features that were observed to be more useful in determining the similarity between any 2 songs.

Comparison of the metadata weights ω across all versions also revealed that these Echo Nest features should be used (in decreasing order of importance): *segment_loudness*, *time signature*, genre, *timbre5*, *flatness*, *timbre7* and *brightness*. We also advise against the use of 2 MSD metadata, namely *mode* and *tempo*. Lastly, Au2seed v3.6, which uses the metadata *time signature*, *segment_loudness*, *brightness* etc., gave us promising results in the human evaluation procedure and users have been quite satisfied with the playlists.

We would like to highlight our main contributions for this project: (i) modified KMT for Gaussian process prior; (ii) a selection of song features that should be used and should be avoided while using the same setup; (iii) a simple and naïve way of interpreting the $\hat{\beta}$ from the optimisation in KMT by way of metadata weights ω; and (iv) the Au2seed playlist generator itself.

The use of the 10,000 MSDS is definitely a limitation as it represents 1% of the true universe. Thus our predictions might also carry that proportion of incorrectness. If we had had unlimited computing power, we would have undoubtedly used the entire MSD. This would probably paint us a better representation of the true universe of songs.

Certain aspects of this project suggest future directions of improvement and more concise evaluation: (i) the small scale of the data (9939 songs might not represent the 'universe' well); (ii) KMT can be further improved to create a better prior; (iii) the number of versions that were tested could be increased to cater for the different combinations of song metadata that can be used as inputs to the Mercer kernels; (iv) the selection of hand-picked song features that should be used and should be avoided, possibly not only limiting to this setup; and (v) the lack of literature in the interpretation of $\hat{\beta}$ obtained from the optimisation procedure in KMT.

On the whole, we hope that our project has benefited not only in the field of playlist generation but also the MIR community as a whole, in one way or another.

Bibliography

[1] J. C. Platt, C. J. Burges, S. Swenson, C. Weare, and A. Zheng, "Learning a gaussian process prior for automatically generating music playlists," in *NIPS*, 2001, pp. 1425–1432.

[2] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[3] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2014. [Online]. Available: http://www.R-project.org/

[4] B. McFee and G. R. Lanckriet, "The natural language of playlists." in *ISMIR*, 2011, pp. 537–542.

[5] T. P. Minka and R. W. Picard, "Learning how to learn is learning with point sets," *Web. Revised*, 1999.

[6] C. K. Williams and C. E. Rasmussen, "Gaussian processes for regression," 1996.

[7] B. McFee and G. R. Lanckriet, "Hypergraph models of playlist dialects," in *ISMIR*. Citeseer, 2012, pp. 343–348.

[8] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.

[9] K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. G. Okuno, "An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 2, pp. 435–447, 2008.

[10] M. Ghazanfar and A. Prugel-Bennett, "An improved switching hybrid recommender system using naive bayes classifier and collaborative filtering," 2010.

[11] G. M. Dakhel and M. Mahdavi, "A new collaborative filtering algorithm using k-means clustering and neighbors' voting," in *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on*. IEEE, 2011, pp. 179–184.

[12] T. Jehan and D. DesRoches, *Analyzer Documentation*, 2014. [Online]. Available: http://developer.echonest.com/docs/v4/_static/AnalyzeDocumentation.pdf

[13] A. Schindler, R. Mayer, and A. Rauber, "Facilitating comprehensive benchmarking experiments on the million song dataset." in *ISMIR*, 2012, pp. 469–474.

[14] Y. Hu and M. Ogihara, "Genre classification for million song dataset using confidence-based classifiers combination," in *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2012, pp. 1083–1084.

[15] A. Schindler and A. Rauber, "Capturing the temporal domain in echonest features for improved classification effectiveness," in *Adaptive Multimedia Retrieval: Semantics, Context, and Adaptation*. Springer, 2014, pp. 214–227.

[16] S. Dieleman, P. Brakel, and B. Schrauwen, "Audio-based music classification with a pretrained convolutional network." in *ISMIR*, 2011, pp. 669–674.

[17] C.-W. Chen, K. Lee, and H.-H. Wu, "Towards a class-based representation of perceptual tempo for music retrieval," in *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*. IEEE, 2009, pp. 602–607.

[18] A. Berenzweig, B. Logan, D. P. Ellis, and B. Whitman, "A large-scale evaluation of acoustic and subjective music-similarity measures," *Computer Music Journal*, vol. 28, no. 2, pp. 63–76, 2004.

[19] G. Bonnin and D. Jannach, "Automated generation of music playlists: Survey and experiments," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 26, 2014.

[20] D. Barber, *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

Appendix A

MSD Metadata & Glossary

Below is a list of all the metadata fields associated with the track “Never Gonna Give You Up” by Rick Astley.

