



Contributions to sequential learning for industrial-scale music recommendation

Walid Bendada

► To cite this version:

Walid Bendada. Contributions to sequential learning for industrial-scale music recommendation. Operations Research [math.OC]. Université Paris sciences et lettres, 2024. English. ⟨NNT : 2024UPSLD056⟩. ⟨tel-04990070⟩

HAL Id: tel-04990070

<https://theses.hal.science/tel-04990070v1>

Submitted on 13 Mar 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization



THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université Paris-Dauphine
Dans le cadre d'une cotutelle avec Deezer

**Contributions to Sequential Learning for Industrial-Scale
Music Recommendation**

Soutenue par

Walid Bendada

Le 12 décembre 2024

École doctorale n°543

**Sciences de la Décision,
des Organisations, de la
Société et de l'Échange**

Spécialité

Informatique

Préparée au

LAMSADE

Composition du jury :

Yann CHEVALEYRE Professeur, Université Paris Dauphine	<i>Président du jury Examineur</i>
Craig MACDONALD Professor, University of Glasgow	<i>Rapporteur</i>
Ludovic DENOYER Core Team Researcher, H Company	<i>Rapporteur</i>
Anna PAPPA Maître de conférences, Université Paris 8	<i>Examinatrice</i>
Marie AL GHOSSEIN Machine Learning Scientist, Crossing Minds	<i>Examinatrice</i>
Thomas BOUABÇA Director of Datascience, Deezer	<i>Directeur de thèse</i>
Tristan CAZENAVE Professeur, Université Paris-Dauphine	<i>Directeur de thèse</i>

Remerciements

Je souhaite remercier en premier lieu mes parents. Par le partage de leur passion pour la connaissance et par le modèle d'intégrité qu'ils incarnent à travers leurs paroles et leurs actions, ils ont contribué plus que quiconque à me faire devenir celui que je suis heureux d'être aujourd'hui.

Je remercie mes directeurs de thèse, Tristan Cazenave, au contact duquel j'ai appris ce qu'est la recherche, et Thomas Bouabça, grâce à qui j'ai pu expérimenter nombre de mes idées chez Deezer.

Je remercie tout particulièrement Guillaume, qui a officieusement endossé le rôle de troisième encadrant ; notre collaboration est à l'origine de l'intégralité de mes succès en tant qu'auteur publié, et j'ai hâte de pouvoir la poursuivre à l'avenir.

Merci à Aurélien d'avoir cru en mon potentiel et d'avoir été fidèle à ses convictions en soutenant mon travail de recherche appliqué au formidable produit qu'est Deezer.

Je remercie toute l'équipe recommandation : Théo, bien sûr, qui m'a toujours suivi dans des projets ambitieux qui n'auraient jamais pu aboutir sans lui, et Léa, que j'ai pris grand plaisir à voir s'affirmer au cours des années passées à travailler ensemble. Leur confiance constamment renouvelée m'honore. Benjamin, qui suit mes progrès avec bienveillance depuis les débuts ; Mathieu, qui a rendu possible la concrétisation de tant d'idées qui, quelques années auparavant, nous paraissaient totalement hors de portée ; Arielle, qui, par sa sollicitude et sa générosité, amplifie les amitiés formidables que ce travail m'a apportées.

Je remercie aussi toute l'équipe data science, et en particulier Mathieu, Céline, Jean-Michel et Clémence. Partager mon quotidien avec vous sera toujours un plaisir.

Merci à l'équipe research de m'accueillir en son sein. Je n'oublierai jamais que c'est grâce à Manuel, qui m'a choisi pour un stage en 2016, que toute cette histoire a commencé. Merci aussi à Romain, qui m'a encouragé à continuer jusqu'au bout lorsque, soudainement, après plus de deux ans de thèse, la passion des mathématiques s'est de nouveau emparée de moi.

Merci à tous les autres qui ont rendu ma vie au travail pendant cette thèse si plaisante : Marin, Tom, Kaveh, Amélie, Kian, Charles, Mark, Thibault, Rani, Man To, Salah, Tina, Domitille, Céline, Sharujah et beaucoup d'autres.

Merci à ceux qui étaient là avant, mais qui sont partis : Jérémy, Alexis, Aurélien, Baptiste, Julien, Bachir, Elisa, Valentin, Dorian, Ange, Cyrille, Félix, etc. Merci à nos stagiaires, Antoine et Alexandre, qui, j'espère, ont appris autant qu'ils nous ont appris. Merci aussi aux stagiaires des autres, d'avoir partagé avec moi ces beaux moments de vie.

Merci à tous ceux qui comptent dans ma vie : Cécile, ton soutien sans faille me remplit de fierté ; ma sœur, Mounia, et mon frère, Imad, ainsi que toute sa famille qui est aussi la mienne. Je suis heureux qu'aujourd'hui encore nous soyons toujours aussi liés.

Sami, ces 15 ans d'amitié n'en sont qu'à leur début, c'est une évidence. Toute ma fanfare, Rolland Merguez. Merci à tous les amis ; notre expérience commune est la chose la plus improbablement géniale qui me soit jamais arrivée. Anaïs, Noé, Valentin, Anaëlle, merci de continuer à être géniaux à chacune de nos retrouvailles. Merci à Pierre et Pierre-Louis d'avoir partagé avec moi leur logement pendant plusieurs années, c'était un grand plaisir. Merci à Karim d'être Karim.

Merci enfin aux membres de mon jury. Merci à mes rapporteurs, Craig Macdonald, dont les retours détaillés m'ont aidé à écrire ce manuscrit dans sa version la plus aboutie, et Ludovic Denoyer, qui, par ses questions pertinentes et provocantes, m'a poussé à interroger mon travail avec une hauteur inédite. Merci à Yann Chevaleyre, qui a présidé mon jury ; j'ai accueilli ses commentaires sur les implications concrètes de mes résultats mathématiques avec le plus grand des plaisirs. Merci aussi à Marie Al-Ghossein et Anna Pappa, qui se sont parfaitement approprié mon travail malgré les délais extrêmement courts qui leur étaient imposés.

À toutes les personnes que j'ai citées, merci.

Car cette thèse, vous y avez tous contribué.

Résumé

Les systèmes de recommandation (SR) aident les utilisateurs de services en ligne à découvrir du contenu intéressant dans des catalogues trop vastes pour être explorés manuellement. Sur les plateformes de streaming musical, les SR sélectionnent, parmi des millions de chansons, albums et artistes, ceux qui ont le plus de chances d'intéresser les utilisateurs, contribuant ainsi à l'initiation ou au renouvellement d'un abonnement.

Historiquement, la recherche en SR s'est focalisée sur la tâche qui consiste à prédire les notes que les utilisateurs attribueront à des articles qu'ils ne connaissent pas, en se basant sur les notes données à d'autres articles par le passé. Les modèles performants dans cette tâche peuvent être utilisés par les SR pour ordonner les articles de leurs catalogues et proposer de nouvelles recommandations. Cette ligne de recherche a mis en avant l'importance de construire des représentations latentes pour les utilisateurs et les articles à l'aide d'espaces vectoriels où la proximité géométrique correspond à une probabilité plus élevée d'interaction.

Cependant, ces modèles ne prennent pas en compte certains aspects clés du streaming musical, comme le fait que les chansons sont souvent écoutées de manière séquentielle, avec des retours réguliers des utilisateurs. De plus, les catalogues musicaux, souvent composés de dizaines de millions de chansons, sont en grande partie inexplorés. Pour ces raisons, la recherche récente en SR s'est tournée vers la modélisation de séquences et l'apprentissage par renforcement. Toutefois, au début de ce projet de recherche, ces approches étaient limitées par des défis de mise à l'échelle qui limitaient leur adoption dans des applications commerciales.

Dans cette thèse, nous présentons plusieurs contributions visant à rendre ces approches utilisables sur des plateformes de streaming avec des millions d'utilisateurs et de chansons. Notre travail est motivé par l'adoption généralisée de deux composants clés des SR modernes : la représentation vectorielle et la recherche efficace de plus proches voisins.

Tout d'abord, nous proposons une méthode générale pour construire des modèles d'extension automatique de playlists compatibles avec une application à grande échelle. Basée sur une stratégie de "représenter puis agréger", cette méthode garantit l'efficacité par conception tout en restant suffisamment flexible pour intégrer une large gamme de techniques de modélisation séquentielle, y compris celles basées sur les Transformers. Suite aux résultats des expériences d'A/B tests grande échelle rapportés dans cette thèse, notre méthode a été adoptée par l'entreprise de streaming musical Deezer pour proposer des chansons à ajouter dans les playlists de leurs utilisateurs.

Nous introduisons également un nouveau schéma d'exploration appelé exploration von Mises-Fisher (vMF-exp), qui permet d'explorer de grands ensembles d'actions lorsque celles-ci sont représentées par des vecteurs de norme unitaire. vMF-exp consiste à échantillonner initialement

un vecteur selon une distribution de von Mises-Fisher, suivie de l’exploration des plus proches voisins de ce vecteur. Cette approche permet de sélectionner un nombre pratiquement illimité d’actions candidates. Nous démontrons que, sous certaines hypothèses théoriques, vMF-exp maintient asymptotiquement la même probabilité d’explorer chaque action que l’exploration Boltzmann, une alternative populaire qui souffre de problèmes de mise à l’échelle en raison de la nécessité de calculer les valeurs de softmax de chaque action. Depuis janvier 2024, vMF-exp est utilisé par Deezer pour recommander des mix inspirés par des chansons initiales à des millions d’utilisateurs, en puisant dans des millions de chansons candidates.

Le cadre de représentation efficace et le schéma d’exploration introduits dans cette thèse devraient encourager de futures initiatives d’apprentissage par renforcement pour la recommandation musicale sur les plateformes de streaming.

Mots clés : Systèmes de recommandation, mise à l’échelle, prise de décision séquentielle, modèles probabilistes, apprentissage automatique, plateformes de streaming musical

Abstract

Recommender Systems (RS) help users of online services discover interesting content from catalogs of items that are too large for manual exploration. Specifically, music streaming platforms use RS to select, from millions of songs, albums, and artists, those most likely to interest users, contributing to the initiation or renewal of paid subscriptions. This selection process is often performed in real-time, repeatedly throughout a user’s journey on the platform, serving specific tasks such as choosing content for the homepage or automatically creating playlists.

Historically, research in RS has focused on predicting, given a dataset of ratings users have already provided, the future ratings they will give to unseen items. Models that perform well in this task can be used by real-world systems to rank items in live catalogs and surface more relevant recommendations. This line of research has highlighted the importance of building meaningful latent representations for users and items, using either content-based or collaborative filtering models. The complexity of these models ranges from linear projectors to deep neural networks, resulting in vector elements in Euclidean space, where geometrical proximity corresponds to a higher probability of interaction.

However, these widely adopted models do not account for key aspects of music consumption, such as the fact that songs are often listened to sequentially within listening sessions, with users regularly providing feedback. Moreover, music catalogs, commonly filled with tens of millions of songs, are mostly uncharted. For these reasons, recent research has shifted toward paradigms inspired by sequence modeling and reinforcement learning. Yet, when this research project began, these promising approaches faced scalability challenges that limited their adoption in commercial applications.

In this thesis, we present several contributions aimed at making these approaches usable in real-world music streaming applications with millions of users and songs. Our work is motivated by the widespread adoption of two key components in modern RS: vector representations of items and efficient nearest-neighbor search.

First, we propose a general framework to build scalable yet effective Automatic Playlist Continuation (APC) models for large-scale applications. Based on a represent-then-aggregate strategy, this framework ensures scalability by design while remaining flexible enough to incorporate a wide range of representation learning and sequence modeling techniques, including those based on Transformers. Following the results of large-scale online A/B test experiments reported in this thesis, our framework has now been adopted by the music streaming company Deezer to propose additional songs to users’ playlists.

We also introduce a novel exploration scheme called von Mises-Fisher exploration (vMF-exp), which is a scalable method for exploring large action sets when actions are represented

by hyperspherical embedding vectors. vMF-exp involves initially sampling a state embedding representation using a von Mises-Fisher hyperspherical distribution, followed by exploring the nearest neighbors of this representation. This approach scales to virtually unlimited numbers of candidate actions. We demonstrate that, under certain theoretical assumptions, vMF-exp asymptotically maintains the same probability of exploring each action as Boltzmann exploration, a popular alternative that suffers from scalability issues due to its requirement to compute softmax values for each action. Since January 2024, vMF-exp has been employed by Deezer to recommend mixes inspired by initial songs to millions of users, drawing from millions of possible actions for each playlist.

The efficient representation framework and exploration scheme introduced in this thesis should foster future attempts to deploy off-policy reinforcement learning models for music recommendation at an industrial scale.

Keywords : Recommendation systems, scaling, sequential decision making, probabilistic models, machine learning, music streaming platforms

Contents

Remerciements	i
Résumé	iii
Abstract	v
Contents	vii
List of Figures	x
List of Tables	xii
List of Acronyms and Symbols	xiii
1 Introduction	1
1.1 Context and Objectives	1
1.2 Overview of Scientific Contributions	4
1.2.1 Thesis Structure	5
2 Background	7
2.1 Music Streaming Platforms	7
2.1.1 Music Catalogs	7
2.1.2 Access to Music	8
2.1.3 Playlists	8
2.1.4 User Engagement	9
2.2 Music Recommendation Tasks	11
2.2.1 Homepage Personalization	11
2.2.2 Similar Items Display	11
2.2.3 Automatic Playlist Continuation	12
2.3 Traditional Recommendation Models	13
2.3.1 Ratings Prediction Task	14
2.3.2 Common Approaches	14
2.3.3 Inference	17
2.3.4 Embedding-Based Recommendation on Deezer	19
2.3.5 Shortcomings of the Ratings Prediction Task	21
2.4 Sequential Recommendation	21
2.4.1 Sequence Modeling	21
2.4.2 Automatic Playlist Continuation as Sequential Recommendation	22
2.4.3 Limitations to Real-World Usage	24

3	A Scalable Framework for Automatic Playlist Continuation	26
3.1	Introduction	26
3.2	Problem Formulation	27
3.2.1	Notation	27
3.2.2	APC	27
3.3	Large-Scale Playlist Continuation	28
3.3.1	Objectives and Requirements	28
3.3.2	A Scalable Framework for APC	29
3.3.3	Examples of RTA Architectures	31
3.3.4	Example of a Training Procedure	33
3.4	Offline Experiments	35
3.4.1	Experimental Setting	35
3.4.2	APC Models	36
3.4.3	Results	39
3.5	Online Experiments	41
3.5.1	APC on a Music Streaming Service	41
3.5.2	Online A/B Test Results	42
3.6	Track Mix Generation on Deezer	43
3.6.1	Track Mix, a Personalized Playlist Generation System on Deezer	43
3.6.2	A/B Test	43
3.7	Conclusion	45
4	On the Consistency of Average Embedding	47
4.1	Introduction	47
4.1.1	Motivations	47
4.1.2	Contributions	48
4.2	Problem Formulation and Evaluation	49
4.2.1	Preliminaries	49
4.2.2	Problem Formulation	50
4.2.3	Problem Evaluation : the Consistency Metric	51
4.3	A Theoretical Analysis of Consistency _k ($\mathcal{X}$)	57
4.3.1	Setting and Assumptions	57
4.3.2	Main Results and Interpretations	57
4.4	From Theory to Practice	60
4.4.1	Experimental Setting	60
4.4.2	Results and Discussion	61
4.5	Conclusion	61
4.A	Proof of Proposition 4.3	62
4.A.1	Preliminaries	62
4.A.2	Distribution of $s_{\text{in}} = s(u_{\text{in}}, \mu_{\mathcal{U}})$ and $s_{\text{out}} = s(u_{\text{out}}, \mu_{\mathcal{U}})$	63
4.A.3	Distribution of $s_{\text{diff}} = s_{\text{in}} - s_{\text{out}}$	64
4.B	Proof of Proposition 4.4	65
5	Exploring Large Action Sets with Hyperspherical Embeddings using von Mises-Fisher Sampling	68
5.1	Introduction	68
5.2	Preliminaries	69
5.2.1	Problem Formulation	69
5.2.2	Limitations of Existing Exploration Strategies	71
5.3	From Boltzmann to von Mises-Fisher (vMF) Exploration	73

5.3.1	von Mises–Fisher Exploration	73
5.3.2	Properties	74
5.4	Theoretical Comparison of vMF-exp and B-exp	75
5.4.1	Setting and Assumptions	75
5.4.2	Results	75
5.4.3	Discussion	76
5.5	Application to Large-Scale Music Recommendation	78
5.5.1	Experimental Setting	79
5.5.2	Results	79
5.6	Conclusion	80
5.A	Asymptotic Behavior of Boltzmann Exploration (Proof of Proposition 5.2)	81
5.B	Asymptotic Behavior of vMF Exploration in $d = 2$ dimensions (Proof of Proposition 5.3, Part 1)	82
5.B.1	Zero-Order Estimate	85
5.B.2	Bounding of the Remainder Term	85
5.C	Asymptotic Behavior of vMF Exploration in $d > 2$ dimensions (Proofs of Proposition 5.3, Part 2, and of Proposition 5.4)	87
5.C.1	Intermediary Lemmas	87
5.C.2	Proof of Proposition 5.3	93
5.D	Similar Asymptotic Behavior of B-exp and vMF-exp for Large Action Sets (Proof of Proposition 5.1)	97
5.E	Link with Thompson Sampling	97
5.F	Monte Carlo Simulations	99
5.G	Sampling from the von Mises-Fisher Distribution	100
5.G.1	Radial-tangent decomposition	100
5.G.2	vMF distribution	100
5.G.3	Sampling $\tilde{t} = \langle V, \tilde{V} \rangle$	100
5.G.4	Sampling $\tilde{V}_O$	100
5.G.5	Wrapping up	101
5.H	Additional Experiments on a public dataset: Glove-25 word embeddings	102
5.H.1	P1: Scalability	102
5.H.2	P2,P3: Unrestricted Radius and Order Preservation	103
5.H.3	Theoretical Approximations	103
6	Opening: Towards Reinforcement Learning for Sequential Music Recommendation	106
6.1	Introduction	106
6.2	Background on Reinforcement Learning	107
6.2.1	Markov Decision Process	107
6.2.2	Policy	108
6.3	Application to Interactive Track Mix	108
6.3.1	Problem Setting	108
6.3.2	Our Approach	109
6.4	Future Work	111
	Conclusion and Future Work	113
	Bibliography	114

List of Figures

1.1	Hierarchy of musical items.	2
2.1	Music engagement mix 2023 (IFPI, 2023).	9
2.2	Market shares of music streaming platforms in 2023.	10
2.3	Homepage of Deezer.	12
2.4	Artist page on Deezer.	13
2.5	T-SNE visualization of song embeddings used on Deezer.	16
2.6	Possible representations of <i>Automatic Playlist Continuation</i> (APC) datasets.	23
3.1	An example of a user playlist on Deezer.	28
3.2	The <i>Represent-Then-Aggregate</i> framework.	32
3.3	Training procedure of an RTA model.	34
3.4	APC on the Million Playlist Dataset for increasing n_{seed}	40
3.5	APC on Deezer.	42
3.6	Results of online A/B test results of RTA for APC.	43
3.7	Interface of Track Mix on the website version of Deezer.	44
3.8	An example of the beginning of a Track Mix playlist on Deezer.	44
3.9	Illustration of the inference steps of the Track Mix feature.	45
4.1	Representing a sequence of embeddings by its average.	49
4.2	Averaging embeddings of songs to obtain a playlist or user embedding.	50
4.3	Average consistency score for RTA models.	52
4.4	Illustration of 3 embeddings of dimension 1.	55
4.5	Illustration of 3 embeddings of dimension 2.	56
4.6	Consistency $_k(\mathcal{X})$ for normal and uniform distributions.	58
4.7	Inner product similarity values between 1 000 random d -dimensional vectors.	59
4.8	Comparison of the Consistency $_k(\mathcal{X})$ scores obtained by a direct <i>computation</i>	59
4.9	Consistency $_k(\mathcal{X})$ scores of real-world embeddings.	60
4.10	Ranking s_{in} and s_{out} statistics.	66
5.1	Messages sent to Deezer from users complaining about their recommendations.	73
5.2	Public tweets from users complaining about their Deezer recommendations on X.	73
5.3	Distribution of von-Mises Fisher distribution and illustration of Voronoï tessellation.	74
5.4	Simulations of von-Mises Fisher probabilities.	77
5.5	Track Mix on Deezer.	78
5.6	von-Mises Fisher exploration in 2 dimensions.	84
5.7	Normal vector of a Voronoï cell on the sphere.	89
5.8	von-Mises Fisher exploration in 3 dimensions.	95

5.9	Monte Carlo simulations for probabilities of vMF-exp on uniformly sampled embeddings.	99
5.10	Comparison of B-exp and vMF-exp on Glove-25 embeddings dataset.	104
5.11	Monte Carlo simulations for probabilities of vMF-exp on Glove-25 embeddings dataset.	105

List of Tables

2.1	A classification of user feedback types.	10
2.2	Public datasets for APC.	23
3.1	Automatic Playlist Continuation (APC) on the Million Playlist Dataset.	39
3.2	Results for online A/B test on Track Mix.	45
5.1	Throughput of ANN algorithms on the Glove-25 embeddings dataset.	103

List of Acronyms and Symbols

ALS	Alternate Least Squares
ANN	Approximate Nearest Neighbors
APC	Automatic Playlist Continuation
B-exp	Boltzmann Exploration
CDF	Cumulative Distribution Function
CF	Collaborative Filtering
CLT	Central Limit Theorem
CNN	Convolutional Neural Network
FM	Factorization Machines
GRU	Gated Recurrent Unit
i.i.d	Independent Identically Distributed
LSH	Locality Sensitive Hashing
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
ML	Machine Learning
MPD	Million Playlist Dataset
NCF	Neural Collaborative Filtering
NDCG	Normalized Discounted Cumulative Gain
NLP	Natural Language Processing
PDF	Probability Density Function
PMI	Pointwise Mutual Information
RL	Reinforcement Learning
RMSE	Root Mean Square Error

RNN Recurrent Neural Networks

RS Recommender Systems

RTA Represent-Then-Aggregate

SVD Singular Value Decomposition

vMF von-Mises Fisher

vMF-exp von Mises-Fisher Exploration

Chapter 1

Introduction

This introductory chapter provides a general overview of this PhD thesis. Firstly, we present the context and objectives of this work, which is at the intersection of large-scale music recommendation and sequential decision making. Then, we detail the scientific contributions as well as the organization of the remainder of this thesis. We also list the publications that resulted from the research conducted during this PhD.

