Audio-Based Cold-Start in Music Recommendation Systems

Rodrigo Carvalho Borges

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Doctor of Science

Program:	Computer Science
Advisor:	Prof. Dr. Marcelo Gomes de Queiroz

During the development if this work, the author received
financial support from CAPES Grant 88881.189985/2018-01

São Paulo
July, 2022

Audio-Based Cold-Start in Music Recommendation Systems

Rodrigo Carvalho Borges

This is the original version of the thesis prepared by candidate Rodrigo Carvalho Borges, as submitted to the Examining Committee.

The content of this work is published under the CC BY 4.0 license
(Creative Commons Attribution 4.0 International License)

Ficha catalográfica elaborada com dados inseridos pelo(a) autor(a)
Biblioteca Carlos Benjamin de Lyra
Instituto de Matemática e Estatística
Universidade de São Paulo

```
Borges, Rodrigo Carvalho
   Audio-Based Cold-Start in Music Recommendation Systems
/ Rodrigo Carvalho Borges; orientador, Marcelo Gomes
de Queiroz. - São Paulo, 2022.
   121 p.: il.


   Tese (Doutorado) - Programa de Pós-Graduação em Ciência
da Computação / Instituto de Matemática e Estatística
/ Universidade de São Paulo.
   Bibliografia
   Versão corrigida


   1. Music Recommendation Systems. 2. Audio Content. 3.
Audio-Based Music Recommendation. 4. Cold-Start. I. Gomes
de Queiroz, Marcelo. II. Título.
```

Bibliotecárias do Serviço de Informação e Biblioteca
Carlos Benjamin de Lyra do IME-USP, responsáveis pela
estrutura de catalogação da publicação de acordo com a AACR2:
Maria Lúcia Ribeiro CRB-8/2766; Stela do Nascimento Madruga CRB 8/7534.

Abstract

Rodrigo Carvalho Borges. **Audio-Based Cold-Start in Music Recommendation Systems**. Thesis (Doctorate). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

Music streaming platforms have become popular in the last decades due to the increasing number of tracks available online. The track catalogues offered by these platforms are usually too big to be searched manually, and automatic recommendation algorithms might be implemented for helping users navigate on these platforms. More specifically, Music Recommendation Systems (MRS) are designed for analyzing user listening behaviours and for predicting the songs that will be played in the near future by one specific user or within a listening session. But in the case new tracks are added to a platform, also known as the cold-start problem, no listening data is available, and the system needs to somehow incorporate these tracks into its recommendation algorithms. In this work, we propose methods that leverage the audio associated with tracks that were recently added to streaming platforms as an alternative for compensating the lack of interaction data.

Our propositions are elaborated considering collaborative filtering (CF), sequence-aware (SA), and stream-based (SB) recommendation systems, and audio files are considered represented as codeword histograms, Mel-spectrograms, and raw waveforms. In the first experiment, we propose a method that applies Convolutional Neural Networks (CNN) for mapping audio content to profiles containing the users who listened to a track. In a second experiment, Recurrent Neural Networks (RNN) are trained for reproducing the audio feature associated with the upcoming tracks within a listening session, given the audio feature associated with the current track. An inverted index structure is used for retrieving tracks given their estimated audio feature in an efficient way. In a third experiment, we propose a model that maps track/track transitions to an audio domain in a multi-level Markov Chain fashion. The method allows dynamic updates, allowing its application to scenarios of data streams.

The experiments were conducted using the LFM-1b music consumption dataset, and audio previews downloaded from Spotify. Our methods presented competitive prediction results in situations of cold-start in the case of CF and SA recommendation systems. The novel stream-based method is able to recommend tracks with an accuracy that is comparable to the accuracy measured for conventional rating-based methods, being based exclusively on audio content.

Keywords: Music Recommendation Systems. Audio Content. Audio-Based Music Recommendation. Cold-Start.

Resumo

Rodrigo Carvalho Borges. **Sistemas de Recomendação de Música Baseados em Áudio**. Tese (Doutorado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

Plataformas de streaming de música se tornaram populares nas últimas décadas devido ao crescente número de faixas disponíveis on-line. Os catálogos de faixas oferecidos por estas plataformas são, geralmente, muito grandes para serem pesquisados manualmente, e algoritmos de recomendação automática podem ser implementados para ajudar os usuários a navegar nestas plataformas. Mais especificamente, Sistemas de Recomendação Musical (MRS) são projetados para analisar os comportamentos de escuta dos usuários e para prever as músicas que serão tocadas em um futuro próximo por um usuário específico ou dentro de uma sessão de escuta. Mas quando novas faixas são adicionadas a uma plataforma, também conhecido como problema de cold-start, os dados de audição não estão disponíveis e o sistema precisa incorporar estas faixas em seus algoritmos de alguma forma. Neste trabalho, propomos métodos que utilizam o áudio associado às faixas que foram recentemente adicionadas às plataformas de streaming como uma alternativa para compensar a falta de dados de interação.

Nossas propostas são elaboradas considerando sistemas de recomendação baseados em Filtragem Colaborativa (CF), em sequências de dados de escuta (SA) e em stream de dados de escuta (SB). Os arquivos de áudio são considerados representados como histogramas de palavra-chave, mel-spectrogramas e formas de onda puras. Em um primeira experimento, propomos um método que aplica Convolutional Neural Networks (CNN) para mapear conteúdo de áudio a um perfil contendo os usuários que ouviram a uma faixa. Em um segundo experimento, Redes Neurais Recorrentes (RNN) são treinadas para reproduzir os conteúdos de áudio associados às próximas faixas dentro de uma sessão de escuta, dado o conteúdo de áudio associado à faixa atual. Uma estrutura de índice invertido é usada para a recuperação de faixas, dado seu conteúdo de áudio de forma eficiente. Em um terceiro experimento, propomos um modelo que mapeia as transições de faixa/faixa para um domínio de áudio utilizando uma cadeia de Markov de vários níveis. O método permite atualizações dinâmicas, permitindo sua aplicação a cenários de intenso fluxo de dados.

Os experimentos foram conduzidos utilizando o conjunto de dados de consumo de música LFM-1b, e previews de áudio baixados de Spotify. Nossos métodos apresentaram resultados de previsão competitivos em situações de cold-start no caso de sistemas de recomendação CF e SA. O novo método baseado em fluxo é capaz de recomendar faixas com uma precisão comparável à precisão medida para métodos convencionais baseados em dados de escuta, sendo baseado exclusivamente no conteúdo de áudio.

Palavras-chave: Sistemas de Recomendação de Música. Conteúdo de Áudio. Sistemas de Recomendação de Música Baseados em Áudio.

List of Figures

2.1 Matrix Factorization. 15
2.2 Audio Representations of a 30-seconds excerpt of "Harder, Better, Faster, Stronger" by Daft Punk. 17
2.3 Triangular Mel filter bank with 50% overlap. 19
2.4 A nonlinear function of a weighted sum of inputs on an artificial neuron. 22
2.5 Variational Autoencoder. 25
2.6 Illustration for 1D and 2D convolutional operations. 29
2.7 GRU4REC applies Recurrent Neural Networks for predicting the next item in the sessions given the current one (figure reproduced from HIDASI *et al.*, 2016). 37
2.8 Neural Attentive Recommendation Machine (NARM) scheme (figure reproduced from J. LI *et al.*, 2017). 38
2.9 Previous audio-base methods proposed for mitigating the cold-start in music recommendation. 46

3.1 Audio-Based Convolutional Variational Autoencoder Recommender (AC-VAE). 53
3.2 Audio-Based Convolutional Regularized Embedding Recommender (ACRE) 55
3.3 Audio-Based GRU4REC (AGRU4REC). 58
3.4 Multi-Level Audio Inverted Index (MLAII) structure for N=6. 59
3.5 Sequential Audio-Based Top-N Autoencoder for Recommendation (SA-TAREC). 62
3.6 Audio-based Transition Tensor Recommendation model (ATTREC). . . . 65

4.1 LFM-1b years histogram. 70
4.2 CNN architectures used in the Collaborative Filtering experiments. . . . 75
4.3 The number of target users separated for evaluating the CF methods. . . 78

A.1 Results measured for the track profile prediction task in the context of collaborative filtering. 100
A.2 Six hand-made clusters obtained from audio-based representations of songs from the LFM-1b dataset. 101
A.3 Results measured for the next track prediction task. 102
A.4 Results measured for the stream-based methods, with K=1 and for 50,000 listening sessions. 103
A.5 Results measured for the stream-based recommendation methods, with K=10 and for 50,000 listening sessions. 104
A.6 Results measured for the stream-based recommendation methods, with K=100, and for 50,000 listening sessions. 105

List of Tables

2.1 Explicit feedback user rating matrix example. Ratings values can vary from 1 to 5. 32

4.1 Datasets description 70
4.2 Results obtained for VAE-CF and WMF methods in the Collaborative Filtering recommendation task. 79
4.3 Recommendation results for the task of track profile prediction in the context of collaborative filtering. 80
4.4 Retrieval results obtained for the Multi-Level Audio Inverted Index (MLAII). 86
4.5 Results measured for the next-track prediction task. 88
4.6 ATTREC recommendation results measured for the first 50,000 listening sessions with different N and α parameters. 94
4.7 Recommendation results measured for the first 50,000 listening sessions according to a simulation of a dynamic recommendation environment. . 95
4.8 Recommendation results measured for the audio-based methods in the task of next-track prediction. The results are separated into overall, warm-start and cold-start scenarios. 95

A.1 Recommendation results obtained for the WMF recommendation method. The results are presented in the following format: confidence type/α/λ. . 99

Contents

1 Introduction **1**
1.1 Context and Motivation . . . 2
1.2 Related Work . . . 4
1.2.1 Collaborative Filtering . . . 4
1.2.2 Sequence-Aware . . . 6
1.2.3 Stream-Based . . . 8
1.3 Research Questions . . . 9
1.4 Contributions . . . 10
1.5 Organization . . . 10

2 Background **13**
2.1 Basic Concepts . . . 14
2.1.1 Vector Quantization . . . 14
2.1.2 Matrix Factorization . . . 15
2.1.3 Markov Models . . . 16
2.2 Audio Representation . . . 16
2.2.1 Time Domain . . . 16
2.2.2 Frequency Domain . . . 18
2.2.3 Task Specific Representations . . . 21
2.3 Deep Learning Methods . . . 21
2.3.1 Neural Networks . . . 21
2.3.2 Autoencoders . . . 24
2.3.3 Convolutional Networks . . . 27
2.3.4 Recurrent Networks . . . 29
2.4 Rating-Based Recommendation . . . 31
2.4.1 Collaborative Filtering . . . 32
2.4.2 Sequence-Aware . . . 35
2.4.3 Stream-Based . . . 39

2.5 Audio-Based Music Recommendation . . . 43
2.5.1 Collaborative Filtering . . . 44
2.5.2 Sequence-Aware . . . 47
2.5.3 Stream-Based . . . 48

3 Methodology 51
3.1 Collaborative Filtering . . . 51
3.1.1 Problem Definition . . . 52
3.1.2 Audio-Based Convolutional Variational Autoencoder Recommender 52
3.1.3 Audio-Based Convolutional Regularized Embedding Recommender 55
3.1.4 Evaluation Metrics . . . 55
3.2 Sequence-Aware . . . 56
3.2.1 Problem Definition . . . 57
3.2.2 Audio-Based GRU4REC . . . 57
3.2.3 Multi-Level Audio Feature Inverted-Index . . . 59
3.2.4 Sequential Audio-Based Top-N Autoencoder Recommender . . . 61
3.2.5 Metrics . . . 62
3.3 Stream-Based . . . 63
3.3.1 Problem Definition . . . 64
3.3.2 Audio Transition Tensor Recommender . . . 64
3.3.3 Metrics . . . 66

4 Experiments and Results 69
4.1 Datasets . . . 69
4.2 Collaborative Filtering . . . 71
4.2.1 Data Preparation . . . 72
4.2.2 Methods . . . 73
4.2.3 Experiments . . . 77
4.2.4 Results . . . 78
4.3 Sequence-Aware . . . 81
4.3.1 Data Preparation . . . 82
4.3.2 Methods . . . 82
4.3.3 Experiments . . . 84
4.3.4 Results . . . 86
4.4 Stream-Based . . . 89
4.4.1 Data Preparation . . . 89
4.4.2 Methods . . . 89
4.4.3 Experiments . . . 91

4.4.4 Results 94

5 Conclusions **97**

Appendixes

A Appendix **99**

A.1 Rating-Based Collaborative Filtering 99

A.2 Cold-Start Collaborative Filtering Results 100

A.3 Audio-Based Artist Clusters 101

A.4 Sequence-Aware Recommendation Results 102

A.5 Stream-Based Recommendation Results 103

A.6 List of Publication 106

References **107**

Chapter 1

Introduction

Digital music became largely available on the internet in the last decades, drastically changing the way people consume music. Today, instead of having individual song collections, many people prefer to pay a monthly fee and get access to platforms hosting huge collections of tracks, available for streaming upon demand. Some popular music streaming platforms nowadays are: Tidal[1], Deezer[2] and Spotify[3].

Most streaming platforms provide their users with search engines, allowing them to search for specific tracks or artists, and some platforms also offer a personalized service capable of suggesting tracks based on a user's listening habits. These personalized services are known as Music Recommendation Systems (MRS) (Schedl *et al.*, 2015). Nowadays, most of these systems are data-driven, meaning that they are trained with data associated with a user's interactions with the platform. Once trained, these systems are capable of suggesting tracks that are relevant to each individual user.

In this work, we consider three categories of methods for MRS: (i) *collaborative filtering* (CF), (ii) *sequence-aware* (SA), and (iii) *stream-based* (SB) methods. CF has been considered the most popular recommendation technique (Koren *et al.*, 2009a); it associates each user with a listening profile, representing the user's music preferences. Similar profiles are then used to produce track suggestions (Goldberg *et al.*, 1992). SA methods consider the sequence of listened tracks and use sequence similarity to produce recommendations (Quadrana *et al.*, 2018). Timestamped sequences can be associated with the entire listening history of a user, or they can be partitioned into uninterrupted listening *sessions*, which are associated with a reduced temporal listening context. Lastly, SB methods are capable of instantly updating their models after each user interaction with the platform (Zhao *et al.*, 2013). This defining capability of SB methods can be associated with both collaborative filtering and sequence-aware methods and emphasizes their real-time adaptability to listening context changes.

Historical listening data, i.e. user/track records, is the most important resource for MRS. In streaming platforms, it is reasonable to assume that new tracks and users, lacking

[1] https://tidal.com/

[2] https://www.deezer.com/

[3] https://www.spotify.com

historical listening data, are constantly being added to the system. This context is usually referred to as *cold-start* (Schein *et al.*, 2002), meaning that these new users or tracks are not associated with any listening preferences, and have to be somehow incorporated into the algorithm. In this scenario, the recommender needs access to other resources of information besides listening data. User and/or track information can be obtained, for example, from social networks or online textual reviews, and in the specific case of music recommenders, the audio content is also available as side information within the platform (Cano *et al.*, 2005b).

This work is focused on the context of cold-start, and we propose methods for mitigating the lack of user listening information using the audio contents. We explore different audio representations, such as Mel-Spectrogram and raw waveform[4], and compare them within collaborative filtering, sequence-aware, and stream-based recommendation tasks. The proposed methods are mostly derived from, but not restricted to, artificial neural networks, including Variational Autoencoders (VAE), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN)[5].

1.1 Context and Motivation

In order to understand the challenges of automatically recommending music, we may imagine the following hypothetical scenario. Consider a person using a music streaming platform and listening to music in three different contexts: at home with kids, walking back home from work, and at the bar with friends; consider also that this person has different music preferences for each of these contexts. In order to tie this hypothetical scenario to our focus on cold-start, imagine that our hypothetical user's favorite artist has just released a new track. Now imagine that this user relies on an automatic recommender system for choosing music tracks, being able to skip suggestions, and also manually selecting desired tracks.

From the perspective of the recommender system, we consider that it has full access to all listening data for all of its users, including users' full listening records (associated with long-term preferences) and ongoing listening sessions (associated with short-term preferences). We also consider that the recommender system has full access to all audio data in the streaming platform[6]. The recommender system is comprised of one or more previously trained models, whose role is to recommend a new track each time the user skips the current track or finishes listening to it. This model should be retrained periodically or at each user interaction with the system.

The main challenges considered here are: (a) suggesting relevant tracks, ideally tracks which the user didn't listen; (b) adapting to context to suggest suitable tracks; (c) dynamically incorporating incoming data from user/track interactions; and (d) incorporating recently released tracks and recommend these to matching users (this is the cold-start

[4] Audio representations are discussed in detail in Section 2.2

[5] Section 2.3 is dedicated to artificial neural networks, where VAE, CNN, and RNN will be discussed in detail.

[6] this may be obvious if the recommender system is part of the streaming platform, but it might also be an independent service.

problem). We discuss in the sequel each of these challenges separately, considering the previously presented recommendation approaches: (i) Collaborative Filtering, (ii) Sequence-Aware, and (iii) Stream-Based methods.

Challenge (a) is closely, but not exclusively, related to recommender methods belonging to category (i). Imagine that the hypothetical user from our previous example listens mostly to Miles Davis and Daft Punk. The algorithm is capable of comparing this user's profile with all other user profiles, and ranking the latter according to similarity. The first-ranked profiles might include tracks by Miles Davis, Daft Punk, and Madonna, and the system might infer that Madonna is a suitable recommendation to our target user. The collaborative approach, described here in a very simplified way, is efficient for finding similarities between user profiles. It is not clear, however, how to consider listening context within a purely CF approach.

Challenge (b) has a closer connection to recommendation strategies belonging to category (ii). Sequence-Aware methods take into account not only long-term listening preferences but also short-term preferences, e.g. sequences of tracks within listening sessions. Sequential methods might thus be more flexible in adapting to new contexts than Collaborative Filtering methods. In our current example, imagine the user leaving home to work, and switching to a faster-paced track. From a global point of view, their listening preferences are still the same, but a sequence-aware recommender might look for similar listening sessions starting with the selected track, and thus suggest a follow-up track that reflects the new listening context.

Stream-based recommendation models (category iii) are those that support dynamic updates, i.e. the incorporation of continuously generated user/track listening data in real-time, allowing every user action to influence the following recommendations. They address the challenge (c) and allow the representation of listening profiles using several simultaneous temporal contexts. Some dynamic models are able to introduce a certain amount of uncertainty in their regular operation, allowing them to *explore* new recommendation possibilities, instead of just *exploiting* its current knowledge. Stream-based systems can operate according to collaborative or sequence-aware approaches.

The last challenge (d), or the cold-start problem, associated with the necessity of incorporating recently released tracks in recommendation systems, can be addressed in collaborative, sequential, and dynamic recommendation approaches. As previously mentioned, user/track interaction data is vital for the operation of recommender systems, and when a new track is inserted into the platform, there has to be a strategy for incorporating this track into new recommendations. One simple idea would be to retrieve music information from the audio signal of the new track; for example, the audio content analysis might identify it as a fast-paced track, or a song for voice and piano. Such descriptions could be aligned with user preferences to support recommendations including this recently released track. The correspondence between user preferences and audio features, however, may not be as straightforward as in the previous example, and more sophisticated approaches are required.

In this work, we propose audio-based strategies for incorporating new tracks into recommendation algorithms, i.e., strategies that rely on music information retrieval for tracks without any associated listening data. Some strategies are meant to support the inclusion of these tracks within collaborative or sequential recommender methods, and some strategies are proposed for operating as end-to-end recommendation systems.

1.2 Related Work

We now review the most relevant attempts at mitigating the cold-start problem from the perspective of collaborative, sequence-aware, and stream-based recommendation systems. We are here mainly interested in the works that used audio as an auxiliary resource for overcoming the lack of interaction data. It is worth mentioning that the majority of works that have addressed this issue were focused on CF.

1.2.1 Collaborative Filtering

Collaborative Filtering (CF) was first proposed as an alternative for filtering content in a shared communication environment (Goldberg *et al.*, 1992), assuming that similar preference profiles (as modelled by the CF mechanism) would imply similarity of subjective (real) preferences. CF recommendation strategies can be separated into at least two categories (Ricci *et al.*, 2011): *neighborhood-based* (Ning *et al.*, 2015), and *model-based* (Koren *et al.*, 2009a).

Neighbourhood-based strategies propose metrics for measuring similarity between user profiles, where profiles that are similar to the target user profile are taken into account in the process of calculating recommendations for this specific user.

Model-based strategies rely on user/item interaction data for training recommendation models, within which *latent variables* (or latent factors) are calculated for each user as a dense and compact representation of the original preference profile. In model-based approaches, the interaction data is usually stored in a big and sparse matrix, the rating matrix, containing numerical values associated with each $(user, item)$ pair: rows of this matrix (ratings by a single user) may be viewed as user profiles and columns (rating of a single item) as item profiles. Instead of explicitly considering the similarity of user profiles as the rationale for the recommendation, these approaches aim at predicting rating values from existing ratings (e.g. using matrix factorization techniques).

In CF approaches, user/item interaction data do not involve timestamps, and any information regarding the temporal sequence of items consumed by a single user is ignored. Some of the most popular approaches for CF are Weighted Matrix Factorization (WMF) (Hu *et al.*, 2008) and more recently, Variational Autoencoders (VAE) (Liang *et al.*, 2018).

When one new item is added to a system operating according to a model-based CF algorithm (i.e. the cold-start problem), we can imagine an empty column (item profile) in the user/item rating matrix, meaning that no interaction information is associated with this specific item. One possible approach for mitigating this lack of information is to approximate the values of this empty column based on metadata associated with the new item (e.g. audio features in music recommendation). In this case, an inference model may be trained based on previous user/item interactions in order to predict the new item profile solely based on the corresponding metadata. Several strategies were proposed for addressing the cold-start problem in the context of recommender systems, which are discussed in the sequel.

One possible option, maybe the most intuitive one, is to first factorize the rating matrix into user and item latent matrices, and then to factorize the item latent matrix into two new matrices, one of them containing given metadata and another latent matrix that is learned (Forbes and Zhu, 2011). A rating is then estimated/predicted from the user profile, the learned latent matrix, and the item metadata, allowing the prediction of ratings of new items from their metadata alone. This method is known as *Content-Boost*.

Regarding the content associated with items, one option is to use textual data associated with items as an alternative resource of information (Gouvert *et al.*, 2018; Fressato *et al.*, 2018; Elahi *et al.*, 2019; Barkan *et al.*, 2019; Bogdanov, Haro, *et al.*, 2013; Volkovs *et al.*, 2017). Textual data can be embedded in the item as metadata (e.g., the release date of a movie or the singer of a song), it may be obtained by expert annotators (who associate semantic tags to items), or it may be gathered from the internet. Expert annotations may be the most meaningful among these, but their production can be time-consuming (Q. Li *et al.*, 2004). It is also important to notice that annotators of real-life platforms often lack expertise and may produce unreliable metadata, especially regarding non-mainstream items or items outside of their own cultural background. This suggests caution when dealing with textual metadata.

Another possible approach for the cold-start problem is to consider combinations of textual and audio data for new tracks (Knees *et al.*, 2006). A multimodal network that combines artist metadata and CNN audio embeddings was proposed by Oramas *et al.*, 2017 for this specific purpose. In this work, matrix factorization is applied to a dataset of user/track interactions in order to produce separate user and track dense representations, and also artist dense representations from track representations. Next, neural networks are trained for learning these track and artist representations having the corresponding content as the input: audio in the case of tracks, and biographical textual data in the case of artists. The previous-to-last layer in both cases is considered as an embedding, and these two combined embeddings are applied for learning the track dense representations obtained in the first stage. The resulting model is able to infer ratings for new tracks, given textual and audio data as input.

In Yoshii *et al.*, 2006 the authors proposed a probabilistic model for suggesting tracks to users based on the tracks' audio content, described as follows. First, Gaussian Mixture Models (GMM) are built from the Mel Frequency Cepstrum Coefficients (MFCC) associated with each track. Then, user preferences towards GMM representations are calculated based on user/track interaction data. And finally, a three-way aspect model is then capable of suggesting tracks to users based on audio contents associated with tracks. A similar approach was proposed in Borges and Queiroz, 2018, but using audio codeword histograms and implicit (binary) feedback.

A new method is proposed for mapping the audio content associated with a track to its corresponding listening profile, i.e. a profile containing the information of which users interacted with that track (Oord *et al.*, 2013). This was the first approach based on deep learning[7], and it allowed the estimation of an empty column that is added to the rating matrix in the case of a new track, as mentioned previously. The proposed methodology consists of two steps: first, the rating matrix is factorized for obtaining compact representations (embeddings) of track and user profiles[8]. Second, a CNN is trained for approximating the track embeddings, given the audio features associated with the corresponding tracks. A new track can have its embedding estimated by the CNN, and this embedding can be, then, multiplied by the user embeddings for obtaining an approximation of its corresponding profile. The authors compare different audio representations (bag-of-words and Mel-spectrograms) used as input data, and different predictors (linear regression, multi-layer perceptron, metric learning-to-rank) applied as estimation methods. A similar approach proposes using the raw waveform associated with tracks as input data (Platt, 2017) and applying a sample-level CNN architecture. This sample-level CNN architecture was originally proposed in the context of auto-tagging (Kim *et al.*, 2018).

Instead of estimating track profiles in two stages, first a matrix factorization, and then an embedding estimation, a new method is proposed by X. Wang and Ye Wang, 2014 for performing both stages simultaneously. The *Hierarchical Linear Model with Deep Belief Networks* (HLDBN) was designed for learning audio embeddings and minimizing the profile estimation error at the same time, within one single optimization process. The method is based on Deep Belief Networks (DBN[9]) and once trained, it can estimate track profiles directly from their corresponding Mel-spectrograms.

In this work, we propose new methods for estimating track listening profiles given their corresponding audio content, with the aim of alleviating the limitations imposed by the cold-start. Our methods are closely related to or could be understood as a continuation of the ones presented in Oord *et al.*, 2013; X. Wang and Ye Wang, 2014; Platt, 2017, and the CNN architecture used in one of the methods was originally proposed in Kim *et al.*, 2018. We test two audio representations (Mel-Spectrograms (Oord *et al.*, 2013; X. Wang and Ye Wang, 2014) and Raw Waveforms Platt, 2017), two matrix factorization methods (WMF (Oord *et al.*, 2013; X. Wang and Ye Wang, 2014; Platt, 2017) and VAE), and two strategies for training the inference model (an end-to-end (X. Wang and Ye Wang, 2014) and a two-step (Oord *et al.*, 2013; Platt, 2017)). To the best of our knowledge, this is the first attempt on using VAE as an alternative for WMF in this specific task.

1.2.2 Sequence-Aware

Another category of recommendation methods, referred to here as sequence-aware (SA) methods, formulate the recommendation task as a sequence prediction task. The temporal context in which tracks were listened is now preserved, and recommendation methods are designed for suggesting the tracks that will be listened in the near future, given the information about the tracks that were listened in the past. Listening events can be considered as ordered in time (*sequence-aware*) (Quadrana *et al.*, 2018), they can

[7] See Section 2.3

[8] More information about matrix factorization can be found in Section 2.1.2

[9] To the best of our knowledge, Deep Belief Networks have the same architecture as fully connected neural networks, or Multi-Layer Perceptrons (MLP). For more information the reader might want to check Section 2.3.1.

be considered as segmented in non-interrupted listening sessions (*session-based*), or can be considered individually, together with the timestamp when the user/track interaction happened (*time-aware*) (Campos *et al.*, 2014). We refer to time-aware and to session-based methods as specific setups that can be considered sequence-aware methods since this is the most general setup, that can be made more specific if necessary.

Markov Chain (MC) might be the most intuitive, yet powerful, model applicable to sequence prediction, that was applied to the task of sequential recommendation (Ludewig and Jannach, 2018; Brian McFee and Lanckriet, 2011; Hosseinzadeh Aghdam *et al.*, 2015). The main assumption in MC is that item/item transitions that happened more frequently in the past are more likely to happen in the future. A tensor factorization method, named Factorizing Personalized Markov Chains (FPMC) (Rendle, Freudenthaler, *et al.*, 2010), combines matrix factorization and personalized transition matrices for predicting the next set of items that will be consumed by users. A deep learning method, based on Gated Recurrent Units (GRU) (Chung *et al.*, 2014), was proposed for predicting the following item within a non-interrupted session, given the previously consumed item (Hidasi *et al.*, 2016); and Neural Attentive Recommendation Machine (NARM) (J. Li *et al.*, 2017) was also proposed for predicting the upcoming item within a session, but this time all previous items from the session are considered in the prediction task. Other methods based on recurrent neural networks were applied to the next-item prediction task, with several improvements (Devooght and Bersini, 2016; Xu *et al.*, 2019; Wu *et al.*, 2019; Q. Liu *et al.*, 2018). RNN-based methods were compared to neighborhod-based methods in Jannach and Ludewig, 2017; Ludewig, Mauro, *et al.*, 2021; Latifi *et al.*, 2021.

In the specific case of music recommendation, the idea of segmenting user/item interactions in non-interrupted sessions is very relevant, especially if compared with other recommendation domains: tracks are short and are usually consumed in listening sessions or in the format of *playlists* (Bonnin and Jannach, 2014). The authors in Chen *et al.*, 2012 propose an algorithm for generating playlists automatically, named Latent Markov Embedding (LME). The LME algorithm calculates embeddings for every track based on playlist co-occurrence, and playlists are generated from a Euclidian space built from those embeddings. A recent work highlights the importance of the context in which users listen to music (Hansen *et al.*, 2020). The authors propose a neural network architecture named CoSeRNN that models users' preferences as a sequence of embeddings.

To the best of our knowledge, one single method was already proposed for incorporating new tracks into a sequential music recommender given their corresponding audio contents (Chou *et al.*, 2016). The method, named Adaptive Linear Mapping Model (ALMM), adapts the content-boost methodology (Forbes and Zhu, 2011) to the next-track recommendation task. ALMM decomposes transitions between tracks observed for each user as a product of three latent matrices, in a fashion similar to FPMC. The three matrices are associated with users, previous tracks, and next tracks. The two last matrices, the ones associated with previous and next tracks, are factorized again as linear products of an audio features matrix and auxiliary matrices that will be learned during the optimization process. When a new track is added to the track set, it can be incorporated into the algorithm by calculating the inner products of its audio feature and the learned auxiliary matrices.

In this work, we propose a method that maps track/track transitions to an audio domain, and that calculates next-track predictions within this new space. When a new track is incorporated it could be included in the new space and be recommended solely based on its audio representation. It differs from ALMM in the sense that track transitions are considered anonymously, meaning that new users can be instantly incorporated into the algorithm.

1.2.3 Stream-Based

In the case of a real-life streaming platform, it might be reasonable to assume that users are frequently interacting with items and that the data originating from these interactions is arriving at the platform as data streams. An automatic recommender operating within this platform is supposed to suggest relevant tracks to users upon demand, and in order to do that, the recommender needs to incorporate the arriving data in real (or almost real) time. The systems capable of adapting themselves dynamically according to incoming data streams were baptized *stream-based* (SB) recommendation systems (Al-Ghossein, Abdessalem, and BARRÉ, 2021).

We differentiate two specific characteristics observed in SB systems: the first one is associated with their capability of adapting themselves when users change their preferences during a listening session (known as *preference shift* (Hariri *et al.*, 2015)), for instance, when a user changes from one music genre to another. The second characteristic is associated with the situation when new items and/or new users are registered in the platform and need to be dynamically incorporated into the recommendation algorithm, i.e. cold-start. In order to incorporate these new elements, systems usually apply strategies associated with the dilemma of *exploration-exploitation*, that is, they need to combine the knowledge they have already acquired (exploitation), with the risk of trying out new possibilities (exploration).

Another observation to be made is that SB systems can operate according to CF or SA approaches. The majority of work on stream-based recommendation is focused on collaborative recommendation when rating matrices are built or factorized dynamically. Another branch of studies is focused on how to update SA methods dynamically. In this work we focus on the situation when new tracks are added to a music recommendation platform that operates suggesting next-tracks within listening sessions, that is, we tackle dynamic cold-start in SA music recommendation systems.

The specific task of recommending music in a dynamic fashion was tackled as a multi-dimensional navigation task, in which each reaction from users is incorporated dynamically, thus having an impact on the next movement in the trajectory (Cardoso *et al.*, 2016). In Pereira *et al.*, 2019, the authors propose an online learning-to-rank scheme that is capable of updating the recommendation model according to implicit feedback provided by users. The recommender is assumed to be a linear model, and its parameters are updated at each round according to the position in which the next track is located in the generated ranking.

Incorporating uncertainty or even a certain amount of stochasticity is one alternative for expanding the system's knowledge of user preferences, as seen in Zhao *et al.*, 2013. Another alternative is to rely on an external source of information about the new users or items that are being added to the platform. A strategy for incorporating textual information about items was proposed in X. Liu and Aberer, 2014, together with a combination of an online and an offline factorization models designed for delivering top-N recommendations dynamically.

A novel approach is proposed for a dynamic content-based music recommender in Xing *et al.*, 2014; X. Wang, Yi Wang, *et al.*, 2014. Ratings given by users are modelled as normal distributions whose parameters are estimated from the combination of two factors, an affinity for the audio content associated with the track, and a factor responsible for diversity. The affinity towards audio features is modelled as an inner product of a user preference variable and the audio features associated with the listened tracks. The diversity factor is implemented as an exponential curve that prevents the recommender to repeat a song that was recently suggested. The authors present an efficient method for estimating the model's parameters given the historical data from each user, which converts the overall updating procedure into a simple task of updating the historical data obtained from each user after a recommendation round. The system, however, iterates through every track for selecting the one that maximizes a quantile value of the estimated distribution, inspired by Bayesian-UCB (Kaufmann *et al.*, 2012), and this can be time-consuming.

Another audio-based strategy proposed for suggesting the next track in an ongoing listening session is proposed in Liebman *et al.*, 2015. DJ-MC is an algorithm that assumes user profiles composed of preferences for audio features obtained from tracks listened to previously, as well as preferences for transitions between those audio features. That is to say, it assumes that users have preferences not just for tracks with certain audio characteristics, but also for certain transitions between those audio characteristics. The method supports online updates, performed after each recommendation round, and it decides about the next track to be recommended according to a strategy that selects the most suitable playlist among a set of pseudo-random generated playlists. DJ-MC is considered the main method to be used as a baseline method due to the fact that it is audio-based, and due to the fact that it suggests the next track in an ongoing listening session.

We propose a new stream-based music recommendation method that summarizes one-dimensional audio features according to their top-N most relevant values, and that stores transitions between those simplified features in a transition tensor. Recommendations are calculated based on the most likely transitions between these audio feature elements, in a Markovian fashion, but considering a transition matrix for each of the N audio elements.

1.3 Research Questions

In this study, we discuss the situation in which new tracks are added to music recommender systems, known as cold-start, and we proposed audio-based methods for incorporating those tracks into recommendation algorithms. Some methods were designed with the aim of incorporating new tracks into recommender systems that are already operating, and some methods were designed for delivering accurate suggestions to users in a content-based fashion (Flexer *et al.*, 2010; Bogdanov, Haro, *et al.*, 2013).

Our research questions are:

- How to integrate new tracks to recommender systems implemented according to CF, SA and SB recommendation strategies, having only access to the audio associated with these tracks?
- Which audio representations (audio feature) are the most suitable in each of these situations?
- Is it possible to build a music audio-based recommender system? Or, do the strategies designed for mitigating cold-start require a pre-existing rating-based recommendation model?

1.4 Contributions

Our contributions are:

- We propose a new method for predicting the top-N users that will interact with a certain track based on its audio content. Our method is based on CNN and VAE networks, and it outperforms methods proposed previously in the literature, both in terms of prediction accuracy and in terms of the time spent in the training process.
- We propose two new methods for predicting the next track within an ongoing listening session given an audio feature associated with the current track. The first method is based on GRU networks, and it is applicable to situations where the audio content associated with the current track is the only information available. This method, however, can not extrapolate its predictions to new tracks, not being adequate for mitigating the cold-start problem. The second method is also based on GRU networks, it is proposed for addressing the same task, but this time it is designed for extrapolating its predictions to new tracks, being adequate for cold-start situations.
- Lastly, we propose a novel method designed for predicting the next track within an ongoing listening session, that is also fed with the audio feature associated with the current track, that is able to incorporate new tracks into the recommendation algorithm, and can be updated dynamically. The method is composed of two modules, an audio transition tensor, within which the likelihood of transitions in the audio domain are registered, and a retrieval module consulted for retrieving candidate tracks given an estimated audio feature. The method can have its parameters updated in real-time, and its results are comparable to results calculated for rating-based methods in a stream-based recommendation task.

1.5 Organization

The text is structured as follows. We start reviewing the most relevant concepts and summarizing the methods used along the text in Chapter 2. First, some basic concepts applied in the text are briefly explained, several audio representations are also introduced, the three main neural network architectures applied by the proposed methods are presented, and recent work on music recommendation is reviewed and discussed under the perspective of cold-start. New methods designed for incorporating new tracks into collaborative, sequential, and dynamic music recommendation methods are introduced in Chapter 3. First, a method is proposed for estimating which users interacted with a certain track given its audio content, a second method is proposed for modelling transition patterns in an audio domain, used for estimating the next track within a listening session, and another method is proposed, also based on audio content, for incorporating new tracks to recommendation algorithms, but this time on the context of stream-based recommendation systems. Experiments conducted with a dataset containing real user/track interaction data are described in Chapter 4. The process used for obtaining and organizing audio files, and matching them to the original dataset containing user/track interactions data is described in detail. Three different experiments were designed for the tasks of collaborative, sequential, and dynamic recommendations, and their results are discussed in detail. Finally, conclusions and future work are presented in Chapter 5.

Chapter 2

Background

In this chapter, we introduce the main concepts and definitions to be considered as the basis for the rest of the work. First, techniques like matrix factorization and vector quantization are briefly explained for helping the reader to go through some methods that will apply those techniques later on in the text.

We proceed by reviewing several music audio representations and commenting on their potential applications. In this work, we don't rely on any reference to symbolic representations of music like music sheets, MIDI protocol or any machine-readable data format that represents music entities. Instead, we consider songs as equivalent to their audio contents.

Next, we introduce Multi-Layer Perceptrons (MLP), Variational Autoencoders (VAE), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), considered the most popular architectures of Artificial Neural Networks (ANN). ANN gained much attention in the last decades, mainly due to its ability to approximate complex functions in a data-driven fashion. These networks are composed of layers of nodes (perceptrons) that are connected by edges in a graph-like topology, and the networks containing more than one internal layer of nodes are usually referred to as Deep Learning (DL) architectures.

We present a bibliographical review on automatic recommendation methods based on user/item interaction data (*rating-based*), and we classify these methods into three categories: collaborative filtering, sequence-aware and stream-based methods. We present a review of music recommendation methods that leverage the audio content associated with tracks in their modelling processes (*audio-based* methods). We provide a detailed description of the problem of incorporating new tracks to music recommendation algorithms, also referred as *New Item* or *Cold-Start* problem. The methods applying both user/item interaction and audio are considered audio-based for the sake of simplicity.

2.1 Basic Concepts

2.1.1 Vector Quantization

Quantization is the process of representing continuous values (or values obtained from large sets) in terms of a finite number of elements (or in terms of a smaller set of elements). It is a popular technique in the field of signal processing, in particular in the analogue to the digital signal conversion process, within which analogue measurements are mapped to a smaller set of digital values (Zölzer, 2008). A quantization procedure usually includes two steps, one of distance measurement, and another one of rounding or truncation. In the first step, input values are compared to all available elements and are mapped to the closest one; in the second step, the same values are truncated for assuming a new value, the one attributed to the closest element.

A specific quantization technique, named *Vector Quantization* (VQ) (Burton *et al.*, 1983), was proposed for representing samples composed of multiple measurements (e.g. a time series), according to a reduced number of clusters originating from the same measurements. First, all measurements associated with all samples are grouped in clusters by similarity; second, a centroid is calculated for each cluster; and third, each sample is represented as a histogram of centroid indexes, each index being associated with one measurement contained in the sample.