Field Name	Type	Dimension	Description	Value(s)
analysis sample rate	float	1	sample rate of the audio used	22050
artist 7digitalid	int	1	ID from 7digital.com, or -1	8707738
artist familiarity	float	1	algorithmic estimation	0.7549176
artist hotttnesss	float	1	algorithmic estimation	0.5074642
artist id	string	1	Echo Nest ID	ARWPYQI1187FB4D55A
artist latitude	float	1	latitude	53.45644
artist location	string	1	location name	“Newton-le-Willows, Merseyside, England”
artist longitude	float	1	longitude	-2.63265
artist mbid	string	1	ID from musicbrainz.org	db92a151-1ac2-438b-bc43-b82e149ddd50
artist mbtags	array string	(4,1)	tags from musicbrainz.org	“uk”, “british”, “english”, “classic pop and rock”
artist mbtags count	array int	(4,1)	tag counts for musicbrainz tags	1, 1, 1, 1
artist name	string	1	artist name	Rick Astley
artist playmeid	int	1	ID from playme.com, or -1	1338
artist terms	array string	(12,1)	Echo Nest tags	dance pop, rock, pop, england, adult contemporary, etc.
artist terms freq	array float	(12,1)	Echo Nest tags weight	1.0000000, 0.7559042, 0.7233428, 0.7229454, 0.6841887, etc.
audio md5	string	1	audio hash code	bf53f8113508a466...
bars confidence	array float	(99,1)	confidence measure	
bars start	array float	(99,1)	start time of each bar	<this song has 99 bars >
beats confidence	array float	(397,1)	confidence measure	
beats start	array float	(397,1)	start time of each beat	<this song has 397 beats >

Field Name	Type	Dimension	Description	Value(s)
danceability	float	1	algorithmic estimation (between 0 and 1, 0 = not analysed)	0
duration	float	1	in seconds	211.69587
end of fade in	float	1	seconds at the beginning of the song	0.139
energy	float	1	energy from listener point of view	0
key	int	1	key the song is in (from 0 to 12)	1 (=C#)
key confidence	float	1	confidence measure	0.324
loudness	float	1	overall loudness in dB	-7.75
mode	int	1	major (=1) or minor (=0); the type of scale from which its melodic content is derived	1
mode confidence	float	1	confidence measure	0.434
release	string	1	album name	Big Tunes - Back 2 The 80s
release 7digitalid	int	1	ID from 7digital.com or -1	786795
sections confidence	array float	(10,1)	confidence measure associated with each section	
sections start	array float	(10,1)	start time of largest grouping in a song, e.g. verse	<this song has 10 sections >
segments confidence	array float	(935,1)	confidence measure for each segment	
segments loudness max	array float	(935,1)	max dB value for each segment	
segments loudness max time	array float	(935,1)	time of max dB value, i.e. end of attack	
segments loudness max start	array float	(935,1)	dB value at onset	
segments pitches	2D array float	(935,12)	chroma feature, one value per note	
segments start	array float	(935,1)	start time of each segment (musical events or note onsets)	<this song has 935 segments >
segments timbre	2D array float	(935,12)	12 MFCC-like features for each segment	
similar artists	array string	(100,1)	Echo Nest artist IDs (similarity algorithm unpublished)	
song hotttnesss	float	1	algorithmic estimation	0.864248830588
song id	string	1	Echo Nest song ID	SOCWJDB12A58A776AF
start of fade out	float	1	start time of the fade out, in seconds, at the end of the song	198.536
tatums confidence	array float	(794,1)	confidence measure	
tatums start	array float	(794,1)	start time of each tatum (smallest rhythmic element)	<this song has 794 tatums >
tempo	float	1	estimated tempo in BPM	113.359
time signature	int	1	estimate of number of beats per bar	4
time signature confidence	float	1	confidence measure	0.634
title	string	1	song title	“Never Gonna Give You Up”
track id	string	1	Echo Nest track ID	TRAXLZU12903D05F94
track 7digitalid	int	1	ID from 7digital.com or -1	8707738
year	int	1	song release year from Musicbrainz or 0	1987

Echonest Analyzer breaks down the audio into musically relevant elements that occur sequenced in time. From smallest to largest those include:

segments a set of sound entities (typically under a second) each relatively uniform in timbre and harmony. Segments are characterised by their perceptual onsets and duration in seconds, *segment_loudness* (dB), pitch and timbral content.

tatums list of tatum markers, in seconds. Tatums represent the lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events (segments).

beats list of beat markers, in seconds. A beat is the basic time unit of a piece of music; for example, each tick of a metronome. Beats are typically multiples of tatums.

bars list of bar markers, in seconds. A bar (or measure) is a segment of time defined as a given number of beats. Bar offsets also indicate downbeats, the first beat of the measure.

sections a set of section markers, in seconds. Sections are defined by large variations in rhythm or timbre, e.g. chorus, verse, bridge, guitar solo, etc. Each section contains its own descriptions of tempo, key, mode, time signature, and loudness.