1.1 Context and Objectives

Recommender Systems (RS) are essential tools for online services that provide access to large catalogs, such as e-commerce websites (Smith et al., 2017; J. Wang et al., 2018), as well as video (Gomez-Uribe et al., 2015; Covington et al., 2016a) and music (Jacobson et al., 2016; Schedl et al., 2018; Bendada, Salha, et al., 2020; Epure et al., 2020) streaming services. They help users navigate vast amounts of content, discover new products, movies, or songs they might enjoy, and enhance user experience and engagement (Bobadilla, Ortega, Hernando, and Gutiérrez, 2013; Schedl et al., 2018; Mu, 2018; S. Zhang et al., 2019).

The French music streaming service Deezer (Deezer, 2024) provides unlimited access to a large catalog exceeding 100 million music tracks. Music catalogs are traditionally organized into albums, which are collections of tracks released simultaneously by the same artist, intended to be experienced as a coherent piece of art. In addition to albums, playlists can be created by users or professional curators looking to craft novel listening experiences using previously released songs. Over the years, the rise of music streaming platforms as the primary means of music consumption has led to a shift from album listening to playlist listening (Jakobsen, 2016). An illustration of the types of items found in a music catalog, as well as their relationships, can be seen in Figure 1.1.

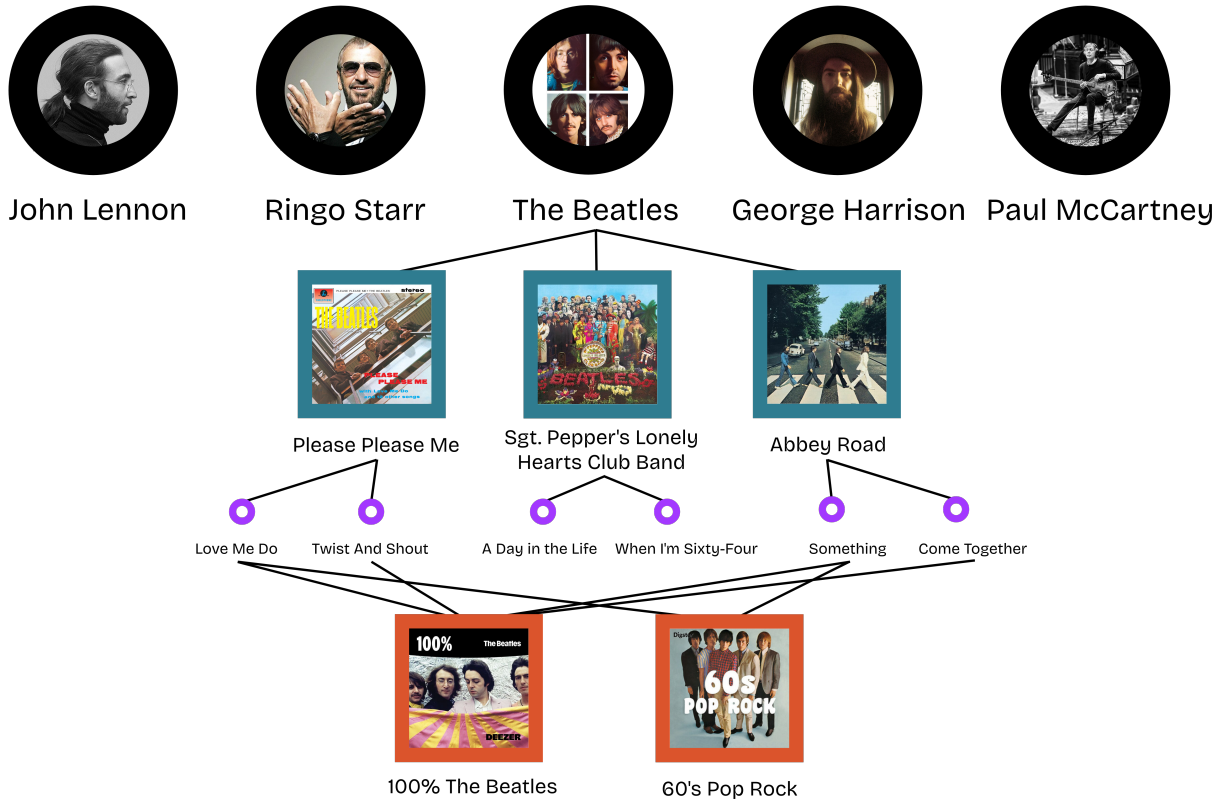


Figure 1.1: Illustration of items available on a music catalog along with their hierarchical relationships. The most elementary item is the *song*, which belongs to an *album* released by one or several *artists*. Additionally, songs can be gathered together by users or professional curators to create *playlists*.

To help its users navigate the catalog and discover new content they might enjoy, Deezer integrates a variety of features that require a large-scale music recommender system. Some of these features are common across many online services, like a personalized homepage filled with algorithmically selected content identified as likely to interest users (Bendada, Salha, et al., 2020), allowing the platform to benefit from methods previously developed in the extensive literature on recommender systems.

Traditionally, research in recommender systems has focused on the ratings prediction problem, exemplified by the Netflix Prize starting in 2007 (Bennett et al., 2007). This task involves predicting future ratings for items based on a dataset of ratings that users have provided for other items, and it is often evaluated as either a regression or a binary classification problem. The rationale behind the extensive work on such problems is that models predicting pointwise estimates of a user’s interest in an item can be used by real-world systems to rank items available in live catalogs and surface more relevant recommendations.

This line of research has highlighted the importance of building meaningful latent representations for users and items, often represented as vector elements in Euclidean space, where geometric proximity correlates with an increased probability of interaction. These representations can be constructed using either content-based (Pazzani et al., 2007; Van Den Oord et al., 2013; Javed et al., 2021) or collaborative filtering models (Y. Hu et al., 2008; Marlin et al., 2009;

Bobadilla, Ortega, Hernando, and Bernal, 2012; H. Wang et al., 2015; Park et al., 2018; Koren, Rendle, et al., 2021), with complexities ranging from linear projectors (Koren, Bell, and Volinsky, 2009) to deep neural networks (He, Liao, et al., 2017a). Such representations can be used to compute pairwise similarities, not only between a user and an item as originally intended, but also between two users or two items, providing a generic solution to a variety of recommendation problems where model inference essentially reduces to nearest neighbor search in a vector space.

However, some key features provided by services such as Deezer are specific to music streaming platforms and center around the automatic creation of personalized mixes of tracks based on certain criteria, such as music genre, mood, or a set of initial songs (Bontempelli et al., 2022; Bendada, Bontempelli, et al., 2023). Building models for these features requires consideration of defining aspects of music consumption that are often overlooked by commonly used recommender system models.

Typically, people listen to songs in sessions that are expected to pertain some form of consistency, violating the independence assumption of traditional ratings prediction models. Also, during these sessions, users may actively provide feedbacks that could be used to dynamically, in an online fashion, update recommendations. Another distinguishing aspect is that music listening is a highly emotional activity (Schedl et al., 2018), leading users to behave in ways that differ from their behavior towards other types of items. For instance, users may crave the same songs on repeat (whereas repeatedly purchasing the same item in a short time frame is relatively rare on e-commerce platforms) or react strongly against algorithms that provide poor recommendations, sometimes resulting in actions as drastic as terminating their subscriptions.

Above all, music catalogs on streaming platforms can nowadays be made of up to hundreds of millions of songs, and most of the features proposed to user require to select in real-time a subset of songs to recommend to users, making it challenging to straightforwardly adopt sophisticated models developed in the context of academic research.

Indeed, to better take into account some of the specificities listed above, part of the recent research on music recommendation (Schedl et al., 2018) has shifted from predicting pointwise probabilities of interactions with items to other paradigms. For instance, sequential recommendation (Guillou et al., 2016; Donkers et al., 2017; Tintarev et al., 2017; D. Wang, Deng, and G. Xu, 2018; Garg et al., 2019; Tsai et al., 2019; S. Wang et al., 2019) aims to predict near-future interactions based on observations of users’ reactions to items within the same sequence. Alternatively, reinforcement learning (Xinxi Wang et al., 2014; Afsar et al., 2021; B. Hu et al., 2017; J.-W. Chang et al., 2021; Sakurai et al., 2022) models recommendation sessions as trajectories in Markov decision processes (Sutton et al., 2018), where the model is expected to maximize the rewards of future trajectories. However, at the beginning of this thesis, scalability issues hindered the widespread adoption of these promising approaches by commercial applications.

1.2 Overview of Scientific Contributions

In this thesis, we present several contributions aimed at making sequential models usable in real-world music streaming applications with millions of users and songs. Key sub-problems addressed in our work include the real-time representation of sequences of listening sessions using neural networks to accurately expand these sequences, as well as the selection of songs based on stochastic exploration policies with properties that make them particularly suited for industrial-scale music recommendations. For both of these tasks, we provide solutions grounded in mathematically proven properties, which are empirically validated through A/B testing on millions of Deezer users, resulting in a positive impact on key performance indicators commonly adopted by the music industry to monitor user satisfaction.

The detailed contributions of this thesis are the following:

- We introduce a framework named *represent-then-aggregate* for building scalable APC (Automatic Playlist Continuation) models. This framework leverages embedding representations for songs and neural network-based aggregation functions trained on sequence modeling tasks to efficiently assign a relevant latent representation to any playlist, allowing real-time recommendation within catalogs of millions of songs.
- We demonstrate that our framework can be effectively implemented in a production service by reporting experiments, conducted on Deezer, which outline significant improvements over a legacy model that relied on averaging song embeddings to represent a playlist.
- We mathematically study the relevance of relying on averages of vectors, a convenient practice commonly performed in both academic and industrial work. Specifically, we introduce the notion of consistency, a desirable property of aggregators in the context of sequence representation, that quantifies the ability of an aggregator to preserve information about the embeddings originally present in a sequence of embeddings. We then mathematically study the problem of averaging *Independent Identically Distributed* (i.i.d) vector embeddings of i.i.d dimensions, and show that the theoretical value of this consistency can be much higher than what is observed in real-world song embeddings constructed using *Singular Value Decomposition* (SVD) (Paterek, 2007) or *Alternate Least Squares* (ALS) (Koren, Bell, and Volinsky, 2009). This key result motivates the design of future embedding construction techniques that account for consistency during their training process to better handle sequences of embeddings.
- We introduce a novel exploration scheme called *von Mises-Fisher Exploration* (vMF-exp). We establish its mathematical relation to the popular method of Boltzmann exploration and demonstrate, both theoretically and through A/B test experiments on Deezer, that it is better suited for exploring large action sets, such as music catalogs.
- We show how the problem of continuously generating song recommendations for users can be formalized as a Markov decision process. By considering that the action to be predicted by the agent is an embedding vector, and that embeddings representing any song from a

music catalog are part of the environment the agent evolves in, our analysis bridges the gap between scalable recommender systems and continuous action space environments, paving the way for large-scale usage of deep reinforcement learning in music streaming platforms.

Overall, our work builds on two essential components of modern recommender systems: the vector representation of items, as well as efficient nearest neighbor search in vector spaces.

1.2.1 Thesis Structure

Chapter 2 In this chapter, we introduce key tools and concepts related to music streaming platforms. We also survey related work in embedding-based recommendation as well sequence modeling and its usage for music recommendation.

Chapter 3 In this chapter, we introduce *represent-then-aggregate*, our framework for scalable automatic playlist continuation models. This framework leverages embedding representations for songs and neural network-based aggregation functions trained on sequence modeling tasks to perform real-time recommendations within catalogs of millions of songs. We report positive results for both offline and online experiments conducted on sequential music recommendation tasks. Additionally, we demonstrate how models trained using our framework can be applied to other tasks, such as the Track Mix feature on Deezer.

Chapter 4 In this chapter, we introduce the notion of consistency, a desirable property of embedding aggregators in the context of sequence representation. We provide an explicit mathematical formulation for the consistency of the average of i.i.d vectors of i.i.d dimensions, demonstrating that the theoretical consistency value of embeddings following this distribution is significantly higher than the one observed in embeddings obtained using SVD or ALS on song co-occurrence data.

Chapter 5 In this chapter, we introduce a novel exploration scheme called von Mises-Fisher exploration. We establish its connection to the popular method of Boltzmann exploration and demonstrate, both theoretically and through experiments on Deezer, that it can be better suited for exploring large action sets, such as music catalogs, than existing alternatives.

Chapter 6 This brief chapter presents directions for future research. In particular we introduce our formalization of the problem of continuously generating song recommendations for users as a Markov decision process where the action space is continuous.

Chapter 7 In this final chapter we summarize the work performed during this PhD.

The research conducted during this PhD resulted in the publication of several scientific articles, which are listed below. In the remainder of this thesis, when a chapter presents published results, the corresponding publication(s) will be specified at the beginning of that chapter.

International Conferences

- Léa Briand, Théo Bontempelli, **Walid Bendada**, Mathieu Morlon, François Rigaud, Benjamin Chapus, Thomas Bouabça, Guillaume Salha-Galvan. *Let's Get It Started: Fostering the Discoverability of New Releases on Deezer*. European Conference on Information Retrieval (**ECIR 2024**).
- **Walid Bendada**, Guillaume Salha-Galvan, Romain Hennequin, Thomas Bouabça, Tristan Cazenave. *On the Consistency of Average Embeddings for Item Recommendation*. Proceedings of the 17th ACM Conference on Recommender Systems (**RecSys 2023**).
- **Walid Bendada**, Théo Bontempelli, Mathieu Morlon, Benjamin Chapus, Thibault Cador, Thomas Bouabça, Guillaume Salha-Galvan. *Track mix generation on music streaming services using transformers*. Proceedings of the 17th ACM Conference on Recommender Systems (**RecSys 2023**).
- **Walid Bendada**, Guillaume Salha-Galvan, Thomas Bouabça, Tristan Cazenave. *A Scalable Framework for Automatic Playlist Continuation on Music Streaming Services*. Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (**SIGIR 2023**).
- Léa Briand, Guillaume Salha-Galvan, **Walid Bendada**, Mathieu Morlon, Viet-Anh Tran. *A semi-personalized system for user cold start recommendation on music streaming apps*. Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (**KDD 2021**).

Workshop

- **Walid Bendada**, Guillaume Salha-Galvan, Romain Hennequin, Théo Bontempelli, Thomas Bouabça, Tristan Cazenave. *vMF-exp: von Mises-Fisher Exploration of Large Action Sets with Hyperspherical Embeddings*. Workshop on Aligning Reinforcement Learning Experimentalists and Theorists, 41st International Conference on Machine Learning (**ICML 2024**).

Code and Data Releases

- Along with each of these articles, we publicly released our source code on a GitHub repository to promote reproducibility and facilitate future use of the proposed methods. For the article (Briand et al., 2021a), we also released a new dataset, which was directly extracted from Deezer's private resources after anonymizing users' private data. More details are provided throughout this thesis, as well as on <https://github.com/deezer>.

Chapter 2

Background

This chapter formally introduces key concepts related to music recommendation, detailing their usage and historical development, as well as the limitations that motivated our research. It does not aim to provide an exhaustive review, but instead focuses on the important notions that will be utilized throughout the remainder of this Ph.D. thesis.

2.1 Music Streaming Platforms

Music streaming platforms are online applications that provide interfaces across various electronic devices, notably mobile and web applications, allowing users to listen to music from catalogs that often exceed 100 million songs. The first streaming platforms emerged in the 2000s, and current services like Deezer, Spotify, and Apple Music have become the most popular means of accessing music. In Figure 2.1, we can see the result of a poll run by the International Federation of the Phonographic Industry, showing that the respondents spent the majority of the time listening to music in 2023 on music streaming platforms. Figure 2.2 gives the repartition of global market shares between the different music streaming platforms. Below, we introduce some relevant concepts to better understand these platforms.

2.1.1 Music Catalogs

Catalogs on streaming platforms are organized according to the standards of the music industry. Each audio file corresponding to a song is located on a page dedicated to the specific album it belongs to, and each album can be found on the page of the artists who released it, possibly involving several artists. This organization offers the dual advantage of providing users with an overview of an album's content (or an artist's discography) while allowing the application's audio player to stream the corresponding audio files consecutively without interruption, creating a seamless listening experience. The audio files for songs, album pages, and artist pages—referred

to for brevity as *songs*, *albums*, and *artists*—form a hierarchical typology of items that collectively define the catalog of a music streaming platform, as illustrated in Figure 1.1.

2.1.2 Access to Music

Given that music streaming platforms do not own the musical content they give access to, and that producing their own artists would position them as competitors of the labels whose catalogs they exploit, platforms can not rely on providing exclusive musical content to distinguish themselves from competitors. In fact, as mentioned above, the amount of songs available on each platform has become extremely large over the years, to a point that it has stopped being a selling argument. Instead, in order to earn additional shares in a market with a significant number of competitors (c.f Figure 2.2 (Mulligan, 2023)), platforms choose to invest in building better application features that help users quickly access the music they want to listen to and discover new musical content. Accessing music is mainly done through one of the following ways:

- Typing a query in the search bar, which prompts the internal search engine to match the user’s query with items in the catalog.
- Selecting items from the user’s personal library, which consists of items previously bookmarked and playlists created by the user.
- Listening to playlists curated by professionals.
- Listening to items suggested by the recommender system.

The first two methods mentioned above are labeled *lean forward* music listening, as they require users to actively decide which songs they want to hear and search for. In contrast, listening to suggested playlists—whether manually or algorithmically created—demands a less active attitude from users of the application, characterizing a *lean back* experience (Katz, 2022). Below, we provide a formal definition for playlists, a key element of music streaming platforms.

2.1.3 Playlists

Playlists are sequences of arbitrary songs displayed on a dedicated page, benefiting from the same advantages as album and artist pages. On most platforms, users can create playlists by providing an initial set of songs, along with optional elements such as a title, a textual description, or a cover image. Additional songs can be added to playlists later on.

Over the years, playlists have become an increasingly popular way to listen to music on streaming platforms. A 2016 market analysis conducted by the Music Business Association revealed that users of streaming platforms spend more time listening to playlists than to albums (Jakobsen, 2016). More recently, Spotify, the largest streaming platform in terms of worldwide subscribers, reported through their founder Daniel Ek that their users have collectively created more than 8 billion playlists (Ek, 2024).

In addition to user-created playlists, platforms also employ professional curators who specialize in music genres or moods to create playlists aimed at reaching the largest possible audience.

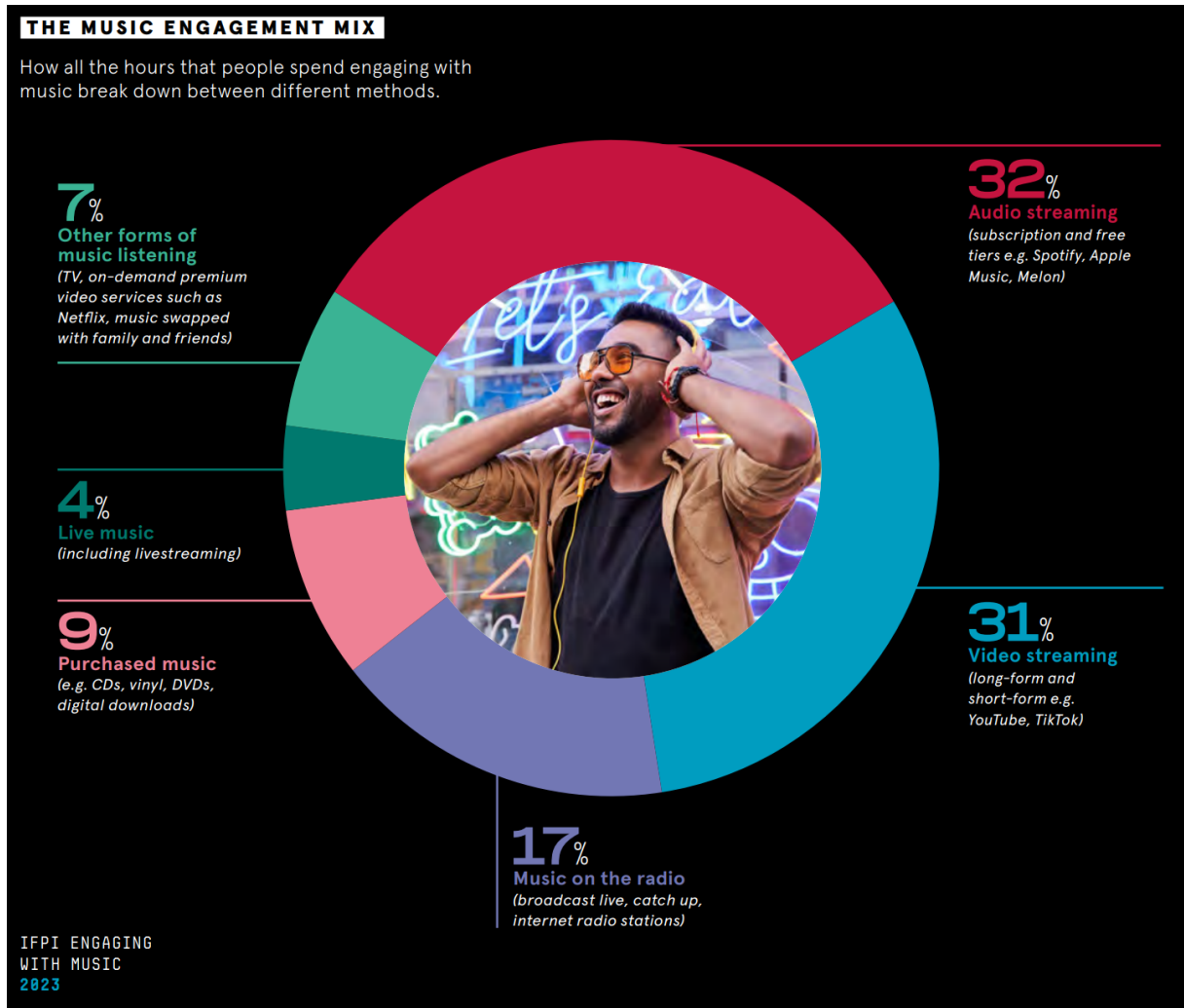


Figure 2.1: In 2023, people spent more time listening to music on streaming platforms than anywhere else (IFPI, 2023).