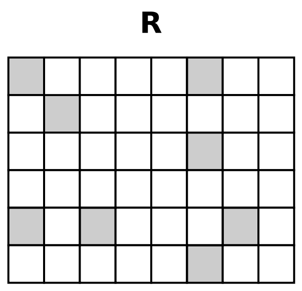
We provide one practical example for the sake of illustration. We have a large set of texts ($t \in T$), each text is composed of a set of sentences ($t = \{s_1, s_2, \dots, s_N\}$), and we want to represent each text according to a reduced vocabulary originated from the content of their sentences. As a first step, (i) all sentences extracted from all texts are grouped in K clusters with the help of a clustering algorithm. Each cluster of sentences has an index in the range $[1, \dots, K]$, and the number of sentences associated with each cluster can vary. As a second step, (ii) K centroids are calculated ($C = \{C_1, C_2, \dots, C_K\}$) for indicating the average content contained in each cluster. And as a third step, (iii) we iterate through all sentences of one specific text, and to each of these sentences, we attribute the index of the closest centroid. A text is now represented as a sequence of centroid indexes ($U = \{u_1, u_2, \dots, u_N\}$), whose elements are calculated by:

$$u_j = \underset{c \in C}{\operatorname{argmin}} \quad \operatorname{dist}(s_j, c) \tag{2.1}$$

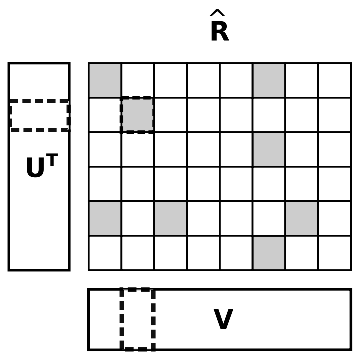
where j is the index of the sentence in the text, and $dist(a, b)$ is a generic function that calculates the distance between a and b. The VQ representation (H) for one specific text with N sentences is expressed as a histogram of size K, whose k^{th} element is calculated with:

$$H_k = \sum_{j=0}^{N} \mathbb{I}[u_j = k]. \tag{2.2}$$

Each text is then represented as a histogram of cluster indexes of size K, that can be interpreted as the likelihood of a text belonging to each sentence cluster.



(a) *Rating Matrix.*



(b) *Matrix Factorization.*

Figure 2.1: *The matrix* **R** *(a) is factorized as the product of two smaller matrices* **U** *and* **V** *(b). Each value R_{ij} of the original matrix is approximated by the inner product between a row u_i^T and a column v_j.*

2.1.2 Matrix Factorization

A factorization, or decomposition, of a matrix, is an equation that expresses this matrix as a product of two or more matrices. Matrices might be factorized in order to simplify the solution of a problem associated with a linear system, as in the case of LU and Cholesky decomposition; they might be factorized for allowing redundancy removal in the context of data compression, as in the case of Singular Value Decomposition (SVD); or they can be factorized for obtaining smaller representations in a dimensionality reduction fashion, as in the case of Alternate Least Square (ALS) (Koren *et al.*, 2009a).

In the specific case of ALS, the original matrix $\mathbf{R} \in \mathbb{N}^{|U| \times |V|}$ is decomposed into two dense and smaller matrices, $\mathbf{U}^T \in \mathbb{N}^{|U| \times K}$, with $K << |V|$ and $\mathbf{V} \in \mathbb{N}^{K \times |V|}$, with also $K << |U|$ (Koren *et al.*, 2009b; Rennie and Srebro, 2005). The original matrix is approximated by $\hat{\mathbf{R}} \approx \mathbf{U}^T\mathbf{V}$ (see Figure 2.1). Smaller matrices are easier to manipulate and dense representations could be used for calculating the similarity between rows or columns with higher accuracy. A probabilistic approach for matrix factorization was also proposed (R. Salakhutdinov and Mnih, 2007), as an alternative for representing latent values as normal distributions.

In the ALS matrix factorization, an optimization process is conducted for minimizing the prediction error of every positive value in the matrix according to:

$$\min_{u,v} \sum_{r_{ij} \neq 0} (r_{ij} - u_i^T v_j)^2 + \lambda(\|u_i\|^2 + \|v_j\|^2), \tag{2.3}$$

where r_{ij} is the actual rating given by the user and $u_i^T v_j$ is the inner product of a row of $\mathbf{U}^T$ and a column of $\mathbf{V}$ corresponding to the predicted values for that rating. The second term in the equation is a regularization factor controlled by a parameter λ, to ensure that the factors u_i and v_j will not grow unreasonably (notice that $\sum_i \|u_i\|^2$ and $\sum_j \|v_j\|^2$ correspond to the squared norms of $\mathbf{U}$ and $\mathbf{V}$, respectively).

2.1.3 Markov Models

Markov Models (MM) can be applied for the task of sequence modelling considering sequential data as a stochastic process over discrete random variables assuming values within a finite set. A Markov Chain (MC) assumes that the probability distribution for the next event depends only on the values of m previous events in the sequence, where m represents the memory of the process. A standard MC of order m can be expressed as

$$p(X_t = x_t | X_{t-1} = x_{t-1}, \dots, X_{t-m} = x_{t-m}), \tag{2.4}$$

where $X_t, \dots, X_{t_m}$ are random variables and $x_t, \dots, x_{t_m}$ their realizations.

In order to represent a MC of order m it is necessary to store all probabilities associated with all possible transitions from a given state $(X_{t-1} = x_{t-1}, \dots, X_{t-m} = x_{t-m})$ to the next value $X_t = x_t$, which requires an $(m + 1)$-dimensional tensor. One limitation imposed by MCs is that the storage space for such a tensor quickly becomes unmanageable when considering every possible transition occurring in the observed data.

2.2 Audio Representation

An audio signal can be represented in several forms, and the decision on which representation to choose will depend on the task considered in the analysis. The simplest representation is the discrete waveform, or *raw waveform* when samples are obtained periodically from the audio signal and stored in a time series format (Oppenheim and Schafer, 2009). The same signal can be also represented in the frequency domain when an emphasis is given to the frequencies of its sinusoidal components. This is achieved through the Fourier Transformation, and different variations that prioritize frequency regions or human physiological aspects are also possible (e.g. *Mel-spectrogram*) Müller *et al.*, 2011. The audio signal can also be represented according to a finite vocabulary obtained from its frequency content, also known as *codewords* (Hoffman *et al.*, 2009; B. McFee *et al.*, 2012; Seyerlehner *et al.*, 2008). Codeword-based representations can be useful in situations where a compact representation is needed. Many hand-crafted features were proposed for several tasks involving music audio signals, as well as combinations of different features for obtaining high-level representations (Bogdanov, Serrà, *et al.*, 2011). With the recent popularization of Artificial Neural Networks (ANN), researchers started experimenting with representations that are meant for one specific purpose, and that is learned during the optimization process performed for one specific task, e.g. chord recognition (Korzeniowski and Widmer, 2016).

2.2.1 Time Domain

An analogue audio signal consists of a continuous function that represents sound pressure as a function of time. A discrete version of that signal can be obtained in the form of a discrete time series by taking periodic measurements. The numeric value of the nth sample in the series is equal to the continuous signal, $x_a(t)$, at time nT (Oppenheim and

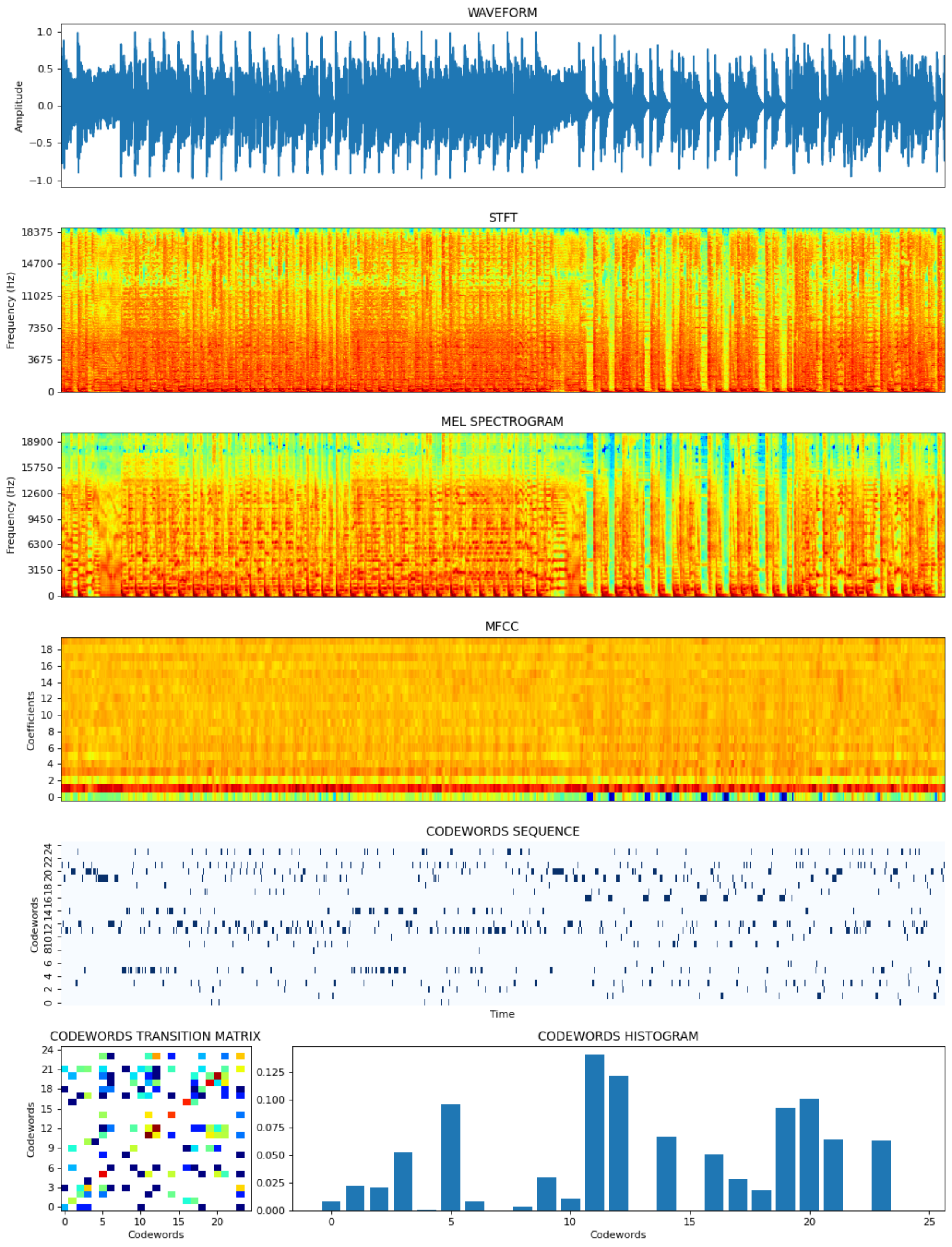


Figure 2.2: *Audio Representations of a 30-seconds excerpt of "Harder, Better, Faster, Stronger" by Daft Punk.*

Schafer, 2009) and is given by

$$x[n] = x_a(nT), \qquad -\infty < n < \infty, \tag{2.5}$$

where T is the *sampling period*. The number of samples measured per second is known as *sampling rate* (SR), and its value is usually selected to be high enough to capture extremely fast variations (i.e. with high-frequency content). The highest frequency that can be detected when adopting a certain sampling rate is equal to half of the sampling rate (SR/2), explained by the *Nyquist Theorem*[1].

The waveform of a 30-seconds excerpt from the song "Harder, Better, Faster, Stronger" by Daft Punk is shown in the first row of Figure 2.2 The waveform was obtained with an SR equal to 22,050 samples per second.

2.2.2 Frequency Domain

Another important representation for an audio signal is its *Frequency Spectrum* (or spectrogram), in which the signal is represented in the frequency domain. The representation of a signal in the frequency domain is obtained by decomposing it into a combination of sinusoidal oscillators. Each oscillator is associated with a complex number containing a value for the magnitude (that indicates the amplitude of the sinusoidal oscillation), and a value for the phase (that indicates the temporal location in the oscillation loop). The decomposition of a discrete signal (as in Equation 2.5) corresponds to the *Discrete Fourier Transformation* (DFT). The Fourier coefficients $X[k]$ are obtained from $x[n]$ in such a way that

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j(2\pi/N)kn}, \quad k = 0, \dots, N-1, \tag{2.6}$$

considering complex exponentials (or sinusoids) with frequencies that are integer multiples of the fundamental frequency $(2\pi/N)$ (Oppenheim and Schafer, 2009).

Short-Time Fourier Transform

It can happen that the frequency components of a signal vary over time, and in this case, a DFT might be applied in a windowed version of the original signal. The windowed version of a DFT is known as *short-time Fourier Transform* (STFT), and it is one of the most common tools for describing the time-varying energy across frequency bands (Müller *et al.*, 2011). Formally, a window function w with size $M << N$ and shifted m samples is multiplied by the original signal, and the DFT of the product is expressed as

$$X[m,k] = \sum_{n=0}^{N-1} x[n]w[n-m]e^{-j(2\pi/N)kn} \quad k = 0, \dots, N-1. \tag{2.7}$$

An STFT representation of the same excerpt mentioned previously is shown in the second row of Figure 2.2. In this specific case, a Hann window function was used, and m and M were set to 2048.

[1] The Nyquist Theorem determines the minimum sampling rate that is necessary for representing a sinusoidal signal of a certain frequency value. For more information about these topics, the reader is referred to Müller, 2015

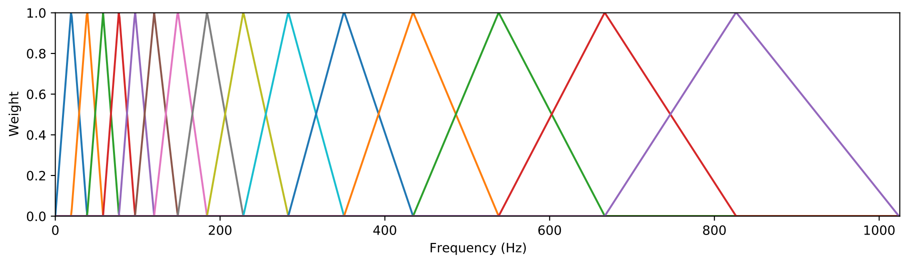


Figure 2.3: *Triangular Mel filter bank with 50% overlap.*

Mel-Spectrogram

The human auditory system distinguishes frequency intervals differently along its hearing range (20 to 20K Hz) (Stevens *et al.*, 1937). In particular, humans can not discern the difference between two closely spaced frequencies, and this effect becomes more pronounced as the frequencies increase.

For this reason, filters with different sizes were proposed for each area of the frequency spectrum according to the Mel Filterbank (Figure 2.3). The most common option is to implement a Mel Filterbank as a bank of triangular filters with 50% of overlapping: the centre frequency of the first filter coincides with the starting frequency of the next one, and the same happens for all filters (Ganchev *et al.*, 2005). The first filter is narrow, and they get wider as frequencies get higher. The filtering process is normalized in such a way that the sum of weights for each triangle is the same.

The Mel Spectrum of $X[k]$ is obtained by multiplying its magnitude by each of the Mel weighting filters:

$$s[m] = \sum_{k=0}^{N-1}[|X[k]| \cdot H_m[k]], \quad 0 \leq m \leq M-1, \tag{2.8}$$

where M is the total number of Mel filters, and $H_m[k]$ is the weight given to the k^{th} energy spectrum bin contributing to the m^{th} output band, expressed as:

$$H_m[k] = \begin{cases} 0, & k < f(m-1) \\ \frac{2(k-f(m-1))}{f(m)-f(m-1)}, & f(m-1) \leq k \leq f(m) \\ \frac{2(f(m+1)-k)}{f(m+1)-f(m)}, & f(m) < k \leq f(m+1) \\ 0, & k > f(m+1) \end{cases} \tag{2.9}$$

where f are the boundary points that specify the M filters (Ganchev *et al.*, 2005). A spectrogram filtered by a Mel Filterbank is a *Mel Spectrogram*. The Mel-spectrogram calculated for the same excerpt mentioned before is presented in the third row of Figure 2.2).

Mel-Frequency Cepstral Coefficients

The so-called cepstral coefficients are calculated as:

$$C[n] = \sum_{m=0}^{M-1} \log_{10}(s[m]) cos\left(\frac{\pi n(m-0.5)}{M}\right), \quad n = 0, \ldots, C-1, \tag{2.10}$$

where C is the number of coefficients. The zeroth coefficient is often excluded since it represents the average log energy of the input signal.

The sound is finally represented as its *Mel-Frequency Cepstral Coefficients* (MFCC) (fourth row of Figure 2.2). In this specific case, the first 20 coefficients are shown.

Codewords

A data-driven audio representation was proposed based on Vector Quantization (VQ) (Section 2.1.1). The main idea is to represent each track from a track set according to a finite vocabulary, named *codewords* (HOFFMAN *et al.*, 2009; B. MCFEE *et al.*, 2012; SEYERLEHNER *et al.*, 2008), learned from the audio features extracted from the whole track set.

We assume a set of tracks ($S \in M$) within which each track is composed of a temporal sequence of audio frames ($S = [s_1, s_2, \dots, s_N]$). Tracks can have different lengths, thus different numbers of audio frames. An audio feature (a) is associated to each audio frame ($f(s_i) = a_i$), and tracks have a corresponding audio feature representation ($A = [a_1, a_2, \dots, a_N]$). The process for calculating a codeword representation can be described in four steps:

- L consecutive audio features are extracted from each track, resulting in $|M| \times L$ extracted features;

- A clustering algorithm is applied for grouping those $|M| \times L$ features in K clusters, and K centroids ($C = [C_1, C_2, \dots, C_K]$) are calculated for representing the average content of each cluster;

- A *Codeword Sequence* ($U = [u_1, u_2, \dots, u_N]$) is obtained from an audio feature representation (A) by calculating the closest centroid to each of its elements.

$$u_j = \underset{c \in C}{\operatorname{argmin}} \|a_j - c\|, \quad 0 < j < N; \tag{2.11}$$

- A *Codeword Transition Matrix* ($T \in \mathbb{R}^{K \times K}$) expresses the probability of transition between consecutive codewords within a sequence ($p(u_j | u_{j-1})$);

- And finally, a *Codeword Histogram* ($H \in \mathbb{N}^K$) summarizes the audio content of a track according to its general codeword composition in the format of a histogram with size K. Each k^{th} element of the histogram calculated for U is obtained with:

$$H_k = \sum_{j=0}^{N} \mathbb{I}[u_j = k]. \tag{2.12}$$

A codeword sequence, a codeword transition matrix, and a codeword histogram representation calculated for our musical excerpt are shown in the fifth and sixth rows of Figure 2.2). The audio feature is the MFCC, the clustering algorithm is K-means and K was set equal to 25.

2.2.3 Task Specific Representations

More recently, methods were designed in such a way that audio representations are learned together with the model parameters, instead of relying on a given filter bank (Jaitly and G. Hinton, 2011; Hamel and Eck, 2010; Dieleman and Schrauwen, 2014; Kim *et al.*, 2018). One argument in favour of those task-specific representations is that the step of calculating high-level audio features, e.g. STFT, can be skipped, thus simplifying the training process (Dieleman and Schrauwen, 2014). Another argument, in the specific case of music-related tasks, is that some of those representations, e.g. MFCC, were first proposed for representing speech signals, with characteristics that are significantly different from music signals (X. Wang and Ye Wang, 2014).

In Dieleman and Schrauwen, 2014 the first layer of an artificial neural network is designed to mimic a log-mel spectrum, and it is learned together with the other parameters of the network. The network was trained for tagging raw audio signals with semantic labels, i.e. auto-tagging. A similar attempt at auto-tagging raw audio signals is described in Kim *et al.*, 2018, but this time convolution operations are applied with higher resolution than before, referred to as *sample-level*. Learned representations are submitted to a careful analysis for identifying properties they might have learned, for example, filtering of a specific frequency range.

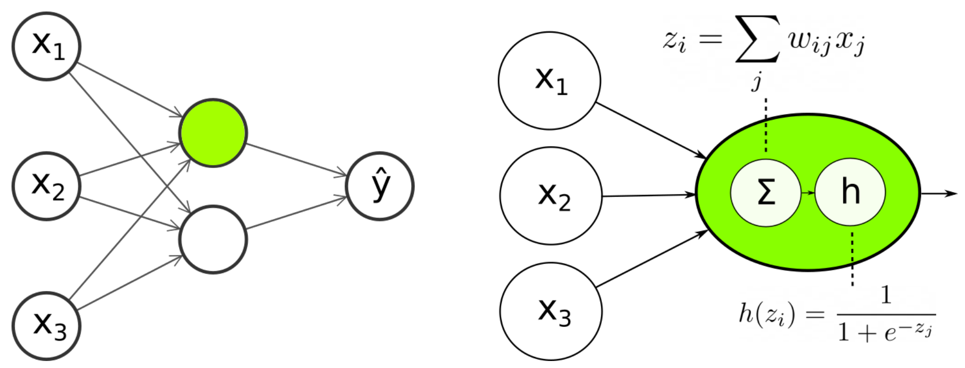
Audio features learned with one specific goal can also be applied to a different task, referred to as *transfer learning* (Hamel and Eck, 2010; Choi *et al.*, 2017). A neural network was trained for recognizing music genres given the DFT extracted from music audios, and the last layer of the network was applied to the task of auto-tagging in Hamel and Eck, 2010. In Choi *et al.*, 2017, a convolutional neural network is trained for mapping Mel-spectrograms to semantic tags, and the network weights are applied as learned features to several music-related tasks, like speech/music classification or emotion prediction. In both cases, the learned representations were responsible for better results than standard hand-crafted features.

2.3 Deep Learning Methods

Artificial neural networks (ANN) gained much attention in the last decades, mainly due to their ability to approximate potentially complex functions in a data-driven fashion. These networks are composed of layers of nodes (perceptrons) that are connected by edges in a graph-like topology, and the networks containing more than one internal layer of nodes are usually referred to as Deep Learning (DL) architectures. These networks have been applied to numerous tasks within several domains, from time series prediction (Chung *et al.*, 2014) to speech recognition (Chorowski *et al.*, 2019), but we are here interested in its application to audio signal processing (Purwins *et al.*, 2019) and to recommender systems (Zhang *et al.*, 2019).

2.3.1 Neural Networks

Neural Networks (NN) are models for approximating functions $y = f(x)$, by combining the outputs of a large number of simple elements within a graph-like topology. Neural



(a) *A Multi-Layer Perceptron (MLP).* **(b)** *A zoom in one perceptron.*

Figure 2.4: *A nonlinear function of a weighted sum of inputs on an artificial neuron.*

Networks thus consist of a combination of two things, a set of *nodes* (also known as perceptrons or units), and a set of directed edges connecting them (Lipton, 2015) (see Figure 2.4). The value v_i of each node (perceptron) i is calculated by applying its activation function h_i to a weighted sum z of the values of its previous nodes j:

$$v_i = h_i\Big(\overbrace{\sum_j w_{ij} \cdot v_j}^{z}_{i}\Big), \tag{2.13}$$

where the weight w_{ij} is associated to the edge connecting j to i.

Typical activation functions h_i are the sigmoid function $\sigma(z) \triangleq 1/(1+e^{-z})$, rectifier linear unit (ReLU) $l(z) \triangleq \max(0, z)$, and hyperbolic tangent function (tanh) $\phi(z) = (e^z - e^{-z})/(e^z + e^{-z})$. Sigmoid functions range from 0 (= $\lim_{z \longrightarrow -\infty} \sigma(z)$) to 1 (= $\lim_{z \longrightarrow \infty} \sigma(z)$), and are monotonically increasing. ReLU improves the performance of neural networks on specific tasks, e.g. in speech processing (Nair and Geoffrey E. Hinton, 2010). And tanh has become a popular option for feedforward neural networks.

Neural networks may be applied to classification tasks, by interpreting output values as probability values, which can be achieved by activation functions. For instance, sigmoid functions already produce values in the range $[0, 1]$ and correspond to probabilities associated with a bell-shaped distribution $\sigma'(z)$.

Feed-forward Networks and Backpropagation

Feed-forward networks are a class of neural networks which organizes the nodes of the network in layers and allows only forward computation in the graph of nodes. This way, the outputs of each layer are considered as the input for the next layer, and this happens successively. The input $\mathbf{x}$ is usually provided by setting the values of the first layer, and the output $\hat{\mathbf{y}}$ is calculated in the last layer. The output is evaluated according to a loss function $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{w})$ that compares the given output $\hat{\mathbf{y}}$ (produced by the network with weights w_{ij}) with the expected output $\mathbf{y} = f(\mathbf{x})$, which is known during the training phase. The learning is accomplished, then, by iteratively updating each weight in the network to minimize the loss function.

The most popular algorithm for training neural networks is *backpropagation* (Rumelhart *et al.*, 1986). This algorithm uses the chain rule to calculate the loss function with respect to each parameter in the network, where weights are adjusted by gradient descent. Currently, neural networks are mostly trained with Stochastic Gradient Descent (SGD) using mini-batches; when considering a mini-batch with size 1, the gradient descent update is:

$$\mathbf{w} \longleftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}, f(\mathbf{x}), \mathbf{w}), \tag{2.14}$$

where $\mathbf{w}$ is the set of weights (network parameters), η is the learning rate and $\nabla_{\mathbf{w}}\mathcal{L}$ is the gradient of the objective function with respect to the parameter $\mathbf{w}$ for a single data sample $(\mathbf{x}, \mathbf{y})$.

In what follows, the process of backpropagation is described step by step. First, a data sample propagates through the network, producing values v_i in each node, and an output $\hat{\mathbf{y}}$ in the last layer. An error value $\mathcal{L}(\hat{y}_k, y_k)$ is computed for each output node k, assuming the output as having k nodes, and the partial derivative is calculated for each incoming edge j according to the chain rule:

$$\Delta w_{kj} = \frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial w_{kj}} = \frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{kj}}. \tag{2.15}$$

considering that $\hat{y}_k = h_k(z_k)$. The first part on the right-hand side expresses how much the loss changes when varying the predicted value for node k, and the second term expresses the dependence of this prediction on the edge coming from j. The first term is denoted as δ_k and it can be expanded, also using the chain rule, as:

$$\delta_k = \frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial z_k} = \frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial h_k(z_k)} \frac{\partial h_k(z_k)}{\partial z_k} = \frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial \hat{y}_k} h_k'(z_k). \tag{2.16}$$

And the second term can be rewritten as:

$$\frac{\partial z_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \Big(\sum_j w_{kj} h_j(z_j) \Big) = h_j(z_j). \tag{2.17}$$

Equation 2.15 is equivalent to:

$$\Delta w_{kj} = \overbrace{h_j(z_j)}^{Equation\ 2.17} \underbrace{\frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial \hat{y}_k} h_k'(z_k)}_{Equation\ 2.16}, \tag{2.18}$$

meaning that the partial derivative of the error function with respect to previous weight w_{kj} is calculated as the product of the error in node k, the derivative of the activation function in that same node, and the current value of node j. The weight values are updated according to a learning rate η that multiples Equation 2.18.

If the node is not an output node, then multiple errors propagate back according to:

$$\frac{\partial \mathcal{L}}{\partial h_i(z_i)} = \sum_x \frac{\partial \mathcal{L}}{\partial z_x} \frac{\partial z_x}{\partial h_i(z_i)} = \sum_x \frac{\partial \mathcal{L}}{\partial z_x} w_{xi}, \tag{2.19}$$

and Equation 2.18 is rewritten as:

$$\Delta w_{ij} = h_j(z_j) h_i'(z_i) \sum_x \frac{\partial \mathcal{L}}{\partial z_x} w_{xi}. \tag{2.20}$$

The equation is used for adjusting all weights in the network starting from the output towards the first layer in the network.

Many variations of SGD can be used to accelerate the learning process. AdaGrad (Duchi *et al.*, 2011) is one of the most popular among these; it adapts the learning rate by caching the sum of squared gradients with respect to each parameter at each time step.

2.3.2 Autoencoders

Autoencoders (AE) are feed-forward networks implemented in an encoder/decoder format, and whose aim is to estimate an output that is as close as possible to the input (G. E. Hinton and R. R. Salakhutdinov, 2006; Sedhain *et al.*, 2015). Typically, the input data is first *encoded* in a compact representation, and after that, this representation is *decoded* back to its original dimension. AE networks usually have at least three layers (e.g. input-hidden-output), and the middle layer is usually referred to as *bottleneck* or *latent representation* (Figure 2.5).

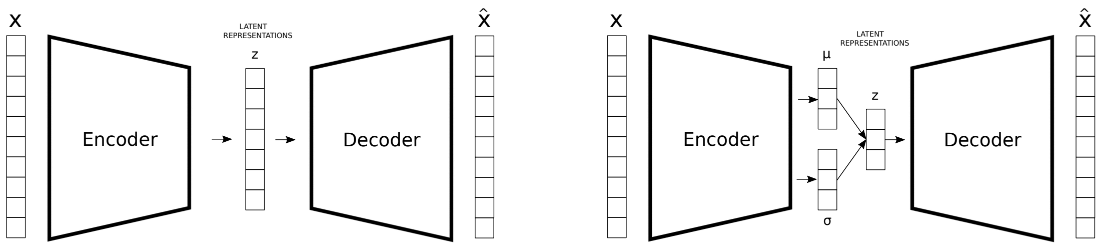
Formally, a AE model assumes an encoder function g parameterized by ϕ and a decoder function f parameterized by θ. The bottleneck layer is calculated with $\mathbf{z} = g_\phi(\mathbf{x})$, where $\mathbf{x}$ is the input data, and the reconstructed input is calculated with $\hat{\mathbf{x}} = f_\theta(g_\phi(\mathbf{x}))$. The training procedure is conducted in order to adjust the parameters ϕ and θ according to a selected loss function, for example, the Mean Squared Error (MSE):

$$\mathcal{L}_{AE}(\mathbf{x}, \theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)} - f_\theta(g_\phi(x^{(i)})))^2, \tag{2.21}$$

considering an input vector of size n, and $x^{(j)}$ as the value in position j in the input vector. A successfully trained model can generate a compact and meaningful representation of the input ($\mathbf{z} = g_\phi(\mathbf{x})$) in a dimensionality-reduction fashion (Kramer, 1991).

The latent representations, however, have no constraint, for example, for maintaining their values within a certain range, or for avoiding overfitting[2]. This motivated the proposition of several adaptations for AE, such as Denoising Autoencoders (Srivastava *et al.*, 2014), Sparse Autoencoders (Makhzani and Frey, 2014), or Contractive Autoencoders (Rifai *et al.*, 2011), among others. We focus on a regularized version of autoencoders, named Variational Autoencoders.

[2] A model is prone to overfitting when it "memorizes" the training data and can not generalize its performance to unseen data, i.e. test data.



(a) *Autoencoder* **(b)** *Variational Autoencoder*

Figure 2.5: *Variational Autoencoder.*

Variational Autoencoders

Variational Autoencoders (VAE) are considered as a probabilistic version of standard Autoencoders (AE), whose latent representations are assumed as Gaussian distributions, and whose encoder/decoder steps are modelled as conditional probabilities (Kingma and Welling, 2019). Assuming latent representations as Gaussian distributions has the desired effect of regularizing the latent space, which enhances the model's generalization ability and, as a consequence, its overall accuracy.

The prior $p_\theta(\mathbf{z})$ is assumed as a Gaussian distribution, $p_\theta(\mathbf{x}|\mathbf{z})$ models the generation of the input data $\mathbf{x}$ conditioned on the latent space $\mathbf{z}$ (decoder), and $p_\theta(\mathbf{z}|\mathbf{x})$ models the estimation of $\mathbf{z}$ from input data $\mathbf{x}$ (encoder). This time, θ denotes the parameters of the distributions to be learned during the training process. See Figure 2.5.

It can happen, however, that the integral of the marginal likelihood $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})dz$ is intractable, meaning that the marginal likelihood can not be evaluated or differentiated. It can happen that posterior distribution $p_\theta(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})/p_\theta(\mathbf{x})$ is also intractable (Kingma and Welling, 2014). The posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is then approximated by a tractable $q_\phi(\mathbf{z}|\mathbf{x})$, defined as $\mathcal{N}(\mathbf{z}; \mu_\phi(\mathbf{x}), \sigma^2_\phi(\mathbf{x}))$, where $\mu_\phi(\mathbf{x})$ and $\sigma^2_\phi(\mathbf{x}))$ are the outputs of the encoder.

Training the model consists of jointly adjusting θ and ϕ for minimizing the Kullback-Leibler (KL) divergence between approximated and original distributions. The KL formula can be expanded as[3]:

$$\begin{aligned}
\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} dz \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} dz \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \Big(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \Big) dz \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} dz \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} dz \\
&= \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \Big[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z}) \Big] \\
&= \log p_\theta(\mathbf{x}) + \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}),
\end{aligned} \tag{2.22}$$

where $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ indicates that $\mathbf{z}$ is generated from distribution $q_\phi(\mathbf{z}|\mathbf{x})$. Re-arranging the terms, we have:

$$\log p_\theta(\mathbf{x}) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \tag{2.23}$$

On the left-hand side of the equation, we have the marginal log-likelihood $\log p_\theta(\mathbf{x})$, which needs to be maximized, and the KL divergence measured between the tractable and the intractable distributions, which needs to be minimized. On the right-hand side we have two corresponding terms, respectively, the expected value of the log-likelihood of generated $\mathbf{x}$ conditioned on $\mathbf{z}$, and the KL divergence measured between the tractable distribution and the prior $p_\theta(\mathbf{z})$.

The Variational Lower Bound (VLB) is defined as the target of a minimization process:

$$\begin{aligned}
\mathcal{L}(\theta, \phi; \mathbf{x}) &= -\log p_\theta(\mathbf{x}) + \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) \\
&= -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})).
\end{aligned} \tag{2.24}$$

When encouraged to learn the first term $\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})$, the VAE is adjusting its predictions to the ground truth data, and when encouraged to learn the second $\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$ it is approximating a theoretical assumed distribution to the one observed in the data.

The log-likelihood is approximated by drawing samples $\mathbf{z}_l$ from $q_\phi(\mathbf{z}|\mathbf{x})$ in such a way it can be expressed as:

$$\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) \simeq \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\mathbf{x}|\mathbf{z}_l). \tag{2.25}$$

But sampling is a stochastic process and gradient can not be backpropagated. In order to

[3] Reproduced from https://lilianweng.github.io/posts/2018-08-12-vae/

make the gradient differentiable with respect to ϕ, the authors in Kingma and Welling, 2014 proposed the reparameterization trick, in which

$$\mathbf{z}_l = \mu_\phi(\mathbf{x}) + \sigma^2_\phi(\mathbf{x}) \odot \epsilon_l, \tag{2.26}$$

where $\epsilon_l \sim \mathcal{N}(0, \mathbf{I})$ and $\odot$ is a element-wise multiplication.

KL divergence can be computed and differentiated without estimation, in the case when the prior $q_\phi(\mathbf{z}|\mathbf{x})$ and the posterior $p_\theta(\mathbf{z})$ approximations are Gaussians (Kingma and Welling, 2014):

$$-\,\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2), \tag{2.27}$$

where J is the dimensionality of $\mathbf{z}$, and μ_j and σ_j are the j-th element of these vectors.

And the new VLB is given by:

$$\mathcal{L}(\theta, \phi; \mathbf{x}) \simeq -\frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\mathbf{x}|\mathbf{z}^l) - \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2), \tag{2.28}$$

where $\mathbf{z}^l = \mu_\phi(\mathbf{x}) + \sigma^2_\phi(\mathbf{x}) \odot \epsilon^l$. The decoding term $\log p_\theta(\mathbf{x}|\mathbf{z}^l)$ will depend on the type of data that is being modelled.

2.3.3 Convolutional Networks

Convolutional Neural Networks (CNN) are variations of feed-forward networks that use convolution in place of general matrix multiplication in at least one layer (Goodfellow *et al.*, 2016). Convolutional layers employ the mathematical operation of convolution (explained in the sequel), which requires the input data to have a grid-like topology. The grid can be 1-D, as in the case of time series; or it can be 2-D, as in the case of images. CNNs became extremely popular in the area of Computer Vision due to their successful performance in a wide range of tasks.

We briefly describe how the input data propagates through a typical CNN with several convolution layers (deep CNN) before entering into details about two important elements: convolutional and pooling layers. First, the input data is submitted to the first convolutional layer, which performs convolution operations given a *kernel function*. The convolution operations generate a set of filtered results, known as *feature maps*. These maps are first submitted to activation functions (*activation layer*), like in the case of feed-forward networks, and then sub-sampled in the following sub-sampling layers, referred to as *pooling layers*. The data propagates until the output layer when an error is calculated, and the weights of the network are then adjusted with backpropagation. A dense layer can be added as a final layer for reducing the dimension of the final output, especially common in the case of classification tasks.

Convolutional Layers

The discrete convolution operation of a function x with a kernel w is denoted as:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a), \tag{2.29}$$

which can be described as the area under the function x weighted by a kernel w shifted by an amount t. In the case where x is multidimensional, convolution is applied on more than one axis at a time. For example, in the case of a 2-dimensional image I and a 2-dimensional kernel K, the convolution is calculated with:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n). \tag{2.30}$$

Two important properties that differentiate CNNs from the feed-forward networks are *sparse connectivity* and *shared weights*. In a conventional feed-forward network, all nodes from one layer are connected to all nodes from the subsequent layer, but in CNNs the connectivity between two successive layers happens through a subset of units. This sparse connectivity is accomplished by making the kernel smaller than the input, which reduces the number of parameters stored and, as a consequence, the memory required for training the model (Goodfellow *et al.*, 2016).

Also, in a convolutional network, each member of the kernel is used at every position of the input, referred to as shared weights. In practice, this means that one set of parameters is learned for the whole input instead of learning a separate set of parameters for every location. This also reduces the memory requirements and is more efficient when compared to conventional matrix multiplications.

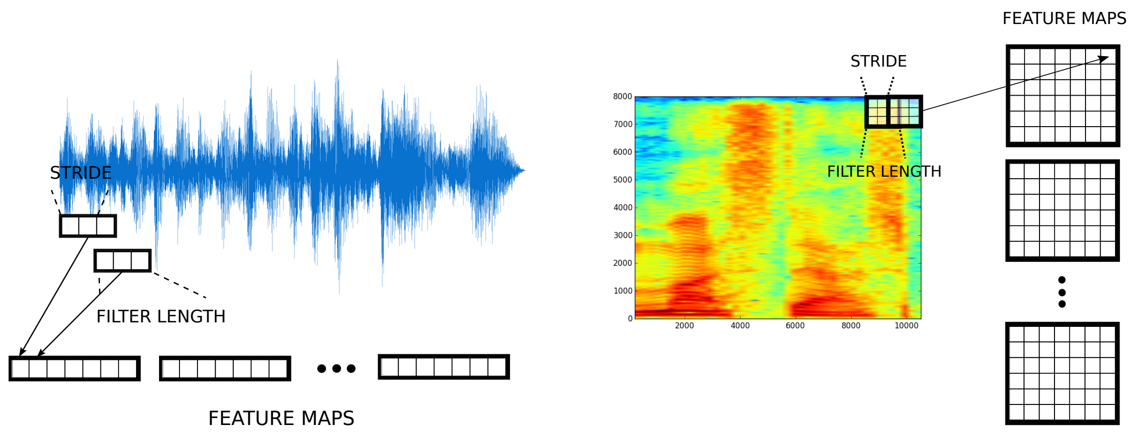
In complex tasks, several kernels[4] might be applied by the same convolutional layer. The motivation for this is that each different kernel can learn a specific function responsible for complementary features in the input. For example, imagine a CNN that is trained for classifying audio Mel-spectrograms according to their corresponding music genres. Drum kicks might be a relevant event for predicting one specific musical genre, and one kernel might end up learning to detect them. Certain harmonic content might be relevant for predicting another musical genre, and another kernel would be necessary for detecting it. Both kernels together would then complement each other in this classification task. In Figure 2.6 we illustrate 1-D and 2-D convolution processes related to audio inputs.

Pooling Layers

The application of one or more kernels to the input data generates one or more feature maps (see Figure 2.6), and these feature maps are submitted to activation functions in order to, for example, convert all values to positive values. The results of an activation layer, with the same dimensions as the original feature map, might need to be reduced to a more compact format, and this reduction is performed by a pooling operation.

A pooling operation replaces several neighbour values in a features map with a value

[4] Kernels can be also referred to as *filters*.



(a) *1D raw audio convolution* **(b)** *2D Mel-spectrogram convolution*

Figure 2.6: *Illustration for 1D and 2D convolutional operations. The filter length determines the number of pixels, in the case of Mel Spectrum, and samples, in the case of the waveform, considered in the convolution operation. The stride value determines the size of the step between convolution operations. A feature map is generated as the result of convolving each filter with the input, and its dimensions depend on the filter size and stride value. We omit the padding and bias for the sake of simplicity.*

that summarizes the original values (Goodfellow *et al.*, 2016). In the case when the input is an image, the whole matrix can be divided into rectangles, and for each rectangle, one value is returned. Max-pooling is a popular option for a pooling operation, and it extracts the highest value among the considered values.

CNNs can be understood as a class of deep models for learning a hierarchy of increasingly complex features. These features might be used in classification tasks for identifying, for example, if an image contains an object or not, with high accuracy. But sometimes the identification task requires detecting events happening on time, and CNNs might present a limited performance. Recurrent networks were proposed taking this specific limitation into account, and are discussed in the sequel.