For every segment, we have:

pitch content is given by a "chroma" vector, corresponding to the 12 pitch classes C, C#, D to B, with values ranging from 0 to 1 that describe the relative dominance of every pitch in the chromatic scale. For example, a C Major chord would likely be represented by large values of C, E and G (i.e. classes 0, 4, and 7) in the vector.

timbre the quality of a musical note or sound that distinguishes different types of musical instruments, or voices. It is a complex notion also referred to as sound colour, texture, or tone quality, and is derived from the shape of a segment's spectro-temporal surface, independently of pitch and loudness. The Echo Nest Analyzer's timbre feature is a vector that includes 12 unbounded values roughly centred around 0. Those values are high level abstractions of the spectral surface, ordered by degree of importance. For completeness however, the first dimension represents the average loudness of the segment; second emphasises brightness; third is more closely correlated to the flatness of a sound; fourth to sounds with a stronger attack; etc. See an image below representing the 12 basis functions (i.e. template segments). The actual timbre of the segment is best described as a linear combination of these 12 basis functions weighted by the coefficient values: timbre = $c1 \times b1 + c2 \times b2 + \ldots + c12 \times b12$, where $c1$ to $c12$ represent the 12 coefficients and $b1$ to $b12$ the 12 basis functions. Timbre vectors are best used in comparison with each other.

Other terminologies are defined below:

key a track-level attribute ranging from 0 to 11 and corresponding to one of the 12 keys: C, C#, D, etc. up to B. If no key was detected, the value is -1. The is equal to 0 or 1 for minor or major and may be -1 in case of no result. Note that the major key (e.g. C major) could more likely be confused with the minor key at 3 semitones lower (e.g. A minor) as both keys carry the same pitches. Harmonic details are given in segments below.

rhythm Beats are subdivisions of bars. Tatums are subdivisions of beats. That is, bars always align with a beat and ditto tatums. Note that a low confidence does not

necessarily mean the value is inaccurate. Exceptionally, a confidence of -1 indicates "no" value: the corresponding element must be discarded. A track may result with no bar, no beat, and/or no tatum if no periodicity was detected. The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4. A value of -1 may indicate no time signature, while a value of 1 indicates a rather complex or changing time signature.

time signature an estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).

Appendix B

User Scores, Ratings & Comments

Subject	No. of Songs	Name of Playlist	Raw Score	*P* Score	Overall User Satisfaction (1-10)	Comments
Serene	16	emo	5/8 8/8	6.25 10	7	”The other songs are also songs that I always listen to...”
Azmi	30	gym	11/12	9.2	9	
Jing Da	20	clubbing	6/9	6.7	7	”Songs 2, 6 and 7 are all quite slow. They are not clubbing songs.”
Fahmi	50	workout	9/22 18/22	4.1 8.2	7	Quite satisfied.
Ben	22	music for the soul	11/12	9.2	8.5	”... I don’t really use mood ... Larger sample size. Interesting though.”
Wei Qi	22	emo	6/7	8.6	8	”I would say not bad the fit. Why such a perfect fit?”
Farah	28	workout	9/12	7.5	9	”I hardly workout anymore LOLOL but yes all those other songs would work.”
Anna	25	workout music	5/11	4.5	7.5	”Cuz I dunno why I like the last one?” (songs towards the end were nicer)
Shaz	37	going to work	9/12	7.5	7	”Some of the songs generated look pretty spot on. I mean I don’t mind listening over and over again on my playlist everyday.”

Table B.1: User *P* Scores, Ratings and Comments

Appendix C

Learning Points of Project

Application Programming Interface

I have learnt how to make API calls to the Echo Nest to access the music database using the R programming software.

Large-Scale Datasets

Raw datasets are usually dirty. A lot of work needs to be done to pre-process the data before using it. Some of the other challenges include handling unfamiliar file formats of data (in this paper, we encountered HDF5 and JSON files) and missing and/or repeated values. We also learnt how to compress files using JSON (lists) and HDF5 (matrices), reducing a 9939×9939 matrix of size 2GB to 70MB.

Speed of Computation

Often the speed of computation is an issue. Below are some of the suggested ways to overcome this:

- Review algorithms
- Review R codes (e.g. there is a slight difference in using strings and numerical values for checking between 2 arrays; the latter is faster)
- Use computers with higher RAM
- Parallelise functions or programs over several RStudio windows and/or computers

Skills Learnt

- Problem-solving skills
- Word processing skills (LaTeX)
- Research skills