These editorialized playlists are typically more prominently featured to users, with the goal of demonstrating the platform’s added value compared to competitors. Since curated playlists are hand-crafted, they tend to be less numerous—usually only a few thousand on a given service. The industrialization of professional-quality playlist creation remains a significant business challenge. Despite advancements in algorithmic recommendations, human curators are still considered highly valuable for their ability to create engaging, hand-crafted playlists that resonate with users.

2.1.4 User Engagement

In the following, the word *items* may designate either one of songs, albums, artists and playlists.

The quality of individual features of a music streaming application is determined by their ability to satisfy a large number of users. However, user satisfaction is not directly observable. In practice, it can be qualitatively estimated by conducting surveys on a subset of users, although this approach is used sparingly due to its time-consuming and potentially costly nature. Alternatively,

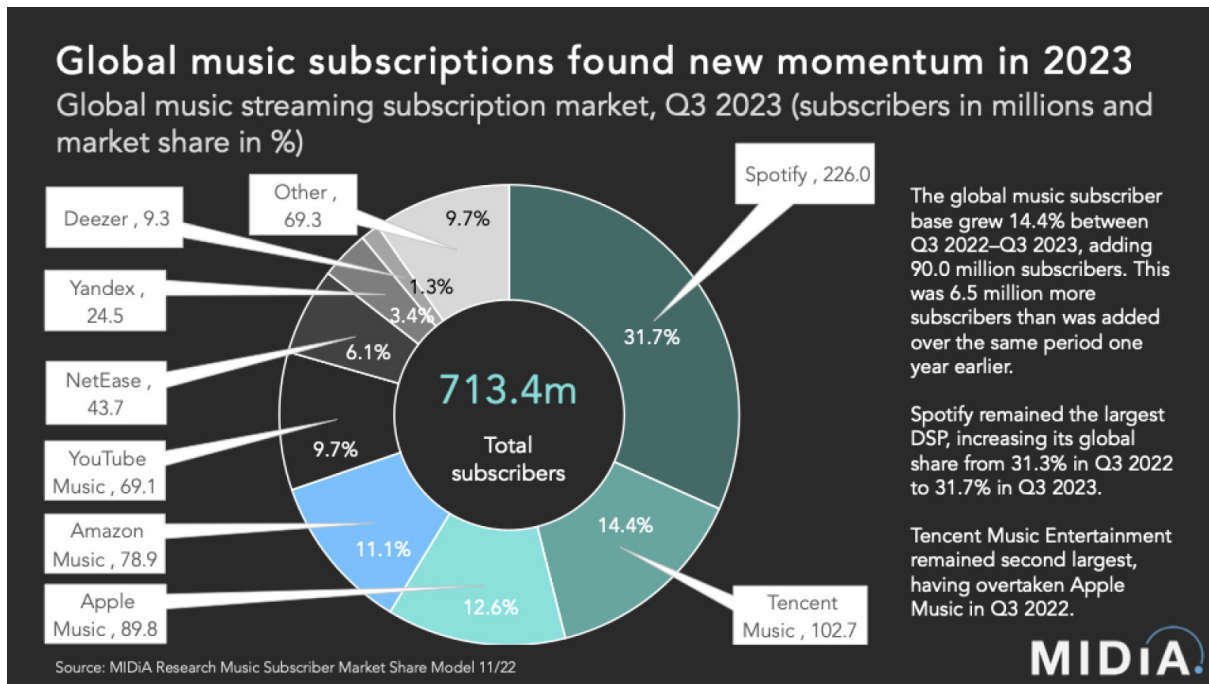


Figure 2.2: Market shares of the biggest music streaming platforms as of 2023 (Mulligan, 2023).

user satisfaction can be inferred by observing how and to what extent users engage with the recommended items. On Deezer, user engagement is monitored by very actions 2.1.

- *Stream*: the action of listening to a song for more than 30 seconds.
- *Skip*: the action of terminating a song before it reaches its end.
- *Collect*: the action of adding any item to the list of personal favorites or adding a song to a personal playlist.
- *Ban*: the action of requesting that a song or artist not be recommended again.

Collecting or banning an item are actions that can be interpreted respectively as positive and negative feedback. Because those actions leave seldom place for ambiguity, they are commonly called *explicit* feedback. Contrastively, streaming a song or skipping it, although leaning respectively towards positiveness and negativeness, are actions that can be somewhat more ambiguous. For instance, a user may not enjoy a song yet still stream it simply because they were away from their device when the song was recommended. Similarly, they may skip a song because the timing for listening to it isn't right although they still like it in general.

Table 2.1: A classification of user feedback types between implicit/explicit and positive/negative.

	Implicit	Explicit
Positive	stream	collect
Negative	skip	ban

In addition to monitoring engagement, user feedback is essential for building an effective recommender system.

2.2 Music Recommendation Tasks

Recommender systems (RS) are, given a user u of an online service, automated systems tasked with finding among a catalog of items $\mathcal{I}$ the ones that will bring satisfaction to the user. RS are leveraged when users are **indecisive** about the item they want to interact with, as opposed to for instance search engines that help users with a specific item in mind. RS are invoked when a user is interacting with a music application to perform a variety of tasks (Briand et al., 2021b; Jacobson et al., 2016; Schedl et al., 2018).

2.2.1 Homepage Personalization

The homepage of an application serves as the main landing screen where users first engage with the platform. Ideally, it should showcase the features and content that are most valuable to each user. Since this assessment is subjective—different users may have varying musical tastes and usage patterns—applications allow homepages to be customized for individual users, potentially resulting in as many unique homepage versions as there are users on the platform. In this context, the recommender system is responsible for determining which content to display for each user.

On Deezer and most music streaming platforms, the homepage features a grid layout with swipeable carousels stacked on top of one another, as illustrated in Figure 2.3. The recommender system must select the lists of items to be displayed in each carousel. Homepage personalization is a challenge encountered by recommender systems across various online services, and has been the focus of extensive work at Deezer in the past years (Bendada, Salha, et al., 2020).

2.2.2 Similar Items Display

For users browsing large catalogs containing hundreds of millions of items, discovering new music can be facilitated by displaying items that are in some way similar to those they already know. In this spirit, platforms commonly showcase a list of other items that are considered similar on each artist and album page. This task is typically managed by the recommender system, as its complexity makes manual handling by professional curators impractical. The artist page for The Beatles on Deezer can be found in Figure 2.4.

Unlike the homepage personalization problem, where the recommender system tailors results based on user information, the similar items displayed when a user visits an artist page is in general independent of user data. This highlights a distinction between **personalization** and **recommendation**. The similar items display task is one of the earliest challenges addressed by recommender systems (Sarwar et al., 2001), particularly in e-commerce, and motivated the development and implementation of efficient *Approximate Nearest Neighbors* (ANN) search solutions (M. Wang et al., 2021). The availability of ANN search systems serves as a central motivation for our work on real-time automatic playlist continuation presented in Chapter 3 and large-scale exploration discussed in Chapter 5.

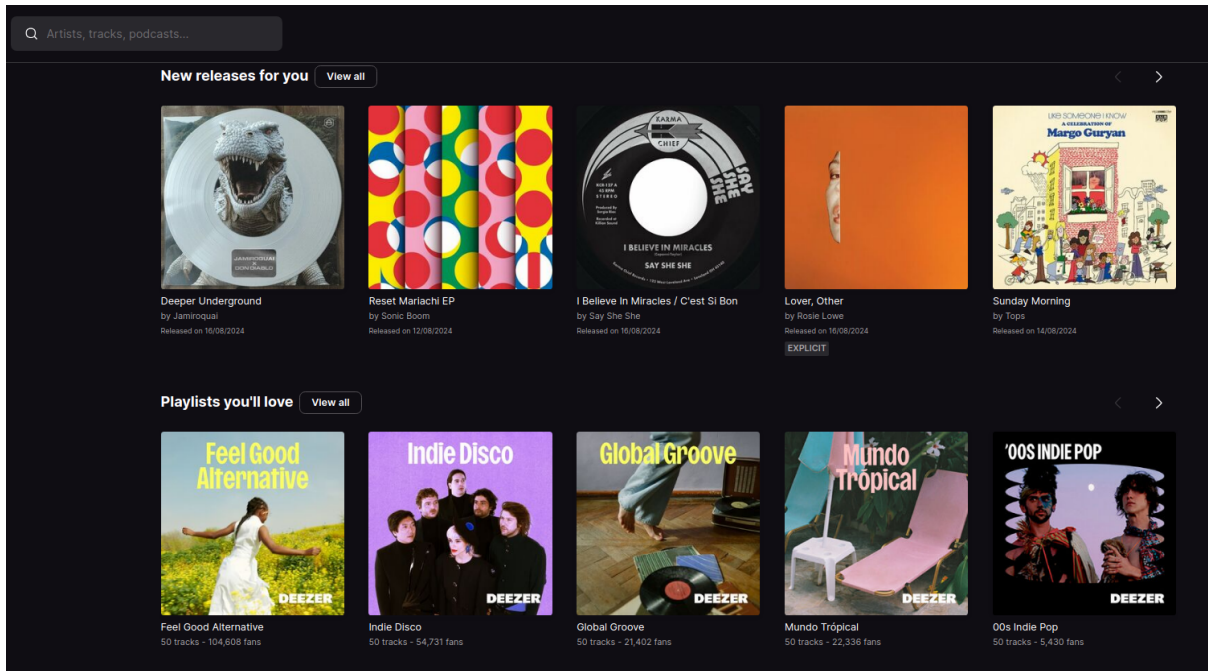


Figure 2.3: Homepage of Deezer displaying a carousel of albums and a carousel of playlists stacked on top of one another.

2.2.3 Automatic Playlist Continuation

As stated in the previous section, the ability for users to create playlists has allowed platforms to own a new kind of exclusive content, unlike music catalogs, which are the property of labels and can be found on all platforms. Moreover, when users create playlists and expand their music libraries, they become less likely to switch to a competitor. This is partly due to the anticipated burden of recreating their work, but also because they fear that the new recommender system will have less data to work with, resulting in less relevant suggestions. To encourage the creation and use of personal playlists, platforms typically offer features that suggest new songs for users to add to their playlists, or even create new personalized playlists from scratch. This task, called *automatic playlist continuation* (APC), is specific to music recommender systems. Unlike the homepage personalization or similar items display task, in this task, the recommended songs are meant to be listened to sequentially by the user, making poor recommendations more consequential than when only a single recommended item is listened to by the user. Another unique aspect is that this task must be performed in real-time, as the input information given to the recommender system (the current content of the user's playlist) is not available until the user starts their session.

For these reasons, automatically extending playlists while preserving their musical characteristics and aligning with user preferences remains a challenging task. (Schedl et al., 2018; C.-W. Chen et al., 2018; Zamani et al., 2019; Volkovs et al., 2018). Although there have been attempts to tackle APC that predate music streaming services (see, e.g., the survey of Bonnin et al. (2014)), the rise of playlist consumption on these platforms has led to APC being regarded as one of the "*most pressing current challenges in music recommender systems research*" by

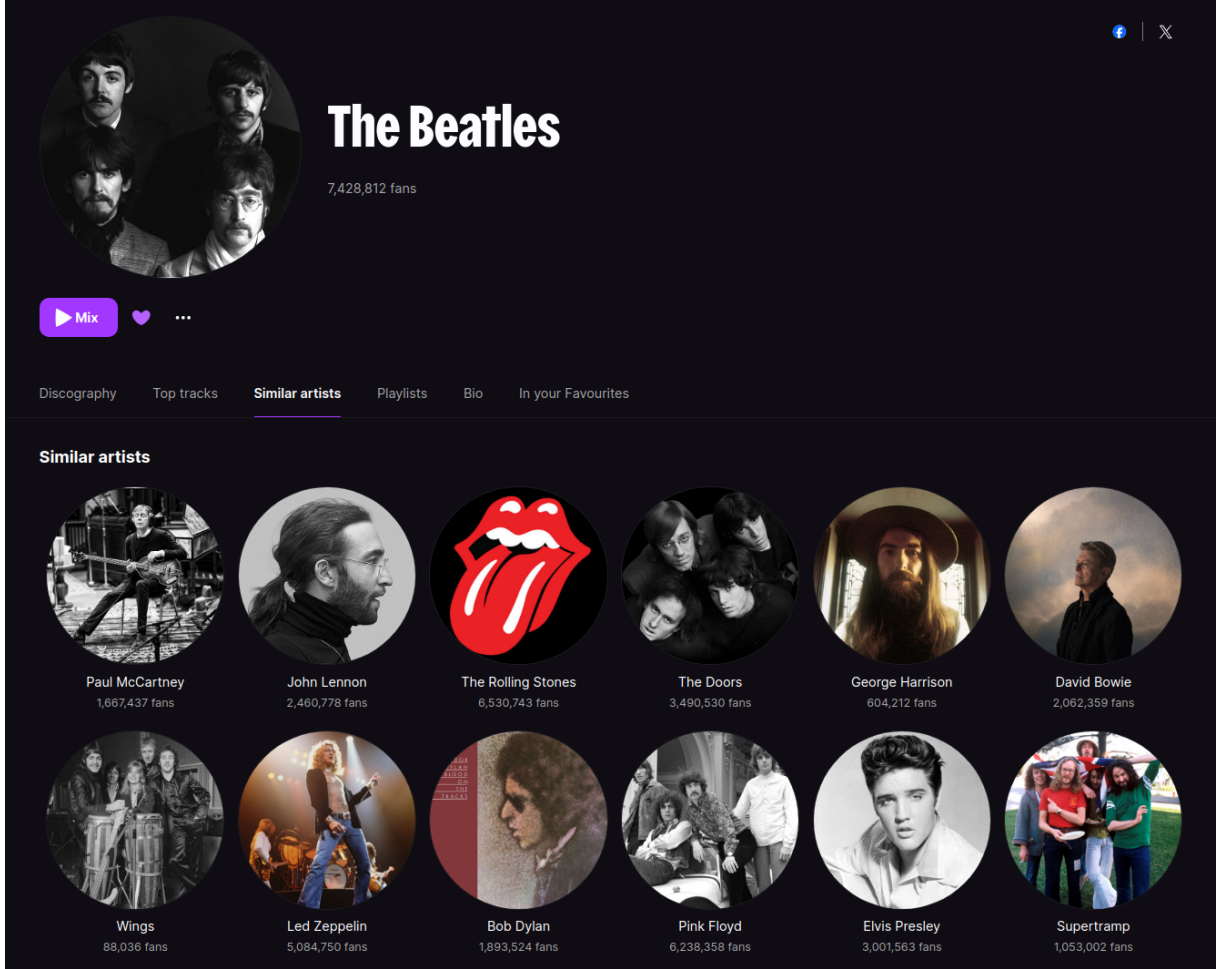


Figure 2.4: Artist page of *The Beatles* on Deezer displaying their similar artists.

Schedl et al. (2018). In Section 2.4.2 we provide an overview of recent approaches proposed to address this problem. In Chapter 3, we detail our framework for building APC models that ensure scalability by design. We also demonstrate how solving the APC task can contribute to building other features where songs are recommended sequentially within a session, such as the Track Mix feature on Deezer.

We have outlined some of the key tasks involving the recommender system of a music streaming platform. Next, we introduce the primary *Machine Learning* (ML) problem that recommendation models are trained to solve, and explain how these trained models can be effectively utilized for recommendations in a music streaming app.

2.3 Traditional Recommendation Models

Given a context and task, recommender systems rely on internal functions, called *models*, that estimate the satisfaction that an item i from a catalog $\mathcal{I}$ will provide to a user u . When such a function $\hat{f}$ is available, it can be used to recommend an item to user u as follows:

- $\forall i \in \mathcal{I}$, estimate $\hat{f}(u, i)$

- Return $i^* = \arg \max_{i \in \mathcal{I}} \hat{f}(u, i)$

The representations of u and i leveraged by the model $\hat{f}$, as well as the nature of the item set $\mathcal{I}$, depend on the context and task. For example, when optimizing the homepage for a user who has just logged in, it can be difficult to anticipate their current intent due to the absence of recent activity. In such cases, it may be useful to leverage any available information previously accumulated about the user to make predictions. In contrast, when a user is browsing a specific artist’s page, focusing solely on that artist and disregarding personal user information might be reasonable for recommending similar artists.

To develop models $\hat{f}$ that generalize across a wide range of recommendation tasks, the research community has extensively studied a surrogate problem known as the ratings prediction task.

2.3.1 Ratings Prediction Task

The ratings prediction task is a problem studied across recommender systems of all types of items, which has received increasing attention following an international competition known as *Netflix Prize* started in 2007 (Bennett et al., 2007). The goal of this task is to predict the numerical rating $r_{u,i}$ that a user u would assign to an item i (such as a movie, product, or book) based on historical data. In the case of music recommendation, the rating to predict can be replaced by an aggregated engagement score, as described in Section 2.1.4. This aggregation results in a dataset $\mathcal{D}$ consisting of triplets $(u, i, r_{u,i})$, which can be interpreted as a sparse matrix where rows correspond to users, columns correspond to items, and each entry in the matrix is the engagement score of a user towards an item (assuming the user has interacted with the item). The objective is to predict the missing ratings in this matrix as accurately as possible. Evaluation is performed on a test set $\mathcal{D}_{test}$ consisting of hidden ratings, using regression metrics such as the *Root Mean Square Error* (RMSE):

$$\text{RMSE}(\hat{f}) = \sqrt{\sum_{(u,i,r_{u,i}) \in \mathcal{D}_{test}} (r_{ui} - \hat{f}(u, i))^2} \quad (2.1)$$

Lower values of RMSE indicate better predictive performance.

Among models studied in the literature on ratings prediction, three types of approaches are of particular interest for our work.

2.3.2 Common Approaches

2.3.2.1 Neighborhood-Based Methods

Neighborhood-based methods for ratings prediction estimate a user’s rating for an item by leveraging the ratings provided by similar users or similar items (Su et al., 2009; Koren and Bell, 2015). In user-based approaches, the predicted rating is derived from the ratings given by users who share similar preferences with the target user. In item-based approaches, the prediction is based on the ratings of items similar to the one being considered, and that the target user has already rated. These methods rely on similarity measures, such as cosine similarity or Pearson

correlation, that are computed within the space defined by the row vectors or column vectors of the sparse interaction matrix. While neighborhood-based methods are straightforward and effective when sufficient similar users or items are present in the dataset, they can encounter difficulties with sparsity and scalability in large datasets.

2.3.2.2 Matrix Factorization Models

Matrix factorization models for ratings prediction decompose the user-item rating matrix into lower-dimensional matrices whose respective column vectors act as latent factors representing users and items (Koren, Bell, and Volinsky, 2009; Su et al., 2009; Koren and Bell, 2015; He, H. Zhang, et al., 2016). These latent factors capture underlying patterns, such as user preferences and item attributes, which are not directly observable. The mathematical assumption underlying these models is that the sparse user-item matrix has a rank much smaller than its dimensions. This implies that the column vectors, each representing the audience of an item, are not linearly independent. An illustration of the relevance of this assumption is, for instance, the fact that when predicting engagement scores for a set of artists, knowing that someone is a fan of the band The Beatles provides useful information for predicting their likelihood of being fans of the individual band members that are Paul McCartney, John Lennon, George Harrison, or Ringo Starr.

The predicted rating is obtained by taking the inner product of the corresponding user and item latent vectors. Models like singular value decomposition (SVD) (Paterek, 2007) and alternating least squares (ALS) (Koren, Bell, and Volinsky, 2009) are commonly employed for this purpose. We detail in Section 2.3.4 how the Deezer recommendation system creates embeddings for a large amount of songs in its catalog.

Matrix factorization models are effective at handling large datasets and generalizing beyond the observed ratings. However, their main shortcomings are their inability to make predictions for cold users or items, i.e those that appear infrequently in the training set, and the impossibility to account for non-linear dependencies between latent factors and predicted ratings.