2.3.4 Recurrent Networks

The main difference between Recurrent Neural Networks (RNN) and conventional feed-forward deep networks is the existence of an internal hidden state inside the units that compose the network (Hidasi *et al.*, 2016). Like feed-forward networks, RNNs have no cycles involving conventional edges, but edges that connect adjacent time steps called recurrent edges. At time t the nodes with recurrent edges receive input from the current data point $\mathbf{x}^{(t)}$ and hidden node values $\mathbf{h}^{(t-1)}$ from the network's previous state (Lipton, 2015). The output $\hat{\mathbf{y}}^{(t)}$ at each time t is calculated given the node values $\mathbf{h}^{(t)}$.

A standard RNN outputs the probability indicating the most likely item to appear as the next one in a sequence, given its current state $\mathbf{h}^{(t)}$. This same hidden state is updated as:

$$\mathbf{h}^{(t)} = \sigma(W^{hx}\mathbf{x}^{(t)} + W^{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h), \tag{2.31}$$

where σ is a smooth function like a logistic sigmoid, $\mathbf{x}^{(t)}$ is the input at time t (represented as a binary vector containing 1 in the position of the current element), and $\mathbf{h}^{(t-1)}$ is the previous state of the hidden state. Matrices W^{hx} and W^{hh} are, respectively, the conventional weights between the input and the hidden layer, and the recurrent weights between the hidden layer and itself in adjacent time steps. Both matrices are adjusted during the training phase.

The output is given by:

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(W^{yh}\mathbf{h}^{(t)} + \mathbf{b}_y), \tag{2.32}$$

where W^{yh} is the matrix containing the weights between the hidden states and the outputs, and $\mathbf{b}_y$ (as well as $\mathbf{b}_h$) is the bias parameter which allows the nodes to learn an offset. The recurrent feedback mechanism memorizes the influence of each past data sample in the hidden state, overcoming the fundamental limitation of MCs.

Learning long-range dependencies with recurrent networks, however, can be challenging (Bengio *et al.*, 1994). Long sequences might correspond to long chains of multiplications occurring in the backpropagation gradient calculations, and gradient values may shrink or expand too rapidly. These were considered as the main barrier for RNNs to succeed in this scenario, known as the problems of *vanishing* and *exploding* gradients.

Truncated backpropagation through time (TBPTT) (Williams and Zipser, 1989) sets a maximum number of time steps along which error can be propagated, when trying to avoid vanishing or exploding gradients. This method addresses the same issues that led to Long Short-Term Memory (LSTM) architectures (Hochreiter and Schmidhuber, 1997). In LSTMs each node in the hidden layer is replaced by a *memory cell*, and each of these cells contains a node with a self-connected recurrent edge of fixed weight, ensuring that the gradient can pass across as many times as necessary without exploding or vanishing.

Gated Recurrent Unit (GRU)

Gated Recurrent Units (GRU) are suitable for modelling long sequence Cho *et al.*, 2014, not being susceptible to the vanishing drawback of standard RNNs. The key distinction between RNNs and GRUs is the mechanism dedicated to updating or resetting the hidden state, known as gating. A reset gate is responsible for controlling how much of the previous state needs to be remembered, and the update gate controls how much of the new state is just a copy of the old state.

According to the same notation presented before, for a given time step t, reset ($\mathbf{r}^{(t)}$) and update ($\mathbf{z}^{(t)}$) gates are calculated as:

$$\mathbf{r}^{(t)} = \sigma(W^{rx}\mathbf{x}^{(t)} + W^{rh}\mathbf{h}^{(t-1)} + \mathbf{b}_r) \tag{2.33}$$

$$\mathbf{z}^{(t)} = \sigma(W^{zx}\mathbf{x}^{(t)} + W^{zh}\mathbf{h}^{(t-1)} + \mathbf{b}_z) \tag{2.34}$$

where $\mathbf{x}^{(t)}$ is the input at that time step, W correspond to weights that will be learned, and $\mathbf{b}$ are biases. Sigmoid is applied to transform the input values to the range (0,1). The reset gate is then integrated into the regular latent space calculation (check Formulas 2.31), which becomes:

$$\mathbf{n}^{(t)} = tanh(W^{nx}\mathbf{x}^{(t)} + W^{nh}(\mathbf{r}^{(t)} \odot \mathbf{h}^{(t-1)}) + \mathbf{b}_n)) \tag{2.35}$$

where $\odot$ is the Hadamard (elementwise) product and $tanh$ is applied to ensure that the values remain in the interval (-1,1). This is also known as a *candidate hidden state*, which still needs to incorporate the update gate. For now, when entries in the reset gate are set to 1, then the new candidate state reminds the hidden state calculated for standard RNN (Formula 2.31). When the reset gate is set to 0 the architecture then resembles a standard MLP having $\mathbf{x}^{(t)}$ in the input.

The new hidden state is calculated with

$$\mathbf{h}^{(t)} = (1 - \mathbf{z}^{(t)}) \odot \mathbf{n}^{(t)} + \mathbf{z}^{(t)} \odot \mathbf{h}^{(t-1)}, \tag{2.36}$$

where $\mathbf{h}^{(t-1)}$ is the hidden state at time $t-1$. The update gate $\mathbf{z}^{(t)}$ determines to which extent the new hidden state $\mathbf{h}^{(t)}$ is inherited from the previous hidden state $\mathbf{h}^{(t-1)}$, and how much of the new candidate state is considered. When $\mathbf{z}^{(t)}$ is close to 1, the previous state is maintained, and the input $\mathbf{x}^{(t)}$ is ignored. When the update gate is set to 0, the new hidden state incorporates the candidate state. Briefly, GRUs have reset gates that help to capture short-term dependencies in sequences, and update gates that help capture long-term dependencies in sequences.

2.4 Rating-Based Recommendation

Recommender systems were originally proposed for helping users in situations of information overload, i.e. when users have too many options available and need to take a decision. A typical recommender is capable of leveraging information on someone's past decisions and suggesting future actions that match their preferences. These systems became popular in the last decades and were applied in many domains like movies, books, news, music, etc.

In this Section, we review the recommendation methods considered the most relevant for this research. These methods are *rating-based*, meaning that they were designed for dealing with user/item interaction data, i.e. ratings, and we separated the methods into three categories: collaborative filtering, sequence-aware and stream-based. We discuss the assumptions and limitations of each category, and we highlight their application to the music domain.

The assumptions for designing recommenders can vary significantly from one domain to the next, and when considering music as the item to be suggested, there are three main specificities that should be considered. First, (i) the duration of a standard song is relatively small if compared to other items like movies, books, and trips, but somehow similar to the time spent when reading articles online. Second, (ii) users repeat songs intentionally and frequently, which impacts the design of prediction algorithms, especially the ones that take the order of events into account. And finally, (iii) music preferences can vary depending on the context, which also imposes new challenges to the automatic recommendation. This might be also the case when considering movies, but still, someone's preference for movies will hardly vary as much as when having a playlist to go to the gym and another one for meditation.

	Song 1	Song 2	Song 3	Song 4	Song 5
User 1	5			2	4
User 2	4	2		3	3
User 3	4	3	5		1
User 4	3		2		

Table 2.1: *Explicit feedback user rating matrix example. Ratings values can vary from 1 to 5.*

2.4.1 Collaborative Filtering

Collaborative Filtering (CF) was first proposed as an alternative for filtering content in a shared communication environment (Goldberg *et al.*, 1992). Back in those days (1990's), users were already overloaded with the number of documents available, and the idea of storing individual feedback was suggested as a resource for filtering uninteresting documents. Formally, users $u \in U$ have access to documents $v \in V$, and the reaction of each user i to each document j is stored in a matrix $\mathbf{R} \in \mathbb{N}^{|U|\times|V|}$, the rating matrix. Each reaction of user i to the item j was considered as a rating r_{ij} (for example between 1 and 5) and the empty values can mean either that the user didn't like one specific item, or that the item was not presented to them.

In the strategy referred to as *explicit feedback*, the rating given to an item is directly provided by the user, and lies within a pre-specified range of values, as illustrated in Table 2.1. The numbers in the matrix indicate the ratings given by a user to a song, and blank cells indicate user-song interactions that haven't happened yet. The recommendation system will then use known rating values to infer the unknown (empty) ratings (this is the prediction phase), in order to recommend the next song in a listening session (recommendation phase).

A first proposition for applying CF assumed that each row $u_i \in \mathbb{N}^{|V|}$ of the rating matrix represented a user profile, indicating preferences for each item, and that similar profiles corresponded to similar preferences. An automatic recommender, known as user-based (Ahmad Wasfi, 1998), was able to gather similar users (i.e. users with similar profiles) and suggest new items to a specific user based on its neighbour profiles. It was also possible to calculate the similarity between items, considering item profiles ($v_j \in \mathbb{N}^{|U|}$), which was referred to as item-based recommendation (Sarwar *et al.*, 2001).

Many approaches were proposed for implementing collaborative recommendations, Bayesian networks, k-nearest neighbours, clustering, and graph-based, among others (Breese *et al.*, 1998), the majority of them focused on the selection of sets of similar users given the consumption profiles. But dealing with extremely big rating matrices turned out to be challenging and inefficient, for example, when calculating the correlation between sparse vectors (users who interacted with few items).

Deep Learning (DL) techniques were recently adapted for the task of inferring users' preferences collaboratively (H. Wang *et al.*, 2015; He *et al.*, 2017), and proved to be especially interesting for modelling non-linear and non-trivial user-item relationships. These

DL techniques are also able to calculate complex representations of the input data due to the characteristic of stacking as many layers as necessary, which is also the reason they are called deep architectures.

The authors in Zhang *et al.*, 2019 point out three main benefits and three drawbacks of applying deep methods in the task of recommendation. According to them, one of the most attractive properties is the fact that they are end-to-end differentiable and provide inductive biases tailored to the input data type. Also, these methods can learn representations that are specific to the task being addressed, bypassing the necessity of hand-crafted or modality-specific features. On the other hand, these methods are widely known for lacking interpretability, requiring significant amounts of data (enough to model a phenomenon), and also for requiring extensive hyperparameter tuning.

We now review some of the most relevant methods for recommendation based on collaborative filtering.

Weighted Matrix Factorization

A factorization model was proposed in Hu *et al.*, 2008, in which there is a distinction between *preference* and *confidence*, which is especially useful in situations where the rating information available corresponds to the number of interactions (e.g. a number of clicks), also known as *implicit feedback*[5]. The new optimization formula is given by:

$$\min_{u,v} \sum_{r_{ij}} c_{ij}(p_{ij} - u_i^T v_j)^2 + \lambda \left(\sum_i \|u_i\|^2 + \sum_j \|v_j\|^2 \right), \tag{2.37}$$

where p_{ij} stands for (binary) preference, and c_{ij} is a confidence parameter. Confidence can take two different forms, a linear model $c_{ij} = 1 + \alpha(r_{ij})$ or a logarithmic model $c_{ij} = 1 + \alpha \log(1 + r_{ij}/\epsilon)$, where α is a factor that multiplies the ratings in order to differentiate them from 0 values, and ϵ is a compensation which depends on how frequently users interact with the same item in the dataset.

By differentiation an analytic expression for u_i is given by (Hu *et al.*, 2008):

$$u_i = (V^T C^i V + \lambda I)^{-1} V^T C^i p(i), \tag{2.38}$$

where C^i is a diagonal matrix, within which $C^i_{jj} = c_{ij}$, V is the matrix containing all v_j, $p(i)$ contains all preferences for user i and λ is the learning rate. Next, matrix V is updated in a similar way, according to:

$$v_j = (U^T C^j U + \lambda I)^{-1} U^T C^j p(j). \tag{2.39}$$

Both matrices, U and V, are updated alternately until convergence.

From this perspective, this can be seen as a classification task, where each item is classified as 0 or 1, but the idea of providing users with a list of potentially interesting items comes closer to the recommendation objective, which corresponds to sorting items

[5] When considering binary implicit feedback, user preferences are indicated as 1's, but 0's do not necessarily indicate that the user dislikes an item, they could also mean that the item was not presented to them.

according to relevance and presenting the most relevant ones in the first positions of the list.

In the specific domain of music recommendation, the number of tracks available for recommendation is usually large and much larger than the number of users. This leads to sparse rating matrices with high dimensions. The WMF technique can be useful in such situations, not only for producing recommendation lists but also for generating dense representations of both users and tracks to be applied in other tasks.

Variational Autoencoders

Recently, Variational Autoencoders (VAE) were adapted for the task of CF with competitive performance. VAE-CF (Liang *et al.*, 2018) is considered the state-of-the-art CF recommender system, due to its accuracy, scalability and robustness when dealing with extremely large datasets. User features are learned from the data, in what is known as the encoder phase, before propagating these through the decoder, where scores are actually attributed to each item.

VAE-CF loss function is the same as the one presented in Equation 2.24, except that now an extra hyperparameter (β) is included for controlling the strength of KL divergence in the optimization process. The new loss function can be expressed as:

$$\mathcal{L}(\theta, \phi; \mathbf{x}) = -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + \beta \cdot \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}) || p_\theta(\mathbf{z})). \tag{2.40}$$

The new hyperparameter β is gradually introduced during the training process in such a way that the network is first adjusted to the first term on the right-hand side of the equation, before being introduced with the second term. This allows the network to first prioritize the quality of the input reconstruction ($\log p_\theta(\mathbf{x}|\mathbf{z})$), before focusing in adjusting its hidden distribution ($p_\theta(\mathbf{z})$) to the one observed in the data ($q_\phi(\mathbf{z}|\mathbf{x})$).

From a user profile presented in its input, the VAE-CF encoder produces a latent representation corresponding to one line in the user matrix U in the case of matrix factorization techniques. If a VAE-CF autoencoder is trained with columns of a rating matrix (item profiles), the latent representations can be thought of as corresponding to columns of the item matrix V.

In Liang *et al.*, 2018, the latent representation of a single user ($\mathbf{z}_u$) is transformed by a nonlinear function $f(\cdot) \in \mathbb{R}^N$, to produce a probability distribution $\pi(\mathbf{z}_u)$ over N items, and the multinomial log-likelihood is given by:

$$\log p(\mathbf{x}_u|\mathbf{z}_u) = \sum_{i=0}^{N} x_{ui} \log \pi_i(\mathbf{z}_u). \tag{2.41}$$

The authors compared this likelihood with two other ones: the same one used in the WMF method, referred to here as a Gaussian likelihood, and a logistic log-likelihood. The multinomial likelihood, however, presented the best overall results and is the one used in our experiments.

Final Considerations

The methodology for testing CF methods consists in removing a small number of known ratings from the original data and applying the prediction method to the remaining ones. After the model is trained, the values removed from the factorization are predicted, and the accuracy is measured as how many of them were correctly predicted.

Some assumptions and limitations of CF recommendation models are:

Assumptions :

- All tracks ever rated or listened to by a user is considered as a listening profile.
- Profiles are assumed as a proxy for preference, and similar profiles are assumed as a similarity of preference among users.

Limitations :

- New users and new items are difficult to incorporate into the algorithm, because of the lack of historical data.
- The sequence in which users listened to tracks is never considered.
- CF relies on the similarity and completeness of user profiles, i.e. theoretically it is as accurate as the amount of listening data contained in the dataset. This assumption can lead to expensive models, that work exclusively in big datasets.

2.4.2 Sequence-Aware

CF methods consider previous interactions of a user as a static profile, that serves as a proxy for preference, and assume similar profiles to indicate affinity between users' musical tastes. But static profiles might be a limited resource for modelling a music listening phenomenon, mainly because they do not consider its corresponding temporal dynamics: people usually listen to several tracks in a row, and it might be reasonable to assume that their sequence is a key factor for a fruitful listening experience.

Differently from the previously described methods, sequence-aware (SA) recommender systems have the ability to handle temporal context information (Quadrana *et al.*, 2018; Jannach, Mobasher, *et al.*, 2020). They consider user interactions as sequences, and recommendation as the task of, giving a slice within a sequence of listened tracks, predicting the next item appearing after the slice. Formally, let $V = \{v_1, v_2, \dots, v_m\}$ denote the set of all songs available in the system. A sequence s is represented as a list $[v_{s,1}, v_{s,2}, \dots, v_{s,n}]$, ordered by the timestamps of the user interactions, where each $v_{s,i} \in V$. The aim of the SA recommender is to predict the song $v_{s,j+1}$ within the sequence, given a slice $[v_{s,i}, v_{s,i+1}, \dots, v_{s,j}]$. The method outputs a probability distribution $\mathbf{p_k} = \mathcal{P}\{v_{s,j+1} = v_k\}$, $\forall k$, and the top-K values as potential candidates for the next item.

The position of the correctly predicted item ($v_{s,j+1}$) in the top-K list has now a different meaning than it had in the collaborative filtering context. There, probabilities reflect a user's global preference for each song, whereas here they reflect the adherence of each song to a specific position in the sequence of previously heard items. An accurate SA recommender is expected to retrieve the correct item in the first positions of the Top-K list, preferably with the lowest possible index. For this reason, it makes perfect sense to use ranking-related metrics for measuring the performance of SA recommenders.

One interesting characteristic of SA methods is their assumption of temporal patterns observed within listening sessions as the resource for making recommendations, regardless of the user who listened to each session. This provides these methods with the ability to incorporate new users (user cold-start) without any extra effort. Remember that in the CF case user and item cold-start were presented as potential problems for the recommendation algorithms, due to the collaborative strategy applied by them.

Markov Models

Markov Models (MM) can be applied for the task of sequence modelling (see Section 2.1.3) and thus, can be also applied for the task of SA recommendation. When MMs are applied for the task of SA recommendation, states are assumed as tracks and the transitions between these states correspond to transitions between consecutive tracks observed in listening sessions.

A common option is to set the memory m equals to one (RENDLE, FREUDENTHALER, *et al.*, 2010), meaning that each listened track depends exclusively on the previous one. When that is the case, then $m = 1$ and the probability of the next track (x_t), given the previous one (x_{t-1}) is given by:

$$p(X_t = x_t | X_{t-1} = x_{t-1}). \tag{2.42}$$

A first-order MC models transitions based on pairs of consecutive tracks appearing in sessions, and predicts the next track based on the most likely transition from the state associated with the current track (ZIMDARS *et al.*, 2001). This can be implemented as a lookup table where transitions are stored, and a transition probability can be consulted very efficiently.

GRU4REC

The first deep learning approach for predicting the next item within a listening session was GRU4REC (HIDASI *et al.*, 2016), based on recurrent neural networks, or more specifically on Gated Recurrent Units (GRU[6]). In this approach, each item (track) within a session is represented as an indicator vector of dimension #tracks, and the model is trained for predicting an indicator vector corresponding to the next item, depending on the current state of the network.

GRU layers are arranged as in Figure 2.7, where each layer stores a hidden state that encodes some previously occurring item in the session. Items are fed according to their sequence in the session, and hidden states are reset after each session. The authors report that the best results are obtained using a single GRU layer, which can be thought of as a Markovian process of order 1.

[6] More details on GRU architecture are brought in Section 2.3.4

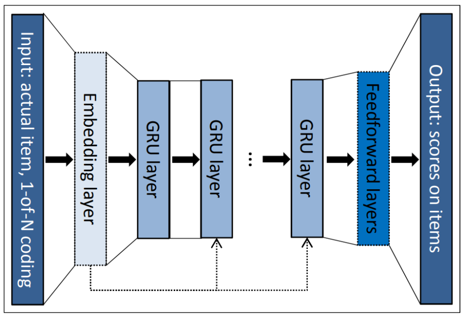


Figure 2.7: *GRU4REC applies Recurrent Neural Networks for predicting the next item in the sessions given the current one (figure reproduced from* Hidasi *et al., 2016).*

A *session-parallel mini-batch* strategy is proposed for handling several sessions in parallel. This strategy arranges sessions with different lengths in batches, in such a way that whenever a session ends, a new one is positioned in the same place in the batch for maintaining continuity. For each new session, the hidden state is reset so the model is not learning about transitions between sessions.

The authors propose two loss functions which have a great impact on their results. Instead of calculating the score for each item at each round of training, as would be usual, samples are taken from the output and the loss associated with a small subset of items is calculated. Items are sampled according to their popularity and negative samples are taken from the previous mini-batch for saving time.

The model can be trained with stochastic gradient descent (SGD) using the following loss functions:

- BPR: The score of a positive item is compared with several negative ones, and their average is considered as the loss. Formally, the loss is calculated with: $L_S = -\frac{1}{N_S}\sum_{j=1}^{N_S}\log(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j}))$, where N_S is the sample size, $\hat{r}_{s,k}$ is the score of item k in session s, i is the desired item, and j are the negative samples.

- TOP1: The loss is considered as a regularized approximation of the relative ranking of the positive item. The relative rank of the relevant item is given by $\frac{1}{N_S}\sum_{j=1}^{N_s} I\{\hat{r}_{s,j} > \hat{r}_{s,i}\}$, which indicates that the relevant item is in a lower position compared to other items, and an extra regularization term forces negative samples to have scores close to zero. The final loss function is given by: $L_S = \frac{1}{N_S}\sum_{j=1}^{N_S}\sigma(\hat{r}_{s,j} - \hat{r}_{s,i}) + \sigma(\hat{r}_{s,j}^2)$.

Neural Attentive Recommendation Machine (NARM)

Another method applicable to SA recommendation, named Neural Attentive Recommendation Machine (NARM) (J. Li *et al.*, 2017), combines the sequential behaviour of a user with an assessment of the main purpose of the current session, identified with the most relevant item browsed by the user. The authors argue for the necessity of considering the main purpose of a user's session as a relevant factor for the recommendation, considering that implicit feedback can sometimes be noisy, for example when an item is clicked by mistake or curiosity. In NARM, sessions are fed to the network in a cumulative fashion, where a session $[x_1, x_2, \cdots, x_{n-1}, x_n]$ is represented by $n - 1$ sub-sessions

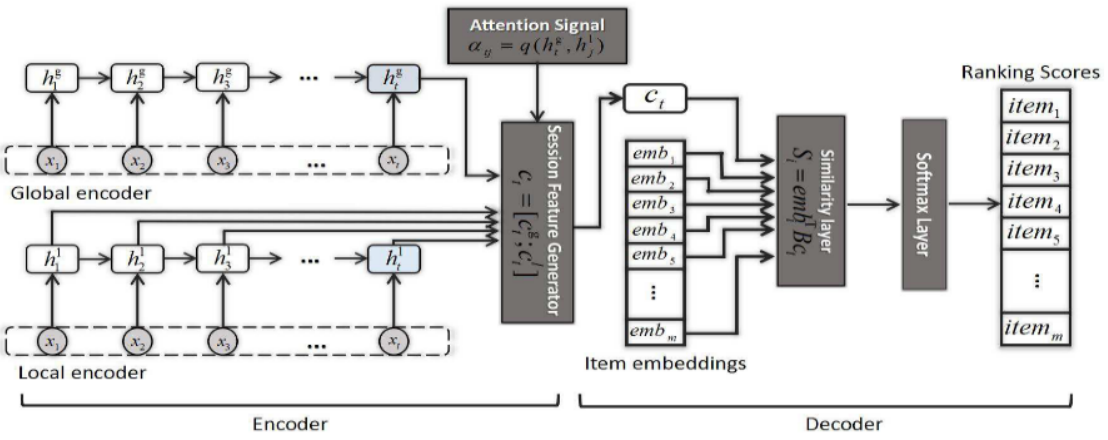


Figure 2.8: *Neural Attentive Recommendation Machine (NARM) scheme (figure reproduced from J. Li et al., 2017).*

$[x_1, x_2, \cdots, x_{n-1}, x_j],\ j = 2, \dots, n.$

The NARM architecture combines two independent encoders, responsible for calculating two representations, one local that highlights the most relevant item within the session, and one global corresponding to a standard recurrent representation, as in Figure 2.8. The steps of the method are as follows:

1. sessions are fed in the input and submitted to a GRU-based recurrent network (global encoder). The last hidden state ($\mathbf{h}_t$) is stored as a general representation of the session $\mathbf{c}_t^g$;

2. An item-level attention mechanism (local encoder) is applied for matching the current state of the network with each item in the input, and for focusing on important items (trying to identify the main purpose of the current session). A function $q(\mathbf{h}_t^g, \mathbf{h}_j^l)$ performs a linear combination of the current state with each input item, generating a local representation $\mathbf{c}_t^l$;

3. Both representations, $\mathbf{c}_t^g$ and $\mathbf{c}_t^l$, are concatenated as $\mathbf{c}_t$, considered as features of the session. $\mathbf{c}_t$ is first submitted to an inner product with an auxiliary matrix ($\mathbf{B}$), and the product of this inner product is multiplied by item embeddings (emb). Both $\mathbf{B}$ and emb are adjusted during the training. The final formulation of the last phase is given by $S_i = emb_i^T \times \mathbf{B} \times \mathbf{c_t}$ for each item i;

4. A softmax layer is applied to the resulting S_i and produces scores for each item.

The model is trained with gradient descent, on a cross-entropy loss:

$$L(p, q) = -\sum_{i=1}^{m} p_i \log(q_i), \tag{2.43}$$

where q is the predicted probability distribution and p is the true distribution.

Final Considerations

Some assumptions and limitations of SA recommendation models are:

Assumptions :

- Sequence-aware methods are able to detect patterns in sequences of items observed in users' sessions and assume these patterns as candidates for future actions.
- Some methods assume listening sessions (in the specific case of music recommender) as anonymous and this alleviates the problem of new users entering the platform.

Limitations :

- The methods are not personalized, as in the collaborative case, and this can potentially lead to undesired consequences, of recommending items users didn't like previously.
- New items are, again, hardly incorporated in the algorithm, because of not having historical data available.
- Recommendation models are learned offline, regardless of the feedback provided by the user during the listening session

2.4.3 Stream-Based

The majority of experiments designed for evaluating CF and SA recommender systems are usually considered in an offline fashion: the model is first trained according to a loss function, and then tested in an unseen slice of the data. But in a more realistic scenario, models are updated after each user's feedback, and their performance is evaluated by summing the accuracy associated with each consecutive recommendation round. This configuration, named *stream-based* (SB), requires the models to be updated in real-time, in such a way that relevant suggestions start being delivered to users as soon as possible.

In an attempt to adapt existing methods to the stream-based scenario, some authors proposed an incremental CF method (Papagelis *et al.*, 2005), in which a matrix is built for storing the similarity between users, and is updated whenever a user finishes interacting with one item. The similarity between two users is understood as the number of items they have in common in their listening profiles, and when a recommendation for one specific user is needed, the most similar profiles can be readily obtained. The same procedure could be also applied for storing similarities between items, and in this case, the next item is selected based on the information about co-occurrence in user profiles or listening sessions (Miranda and Jorge, 2009). Several dynamic local models are frequently updated for delivering online recommendations to groups of users gathered by similarity in Al-Ghossein, Abdessalem, and Barré, 2018. A global model is also maintained and updated, and both suggestions, calculated by the local and by the global model, are combined in a final top-K list for a target user.

Neighbourhood-based (NB) methods were also adapted for handling real-time updates and were considered as competitive baselines for evaluating next-item recommendation algorithms (Jannach and Ludewig, 2017). Given an ongoing session, the main idea is to find the most similar past sessions containing those elements and use them for calculating recommendations. The similarity function compares two sessions represented in the item space, instead of performing item/item comparisons as in CF methods. In order to allow efficient lookups for online recommendations, the authors propose storing sessions in an in-memory index structure (cache) (Jannach and Ludewig, 2017; Ludewig and Jannach, 2018). Variants of this method, that emphasize the more recent events of a session by weighting the items according to the timestamp of the interaction, were proposed in Ludewig and Jannach, 2018; Garg *et al.*, 2019.

Dynamic matrix factorization methods were also proposed as an alternative for suggesting items to users and incorporating their feedback instantly (Rendle and Schmidt-Thieme, 2008; Vall *et al.*, 2019; Zhao *et al.*, 2013). The main idea behind these approaches is to first, initialize users and items latent variables, and then update a new user (item) latent variable, whenever a new user (item) is added in the system (Rendle and Schmidt-Thieme, 2008; Vall *et al.*, 2019). But dynamic matrix factorization approaches do not include strategies for actively incorporating new users or items: new users or items correspond to empty columns or rows in the rating matrix, and these can not be incorporated into the algorithm unless they are actively suggested to new users, in the case of items, or are offered new items, in the case of users.

A similar technique is explored in an interactive recommendation environment when item features are learned a priori and maintained fixed, while new users are included in the recommendation algorithm for simulating a user cold-start situation (Zhao *et al.*, 2013). Items' and users' latent variables are modelled as normal distributions, and different strategies of item selection are compared taking the exploration/exploitation dilemma into consideration. This time, not just latent variables are updated dynamically, but uncertainty (the variance estimated for the normal distribution) is incorporated into the item selection procedure, according to the Upper Confidence Bound (UCB) principle. It is worth mentioning that in this specific case cold-start users are considered as users with short profiles (users who rated few items), and not as new users which have just entered the platform.

Multi-Armed Bandits (MAB) became a popular setup for designing dynamic CF methods, due to their formulation proposed in the context of active learning in dynamic environments. The strategy is adapted to recommender systems considering decisions as suggestions of items and considering the reward as corresponding to users' satisfaction (Sanz-Cruzado *et al.*, 2019). The rewards obtained from taking decisions can sometimes be modelled as probability distributions Kaufmann *et al.*, 2012 and can be also assumed as changing over time, described as non-stationary bandits (Auer *et al.*, 2019). Decisions can also be taken considering the current context, i.e. a vector containing information about the target of the recommendation, known as *Contextual Bandits* (L. Li *et al.*, 2010).

Adapting MAB to recommendation systems, however, can have some specificities. First, the number of decisions (arms) can't be too large, for example, when considering the act of recommending one single track as corresponding to a decision of a music recommender. In such cases exploring all possibilities and identifying the optimal ones can take too long, and it even may not be viable. Dealing with a more restricted set of options was proposed in Sanz-Cruzado *et al.*, 2019, in which arms were modelled as users in the context of a nearest-neighbour recommender; or in S. Li *et al.*, 2016, in which items are grouped in clusters by similarity and suggestions are made according to historical interactions between users and clusters.

Another limitation of applying MAB to recommendation systems is the correspondence between identifying the best items to be recommended to each user and learning an independent model per user. Learning independent models does not comprise the dependence among users, considered the main characteristic of collaborative strategies. In order to address this limitation, a Collaborative Filtering Bandits was proposed (S. Li *et al.*, 2016), in which users and items are separated into clusters and at each recommendation round the rewards are used to update the composition of such clusters.

Here, we restrict ourselves to methods that can operate in stream-based scenarios, and that can be adapted to the task of providing next item recommendations, aligned with the objective of SA methods. For this reason, we decided to consider SKNN, VSKNN and STAMP methods as baselines in our experiments. Those methods are explained in more detail in the sequel.

SKNN

In Session-Based kNN (SKNN), the main idea is to find the k most similar past sessions N_s containing the same elements of a given session s. This is achieved by comparing their binary vectors over the item space using any suitable similarity measure $sim(s_1, s_2)$, such as the cosine similarity or the Jaccard distance (Ludewig and Jannach, 2018). The score of a recommendable item i for session s is defined by:

$$\text{score}_{\text{SKNN}}(i, s) = \sum_{n \in N_s} \text{sim}(s, n) \times \mathbb{I}_n(i), \tag{2.44}$$

where $\mathbb{I}$ is 1 when $i \in n$ and 0 otherwise. SKNN method, however, does not consider the order of the elements in a session.

VSKNN

Vector Multiplication Session-Based kNN (V-SKNN) (Ludewig and Jannach, 2018) is a variant of SKNN that emphasizes the more recent events of a session by weighting the items, in such a way that the weight of the last item of the session is 1, and the values decay according to a linear function. The V-SKNN method uses the dot product between the weight-encoded ongoing session vector and the binary-encoded past session as the similarity function.

$$\text{score}_{\text{VSKNN}}(i, s) = \sum_{n \in N_s} \frac{\overrightarrow{s_w} \cdot \overrightarrow{s_n}}{\sqrt{l(s) \cdot l(s_n)}} \times \mathbb{I}_n(i), \tag{2.45}$$

where $\overrightarrow{s_w}$ is the weighted version of the ongoing session, $\overrightarrow{s_n}$ is the binary version of a candidate session n, and $l(\cdot)$ denotes the length of the session.

STAN

The *Sequence and Time Aware Neighborhood* approach (STAN) is proposed in Garg *et al.*, 2019 to extend SKNN method in order to incorporate sequential information. As in V-SKNN, the method also uses decay factors, but in multiple levels. The new score is calculated with:

$$\text{score}_{\text{STAN}}(i, s) = \sum_{n \in N_s} \underbrace{\frac{\overrightarrow{s_w} \cdot \overrightarrow{s_n}}{\sqrt{l(s) \cdot l(s_n)}}}_{sim_1(s,s_n)} \overbrace{e^{\left(\frac{t(s)-t(s_n)}{\lambda_2}\right)}}^{w_2(s_n|s)} \underbrace{e^{\left(-\frac{|p(i,n)-p(i^*,n)|}{\lambda_3}\right)}}_{w_3(i|s,n)} \mathbb{I}_n(i), \tag{2.46}$$

where $p(i, s)$ denotes the position of item i in session s, $t(s)$ denotes the timestamp of the most recent item in s, i^* is the index of an item observed in both s and in n sessions, and λs are fixed and positive values. sim_1 ensures that recent items are more relevant, w_2 ensures that sessions farther from session s are assigned with lower weight, and w_3 ensures that items occuring in both s and n are also weighted according to how recently they were observed.

Final Considerations

SB methods do not have a specific strategy for actively incorporating new items in their algorithms, nevertheless, they rely on calculations of similarity between sessions, allowing new items to become recommendation candidates as soon as they participate in a first listening session. While in the case of CF and SA the models would need to be retrained when a new item is added to the item set, now this is accomplished by forcing new items to take part in their first listening sessions. This is not yet a strategy to overcome the cold-start problem, but it alleviates the issue.

SB methods are usually evaluated with experiments that simulate a dynamic recommendation situation. Instead of training and testing the models with different slices of the data, now the recommenders start without any knowledge of users' preferences, and after each interaction, the models are updated with the most recent data. Their performances are usually evaluated by summing the overall accuracy and two main desired metrics can be calculated: how fast the model learns to make relevant recommendations, and the overall accuracy achieved along all sessions.

Some assumptions and limitations observed for SB recommendation models are:

Assumptions :

- Stream-based methods assume user/item interaction data arriving as data streams, and thus, dynamic updating and retrieval procedures are usually applied;

Limitations :

- No specific strategy is proposed for incorporating new items in stream-based recommendation algorithms;

- Retrieving procedures that compare a current element with all elements to find the best option are usually unfeasible in the case of large catalogues of items, as music catalogues usually are;

2.5 Audio-Based Music Recommendation

Typically, music recommender methods rely on user/track interaction data for calculating suggestions of tracks, known as rating-based methods. But it can also happen that music recommender methods use the audio signals associated with each track for calculating recommendations to users, known as audio-based methods.

The audio signals associated with the tracks might be used by these methods in two possible ways: the audio signals can serve as the primary resource for providing recommendations, or they can be used as an auxiliary source of information by rating-based strategies. In the former case, tracks are suggested based on similarities measured within the audio domain, which prevents the cold-start problem but imposes severe limitations on the quality of recommendations (Flexer *et al.*, 2010). In the latter case, information extracted from the audio signal is incorporated in methods that were originally designed to operate with user/item interaction data, for alleviating situations of item cold-start (Forbes and Zhu, 2011).

Music recommenders that rely solely on audio information have considered the recommendation task as a task of selecting tracks from an audio-based representation space according to a set of tracks a user has listened to before or is currently listening to (Cano *et al.*, 2005a). This representation space is built in such a way that similar tracks are supposed to be located next to each other, which makes strategies for measuring similarity between pairs of tracks an essential choice in this context (Slaney *et al.*, 2008; Logan and Salomon, 2001; Bogdanov, Serrà, *et al.*, 2011; Hoffman *et al.*, 2008). A clear definition of what it means for two audios to be similar to each other, however, has never reached a consensus and is beyond the scope of this work.

Audio files can be also applied by rating-based methods for helping them mitigate the cold-start limitation. In Q. Li *et al.*, 2004, users are clustered according to listening habits, and track audios are clustered into music genres. Preferences may be modelled for each user cluster and musical genre, mitigating the lack of interaction information for new tracks. The idea that the similarity between tracks can be defined through user access patterns, and that this similarity can be estimated from the audio domain, was explored in Shao *et al.*, 2009. A similar idea, based on learning-to-rank, was proposed in B. McFee *et al.*, 2012. When given a query track, the ranking system retrieves other tracks sorted by relevance according to user access patterns, and a corresponding ranking is simultaneously learned using the query audio as input. After training, the ranking system is supposed to retrieve relevant tracks when queried with the audio of a new track, i.e., as a *query-by-example* system.

A novel approach is proposed for a dynamic content-based music recommender in Xing *et al.*, 2014; X. Wang, Yi Wang, *et al.*, 2014. Ratings given by users are modelled as a combination of two factors, an affinity for the audio content, and a factor responsible for diversity. The affinity for audio features is modelled as an inner product of a user preference variable and the audio features of listened tracks. The diversity is implemented with an exponential curve that prevents the recommender to repeat a song that was recently suggested. The probability of a repeated song being suggested again increases with time when the so-called forgetting curve tends to zero. The parameters to be estimated are then the user preference for audio features and the decaying factor of the forgetting curve. The authors present an efficient method for estimating both parameters given the historical data from one specific user, and the system is capable of real-time adjustments in its parameters by simply updating the historical data obtained from users. The system, however, iterates through every track for selecting the one that maximizes the quantile value of the estimated distribution, inspired by Bayesian-UCB (Kaufmann *et al.*, 2012), and this can be time-consuming.

We now describe the methods selected in our experiments, which are applicable to situations of item cold-start in three music recommendation scenarios: collaborative filtering, sequence-aware and stream-based. These methods were selected considering their applicability to the tasks addressed here, and to the conditions imposed by the experiments.

2.5.1 Collaborative Filtering

We assume a set of tracks $s \in S$, a set of users $p \in P$, and a function $f(\cdot)$ for calculating a generic audio feature a in such a way that $f(s_i) = a_i$. Matrix $\mathbf{A}$ contains the features calculated for all tracks. Matrix $\mathbf{R}$ is a rating matrix, and the rating given by user p_i to track s_j is given by R_{ij}.

Content-Boosted Matrix Factorization (CBMF)

Content-Boosted Matrix Factorization (CBMF) was proposed for integrating an audio feature matrix ($\mathbf{A}$) in the matrix factorization process (see Section 2.4.1). CBMF factorizes the rating matrix $\mathbf{R}$ into user and track latent variables, $\mathbf{U}^T$ and $\mathbf{V}$ respectively, and factorizes $\mathbf{V}$ as a linear product between the audio features matrix $\mathbf{A}$ and an auxiliary matrix $\mathbf{\Phi}$ (Forbes and Zhu, 2011).

The factorization process consists of several optimization rounds, and at each round the rating matrix is factorized as $\hat{\mathbf{R}} \approx \mathbf{U}^T\mathbf{V}$, and $\mathbf{V}$ is also factorized as $\mathbf{V} \approx \mathbf{\Phi}\mathbf{A}^T$. The new $\hat{\mathbf{R}}$ can then be decomposed as:

$$\hat{\mathbf{R}} \approx \mathbf{U}^T \cdot \mathbf{\Phi}^T \cdot \mathbf{A}^T \tag{2.47}$$

where $\hat{\mathbf{R}} \in \mathbb{R}^{|U|\times|S|}$, $\mathbf{U}^T \in \mathbb{R}^{|U|\times k}$, $\mathbf{A} \in \mathbb{R}^{|S|\times|a|}$, and $\mathbf{\Phi} \in \mathbb{R}^{|a|\times k}$. k is the size of the dimension of latent variables. The optimization process is performed with the aim of minimizing:

$$\min_{u_i,\Phi} \sum_{i,j} (r_{ij} - u_i^T \Phi^T a_j^T)^2 + \lambda \Big(\sum_{u \in U} ||u||^2 + ||\Phi||^2 \Big), \tag{2.48}$$

where λ is the learning rate. Once the optimization process is finished, a rating R_{ij} is estimated as two consecutive products: first, $\mathbf{\Phi}^T$ is multiplied by one row a_j^T, and the result is multiplied by one row u_i^T. When a new track is added to the system, its rating can be estimated without the need to re-calculating the matrices.