2.3.2.3 Deep Learning Models

Deep learning models for ratings prediction leverage neural networks to capture complex, non-linear patterns and interactions between users and items (H. Wang et al., 2015; Covington et al., 2016a; He, Liao, et al., 2017b; Mongia et al., 2020). Unlike matrix factorization models, deep learning approaches, such as *Neural Collaborative Filtering* (NCF) (He, Liao, et al., 2017b) and autoencoders, utilize multiple layers to learn intricate relationships and high-level features. While matrix factorization focuses on linear interactions between users and items, deep learning can model more complex dependencies and feature interactions. However, deep learning models often require more data and computational power to train effectively compared to matrix factorization methods, which are generally more interpretable and less resource-intensive. More importantly, the possibility to efficiently find the item that maximizes the output prediction of a neural network, which as reminded earlier in this section is at the core of real-time recommendation, highly depends on the specific architecture of the neural network.

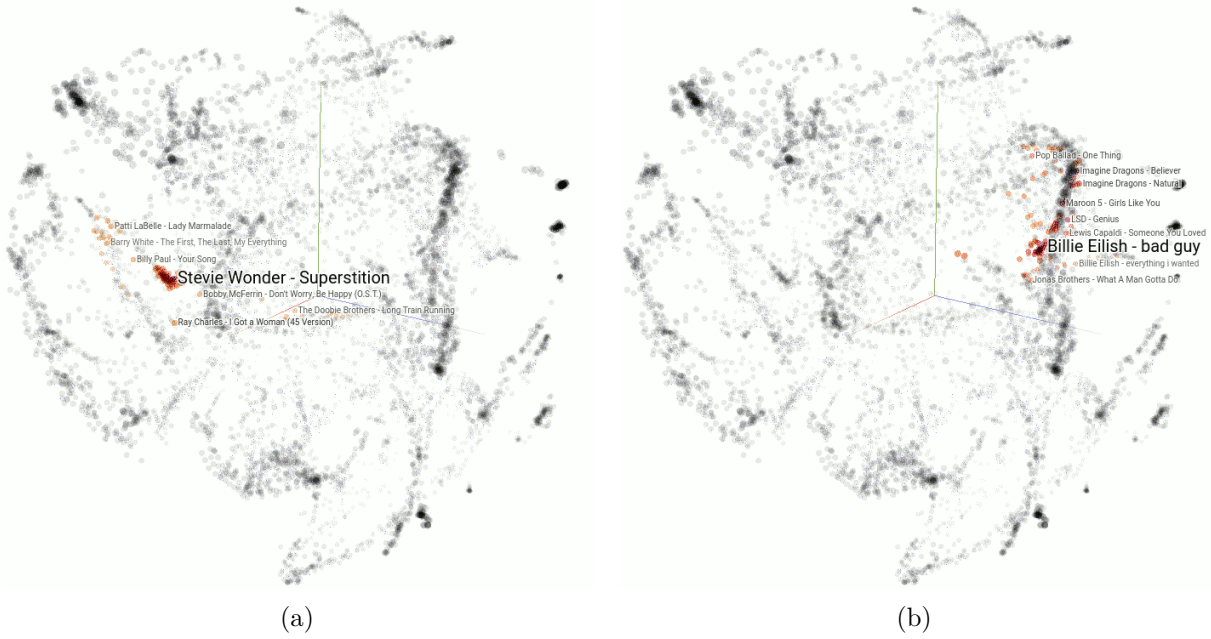


Figure 2.5: 3d T-SNE visualization of the set of song embeddings used the Deezer app. The songs (a) *Superstition* by Stevie Wonder and (b) *Bad Guy* by Billie Eilish are identified along with their nearest neighbors.

2.3.2.4 Collaborative Filtering vs Content-based

Neighborhood-based methods, matrix factorization models, and deep learning models are all examples of *Collaborative Filtering* (CF) methods (Su et al., 2009; Koren and Bell, 2015), which aim to predict user preferences based on historical interactions between users and items. In contrast, content-based models predict ratings based on the features and attributes of the item—such as musical genre or audio characteristics like key and tempo—and the user’s past preferences for similar attributes. In this thesis, while we primarily focus on collaborative filtering models, we also provide an example of a model trained for automatic playlist continuation that enhances performance by incorporating content information.

2.3.2.5 Mathematical Interpretation

Although seemingly different in their approaches, the families of models mentioned above share the common assumption that there exists a function h mapping items from $\mathcal{I}$ into a space $\mathcal{E}$ equipped with a norm $\|\cdot\|$, and that for each user u there exists a function $\hat{f}_u : \mathcal{E} \rightarrow \mathbb{R}$ such that

$$\hat{f}(u, i) = \hat{f}_u(h(i)) \quad (2.2)$$

and $\hat{f}_u$ is **Lipschitz continuous**.

In other words, models assume that there exists a space $\mathcal{E}$ where items can be represented, such that if two items have close images in $\mathcal{E}$, we can expect users to rate them similarly. A major challenge is to find a transformation h that can map elements of $\mathcal{I}$ to their image in $\mathcal{E}$. To narrow

down the search for possible transformations, content-based methods add the constraint that h should map an item to its audio features, for instance, while collaborative-filtering methods use mapping functions based on item co-occurrences.

On top of being informative about the similarities between content-based and CF approaches, the decomposition of Equation (2.2) also outlines how learning representations for items is a major aspect of model training. This observation is central to our framework on scalable automatic playlist continuation developed in Chapter 3.

2.3.3 Inference

As stated above, once a model $\hat{f}$ has been trained, it can be used to recommend an item to a user u by computing the argmax of $\hat{f}(u, \cdot)$ over the set of recommendable items $\mathcal{I}$. Depending on the nature of $\hat{f}$ and the cardinality of $\mathcal{I}$, this operation can be very time-consuming, clashing with the real-time imperatives of user-oriented applications. From an infrastructure perspective, several architectural choices can be considered to circumvent this issue.

2.3.3.1 Online vs Offline Operations

Offline operations involve tasks that are precomputed in advance before a user starts interacting with the app, while online operations are real-time processes that respond instantly to user interactions and new data inputs. Apps try to minimize the number of online operations to reduce system latency and ensure a seamless user experience. This is possible when the computation of an operation can be done without having to wait for user interaction with the app. For instance, similar artists on Deezer can be computed in advance and stored in a database from which reading is a constant time operation. Unfortunately, other recommendation tasks, such as personalizing homepages, which must be updated as users return to the homepage during their journey, or expanding a playlist that has just been created, rely on recent user interactions as input and cannot be performed offline. This is a major problem in music recommendation since the set $\mathcal{I}$ from which to choose commonly spans millions of items.

2.3.3.2 Approximate Nearest Neighbors

Within a catalog, one can expect to find more than one item capable of satisfying a user. This means that instead of recommending the item i^* maximizing $\hat{f}$, in practice an item $\tilde{i}$ with a high enough value of $\hat{f}(u, \tilde{i})$ can be selected. To this end, ANN systems that can quickly find similar vectors within large sets of embeddings by trading off a small amount of accuracy for significant speed improvements are used (Li et al., 2019). Specialized indexing techniques such as *Locality Sensitive Hashing* (LSH) (Indyk et al., 1998) or Product Quantization (Jegou et al., 2010) are employed to enable fast retrieval even in item sets of billions of elements. Some limitations for ANN search systems are:

- They do not guarantee that the item retrieved is the most similar (which, as stated above, can up to a margin be acceptable for music recommendation).

- They only evaluate similarity with a subset of items, making it difficult to recommend items that aren't considered similar by the current model but might actually bring higher satisfaction than expected. In Chapter 5, we introduce a new exploration method to mitigate this limitation.
- They require that items be represented as single vectors, which constrains the class of models that can be used for real-time music recommendation.

However, they presently constitute an essential component of real-world recommender systems.

2.3.3.3 Two-Stage Approach

The two-stage approach is a popular architecture of real-world recommender systems (Christakopoulou et al., 2018; Huang et al., 2021) that aims to overcome the third limitation of ANN search systems listed above. It involves a first stage where a broad set of potential recommendations is generated using simple models that are compatible with ANN, and a second stage where this subset is refined and ranked using more sophisticated models, such as deep neural networks. The claimed advantage of this approach is to combine speed with accuracy, ensuring both efficient candidate generation and precise final recommendations.

However, it has the downside of requiring the training, deployment, and maintenance of several models for production, resulting in a system with higher maintenance imperatives than single-stage approaches.

Moreover, if the first model filters out candidates that could have brought the most satisfaction to users, then recommendations will be sub-optimal regardless of the complexity and effort put into training the second model, limiting its impact. We argue that the impact of the second, sophisticated model, becomes even lower for very engaged users who frequently reconnect to a music application.

Indeed, recommendations need to be refreshed regularly, as the perceived value of an item tends to eventually decrease with repeated exposure (Sguerra et al., 2023). To prevent user fatigue, systems often implement explicit rules so that an item cannot be recommended several times to a user within a given time frame. However, simpler models with linear predictions, like matrix factorization, have constant (often low-valued) first derivatives. This means that they nearly output the same value when the input doesn't change much (c.f Section 2.3.2.5 above). Concretely, there is a risk that the list of candidates generated for a user remains nearly identical if the user's profile has not changed much between two sessions. Consequently, the pool of effective candidates, i.e those selected in the first stage but not already recommended in previous sessions, available to the second model is expected to only get smaller. In the worst-case scenario of a power-user frequently using the app and an overly simple first-stage model, the role of the (possibly sophisticated and resource-demanding) second model would basically consist of permuting a frozen list of candidates over time.

Although convenient in the general case, we believe that for music recommendation—where a significant portion of users of streaming applications log in every day—relying on a two-stage approach with a simple first model and a heavily tuned second model could prove detrimental

users long-term experience. In Chapter 3, we introduce a class of models that are compatible with ANN search, encompassing both simple matrix factorization as well as some deep neural network architectures, and show how it can be applied to real-time automatic playlist continuation.

2.3.4 Embedding-Based Recommendation on Deezer

The recommendation system of Deezer has historically relied on latent representations for a significant subset of the catalog. Several computation methods have been considered and compared with regard to their ability to perform well on downstream recommendations tasks. Throughout this thesis, we will present experiments run on the production system of Deezer, which currently relies on song embeddings computed on a weekly basis with the following steps:

- Compiling a dataset of lists of tracks, either from user-made playlists or users' lists of favorite songs.
- Computing co-occurrences of songs and storing them in a square sparse matrix M .
- Transforming M so that each entry M_{ij} is replaced by the *Pointwise Mutual Information* (PMI) (Fano, 1949) between the songs i and j if the PMI is positive, else M_{ij} is set to 0.
- Performing a truncated singular value decomposition (S. Banerjee et al., 2014) of the matrix M .
- Storing U_d , the truncated matrix of the first d left singular vectors of M , after normalizing the rows of U_d so that they all consist in unit-norm vectors. Each row vector of U_d is the latent representation of a song.

We detail below the motivations for the last 3 steps of the process.

2.3.4.1 Pointwise Mutual Information

PMI helps account for discrepancies in popularity by adjusting for the likelihood that two songs co-occur simply because they are individually popular, rather than being meaningfully related. The raw co-occurrence counts between songs i and j tend to be biased by how frequently each song appears in the dataset. Popular songs will naturally have higher co-occurrence counts with other songs, even if there is no strong connection between them. PMI corrects for this by normalizing the co-occurrence of i and j relative to their independent frequencies. Formally, PMI is defined as:

$$\text{PMI}(i, j) = \log \left(\frac{P(i, j)}{P(i)P(j)} \right) \quad (2.3)$$

Here, $P(i, j)$ is the probability of songs i and j co-occurring, and $P(i)$ and $P(j)$ are the marginal probabilities of songs i and j appearing on their own. By dividing the joint probability $P(i, j)$ by the product of the independent probabilities $P(i)$ and $P(j)$, PMI removes the influence of individual song popularity. A positive PMI value thus indicates that the songs co-occur more frequently than would be expected by chance based on their popularity.

2.3.4.2 Truncated Singular Value Decomposition

The resulting square PMI matrix M is factorized using truncated singular value Decomposition which, given a value d , finds the matrices U_d and V_d whose column vectors are respectively the first d left singular vector of M and the first d right singular vector of M , as well as the diagonal matrix Σ_d containing the first d singular values of M . The motivation for performing truncated SVD is that the matrix:

$$M_d = U_d \Sigma_d V_d^T \quad (2.4)$$

verifies the property:

$$\|M - M_d\|_F = \min_{\text{rank}(A) \leq d} \|M - A\|_F \quad (2.5)$$

with $\|\cdot\|_F$ the Frobenius norm. In other words, M_d is the best matrix of rank d to reconstruct the PMI matrix M . For two songs i and j , the element of M_d at row i and column j , noted $M_{d,ij}$, should thus be close to the PMI of i and j . However since M_d is by construction of lower rank than M , it will not be able to perfectly reconstruct the sparse PMI matrix, which means that many elements of M that have a value of zero will be approximated by an element of M_d with a value different than zero, thus surfacing similarities between songs that were not directly observed in the PMI matrix.

2.3.4.3 Storing the Normalized Left Singular Vector

Conveniently, to compute the values of M_d , the system only needs to store the left matrix U_d instead of the three matrices U_d , Σ_d and V_d . This is possible because:

- The matrix M is symmetric $\implies U_d = V_d$.
- The first singular values decrease slowly $\implies \Sigma_d \approx \sigma I_d$,

with σ a scalar and I_d the identity matrix. For a pair of songs i and j , $M_{d,ij}$ can thus be approximately computed using the row vectors of U :

$$M_{d,ij} \approx \sigma U_{d,i}^T U_{d,j}. \quad (2.6)$$

Moreover, studies on embedding representations have shown that for item recommendations, the information encoded by the norm of each vector is related to the popularity of the item (Afchar et al., 2023). Indeed, popular items still have more non-zero entries in the PMI matrix than regular items, which incentives the embeddings of popular items to increase their norm so as to have a high inner product similarity with vectors located on the same half space.

To mitigate the influence of popularity in applications where nearest neighbors must be retrieved with inner product, Deezer thus chooses to store normalized embeddings. The information of item popularity can still be independently stored in case it is needed by the recommender system.

Song embeddings can then be used to build representations for other types of items using the hierarchical relationships of music catalogs highlighted in Figure 1.1 of Chapter 1. Song embeddings can also be used to build embeddings of users whose library they belong to.

2.3.5 Shortcomings of the Ratings Prediction Task

Models trained for ratings prediction are useful on music recommendation tasks where a user will likely interact with only one item, even if several of them have been suggested. This is the case in the homepage personalization task; as illustrated in Figure 2.3, homepages of music streaming platforms choose to display mostly playlists, albums, or artists. Each of these items offers an entire autonomous listening session to the user. As a consequence, such items are seldom listened to one after another, just like movies on video platforms are rarely watched back-to-back, and there is thus low incentive to consider the coherence of the set of items being recommended on a homepage. By contrast, when performing automatic playlist continuation, each recommended item is a song with a duration of only a few minutes. By sequentially recommending songs, a complete listening session can be created for a user. Playlist continuation requires estimating the likelihood that an **entire sequence of tracks** satisfies a user, and ratings prediction models do not explicitly account for sequence. As a result, these models may not effectively capture the dynamic aspects of playlist composition, making them less suited for this task, which is overwhelmingly frequent on music streaming applications. For this reason, in recent years, following the astonishing results observed in other machine learning domains such as *Natural Language Processing* (NLP), focus in music recommendation has shifted to alternative ML tasks, such as sequence modelling.

2.4 Sequential Recommendation

Differing from the ratings prediction task, the sequential recommendation problem (S. Wang et al., 2019; Dang et al., 2020) involves predicting the next item in a sequence based on the previous items, explicitly assuming that the short-term historical interactions of a user contain the necessary information to successfully make the desired predictions (Vasile et al., 2016; D. Wang, Deng, X. Zhang, et al., 2016; Donkers et al., 2017; L. Guo et al., 2019; Yuan et al., 2019). It draws inspiration from the more general ML problem of sequence modeling.

2.4.1 Sequence Modeling

Sequence modeling (Goodfellow et al., 2016) involves learning patterns in ordered data where the relationships between elements are crucial for understanding and prediction. The goal is to capture the dependencies between elements, called tokens, so that, given a sequence of previous tokens, the model can predict the next one. For instance, in NLP, sentences are sequences of words, and the most popular example of a sequence modeling task is the language modeling task, where the objective is to predict the next word in a sentence based on the preceding words.

Formally, given a sequence of tokens $x_1, x_2, \dots, x_T$, sequence modeling aims at capturing the likelihood of observing the entire sequence $P(x_1, x_2, \dots, x_T)$. Given that the ordering of

the tokens is a defining aspect of the sequence, this probability is equal to the product of the conditional probabilities of each token given the previous tokens. This can be expressed as:

$$P(x_1, x_2, \dots, x_T) = P(x_1) \cdot P(x_2 | x_1) \cdot P(x_3 | x_1, x_2) \cdot \dots \cdot P(x_T | x_1, x_2, \dots, x_{T-1}) \quad (2.7)$$

In recent years, deep learning has been notoriously successful at this probability estimation task by leveraging specialized families of architectures like *Recurrent Neural Networks* (RNN), among which the families of *Long Short-Term Memory* (LSTM) (Graves et al., 2012) and *Gated Recurrent Unit* (GRU) (Chung et al., 2014) models, and more recently transformers (Vaswani et al., 2017). Following this success, research on recommender systems has studied the analogous task of next-item recommendation, with positive results using models based on transformer architectures (F. Sun et al., 2019; W.-C. Kang et al., 2018). Of particular interest is the analogy between words within a sentence and songs within a playlist, which has motivated the community of recommender systems to consider automatic playlist continuation as a sequence modeling task.

2.4.2 Automatic Playlist Continuation as Sequential Recommendation

This section further details recent attempts at the task of APC from the research community.

2.4.2.1 Limits of Collaborative Filtering

Historically, collaborative filtering has been a prevalent approach for APC (Faggioli et al., 2018; Ludewig, Kamehkhosh, et al., 2018; Volkovs et al., 2018; Schedl et al., 2018). As illustrated in Figure 2.6a, CF methods for APC usually represent playlist datasets as sparse matrices where each row corresponds to a playlist, each column corresponds to a song, and the associated binary value indicates whether the song appears in the playlist. In essence, they aim to adopt the same paradigm as the ratings prediction problem, where users are replaced with individual playlists and ratings are binary, encoding the information of which song occurs in which playlist. Consequently, collaborative filtering methods based on neighborhood (Faggioli et al., 2018; Volkovs et al., 2018) or matrix factorization (Koren, Bell, and Volinsky, 2009; Ludewig, Kamehkhosh, et al., 2018) can be adapted to the APC problem. As these CF methods usually fail to provide recommendations for songs absent from existing playlists (Rendle, 2010; Gope et al., 2017), researchers have also proposed to integrate metadata information into predictions, with the underlying assumption that co-occurrence information of frequent songs should flow to rare songs sharing similar attributes. A notable example of a family of CF models integrating metadata is *Factorization Machines* (FM) (Rendle, 2010). FM represent metadata in the same embedding space as playlists and songs and add inner products between embedding vectors of various metadata to the original prediction score. They have been proposed for APC (Ludewig, Kamehkhosh, et al., 2018; Yang et al., 2018; Rubtsov et al., 2018; Ferraro, Bogdanov, et al., 2018). However, more recent research on APC has extensively focused on leveraging models successful in sequence-based and session-based recommendation.

Table 2.2: Public datasets for APC.

Dataset	Songs	Playlists	Interactions
AOTM (McFee et al., 2012)	91 166	27 005	306 830
NOWPLAYING (Zangerle et al., 2014)	75 169	75 169	271 177
30MUSIC (Turrin et al., 2015)	210 633	37 333	638 933
MELON (Ferraro, Y. Kim, et al., 2021)	649 091	148 826	5 904 718
MPD (C.-W. Chen et al., 2018)	2 262 292	1 000 000	66 346 428

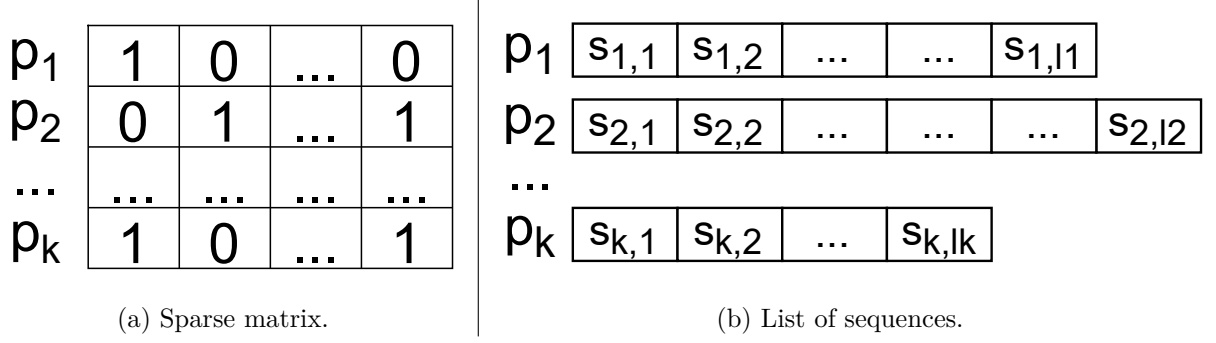


Figure 2.6: An APC dataset represented as (a) a sparse playlist-song occurrence matrix, (b) a list of uneven song sequences.

2.4.2.2 Sequential approach

Drawing on the analogy that exists between words within sentences on the one hand, and songs within playlists on the other hand, an increasing number of research works on APC have proposed to represent playlist datasets as uneven lists of song sequences, as illustrated in Figure 2.6b. These studies have adapted techniques already successful in other sequence modeling tasks, notably language modeling, to extend the ordered sequences of songs that playlists naturally constitute. More specifically, they have introduced APC models based on word2vec (Vasile et al., 2016; D. Wang, Deng, X. Zhang, et al., 2016), recurrent neural networks (Jing et al., 2017; Donkers et al., 2017), convolutional neural networks (Yuan et al., 2019), and attention mechanisms (L. Guo et al., 2019).

In the experimental evaluations of these studies, such sequential methods often outperform the pure CF approaches mentioned above. However, these promising results were obtained on relatively small subsets¹ of real-world datasets (see, e.g., the first four lines of Table 2.2) due to the absence of larger public real-world datasets for an extended period. Hence, their practical effectiveness in large-scale applications involving millions of songs and playlists still needs to be fully demonstrated. Fortunately, studying the behavior of APC models became possible following the Spotify Challenge and the release of the *Million Playlist Dataset* (MPD).

¹In particular, the range of recommendable songs was often restricted to the most popular ones. Besides scalability concerns, such a restriction also questions the ability of these models to recommend songs from the *long tail*, a matter of great importance considering that the size of the catalog is one of the specificities of music recommendation (Schedl et al., 2018) and that numerous models tend to be biased toward popular artists (Kowald et al., 2020).

2.4.2.3 Spotify Challenge

To foster research on large-scale APC, the 12th ACM Conference on Recommender Systems hosted the "RecSys 2018 APC Challenge" (C.-W. Chen et al., 2018), evaluated on the Million Playlist Dataset (MPD), which was subsequently released to the public. Composed of one million playlists created by Spotify users, the MPD stands out for its size compared to other public datasets listed in Table 2.2, as well as for its sparsity. Indeed, while the dataset contains more than two million songs, roughly 60% of them appear fewer than twice within playlists (C.-W. Chen et al., 2018). Spotify also provided supplementary information on songs and playlists, which often helps mitigate the cold start problem induced by data sparsity (Briand et al., 2021b; Salha-Galvan, Hennequin, et al., 2021). In 2019, Zamani et al. (2019) analyzed key insights from the challenge, noting that many teams leveraged:

- Ensemble architectures: several teams combined multiple APC models to provide different "candidate" songs for extending playlists.
- Two-stage architectures, as defined in Section 2.3.3.3.

Perhaps surprisingly, the modern sequence modeling neural architectures mentioned in the previous section were seldom used for candidate retrieval in the solutions proposed for the Spotify Challenge (Zamani et al., 2019).

2.4.3 Limitations to Real-World Usage

Scalability is essential for industrial applications in music streaming services, which must select the best songs to recommend for automatic playlist continuation (APC) from large catalogs containing tens of millions of candidates. In addition to the number of songs to manage during the training of the APC model, the scalability of inference is also crucial, even though it is sometimes overlooked. As mentioned earlier, music streaming services typically perform APC online, repeatedly, and in real-time. Users regularly create new playlists and modify existing ones, and they expect the system to provide updated recommendations with minimal delay.

There is, however, a discrepancy between the complexity and evaluation procedures of APC models proposed in the recent scientific literature and the scalability requirements associated with industrial applications. As an illustration of this discrepancy, we stress again that modern sequence modeling techniques were seldom used during the RecSys 2018 APC Challenge (Jannach et al., 2020; Zamani et al., 2019). When considering either candidate selection models in ensemble/two-stage architectures, or single-stage architectures that directly score all songs, most approaches relied on simpler and faster non-parametric nearest neighbor models (i.e not relying on neural networks), applied to various CF-based song/playlist embedding representations learned using models from Section 2.4.2.1.

In summary, most teams favored simpler and more scalable methods over those presented in Section 2.4.2.2, despite their promising performances on smaller datasets. We believe this challenge highlighted the current gap between the complexity and evaluation procedures of APC models proposed in recent scientific literature and the simpler models successfully used

in large-scale applications. This observation reopens the question of whether recommender systems actually benefit from the extensive literature on neural-based recommendation, a recurring trope in the recent literature of RS (Ferrari Dacrema et al., 2019; Rendle et al., 2020; Ludewig, Mauro, et al., 2021). Regarding the most recent class of sequence modeling neural architectures, i.e., Transformers (Vaswani et al., 2017), they have been proposed for sequential product recommendation (see, e.g., SASRec and BERT4Rec (W.-C. Kang et al., 2018; F. Sun et al., 2019)), but at the beginning of this thesis, their usage in industrial, large-scale APC had not been considered.

Chapter 3

A Scalable Framework for Automatic Playlist Continuation

This chapter presents research conducted with Guillaume Salha-Galvan, Thomas Bouabça, and Tristan Cazenave, and published in the proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2023) (Bendada, Salha-Galvan, Bouabça, et al., 2023). Also the last section summarizes the work presented in Bendada, Bontempelli, et al. (2023) at (RecSys 2023).

3.1 Introduction

This chapter is dedicated to the problem of Automatic Playlist Continuation (APC) in the context of a music streaming platform. For background on this topic, refer to Section 2.2 and Section 2.4 in Chapter 2.

In the present chapter, we detail the following contributions:

- We introduce a principled framework to build scalable yet effective APC models. Based on a *represent-then-aggregate* strategy, it permits incorporating a wide range of complex APC models into large-scale systems suitable for industrial applications. Our framework systematically decomposes these models into a part handling song representation learning, and a part dedicated to playlist-level sequence modeling.
- We illustrate the possibilities induced by our framework, showing how one can design scalable APC models benefiting from the most popular architectures for sequence modeling, e.g., recurrent neural networks (Hidasi et al., 2016) or Transformers (Vaswani et al., 2017), combined with complex song representation learning models, e.g., neural architectures processing metadata (Rendle, 2010).

- We demonstrate the empirical relevance of our framework for large-scale APC through in-depth experimental validation on Spotify’s Million Playlist Dataset (MPD), the largest public APC dataset. We publicly release our source code on GitHub to ensure the reproducibility of our results.
- We also describe how, in 2022, we leveraged this framework to improve APC in production on Deezer. We present results from a large-scale online A/B test on users from this service, emphasizing the practical impact of our framework in such a real-world application.
- Finally, we describe an additional live experiment ran on Deezer in 2023 to assess the usability and performances of a represent-then-aggregate model trained for APC on the related task of *Track Mix* generation, where models must create an infinite mix starting from a single track.

3.2 Problem Formulation

We first introduce a formalization of the problem of automatic playlist continuation.

3.2.1 Notation

In this chapter we consider a sub-catalog $\mathcal{S} = \{s_1, \dots, s_N\}$ of $N \in \mathbb{N}^*$ songs available on a music streaming service. Users from this service can create playlists containing these songs, as in Figure 3.1. They can update their existing playlists at any time, by adding, removing, or reordering songs.

We denote by $L \in \mathbb{N}^*$ the maximum length for a playlist, a parameter fixed by the service. We denote by $\mathcal{P}$ the set of all playlists that can be created from $\mathcal{S}$:

$$\mathcal{P} = \bigcup_{l=1}^L \mathcal{S}^l. \quad (3.1)$$

Lastly, we associate each song $s \in \mathcal{S}$ with a descriptive tuple m_s of size $M \in \mathbb{N}^*$. This tuple captures metadata information on the song, e.g., the name of the artist or the album it belongs to:

$$m_s \in \mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2 \times \dots \times \mathcal{M}_M. \quad (3.2)$$

In this equation, each $\mathcal{M}_i$ denotes the set of possible values for a given metadata type, e.g., the set of possible artists.

3.2.2 APC

In such a setting, the APC problem consists in extending a playlist $p \in \mathcal{P}$ by adding more songs matching the target characteristics of p (Schedl et al., 2018; C.-W. Chen et al., 2018). Formally, an APC model is a function:

$$f: \mathcal{P} \times \mathcal{S} \rightarrow \mathbb{R}, \quad (3.3)$$

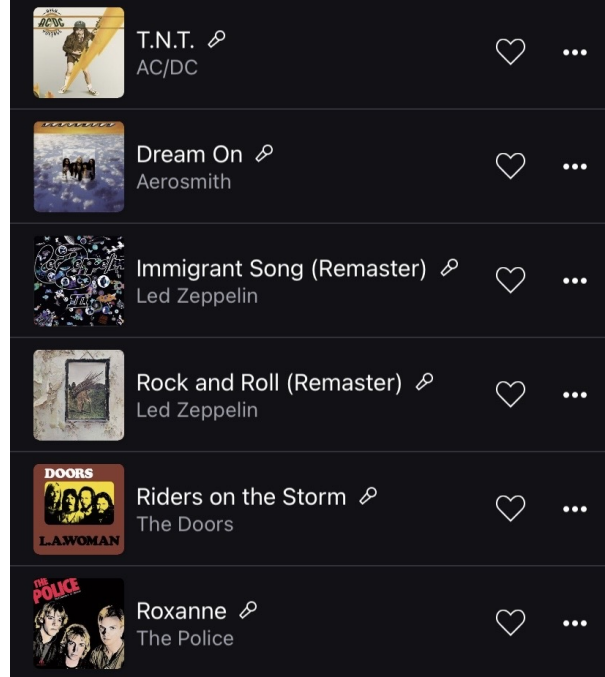


Figure 3.1: An example of a user playlist on Deezer.

associating each $p \in \mathcal{P}$ and $s \in \mathcal{S}$ with a “relevance” or “similarity” score $f(p, s)$. The higher the score, the more likely s will fit well as a continuation of p , according to f . To assess and compare the performance of APC models, the standard methodology consists in:

- Collecting a set of test playlists, unseen during training.
- Then, masking the last songs of these test playlists.
- Finally, evaluating the ability of each model to complete these partially masked sequences using song relevance scores, computing metrics such as the ones we will consider in Section 3.4.

In essence, the task is a sequential recommendation tasks where the item set is a large scale catalog of songs.

3.3 Large-Scale Playlist Continuation

In this section, we present and analyze our proposed framework to build scalable yet effective APC models for large-scale applications.

3.3.1 Objectives and Requirements

Our main goal is to provide a comprehensive set of guidelines for implementing APC models that are scalable by design. Additionally, we aim to maintain flexibility and generality in our approach.

Specifically, in this chapter, our definition of *scalability* is twofold. Firstly, we want to build models that can handle large datasets with millions of songs and users, as is often required in real-world applications on music streaming services. Furthermore, insights from industrial practitioners have emphasized that scalability is also essential during the *inference* phase (Rendle et al., 2020). In practice, while retraining an APC system is often required in production environments to incorporate new data (e.g., get information on new songs and playlists, and consequently update model parameters), these updates can be purposely delayed until a batch of new events has been observed. This allows model training operations to be performed *offline on a regular schedule*, such as once per day, and enables the model to take more time to update its parameters (Ricci et al., 2011). In contrast, the inference process, repeatedly making recommendations to users, must be accomplished:

- *Online* on the service: as users regularly create and update playlists, the exact song sequence that must be extended by the APC model can *not* be processed offline in advance.
- With minimal delay: to minimize latency costs when providing updated recommendations to users. For example, on a global service like Deezer, an inference time longer than *a few tens of milliseconds* would be unacceptable in production.

For these reasons, scalability of inference will be particularly crucial in our framework. Consequently, we will aim to minimize the number of *online* operations whose complexity depends on the size of the dataset, described by N , L , M , and $\max_{1 \leq k \leq M} |\mathcal{M}_k|$. In practice, the number of songs N is the largest of these parameters. Therefore, we will prioritize the *offline* pre-computation of operations depending on N , whenever possible.

3.3.2 A Scalable Framework for APC

We now formally introduce our framework for APC at scale. Without loss of generality, we focus¹ on single-stage models $f: \mathcal{P} \times \mathcal{S} \rightarrow \mathbb{R}$ directly scoring each song $s \in \mathcal{S}$ with a similarity score $f(p, s)$ for extending some playlist $p \in \mathcal{P}$.

Also, we focus² on APC models learning “embedding” vectorial representations for any playlist p and song s . We denote them by $\mathbf{h}_p \in \mathbb{R}^D$ and $\mathbf{h}_s \in \mathbb{R}^D$, respectively, for some embedding dimension $D \in \mathbb{N}^*$. Such models can be written as follows:

$$f(p, s) = \text{SIM}(\mathbf{h}_p, \mathbf{h}_s), \quad (3.4)$$

for some similarity function $\text{SIM}: \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$. For each element of Equation (3.4), we establish properties that must be verified in our framework, following the principles from Section 3.3.1.

¹Single-stage APC models stemming from our scalable framework could be integrated into two-stage/ensemble systems mentioned in Section 2.3.3.3 of Chapter 2.

²We provide a justification for this constraint in Section 3.3.2.3.

3.3.2.1 Song Representation

Firstly, we note that each song embedding vector $\mathbf{h}_s$ depends on the song $s \in \mathcal{S}$ and, possibly, the metadata $m_s \in \mathcal{M}$. Since one embedding vector must be computed for each of the N songs of the catalog, we require to perform this operation *offline in advance*, with its result stored to be rapidly accessible with a simple read operation. Consequently, song embedding vectors should *not* depend on the playlist p to extend. Formally, in our framework, each $\mathbf{h}_s$ vector should be computed via a function³:

$$\phi: \mathcal{S} \rightarrow \mathbb{R}^D, \quad (3.5)$$

learning a unique vectorial representation for each song.

3.3.2.2 Playlist Representation

On the other hand, the representation $\mathbf{h}_p$ of each playlist $p \in \mathcal{P}$ can not be computed offline in advance, as playlists can be created and updated at any time by users (see Sections 5.2.1 and 3.3.1). Hence, according to our requirements, the playlist-level representation learning operations should not depend on a large set of inputs. We propose to learn $\mathbf{h}_p$ solely from the song sequence $(s_{p1}, \dots, s_{pl})$ characterizing p . Since large-scale APC usually implies dealing with sparse data (C.-W. Chen et al., 2018), our framework also aims to benefit from parameter sharing whenever possible, by adding two constraints on the function learning $\mathbf{h}_p$:

- It should leverage song representations $\mathbf{h}_s$ as input.
- It should be independent of the playlist length $l \in \{1, \dots, L\}$.

Hence, by defining $\mathcal{E} = \bigcup_{l=1}^L \mathbb{R}^{D \times l}$, each $\mathbf{h}_p$ should be computed via:

$$g: \mathcal{E} \rightarrow \mathbb{R}^D, \quad (3.6)$$

a function *aggregating* representations from the songs present in the playlist p into a single playlist embedding representation $\mathbf{h}_p$.

3.3.2.3 Similarity

The final scoring operation $f(p, s) = \text{SIM}(\mathbf{h}_p, \mathbf{h}_s)$, estimating how likely each song s will fit well as a continuation of some playlist p using the above $\mathbf{h}_s$ and $\mathbf{h}_p$, constitutes the complexity bottleneck of the APC prediction process. Indeed, it depends on N , assuming that the APC model actually considers all songs from $\mathcal{S}$ when making recommendations (and not some approximate subset). Moreover, it can not be pre-computed offline beforehand, as it depends on the playlist representation, which is itself computed online when the model observes the exact song sequence to extend. Although similarity functions based on neural networks have been proposed (He, Liao, et al., 2017a), Rendle et al. (2020) have demonstrated that they do not outperform the traditional inner product, whose ability to rapidly score millions of items makes it a natural choice for use as a similarity function in our framework: $\langle \mathbf{h}_p, \mathbf{h}_s \rangle$.

³We imply the metadata in dependencies, as each m_s exclusively depends on s .

3.3.2.4 Represent-Then-Aggregate

In summary, as illustrated in Figure 3.2, our framework constrains APC models to adopt the structure:

$$f(p, s) = \langle g(\mathbf{h}_{s_{p1}}, \dots, \mathbf{h}_{s_{pl}}), \mathbf{h}_s \rangle, \text{ with } \forall k \in \mathcal{S}, \mathbf{h}_k = \phi(k). \quad (3.7)$$

As our framework involves a function ϕ representing songs and a function g aggregating these representations, we refer to our framework as *Represent-Then-Aggregate* (RTA) in the following. Our experiments in Sections 3.4 and 3.5 will empirically confirm the scalability of various APC models complying with this framework.

3.3.3 Examples of RTA Architectures

While ensuring scalability by design, our RTA framework remains flexible enough to incorporate a wide range of modern models. In particular, our experiments will consider the following options for song representation learning and playlist-level aggregation.

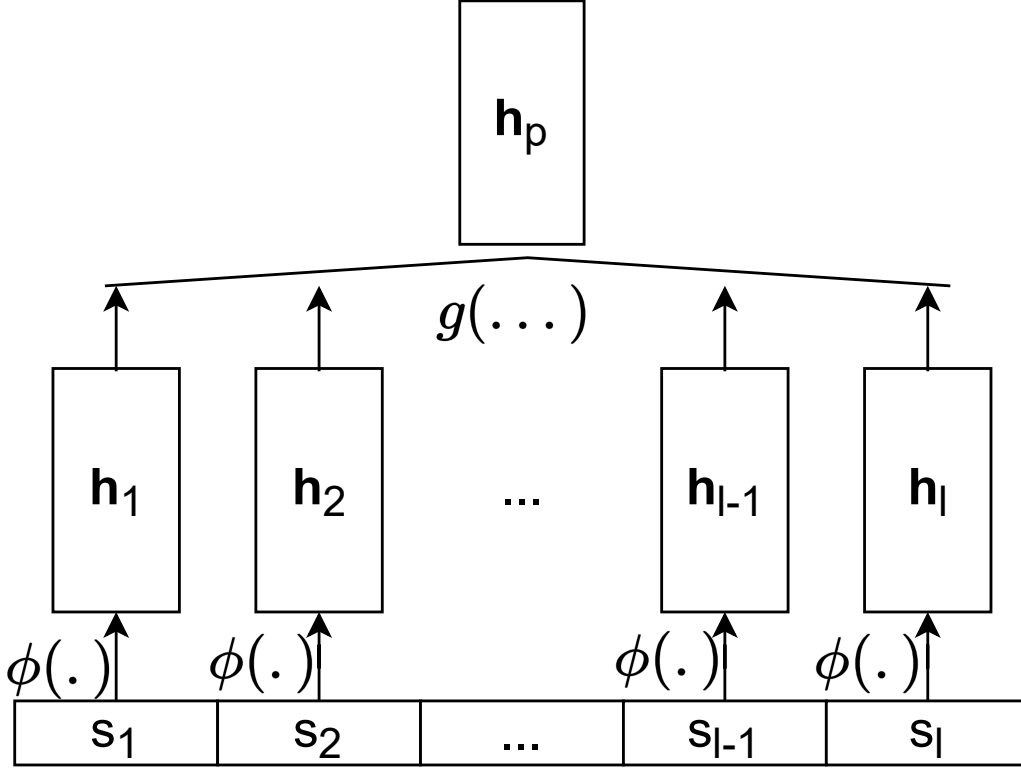
3.3.3.1 Song Representation Function ϕ

- $\mathbf{h}_s = \phi_e(s) = \mathbf{e}_s$: we directly associate each song s with a vector $\mathbf{e}_s \in \mathbb{R}^D$ using a representation learning model trained from playlist-song occurrence data (see Section 3.4.2.1).
- $\mathbf{h}_s = \phi_{\text{FM}}(s) = \sum_{m \in m_s} \mathbf{e}_m$: we represent each song by the sum or average of embedding vectors associated with its metadata, which we themselves learn using a model trained from occurrence data. We refer to this approach as FM due to its connection to Factorization Machines (Rendle, 2010).
- $\mathbf{h}_s = \phi_{\text{NN}}(s) = \text{NN}(\{\mathbf{e}_m, \forall m \in m_s\})$: we represent each song by the output of a neural network NN processing song metadata. This encompasses any architecture leveraging a *tower* to summarize heterogeneous and sparse song information in a single embedding (Covington et al., 2016a). Our experiments will consider an attention-based neural architecture similar to the one of Song et al. (Song et al., 2019).

3.3.3.2 Playlist Sequence Modeling / Aggregation Function g

- $g_{\text{AVG}}(\mathbf{h}_{s_{p1}}, \dots, \mathbf{h}_{s_{pl}}) = \frac{1}{l} \sum_{i=1}^l \mathbf{h}_{s_{pi}}$: the playlist representation is the average representation of the songs it contains⁴.
- $g_{\text{CNN}}(\mathbf{h}_{s_{p1}}, \dots, \mathbf{h}_{s_{pl}}) = \text{CNN}([\mathbf{h}_{s_{p1}}; \dots; \mathbf{h}_{s_{pl}}])$: a gated *Convolutional Neural Network* (CNN) processing a concatenation of song representations learns the playlist representation. A key component of the ensemble model that won the RecSys 2018 APC challenge (Volkovs et al., 2018), the use of such gated CNN for APC was inspired by its successful use in language modeling (Dauphin et al., 2017).

⁴As this convenient aggregation strategy is commonly found in RS, the next chapter will be dedicated to studying the implications of using this approach for representing sequences of embeddings.

Figure 3.2: Our *Represent-Then-Aggregate* framework for APC.