Deep Convolutional Matrix Factorization (DCMF)

A new method was proposed for mapping audio features to latent variables generated by matrix factorization, but instead of relying on a linear model, a Convolutional Neural Network (CNN) is applied (Oord *et al.*, 2013). The proposed method, referred to here as *Deep Convolutional Matrix Factorization* (DCMF), is separated into two steps: first, the rating matrix is factorized with WMF (see Section 2.4.1), and then a CNN is trained for mapping audio features to track latent variables.

Let $g_\theta(\cdot)$ be a function responsible for mapping audio features a_j to latent variables z_j, and whose parameters θ need to be optimized. The authors propose two objective functions:

$$\min_\theta \sum_j ||v_j - \overbrace{g_\theta(a_j)}^{z_j}||^2, \tag{2.49a}$$

$$\min_\theta \sum_{i,j} (r_{ij} - u_i^T \overbrace{g_\theta(a_j)}^{z_j})^2. \tag{2.49b}$$

The objective 2.49a approximates **Z** and **V**, according to a regression error measurement (Figure 2.9 (a)). The objective 2.49b approximates the rating matrix **R** directly, through a pre-computed user latent variable **U**[7] (Figure 2.9 (b)). The authors compare different audio representations applied to different predictors designed for the same task for comparison, and the CNN approach applied to Mel-spectrograms presented the best results.

The function $g_\theta(\cdot)$ is implemented as a CNN, but little information is provided about its architecture. We consider this method in our experiments, but with no guarantee that the proposed CNN is similar to the one used in the original work.

Hierarchical Linear Model with Deep Belief Networks (HLDBN)

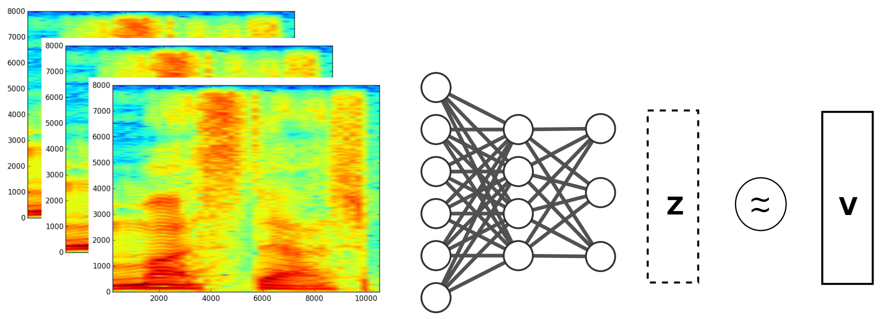
Hierarchical Linear Model with Deep Belief Networks (HLDBN) was proposed for learning tracks and users' latent variables at the same time (X. Wang and Ye Wang, 2014). Differently from previous methods, HLDBN does not require a pre-computation of a matrix factorization, and it was designed to optimize all its parameters at once.

The new user latent variable is denoted by β (Figure 2.9 (c)), and it is assumed as being drawn from a normal distribution $\beta_i \sim \mathcal{N}(\mu, \sigma_i^2 \mathbf{I})$. The rating given by users are also assumed as drawn from a normal distribution $r_{ij}|\beta_i, a_j \sim \mathcal{N}(\beta_i' a_j, \sigma_R^2)$. σ_i and σ_R are the corresponding variances.

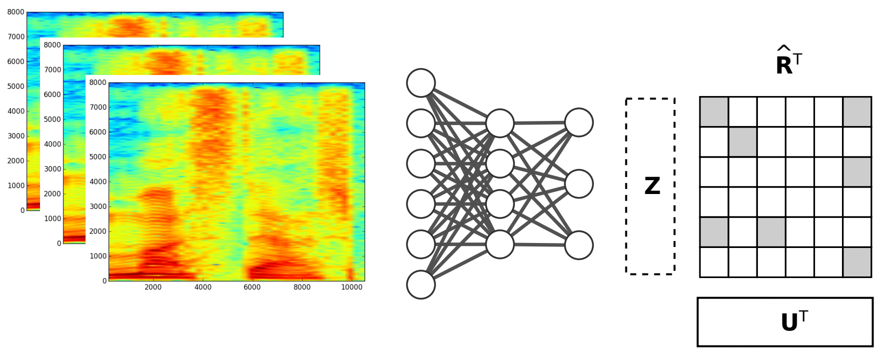
The new error function to be minimized is given by:

$$L_{HLDBN} = \sum_{i,j} (r_{ij} - \beta_i' \overbrace{g_\theta(a_j)}^{z_j})^2 + \lambda \sum_i \|\beta_i - \mu\|^2, \tag{2.50}$$

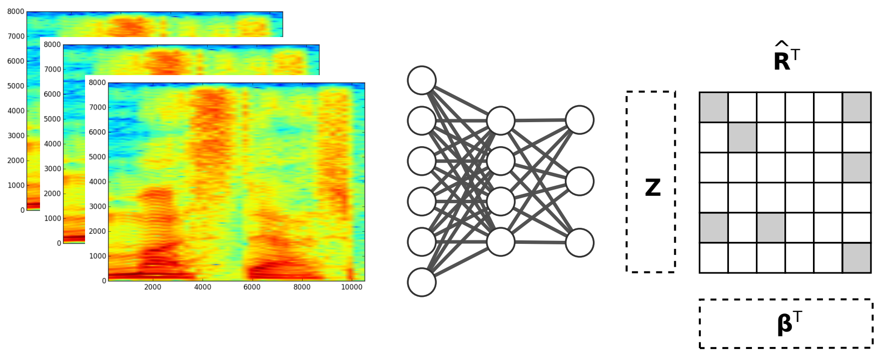
[7] In the original, the authors are considering c_{ij} and p_{ij} instead of r_{ij} in the second objective, like in Equation 2.37, but the latter is presented here for the sake of simplicity



(a) *Deep Convolutional Matrix Factorization (DCMF) objective #1.*



(b) *Deep Convolutional Matrix Factorization (DCMF) objective #2.*



(c) *Hierarchical Linear Model with Deep Belief Networks (HLDBN).*

Figure 2.9: *Previous audio-base methods proposed for mitigating the cold-start in music recommendation. (a) A CNN is trained for learning embeddings* $\mathbf{Z}$ *that approximate the item latent representations* $\mathbf{V}$ *from Figure 2.1. (2) A CNN is trained for learning embeddings* $\mathbf{Z}$ *that multiple the user latent* $\mathbf{U}$ *representations from Figure 2.1 for approximating the rating matrix* $\mathbf{R}$*. (c) A DBN and an auxiliary matrix* β *are trained jointly in such a way that the product between the generating embedding* $\mathbf{Z}$ *and matrix* β *approximate the rating matrix* $\mathbf{R}$*. Solid lines indicate static variables, dashed lines indicate variables that are being trained.*

where μ is the average item latent variable, and the regularization factor ensures that item latent variables do not deviate too much from the average. λ is the learning rate.

The function $g_\theta(\cdot)$ was now implemented as a Deep Belief Network (DBN), and the method was also tested with the Mel-spectrograms representation. The results presented by the authors suggest that user latent variables optimized specifically for this task (β) provide better results than user latent variables derived from matrix factorization ($\mathbf{U}$).

2.5.2 Sequence-Aware

Adaptive Linear Mapping Model (ALMM)

Adaptive Linear Mapping Model (ALMM) (Chou *et al.*, 2016) adapts Factorization Machines (Rendle, 2012), and the content-boost method (see Section 2.5.1) for the task of audio-based next-track recommendation. In short, ALMM decomposes personalized matrices containing transitions between tracks as two consecutive products of three latent variables, in a similar way to FPMC (Rendle, Freudenthaler, *et al.*, 2010). The three matrices are considered latent representations of users, previous tracks, and next tracks. The two last matrices, the ones associated with previous and next tracks, are again factorized as linear products of an audio features matrix and auxiliary matrices, just like in the content-boost method. When a new track is added to the track set, it can be incorporated into the algorithm by calculating the inner products of its audio feature and the learned auxiliary matrices.

Let $\{s_1^u, s_2^u, \dots, s_{t-1}^u\}$ be the tracks listened by user u before time t, ordered by timestamp. Let $\mathbf{L}^u \in \mathbb{N}^{|S|\times|S|}$ be a matrix containing all song transitions associated with user u, in such a way that $L_{i,j}^u$ indicates how many time this user listened to track indexed by j after listening to track indexed by i. And let $\mathbf{L} \in \mathbb{N}^{|S|\times|S|\times|U|}$ be a tensor containing every $\mathbf{L}^u$. The authors consider the task of, given a track s_{t-1}^u listened by user u at time $t-1$, predicting the track s_t^u that will be listened next (i.e. estimating $p(s_t^u|s_{t-1}^u)$).

A new tensor $\mathbf{C}$ is obtained by calculating the *confidence* associated to each value in $\mathbf{L}$, like in Hu *et al.*, 2008. Confidence values are calculated as $C_{i,j}^u = 1 + \log(1 + P_{i,j}^u)$, where $P_{i,j}^u$ is the binary value associated with $L_{i,j}^u$: it is 1 if $L_{i,j}^u > 0$ and 0 otherwise. $\mathbf{C}$ is decomposed by minimizing:

$$\min_{U^*,X^*,Y^*} \sum_{(u,i,j)} (C_{i,j}^u - \mathbf{U}_u^T\mathbf{X}_i - \mathbf{U}_u^T\mathbf{Y}_j - \mathbf{X}_i^T\mathbf{Y}_j)^2 + \lambda_U||\mathbf{U}_u||^2 + \lambda_X||\mathbf{X}_i||^2 + \lambda_Y||\mathbf{Y}_j||^2, \tag{2.51}$$

where $\mathbf{U}_u \in \mathbb{R}^k$ is a user latent vector, $\mathbf{X}_i \in \mathbb{R}^k$ is a previous track latent vector, and $\mathbf{Y}_j \in \mathbb{R}^k$ is a current track latent vector. k is the latent dimension. The regularization factors on the right-hand side are introduced for maintaining the norm of latent variables as close as possible to zero.

In order to integrate the audio feature matrix $\mathbf{A}$ in the decomposition process, matrices $\mathbf{X}$ and $\mathbf{Y}$ are factorized once more in such a way that $\mathbf{X} \approx \mathbf{\Phi}^X \cdot \mathbf{A}^T$, and $\mathbf{Y} \approx \mathbf{\Phi}^Y \cdot \mathbf{A}^T$. The final loss function applied in the minimization process is expressed by:

$$\begin{aligned}\min_{U^*,\Phi^X*,\Phi^{Y*}} \sum_{(u,i,j)} (C_{i,j}^u - \mathbf{U}_u^T\mathbf{\Phi}^X\mathbf{A}_i - \mathbf{U}_u^T\mathbf{\Phi}^Y\mathbf{A}_j - (\mathbf{\Phi}^X\mathbf{A}_i)^T(\mathbf{\Phi}^Y\mathbf{A}_j))^2 \\ + \lambda_U||\mathbf{U}_u||^2 + \lambda_{\Phi^X}||\mathbf{\Phi}^X||^2 + \lambda_{\Phi^Y}||\mathbf{\Phi}^Y||^2.\end{aligned} \tag{2.52}$$

A summary of the optimization process is reproduced in Algorithm 1.

Algorithm 1: Algorithm for Adaptive Linear Mapping Model (ALMM) (reproduced from Chou *et al.*, 2016 with few modifications).

```
Data: confidence matrix $C$, audio features $A$, number of interactions $M$

Initialize $\mathbf{X}$ and $\mathbf{Y}$ ;
repeat
    for $u \in N_u$ do
        Update $\mathbf{U}_u$
    end
    for $i \in N_s$ do
        Update $\mathbf{X}_i$
    end
    $\mathbf{\Phi}^X \longleftarrow \mathbf{X}\mathbf{A}^T(\mathbf{A}\mathbf{A}^T + \lambda_x \mathbf{\Phi}^X)^{-1}$;
    $\mathbf{X} \longleftarrow \mathbf{\Phi}^X \mathbf{A}$ ;
    for $j \in N_s$ do
        Update $\mathbf{Y}_j$
    end
    $\mathbf{\Phi}^Y \longleftarrow \mathbf{Y}\mathbf{A}^T(\mathbf{A}\mathbf{A}^T + \lambda_x \mathbf{\Phi}^Y)^{-1}$;
    $\mathbf{Y} \longleftarrow \mathbf{\Phi}^Y \mathbf{A}$ ;
until Convergence;
return $\mathbf{U}_u$, $\mathbf{\Phi}^X$, $\mathbf{\Phi}^Y$
```

2.5.3 Stream-Based

DJ-MC

DJ-MC (Liebman *et al.*, 2015) is an audio-based framework designed for recommending song sequences to users in a personalized fashion, that is, taking into consideration the preference of users for audio features associated with tracks listened previously. The framework assumes users as having preferences not just for tracks with certain audio characteristics, but also for certain transitions between those audio characteristics. DJ-MC is presented as a reinforcement learning strategy, it allows updates to its model after each user feedback, and it delivers suggestions in the format of track sets, or playlists.

A total of 34 audio descriptors are extracted from the available tracks, from pitch dominance to variance in timbre. The extracted audio descriptors are real-valued time series and are converted to binary vectors according to a 10-percentile quantization process[8]. The final feature vector calculated for track a, denoted as $\theta_s(a)$, has dimensions $\#bins \times \#descriptors = 10 \times 34$, consisting of 34 number 1's located in coordinates corresponding to the bins track a populates for each descriptor.

An audio transition feature is modelled, but not yet calculated, in a similar fashion, but considering a potential transition between two feature vectors. A transition from track a_i to track a_j, denoted as $\theta_t(a_i, a_j)$, is obtained in such a way that one transition matrix ($\#bins \times \#bins$) is calculated for each descriptor. The final feature is represented as a tensor

[8] The reader might want to read the original text for more details (Liebman *et al.*, 2015)

Algorithm 2: Algorithm for updating the DJ-MC recommendation method (reproduced from Liebman *et al.*, 2015 with few modifications).

```
Data: set $A$ of all tracks, rounds of recommendation $K$

for $i \in \{1, \dots, K\}$ do
    recommend track $a_i$ and obtain reward $r_i$ ;
    let $\overline{r} = average(\{r_1, \dots, r_{i-1}\})$ ;
    $r_{incr} = \log(r_i/\overline{r})$ ;
    $w_s = \frac{R_s(a_i)}{R_s(a_i)+R_t(a_{i-1},a_i)}$ ;
    $w_t = \frac{R_t(a_{i-1},a_i)}{R_s(a_i)+R_t(a_{i-1},a_i)}$ ;
    $\phi_s = \frac{i}{i+1} \cdot \phi_s + \frac{i}{i+1} \cdot \theta_s \cdot w_s \cdot r_{incr}$ ;
    $\phi_t = \frac{i}{i+1} \cdot \phi_t + \frac{i}{i+1} \cdot \theta_t \cdot w_t \cdot r_{incr}$ ;
    normalize $\phi_s$ and $\phi_t$ for each descriptor independently ;
end
```

with dimensions $\#bins \times \#bins \times \#descriptors = 10 \times 10 \times 34$, consisting of 34 number 1's indicating the transitions observed for each descriptor.

Users are assumed as having preferences for audio features and preferences for audio feature transitions denoted, respectively, as ϕ_s and ϕ_t. It is also assumed that those preferences can be estimated from recommendation rounds, in which users are presented with tracks, and have the options to like or dislike them. The act of liking or disliking is considered as a reward $R(s, a)$ associated with the action of suggesting one track $a \in A$, after listening to track $s \in S$.

The reward obtained from suggesting track a when the user has finished listening to track s is factorized as a combination of two factors: a preference for audio features, and a preference for audio feature transitions:

$$R(s, a) = R_s(a) + R_t(s, a). \tag{2.53}$$

The affinity of user u for track a is given by $R_s(a) = \phi_s(u) \cdot \theta_s(a)$, and the affinity of the same user for this audio transition is calculated as $R_t(s, a) = \phi_t(u) \cdot \theta_t(s, a)$. The higher the affinity user u has for audio feature $\theta_s(a)$ and for audio transition feature $\theta_t(s, a)$, the higher is the resulting reward $R(s, a)$ obtained from this specific suggestion.

The method supports online updates, performed after each recommendation round, which consists of updating variables ϕ_s and ϕ_t. The details of how these variables are updated are shown in Algorithm 2. And finally, the next track is recommended according to a strategy that selects the most suitable playlist among a set of pseudo-random generated playlists. In other words, given the desired size M of a playlist, N pseudo-random playlists are generated and the one that suits better the user preference is selected among those N.

DJ-MC is considered the main method to be used in the experiments method due to the fact that it is audio-based, it can be adapted to the task of next-track recommendation in an ongoing listening session, and it supports big amounts of tracks.

Chapter 3

Methodology

In this chapter, we present several novel music recommender methods that were designed with the aim of mitigating the cold-start issue, and which use exclusively audio features as input. The methods are presented according to the three recommendation categories considered in this text: Collaborative Filtering, Sequence-Aware or Stream-Based methods.

3.1 Collaborative Filtering

The cold-start problem refers to the inclusion of new items within a recommender system, for which there exists no corresponding listening data. In the specific case of a collaborative music recommender, a new track corresponds to an empty column in the rating matrix, and an auxiliary method is needed in order to integrate this new track in the current recommendation algorithm. We consider that this auxiliary method has access to the audio features associated with all tracks, and is trained with past listening data in order to estimate user/item interactions from audio features. Once trained, this auxiliary method is supposed to predict, from the audio data of a new track, users that will most likely interact with this track.

The methods proposed here assume a *latent factor structure*[1], and they operate according to two strategies: in the *two-step* strategy, a model is trained to learn the embeddings encoded by a pretrained collaborative recommender, while in the *one-step* strategy, a model is trained to directly estimate user/track interaction data in an end-to-end fashion. Both strategies apply audio features to predict listening profiles.

In the sequel, we formalize the cold-start problem and present the details of the proposed methods, which will be subjected to an experimental evaluation in Chapter 4.

[1] By *latent factor structure* we assume models that are built according to an encoder-decoder architecture. The intermediate, or encoded, representation of the input data is referred to as embedding or latent variables.

3.1.1 Problem Definition

Let $u \in U$ be a set of users and let $s \in S$ be a set of tracks. Matrix $\mathbf{R} \in \mathbb{R}^{|S| \times |U|}$ is a rating matrix[2], where r_{ij} is the rating given to track s_i by user u_j. Let R_i be the i-th row of the matrix $\mathbf{R}$, which corresponds to a binary listening profile associated with track s_i. A track listening profile contains 1 at the indexes of users who listened to that track, and 0 otherwise.

Let $Z_k \in \mathbb{R}^M$ be the embedding associated with a track listening profile R_k, and let $f_\theta(\cdot)$ be an encoder function $f : \mathbb{R}^{|U|} \mapsto \mathbb{R}^M$ that maps track listening profiles to their corresponding embeddings $f_\theta(R_k) = Z_k$, using θ as parameters for the encoder/decoder. Let $g_\theta(\cdot)$ be the corresponding decoder function $g : \mathbb{R}^M \mapsto \mathbb{R}^{|U|}$ that maps embeddings back to their corresponding listening profiles $g_\theta(Z_k) = \hat{R}_k \approx R_k$. Please observe that g_θ is usually *not* the inverse of f_θ, but we expect the training process to minimize the encoder/decoder error, so that $g_\theta(f_\theta(R)) \approx R$.

Let $A_k \in \mathbb{R}^P$ be an audio feature associated with track s_k, and let $Z'_k \in \mathbb{R}^M$ be an embedding estimated from A_k. Let $h_\phi : \mathbb{R}^P \mapsto \mathbb{R}^M$ be an estimation function that maps audio features to their estimated embeddings $h_\phi(A_k) = Z'_k$. Our aim is to adjust parameters θ and ϕ in such a way that the approximation $g_\theta(h_\phi(A)) \approx R$ is as accurate as possible. The set of users that will most likely interact with a new track can then be estimated from its corresponding audio feature.

3.1.2 Audio-Based Convolutional Variational Autoencoder Recommender

The *Audio-Based Convolutional Variational Autoencoder Recommender* (ACVAE) is a novel two-step model composed by a VAE (see Section 2.3.2) and a CNN (see Section 2.3.3) network. The VAE is applied in the encoding/decoding process associated with track listening profiles, and the CNN is applied in the process of estimating audio embeddings.

The training process consists of three stages, as illustrated in Figure 3.1. In the first stage, named *VAE Calibration*, a VAE is presented with a known track listening profile and is calibrated in order to reproduce an estimated version of this profile in the output. The calibration process corresponds to a joint calibration of an encoder VAE_Enc and a decoder VAE_Dec. In the second stage, named *Audio Embedding Pretraining*, a convolutional neural network is pretrained for mapping audio features to audio embeddings that match the embeddings estimated in the previous stage by the encoder VAE_Enc. In the third stage, named *Profile Prediction*, the pretrained CNN is attached to the calibrated decoder and is submitted to a new training round, in order to improve the predictions of track listening profiles. The three stages are described in the sequel.

1. *VAE Calibration*: First, a VAE is submitted to a conventional CF training procedure, in which its Encoder and Decoder parameters are adjusted for estimating ratings $\hat{\mathbf{R}} \approx \mathbf{R}$ (the process is illustrated in step number 1 of Figure 3.1). Functions $f_\theta(\cdot)$ and

[2] The rating matrix is presented having tracks as rows and users as columns, differently from how it is usually presented (as in the case of Section 2.4.1). The idea is to simplify the notation for track listening profiles, which is the focus of this section.

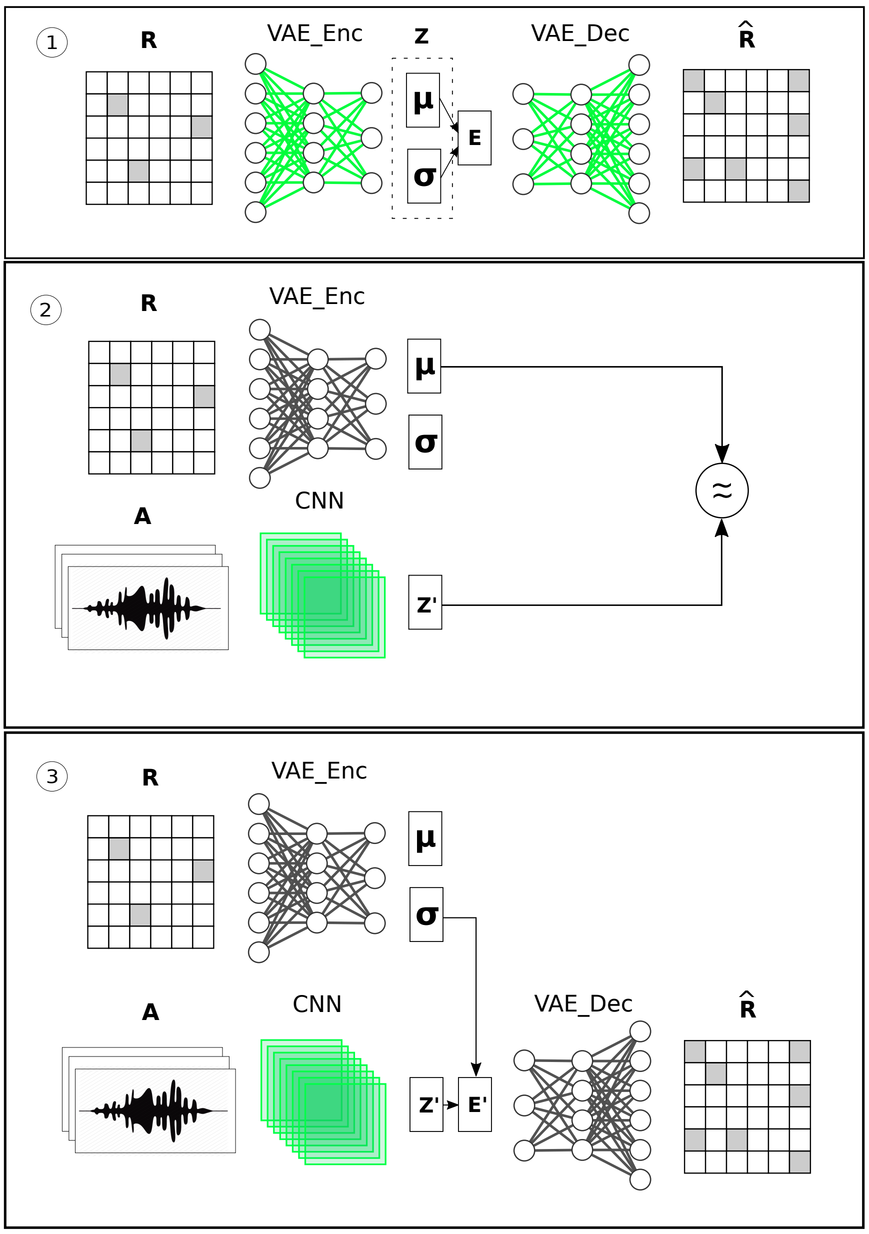


Figure 3.1: *Audio-Based Convolutional Variational Autoencoder Recommender (ACVAE) is a method for mitigating cold-start in music recommendation systems. The green colour indicates networks that are being adjusted in each specific stage, and the black colour indicates networks that are kept fixed. In stage (1) a VAE is submitted to a conventional CF training procedure for adjusting encoder and decoder weights in order to approximate user/track interaction data. The embeddings* $\mathbf{Z}$ *are modeled as Gaussian distributions* $\mathcal{N}(\mu, \sigma)$ *and samples* $\mathbf{E}$ *are obtained from these distributions. In stage (2) a CNN is trained to generate audio embeddings* $\mathbf{Z}'$ *that match the mean values of the embeddings generated in the previous step. In stage (3) the same CNN is fine-tuned according to the trained decoder, in order to approximate the user/track interaction training data used in stage 1. The new samples* $\mathbf{E}'$ *are obtained from the new embeddings modeled as* $\mathcal{N}(\mathbf{Z}', \sigma)$*. Once trained, the CNN can predict user/track interactions given the audio features associated with a new track.*

$g_\theta(\cdot)$ are implemented as MLPs, and the embeddings $\mathbf{Z}$ are assumed to be normal distributions $\mathcal{N}(\mu, \sigma)$. For the sake of clarity, given the input R, the values estimated for μ are expressed as $\mu_\theta(R)$ and the values estimated for σ are expressed as $\sigma_\theta(R)$.

Samples $\mathbf{E} \sim \mathbf{Z}$ are taken from the estimated embedding space and are propagated to the decoder. But a sampling operation is not differentiable, thus not allowing the optimization through backpropagation. In order to make the gradient differentiable with respect to θ (Kingma and Welling, 2014) proposed the reparameterization trick, in which

$$\mathbf{E} = \mu_\theta(\mathbf{R}) + \sigma_\theta^2(\mathbf{R}) \odot \boldsymbol{\epsilon}, \tag{3.1}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ and $\odot$ is an element-wise multiplication. $\mathbf{E}$ is submitted to $g_\theta(\cdot)$ for estimating the original profiles $\mathbf{R}$, i.e. $g_\theta(\mathbf{E}) \approx \hat{\mathbf{R}}$.

The loss function considered for adjusting θ parameters is given by (see Section 2.4.1 and Section 2.3.2 for more information):

$$\mathcal{L}_\theta(R_k) \simeq -\sum_{i=1}^{|S|} R_k^i \log \hat{R}_k^i - \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_\theta^j(R_k))^2) - (\mu_\theta^j(R_k))^2 - (\sigma_\theta^j(R_k))^2), \tag{3.2}$$

where $\hat{R}_k \approx g_\theta(E_k)$ and J is the dimensionality of E_k. R^i and $\hat{R}^i$ are the i-th elements of these vectors, and μ^j and σ^j are the j-th element of these vectors.

2. *Audio Embedding Pre-Training*: Second, $h_\phi(\cdot)$ is implemented as a CNN, and its parameters are adjusted for generating audio embeddings $\mathbf{Z}'$ that match the mean values $\mu_\theta(\mathbf{R})$ of the previously generated embeddings (the process is illustrated in step number 2 of Figure 3.1). The aim of the optimization process is to adjust the parameters ϕ in order to minimize the Mean Square Error (MSE) between both embeddings:

$$\mathcal{L}_\phi(\mu_\theta(R), Z') = \sum_{j=1}^{J}(\mu_\theta^j(R) - Z'^j)^2 \tag{3.3}$$

where $Z' \approx h_\phi(A)$ and J is the dimensionality of Z'. The parameters θ were adjusted in the previous step, and are now maintained fixed.

3. *Profile Prediction*: Finally, parameters ϕ are fine-tuned to minimize a loss function that is similar to the one from step 1, except that now the $\mu_\theta(\mathbf{R})$ term is substituted by the pre-trained $\mathbf{Z}'$. The new loss function is given by:

$$\mathcal{L}_\phi(A_k, R_k) \simeq -\sum_{i=1}^{|S|} R_k^i \log \hat{R}_k^i - \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_\theta^j(R_k))^2) - (Z_k'^j)^2 - (\sigma_\theta^j(R_k))^2), \tag{3.4}$$

where $\hat{R}_k \approx g_\theta(E_k')$, $Z_k' \approx h_\phi(A_k)$, $E_k' = Z_k' + \sigma_\theta^2(R_k) \odot \epsilon$, and J is the dimensionality of E_k'. R^i and $\hat{R}^i$ are, again, the i-th element of these vectors, and μ^j, σ^j and Z'^j are the j-th element of these vectors.

After the three steps are finished, the calibrated Encoder and samples $\mathbf{E}$ are not needed anymore. New track listening profiles can be estimated directly from audio features in such a way that $g_\theta(h_\phi(A)) = \hat{R} \approx R$.

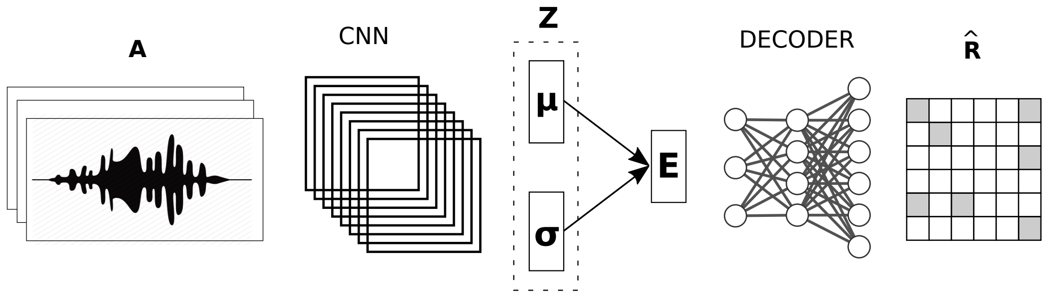


Figure 3.2: *Audio-Based Convolutional Regularized Embedding Recommender (ACRE).*

3.1.3 Audio-Based Convolutional Regularized Embedding Recommender

Audio-Based Convolutional Regularized Embedding Recommender (ACRE) is a one-step model designed for predicting listening profiles associated with tracks, given its corresponding audio features. The embeddings are modelled as Gaussian distributions, which have the effect of regularization and which allow the model to be trained in an end-to-end fashion.

$h_\phi(\cdot)$ is implemented as a CNN, and $g_\theta(\cdot)$ is implemented as an MLP network. The embeddings **Z** are, again, modeled as Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$, but they are now estimated from audio features **A**. This time, given an audio feature A, mean values are expressed as $\mu_\phi(A)$ and standard deviation values are expressed as $\sigma_\phi(A)$.

The variable **E** is obtained by taking samples from **Z**, according to the reparameterization trick (Kingma and Welling, 2014), and is expressed as:

$$\mathbf{E} = \mu_\phi(\mathbf{A}) + \sigma_\phi^2(\mathbf{A}) \odot \boldsymbol{\epsilon}. \tag{3.5}$$

Parameters θ and ϕ are jointly optimized, by minimizing the following loss function:

$$\mathcal{L}_{\theta,\phi}(A_k, R_k) \simeq -\sum_{i=1}^{|S|} R_k^i \log \hat{R}_k^i - \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_\phi^j(A_k))^2) - (\mu_\phi^j(A_k))^2 - (\sigma_\phi^j(A_k))^2), \tag{3.6}$$

where $\hat{R}_k \approx g_\theta(E_k)$, $E_k = \mu_\phi(A_k) + \sigma_\phi^2(A_k) \odot \epsilon$, and J is the dimensionality of E_k. R^i and $\hat{R}^i$ are the i-th element of these vectors, and μ^j, σ^j and Z^j are the j-th element of these vectors.

Once trained, the ACRE model might be applied for estimating track listening profiles given their corresponding audio features, in such a way that $g_\theta(\mu_\phi(A)) = \hat{R} \approx R$.

3.1.4 Evaluation Metrics

In order to evaluate the performance of these methods, a dataset containing information on which users interacted with a set of tracks is needed, as well as audio features associated with these tracks. For each track $s \in S$, there is a corresponding listening profile R, and a corresponding audio feature A. The set of tracks is separated into train/validation/test slices, the first for training the model, the second for monitoring the training and the third for testing the model's overall performance.

The performance is evaluated by feeding the model with audio features associated with the test tracks, and comparing their estimated listening profiles $\hat{R}$ with the original ones R. For each audio feature presented in the input, the model attributes scores to each user $u \in U$ for indicating their willingness in interacting with that specific track. The scores are ordered for positioning the most likely users in the first positions, and the ordered list is truncated at position K. The top-K most likely users are compared with the original profile for measuring the performance.

Let $\{u_1, u_2, \dots, u_K\}$ be the ordered list of users estimated by the model, truncated at K, let A_i be the audio feature associated with track s_i, and let R_i be its corresponding listening profile. The truncated version of Recall and Precision was used for measuring the quality of the results, and they are explained in the sequel.

Precision at K (PREC@K) measures the relative number of correct predictions in the first K ranked suggestions:

$$PREC@K(A_i) = \frac{1}{K} \sum_{k=1}^{K} \mathbb{I}[u_k \in R_i], \tag{3.7}$$

where $\mathbb{I}$ is an indicator function. This can be interpreted as the percentage of users retrieved by the model, up to position K in the ranked list, that actually listened to the track.

Recall at K (REC@K) measures the relative number of correct predictions with respect to the complete listening profile:

$$REC@K(A_i) = \frac{1}{|R_i|} \sum_{k=1}^{K} \mathbb{I}[u_k \in R_i]. \tag{3.8}$$

It can be interpreted as the percentage of the users who listened to the track that was retrieved by the model up to position K in the ordered list.

3.2 Sequence-Aware

SA recommender systems assume a temporal dependency among consecutive tracks listened to by users within listening sessions, and recommendations of next tracks are provided based on the sequence of previously listened tracks. Relying solely on previously listened tracks for providing recommendations alleviates the problem of *user cold-start*: users who have never listened to any track, but who can start receiving recommendations by simply selecting one first track. But the problem of *track cold-start*, referring to the addition of new tracks into an active recommender system, is still an issue, and it can only be addressed by strategies that aggregate additional information about tracks, e.g. their audio content.

We propose new SA music recommendation methods that were designed to suggest the next track to a user based on audio features[3] associated with the tracks listened to previously within a listening session. A first model, named *Audio-Based GRU4REC*, was designed to suggest the next track given the audio feature associated with the current track. This method is audio-based but does not address the track cold-start issue, and it is considered as a baseline in the experimental evaluation. A second model, named *Sequential Audio-Based Autoencoder for Recommendation* was designed to predict the audio feature associated with the next track given the audio feature associated with the track listened to previously. Retrieving a track by its corresponding audio feature can, however, be time-consuming, and a hash scheme is proposed, named *Multi-Level Audio Feature Inverted Index*. The hash scheme allows efficient retrieval of a track given its audio feature by sorting the elements of an audio feature by relevance and using the top-K most relevant elements for searching for their corresponding track in an inverted index structure.

3.2.1 Problem Definition

A listening session of size T is denoted as $\{s^{(1)}, s^{(2)}, \dots, s^{(T)}\}$, where $s^{(t)} \in S$ is the track observed at instant t, with $0 < t \leq T$. A temporal dependency among consecutive tracks is assumed according to the conditional probabilities $p(s^{(t)}|s^{(t-1)}, \dots, s^{(t-m)})$, taking the previous m tracks into consideration. A temporal dependency between audio features is also assumed, corresponding to the probabilities $p(A^{(t)}|A^{(t-1)}, \dots, A^{(t-m)})$, where $A^{(t)}$ is the audio feature associated with track $s^{(t)}$ observed at instant t. And finally, we assume also a dependency between the current track and the previous audio features, expressed as $p(s^{(t)}|A^{(t-1)}, \dots, A^{(t-m)})$.

Our aim is to train a model that is able to predict the upcoming track $s^{(t+1)}$ given the audio feature associated with the current track $A^{(t)}$. In other words, a model that estimates $p(s^{(t+1)}|A^{(t)})$.

3.2.2 Audio-Based GRU4REC

Audio-Based GRU4REC (AGRU4REC) was inspired in GRU4REC (Hidasi *et al.*, 2016), originally proposed as a rating-based model. AGRU4REC suggests a next track within a listening session given an audio feature associated with the current track.

The model consists of three stages described as follows. First, a function $f(\cdot)$ maps an audio feature $A^{(t)}$ to an audio embedding $D^{(t)}$, in such a way that $f(A^{(t)}) = D^{(t)}$. Second, another embedding is calculated by a function $g(\cdot)$ with memory, i.e. a function that is able to store its parameters so they can be used in the next round of recommendation. Let $g(\cdot)$ be the function that maps the audio embedding to the new embedding, named sequence-aware embedding $E^{(t)}$, and let $H^{(t)}$ be the current state of function $g(\cdot)$. At instant t, a sequence-aware embedding is calculated considering the state stored at instant $t-1$, in such a way that $g(D^{(t)}, H^{(t-1)}) = E^{(t)}$. When a listening session ends, the state H is reset, assuming that listening sessions are independent of each other. Finally, a function $q(\cdot)$ maps the session-aware embedding to the scores corresponding to the next track in the session $Y^{(t+1)}$, in such a way that $q(E^{(t)}) = Y^{(t+1)}$. The output $Y^{(t+1)}$ has size $|S|$, and contains

[3] These methods presuppose a choice for a specific audio feature representation for the tracks, which is simply referred to in the sequel as "the audio feature".

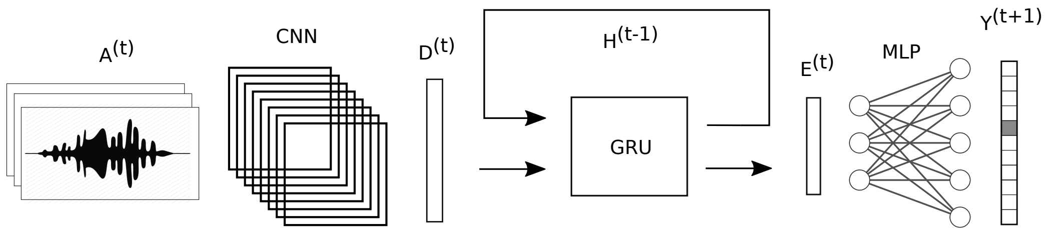


Figure 3.3: *Audio-Based GRU4REC (AGRU4REC), inspired in GRU4REC* Hidasi et al., 2016.

the scores attributed to each track $s \in S$. The highest the score attributed to a track, the higher the probability that this track is the next one in a current listening session.

Function $f(\cdot)$ is implemented with a CNN, function $g(\cdot)$ is implemented with a GRU network (see Section 2.3.4), and function $q(\cdot)$ is implemented with an MLP. The hidden state of the GRU network H is initialized containing zeros, and the training process is summarized in the sequel.

The audio embedding $D^{(t)}$ is first obtained from its corresponding audio feature $A^{(t)}$ (Left-hand side of Figure 3.3) and it propagates to the GRU network. The *reset* ($R^{(t)}$) and *update* ($Z^{(t)}$) gates of the GRU network are the first parameters to be adjusted, respectively, with equations:

$$R^{(t)} = \sigma(\mathbf{W}^{rs}D^{(t)} + \mathbf{W}^{rh}H^{(t-1)} + B_r) \tag{3.9}$$

$$Z^{(t)} = \sigma(\mathbf{W}^{zs}D^{(t)} + \mathbf{W}^{zh}H^{(t-1)} + B_z) \tag{3.10}$$

where $\mathbf{W}^{xy}$ are weight matrices for mapping x to y, to be adjusted during the training, and B_r and B_z are biases. Sigmoid is applied to transform the input values to the range (0,1). When presenting the audio embedding corresponding to the first track of each listening session, $H^{(t-1)}$ is set equal to zero for ensuring independency between sessions, and the second terms of both equations are not considered in the calculation of $R^{(t)}$ and $Z^{(t)}$.