- $g_{\text{GRU}}(\mathbf{h}_{s_{p_1}}, \dots, \mathbf{h}_{s_{p_l}}) = \text{GRU}([\mathbf{h}_{s_{p_1}}; \dots; \mathbf{h}_{s_{p_l}}])$: a recurrent neural network (RNN) composed of gated recurrent units (GRU) learns the playlist representation. Our experiments will consider an architecture similar to the one Hidasi et al. (2016) but with a different loss, presented in Section 3.3.4.
- $g_{\text{Transformer}}(\mathbf{h}_{s_{p_1}}, \dots, \mathbf{h}_{s_{p_l}}) = \text{Decoder}([\mathbf{h}_{s_{p_1}}; \dots; \mathbf{h}_{s_{p_l}}])$: the Decoder part of a Transformer network (Vaswani et al., 2017) learns the playlist representation. The choice of keeping only the Decoder for APC was motivated by the recent surge of GPT models, which have demonstrated state-of-the-art performances on the analogous task of sentence continuation (Radford et al., 2018).

The neural networks architectures introduced above share the common ability to output a fixed size embedding given a sequence of arbitrary size; with the aim of emphasizing the commonalities of these approaches rather than their differences, we deliberately omit providing a detailed description of their respective architecture. We refer the interested reader to (Vaswani et al., 2017) for a detailed description of the Transformer architecture and to (Goodfellow et al., 2016) for the other types of neural networks.

3.3.3.3 Limitations

These examples illustrate the versatility of our RTA framework. Simultaneously, we are also aware of some limitations it imposes. We discuss two examples of approaches that can not be express by equation (3.7).

Neural Collaborative Filtering (NCF) The claimed superiority of NCF models (He, Liao, et al., 2017b) on recommendation tasks stems from their ability to consider mathematically complex functions of user and items features together, like high-order polynomials, to make predictions. They do so by concatenating user and item features early in the input transformation process, before several layers of neural networks are applied, contrasting with our approach of comparing playlist and song representations in the last operation of score prediction Equation (3.7). We argue that discarding NCF models in our framework for large scale real-time APC is a minor inconvenient, since those models are inefficient when it comes to scoring in parallel and ranking a large set of songs given a user. In particular, they do not benefit from common approximate nearest neighbors search approaches, and we believe that their usage in real-world applications is conditioned on significant advancement regarding the efficiency of inference of this family of models. In fact, this reflection can be generalized to any approach where playlist and song features are first concatenated before a non-linear model is tasked with outputting a score used for ranking songs.

Word2vec Word2vec models (Mikolov, K. Chen, et al., 2013; Mikolov, Sutskever, et al., 2013) leverage **different** token representations depending on whether the token is seen as the input of the model, or considered as a possible predicted output. The training objective of Word2vec can be understood as making sure that the embedding from the **first layer** representing a word is close the embeddings from the **second layer** present in its context (or the other way around). This clever approach ensures that when extracting only one layer as a dictionary of embeddings, typically the first one, words that are close together are not the ones that often co-occur, but the ones that can often be interchanged. This is especially useful in NLP since the syntax and grammar of languages makes it rare to see synonyms occur in the same sentence, which is why embedding representations based on word co-occurrence, like matrix factorization methods, fail at building latent spaces where similarity accounts for semantic proximity (see the discussion by Goldberg et al. (2014)).

Although Word2vec models have been proposed for recommending items (Vasile et al., 2016; D. Wang, Deng, X. Zhang, et al., 2016), we believe that in the case of APC, songs appearing in the same playlist should have close representations, i.e., songs should be similar to their context. Indeed songs that often appear together are likely to also be substitutable (an assumption at the core of matrix factorization techniques (Koren, Bell, and Volinsky, 2009)). In fact, one could expect from a good model for APC that the representation it uses for a playlist of size 1 be very close to the representation of the unique song present in the playlist, which is unlikely to be the case with Word2vec models. For this reason, our framework excludes these approaches in favor of more parameter sharing between playlist and song representation functions.

3.3.4 Example of a Training Procedure

While our RTA framework remains general, we suggest the following training procedure, which can be applied to optimize all models stemming from this framework. Our experiments in this chapter will adopt this same procedure to train all RTA-based APC models.

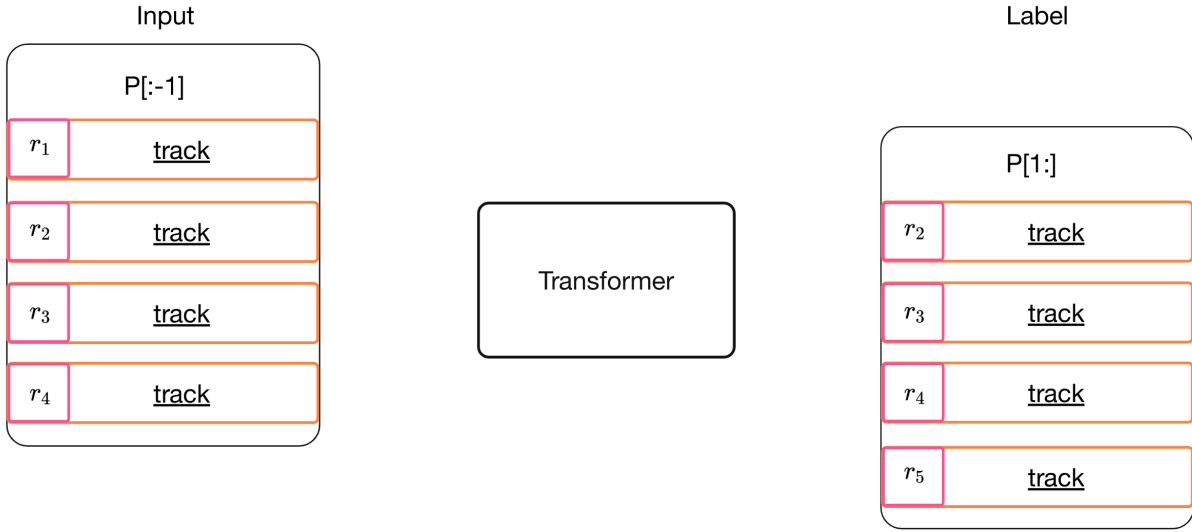


Figure 3.3: Training procedure of an RTA model using a transformer. The initial sequence of songs is shifted by 1 position to create the target to predict.

Our preliminary experiments have shown that leveraging pre-trained embedding vectors from a weighted regularized matrix factorization as initial song representations improves performances. Therefore, we propose to start by factorizing⁵ the playlist-song occurrence matrix of the APC problem under consideration, associating each song s with some $\mathbf{e}_s \in \mathbb{R}^D$ used for initialization. Simultaneously, we initialize embedding vectors of metadata information by averaging vectors $\mathbf{e}_s$ of songs sharing the same metadata value.

Starting from these initial representations, we jointly optimize the weights of the models selected as ϕ and g , using batches of playlists of varying lengths. For each playlist p of length $l \in \{2, \dots, L\}$, we create $l - 1$ sub-playlists denoted $p_{:i}$ using the first i songs of p , with $i \in \{1, \dots, l - 1\}$. Then, we sample a negative song set $\mathcal{S}^-(p)$ from $\mathcal{S} \setminus p$. As we expect the APC model to return a high similarity scores for $f(p_{:i}, p_{s_{i+1}})$ (a sub-playlist and the song immediately following) and a lower score for songs that are absent from p , our procedure optimizes model weights via gradient descent minimization (Goodfellow et al., 2016) of the loss $\mathcal{L}(p) = \mathcal{L}_{\text{pos}}(p) + \mathcal{L}_{\text{neg}}(p)$, with:

$$\mathcal{L}_{\text{pos}}(p) = - \sum_{i=1}^{l-1} \log \left(\frac{1}{1 + e^{-f(p_{:i}, p_{s_{i+1}})}} \right), \quad (3.8)$$

and:

$$\mathcal{L}_{\text{neg}}(p) = - \sum_{i=1}^{l-1} \sum_{s^- \in \mathcal{S}^-(p)} \log \left(1 - \frac{1}{1 + e^{-f(p_{:i}, s^-)}} \right). \quad (3.9)$$

⁵We use the Python library *implicit* in experiments: <https://github.com/benfred/implicit>

3.4 Offline Experiments

We now present an experimental evaluation of our framework on the Million Playlist Dataset (MPD) (C.-W. Chen et al., 2018; Zamani et al., 2019). We release our source code on GitHub⁶, to ensure the reproducibility of our results and to encourage the future usage of our framework.

3.4.1 Experimental Setting

3.4.1.1 Dataset

We consider the entire Million Playlist Dataset (MPD) for our experiments. This dataset includes one million playlists created by Spotify users, using more than two million songs (see Table 2.2 from Chapter 2). For each song s , we have metadata information⁷ corresponding to its artist (art_s), the album it belongs to (alb_s), the duration of the song (dur_s), and the number of times it occurs in the dataset, a metric referred to as popularity (pop_s). Hence, $m_s = (\text{art}_s, \text{alb}_s, \text{dur}_s, \text{pop}_s)$. As dur_s and pop_s are numerical values that can not be easily mapped to embedding vectors, we group similar values together into buckets:

- For the duration, we create linear buckets of 30 seconds each, with songs longer than 20 minutes being assigned to the same bucket. We have: $\text{bucket}_{\text{dur}}(s) = \min(40, \left\lceil \frac{\text{dur}_s}{30} \right\rceil)$.
- For the popularity, we create buckets using a logarithmic scale, so that each small value has its own bucket, while higher values are assigned to the same bucket. By setting $\alpha = 45\,000$, i.e., the highest number of occurrences observed in the dataset, we have: $\text{bucket}_{\text{pop}}(s) = \min(100, 1 + 100 \times \left\lfloor \frac{\log(\text{pop}_s)}{\log(\alpha)} \right\rfloor)$.

3.4.1.2 Task

We consider a large-scale APC evaluation task similar to the one presented in Section 3.2.2. Specifically, we start by randomly sampling 20 000 playlists of length $l \geq 20$ songs from the MPD, to constitute a validation set and a test set of 10 000 playlists each. The remaining 980 000 playlists constitute our training set.

Then, we mask the last songs of test playlists, so that only their n_{seed} first songs are visible. We consider ten different configurations, with n_{seed} varying from 1 to 10, randomly selecting 1 000 test playlists for each configuration. Our experiments consist in assessing the ability of several APC models trained on the 980 000 train playlists (see Section 3.4.2) to retrieve the masked songs of each test playlist. Consistently with the RecSys 2018 APC Challenge, we require each APC model to predict a ranked list of $n_{\text{reco}} = 500$ candidate songs to continue each test playlist.

⁶ <https://github.com/deezer/APC-RTA>

⁷The MPD also includes the *title* of each playlist as an additional information (C.-W. Chen et al., 2018). However, Zamani et al (Zamani et al., 2019) have reported that this information did not significantly improve performances during the RecSys 2018 APC Challenge. Moreover, titles are often unavailable for APC, as well as for closely related tasks such as radio generation/personalization (Bontempelli et al., 2022). For these reasons, we omit playlist titles in these experiments.

3.4.1.3 Metrics

In the following, we analyze nine different metrics to evaluate each model on the specified APC task. Firstly, we consider five *accuracy-oriented* metrics. Besides the prevalent Precision and Recall scores (Schedl et al., 2018), we report the three metrics used for evaluation during the RecSys 2018 APC Challenge (C.-W. Chen et al., 2018):

- Normalized Discounted Cumulative Gain (NDCG) (Järvelin et al., 2002): acts as a measure of ranking quality. It increases when the ground truth masked songs are placed higher in the ranked list of candidate songs recommended by the APC model (Y. Wang et al., 2013).
- R-Precision (Schütze et al., 2008): jointly captures the proportion of masked songs and artists recommended by the model. Artist matches increase the score even if the predicted song is incorrect (C.-W. Chen et al., 2018).
- Clicks: indicates how many batches of ten candidate songs must be recommended (starting with the top ten candidates) before encountering the first ground truth song. Unlike previous metrics, Clicks should, therefore, be minimized (C.-W. Chen et al., 2018).

Zamani et al. (Zamani et al., 2019) provide exact formulas for these three metrics. In addition, we compute two *popularity-oriented* scores, described by Ludewig and Jannach (Ludewig and Jannach, 2018). They monitor the tendency of each model to cover the entire catalog when providing recommendations:

- Coverage: computes the percentage of songs from the catalog recommended at least once during the test phase.
- Popularity bias: averages the popularity of recommended songs, computed by counting the occurrences of each song in the training set and applying min-max normalization (Ludewig and Jannach, 2018). Low scores indicate that less popular songs are recommended.

Finally, to compare the *scalability* of each model, we compute:

- Training time: the time required for a model to learn from the training set until no more improvement on the validation set could be observed, regarding any accuracy-oriented metrics.
- Inference time: the average time required to recommend a list of $n_{\text{reco}} = 500$ candidate songs to extend a test playlist.

3.4.2 APC Models

We compare the scalability and performance of ten APC models.

3.4.2.1 Models based on our RTA framework

Firstly, we consider six different RTA models, built by leveraging the following song representation and playlist aggregation/modeling functions:

- MF-AVG⁸: employs the representation function ϕ_{e-MF} (see below), combined with the aggregation function g_{AVG} .
- MF-CNN⁸: ϕ_{e-MF} combined with g_{CNN} .
- MF-GRU: ϕ_{e-MF} combined with g_{GRU} .
- MF-Transformer: ϕ_{e-MF} combined with $g_{Transformer}$.
- FM-Transformer: ϕ_{FM} combined with $g_{Transformer}$.
- NN-Transformer: ϕ_{NN} combined with $g_{Transformer}$.

In the above list, we use the notation from Section 3.3.3. We additionally denote by ϕ_{e-MF} a particular example of representation learning function ϕ_e . Specifically, ϕ_{e-MF} initializes song embedding vectors using the weighted regularized matrix factorization mentioned in Section 3.3.4 and directly refines these representations via the minimization of the loss defined in this same section. All six models learn embedding vectors of dimension $D = 128$. We optimize them via the procedure of Section 3.3.4, using stochastic gradient descent (Goodfellow et al., 2016) with batches of 128 playlists and 100 negative samples⁹. We tune all hyperparameters to maximize NDCG scores on the validation set, using the *Optuna* library (Akiba et al., 2019) for efficient hyperparameter search. For brevity, we report all optimal hyperparameter values in our GitHub repository⁶. All g_{CNN} , g_{GRU} , $g_{Transformer}$, and ϕ_{NN} models have from 1 to 3 layers. The width of hidden layers went from 128 to 1024 units. The kernel size for g_{CNN} models went from 2 to 5. The number of heads for $g_{Transformer}$ and ϕ_{NN} is a power of 2 ranging from 2^1 to 2^6 . We test learning rates ranging from 10^{-3} to 1, weight decays from 10^{-9} to 10^{-4} , and dropout rates from 0 to 0.5 (Srivastava et al., 2014). For every model, we half the learning rate at each epoch and performed early stopping using the validation set (Goodfellow et al., 2016). We use Python, training models on a single CPU machine with 25 GB of RAM and a Tesla P100 GPU accelerator.

3.4.2.2 Baselines

By studying these six different RTA models, our goal is to measure how well various modern methods would scale and perform using our framework, including techniques previously overlooked for large-scale APC. To establish reference points, we also concurrently evaluate and compare four non-RTA baselines:

- SKNN (Hariri et al., 2015): a session-based nearest neighbors approach that, despite its apparent simplicity, can reach competitive performances with respect to CNN and GRU models (Ludewig, Mauro, et al., 2021).

⁸ The ensemble system that won the RecSys 2018 Challenge included such a model (Volkovs et al., 2018).

⁹ Note that the common learning procedure used for all models is the reason why MF-Transformer isn't referred to as "SASRec" (W.-C. Kang et al., 2018) even though the architectures are essentially the same.

- VSKNN (Ludewig and Jannach, 2018): a variant of SKNN considering the song order within the playlist sequence as well as its popularity.
- STAN (Garg et al., 2019): a variant of SKNN capturing the position of the song in the playlist, information from past playlists, and the position of recommendable songs in neighboring playlists.
- VSTAN (Ludewig, Mauro, et al., 2021): combines the ideas of STAN and VSKNN into a single approach, and adds sequence-based item scoring.

We select these four baselines for two reasons:

- Firstly, regarding performances, they tend to surpass alternatives in the recent empirical analysis of Ludewig et al. (Ludewig, Mauro, et al., 2021).
- Secondly, although they were evaluated on smaller datasets, we could train them on the MPD using our machines.

We tune all hyperparameters by maximizing *Normalized Discounted Cumulative Gain* (NDCG) validation scores. Our set of possible values for hyperparameters is similar to Ludewig et al. (Ludewig, Mauro, et al., 2021). These four models require timestamps of actions performed on the items of each sequence. As the MPD includes the timestamp of the latest update of each playlist and the duration of each song, we made the simplifying assumption that each song had been added right after the previous one had been listened to once.

3.4.2.3 Scope Limitation

For completeness, we note that we initially considered other baselines, which we ultimately discarded for scalability reasons. Most notably, we will not compare to the multimodal CF model from Yang et al. (Yang et al., 2018) and the hybrid model from Rubtsov et al. (Rubtsov et al., 2018). While reaching promising results during the RecSys 2018 APC Challenge, both of these two models require more than *four full days* to train on the MPD using their respective implementations (even on a machine with 28 cores and 200GB of RAM, for the latter one (Rubtsov et al., 2018)). Considering that the MPD represents only a small fraction of playlists created by Spotify users, these models would require even more time and resources to be used by such a service. As scalability is at the core of our work, our experiments, therefore, discard these demanding baselines.

In addition, as explained in Section 3.3.2, we focus on single-stage APC models, directly processing and scoring all songs from $\mathcal{S}$. For clarity of exposure and comparisons, our experiments do not include systems aggregating *multiple* song scoring models. Nonetheless, it bears mentioning that the ten models from our experiments could easily be used as the first “candidate retrieving” part of a two-stage architecture (see Section 3.5.1.2), or as components in a larger ensemble system (e.g., Volkovs et al. (2018) themselves combined MF-AVG and MF-CNN in their ensemble during the challenge). As future work, we plan to extend the scope of this chapter, by examining the use of several RTA models *in conjunction* for large-scale APC.

Models	Precision (in %)	Recall (in %)	R-Precision (in %)	NDCG (in %)	Clicks (in number)	Popularity (in %)	Coverage (in %)	Total time for model training	Acceptable for MSS?	Inference time by test playlist	Acceptable for MSS?
Baselines											
SKNN	4.93 $\pm$ 0.09	36.94 $\pm$ 0.49	21.42 $\pm$ 0.30	27.66 $\pm$ 0.40	3.82 $\pm$ 0.22	14.86 $\pm$ 0.17	12.64	4 min	✓	~ 0.5 sec	×
VSKNN	4.93 $\pm$ 0.09	36.83 $\pm$ 0.49	21.27 $\pm$ 0.30	27.54 $\pm$ 0.40	3.84 $\pm$ 0.22	15.18 $\pm$ 0.17	12.28	4 min	✓	~ 0.5 sec	×
STAN	4.32 $\pm$ 0.09	32.73 $\pm$ 0.48	19.12 $\pm$ 0.28	24.26 $\pm$ 0.38	4.73 $\pm$ 0.23	13.43 $\pm$ 0.16	27.03	4 min	✓	~ 0.5 sec	×
VSTAN	4.59 $\pm$ 0.09	34.84 $\pm$ 0.49	20.33 $\pm$ 0.30	25.99 $\pm$ 0.40	4.49 $\pm$ 0.23	11.35 $\pm$ 0.16	20.54	3 min	✓	~ 0.5 sec	×
RTA Models											
MF-AVG	4.43 $\pm$ 0.09	32.64 $\pm$ 0.46	19.80 $\pm$ 0.28	24.46 $\pm$ 0.37	5.95 $\pm$ 0.28	21.37 $\pm$ 0.19	1.07	15 min	✓	< 0.01 sec	✓
MF-CNN	5.00 $\pm$ 0.09	37.89 $\pm$ 0.45	21.15 $\pm$ 0.26	26.83 $\pm$ 0.35	3.13 $\pm$ 0.19	15.43 $\pm$ 0.14	3.99	~ 5 h	✓	< 0.01 sec	✓
MF-GRU	5.16 $\pm$ 0.09	39.16 $\pm$ 0.46	21.83 $\pm$ 0.27	28.22 $\pm$ 0.36	2.77 $\pm$ 0.18	12.70 $\pm$ 0.13	3.18	~ 7 h	✓	< 0.01 sec	✓
MF-Transformer	5.20 $\pm$ 0.09	39.76 $\pm$ 0.47	22.30 $\pm$ 0.28	29.04 $\pm$ 0.38	2.60 $\pm$ 0.17	10.46 $\pm$ 0.13	5.35	~ 5 h	✓	< 0.01 sec	✓
FM-Transformer	5.31 $\pm$ 0.09	40.46 $\pm$ 0.47	22.29 $\pm$ 0.27	29.21 $\pm$ 0.37	2.52 $\pm$ 0.17	11.74 $\pm$ 0.13	10.18	~ 6 h	✓	< 0.01 sec	✓
NN-Transformer	5.26 $\pm$ 0.09	40.17 $\pm$ 0.47	22.18 $\pm$ 0.27	29.14 $\pm$ 0.37	2.17 $\pm$ 0.15	10.33 $\pm$ 0.13	11.81	~ 6 h	✓	< 0.01 sec	✓

Table 3.1: Automatic Playlist Continuation (APC) on the Million Playlist Dataset (MPD) (C.-W. Chen et al., 2018), using various models stemming from our proposed Represent-Then-Aggregate (RTA) framework and other baselines. Scores are computed on test playlists and averaged for n_{seed} varying from 1 to 10. All models are required to provide a ranked list of $n_{\text{reco}} = 500$ candidate songs to continue each test playlist. All embedding models learn representations of dimension $D = 128$, with other hyperparameters set as described in Section 3.4.2. The two columns “Acceptable for MSS?” indicate whether the reported training and inference times would be acceptable (✓) or not (×) for APC on a Music Streaming Service (MSS), and are discussed in Section 3.4.3.2.