A *candidate hidden state* $N^{(t)}$ is calculated, incorporating the reset gate:

$$N^{(t)} = tanh(\mathbf{W}^{ns}D^{(t)} + \mathbf{W}^{nh}(R^{(t)} \odot H^{(t-1)}) + B_n)) \tag{3.11}$$

where $\odot$ is the Hadamard (elementwise) product and $tanh$ is applied to ensure that the values remain in the interval (-1,1). For now, when entries in the reset gate are set to 1, then the candidate's new state reminds the hidden state calculated for standard RNN (Equation 2.31). When the reset gate is set equal to 0 the architecture reminds of a standard MLP having $D^{(t)}$ in the input.

The final hidden state incorporates the update gate, and is calculated with:

$$H^{(t)} = (1 - Z^{(t)}) \odot N^{(t)} + Z^{(t)} \odot H^{(t-1)}, \tag{3.12}$$

where $H^{(t-1)}$ is the hidden state at time $t-1$. The update gate $Z^{(t)}$ determines to which extent the new hidden state $H^{(t)}$ is inherited from the previous hidden state $H^{(t-1)}$, and how much of the new candidate state is considered.

The session-aware embedding $E^{(t)}$ is a copy of $H^{(t)}$, and $Y^{(t+1)}$ is obtained from $E^{(t)}$, considering that $Y^{(t+1)} = (q(E^{(t)}))$ (Right hand side of Figure 3.3). The scores attributed to

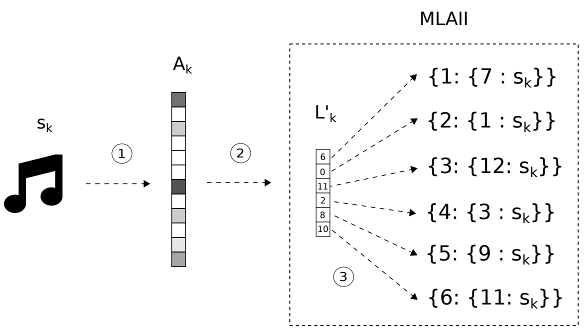


Figure 3.4: *Multi-Level Audio Inverted Index (MLAII) structure for N=6.*

every track are sorted for positioning the most relevant tracks in the first positions.

The model is trained for minimizing the TOP1 loss function (first presented in Section 2.4.2), calculated as:

$$Loss = \frac{1}{|S|}\sum_{j=1}^{|S|} \sigma(\hat{y}_j - \hat{y}_i) + \sigma(\hat{y}_j^2), \tag{3.13}$$

where $\hat{y}_i$ is the score given to the right track $s^{(t+1)}$, and $\hat{y}_j$ is the score given to any other track observed within a mini-batch (negative samples). An extra regularization term forces negative samples to have scores close to zero.

3.2.3 Multi-Level Audio Feature Inverted-Index

Multi-Level Audio Feature Inverted Index (MLAII) is an efficient structure implemented for retrieving tracks given their corresponding audio features. The MLAII structure assumes audio features associated with tracks in the format of numerical vectors, and it is composed of *indexing* and *retrieval* modules. During the indexing phase, the most relevant values of audio features are stored in a *inverted-index* structure, ordered by relevance. In the retrieval phase, the most relevant values of an audio feature are used as a query for obtaining the number of the track with which that audio feature is associated. The size of the query can be reduced for increasing efficiency, and the accuracy of the retrieval task will depend on the selected audio feature.

$A_k \in \mathbb{R}^C$ is an audio feature associated with track s_k, and A_k^l refers to the l-th value of A_k. A function $ArgSort(\cdot)$ is applicable to A, and retrieves a vector L containing the indexes of A ordered by relevance, in such a way that $A^{L^p} > A^{L^q}, \forall p < q$. And lastly, we assume a key/value structure, represented here as $\{key : value\}$, within which values can be indexed by a respective numerical key, and from which the same values can be efficiently retrieved by using their key as a query.

We discuss the indexing and retrieval modules in detail, considering the task of retrieving the right track s_k by submitting its corresponding audio feature A_k.

Indexing

1. In step number 1 of Figure 3.4, an audio feature A_k of size C is obtained from track s_k. There are no restrictions regarding the sparsity or the size of the audio feature, as long as it can be represented as a numerical vector $A_k \in \mathbb{R}^C$.

2. In step 2, function $ArgSort$ is applied to the audio feature for obtaining its indexes ordered by relevance: $L_k = ArgSort(A_k)$. The vector L_k has the same size as A_k, and a truncated version L'_k is obtained by selecting its first N values. The truncated version is represented as $L'_k = \{L^1_k, L^2_k, \dots, L^N_k\}$. In Figure 3.4, N was set equal to 6.

3. In step 3, each value of L'_k is stored in two multi-level inverted-index structures, corresponding to two key/value structures. In a first structure, the value L'^i_k is associated with track s_k, in such a way that $\{L'^i_k : s_k\}$, for $1 < i < N$ (The inner structure in the right hand side of Figure 3.4). In a second structure, the first structure is associated with index i of L'^i_k, in such a way that $\{i : \{L'i_k : s_k\}\}$ (The outer structure in the right hand side of Figure 3.4).

Retrieval

Retrieving songs from the MLAII structure repeats steps 1 and 2 described in the indexing phase, and once the L'_k vector is available, the multi-level inverted index can be consulted for obtaining a set of candidate tracks that is supposed to contain s_k.

Consulting MLAII with L'^1_k will retrieve all tracks that were indexed previously in the structure, whose most relevant audio feature coincides with the most relevant audio feature of L'_k. Consulting for L'^2_k will retrieve all tracks whose second most relevant audio feature coincides with the second most relevant audio feature of L'_k, and this process repeats N times.

The resulting set contains all tracks that coincide at least once with L'_k in the same position n, that is, $L'^n_k = L'^n_j, \forall j \in S \setminus s_k$. This set contains repetitions, meaning that tracks might coincide with L'_k more than once. The number of occurrences of track s_j in the resulting set given A_k can be calculated as:

$$Count(s_j, A_k) = \sum_{n=1}^{N} \mathbb{I}[L'^n_k = L'^n_j], \tag{3.14}$$

where $\mathbb{I}$ is an indicator function, that returns 1 when the condition is true, and 0 otherwise. L'_k corresponds to the top-N most relevant values of A_k, ordered by relevance. Tracks are ordered by the number of occurrences in the resulting set in an output list, for positioning the highest number of coincidences in the first positions of the list.

The MLAII structure retrieves a track given its top-N most relevant audio feature values, making all other values irrelevant for the retrieval task. Moreover, the order in which the top-N most relevant values are presented is the only information that is important, making the values themselves also irrelevant to the task.

3.2.4 Sequential Audio-Based Top-N Autoencoder Recommender

Sequential Audio-Based Top-N Autoencoder Recommender (SATA-REC) is a method designed for predicting the next track within a listening session given the audio feature associated with the current track. Differently from AGRU4REC, SATA-REC is able to incorporate new tracks to its model without retraining, thus mitigating the cold-start problem.

The SATA-REC reproduces the first two stages presented for AGRU4REC, with a key difference in the third stage: instead of estimating the next track from a sequence-aware embedding, the model now estimates a simplified version of the audio feature associated with the next track. This new audio feature is submitted to a MLAII structure, which finally retrieves the most likely candidate for the next track. The audio feature used as input for SATA-REC does not have to be necessarily the same audio feature estimated from the sequence-aware embedding, e.g. the model might be trained with Mel-spectrograms, and be adjusted for estimating codeword histograms as auxiliary audio features. The auxiliary audio feature, however, must be the same one used for indexing the MLAII structure.

The training process of SATA-REC takes into account the fact that the only important information for the MLAII structure is the order in which the top-N most relevant audio feature elements are presented, and it estimates a simplified audio feature that is restricted to N values. The model is optimized according to a loss function expressed as the summation of N cross-entropy values calculated for each of the top-N values. A weight is given to each of these cross-entropy values, in such a away that higher weights are given to the first values, and the weight decreases linearly for emphasizing the higher audio feature indexes despite of the lower ones.

Let $A^{(t)}$ be an audio feature associated with the track observed at instant t, and let $A^{(t+1)}$ be an audio feature associated with the track observed at instant $t+1$, i.e. the next track. Let $L^{(t+1)}$ be a vector containing the indexes of $A^{(t+1)}$ ordered by relevance, calculated in the same way as in Subsection 3.2.3, and let $L'^{(t+1)}$ be its reduced version, truncated in N.

$A'^{(t+1)}$ denotes a simplified version of $A^{(t+1)}$, calculated with:

$$A'^{(t+1)}(i) = \begin{cases} A^{(t+1)}(i), & \text{if } i \in L'^{(t+1)} \\ 0, & \text{if } i \notin L'^{(t+1)} \end{cases}, \quad \text{with} \quad 1 < i \leq |A|, \tag{3.15}$$

where $A'^{(t+1)}(i)$ is the i-th element of $A'^{(t+1)}$.

The two stages applied for estimating the sequence-aware embeddings ($E^{(t)}$) in AGRU4REC are the same ones used here, whose result is expressed in Equation 3.12 (remember that $E^{(t)}$ is a copy of $H^{(t)}$). In the third stage, $E^{(t)}$ is multiplied by a weight matrix $\mathbf{M} \in \mathbb{R}^{|E| \times |A|}$, for estimating a simplified version of the audio feature associated with the next track, $\hat{A}'^{(t+1)}$, in such a way that $\hat{A}'^{(t+1)} = E^{(t)}\mathbf{M}$ (right-hand side of Figure 3.5).

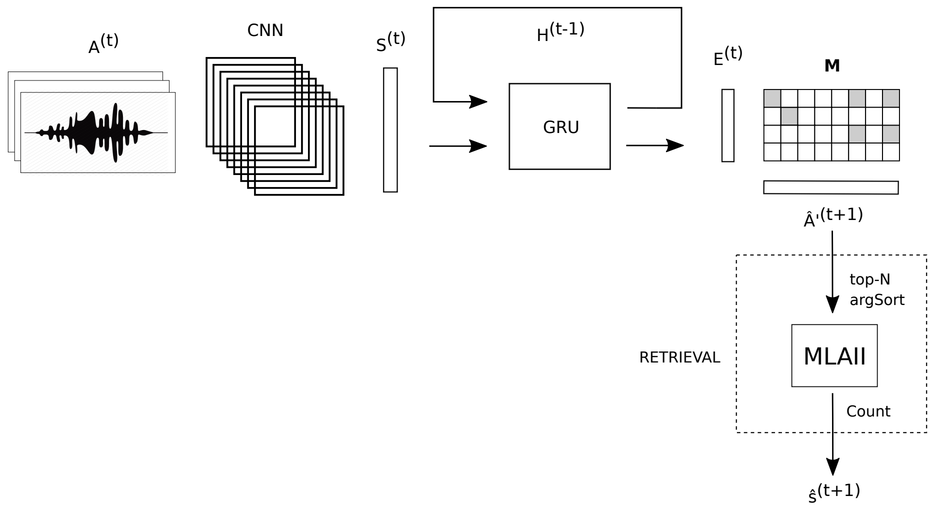


Figure 3.5: *Sequential Audio-Based Top-N Autoencoder for Recommendation (SATAREC).*

The loss function used to optimize the model's parameters is expressed as:

$$\mathcal{L}(A^{(t+1)}, \hat{A}'^{(t+1)}) = -\sum_{n=1}^{N} w(n) \sum_{j=1}^{|A|} \hat{A}'^{(t+1)}(j) \log \mathbb{I}[L'^{(t+1)}(n)](j) \tag{3.16}$$

where $\mathbb{I}[k]$ is an indicator vector of dimension $|A|$ containing 1 in position k, and 0 otherwise. A weight vector is defined as $w(n) = 1/n$ for ensuring that higher emphasis is given to the first N positions of $L'^{(t+1)}(n)$.

Once trained, the model estimates $\hat{A}'^{(t+1)}$ given an audio feature $A^{(t)}$, and the MLAII can be consulted with $\hat{A}'^{(t+1)}$ for retrieving the best candidate for the next track $\hat{s}^{(t+1)}$.

3.2.5 Metrics

In order to evaluate the performance of these methods, a dataset containing listening sessions is needed, as well as audio features associated with the tracks within these sessions. Sessions are sorted by timestamp and separated into train/validation/test slices, the first for training the model, the second for monitoring the training and the third for testing the model's performance. The model is trained with events that happened earlier in time, and is tested with more recent ones, in order to ensure its generalization .

At each prediction round, the probability of each track being the next one in the session (target) is calculated by the model, given the audio feature associated with the current track (query). A list containing the probability associated with all tracks is ordered, for positioning the most likely tracks in the first positions, and it is truncated at position K. The prediction is considered successful in the case where the truncated list contains the next track.

Let $s^{(t)}$ be the current track in a listening session, let $s^{(t+1)}$ be the next track, and let P be a list of tracks returned by the model, ordered by relevance, given audio features $A^{(t)}$. Recall and Mean Reciprocal Rank were selected for measuring the performance of the models, and are explained in the sequel.

Recall at K (REC@K) measures if the target track is among the top-K predicted values:

$$REC@K(A^{(t)}) = \sum_{k=1}^{K} \mathbb{I}[P(k) = s^{(t+1)}], \tag{3.17}$$

where $\mathbb{I}$ is an indicator function that returns 1 if the condition is true, and 0 otherwise, and $P(k)$ stands for the k-th element of P.

Mean Reciprocal Rank (MRR) is the inverse position of the correct prediction:

$$MRR@K(A^{(t)}) = \frac{1}{f(P, s^{(t+1)}, K)}, \tag{3.18}$$

where

$$f(P, s, K) = \min\{k \leq K \text{ such that } s_k \in P\},$$

i.e., $f(P, s, K)$ is the rank of the target track within the first K recommended tracks of list S. If the first K tracks of P do not include the target track, then $f(P, s, K) = \infty$ and its inverse in the expression of $MRR@K$ is 0.

3.3 Stream-Based

Stream-Based recommendation systems assume that users are frequently interacting with tracks and that the data resulting from these interactions are constantly arriving in the system as data streams. The systems designed to operate as SB systems need to take two requirements into consideration: first, they need to incorporate incoming data as fast as possible, in order to deliver up-to-date suggestions, and second, they need to incorporate new tracks that were recently added to the recommendation algorithm, not associated with any previous interaction data, i.e. cold-start.

Previous methods proposed for incorporating incoming data were usually based on dynamic procedures for updating the recommendation models, in such a way that new interactions could be assimilated by the model without retraining. This capability allows the models to not just provide suggestions to users informed by the latest interactions from other users, but also to adapt their suggestions to sudden changes in listening behaviours, known as preference shifts.

Allowing dynamic updates to the model's parameters, however, does not guarantee that recently added tracks will be eventually suggested to users, and end up incorporated into the recommendation algorithm. Suggesting a track that was never heard by any user requires an active attitude from the recommender, of identifying the users that are potentially willing to interact with that new track, and suggesting it to them. One possible option for incorporating new tracks dynamically is to design a recommender that recommends tracks based on their audio content. In case a new track is added to the system, it could be suggested based on its audio, even if it had not been listened to by any user previously.

We present a recommendation model named *Audio-Based Transition Tensor* that allows dynamic updates to its parameters, and that suggests tracks based on their audio content, thus overcoming the cold-start limitation. The model assumes audio features in the format of numerical vectors, that can be sorted according to relevance. The sorted version of the audio features have now their most relevant values in the first positions of the vector, and the original indexes in which these values were located before are also stored. Lists containing these most relevant indexes are truncated at a specific position, and their truncated versions are considered as auxiliary audio features. Transitions between consecutive tracks are mapped to transitions between those auxiliary audio features, and recommendations for the next tracks within listening sessions are calculated based on transitions that are more likely to happen, in a Markov Chain fashion. The tracks corresponding to predicted auxiliary features are finally retrieved from a previously indexed MLAII structure. New transitions can be incorporated dynamically into the model, based on their corresponding auxiliary audio features, and new tracks can be added to the MLAII, independently from the recommendation process.

3.3.1 Problem Definition

The problem is defined similarly as in Sections 3.2.1 and 3.2.3, but with a slightly different notation. Let $A \in \mathbb{R}^C$ be an audio feature, and let A_l refer to the l-th value of A. A function $ArgSort(\cdot)$ is applicable to A, and retrieves a vector $L' = ArgSort(A)$ containing the indexes of A ordered by relevance, in such a way that $A_{L'_p} > A_{L'_q}, \forall p < q$. The vector L' has the same dimension as A, and a truncated version L is obtained by selecting its first N values, named *auxiliary audio feature.* The auxiliary audio feature is represented as $L = \{L'_1, L'_2, \ldots, L'_N\}$.

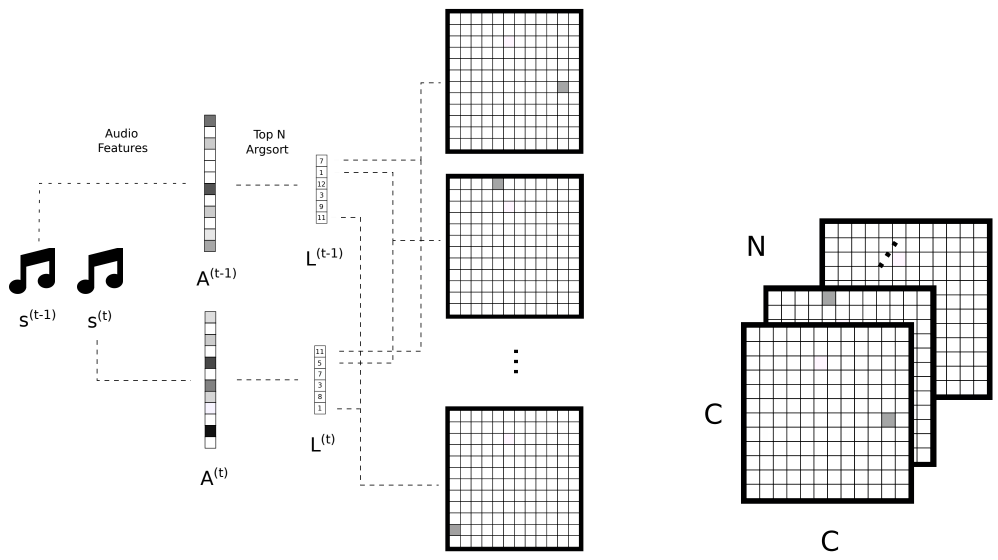
A listening session of size T is denoted as $\{s^{(1)}, s^{(2)}, \ldots, s^{(T)}\}$, where $s^{(t)} \in S$ is the track observed at instant t, with $0 < t \leq T$. A temporal dependency between auxiliary audio features is assumed, expressed as $p(L^{(t)}|L^{(t-1)})$, where $L^{(t)}$ is the auxiliary audio feature associated with track $s^{(t)}$ observed at instant t. And finally, we assume also a temporal dependency between the current track and the previous auxiliary audio features, expressed as $p(s^{(t)}|L^{(t-1)})$.

Lastly, we assume a temporal depency between the values of an auxiliary audio feature, expressed as $p(L_k^{(t)}|L_k^{(t-1)})$ for $0 < k \leq N$. That said, the upcoming auxiliary audio feature can be estimated as $L^{(t+1)} = \{p(L_1^{(t+1)}|L_1^{(t)}), p(L_2^{(t+1)}|L_2^{(t)}), \ldots, p(L_N^{(t+1)}|L_N^{(t)})\}$.

3.3.2 Audio Transition Tensor Recommender

The *Audio Transition Tensor Recommender* (ATTREC) is a novel method designed for predicting an upcoming track within a listening session given the audio feature associated with the current track. The new method summarizes audio features for obtaining a compact and non-redundant audio-based representation of tracks, and transition patterns between the values contained in this new representation are used for estimating the upcoming tracks.

The method is composed of two modules, a first module is applied for mapping transitions between auxiliary audio features in such a way that upcoming auxiliary audio features can be obtained given the current one. A second module is applied for retrieving tracks given its corresponding auxiliary audio feature. The first module is implemented



(a) *Transition between two consecutive tracks* **(b)** *Audio Transition Tensor*

Figure 3.6: *Audio-based Transition Tensor Recommendation model (ATTREC).*

as multiple transitions matrices, corresponding to each position of the auxiliary audio feature, named *Audio Transition Tensor*, and the second module is implemented as an MLAII structure (see Section 3.2.3). The ATTREC method can be updated dynamically, allowing its operation in an SB scenario, and the method is audio-based, thus overcoming the limitation imposed by cold-start.

The process used for training the ATTREC method is illustrated in Figure 3.6 and explained in the sequel. Audio features $A^{(t-1)} \in \mathbb{R}^C$ and $A^{(t)} \in \mathbb{R}^C$, corresponding to each track/track transition observed within a listening session is first calculated, ordered by relevance, and truncated at the N-th position for obtaining their respective auxiliary audio features (C is set equal to 12 and N is set equal to 6 in Figure 3.6). The preceding and the current auxiliary audio features are denoted, respectively, as $L^{(t-1)}$ and $L^{(t)}$. Note that auxiliary audio features are represented as integer index numbers, differently from audio features, which are represented as real numbers.

The N positions of an auxiliary audio feature are assumed independent from each other, and the transitions between them are modelled independently. The transitions related to the most relevant position of an auxiliary audio feature, i.e. $L_1^{(t-1)}/L_1^{(t)}$, are modeled as a Markov Chain (see Section 2.4.2), as illustrated in the right upper corner of Figure 3.6a. In this example, a transition observed from an audio feature whose most relevant value is located at position 7, is mapped to an audio feature whose most relevant value is located in position 11. The process is repeated N times, corresponding to each position of the auxiliary audio feature (and corresponding to each matrix in Figure 3.6a), and it is also repeated for each transition observed in the dataset. Once trained, ATTREC will have mapped N transition matrices, corresponding to the N positions of an auxiliary audio feature (Figure 3.6b), and these matrices can be finally used for estimating the upcoming track within a listening session.

Let T denote the matrix (upper matrix in Figure 3.6a) corresponding to the most relevant audio feature value, i.e. A_{L_1}, and let T_i denote one row of such matrix, with $1 < i \leq C$. The value $T_{i,j}$ is interpreted as the probability of transition between an audio feature whose most relevant value is located in position i to an audio feature whose most relevant value is located in position j. The same interpretation is applied to all N matrices, corresponding to the N-th most relevant positions of audio features.

The transition matrices are considered independent from each other and are applied independently in the process of retrieving tracks from the MLAII structure. In this process, the value of each position of the current auxiliary audio features $L^{(t)}$ is used for retrieving tracks from the MLAII structure, and the track that is retrieved the most is considered the best candidate for the upcoming track. In other words, let $T_{L_1^{(t)}}$ be the row of the transition matrix corresponding to $N = 1$, containing the probability values $p(L_1^{(t+1)}|L_1^{(t)})$ (row number 7 of the upper matrix in Figure 3.6a). The value of $L_1^{(t)}$ is known, and the upcoming $p(L_1^{(t+1)})$ values can be obtained directly from $T_{L_1^{(t)}}$.

In our example, considering matrix T, position 11 presents the highest probability $p(L_1^{(t+1)})$, and tracks corresponding to this position could be obtained from the MLAII, as in Section 3.2.3. But in this example, one single transition was mapped, corresponding to a hypothetical first transition observed in the dataset. After many transitions, $T_{L_1^{(t)}}$ will contain several values corresponding to different transition probabilities, and a parameter α is used for determining the proportion of the values of $T_{L_1^{(t)}}$ that will be submitted to MLAII. The new function for calculating the number of occurrences of track s_j in the resulting set obtained from MLAII, given an auxiliary audio feature L, is expressed as:

$$Count(s_j, L) = \sum_{n=1}^{N} \sum_{p \in P} \mathbb{I}[T^n_{L_n,p} = L^j_n], \tag{3.19}$$

where $P = f(T^n_{L_n}, \alpha)$, considering a function $f(\cdot)$ returning the positions of $T^n_{L_n}$ whose values are higher than $\alpha \times max(T^n_{L_n})$. L^j denotes the auxiliary audio feature associated to track s_j, T_n denotes the n-th transition matrix, and the superscript (t) is removed from $L^{(t)}$ for the sake of notation.

Tracks are ordered based on the number of occurrences returned by $Count(\cdot)$, and the tracks observed the most are considered as the best candidates for the upcoming track in the listening session.

3.3.3 Metrics

A dataset containing listening sessions and audio features associated with the tracks within these listening sessions is applied for measuring the performance of SB recommendation methods. The sessions are sorted by their timestamp and are submitted to the recommendation methods one by one, starting from the most distant in time and moving towards the most recent ones. The tracks observed within each listening session are also sorted by timestamp, and the recommender is asked to predict the upcoming track, given the information about the current one. The first track of each session is assumed as given, and the predictions are evaluated starting from the second position.

The recommendation models are initialized with no information and are updated after each prediction round. The average accuracy is measured for each session, and the results are presented as a temporal evolution of these values. The aim is to observe how much and how fast the models are learning.

Let $L = \{s^{(1)}, s^{(2)}, \dots, s^{(M)}\}$ be the listening session under evaluation. For each audio feature $A^{(t)}$ (query), associated with each track $s^{(t)}$, a list $P^{(t+1)}$ is returned by the recommender containing the probability that each track $s \in S$ is the next track $s^{(t+1)}$ (target). The list $P^{(t+1)}$ is a ordered list, containing the most relevant tracks in the first positions, and a recommendation round is considered successful if track $s^{(t+1)}$ is among the first K positions of $P^{(t+1)}$.

Hit Rate at K (HR@K) is applicable to listening sessions, and it measures the proportion of recommendation rounds in which the target track was among the top-K predict values:

$$HR@K(L) = \frac{1}{M}\sum_{m=1}^{M} \mathbb{I}[s^{(m+1)} \in P_K^{(m+1)}], \tag{3.20}$$

where $\mathbb{I}$ is an indicator function that returns 1 if the condition is true, and 0 otherwise, and $P_K^{(t+1)}$ stands for a reduced version of $P^{(t+1)}$, truncated in the K-th position.

Mean Reciprocal Rank (MRR) is also applicable to listening sessions, and it measures the average of the inverse position of a correct prediction:

$$MRR@K(L) = \frac{1}{M}\sum_{m=1}^{M} \frac{1}{f(P^{(m+1)}, s^{(m+1)}, K)}, \tag{3.21}$$

where

$$f(P, s, K) = \min\{k \leq K \text{ such that } s_k \in P\},$$

i.e., $f(P, s, K)$ is the rank of the target track within the first K recommended tracks of list S. If the first K tracks of P do not include the target track, then $f(P, s, K) = \infty$ and its inverse in the expression of $MRR@K$ is 0.

Chapter 4

Experiments and Results

In this chapter, we describe the procedures applied for acquiring and preparing the data, and the procedures applied for training and testing the recommendation methods.

Regarding the acquisition and preparation of the data, we provide an overview of the dataset selected for the experiments, named LFM-1b (SCHEDL, 2016), which contains approximately 1 billion user/track interaction events. 30-second audio clips associated with some of these tracks were downloaded from the Spotify website, with the help of an API, and were applied to a feature extraction procedure, for obtaining Codeword Histogram, MCFF, Mel-Spectrogram and Raw Waveform representations of each of those audio files.

The experiments are presented separately considering Collaborative Filtering, Sequence-Aware and Stream-Based recommendation methods. Details are provided about the architectures of neural networks, as well as the hyperparameters used in the experiments. Visualizations of the results are provided, and the results measured for cold-start items are presented separately from the overall performance, whenever possible.

4.1 Datasets

The LFM-1b (SCHEDL, 2016) was selected as the dataset to be used in the experiments, considering the fact that it is one of the biggest datasets publicly available containing user/track interaction information. LFM-1b was compiled with data extracted from the LastFM[1] streaming platform, and it is composed of more than 1 billion entries containing values for user, artist, album, track and timestamp. Each entry corresponds to a listening event of a user who listened to a certain track in a specific timestamp, within the period between 2005 and 2014. Each track is associated with an album and with an artist.

The number of listening events observed for each year can vary significantly, as one can see in Figure 4.1. This difference can be problematic for the experiments, considering the possibility that tracks released, for example in 2012, might be associated with a substantially

[1] https://www.last.fm/

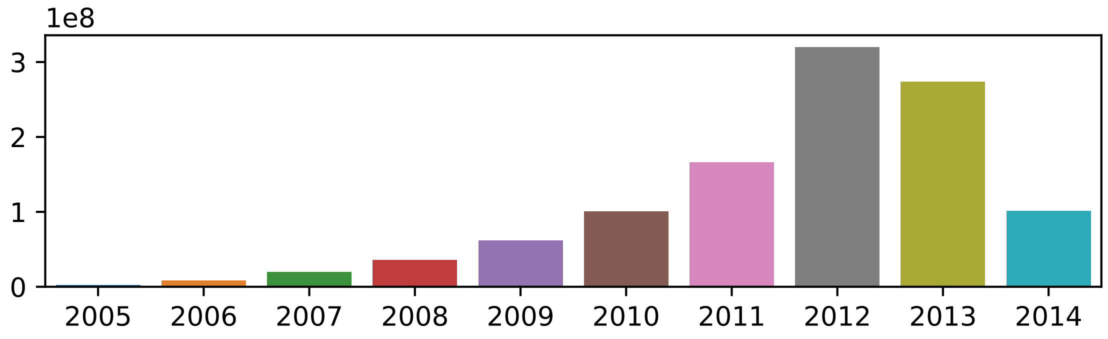


Figure 4.1: *LFM-1b years histogram.*

Dataset	# events	# users	# tracks	#sessions
LFM-1b_2013	273,444,036	84,826	14,530,826	19,023,734
LFM-1b_2011_2013	759,241,825	119,905	25,308,850	53,449,318

Table 4.1: *Datasets description*

higher number of listening events than a track released, for example, in 2007. Models might end up more exposed to certain tracks, despite others, and this can introduce bias in the results (Bellogín *et al.*, 2017). We filtered user/track interactions observed between 2011 to 2013, corresponding to approximately 70% of the whole dataset, and we filtered interactions observed for the year 2013. The former subset is referred to as LFM-1b_2011_2013, and the latter is referred to as LFM-1b_2013 subsets. Information about both subsets can be seen in Table 4.1.

Audio files corresponding to tracks from LFM-1b_2011_2013, that were listened by at least 10 users, were downloaded from the Spotify website with the help of their API[2]. Spotify also provides a Python library named Spotipy[3], with which one can search for track information by using artist and track names as a query. The URL corresponding to a 30-second preview of each song is retrieved, and it can be used to download the audio clip. We were able to download audio previews associated, with 368,221 tracks with 328,189 unique audio files (40,032 repeated ones). All the information extracted from Spotify's website was exclusively used for research purposes.

Mel-spectrograms were calculated for all tracks using Librosa[4] Python library. The sampling rate of audio files was adjusted to 22,050, and the length of the FFT was set equal to 2048. Spectrograms were calculated with window and hop sizes equal to 2048 and to 512 samples, respectively, and the window function selected was the Hanning function. Finally, 128 Mel filters are used, and the final Mel-spectrogram has dimensions 128×1292. The magnitudes of the spectrograms were compressed by a nonlinear curve $\log(1 + C|A|)$ where A is the magnitude and C is set to 10, as suggested in Kim *et al.*, 2018. The audio files were also stored in raw format, with dimension $661,500$, referred to as raw audios.

[2] https://developer.spotify.com/documentation/web-api/

[3] https://github.com/plamere/spotipy

[4] https://librosa.org/doc/latest/index.html

The 20 first Mel Frequency Cepstrum Coefficients (MFCC) were also calculated for all tracks using the Librosa library, with dimensions 20×1292, and were applied as audio features in the process for calculating audio codewords (see Section 2.2.2), described in three steps. In a first step, the first and second derivatives of 100 consecutive frames selected from a random position of each MFCC were calculated. The original 100 frames, and their corresponding first and second derivatives were concatenated vertically, named ΔMFCC, in order to enrich the original audio feature with extra information about temporal variations. The consecutive frames had dimensions 20×100, and the new ΔMFCC had dimensions 60×98, considering the difference between the two frames that were removed for matching the dimensions of the second derivative. ΔMFCCs were calculated for each track in the dataset and were concatenated horizontally, ending up with dimensions $32,984,313 \times 60$.

In a second step, the concatenated ΔMFCCs were clustered in 4,000 clusters (Oord *et al.*, 2013), using the K-means clustering algorithm. In the clustering process, each frame from each ΔMFCC was considered independent from any other one, and the frames were grouped in 4,000 groups according to their proximity. The resulting cluster centres (centroids) were then used in the following step when codewords were calculated for each track.

In the third and last step, a ΔMFCC was calculated for each track, this time considering all frames of the corresponding MFCC, and each of the 1292 frames was compared to all centroids for identifying the closest ones. The idea is to represent each ΔMFCC according to a finite vocabulary, i.e. the cluster centroids, in the format of a histogram of occurrences. In order to reduce the problem of having frames equally close to two or more centroids, we calculated the closest 3 centroids (B. McFee *et al.*, 2012), and the centroids calculated for all ΔMFCC frames are gathered in a histogram. The final histogram codeword has dimension 4000, and it was normalized. The process is repeated for all tracks in the dataset.

4.2 Collaborative Filtering

In this round of experiments, the ACVAE and the ACRE recommendation methods are evaluated in the task of predicting which users would interact with a track given its audio feature, i.e. cold-start. The models are first submitted to a training process, for having their weights adjusted according to a slice of the dataset separated for training, and once trained, the models are evaluated in the task of estimating which users interacted with the tracks from a different slice of the data, separated for testing (and never seen by the model). A slice of the data is also separated for monitoring the training process, named validation subset, and it is applied in the same way as the test slice, but for evaluating the model after each training round.

The experiments were not designed just to reveal the most efficient method, but to reveal also the most appropriate audio feature applied in the task. Three audio features are considered in the experiments: codeword histograms, Mel-spectrograms and raw waveforms. These audio features are mapped to listening profiles, represented as binary vectors, containing 1 in the position of users who interacted with a certain track, and 0 otherwise.

ACVAE is a two-step method, built on top of a previously trained CF recommendation model. The previously trained CF model is implemented according to a latent factor or encoder/decoder structure, that is to say, the input data is first encoded in a compact representation, named embedding, and decoded back to its original dimension. ACVAE needs access to both encoder and decoder separately, considering that the model was designed to reproduce the encoded embeddings given audio features as inputs. The new embeddings, named audio embeddings, propagate to the trained decoder, and listening profiles are finally estimated. ACRE, on the other hand, is a one-step method, which can be trained directly with audio features and listening profiles, in an end-to-end fashion.

In what follows, we describe the process applied for preparing the data used to train the CF recommendation and the audio-based models. We also present the implementations of two latent factor CF models, a first one based on matrix factorization, named WMF, and a second one based on Variational Autoencoder, named VAE-CF. Next, we provide implementation details of the audio-based methods, with special attention to the architectures selected for the CNNs. And finally, the results measured for CF and cold-start tasks are presented for all combinations of methods and audio features.

4.2.1 Data Preparation

When preparing the data for the CF algorithms, we selected users and tracks from the LFM-1b_2011_2013 subset with at least 10 entries, defined as the minimum number of interactions. When considering the tracks associated with audio files, tracks with the same filename were interpreted as repeated entries, and the duplicates were removed as explained in the sequel. First, the number of occurrences of each of these tracks was calculated, and every time a repeated file name was detected, the track number was replaced by the track number with the same file name that was associated with the highest number of occurrences. The idea was to substitute repeated tracks by the one with a higher chance of being the "right" track. Finally, repeated listening events (i.e. the same user and the same track) were removed for obtaining binary listening profiles. The number of interactions was reduced to 203,001,538, and the number of tracks to 2,625,670.

The data was separated into train/valid/test slices, corresponding to, respectively, proportions of 80/10/10% of the dataset. We wanted to ensure that all tracks associated with audio files would have been exposed to the CF recommenders when applying these recommenders to the following audio-based experiments. In order to ensure that, we removed all items with audio from the dataset, and among the remaining ones, we selected a random slice of 10% of the tracks (262,567) for validation and another random slice of 10% of the tracks (262,567) for testing. The remaining 80% of the tracks (2,100,536), including all tracks associated with audio files, were considered as the training slice.

The listening profiles were separated as validation and test data slices were also separated as query and target, corresponding to proportions of 80/20% of the users who listened to each track, respectively. For example, in the case of a track with a listening profile $\{u_1, u_2, \dots, u_{10}\}$, eight random users were selected as a query, and the two remaining ones were considered as the target. The CF algorithms are presented with a query and are asked to predict the whole listening profile, including the target users. The performance of these methods is measured according to the proportion of target users retrieved among the most relevant scores calculated in the output. That said, a rating matrix was built with dimensions $2,625,670 \times 119,824$, corresponding to $\#tracks \times \#users$. Note that the matrix has dimensions that match the total number of tracks and users, even though some data was removed from the dataset for validating and testing the methods. That is because the interaction data removed for validation and testing correspond to entries in listening profiles, thus not affecting the matrix overall dimensions.

When preparing the data for the audio-based algorithms, we separated the tracks associated with audio files in random train/validation/test slices according to 80/10/10% proportions of the number of audio files. 262,553 tracks were separated for training, 32,818 tracks were separated for validating and 32,818 tracks were separated for testing the models. This time, the performance is measured based on how many of the users who actually listened to a track are retrieved by the audio-based model.

4.2.2 Methods

The methods applied to the CF task were the WMF (Hu *et al.*, 2008) and the VAE-CF (Liang *et al.*, 2018). WMF is a matrix factorization method, proposed for decomposing a user/track interaction matrix as a product of two smaller matrices. The smaller matrices are considered embeddings, one associated with tracks and the other associated with users. The track embedding matrix is used in the second step of the ACVAE training procedure when audio embeddings are approximated, and the user embedding matrix is used in the third step when listening profiles are actually estimated.

The VAE-CF method, on the other hand, is composed of an encoder and a decoder, and it only generates track embeddings. These embeddings are also used in the second step of the ACVAE training procedure, and in the third step, the decoder is applied for mapping embeddings to listening profiles. More details about the implementations of WMF and VAE-CF are presented in the sequel.

- **Weighted Matrix Factorization (WMF)**[5] The embedding size was set equals to 200, as suggested in Liang *et al.*, 2018, and the remaining parameters, α, λ, ϵ and the type of confidence (linear or logarithmic) were selected through grid search (All results can be seen in Table A.1). The best results were observed for Linear confidence, $\alpha = 200$ and $\lambda = 1e-5$, all applied to Equation 2.37.
- **Variational Autoencoder Collaborative Filtering (VAE-CF)**[6] The embedding size was also set equals to 200, for ensuring comparability, and the MLP was implemented with one hidden layer, in such a way that the network dimensions were set as $[\#users \longrightarrow 600 \longrightarrow 200 \longrightarrow 600 \longrightarrow \#users]$. The model was trained for 200 epochs, and the learning rate, which started equal to $1e-4$, was reduced by a factor of 0.1 in epochs 100 and 150. A dropout layer was added as the first layer of the encoder, with a dropout rate equal to 0.5, for ensuring generalization, and a Tanh layer was also added between every two internal layers, for maintaining values between -1 and 1.

The methods applied to the CS task were DCMF (Oord *et al.*, 2013) and HLDBN (X.

[5] https://github.com/benanne/wmf

[6] https://github.com/cydonia999/variational-autoencoders-for-collaborative-filtering-pytorch

Wang and Ye Wang, 2014). DCMF was the first method proposing CNNs for mapping audio content to embeddings generated by a WMF matrix factorization. The authors proposed two different objectives expressed as two errors functions: the first compares the new audio embeddings generated by a CNN with the track embeddings generated by WMF, and the second error function compares the product between these new embeddings and the WMF user embeddings with the original listening profiles. The method, considered here as a two-step method, was first proposed using Mel-spectrograms and codeword histograms as audio features (Oord *et al.*, 2013), and it was later adapted for using raw audio (Platt, 2017).

Later on, HLDBN was proposed also for mapping audio features to embeddings, with the help of neural networks, but this time the embeddings were learned jointly with the network weights. The original work uses Deep Belief Networks, but we adapted the model to use CNN for the sake of comparison. According to the authors, user embeddings that are estimated jointly with the network parameters should provide better results than if held fixed like in DCMF.

Different input formats require different CNN architectures, and three architectures are proposed here, one for each audio feature (shown in Figure 4.2). In the case of codeword histograms, five consecutive linear layers are used, and the dimension of the audio features is gradually reduced for matching the embedding size. Each codeword histogram has a size of 4,000, and it is gradually reduced to 200, as shown on the right-hand side of Figure 4.2. Each linear layer is applied together with batch normalization and Tanh operations. The batch normalization improves the stability of the learning process of deep networks (Ioffe and Szegedy, 2015), and the Tanh keeps values within the range $[-1, 1]$. A dropout layer with a dropout rate of 0.5 is added to the output of the network.