3.4.3 Results

3.4.3.1 Performance

Table 3.1 reports all metrics¹⁰ on test playlists, averaged for n_{seed} varying from 1 to 10, along with 95% confidence intervals. Firstly, we confirm previous insights (Ludewig, Mauro, et al., 2021) claiming that properly tuned nearest neighbors models, such as SKNN and VSKNN, can reach comparable performances with respect to some neural models (e.g., with a 27.66% NDCG for SKNN, vs. 26.83% for MF-CNN and 28.22% for MF-GRU). Regarding accuracy-oriented metrics, STAN, VSTAN, and MF-AVG tend to underperform, while Transformer-based models using our RTA framework achieve the best results (e.g., with a top 40.46% Recall for FM-Transformer). The FM-Transformer variant, leveraging Factorization Machine methods for song representation learning, slightly surpasses MF-Transformer and NN-Transformer on three metrics, even though all scores are statistically very close. We note that, in particular, Transformer-based models outperform alternatives in terms of ranking quality (NDCG, Clicks), a valuable property for real-world usage. This validates the relevance of our work, which specifically aims to facilitate and support the incorporation of such modern sequence modeling techniques into large-scale APC systems.

Figure 3.4 shows the evolution of different metrics as the number of visible songs n_{seed} in test playlists increases. While confirming previous conclusions, the figure also highlights that sequence modeling neural architectures struggle on very short playlists. For $n_{\text{seed}} \leq 2$, VSKNN and SKNN even surpass Transformers. Another interesting observation from Figure 3.4 and Table 3.1 relates to popularity-oriented scores. On average, MF-AVG provides the most mainstream recommendations (with a top 21.37% Popularity score, and a bottom 1.07% Coverage of the catalog), followed by VSKNN and SKNN. On the contrary, the good performances of

¹⁰Recall: we aim to *minimize* Clicks and Popularity. We report no confidence intervals for Coverage as it considers the *total* number of recommended songs for the test set.

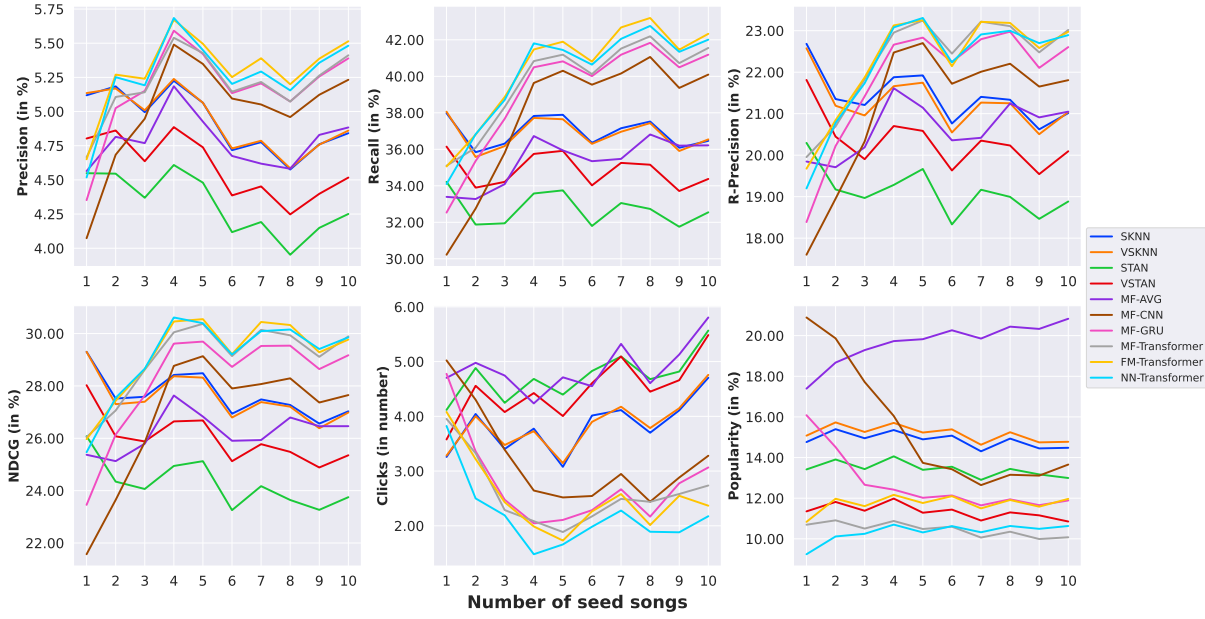


Figure 3.4: Automatic Playlist Continuation (APC) on the Million Playlist Dataset (MPD) (C.-W. Chen et al., 2018), using the setting and models from Table 3.1. We split scores depending on n_{seed} , i.e., the number of visible songs in each test playlist.

Transformers do not stem from focusing only on popular songs. Interestingly, STAN and VSTAN recommend the most diverse songs (e.g., with a top 27.03% Coverage for STAN), but at the cost of a deteriorated performance.

3.4.3.2 Scalability

We now discuss scalability metrics displayed on the right side of Table 3.1. They are the ones that best showcase the relevance of our RTA framework for large-scale industrial applications. While baselines were faster to train (3 to 4 minutes on our machines), the training times associated with RTA-based models (15 minutes to a few hours) would remain acceptable for APC on a music streaming service. Indeed, as explained in Section 3.3.1, training operations do not have to be executed in real-time on such a service. They can be performed *offline* on a regular schedule (e.g., once per day) to update the parameters of the production model. In practice, all models evaluated in Table 3.1 could integrate new input data overnight.

In contrast, the *inference* process must be efficient on a music streaming service. As detailed in Section 3.3.1, it would usually need to be *repeatedly* performed *online*, to provide recommendations *in real-time* to many users. In particular, we explained in this previous section that an inference time longer than a few tens of milliseconds by playlist would be unacceptable in the production environment of a service like Deezer. In Table 3.1, all baselines require 500 milliseconds (0.5 seconds) to extend each test playlist with a list of 500 recommended songs. All RTA models perform this same operation in less than 10 milliseconds, even the most advanced models like NN-Transformer, using an attention-based network for song representation and a Transformer for playlist modeling. While all models could benefit from better hardware, overall,

the inference times associated with our RTA models make them more suitable for production usage. These results highlight the ability of our framework to integrate complex but promising models, previously overlooked for large-scale APC, into effective systems meeting the scalability requirements associated with real-world industrial applications.

3.5 Online Experiments

This section showcases how our framework recently helped Deezer improve APC at scale. Our objective in this section is not to re-evaluate all models from Section 3.4, but rather to complete our analysis by further illustrating the practical relevance of our framework, through its successful application in an industrial setting.

3.5.1 APC on a Music Streaming Service

The global music streaming service Deezer offers an APC feature, illustrated in Figure 3.5. It allows millions of Deezer users to automatically extend the playlists they have created on the service. In April 2022, we conducted a large-scale online A/B test on this feature, aiming to improve recommendations by leveraging the possibilities induced by our RTA framework.

3.5.1.1 Reference Model

To extend user playlists, Deezer has been historically relying on “Mix-SVD”, an internal latent model for collaborative filtering (Koren and Bell, 2015; Bokde et al., 2015) that will act as a baseline for the Transformer in both section 3.5.2 and section 3.6.2. By factorizing a pointwise mutual information matrix based on track co-occurrences in user playlists and lists of favorites, using singular value decomposition (SVD) (S. Banerjee et al., 2014), this model learns vector representations of tracks in an embedding space where proximity reflects user preferences (Briand et al., 2021a). Mix-SVD is suitable for large-scale production use on a service like Deezer, as it behaves similarly to the baseline “MF-AVG” from Section 3.4. Embedding vectors of millions of tracks undergo weekly updates and are exported in a Cassandra cluster. Computation services run on a Kubernetes cluster. We compute playlist generation operations on a Scala server. In particular, we use approximate nearest neighbors techniques (Li et al., 2019) for efficient similarity search, via a Golang application incorporating the Faiss library (Johnson et al., 2019).

3.5.1.2 RTA Model

During the test, we used our RTA-based MF-Transformer, one of the three best models from Section 3.4, to provide APC to a randomly selected cohort of test users.

We trained MF-Transformer and Mix-SVD on the same private dataset of 25 million user playlists. Both models chose recommendations from a pool of two million recommendable songs. As illustrated in Figure 3.5, users requesting a playlist continuation were presented with a vertically scrollable list of 100 songs, with the five first ones being initially visible on mobile screens. Users were unaware of the algorithm change.

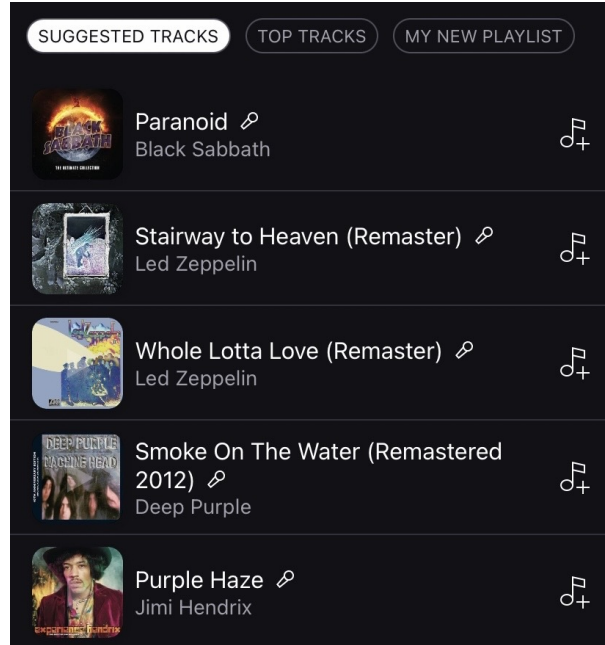


Figure 3.5: APC on Deezer: songs recommended by our MF-Transformer to continue the playlist from Figure 3.1.

3.5.2 Online A/B Test Results

Firstly, our RTA framework proved to be a valuable asset in integrating MF-Transformer into Deezer’s production environment. It permitted the successful integration of this model, resulting in its ability to perform APC at a minimal latency cost for users. As an illustration, we observed a 99th percentile inference time of only 12 milliseconds on the service, unnoticeable by users (this corresponds to the time to compute the playlist representation, score two million songs, and return 100 recommendations, using an AMD EPYC 7402P CPU machine with ten threads by inference process). This example demonstrates how our framework can enable practitioners to effectively leverage modern APC models such as Transformers, that may have otherwise been left out for alleged scalability shortcomings.

Regarding performances, our test confirmed the superiority of MF-Transformer over Mix-SVD on Deezer. Using MF-Transformer improved our internal performance indicators for the APC feature, such as the Add-To-Playlist rate, i.e., the percentage of recommended songs added to playlists by users. For confidentiality reasons, we do not report exact Add-To-Playlist rates in this chapter, nor the number of users involved in each cohort. Instead, Figure 3.6 presents *relative* Add-To-Playlist rates with respect to the reference Mix-SVD. On average, users exposed to MF-Transformer added 70% more recommended songs to their playlists than those in the reference cohort.

Following this conclusive test, MF-Transformer has been deployed to all users, becoming the default mean of automatically extending playlists on Deezer.

Later on, it was also applied to a related task of sequential song recommendation, in an experiment described in the following section.

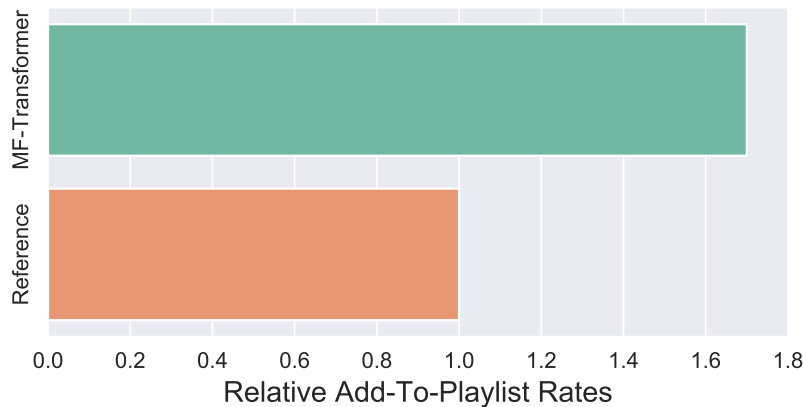


Figure 3.6: Online A/B test: relative Add-To-Playlist rates compared to the reference model from Deezer. Differences are statistically significant at the 1% level ($p\text{-value} < 0.01$).

3.6 Track Mix Generation on Deezer

In this section, we wish to assess to what extent models trained for APC can be leveraged for a distinct yet related task on Deezer that we detail below.

3.6.1 Track Mix, a Personalized Playlist Generation System on Deezer

Track Mix is a playlist generation system, accessible by millions of Deezer users since its worldwide release on the homepage of this service in 2022. As illustrated in Figure 3.7, it materializes as a personalized shortlist of up to 12 music tracks, selected¹¹ from the ones previously liked or regularly listened to by each user. They are dynamically updated at each connection to the service. As illustrated in Figure 3.8, a click on one of them generates a “mix” playlist composed of the selection and 39 other similar tracks. This is historically done by Mix-SVD (see Section 3.5.1.1); this model identifies the closest neighbors of the selected track in the embedding space, and reorders them using internal rules¹¹ to generate playlists.

3.6.2 A/B Test

3.6.2.1 Setting

During this test, we used MF-Transformer to generate Track Mix playlists for a randomly selected cohort of test users, and Mix-SVD for others. Table 3.2 reports the relative performance of MF-Transformer compared to Mix-SVD. Four metrics compare listening times for each group, and two metrics evaluate the music discovery aspect via collect actions¹².

¹¹ Technical details on the selection process for initial track and on playlist reordering are omitted as they can regularly change and fall outside the scope of this study.

¹² The number of “collect” actions is the number of recommended tracks that users added to their list of favorites or to their personal playlists.

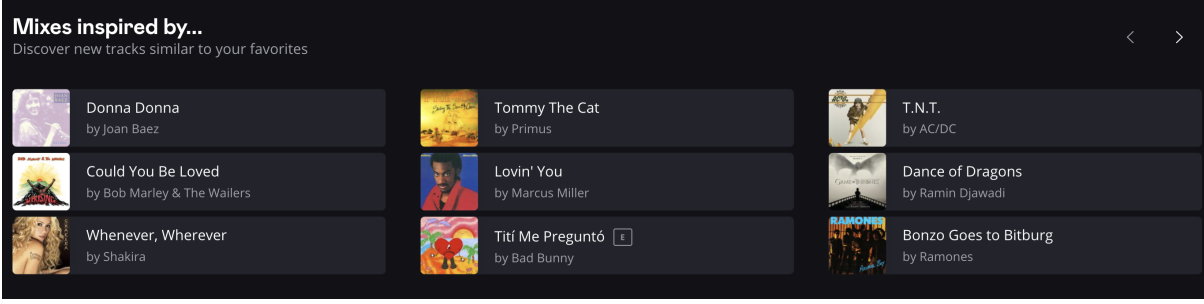


Figure 3.7: Interface of Track Mix on the website version of Deezer. The mobile app proposes a comparable feature. Track Mix presents a personalized and dynamic shortlist of music tracks among the ones previously liked and listened to by each user. A click on a track generates a mix playlist “inspired by” this track, with the aim of supporting users in discovering new tracks similar to their favorite ones.

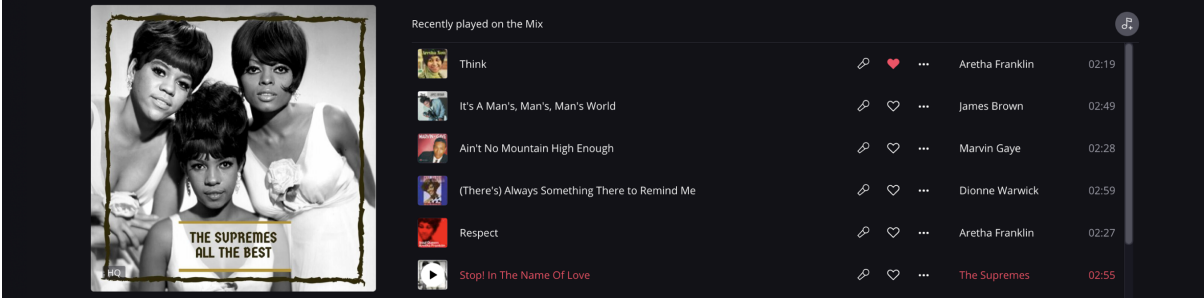


Figure 3.8: An example of the beginning of a Track Mix playlist on Deezer, generated with MF-Transformer from “Think” by Aretha Franklin.

3.6.2.2 Results

We observe that using MF-Transformer enhances the listening times of Track Mix playlists, according to all four metrics under consideration. This positive result indicates a higher usage of the feature on Deezer. However, the number of collect actions simultaneously diminishes in the MF-Transformer cohort. A close examination of results reveals that this decrease primarily affects users with over a month of activity (seniority > 30 days on Table 3.2). Conversely, MF-Transformer boosts collect actions for new users (seniority ≤ 30 days on Table 3.2). Hence, using MF-Transformer would facilitate the acquisition of preference information on these new users, which could benefit all usage-based recommender systems on Deezer by helping to overcome cold start issues (Briand et al., 2021a). We note that MF-Transformer tends to recommend more popular tracks than Mix-SVD. This increased mainstreamness might explain the lower collection levels among regular users. Indeed, they might have already liked these popular tracks. Moreover, they might have different expectations regarding music discovery. Having an already established library of favorite content, they might be more open to and even looking for specialized/niche recommendations. While Table 3.2 supports these assumptions, more investigations will be required in future work for confirmation. Our test also encourages the development of an improved MF-Transformer addressing popularity biases (Schedl et al., 2018) on regular users, e.g., by controlling popularity levels in the training dataset and the optimized loss.

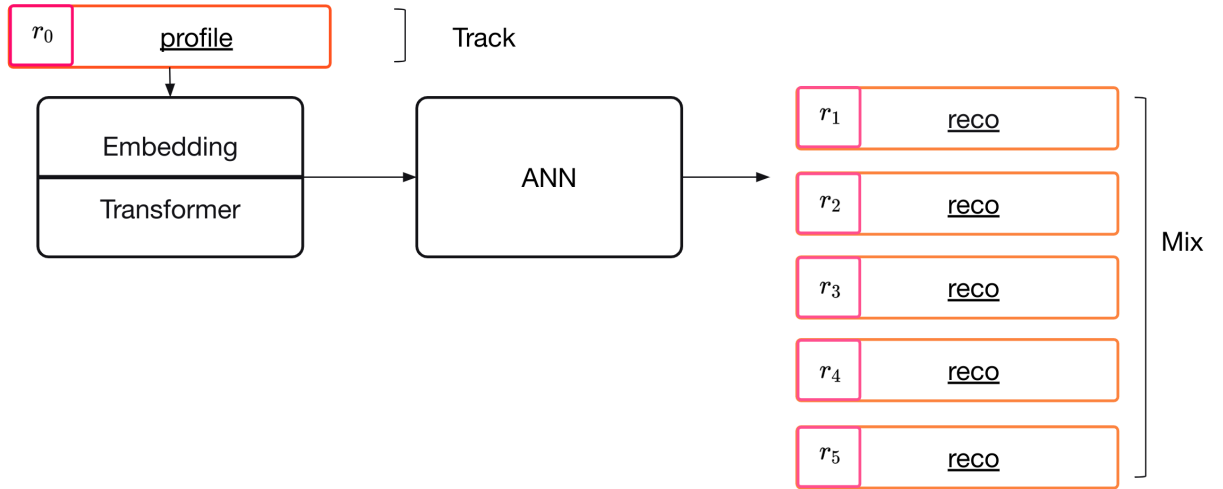


Figure 3.9: Illustration of the steps involved in the Track Mix feature

Table 3.2: Online A/B test on Track Mix: relative performance of MF-Transformer compared to Mix-SVD. Results are computed on French Deezer Premium users in March-April 2023. All improvements/declines are statistically significant at the 1% level (p-value < 0.01). We note that, although this table focuses on France, i.e., Deezer’s home market, we observed comparable results in other countries.

Metric Type		Metric Description	Relative performance of MF-Transformer		
			Results on all users	Results on users with seniority ≤ 30 days	Results on users with seniority > 30 days
User-centric	Listening time	Median daily listening time per user	+ 6.84%	+ 7.75%	+ 6.83%
	Music discovery	Average daily number of streams per user	+ 4.23%	+ 5.07%	+ 4.26%
		Average daily number of collect actions ¹² per user	- 2.48%	+ 20.26%	- 3.79%
Playlist-centric	Listening time	Average listening time per generated playlist	+ 5.12%	+ 6.62%	+ 4.98%
	Music discovery	Percentage of generated playlists listened to for >15min	+ 6.56%	+ 8.69%	+ 6.20%
		Average number of collect actions ¹² per generated playlist	- 1.98%	+ 24.10%	- 3.88%

3.7 Conclusion

In this chapter, we introduced a general framework to build scalable APC models meeting the expected requirements associated with large-scale industrial applications, e.g., on music streaming services. We provided a detailed overview of its possibilities and limitations, showing its versatility in incorporating a wide range of advanced representation learning and sequence modeling techniques, often overlooked in previous large-scale APC experiments due to their complexity. We demonstrated the empirical relevance of our framework through both offline experiments on the largest public dataset for APC, and online experiments, improving APC at scale on a global music streaming service. As future work on this task, we would like to integrate novel ideas on training procedures more adapted to the large-scale setting, such as specific loss functions (A. V. Petrov et al., 2023) and negative sampling strategies (A. Petrov et al., 2022).