The architecture selected for mapping Mel-spectrograms to audio embeddings has five convolutional layers, followed by a linear layer, used for reducing the embedding size. Mel-spectrograms, whose original dimensions were 128×1292, are sliced in 10 slices of 128×115, corresponding to approximately 2.7 seconds of audio, and these slices are submitted to the network separately. All slices calculated from the same Mel-spectrogram are presented to the network as corresponding to the same listening profile, to ensure that the spectral content is presented to the network in its totality.

Each 2-dimensional convolutional layer is applied together with batch normalization, a ReLu and max pooling operations. The ReLu operation keeps values greater than or equal to zero, and the max pooling performs a summarizing operation in the layer input, by extracting the highest value among consecutive 3×3 rectangles. The number of filters applied at each convolutional layer is gradually increased, from 1 to 512, and the size of the image that propagates to the following layer is gradually reduced. The last convolutional layer contains 512 filters of size 1×1, to which a dropout layer with a dropout rate of 0.5 is applied, for improving robustness. An illustration of the architecture is shown in the middle of Figure 4.2.

A similar architecture was selected for mapping raw waveforms to audio embeddings. The architecture proposed in Kim *et al.*, 2018 was adopted and adapted to the task addressed here. The architecture was first proposed for automatically tagging music excerpts with semantic tags, and it was adapted for the task of predicting the users who listened to

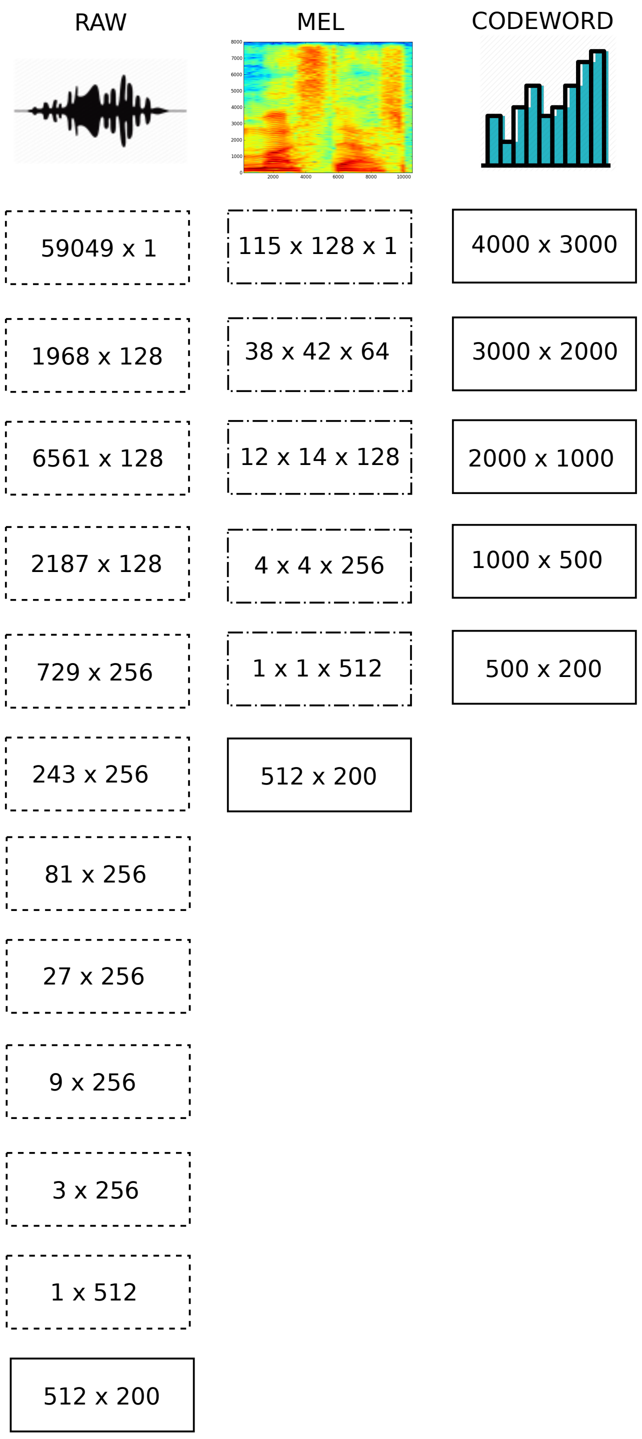


Figure 4.2: *CNN architectures used in the Collaborative Filtering experiments. Solid lines represent fully connected layers, dash-dotted lines represent 2-dimensional convolutional layers, and dotted lines represent one-dimensional convolutional layers. The number inside fully connected layers has the format (number input nodes x number output nodes). The number inside 2-dimensional convolutional layers has the format (feature map size #1 x feature map size #2 x number of feature maps). The number inside 1-dimensional convolutional layers has the format (feature map size x number of feature maps).*

tracks given audio features. The input size of 59,049 samples was kept as proposed by the authors, which also corresponds to approximately 2.7 seconds of the original audio sampled with a 22,050 sampling rate. Once again, the raw audio was separated into 10 non-overlapping slices, and the slices were presented to the network separately for ensuring that the network is exposed to the audio content in its integrity.

The convolutional layers are now applying 1-dimensional convolutional operations, followed again by batch normalization, ReLu and max pooling operations. The motivation behind batch normalization, ReLu and max pooling is the same as in the case of 2-dimensional convolutional layers, even though they are now operating in one-dimensional input data. Max pooling operations are now performed in every 3 sample, and a dropout layer with a dropout layer of 0.5 is applied in the output of the network. The number of filters applied in each convolutional layer can be seen on the left-hand side of Figure 4.2.

The audio-based methods applied to the CS task are now presented considering the CNN architectures mentioned previously.

- **Deep Convolutional Matrix Factorization (DCMF)**. The method is applied on top of tracks and user embeddings obtained from WMF. Track embeddings with dimensions $\#tracks \times 200$ are applied in the first step of the training procedure, and user embeddings with dimensions $\#users \times 200$ are applied in the second step. In the first step, 2-dimensional CNNs are trained for mapping Mel-spectrograms to track embeddings, and the CNN weights are adjusted until the error becomes stable for three consecutive training rounds. When this happens, the learning rate decreases from an initial value of 0.001 to 10% of this value, and the training procedure switches to the second step when the network's weights start being adjusted according to a second objective. In this second step, the generated audio embeddings are not compared to track embeddings anymore but are multiplied by the user embeddings for obtaining the original listening profiles.

  The training process is performed until the learning rate reaches the value of 1e-7, considered a fairly low learning rate. The model is optimized according to Equation 2.37, with the difference that the regularization factor is now applied for minimizing only the track embeddings' norm ($||v||$). The user embedding ($||u||$) is kept fixed, not being a target of the optimization process. The method that applies Mel-spectrograms as inputs is referred to as DCMF (MEL), and the methods that apply raw waveforms and codeword histograms are referred to as DCMF (RAW) and DCMF (CODE), respectively. Results are presented separately for the first and the second objectives.

- **Hierarchical Linear Model with Deep Belief Networks (HLDBN)** The method is submitted to a slightly simpler training procedure than in the case of DCMF, considering that it does not operate on top of any other method and that it can be trained in an end-to-end fashion. This time, the audio features are mapped to audio embeddings, and the audio embeddings are multiplied by an auxiliary matrix with dimensions $\#users \times 200$. This auxiliary matrix is adjusted together with network weights, in one single optimization process.

  The original model applied Deep Belief Networks (DBN) for mapping audio features to audio embeddings, but we adapted the model to use CNNs for the sake of comparison. We tested the regularization factor shown in Equation 2.50, but the results were better when applying the regularization factors from Equation 2.37. We decided to report the latter results. The HLDBN method trained with codeword histograms, Mel-spectrograms and raw waveforms are referred to respectively as HLDBN (CODE), HLDBN (MEL) and HLDBN (RAW).

4.2.3 Experiments

After setting up the other methods, the implementation of ACVAE and ACRE recommender methods are presented in detail. ACVAE is somehow similar to DCMF, taking into account that both methods are built on top of an auxiliary CF method, and ACRE is somehow similar to HLDB, taking into account that both methods are trained in an end-to-end fashion.

- **Audio-Based Convolutional Variational Autoencoder (ACVAE) recommender**[7]. The method is built on top of a previously trained VAE-CF, and it uses VAE-CF's calibrated encoder and decoder in two consecutive steps of the training process. First, audio embeddings generated by a CNN are adjusted for matching the mean values of embeddings generated by the VAE-CF encoder. Listening profiles and audio features are presented simultaneously to the VAE-CF and to the CNN, respectively, and a regression task is performed for approximating the values. Second, the same CNN is fine-tuned for approximating the original listening profiles, with the help of the calibrated decoder, that maps the audio embedding to listening profiles. The calibrated encoder is also used in this second step, for providing the embeddings' variance values, to which the CNN was not exposed yet. The idea is that these variance values will contribute to the training process by helping CNN learn more robust audio embeddings. The process is illustrated in Section 3.1.2. Remember that variance values obtained from the calibrated encoder are used during the training procedure of the ACVAE method but are omitted in the validation and testing procedures. The ACVAE method trained with codeword histograms, Mel-spectrograms and raw waveforms is reported, respectively, as ACVAE (CODE), ACVAE (MEL) and ACVAE (RAW).
- **Audio-Based Convolutional Regularized Embedding (ACRE) recommender**[8]. The audio features are now mapped to audio embeddings, which are modelled as normal distributions. After obtaining the values for mean and variance, samples are taken and submitted to the decoder, which maps the samples to the listening profiles. The variance values are used during training and are omitted during the validation and test procedures. The process is illustrated in Section 3.1.3. The ACRE method trained with codewords histograms, Mel-spectrograms and raw waveforms is reported, respectively, as ACRE (CODE), ACRE (MEL) and ACRE (RAW).

All methods were trained with the data slice separated for training, and the performance

[7] https://github.com/rcaborges/ACVAE

[8] https://github.com/rcaborges/ACRE

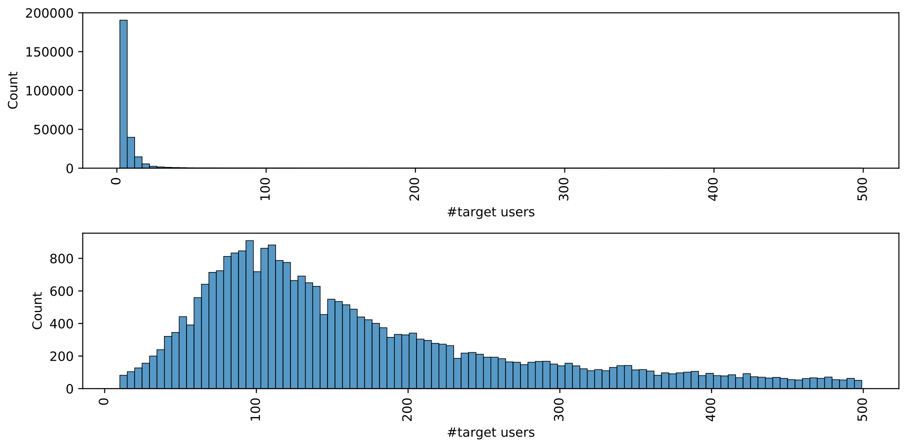


Figure 4.3: *(Top) The number of target users separated for evaluating the CF methods. (Bottom) The number of users associated with each track is separated for evaluating the CS methods. Tracks associated with more than 500 target users were removed for the sake of visualization.*

metrics were measured for the data slice separated for validation, after each training round. The trained methods are applied to the slice of the data separated for testing, and the results are presented in the following section. The methods were also optimized with Stochastic Gradient Descent (SGD), with parameters weight decay (equal to 1e-6) and momentum (equal to 0.9). The learning rates were set equal to 0.001, and this value is multiplied by 0.1 when the loss function is stable for three consecutive epochs.

4.2.4 Results

The results measured for the CF recommendation methods and the results measured for the CS methods are presented separately. The former methods are discussed briefly, whereas the latter methods are discussed in more detail, considered as the main task in these rounds of experiment.

In the CF task, 20% of the users who interacted with the test tracks are randomly removed from their respective listening profiles, and considered target users and the models are asked to predict those missing values. In the CS task, the methods are asked to predict the users who interacted with the test tracks, given their respective audio features. The number of values to be predicted by the methods in both tasks is considerably different, as one can see in Figure 4.3, and this will have an impact on the evaluation metrics, as explained in the sequel.

The results obtained for VAE-CF are substantially better than the results obtained for the WMF method in the CF task, as one can see in Table 4.2. It is worth mentioning that Precision results decrease as the K value increases, while in the case of Recall, the performance of the methods increases as K gets higher, regardless of the K value. This can be explained by formulas 3.8 and 3.7. Both metrics, Precision and Recall, have the same numerator, associated with the number of correct users predicted by the method in the first K positions, the former metric, however, divides this result by an increasing value of K, while the latter divides the result by a fixed number of target users associated with each track.

	PRECISION			RECALL		
	@1	**@10**	**@100**	**@1**	**@10**	**@100**
WMF	0.062	0.040	0.017	0.014	0.080	0.304
VAE	0.221	0.116	0.034	0.046	0.208	0.517

Table 4.2: *Results obtained for VAE-CF and WMF methods in the Collaborative Filtering recommendation task.*

The results obtained for the CS task, i.e. predicting the users who interacted with a track given its audio feature, are shown in Table 4.3. The values measured for Recall are, in general, smaller than the values measured for Precision, and this can be, once more, explained with equations 3.8 and 3.7. Recall values are affected by the number of target users, as discussed before, while Precision values are not.

The ACRE methods applied with Mel-spectrograms and raw waveforms present the best overall results, regardless of the evaluation metric or the audio feature selected in the experiment. When considering the results separated by audio feature, the ACRE has also the best results if compared to DCMF, ACVAE and HLDBN methods. There is no clear distinction between one-step and two-step models, and raw waveforms seem to provide slightly better Recall results, followed by Mel-spectrograms and codeword histograms. Graphs illustrating Recall and Precision values measured for K values within the interval between 1 and 100 are presented in Figure A.1. The results are separated for all combinations of methods and audio features and are also presented separately by audio feature for the sake of visualization.

It is worth mentioning that the models trained with codeword histograms are evaluated with fewer samples than the models trained with Mel-spectrograms and raw waveforms. Even though the number of tracks is the same in both cases, Mel-spectrograms and raw waveforms are sliced into 10 samples each, thus generating a larger number of samples considered in the evaluation process.

The audio embeddings associated with all tracks in the dataset were calculated by using the ACRE(MEL) method and were reduced to a 2-dimensional representation with the t-sne algorithm. Six hand-made clusters were built for illustrating the location of six different music styles within this new representation space, presented in Figure A.2.

METHOD		**REC@1**	**REC@10**	**REC@100**	**PREC@1**	**PREC@10**	**PREC@100**
OBJ1	DCMF (CODE)	0.001	0.004	0.024	0.114	0.083	0.057
	DCMF (MEL)	0.001	0.007	0.042	0.175	0.134	0.092
	DCMF (RAW)	0.000	0.002	0.008	0.058	0.039	0.023
	ACVAE (CODE)	0.001	0.003	0.020	0.111	0.079	0.048
	ACVAE (MEL)	0.001	0.006	0.038	0.171	0.132	0.086
	ACVAE (RAW)	0.001	0.006	0.040	0.173	0.134	0.089
OBJ2	DCMF (CODE)	0.001	0.006	0.040	0.155	0.119	0.082
	DCMF (MEL)	0.001	0.008	0.046	0.204	0.146	0.094
	DCMF (RAW)	0.001	0.008	0.047	0.213	0.149	0.098
	ACVAE (CODE)	0.001	0.005	0.028	0.155	0.106	0.064
	ACVAE (MEL)	0.001	0.008	0.044	0.200	0.148	0.094
	ACVAE (RAW)	0.001	0.009	0.053	0.222	0.170	0.112
HLDBN (CODE)		0.000	0.002	0.013	0.072	0.062	0.035
HLDBN (MEL)		0.001	0.006	0.033	0.188	0.131	0.080
HLDBN (RAW)		0.001	0.007	0.040	0.211	0.151	0.094
ACRE (CODE)		0.001	0.007	0.042	0.209	0.153	0.097
ACRE (MEL)		0.001	**0.010**	**0.057**	**0.261**	**0.194**	0.125
ACRE (RAW)		0.001	**0.010**	**0.057**	0.240	0.188	**0.124**

Table 4.3: *Recommendation results for the task of track profile prediction in the context of collaborative filtering. The best results are highlighted in boldface, and the second best is highlighted with underlined.*

4.3 Sequence-Aware

In this round of experiments, the AGRU4REC and the SATAREC methods are evaluated in the task of predicting the upcoming track within a listening session, given the audio feature associated with the current track. The former method is trained to predict the upcoming track, given the audio feature associated with the current track, but it can not extrapolate to new tracks that have never participated in any listening session. The latter method is trained to predict the audio feature associated with the upcoming track, given the audio feature associated with the current track. An auxiliary inverted index structure is indexed in such a way that a track can be efficiently retrieved given its corresponding audio feature. The SATAREC method, then, can extrapolate its predictions to new tracks, thus mitigating the cold-start limitation.

The listening sessions contained in the dataset are ordered by timestamp and are separated into three slices, one for training, one for validating and one for testing the models. The models are trained with the 80% oldest listening sessions, i.e. with smaller timestamps, and are tested with the most recent 10% of the listening sessions. Another 10% of the sessions, located between the train and test slices, is used for validating the models after each training round.

During the training procedure, each track/track transition and its corresponding audio features are applied for adjusting the models. Once trained, the models are consulted with an audio feature associated with a track from a listening session, and a list containing scores associated with each track from the dataset is returned. The scores are interpreted as a proxy for the probability of each track being the next track within the listening session, and a successful prediction round is the one for which the next track is among the top-K scores calculated by the model. AGRU4REC and SATAREC are compared with rating-based models designed for the same task, and results are reported separately for tracks observed in both train and test data slices, considered as *warm-start* tracks, and for tracks appearing for the first time in the test slice, considered as *cold-start* tracks.

As well as in the CF experiments, the experiments are designed to reveal the most appropriate audio feature for the given task, and three versions of each model are trained having codeword histograms, Mel-spectrograms and raw waveforms as input. In the case of codeword histograms, each track is associated with one histogram, which is used during the training and testing processes. In the case of Mel-spectrograms and raw waveforms, the audio features are separated into 10 non-overlapping slices, and at each round, a random slice is considered as the audio feature. This decision is based on the assumption that each track participates in several track/track transitions, and if in each of these occurrences a random slice of the audio feature is selected, it might be reasonable to say that all slices will end up being exposed to the model. Assuming audio features as a random slice taken from the original Mel-spectrogram or raw waveforms also substantially reduces the training time.

A Multi-Level Audio-Based Inverted Index (MLAII) structure is tested in the task of retrieving the right track given its top-N codeword histogram values. First, the MLAII structure is indexed with codeword histograms associated with all tracks available in the dataset, and once indexed, the structure is consulted with the top-N histogram values associated with a certain track, and a list of tracks that could potentially be the one associated with that codeword histogram is returned. The aim of this test is to reveal: (i) whether the structure is able to return the right track given its top-N codeword histogram values, and if that is the case, (ii) what is the minimum N value necessary for a successful retrieval.

In what follows, we go through the process applied for preparing the data, which includes calculating listening sessions and filtering the sessions associated with audio features. The rating-based and audio-based methods are presented in the sequel, along with their implementation details. Some details about the implementation of AGRU4REC and of SATAREC methods are also presented, as well as the results obtained from all methods in the next-track prediction task. The results obtained from the experiment of retrieving tracks from an MLAII structure given the top-N codeword histogram values are also presented, for several values of N.

4.3.1 Data Preparation

The LFM-1b dataset is composed of approximately one billion entries, each of which corresponds to an event of a user who listened to a track. The dataset provides numerical identifications for the user, for the track, for the artist and the album associated with the track, and a timestamp, but no information is provided about listening sessions. In order to group user/track interactions in listening sessions, we separated the tracks listened by the same user, ordered these events by timestamp, and sessions are assumed as non-interrupted sequences of listening events. More specifically, a session is assumed as starting with the first track of the list, and whenever an interval between adjacent tracks longer than 30 minutes is observed, the current session is finished and the following track is assumed to belong to a new session. This process is repeated for all tracks listened to by all users.

We iterated through the LFM-1b_2013 subset containing around 19,000,000 of these listening sessions, and we selected 889,968 ones containing tracks for which audio files were downloaded. Sessions with less than 5 events, with more than 100 events, and with less than 2 unique tracks were discarded. The remaining sessions were ordered by timestamp and were split into train/valid/test subsets. 711,355 sessions (corresponding to 7,542,193 listening events) were separated for training, 88,919 sessions (corresponding to 945,039 listening events) were separated for validating, and 88,920 sessions (corresponding to 952,473 listening events) were separated for testing the methods. The idea is to simulate a situation when the models are exposed to events that happened in the past, and once trained, they are consulted for predicting events that are about to happen in the near and mid-term future.

4.3.2 Methods

The methods considered in this round of experiments belong to three groups: the rating-based methods, the audio-based methods that were not designed to mitigate cold-start, and the audio-based methods that were designed to mitigate the cold-start issue. The first group comprises conventional recommender methods, that learn from past track/track transitions, and that are able to predict an upcoming track within a listening session. The second group of methods comprises methods that are also trained to predict the upcoming track within a listening session, but that are fed with the audio feature associated with the track being currently listened to. The third group of methods comprises the methods that are, once more, applied to predict the upcoming track within a listening session, but that are actually trained to predict the audio feature associated with the upcoming track. This predicted audio feature is submitted to an audio-based inverted index structure, designed to retrieve a track given its audio feature.

The rating-based methods considered in the experiments are:

- **Markov Chain (MC)**[9]. This is a simple Markov chain method with memory equals to one and with a limited number of candidates associated with each track. A lookup table is built with dimensions $\#tracks \times \#tracks$, within which every track/track transition observed in the training data is stored. The rows correspond to previous tracks, and the columns correspond to the next tracks. The number of potential transitions considered for each track is limited to 100, and when consulted for the most likely transition, the MC method returns the candidate tracks ordered by the number of occurrences.

- **GRU4REC**[10]. Differently from the MC method, this method has an internal state that is updated at each track transition, and that is reset when a listening session ends. The tracks are represented as indicator vectors, before being submitted as input data to the GRU network. The output of the model is a vector containing the probabilities of each track being the upcoming one within a current listening session. The loss function, as well as the optimizer, were maintained the same as in the original article, respectively, TOP1 and Adagrad. The hidden size was set equal to 100, and a Tanh activation layer is added at the output of the network, for restricting values within the interval [-1,1].

- **Neural Attentive Recommendation Machine (NARM)**[11]. As well as GRU4REC, NARM method is based in GRU networks. This method, however, considers all previous tracks as an input for the network in a training/prediction round, for example: a session $\{s_1, s_2, s_3, s_4\}$ is separated in queries $\{s_1\}$, $\{s_1, s_2\}$ and $\{s_1, s_2, s_3\}$, with the corresponding targets $\{s_2\}$, $\{s_3\}$ and $\{s_4\}$. Also, an attention layer is responsible for emphasizing the most relevant track in the session. All parameters were maintained the same as in the original work. The embedding dimension was set equal to 50, the hidden size was set equal to 100, and the learning rate was set equal to 0.001.

The audio-based method is:

- **Audio-Based GRU4REC (AGRU4REC)**[12]. The method derives from GRU4REC, and instead of having tracks represented as indicator vectors, tracks are considered as their corresponding audio features. A CNN is attached before the GRU network, responsible for converting audio features, like Mel-spectrograms or raw waveforms,

[9] https://github.com/rn5l/session-rec/blob/master/algorithms/baselines/markov.py

[10] https://github.com/hungthanhpham94/GRU4REC-pytorch

[11] https://github.com/Wang-Shuo/Neural-Attentive-Session-Based-Recommendation-PyTorch

[12] https://github.com/rcaborges/AGRU4REC

into a one-dimensional audio embedding compatible with the input format of the GRU network. This GRU network outputs scores that are submitted to a Tanh activation layer, for restricting values within the interval [-1,1]. The final scores are interpreted as the probability of each track being the upcoming track, just as in the original method. The CNNs applied here are the same ones shown in Figure 4.2 and described in Section 4.2.2, and parameters such as hidden size, activation function and the optimizer are inherited from the GRU4REC method.

The audio-based cold-start method considered in the experiments is:

- **Adaptive Linear Mapping Model (ALMM)**[13]. The method was designed for suggesting the next tracks to users given an audio feature associated with the current tracks, in a personalized fashion. The method is based on the idea that personalized transition matrices can be factorized in three embedding matrices (one corresponding to users, one corresponding to previous tracks, and one corresponding to upcoming tracks), and that the probability of a transition between two tracks for one specific user can be estimated as a summation of three consecutive products involving the three embedding matrices. Two of these three matrices, the ones associated to previous and next tracks, are factorized once again, as products of an audio feature and an auxiliary mapping matrices. Tracks are then suggested to users based on their audio features, and new tracks can be suggested with the help of the auxiliary mapping matrix. For more details about the method, the reader might want to check Section 2.5.2.

  The original formulation of the method, however, didn't work with the interaction data and with the audio features applied here. We customized the method in such a way that listening events are not associated with users anymore, and instead, one single $track \times track$ transition matrix is used for registering all transitions observed in the training set. This new transition matrix is factorized as the product of two embedding matrices, one corresponding to the previous, and another one corresponding to the next tracks. These embedding matrices are factorized once again, as the product of an audio feature matrix and an auxiliary matrix, as in the original method. The probability of a transition between two tracks can now be calculated as the inner product of their respective estimated embeddings, obtained as a product between their respective audio features and their respective auxiliary matrices.

  The method was implemented with an embedding size equal to 200, and codeword histograms were selected as audio features. The learning rate was set equal to 0.0001, and transition values are considered as their original values, without a confidence weighting function, like the one applied in the original work.

4.3.3 Experiments

The aim of this study is to mitigate the limitation imposed on music recommendation systems when new tracks are introduced to their algorithms. These systems are heavily dependent on user feedback information, and they can only incorporate new tracks into

[13] https://github.com/fearofchou/ALMM

their algorithms if extra information, e.g. audio features, is provided. In this round of experiments, we simulate situations in which music recommenders are trained with a set of track/track transitions, and are tested with a different set of track/track transitions, including tracks that were never exposed to the recommender. The method introduced in the sequel was designed to suggest tracks based on their audio features, in such a way that new tracks can be introduced to the recommender system, with no need for retraining its model.

- **Sequential Audio-Based Top-N Autoencoder Recommender (SATAREC)**[14] - The method was implemented based on AGRU4REC, with the difference that the prediction target is now the codeword histogram associated with the upcoming track, instead of the upcoming track index, as in the original method. In fact, the order in which the top-N values of this histogram is organized is the real prediction target, since this is the information that will be used for retrieving the candidate tracks from the MLAII structure, described as follows.

  An MLAII structure is indexed with codeword histograms calculated for all tracks from the dataset. The indexing process is as simple as selecting the index of the top-N values, and storing these indexes in an inverted index structure in the $\{key : value\}$ format. N independent structures are indexed, corresponding to the top-N positions of the histograms, resulting, thus, in a multi-level inverted index structure. Once indexed, the MLAII structure is expected to retrieve a list of tracks given a codeword histogram. The list is ordered according to the likelihood associated with each track, in such a way that the probability that the track is the right one decreases as the position in the list increases.

  The CNN architectures mentioned previously were adopted again, as well as the hidden size, optimizer, learning rate, weight decay, and momentum. A new loss function was implemented according to the aim of the method, which is to reproduce the right order of the top-N values of the codeword histogram associated with the next track within a listening session. The new loss function is the summation of N cross-entropy values calculated for each of the top-N values. A weight is given to each of these cross-entropy values, in such a way that higher weights are given to the first values, and the weight decreases linearly, emphasizing the higher histogram indexes despite the lower ones.

All methods were trained for 50 rounds, and the results are reported considering the data slice separated for testing the models, in three scenarios: warm-start, cold-start and overall. In the warm-start scenario, the performance of the methods is measured considering the track/track transitions to tracks that participate in the training data. In the cold-start scenario, the performance of the methods is measured considering the transitions to tracks that only participate in the test data, i.e. never exposed to the models. And in the overall scenario, the models are evaluated considering the whole test data.

The results obtained for the audio-based model are reported with references to the audio feature used in the experiments. Methods trained with codewords, Mel-spectrograms and raw waveforms are reported, respectively, as CODE, MEL and RAW.

[14] https://github.com/rcaborges/SATAREC

	REC@1	REC@20	REC@100
N=1	0.014	0.221	0.722
N=2	0.499	0.995	1.000
N=5	0.998	0.999	1.000
N=10	0.999	1.000	1.000

Table 4.4: *Retrieval results obtained for the Multi-Level Audio Inverted Index (MLAII).*

4.3.4 Results

Before discussing the results obtained for the audio-based recommendation methods, we report the results obtained for the MLAII structure in the task of retrieving the right track, given its corresponding codeword histogram. The results obtained for the retrieval task are shown in Table 4.4 for different values of N.

The multi-level inverted index structure can be used for indexing and retrieving any audio feature, as long as it can be represented as a one-dimensional variable[15]. We evaluated the MLAII's performance considering codeword histograms with dimension 4,000. The size of the histograms is relatively big, which produces histograms that are sparse, and which probably contributed to the results obtained in this retrieval task. When considering only the most relevant position of the histogram, N=1, the structure was able to return the right track among the first 100 ones for more than 70% of the tracks. For N greater than 5, the MLAII structure has practically 100% of efficiency in the task of retrieving tracks given their respective codeword histogram.

The results obtained for the rating-based, audio-based, and audio-based cold-start recommendation methods are presented in Table 4.5. One first thing to be noticed in these results is the fact that recommendation results obtained for the rating-based methods are substantially superior to the results obtained for the audio-based methods. Also, audio-based methods perform substantially better than audio-based cold-start methods.

One of the questions proposed in this study addresses the existence of temporal patterns observed in the audio features associated with tracks composing listening sessions. The AGRU4REC method is proposed motivated by this question. The AGRU4REC combines convolutional and recurrent networks, the former calculates an audio embedding from the input audio feature, and the latter is trained to learn the temporal patterns in audio embeddings within listening sessions. The method performed reasonably well, and the results corroborate the hypothesis suggested in the research question.

The SATAREC method was designed for mapping input to output audio features, with the aim of mitigating the cold-start problem in the next-track recommendation task, but its performance was not good. The method was expected to have an inferior overall performance, which happened indeed, but it was also expected to generalize its predictions to new tracks introduced for the first time in the test subset. Its performance in this subset, however, was very poor.

5,459 tracks are only observed in the data subset separated for testing the methods,

[15] By one-dimensional variable we mean a vector $\mathbb{R}^{Nx1}$ with N positions.

considered as the cold-start tracks. 44,638 transitions involving these tracks are evaluated separately, considered as the performance of the methods in a cold-start scenario. The SATAREC performance in the cold-start scenario was lower than expected, possibly because of an inadequate problem formulation. The ALMM performance was also poor, both in the overall and in the cold-start scenarios.

There is no consensus on which audio feature is the best feature for the next-track prediction task. In this study, we adopted codeword histograms, Mel-spectrograms and raw waveforms audio features, and there was no clear superiority of one specific audio feature. One possible reason for the good results measured for the SATAREC (CODE) method, is the fact that CNNs were trained with a random slice of Mel-spectrogram and raw waveforms, while the codeword histogram summarizes the whole audio excerpt, and might provide better resources in the specific task.

Even though the AGRU4REC method has a limited application, not being able to mitigate the cold-start problem, this method can still be an interesting method for other reasons. For example, the audio embedding calculated as an input for the GRU network might be used as an audio feature, containing information about the temporal context from listening sessions.

			REC@1	REC@10	REC@100	MRR@1	MRR@10	MRR@100
Overall	Rating-Based	MC	**0.691**	**0.779**	0.803	**0.691**	**0.721**	**0.722**
		GRU4REC	0.619	0.770	**0.847**	0.619	0.667	0.670
		NARM	0.667	0.761	0.837	0.667	0.695	0.699
	Audio-Based	AGRU4REC (CODE)	**0.294**	**0.471**	0.633	**0.294**	**0.348**	**0.354**
		AGRU4REC (MEL)	0.223	0.436	0.630	0.223	0.288	0.295
		AGRU4REC (RAW)	0.245	0.450	**0.643**	0.245	0.307	0.315
		ALMM*	0.003	0.023	0.104	0.003	0.007	0.010
		SATAREC (CODE)*	**0.078**	**0.130**	**0.230**	**0.078**	**0.094**	**0.097**
		SATAREC (MEL)*	0.062	0.104	0.177	0.062	0.074	0.076
		SATAREC (RAW)*	0.065	0.116	0.199	0.065	0.081	0.083
Warm-Start	Rating-Based	MC	**0.729**	**0.821**	0.847	**0.729**	**0.760**	**0.762**
		GRU4REC	0.652	0.812	**0.893**	0.652	0.703	0.707
		NARM	0.704	0.803	0.883	0.704	0.734	0.737
	Audio-Based	AGRU4REC (CODE)	**0.310**	**0.498**	0.667	**0.310**	**0.368**	**0.374**
		AGRU4REC (MEL)	0.234	0.457	0.664	0.234	0.302	0.310
		AGRU4REC (RAW)	0.260	0.477	**0.680**	0.260	0.326	0.334
		ALMM*	0.003	0.023	0.107	0.003	0.008	0.010
		SATAREC (CODE)*	**0.084**	**0.137**	**0.244**	**0.084**	**0.101**	**0.104**
		SATAREC (MEL)*	0.064	0.109	0.185	0.064	0.077	0.079
		SATAREC (RAW)*	0.069	0.122	0.208	0.069	0.085	0.088
Cold-Start	Audio-Based	ALMM*	0.000	0.000	0.000	0.000	0.000	0.000
		SATAREC (CODE)*	0.000	**0.003**	**0.014**	0.000	0.001	0.001
		SATAREC (MEL)*	0.000	0.002	0.008	0.000	0.001	0.001
		SATAREC (RAW)*	0.000	0.002	0.008	0.000	0.001	0.001

Table 4.5: *Results measured for the next-track prediction task. The methods marked with a star were designed with the aim of mitigating the cold-start problem. Results are reported separately for: all transitions in the test subset (overall), for the transitions to tracks that were already observed in the training subset (warm-start), and for transitions to tracks that appear in the test slice for the first time (cold-start). The best results are highlighted in boldface for each category (rating-based, audio-based and audio-based cold-start recommendation methods), within each scenario (overall, warm-start and cold-start) separately.*

4.4 Stream-Based

The recommender system methods presented so far were designed under the assumption that a static dataset containing user/track or track/track interactions is available for adjusting and evaluating their models. The models are usually exposed to a significantly large number of interactions, for learning the patterns contained in the data, and once trained, the models are tested in a new slice of the data, for evaluating their prediction performance. In a real recommendation situation, however, users are continuously interacting with tracks, and streams of data are being frequently produced. The systems designed under these considerations, the so-called stream-based recommender systems, are capable of incorporating interaction data streams in (almost) real-time, by allowing dynamic updates in their parameters. This way, stream-based methods can perform uninterruptedly, with no need for freezing the recommendation process for retraining their models.

When considering real recommendation situations, it might be also reasonable to assume that new tracks are being added to the recommender system catalogue, with no historical interaction data, i.e. cold-start. In this case, dynamic parameter updating is not enough, and another mechanism is necessary for incorporating these new tracks into the recommendation algorithms. In this section, we evaluate the novel *Audio Transition Tensor for Recommendation* (ATTREC) method, designed for incorporating interaction data streams, and for incorporating new tracks given their respective audio features. The new method maps track/track interactions to an audio domain and recommendations are calculated solely based on audio features associated with tracks. The method is evaluated assuming codeword histogram as audio features, and the MLAII structure mentioned in the previous section is applied for retrieving tracks given their respective audio features.

The algorithm applied for evaluating the stream-based recommendation methods does not consider the dataset separated in train/valid/test slices anymore, instead, it considers all listening sessions ordered by timestamp and each track/track transition as a prediction round. When a session ends, the recommendation models are updated, and the algorithm moves to the following session. Methods are expected to perform poorly at the beginning when few track/track transitions were observed, and they are expected to improve their predictions over time, as far as information about transitions accumulates.

4.4.1 Data Preparation

The same 889,968 listening sessions mentioned in the previous section, referred to as the sessions containing tracks for which audio files were downloaded, are considered here. Sessions with less than 5 events, with more than 100 events, and with less than 2 unique tracks were discarded, reducing the dataset to 889,194 sessions. These sessions are ordered according to the timestamp of their first listening event and are submitted, one by one, to the evaluation algorithm designed for conducting the experiments.

4.4.2 Methods

Four rating-based and one audio-based recommendation method were selected for the stream-based experiments. Three of the rating-based methods calculate recommendations based on the k-nearest-neighbour sessions, named kNN methods[16], and the other method is based on Markov Chains[17]. The audio-based method assumes that users' preferences can be decomposed in two terms: one related to a preference towards audio features associated with tracks, and another related to a preference towards transitions between these audio features.

The rating-based methods are:

- **Dynamic Markov Chain (DMC)**. The DMC method is practically the same as the MC method mentioned in the SA experiments, except that now there is no restriction on the number of candidates associated with each track. The equivalent of a lookup table, with dimensions $\#track \times \#tracks$, is maintained up-to-date with the information of every track/track transition ever observed. When consulted with a track, the method returns the tracks with a higher chance of being the next track within a listening session, and after each session, the lookup table is updated for keeping the method up-to-date.
- **Session-Based kNN (SKNN)**. The method expands the idea of collaborative filtering, which was originally proposed oriented to user listening profiles, to listening sessions. Similar listening sessions are assumed as sharing the same tracks, and the top-k most similar sessions are selected as a resource for calculating suggestions whenever an ongoing listening session is presented as a query. At each recommendation round, the Jaccard distance is applied for selecting the 100 most similar listening sessions, out of the 1,000 most recent ones, and scores are given to each of the tracks from the dataset (see Equation 2.44). Listening sessions are indexed as soon as they are finished, for keeping the model always updated.
- **Vector Multiplication Session-Based kNN (VSKNN)**. The method is a variant of SKNN that emphasizes the more recent events of the listening session presented as a query, by weighting the items according to their positions within the session. A weight equal to 1 is attributed to the most recent track, and the weights decay linearly towards the first track of the session. The idea is that the most recent track is the most relevant one in the recommendation task and that this relevance decays as the relative position of a track gets higher. The same distance is applied for measuring the distance between sessions, the number of the most similar sessions and the number of recent sessions considered in the calculations are the same as in the SKNN case, and scores are calculated with Equation 2.45.
- **Sequence and Time Aware Neighborhood (STAN)**. The method expands the idea of attributing weights to tracks within a listening session by emphasizing the most recent ones, proposed in VSKNN, by including new terms in the formula used for calculating the scores. A first new term ensures that sessions farther from the query session are assigned with lower weight, and another new term ensures that items occurring in both sessions, the query and the candidate sessions, are also weighted according to how recently they were observed. The 100 most similar sessions are again considered as potential candidates for the recommendation, but

[16] https://github.com/rn5l/session-rec/tree/master/algorithms/knn

[17] https://github.com/rn5l/session-rec/blob/master/algorithms/baselines/markov.py

this time, these 100 sessions are selected from a pool containing the 5,000 most recent ones.

The audio-based method is:

- **DJ-MC**. The method is based on the idea that users' preferences are reflected in the audio features associated with tracks, and that these preferences can be used for predicting an upcoming track within a listening session, given the audio feature associated with the current track. The method was originally proposed for generating playlists automatically, and it performed poorly in the task of next-track prediction. This led us to propose three modifications in its original formulation, described as follows. The modifications were also motivated by comparability and efficiency.

  The original method proposed the use of several audio features (e.g. pitch dominance, variance in timbre, among others), but it was here adapted for using codeword histogram audio features for the sake of comparison. In the original work, audio features are considered as real numbers that are quantized into 10-percentile bins, taking the whole dataset into account. Each track ends up being represented as several indicator vectors, each one associated with one of these audio features. Here, the index of the highest value of the codeword histogram is considered equivalent to the bin number, and as a consequence, transitions between tracks, that were originally represented as transitions between those predominant bins, are represented as transitions between the indexes of the highest value of the codeword histograms.

  Two user profiles are kept up-to-date, one expressing users' preferences towards audio features (audio profile), and one expressing users' preferences towards transitions between those audio features (transition profile). When consulted with an audio feature associated with a current track, the method selects among all tracks, which are the ones that fit better with both users' audio profiles. The original method selects candidate tracks from pseudo-randomly generated listening sessions, but this strategy turned out to be too time-consuming for the dataset applied in our experiments. Instead, the method was adapted for using the MLAII structure for retrieving candidate tracks given the users' audio profiles. At each recommendation round, the tracks that share the same top codeword histogram index with the user audio profile are retrieved from MLAII, as well as the tracks that match with the most likely transition observed in the user transition profile. All tracks are gathered and ordered according to the number of occurrences in the tracks returned by the MLAII structure.

  And finally, storing one transition profile for each user turned out to be too expensive as well. Instead, we considered one single transition profile that is shared among all users. This alleviated the memory requirements and improved the recommendation results.

4.4.3 Experiments

We now present some details on the implementation of the ATTREC method, and the algorithm used for evaluating the stream-based recommendation methods. The ATTREC method is somehow related to the transition profiles presented in the DJ-MC method, and to the retrieval stage of the SATAREC method. In the new stream-based method, track/track transitions are associated with transitions between their respective codeword histograms, histograms are summarized as their top-N most relevant indexes, and this information is stored in the memory. In DJ-MC personalized transition profiles were also built with information about transitions between the most relevant bins of audio features involved in the transition. Now, N transition matrices are built for storing the transitions between the N most relevant indexes of audio features, and the transitions between audio contents are shared among users. The new N transition matrices, or a transition tensor, are then used to predict an upcoming track within a listening session by looking up the most likely N indexes stored in the memory, given the top-N indexes associated with the current track. The process for obtaining candidate tracks given potential top-N codeword histogram indexes is the same one applied in SATAREC.

The stream-based recommendation methods were evaluated according to a simulation in which listening sessions are revealed one by one and starting from the second position of each session, models are consulted for the upcoming track (target track), given the current track or the previous tracks from the session (query tracks). A successful prediction round is one in which the target track is among the top-K tracks returned by the methods.

- **Audio Transition Tensor for Recommendation (ATTREC)**[18]. The method assumes that there are temporal patterns in audio features associated with tracks composing listening sessions and that these patterns can be used for suggesting tracks to users. The method also assumes an auxiliary inverted index structure, from which tracks can be retrieved given their respective audio features, and to which new tracks can be added asynchronously.

  More specifically, ATTREC was evaluated with codeword histograms associated with tracks from the LFM-1b_2013 dataset. At each prediction round, the current codeword histogram is summarized as the indexes corresponding to its N most relevant values, and this summarized version is submitted to a prediction stage. In this stage, each of these N values is consulted in the transition tensor for the most likely transitions in that specific index, and the candidate transitions are retrieved from the MLAII structure. For example, if the current histogram's highest value is located in position 10, then the 10-th row of the first transition matrix (corresponding to N=1) is consulted for the most likely transitions. Imagine that in most of the transitions observed previously, the most relevant histogram position moves from index 10 to indexes 5 and 11. The tracks whose most relevant histogram positions are located in 5 and 11 are retrieved from the MLAII structure, and the same procedure is repeated N times. All tracks retrieved from the MLAII structure are ordered by occurrence, and the tracks that repeat the most are considered the best candidates for the next track within the listening session.

  When a session is finished, the codeword histograms associated with all its transitions are used to update the transition tensor. For each transition, the corresponding transition matrix is updated by summing one to the value located in the row that matches the corresponding position of the previous track, and the column that

[18] https://github.com/rcaborges/ATTREC

matches the corresponding position of the next track. The ATTREC method has two parameters whose values will be tested during the experiments, the first one is the size of the summarized version of the audio feature, referred to as N, and the second is the number of potential transitions considered in the prediction stage, referred to as α. The parameter α is defined within the interval $[0, 1]$, as a threshold above which all transitions are considered candidate transitions.

The stream-based methods were evaluated according to an Algorithm 3, which reveals listening sessions, one by one, and updates the recommendation models after each session. Starting from line 5 of the algorithm, listening sessions are selected one by one, from the least to the most recent. A counter is triggered for counting from 1 to the length of the session, for simulating a user that listens to tracks one after the other (line 7). Each iteration of the counter is interpreted as a moment in time when song $j - 1$ of the session is being listened to. All previous tracks within the session are considered as a query (line 8), and the next one j is considered as a prediction target (line 10). The audio feature is obtained from the most recent track in the query (line 9), and all the information, query, target and audio feature, are submitted to a method that retrieves the candidates for the upcoming track (line 11). HitRate@K is measured with Equation 3.20 (line 12), and MRR@K is measured with Equation 3.21 (line 13). When a listening session ends, the model is updated with the information of the current session (line 15).

Algorithm 3: Algorithm for the stream-based recommendation evaluation.

```
   Data: set $S$ of all sessions
 1 int $hr = 0$ ;
 2 int $mrr = 0$ ;
 3 int $k = 10$ ;
 4
 5 for $i = 0 : 50,000$ do
 6     $session = S[i]$;
 7     for $j = 1 : length(session)$ do
 8         $query = session[: j]$ ;
 9         $audio\_query$ = model.Retrieve_Audio($query[-1]$) ;
10         $target = session[j]$ ;
11         $I_k$ = model.Retrieve_TopK_Suggestions($query, audio\_query, k$);
12         $hr += len(I_k \cap target)$ ;
13         $mrr += 1/rank(I_k, target, k)$ ;
14     end
15     model.train($session$);
16 end
```

The average HR@K and average MRR@K measured for the first 50,000 is reported in the results. Rating-based methods are not prepared for suggesting tracks that never took part in any listening sessions, i.e. cold-start. In order to evaluate the performance of audio-based methods in transitions to cold-start tracks, we measured the proportion of right predictions when considering only these tracks.

4.4.4 Results

The ATTREC method has two parameters that should be defined before its application to the final recommendation task, the N and the α parameters. The former defines the number of indexes that will be considered in the simplified audio feature. The latter defines the proportion of potential transitions that will be considered in the prediction task. We tested the N parameter assuming values 10, 100 and 500, and we tested the α parameter assuming values 0.1, 0.5 and 0.99. We evaluated all parameter combinations, and the results are presented in Table 4.6.

		HR@1	**HR@10**	**HR@100**	**MRR@1**	**MRR@10**	**MRR@100**
α=0.1	N=10	0.210	0.240	0.262	0.210	0.219	0.220
	N=100	0.362	0.385	0.400	0.362	0.370	0.370
	N=500	0.362	0.385	0.400	0.362	0.370	0.370
α=0.5	N=10	0.294	0.323	0.356	0.294	0.304	0.305
	N=100	**0.429**	0.461	0.476	**0.429**	0.442	0.442
	N=500	**0.429**	0.461	0.476	**0.429**	0.442	0.442
α=0.99	N=10	0.337	0.562	0.571	0.337	0.421	0.421
	N=100	0.389	**0.566**	**0.572**	0.389	**0.458**	**0.458**
	N=500	0.389	**0.566**	**0.572**	0.389	**0.458**	**0.458**

Table 4.6: *ATTREC recommendation results measured for the first 50,000 listening sessions with different N and α parameters. The best results are highlighted in boldface.*

The new stream-based recommendation method seems to be sensitive to N and to α, and interestingly enough, there are no parameter values that perform better according to all performance metrics. Smaller α values seem to provide better results for HR@1 and MRR@1, while greater α values generate better results in the metrics considering more positions in the recommendation list. In the case of parameter N, simplified audio features of higher dimensions seem to perform better than smaller ones, but only up to a certain upper limit, after which results stabilize. We decided to set N equal to 100, and α equals to 0.99 based on these observations.

All stream-based recommendation methods are submitted to an evaluation algorithm, shown in Algorithm 3, and are evaluated according to the task of predicting the next track within a listening session, given all information associated with the previous tracks of that same session. The results are shown in Table 4.7.

The MC method is more effective at predicting the correct track in the first position of the predictions list, expressed in HR@1 and MRR@1. The method is also more effective in predicting the correct tracks in better positions in longer prediction lists, expressed in MRR@10 and MRR@100. The VSKNN method is the most efficient in retrieving the right next track among the first 100 most likely tracks, expressed in HR@10.

The ATTREC presented results that are surprisingly good, taking into account that it is an exclusively audio-based method that is being compared to rating-based methods. The ATTREC method presented the second best result in five out of six performance metrics.

One possible explanation for these results is the fact that the ATTREC method can be understood as an equivalent version of the MC method, transposed to the audio domain. Both methods rely on the assumption that transitions that were observed more frequently in the past are the most likely transitions to happen in the future. With a substantial difference: one operates with track indexes (MC), and the other with audio feature indexes associated with these tracks (ATTREC).

		HR@1	**HR@10**	**HR@100**	**MRR@1**	**MRR@10**	**MRR@100**
Rating-Based	DMC	**0.517**	0.566	0.567	**0.517**	**0.538**	**0.537**
	SKNN	0.028	0.503	0.674	0.027	0.137	0.147
	VSKNN	0.034	0.557	**0.687**	0.034	0.161	0.168
	STAN	0.004	**0.606**	0.660	0.004	0.216	0.458
Audio-Based	DJ-MC	0.005	0.031	0.078	0.005	0.012	0.013
	ATTREC	0.389	0.566	0.572	0.389	0.458	0.458

Table 4.7: *Recommendation results measured for the first 50,000 listening sessions according to a simulation of a dynamic recommendation environment. ATTREC is evaluated with N=100 and α=0.99. The best results are highlighted in boldface and the second best results are highlighted underlined.*

The experiments considered codeword histograms as audio features, and this might be another reason for the obtained results. Codeword histograms were calculated with a significantly high number of centroids, which generates sparse histograms that are unique to each track. The experiments conducted for evaluating the MLAII structure, shown in Table 4.4, revealed that the two most relevant values of a histogram were enough information for retrieving the right track from the inverted index structure among the first 20 positions (REC@20 with N=2).

Finally, we submitted the two audio-based methods, DJ-MC and ATTREC, to the same cold-start evaluation procedure applied to the SA methods. The methods were trained with a data slice separated for training and were tested in overall, warm-start and cold-start scenarios. The results are presented in Table 4.8.

		HR@1	**HR@10**	**HR@100**	**MRR@1**	**MRR@10**	**MRR@100**
Overall	DJ-MC	0.002	0.010	0.026	0.002	0.004	0.004
	ATTREC	0.443	0.667	0.684	0.443	0.535	0.536
Warm-Start	DJ-MC	0.002	0.011	0.027	0.002	0.004	0.004
	ATTREC	0.431	0.662	0.680	0.431	0.528	0.529
Cold-Start	DJ-MC	0.000	0.000	0.002	0.000	0.000	0.000
	ATTREC	0.622	0.795	0.804	0.622	0.697	0.698

Table 4.8: *Recommendation results are measured for the audio-based methods in the task of next-track prediction. Results are separated into overall, warm-start and cold-start scenarios. ATTREC is evaluated with N=100 and α=0.99.*

The DJ-MC method presented a low performance in the three scenarios, and the ATTREC presented surprisingly good results, in particular, when evaluated in the cold-start scenario.

Chapter 5

Conclusions

In this study, we explored the use of audio features as an alternative for mitigating the cold-start problem, widely known in the recommendation domain. The study was separated into three parts, corresponding to three categories of recommendation methods: collaborative filtering (CF), sequence-aware (SA) and stream-based (SB) methods. The first two categories correspond to two formulations of the recommendation task, while the third one corresponds more closely to the circumstances in which recommendations are being made. One could argue that CF and SA are two ways formulations for the recommendation task and that both formulations can be evaluated in static or SB scenarios. If seen from this perspective, this study reports results for the static evaluation of both CF and SA methods, and for the SB evaluation of SA methods.

In the CF case, recommendation was tackled as a ranking task, i.e. positioning the right users who interacted with tracks in lower positions than the ones who haven't interacted, whereas most studies previously published considered the task as a matrix completion task, i.e. predicting the values that are missing in a rating matrix. In order words, here the evaluation process considered the top-K elements from the prediction list on the evaluation process, while most of the previous studies adopted RMSE or AUC metrics for measuring the methods' performances. The approximation error measured with RMSE expresses the quality of the approximation, which is not directly related to the recommendation quality, and AUC evaluates the task as a binary classification. In our favour, methods designed for ranking the predicted values according to target values can not be evaluated in a binary classification task, while the methods designed for matrix completion can be evaluated according to ranking performance metrics.

The DCMF and the HLDBN methods, adopted as reference methods, were implemented with some particularities. In the case of DCMF, limited information about the model's architecture is provided, and parameters like the number of layers used in the network, the kernel size and the hyperparameters were not mentioned in the original articles, which led us to adapt the CNN architecture according to the architectures proposed here. In the HLDBN method, the user ratings and the user embeddings are modelled as normal distributions, whose parameters are supposed to be learned during the optimization process. We were not able to reproduce these formulations, and instead, both variables were modelled as real numbers. The HLDBN results, then, still have space for improvement.

The AGRU4REC method was proposed as a baseline method, and it performed surprisingly well in the next-track prediction task. The method was proposed motivated by the question if temporal patterns are reflected in audio features associated with tracks, and if these audio features can be used to predict an upcoming track within a listening session. In its best version, the method was able to predict approximately 60% of the upcoming tracks among the first 100 predicted values, given the audio feature associated with the current track. The AGRU4REC method, however, is limited by the cold-start problem, not being able to predict tracks that were never exposed to the model, which led us to propose the SATAREC method. The SATAREC method was designed to mitigate the cold-start limitation, but it presented a limited performance in the next-track prediction task when compared to rating-based and AGRU4REC methods. As future work, we plan to try different architectures and new parameters in order to improve its performance. Nevertheless, the SATAREC method outperformed its competitor ALMM in the standard next-track prediction evaluation.

Incorporating new tracks to SA music recommenders is a challenging task, and incorporating new tracks to SA music recommenders dynamically is even more challenging. An audio-based SB method, named ATTREC, was proposed for recommending tracks dynamically, whose performance in the next-track prediction task is comparable to rating-based methods. The method was evaluated according to an algorithm that simulates an online environment, by revealing listening sessions one by one, according to which the prediction accuracy was measured. The performance of ATTREC under the cold-start scenario was measured according to the SA methodology, which separates a subset of recent listening sessions for evaluation, which is considered more realistic than in the dynamic methodology. In this dynamic methodology tracks would be considered as new tracks only the first time they participate in a listening session, but thie considerably restricts the notion of new tracks.

In the three cases, CF, SA and SB, recommendation methods were proposed, which are able to suggest tracks to users based on the tracks' audio contents. We haven't proposed, however, a methodology for integrating audio-based predictions into rating-based methods in such a way that new tracks can be included in rating-based methods that are already operating. This specific task will be addressed in future work.

When starting this study, no dataset was available containing user/track interaction data and audio excerpts associated with the corresponding tracks. Our decision was to select the biggest dataset publicly available (LFM-1b) and to download the corresponding audio files from the Spotify website using an available API. As far as we know no other study has applied the same dataset and audio files to a similar task, which prevented us from comparing our results with previously reported results.

Finally, scalability, diversity, fairness, and coverage are well-known challenges and established research topics in the area of recommender systems, but they were not mentioned here. Scalability is especially problematic in the case of music recommenders due to the size of the catalogue available for a recommendation, which is usually much bigger than in other domains. Introducing these challenges as extra tasks to be addressed together with cold-start are proposed as future work.

Appendix A

Appendix

A.1 Rating-Based Collaborative Filtering

	PRECISION			RECALL		
	@1	**@10**	**@100**	**@1**	**@10**	**@100**
Lin/100/1e-05	0.062	0.040	0.017	0.014	0.080	0.304
Lin/100/0.001	0.062	0.040	0.017	0.014	0.081	0.305
Lin/100/0.1	0.063	0.040	0.018	0.014	0.080	0.303
Lin/200/1e-05	0.062	0.040	0.017	0.014	0.080	0.304
Lin/200/0.001	0.062	0.040	0.017	0.014	0.081	0.305
Lin/200/0.1	0.063	0.040	0.018	0.014	0.080	0.303
Log/100/1e-05	0.054	0.037	0.017	0.014	0.080	0.312
Log/100/0.001	0.054	0.038	0.017	0.013	0.080	0.313
Log/100/0.1	0.054	0.038	0.017	0.013	0.080	0.312
Log/200/1e-05	0.054	0.037	0.017	0.013	0.080	0.312
Log/200/0.001	0.054	0.038	0.017	0.013	0.080	0.313
Log/200/0.1	0.054	0.038	0.017	0.013	0.080	0.312
None/100/1e-05	0.064	0.040	0.018	0.014	0.079	0.297
None/100/0.001	0.065	0.040	0.017	0.014	0.079	0.298
None/100/0.1	0.065	0.040	0.017	0.014	0.079	0.298
None/200/1e-05	0.064	0.040	0.018	0.014	0.079	0.297
None/200/0.001	0.065	0.040	0.017	0.014	0.079	0.298
None/200/0.1	0.065	0.040	0.017	0.014	0.079	0.298

Table A.1: *Recommendation results obtained for the WMF recommendation method. The results are presented in the following format: confidence type/α/λ.*

A.2 Cold-Start Collaborative Filtering Results

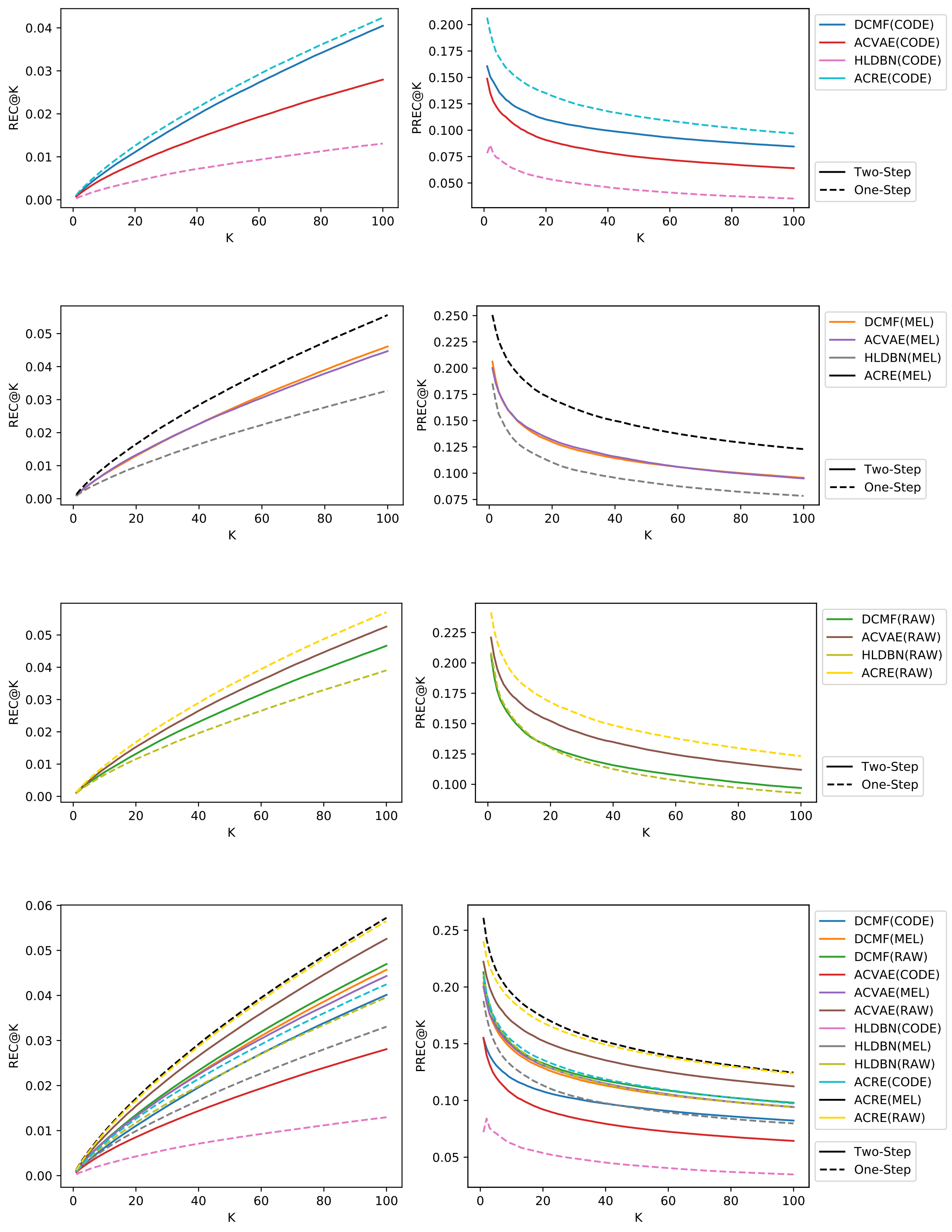
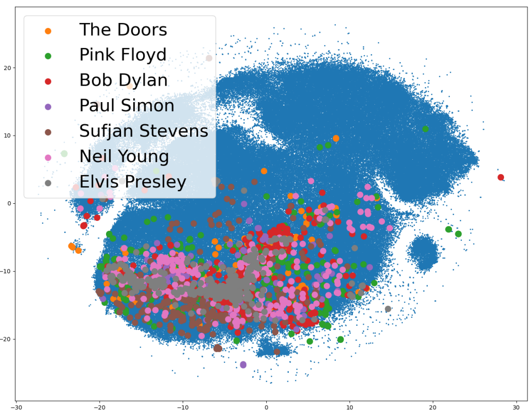
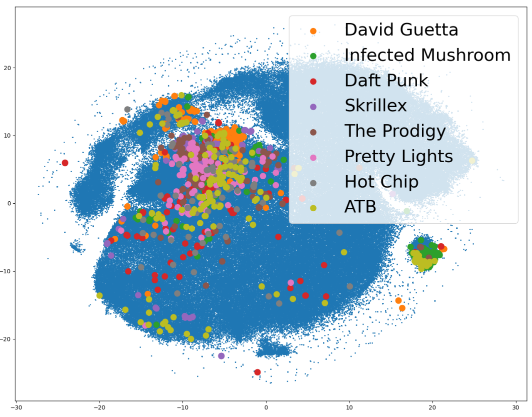


Figure A.1: *Results measured for the track profile prediction task, in the context of collaborative filtering. Results are presented separately for each audio feature. (First) Results measured for code-word histograms. (Second) Results measured for Mel-spectrogram. (Third) Results measured for raw waveform. (Fourth) Results measured for all audio features and methods.*

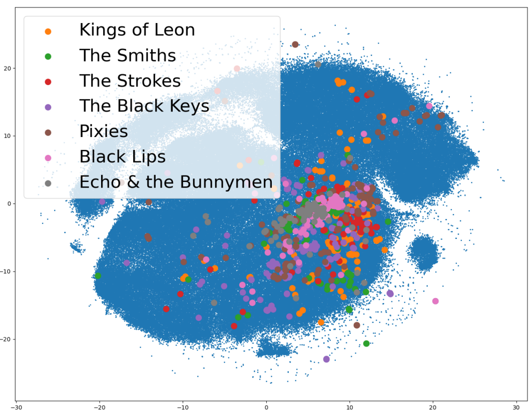
A.3 Audio-Based Artist Clusters



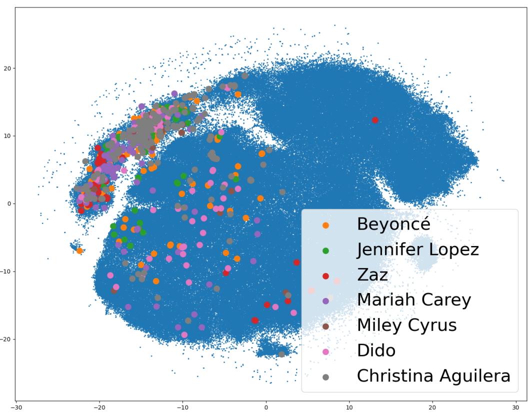
(a) first



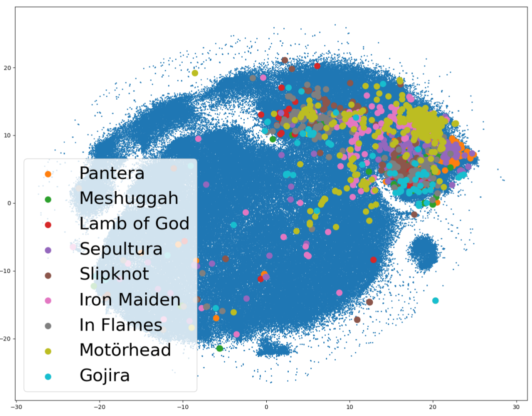
(b) second



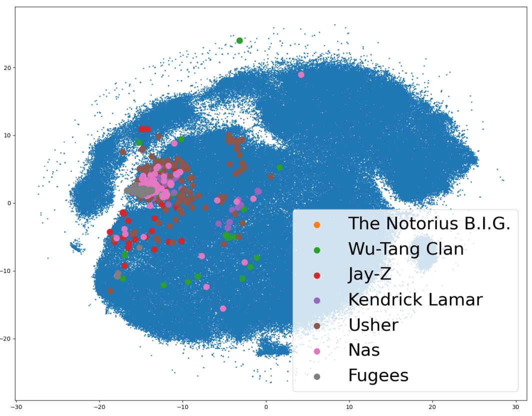
(c) third



(d) fourth



(e) fifth



(f) sixth

Figure A.2: *Six hand-made clusters obtained from audio-based representations of songs from the LFM-1b dataset.*

A.4 Sequence-Aware Recommendation Results

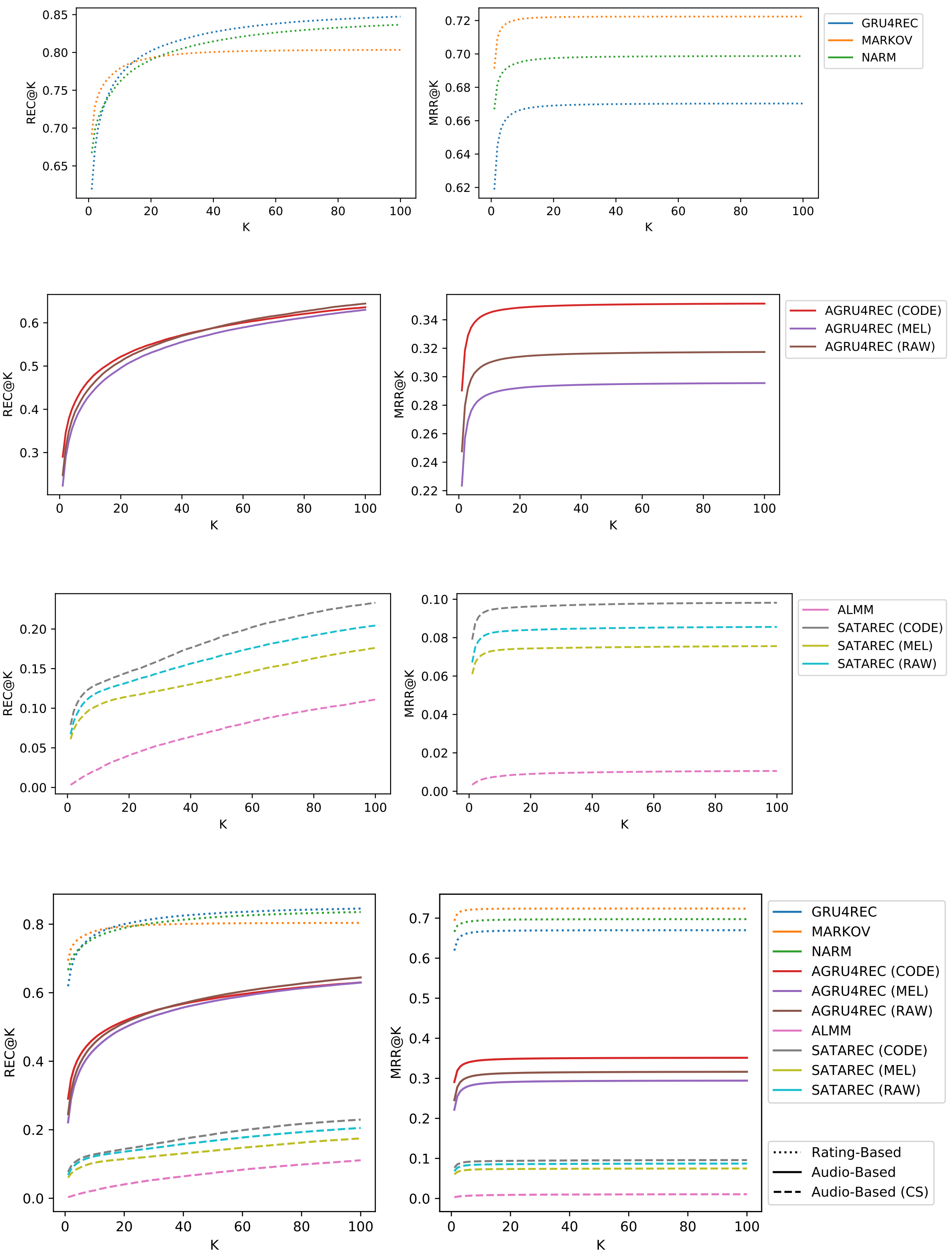


Figure A.3: *Results measured for the next-track prediction task. (First) Results measured for rating-based methods. (Second) Results measured for audio-based methods. (Third) Results measured for audio-based methods designed for mitigating the cold-start problem. (Fourth) Results measured for all methods.*

A.5 Stream-Based Recommendation Results

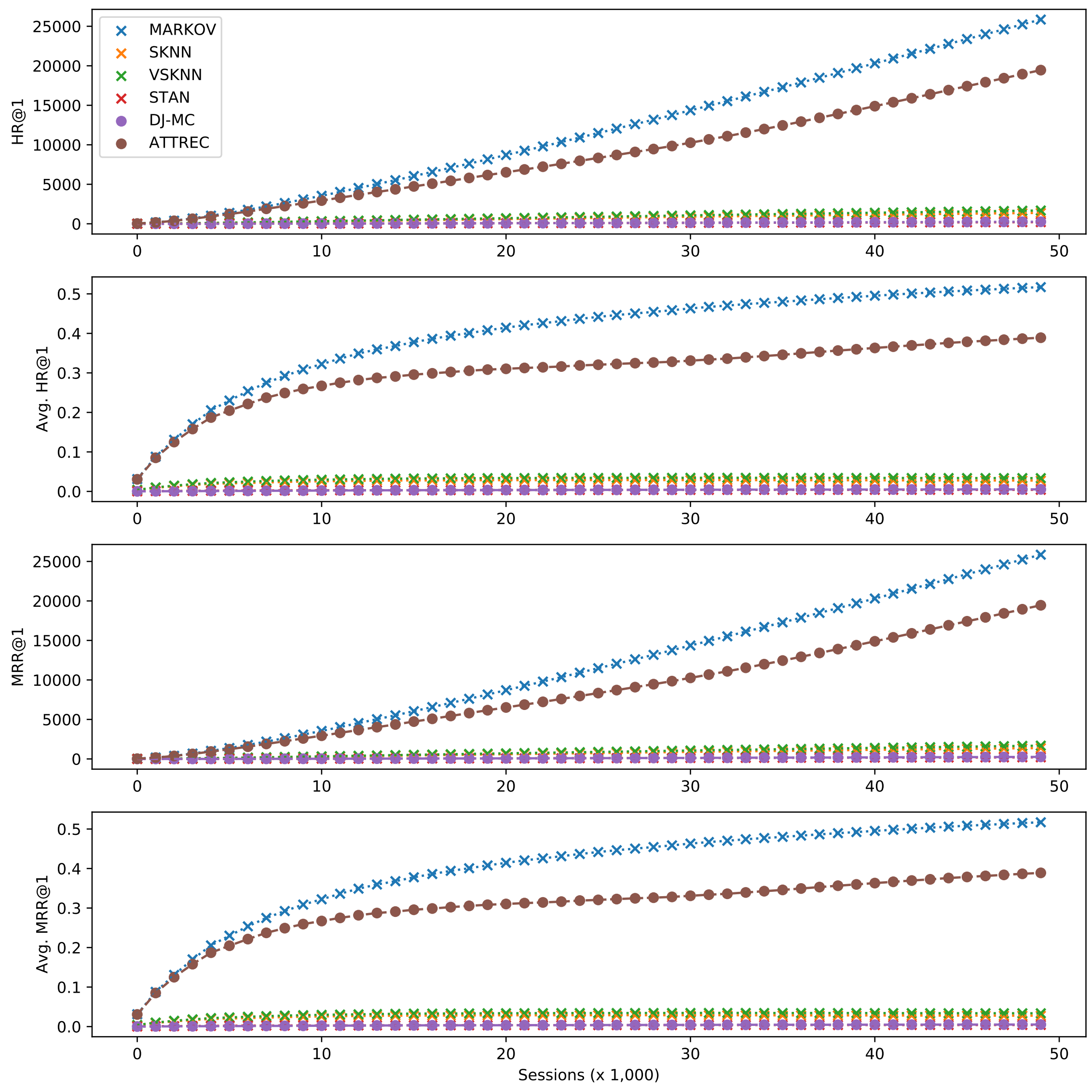


Figure A.4: *Results measured for the stream-based recommendation methods, with K=1 and for 50,000 listening sessions. (First) Accumulated HR@1 along the 50,000 listening sessions. (Second) The accumulated HR@1 is divided by the number of sessions. (Third) Accumulated MRR@1 along the 50,000 listening sessions. (Fourth) The accumulated MRR@1 is divided by the number of sessions.*

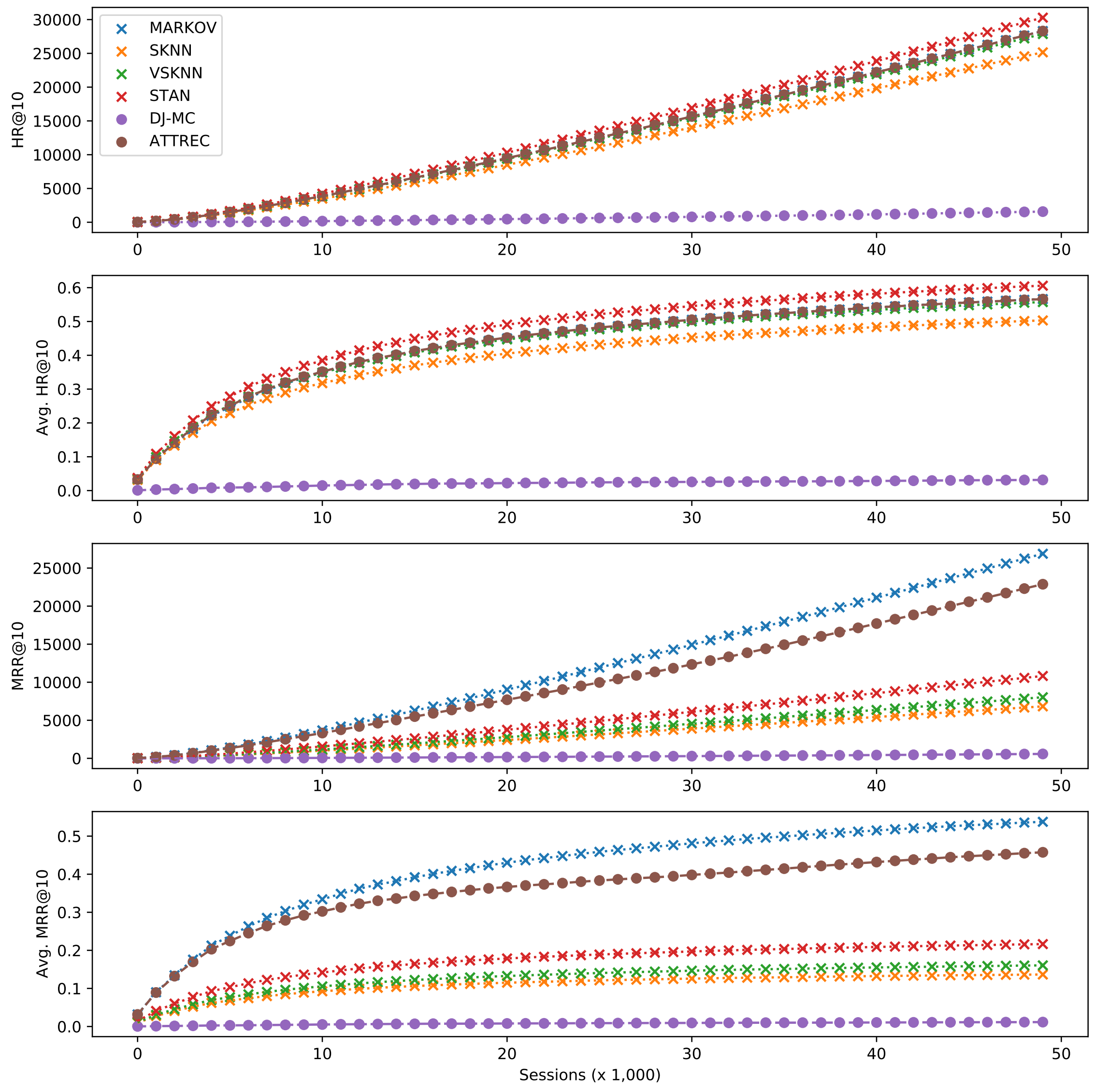


Figure A.5: *Results measured for the stream-based recommendation methods, with K=10 and for 50,000 listening sessions. (First) Accumulated HR@10 along the 50,000 listening sessions. (Second) The accumulated HR@10 is divided by the number of sessions. (Third) Accumulated MRR@10 along the 50,000 listening sessions. (Fourth) The accumulated MRR@10 is divided by the number of sessions.*

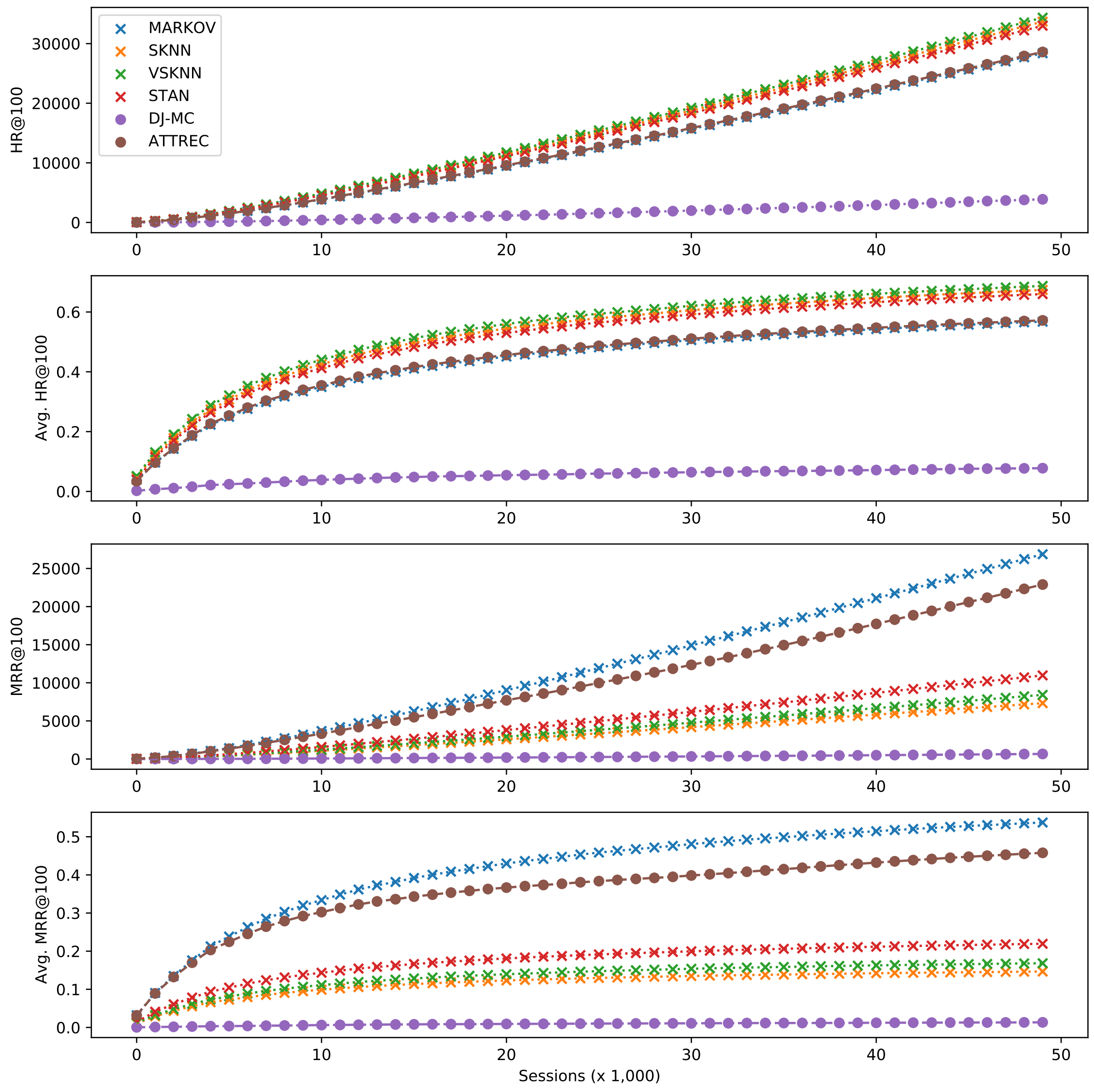


Figure A.6: *Results measured for the stream-based recommendation methods, with K=100 and for 50,000 listening sessions. (First) Accumulated HR@100 along the 50,000 listening sessions. (Second) The accumulated HR@100 is divided by the number of sessions. (Third) Accumulated MRR@100 along the 50,000 listening sessions. (Fourth) The accumulated MRR@100 is divided by the number of sessions.*

A.6 List of Publication

Journal Articles

- TOFANI, A.; BORGES, R.; QUEIROZ M. . Dynamic session-based music recommendation using information retrieval techniques. *User Modeling and User-Adapted Interaction* (Accepted for publication).
- SIMURRA, I; BORGES, R. . Analysis of Ligeti's Atmosphères by Means of Computational and Symbolic Resources. *Revista Música*, 21(1), 369-394. https://doi.org/10.11606/rm.v21i1.188846.
- BORGES, R.; QUEIROZ, M. . Automatic Music Recommendation Based on Acoustic Content and Implicit Listening Feedback. *Revista Música Hodie*, [S.l.], v. 18, n. 1, p. 31 - 43, jun. 2018. ISSN 1676-3939 https://doi.org/10.5216/mh.v18i1.53569.

Proceedings

- QUEIROZ, M.; BORGES, R. C. . Chroma Interval Content as a Key-Independent Harmonic Progression Feature. Proceedings of MMRP 2019.
- BORGES, R. C.; QUEIROZ, M. . Evolution of timbre diversity in a dataset of brazilian popular music: 1950-2000. Proceedings of SYSMUS 2018.
- SIMURRA, I. E.; BORGES, R. C. . Combining Automatic Segmentation and Symbolic Analysis based on Timbre Features – A Case Study from Ligeti's Atmosphères. Proceedings of SYSMUS 2018.
- BORGES, R. C.; QUEIROZ, M. . A Probabilistic Model For Recommending Music Based on Acoustic Features and Social Data. Proceedings of SBCM 2017.

References

[Ahmad Wasfi 1998] Ahmad M. Ahmad Wasfi. “Collecting user access patterns for building user profiles and collaborative filtering”. In: *Proceedings of the 4th International Conference on Intelligent User Interfaces.* IUI ’99. Los Angeles, California, USA, 1998, pp. 57–64. isbn: 1581130988. doi: 10.1145/291080.291091. url: https://doi.org/10.1145/291080.291091 (cit. on p. 32).

[Auer *et al.* 2019] Peter Auer *et al.* “Achieving optimal dynamic regret for non-stationary bandits without prior information”. In: *Proceedings of the Thirty-Second Conference on Learning Theory.* Vol. 99. Proceedings of Machine Learning Research. PMLR, 2019, pp. 159–163. url: https://proceedings.mlr.press/v99/auer19b.html (cit. on p. 40).

[Barkan *et al.* 2019] Oren Barkan, Noam Koenigstein, Eylon Yogev, and Ori Katz. “Cb2cf: a neural multiview content-to-collaborative filtering model for completely cold item recommendations”. In: *Proceedings of the 13th ACM Conference on Recommender Systems.* RecSys ’19. Copenhagen, Denmark: Association for Computing Machinery, 2019, pp. 228–236. isbn: 9781450362436. doi: 10.1145/3298689.3347038. url: https://doi.org/10.1145/3298689.3347038 (cit. on p. 5).

[Bellogín *et al.* 2017] Alejandro Bellogín, Pablo Castells, and Iván Cantador. “Statistical biases in information retrieval metrics for recommender systems”. In: *Inf. Retr. J.* 20.6 (2017), pp. 606–634. url: https://doi.org/10.1007/s10791-017-9312-z (cit. on p. 70).

[Bengio *et al.* 1994] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. doi: 10.1109/72.279181 (cit. on p. 30).

[Bogdanov, Haro, *et al.* 2013] Dmitry Bogdanov, Martín Haro, *et al.* “Semantic audio content-based music recommendation and visualization based on user preference examples”. In: *Information Processing and Management* 49.1 (2013), pp. 13–33. issn: 0306-4573. doi: https://doi.org/10.1016/j.ipm.2012.06.004. url: https://www.sciencedirect.com/science/article/pii/S0306457312000763 (cit. on pp. 5, 9).

[Bogdanov, Serrà, *et al.* 2011] Dmitry Bogdanov, Joan Serrà, Nicolas Wack, Perfecto Herrera, and Xavier Serra. “Unifying low-level and high-level music similarity measures”. In: *IEEE Transactions on Multimedia* 13.4 (2011), pp. 687–701. doi: 10.1109/TMM.2011.2125784 (cit. on pp. 16, 43).

[Bonnin and Jannach 2014] Geoffray Bonnin and Dietmar Jannach. “Automated generation of music playlists: survey and experiments”. In: *ACM Comput. Surv.* 47.2 (2014). issn: 0360-0300. doi: 10.1145/2652481. url: https://doi.org/10.1145/2652481 (cit. on p. 7).

[Borges and Queiroz 2018] Rodrigo Borges and Marcelo Queiroz. “Automatic music recommendation based on acoustic content and implicit listening feedback”. In: *Revista Música Hodie* 18.1 (2018), pp. 31–43. issn: 1676-3939. doi: 10.5216/mh.v18i1.53569. url: https://www.revistas.ufg.br/musica/article/view/53569 (cit. on p. 5).

[Breese *et al.* 1998] John S. Breese, David Heckerman, and Carl Kadie. “Empirical analysis of predictive algorithms for collaborative filtering”. In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence.* UAI’98. Madison, Wisconsin, 1998, pp. 43–52. isbn: 155860555X (cit. on p. 32).

[Burton *et al.* 1983] D. Burton, J. Shore, and J. Buck. “A generalization of isolated word recognition using vector quantization”. In: *ICASSP ’83. IEEE International Conference on Acoustics, Speech, and Signal Processing.* Vol. 8. 1983, pp. 1021–1024. doi: 10.1109/ICASSP.1983.1171915 (cit. on p. 14).

[Campos *et al.* 2014] Pedro G. Campos, Fernando Diez, and Iván Cantador. “Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols”. In: *User Model. User Adapt. Interact.* 24.1-2 (2014), pp. 67–119. doi: 10.1007/s11257-012-9136-x. url: https://doi.org/10.1007/s11257-012-9136-x (cit. on p. 7).

[Cano *et al.* 2005a] Pedro Cano, Markus Koppenberger, and Nicolas Wack. “Content-based music audio recommendation”. In: *Proceedings of the 13th Annual ACM International Conference on Multimedia.* MULTIMEDIA ’05. Hilton, Singapore, 2005, pp. 211–212. isbn: 1595930442. doi: 10.1145/1101149.1101181. url: https://doi.org/10.1145/1101149.1101181 (cit. on p. 43).

[Cano *et al.* 2005b] Pedro Cano, Markus Koppenberger, and Nicolas Wack. “Content-based music audio recommendation”. In: *Proceedings of the 13th ACM International Conference on Multimedia, Singapore, November 6-11, 2005.* ACM, 2005, pp. 211–212. url: https://doi.org/10.1145/1101149.1101181 (cit. on p. 2).

[Cardoso *et al.* 2016] João Paulo V. Cardoso *et al.* “Mixtape: direction-based navigation in large media collections”. In: *Proceedings of the 17th International Society for Music Information Retrieval Conference, ISMIR 2016, New York City, United States, August 7-11, 2016.* 2016, pp. 454–460 (cit. on p. 8).

[Chen *et al.* 2012] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. “Playlist prediction via metric embedding”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’12. Beijing, China, 2012, pp. 714–722. isbn: 9781450314626. doi: 10.1145/2339530.2339643. url: https://doi.org/10.1145/2339530.2339643 (cit. on p. 7).

[Cho *et al.* 2014] Kyunghyun Cho *et al.* “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar.* ACL, 2014, pp. 1724–1734. doi: 10.3115/v1/d14-1179. url: https://doi.org/10.3115/v1/d14-1179 (cit. on p. 30).

[Choi *et al.* 2017] Keunwoo Choi, György Fazekas, Mark B. Sandler, and Kyunghyun Cho. “Transfer learning for music classification and regression tasks”. In: *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017.* 2017, pp. 141–149. url: https://ismir2017.smcnus.org/wp-content/uploads/2017/10/12%5C_Paper.pdf (cit. on p. 21).

[Chorowski *et al.* 2019] Jan Chorowski, Ron J. Weiss, Samy Bengio, and Aäron van den Oord. “Unsupervised speech representation learning using wavenet autoencoders”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.12 (2019), pp. 2041–2053. doi: 10.1109/TASLP.2019.2938863 (cit. on p. 21).

[Chou *et al.* 2016] Szu-Yu Chou, Yi-Hsuan Yang, Jyh-Shing Roger Jang, and Yu-Ching Lin. “Addressing cold start for next-song recommendation”. In: *Proceedings of the 10th ACM Conference on Recommender Systems.* RecSys ’16. Boston, Massachusetts, USA, 2016, pp. 115–118. isbn: 9781450340359. doi: 10.1145/2959100.2959156. url: https://doi.org/10.1145/2959100.2959156 (cit. on pp. 7, 47, 48).

[Chung *et al.* 2014] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. url: http://arxiv.org/abs/1412.3555 (cit. on pp. 7, 21).

[Devooght and Bersini 2016] Robin Devooght and Hugues Bersini. “Collaborative filtering with recurrent neural networks”. In: *CoRR* abs/1608.07400 (2016). arXiv: 1608.07400. url: http://arxiv.org/abs/1608.07400 (cit. on p. 7).

[Dieleman and Schrauwen 2014] S. Dieleman and B. Schrauwen. “End-to-end learning for music audio”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2014, pp. 6964–6968. doi: 10.1109/ICASSP.2014.6854950 (cit. on p. 21).

[Duchi *et al.* 2011] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *J. Mach. Learn. Res.* 12.null (July 2011), pp. 2121–2159. issn: 1532-4435 (cit. on p. 24).

[Elahi *et al.* 2019] Ehtsham Elahi, Wei Wang, Dave Ray, Aish Fenton, and Tony Jebara. “Variational low rank multinomials for collaborative filtering with side-information”. In: *Proceedings of the 13th ACM Conference on Recommender Systems*. RecSys ’19. Copenhagen, Denmark: Association for Computing Machinery, 2019, pp. 340–347. isbn: 9781450362436. doi: 10.1145/3298689.3347036. url: https://doi.org/10.1145/3298689.3347036 (cit. on p. 5).

[Flexer *et al.* 2010] Arthur Flexer, Martin Gasser, and Dominik Schnitzer. “Limitations of interactive music recommendation based on audio content”. In: *AM ’10, The 5th Audio Mostly Conference, Piteå, Sweden, September 15-17, 2010*. ACM, 2010, p. 13. doi: 10.1145/1859799.1859812. url: https://doi.org/10.1145/1859799.1859812 (cit. on pp. 9, 43).

[Forbes and Zhu 2011] Peter Forbes and Mu Zhu. “Content-boosted matrix factorization for recommender systems: experiments with recipe recommendation”. In: *Proceedings of the Fifth ACM Conference on Recommender Systems*. RecSys ’11. Chicago, Illinois, USA, 2011, pp. 261–264. isbn: 9781450306836. doi: 10.1145/2043932.2043979. url: https://doi.org/10.1145/2043932.2043979 (cit. on pp. 5, 7, 43, 44).

[Fressato *et al.* 2018] Eduardo Pereira Fressato, Arthur Fortes da Costa, and Marcelo Garcia Manzato. “Similarity-based matrix factorization for item cold-start in recommender systems”. In: *7th Brazilian Conference on Intelligent Systems, BRACIS 2018, São Paulo, Brazil, October 22-25, 2018*. IEEE Computer Society, 2018, pp. 342–347. doi: 10.1109/BRACIS.2018.00066. url: https://doi.org/10.1109/BRACIS.2018.00066 (cit. on p. 5).

[Ganchev *et al.* 2005] Todor Ganchev, Nikos Fakotakis, and Kokkinakis George. “Comparative evaluation of various mfcc implementations on the speaker verification task”. In: *Proceedings of the SPECOM* 1 (Jan. 2005) (cit. on p. 19).

[Garg *et al.* 2019] Diksha Garg, Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. “Sequence and time aware neighborhood for session-based recommendations: stan”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR’19. Paris, France, 2019, pp. 1069–1072. isbn: 9781450361729. doi: 10.1145/3331184.3331322. url: https://doi.org/10.1145/3331184.3331322 (cit. on pp. 40, 42).

[Al-Ghossein, Abdessalem, and BARRÉ 2021] Marie Al-Ghossein, Talel Abdessalem, and Anthony BARRÉ. “A survey on stream-based recommender systems”. In: *ACM Comput. Surv.* 54.5 (May 2021). issn: 0360-0300. doi: 10.1145/3453443. url: https://doi.org/10.1145/3453443 (cit. on p. 8).

[Al-Ghossein, Abdessalem, and Barré 2018] Marie Al-Ghossein, Talel Abdessalem, and Anthony Barré. "Dynamic local models for online recommendation". In: *Companion Proceedings of the The Web Conference 2018*. WWW '18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1419–1423. isbn: 9781450356404. doi: 10.1145/3184558.3191586. url: https://doi.org/10.1145/3184558.3191586 (cit. on p. 39).

[Goldberg *et al.* 1992] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. "Using collaborative filtering to weave an information tapestry". In: *Commun. ACM* 35.12 (1992), pp. 61–70. issn: 0001-0782. doi: 10.1145/138859.138867. url: https://doi.org/10.1145/138859.138867 (cit. on pp. 1, 4, 32).

[Goodfellow *et al.* 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 27–29).

[Gouvert *et al.* 2018] Olivier Gouvert, Thomas Oberlin, and Cédric Févotte. "Matrix co-factorization for cold-start recommendation". In: *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*. 2018, pp. 792–798. url: http://ismir2018.ircam.fr/doc/pdfs/142%5C_Paper.pdf (cit. on p. 5).

[Hamel and Eck 2010] Philippe Hamel and Douglas Eck. "Learning features from music audio with deep belief networks". In: *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13, 2010*. International Society for Music Information Retrieval, 2010, pp. 339–344. url: http://ismir2010.ismir.net/proceedings/ismir2010-58.pdf (cit. on p. 21).

[Hansen *et al.* 2020] Casper Hansen *et al.* "Contextual and sequential user embeddings for large-scale music recommendation". In: *Fourteenth ACM Conference on Recommender Systems*. RecSys '20. Virtual Event, Brazil, 2020, pp. 53–62. isbn: 9781450375832. doi: 10.1145/3383313.3412248. url: https://doi.org/10.1145/3383313.3412248 (cit. on p. 7).

[Hariri *et al.* 2015] Negar Hariri, Bamshad Mobasher, and Robin Burke. "Adapting to user preference changes in interactive recommendation". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. AAAI Press, 2015, pp. 4268–4274 (cit. on p. 8).

[He *et al.* 2017] Xiangnan He *et al.* "Neural collaborative filtering". In: *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*. 2017, pp. 173–182. doi: 10.1145/3038912.3052569. url: https://doi.org/10.1145/3038912.3052569 (cit. on p. 32).

[Hidasi *et al.* 2016] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. “Session-based recommendations with recurrent neural networks”. In: *4th International Conference on Learning Representations, ICLR*. 2016 (cit. on pp. 7, 29, 36, 37, 57, 58).

[G. E. Hinton and R. R. Salakhutdinov 2006] G. E. Hinton and R. R. Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *Science* 313.5786 (2006), pp. 504–507. doi: 10.1126/science.1127647. eprint: https://www.science.org/doi/pdf/10.1126/science.1127647. url: https://www.science.org/doi/abs/10.1126/science.1127647 (cit. on p. 24).

[Hochreiter and Schmidhuber 1997] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. issn: 0899-7667. doi: 10.1162/neco.1997.9.8.1735. url: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. on p. 30).

[Hoffman *et al.* 2008] Matthew D. Hoffman, David M. Blei, and Perry R. Cook. “Content-based musical similarity computation using the hierarchical dirichlet process”. In: *ISMIR 2008, 9th International Conference on Music Information Retrieval, Drexel University, Philadelphia, PA, USA, September 14-18, 2008*. 2008, pp. 349–354. url: http://ismir2008.ismir.net/papers/ISMIR2008%5C_130.pdf (cit. on p. 43).

[Hoffman *et al.* 2009] Matthew D. Hoffman, David M. Blei, and Perry R. Cook. “Easy as CBA: A simple probabilistic model for tagging music”. In: *Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, Kobe International Conference Center, Kobe, Japan, October 26-30, 2009*. International Society for Music Information Retrieval, 2009, pp. 369–374. url: http://ismir2009.ismir.net/proceedings/OS5-2.pdf (cit. on pp. 16, 20).

[Hosseinzadeh Aghdam *et al.* 2015] Mehdi Hosseinzadeh Aghdam, Negar Hariri, Bamshad Mobasher, and Robin Burke. “Adapting recommendations to contextual changes using hierarchical hidden markov models”. In: *Proceedings of the 9th ACM Conference on Recommender Systems*. RecSys ’15. Vienna, Austria, 2015, pp. 241–244. isbn: 9781450336925. doi: 10.1145/2792838.2799684. url: https://doi.org/10.1145/2792838.2799684 (cit. on p. 7).

[Hu *et al.* 2008] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative filtering for implicit feedback datasets”. In: *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. ICDM ’08. 2008, pp. 263–272. isbn: 978-0-7695-3502-9 (cit. on pp. 4, 33, 47, 73).

[Ioffe and Szegedy 2015] Sergey Ioffe and Christian Szegedy. “Batch normalization: accelerating deep network training by reducing internal covariate shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 448–456 (cit. on p. 74).

[Jaitly and G. Hinton 2011] N. Jaitly and G. Hinton. “Learning a better representation of speech soundwaves using restricted boltzmann machines”. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2011, pp. 5884–5887. doi: 10.1109/ICASSP.2011.5947700 (cit. on p. 21).

[Jannach and Ludewig 2017] Dietmar Jannach and Malte Ludewig. “When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation”. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems.* New York, NY, USA: ACM, 2017, pp. 306–310. isbn: 9781450346528. doi: 10.1145/3109859.3109872. url: https://dl.acm.org/doi/10.1145/3109859.3109872 (cit. on pp. 7, 40).

[Jannach, Mobasher, *et al.* 2020] Dietmar Jannach, Bamshad Mobasher, and Shlomo Berkovsky. “Research directions in session-based and sequential recommendation”. In: *User Model. User Adapt. Interact.* 30.4 (2020), pp. 609–616 (cit. on p. 35).

[Kaufmann *et al.* 2012] Emilie Kaufmann, Olivier Cappe, and Aurelien Garivier. “On bayesian upper confidence bounds for bandit problems”. In: *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics.* Vol. 22. Proceedings of Machine Learning Research. 2012, pp. 592–600 (cit. on pp. 9, 40, 44).

[Kim *et al.* 2018] Taejun Kim, Jongpil Lee, and Juhan Nam. “Sample-level cnn architectures for music auto-tagging using raw waveforms”. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE. 2018 (cit. on pp. 6, 21, 70, 74).

[Kingma and Welling 2014] Diederik P. Kingma and Max Welling. “Auto-encoding variational bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.* 2014. url: http://arxiv.org/abs/1312.6114 (cit. on pp. 25, 27, 54, 55).

[Kingma and Welling 2019] Diederik P. Kingma and Max Welling. “An introduction to variational autoencoders”. In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. issn: 1935-8237. doi: 10.1561/2200000056. url: http://dx.doi.org/10.1561/2200000056 (cit. on p. 25).

[Knees *et al.* 2006] Peter Knees, Tim Pohle, Markus Schedl, and Gerhard Widmer. “Combining audio-based similarity with web-based data to accelerate automatic music playlist generation”. In: *Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval.* MIR ’06. Santa Barbara, California, USA, 2006, pp. 147–154. isbn: 1595934952. doi: 10.1145/1178677.1178699. url: https://doi.org/10.1145/1178677.1178699 (cit. on p. 5).

[Koren *et al.* 2009a] Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix factorization techniques for recommender systems”. In: *Computer* 42.8 (2009), pp. 30–37. doi: 10.1109/MC.2009.263 (cit. on pp. 1, 4, 15).

[Koren *et al.* 2009b] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems". In: *Computer* 42.8 (2009), pp. 30–37. issn: 0018-9162. doi: 10.1109/MC.2009.263. url: https://doi.org/10.1109/MC.2009.263 (cit. on p. 15).

[Korzeniowski and Widmer 2016] Filip Korzeniowski and Gerhard Widmer. "Feature learning for chord recognition: the deep chroma extractor". In: *Proceedings of the 17th International Society for Music Information Retrieval Conference, ISMIR 2016, New York City, United States, August 7-11, 2016.* 2016, pp. 37–43. url: https://wp.nyu.edu/ismir2016/wp-content/uploads/sites/2294/2016/07/178%5C_Paper.pdf (cit. on p. 16).

[Kramer 1991] Mark A. Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE Journal* 37.2 (1991), pp. 233–243. doi: https://doi.org/10.1002/aic.690370209. eprint: https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690370209. url: https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209 (cit. on p. 24).

[Latifi *et al.* 2021] Sara Latifi, Noemi Mauro, and Dietmar Jannach. "Session-aware recommendation: a surprising quest for the state-of-the-art". In: *Information Sciences* 573 (2021), pp. 291–315. issn: 0020-0255. doi: https://doi.org/10.1016/j.ins.2021.05.048 (cit. on p. 7).

[J. Li *et al.* 2017] Jing Li *et al.* "Neural attentive session-based recommendation". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management.* CIKM '17. Singapore, Singapore, 2017, pp. 1419–1428. isbn: 9781450349185. doi: 10.1145/3132847.3132926. url: https://doi.org/10.1145/3132847.3132926 (cit. on pp. 7, 37, 38).

[L. Li *et al.* 2010] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010.* ACM, 2010, pp. 661–670. doi: 10.1145/1772690.1772758. url: https://doi.org/10.1145/1772690.1772758 (cit. on p. 40).

[Q. Li *et al.* 2004] Qing Li, Byeong Man Kim, Dong Hai Guan, and Duk whan Oh. "A music recommender based on audio features". In: *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* SIGIR '04. Sheffield, United Kingdom, 2004, pp. 532–533. isbn: 1581138814. doi: 10.1145/1008992.1009106. url: https://doi.org/10.1145/1008992.1009106 (cit. on pp. 5, 43).

[S. Li *et al.* 2016] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. “Collaborative filtering bandits”. In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval.* SIGIR ’16. Pisa, Italy: Association for Computing Machinery, 2016, pp. 539–548. ISBN: 9781450340694. DOI: 10.1145/2911451.2911548. URL: https://doi.org/10.1145/2911451.2911548 (cit. on p. 41).

[Liang *et al.* 2018] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. “Variational autoencoders for collaborative filtering”. In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW.* 2018, pp. 689–698. DOI: 10.1145/3178876.3186150 (cit. on pp. 4, 34, 73).

[Liebman *et al.* 2015] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. “DJ-MC: A reinforcement-learning agent for music playlist recommendation”. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015.* ACM, 2015, pp. 591–599. URL: http://dl.acm.org/citation.cfm?id=2772954 (cit. on pp. 9, 48, 49).

[Lipton 2015] Zachary Chase Lipton. “A critical review of recurrent neural networks for sequence learning”. In: *CoRR* abs/1506.00019 (2015). URL: http://arxiv.org/abs/1506.00019 (cit. on pp. 22, 29).

[Q. Liu *et al.* 2018] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. “Stamp: short-term attention/memory priority model for session-based recommendation”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* KDD ’18. London, United Kingdom, 2018, pp. 1831–1839. ISBN: 9781450355520. DOI: 10.1145/3219819.3219950. URL: https://doi.org/10.1145/3219819.3219950 (cit. on p. 7).

[X. Liu and Aberer 2014] Xin Liu and Karl Aberer. “Towards a dynamic top-n recommendation framework”. In: *Proceedings of the 8th ACM Conference on Recommender Systems.* RecSys ’14. Foster City, Silicon Valley, California, USA: Association for Computing Machinery, 2014, pp. 217–224. ISBN: 9781450326681. DOI: 10.1145/2645710.2645720. URL: https://doi.org/10.1145/2645710.2645720 (cit. on p. 9).

[Logan and Salomon 2001] B. Logan and A. Salomon. “A music similarity function based on signal analysis”. In: *IEEE International Conference on Multimedia and Expo, 2001. ICME 2001.* 2001, pp. 745–748. DOI: 10.1109/ICME.2001.1237829 (cit. on p. 43).

[Ludewig and Jannach 2018] Malte Ludewig and Dietmar Jannach. “Evaluation of session-based recommendation algorithms”. In: *User Modeling and User-Adapted Interaction* 28.4-5 (2018), pp. 331–390. ISSN: 15731391. DOI: 10.1007/s11257-018-9209-6. arXiv: 1803.09587 (cit. on pp. 7, 40, 41).

[Ludewig, Mauro, *et al.* 2021] Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. “Empirical analysis of session-based recommendation algorithms: A comparison of neural and non-neural approaches”. In: *User Modeling and User-Adapted Interaction* 31.1 (2021), pp. 149–181. issn: 15731391. doi: 10.1007/s11257-020-09277-1 (cit. on p. 7).

[Makhzani and Frey 2014] Alireza Makhzani and Brendan J. Frey. “K-sparse autoencoders”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.* Ed. by Yoshua Bengio and Yann LeCun. 2014. url: http://arxiv.org/abs/1312.5663 (cit. on p. 24).

[B. McFee *et al.* 2012] B. McFee, L. Barrington, and G. Lanckriet. “Learning content similarity for music recommendation”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.8 (2012), pp. 2207–2218. doi: 10.1109/TASL.2012.2199109 (cit. on pp. 16, 20, 43, 71).

[Brian McFee and Lanckriet 2011] Brian McFee and Gert R. G. Lanckriet. “The natural language of playlists”. In: *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami, Florida, USA, October 24-28, 2011.* 2011, pp. 537–542. url: http://ismir2011.ismir.net/papers/PS4-11.pdf (cit. on p. 7).

[Miranda and Jorge 2009] Catarina Miranda and Alípio Mário Jorge. “Item-based and user-based incremental collaborative filtering for web recommendations”. In: *Progress in Artificial Intelligence.* Ed. by Luís Seabra Lopes, Nuno Lau, Pedro Mariano, and Luís M. Rocha. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 673–684. isbn: 978-3-642-04686-5 (cit. on p. 39).

[Müller 2015] Meinard Müller. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications.* 1st. Springer Publishing Company, Incorporated, 2015 (cit. on p. 18).

[Müller *et al.* 2011] Meinard Müller, Daniel P. W. Ellis, Anssi Klapuri, and Gaël Richard. “Signal processing for music analysis”. In: *IEEE Journal of Selected Topics in Signal Processing* 5.6 (2011), pp. 1088–1110. doi: 10.1109/JSTSP.2011.2112333 (cit. on pp. 16, 18).

[Nair and Geoffrey E. Hinton 2010] Vinod Nair and Geoffrey E. Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning.* ICML’10. Haifa, Israel: Omnipress, 2010, pp. 807–814. isbn: 9781605589077 (cit. on p. 22).

[Ning *et al.* 2015] Xia Ning, Christian Desrosiers, and George Karypis. “A comprehensive survey of neighborhood-based recommendation methods”. In: *Recommender Systems Handbook.* Boston, MA: Springer US, 2015, pp. 37–76. isbn: 978-1-4899-7637-6. doi: 10.1007/978-1-4899-7637-6_2. url: https://doi.org/10.1007/978-1-4899-7637-6_2 (cit. on p. 4).

[Oord *et al.* 2013] Aäron van den Oord, Sander Dieleman, and Benjamin Schrauwen. "Deep content-based music recommendation". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. 2013, pp. 2643–2651 (cit. on pp. 5, 6, 45, 71, 73, 74).

[Oppenheim and Schafer 2009] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. 3rd. USA: Prentice Hall Press, 2009. isbn: 0131988425 (cit. on pp. 16, 18).

[Oramas *et al.* 2017] S. Oramas, O. Nieto, M. Sordo, and Xavier Serra. "A deep multimodal approach for cold-start music recommendation". In: *2nd Workshop on Deep Learning for Recommender Systems, at RecSys 2017*. Como, Italy, 2017. url: https://arxiv.org/abs/1706.09739 (cit. on p. 5).

[Papagelis *et al.* 2005] Manos Papagelis, Ioannis Rousidis, Dimitris Plexousakis, and Elias Theoharopoulos. "Incremental collaborative filtering for highly-scalable recommendation algorithms". In: *Foundations of Intelligent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 553–561. isbn: 978-3-540-31949-8 (cit. on p. 39).

[Pereira *et al.* 2019] Bruno L. Pereira, Alberto Ueda, Gustavo Penha, Rodrygo L. T. Santos, and Nivio Ziviani. "Online learning to rank for sequential music recommendation". In: *Proceedings of the 13th ACM Conference on Recommender Systems*. RecSys '19. Copenhagen, Denmark: Association for Computing Machinery, 2019, pp. 237–245. isbn: 9781450362436. doi: 10.1145/3298689.3347019. url: https://doi.org/10.1145/3298689.3347019 (cit. on p. 8).

[Platt 2017] Devin Platt. "Content-Based Music Recommendation with the LFM-1b Dataset and Sample-Level Deep Convolutional Neural Networks". MA thesis. UC San Diego, 2017 (cit. on pp. 6, 74).

[Purwins *et al.* 2019] H. Purwins *et al.* "Deep learning for audio signal processing". In: *IEEE Journal of Selected Topics in Signal Processing* 13.2 (2019), pp. 206–219. doi: 10.1109/JSTSP.2019.2908700 (cit. on p. 21).

[Quadrana *et al.* 2018] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. "Sequence-aware recommender systems". In: *ACM Comput. Surv.* 51.4 (2018). issn: 0360-0300. doi: 10.1145/3190616. url: https://doi.org/10.1145/3190616 (cit. on pp. 1, 6, 35).

[Rendle 2012] Steffen Rendle. "Factorization machines with libfm". In: *ACM Trans. Intell. Syst. Technol.* 3.3 (2012). issn: 2157-6904. doi: 10.1145/2168752.2168771. url: https://doi.org/10.1145/2168752.2168771 (cit. on p. 47).

[Rendle, Freudenthaler, *et al.* 2010] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. "Factorizing personalized markov chains for next-basket recommendation". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. 2010, pp. 811–820 (cit. on pp. 7, 36, 47).

[Rendle and Schmidt-Thieme 2008] Steffen Rendle and Lars Schmidt-Thieme. “Online-updating regularized kernel matrix factorization models for large-scale recommender systems”. In: *Proceedings of the 2008 ACM Conference on Recommender Systems*. RecSys ’08. Lausanne, Switzerland: Association for Computing Machinery, 2008, pp. 251–258. isbn: 9781605580937. doi: 10.1145/1454008.1454047. url: https://doi.org/10.1145/1454008.1454047 (cit. on p. 40).

[Rennie and Srebro 2005] Jasson D. M. Rennie and Nathan Srebro. “Fast maximum margin matrix factorization for collaborative prediction”. In: *Proceedings of the 22nd International Conference on Machine Learning*. ICML ’05. Bonn, Germany, 2005, pp. 713–719. isbn: 1595931805. doi: 10.1145/1102351.1102441. url: https://doi.org/10.1145/1102351.1102441 (cit. on p. 15).

[Ricci *et al.* 2011] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, eds. *Recommender Systems Handbook*. Springer, 2011. isbn: 978-0-387-85819-7 (cit. on p. 4).

[Rifai *et al.* 2011] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. “Contractive auto-encoders: explicit invariance during feature extraction”. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML’11. Bellevue, Washington, USA: Omnipress, 2011, pp. 833–840. isbn: 9781450306195 (cit. on p. 24).

[Rumelhart *et al.* 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning internal representations by error propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362. isbn: 026268053X (cit. on p. 23).

[R. Salakhutdinov and Mnih 2007] Ruslan Salakhutdinov and Andriy Mnih. “Probabilistic matrix factorization”. In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS’07. 2007, pp. 1257–1264. isbn: 9781605603520 (cit. on p. 15).

[Sanz-Cruzado *et al.* 2019] Javier Sanz-Cruzado, Pablo Castells, and Esther López. “A simple multi-armed nearest-neighbor bandit for interactive recommendation”. In: *Proceedings of the 13th ACM Conference on Recommender Systems*. RecSys ’19. 2019, pp. 358–362 (cit. on pp. 40, 41).

[Sarwar *et al.* 2001] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. “Item-based collaborative filtering recommendation algorithms”. In: *Proceedings of the 10th International Conference on World Wide Web*. WWW ’01. Hong Kong, Hong Kong, 2001, pp. 285–295. isbn: 1581133480. doi: 10.1145/371920.372071. url: https://doi.org/10.1145/371920.372071 (cit. on p. 32).

[Schedl 2016] Markus Schedl. “The lfm-1b dataset for music retrieval and recommendation”. In: *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*. ICMR ’16. New York, New York, USA, 2016, pp. 103–110. isbn: 978-1-4503-4359-6. doi: 10.1145/2911996.2912004. url: http://doi.acm.org/10.1145/2911996.2912004 (cit. on p. 69).

[Schedl *et al.* 2015] Markus Schedl, Peter Knees, Brian McFee, Dmitry Bogdanov, and Marius Kaminskas. “Music recommender systems”. In: *Recommender Systems Handbook*. Boston, MA: Springer US, 2015, pp. 453–492. isbn: 978-1-4899-7637-6. doi: 10.1007/978-1-4899-7637-6_13. url: https://doi.org/10.1007/978-1-4899-7637-6_13 (cit. on p. 1).

[Schein *et al.* 2002] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. “Methods and metrics for cold-start recommendations”. In: *SIGIR 2002: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 11-15, 2002, Tampere, Finland*. ACM, 2002, pp. 253–260. doi: 10.1145/564376.564421. url: https://doi.org/10.1145/564376.564421 (cit. on p. 2).

[Sedhain *et al.* 2015] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. “Autorec: autoencoders meet collaborative filtering”. In: *Proceedings of the 24th International Conference on World Wide Web*. WWW ’15 Companion. Florence, Italy, 2015, pp. 111–112. isbn: 9781450334730. doi: 10.1145/2740908.2742726. url: https://doi.org/10.1145/2740908.2742726 (cit. on p. 24).

[Seyerlehner *et al.* 2008] Klaus Seyerlehner, Gerhard Widmer, and Peter Knees. “Frame level audio similarity - a codebook approach”. In: *Proceedings of the 11th International Conference on Digital Audio Effects (DAFx-08)*. 2008 (cit. on pp. 16, 20).

[Shao *et al.* 2009] B. Shao, D. Wang, T. Li, and M. Ogihara. “Music recommendation based on acoustic features and user access patterns”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 17.8 (2009), pp. 1602–1611. doi: 10.1109/TASL.2009.2020893 (cit. on p. 43).

[Slaney *et al.* 2008] Malcolm Slaney, Kilian Q. Weinberger, and William White. “Learning a metric for music similarity”. In: *ISMIR 2008, 9th International Conference on Music Information Retrieval, Drexel University, Philadelphia, PA, USA, September 14-18, 2008*. 2008, pp. 313–318. url: http://ismir2008.ismir.net/papers/ISMIR2008%5C_148.pdf (cit. on p. 43).

[Srivastava *et al.* 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. issn: 1532-4435 (cit. on p. 24).

[Stevens *et al.* 1937] S. S. Stevens, J. Volkmann, and E. B. Newman. “A scale for the measurement of the psychological magnitude pitch”. In: *The Journal of the Acoustical Society of America* 8.3 (1937), pp. 185–190. doi: 10.1121/1.1915893. url: https://doi.org/10.1121/1.1915893 (cit. on p. 19).

[Vall *et al.* 2019] Andreu Vall *et al.* “Feature-combination hybrid recommender systems for automated music playlist continuation”. In: *User Modeling and User-Adapted Interaction* 29.2 (Apr. 2019), pp. 527–572. issn: 0924-1868. doi: 10.1007/s11257-018-9215-8. url: https://doi.org/10.1007/s11257-018-9215-8 (cit. on p. 40).

[Volkovs *et al.* 2017] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. “Dropoutnet: addressing cold start in recommender systems”. In: *Advances in Neural Information Processing Systems.* Ed. by I. Guyon *et al.* Vol. 30. Curran Associates, Inc., 2017. url: https://proceedings.neurips.cc/paper/2017/file/dbd22ba3bd0df8f385bdac3e9f8be207-Paper.pdf (cit. on p. 5).

[H. Wang *et al.* 2015] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. “Collaborative deep learning for recommender systems”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* KDD ’15. Sydney, NSW, Australia, 2015, pp. 1235–1244. isbn: 9781450336642. doi: 10.1145/2783258.2783273. url: https://doi.org/10.1145/2783258.2783273 (cit. on p. 32).

[X. Wang and Ye Wang 2014] Xinxi Wang and Ye Wang. “Improving content-based and hybrid music recommendation using deep learning”. In: *Proceedings of the 22nd ACM International Conference on Multimedia.* MM ’14. 2014, pp. 627–636. isbn: 9781450330633. doi: 10.1145/2647868.2654940. url: https://doi.org/10.1145/2647868.2654940 (cit. on pp. 6, 21, 45, 73).

[X. Wang, Yi Wang, *et al.* 2014] Xinxi Wang, Yi Wang, David Hsu, and Ye Wang. “Exploration in interactive personalized music recommendation: a reinforcement learning approach”. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 11.1 (2014) (cit. on pp. 9, 43).

[Williams and Zipser 1989] Ronald J. Williams and David Zipser. “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks”. In: *Neural Computation* 1.2 (June 1989), pp. 270–280. issn: 0899-7667. doi: 10.1162/neco.1989.1.2.270. eprint: https://direct.mit.edu/neco/article-pdf/1/2/270/811849/neco.1989.1.2.270.pdf. url: https://doi.org/10.1162/neco.1989.1.2.270 (cit. on p. 30).

[Wu *et al.* 2019] Shu Wu *et al.* “Session-based recommendation with graph neural networks”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI.* 2019, pp. 346–353. doi: 10.1609/aaai.v33i01.3301346. url: https://doi.org/10.1609/aaai.v33i01.3301346 (cit. on p. 7).

[Xing *et al.* 2014] Zhe Xing, Xinxi Wang, and Ye Wang. “Enhancing collaborative filtering music recommendation by balancing exploration and exploitation”. In: *Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR 2014, Taipei, Taiwan, October 27-31, 2014.* 2014, pp. 445–450 (cit. on pp. 9, 43).

[Xu *et al.* 2019] Chengfeng Xu *et al.* “Recurrent convolutional neural network for sequential recommendation”. In: *The World Wide Web Conference.* WWW ’19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 3398–3404. ISBN: 9781450366748. DOI: 10.1145/3308558.3313408. URL: https://doi.org/10.1145/3308558.3313408 (cit. on p. 7).

[Yoshii *et al.* 2006] Kazuyoshi Yoshii, Masataka Goto, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G. Okuno. “Hybrid collaborative and content-based music recommendation using probabilistic model with latent user preferences”. In: *ISMIR 2006, 7th International Conference on Music Information Retrieval, Victoria, Canada, 8-12 October 2006, Proceedings.* 2006, pp. 296–301 (cit. on p. 5).

[Zhang *et al.* 2019] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. “Deep learning based recommender system: a survey and new perspectives”. In: *ACM Comput. Surv.* 52.1 (2019). ISSN: 0360-0300. DOI: 10.1145/3285029. URL: https://doi.org/10.1145/3285029 (cit. on pp. 21, 33).

[Zhao *et al.* 2013] Xiaoxue Zhao, Weinan Zhang, and Jun Wang. “Interactive collaborative filtering”. In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management.* CIKM ’13. San Francisco, California, USA, 2013, pp. 1411–1420. ISBN: 9781450322638. DOI: 10.1145/2505515.2505690. URL: https://doi.org/10.1145/2505515.2505690 (cit. on pp. 1, 8, 40).

[Zimdars *et al.* 2001] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. “Using temporal data for making recommendations”. In: *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence.* UAI’01. 2001, pp. 580–588. ISBN: 1558608001 (cit. on p. 36).

[Zölzer 2008] Udo Zölzer. “Digital audio signal processing”. In: John Wiley & Sons, Ltd, 2008. Chap. 2, pp. 21–62. ISBN: 9780470680018. DOI: https://doi.org/10.1002/9780470680018.ch2. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470680018.ch2 (cit. on p. 14).