This chapter has outlined the importance of two sub-tasks involved in sequential recommendation: the representation learning task, which heavily relies on embedding learning, and the aggregation function learning task. In the case of APC, deep learning models achieve superior performance compared to simple aggregators like averaging, as shown in Section 3.4. However, the need for aggregating a sequence of embeddings occurs regularly when working with recommender systems. Indeed, the hierarchical structure of music items, illustrated in Figure 1.1, makes it possible to build representations of albums, artists or playlists by aggregating the representations of their corresponding songs. For a music streaming service, it is often not worth it to train complex deep learning models for every situation that requires aggregating a set of embeddings, and most of the time those sets are simply represented by the average of all of the embeddings present in it. The relevance of this practice, which popularity likely stems from its convenience, is the subject of the next chapter.

Chapter 4

On the Consistency of Average Embedding

This chapter presents research conducted with Guillaume Salha-Galvan, Romain Hennequin, Thomas Bouabça, and Tristan Cazenave, and published in the proceedings of the 17th ACM Conference on Recommender Systems (RecSys 2023) (Bendada, Salha-Galvan, Hennequin, Bouabça, et al., 2023).

4.1 Introduction

4.1.1 Motivations

A prevalent practice in industry-oriented research, including in the work presented in the previous chapter, consists in **averaging item embeddings** to obtain embeddings for users or higher-level concepts in the same vector space (Briand et al., 2021a; Hansen et al., 2020; Pal et al., 2020; J. Wang et al., 2018; Grbovic et al., 2018; Okura et al., 2017). As an illustration, Spotify learns embedding representations of listening sessions by averaging pre-computed embeddings of the music tracks listened to during these sessions (Hansen et al., 2020). This service also computes “long-term” user embeddings, used for recommendation purposes, by averaging the session embeddings associated with each user. Deezer computes embeddings for several types of recommendable music collections, such as playlists and albums, by averaging embeddings of the music tracks present in these collections (Briand et al., 2021a). Yahoo averages embeddings of the news articles previously browsed by each user to represent them in the same embedding space as articles and provide personalized news recommendations (Okura et al., 2017). Alibaba averages side information embeddings, including category and brand embeddings, to obtain product embeddings for cold start recommendation on the Taobao e-commerce platform (J. Wang et al., 2018).

However, despite its prevalence, this averaging practice is often adopted without explicit justification from a theoretical standpoint. As we detail in Section 4.2, the rationale for averaging

item embeddings mainly stands in the simplicity and scalability of this approach (Covington et al., 2016b; Briand et al., 2021a). Yet, it is unclear to which extent averaging item embeddings guarantees to provide faithful user or higher-level concept representations for recommendation. For instance, assuming we represent a user by the average embedding of their previously consumed items, to what degree would the neighboring items of this average constitute relevant recommendations for this user? While the impact of averaging and other pooling operations has been studied in other fields, notably natural language processing (NLP) (Coleman, 2020; Arora et al., 2017), to our knowledge, the consistency of averaging operations remains relatively understudied in the specific context of a recommender system.

4.1.2 Contributions

In this chapter, we propose to investigate these important considerations, making the following contributions:

- Firstly, we define $\text{Consistency}_k(\mathcal{X})$, a general expected precision score introduced in this study to measure the consistency of an average embedding relative to the items it summarizes, from a recommendation standpoint.
- Secondly, we examine the consistency of averaging operations in a *theoretical* setting with general assumptions on item embeddings, providing an in-depth analysis of the expression of $\text{Consistency}_k(\mathcal{X})$ in this setting.
- Thirdly, we analyze the *empirical* behavior of this score on real-world data. Our experiments consider three variants of large-scale music track embeddings from Deezer data, two of them being publicly available (Briand et al., 2021a).
- Lastly, we discuss the discrepancies between our theoretical and empirical results, emphasizing that real-world averages are less consistent for recommendation. This discussion paves the way for future research to better align embedding representation of real-world data with our theoretical assumptions. Overall, we believe this study will be insightful for researchers and practitioners aiming to improve the faithfulness of average embeddings in their recommender systems.

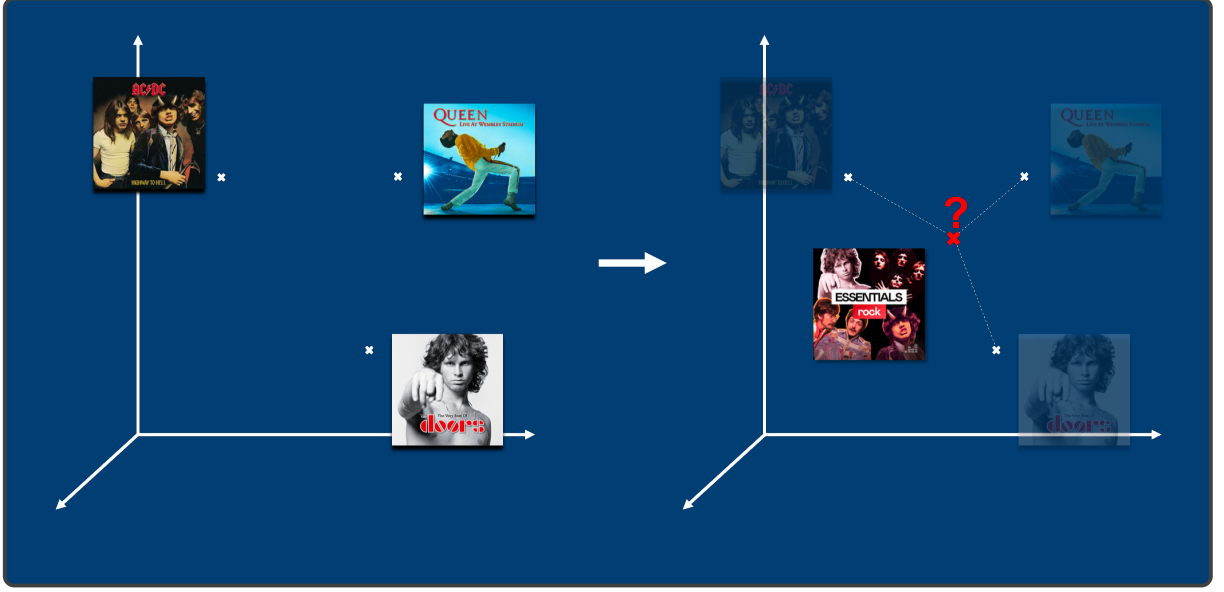


Figure 4.1: Representing a sequence of embeddings by its average is a common yet understudied practice.

4.2 Problem Formulation and Evaluation

4.2.1 Preliminaries

4.2.1.1 Mathematical Notation

Throughout this chapter, we consider a set $\mathcal{X} = \{X_i\}_{1 \leq i \leq N}$ of $N \in \mathbb{N}^*$ real-valued vectors of dimension $d \in \mathbb{N}^*$, i.e., $\forall i \in \{1, \dots, N\}, X_i = (X_{i,1}, X_{i,2}, \dots, X_{i,d})^\top \in \mathbb{R}^d$. We denote by $\mathcal{X}_k$ the set of subsets of $\mathcal{X}$ of cardinality k , for any $k \in \{1, \dots, N\}$. For any vector $z \in \mathbb{R}^d$ and $k \in \{1, \dots, N\}$, we define $\mathcal{X}_k(z) \in \mathcal{X}_k$ as the set of the k nearest neighbors¹ of z among the N elements of $\mathcal{X}$, according to some *similarity metric* $s : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, i.e.,

$$\mathcal{X}_k(z) = \arg \max_{\mathcal{Y} \in \mathcal{X}_k} \sum_{y \in \mathcal{Y}} s(z, y). \quad (4.1)$$

Moreover, for any subset $\mathcal{U} = \{u_i\}_{1 \leq i \leq k} \subseteq \mathcal{X}$ of k vectors, we denote the *center* or *average* of $\mathcal{U}$ as $\mu_{\mathcal{U}} = \frac{1}{k} \sum_{u_i \in \mathcal{U}} u_i$, and we denote the complementary set of $\mathcal{U}$ within $\mathcal{X}$ as $\overline{\mathcal{U}}$, i.e., $\overline{\mathcal{U}} = \mathcal{X} \setminus \mathcal{U} = \{X \in \mathcal{X}, X \notin \mathcal{U}\}$.

4.2.1.2 From Mathematics to Recommender Systems

In the context of an embedding-based recommender system (S. Zhang et al., 2019; Koren and Bell, 2015), $\mathcal{X}$ would correspond to a set of d -dimensional embedding representations associated with all recommendable *items* in a catalog, while $\mathcal{U} \in \mathcal{X}_k$ would correspond to the embeddings of a subset of k items. For instance, $\mathcal{X}$ could represent music tracks on a music streaming service, and

¹This set might not be unique if z is equidistant to several elements of $\mathcal{X}$, in which case $\mathcal{X}_k(z)$ can be drawn uniformly from all complying sets.



Figure 4.2: Averaging embeddings of songs from a playlist (**Ex 1**) or user’s listening history (**Ex 2**), to obtain a playlist or user embedding.

$\mathcal{U}$ could represent the tracks present in a playlist or album of length k (Hansen et al., 2020; Briand et al., 2021a). Alternatively, $\mathcal{X}$ could represent all products from an e-commerce platform and $\mathcal{U}$ the shopping cart of a user (J. Wang et al., 2018; Wu et al., 2017). At this stage, we do not make assumptions regarding the representation learning technique used to learn these embeddings.

4.2.2 Problem Formulation

4.2.2.1 Averaging Embeddings

As illustrated in Section 4.1.1, recommender systems frequently average item embeddings. Using the above notation, the action of averaging all embeddings of an item collection $\mathcal{U}$ to represent this collection translates to using $\mu_{\mathcal{U}}$ as the representation. As $\mu_{\mathcal{U}} \in \mathbb{R}^d$, one can interpret this vector as a new embedding in the same vector space as items, and, therefore, measure the similarity between $\mu_{\mathcal{U}}$ and other items using the similarity metric s . Average embeddings are faster and simpler to compute than alternatives such as neural aggregations (He, Liao, et al., 2017b; Hidasi et al., 2015). They have also been praised for their scalability, as they provide fixed-size representations independently of the number of items in $\mathcal{U}$ (Covington et al., 2016b; Briand et al., 2021a).

4.2.2.2 How Relevant is this Practice?

This chapter aims to rigorously investigate the consistency of this averaging practice. For instance, if we represent a user by the average embedding $\mu_{\mathcal{U}}$ of the items $\mathcal{U}$ this user has consumed or liked, do the items similar to $\mu_{\mathcal{U}}$ in the embedding space (according to s) also constitute items that the user would like? In the same way, if $\mu_{\mathcal{U}}$ summarizes an album composed of the tracks in $\mathcal{U}$, are tracks similar to $\mu_{\mathcal{U}}$ also similar to tracks in $\mathcal{U}$?

Intuitively, for $\mu_{\mathcal{U}}$ to be faithful to $\mathcal{U}$, we expect $\mu_{\mathcal{U}}$ to remain similar to the original items from $\mathcal{U}$. For instance, the user embedding should remain similar to the items the user has already liked, and the album embedding to the tracks contained in the album. For this reason, this chapter focuses on the following specific research question: *to which extent and under which conditions does $\mu_{\mathcal{U}}$ remain similar to items from $\mathcal{U}$, i.e., to the items used for its construction?*

4.2.3 Problem Evaluation : the Consistency Metric

4.2.3.1 Definition

To evaluate the consistency of $\mu_{\mathcal{U}}$ in accordance with our formulation in Section 4.2.2.2, we propose the following score:

$$\text{Consistency}_k(\mathcal{X}) = \mathbb{E}_{\mathcal{U} \in \mathcal{X}_k} [\text{Precision}_k(\mathcal{U})], \text{ where } \text{Precision}_k(\mathcal{U}) = \frac{|\mathcal{X}_k(\mu_{\mathcal{U}}) \cap \mathcal{U}|}{k} \text{ and } k \in \{1, \dots, N\}. \quad (4.2)$$

In essence, $\text{Precision}_k(\mathcal{U})$ measures the percentage of items from $\mathcal{U}$ among the k nearest neighbors of $\mu_{\mathcal{U}}$ in $\mathcal{X}$. A perfect precision of 1 indicates that $\mathcal{X}_k(\mu_{\mathcal{U}}) = \mathcal{U}$. Therefore, higher values of $\text{Consistency}_k(\mathcal{X})$ indicate that, on expectation, average embeddings computed using elements of $\mathcal{X}$ will comprise more items used for their construction in their neighborhood.

4.2.3.2 Interpretation

$\text{Consistency}_k(\mathcal{X})$ is akin to a score of expected precision where the ground truth to be predicted is the original input itself. This differs from the methodology used, for instance, to evaluate APC models in Chapter 3, where the ground truth is made of the next elements of the sequence, and is more similar to the training objective of an auto-encoder (Hinton et al., 2006). Our motivation for the use of this metric is the observation that average is used in a variety of scenarios that go beyond APC, and that to be robust to a wide range of downstream tasks, an aggregator should result in a representation of a set that allows **reconstructing the original set** as faithfully as possible. In fact, we believe that the initial motivation for using the average of a set of embeddings might stem from the property that it is always the point that minimizes the sum of squared Euclidean distances to the elements of the sets:

$$\mu_{\mathcal{U}} = \arg \min_{x \in \mathbb{R}^d} \sum_{u \in \mathcal{U}} \|x - u\|^2. \quad (4.3)$$

Using the average guarantees that the representation of a set stays close to every element of the set according to the Euclidean distance. In the same spirit, but pushing further, consistency measures to what extent the representation stays **closer** to elements of the set than to elements outside of the set, according to the similarity measure that is relevant for the problem at hand.

4.2.3.3 Distribution of Sequences

The notation $\mathbb{E}_{\mathcal{U} \in \mathcal{X}_k} [\cdot]$ assumes that the subset of items $\mathcal{U}$ is a random variable that is uniformly distributed on $\mathcal{X}_k$ (the set of subsets of $\mathcal{X}$ with size k). In the case of items being songs and subsets being playlists, this would mean that any collection of song is equally likely to be a playlist of a music streaming platform. This assumption is necessarily violated, as in reality $\mathcal{U}$ is distributed following an unknown data generating function P_{data} that defines how likely users are to create a playlist. However, if we had included the distribution of subsets $\mathcal{U}$ in our definition of consistency, any theoretical result would then have been problem specific. Instead, we chose to study consistency as defined in Equation (4.2) and interpret it as a measure for the robustness

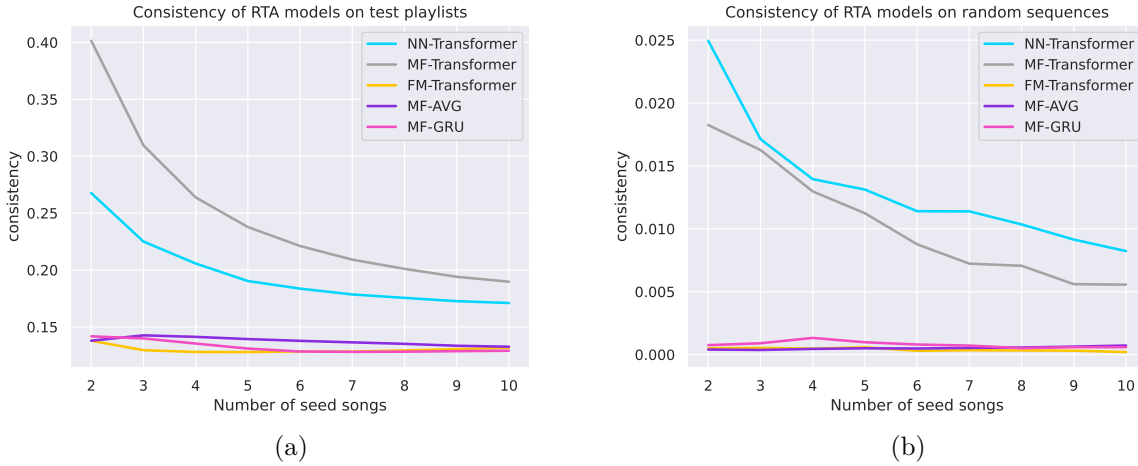


Figure 4.3: Average consistency score for RTA models, first (a) computed on playlists from the test set of Chapter 3, then (b) on random sequences of songs. By both evaluations of consistency, average is outperformed by Transformer-based models.

of the representation power of the average across tasks. To validate that results found under the assumption that $\mathcal{U}$ is uniformly distributed on $\mathcal{X}_k$ can be insightful for problems where this distribution changes, we computed the consistency of several aggregators trained in Chapter 3, first on the test set of 10 000 playlists, then on a set of playlists of same size artificially generated by uniformly sampling songs. Figure 4.3 shows that aggregator functions are ranked similarly when using the two computation methods, with average being a less consistent aggregator than the trained Transformers. This comforted us in a definition for consistency equal to the one of Equation (4.2), leading to the general theoretical results of Section 4.3.

4.2.3.4 Examples and intuitions

Before delving into the mathematical analysis of Section 4.3 for i.i.d embeddings with i.i.d dimensions, we propose to study a couple of examples of embedding distributions that should help understand what factors can influence the consistency, as defined in Equation (4.2), of a set of embeddings. In the following, we assume that s is the prevalent *inner product similarity*: $\forall (x, y) \in \mathbb{R}^d \times \mathbb{R}^d, s(x, y) = x^\top y = \sum_{i=1}^d x_i y_i$.

1-dimensional embeddings: An extreme scenario would be if each embedding were only a single scalar. Although we give this example mainly for its illustrative ability, one could imagine for instance that a recommender system could need to summarize the affinity of their users towards a specific dimension of musical content, for instance the release dates of the songs listened to or the energy they carry. In this situation, illustrated in Figure 4.4, the following proposition holds:

Proposition 4.1. If $d = 1$, $N > 2$ and all of the elements of $\mathcal{X}$ are distinct, then:

$$\forall k \in \llbracket 2, N - 1 \rrbracket, \text{Consistency}_k(\mathcal{X}) < 1 \quad (4.4)$$

Proof. If $d = 1$, then all elements of $\mathcal{X}$ are scalars. Let $x_{\min}$ and $x_{\max}$ be respectively the smallest and greatest (distinct) elements of $\mathcal{X}$. Since $N > 2$ and all of the elements of $\mathcal{X}$ are distinct, then $\forall k \in \llbracket 2, N - 1 \rrbracket, \exists \mathcal{U}_k \in \mathcal{X}_k$ such that $x_{\min} \in \mathcal{U}_k$, $x_{\max} \in \mathcal{U}_k$ and $\overline{\mathcal{U}_k} \neq \emptyset$. For k fixed, let's call $\mu_{\mathcal{U}_k}$ the average of $\mathcal{U}_k$.

We will now show that the N -th nearest neighbour of $\mu_{\mathcal{U}_k}$ is always an element of $\{x_{\min}, x_{\max}\}$. Indeed, for any $x \in \mathcal{X} \setminus \{x_{\min}, x_{\max}\}$:

$$\begin{aligned} s(\mu_{\mathcal{U}_k}, x) - s(\mu_{\mathcal{U}_k}, x_{\min}) &= \langle \mu_{\mathcal{U}_k}, x - x_{\min} \rangle \\ &= \mu_{\mathcal{U}_k}(x - x_{\min}) \end{aligned} \quad (4.5)$$

which is a real number with the same sign as $\mu_{\mathcal{U}_k}$. On the contrary, $s(\mu_{\mathcal{U}_k}, x) - s(\mu_{\mathcal{U}_k}, x_{\max})$ has the opposite sign of $\mu_{\mathcal{U}_k}$. Since this is true for all $x \in \mathcal{X} \setminus \{x_{\min}, x_{\max}\}$, this necessarily means that :

$$\begin{cases} \text{if } \mu_{\mathcal{U}_k} > 0 & \text{then } x_{\min} \text{ is the } N\text{-th neighbor of } \mu_{\mathcal{U}_k}, \\ \text{if } \mu_{\mathcal{U}_k} < 0 & \text{then } x_{\max} \text{ is the } N\text{-th neighbor of } \mu_{\mathcal{U}_k}. \end{cases} \quad (4.6)$$

From the definition of consistency of Equation (4.2), this directly implies that:

$$\forall k \in \llbracket 2, N - 1 \rrbracket, \text{Consistency}_k(\mathcal{X}) < 1. \quad (4.7)$$

□

What proposition 4.1 and its proof teach us is that for the extreme case of 1-dimensional embeddings, averaging a set of very different values, such as the minimal and the maximal one, leads to a representation that can not remain close to every element, and even worse, it ends up being similar to intermediate values that did not belong to the original set.

This behavior is perhaps the most intuitive one as it can be visually observed when studying embeddings in 1 or 2 dimensions. When averaging heterogeneous sets of items, the result ends up in a middle position at the risk that other unrelated items are already present in the same area. This behavior of low-dimensional embeddings might be the root cause for some distrust towards recommender systems ability to adapt to users with unusual taste.

Indeed, imagine for instance that the elements of $\mathcal{X}$ each represent a different musical genre, spread along a single axis indicating their popularity. In this scenario, the leftmost embedding $x_{\min}$ could represent for instance classical music, while the rightmost embedding $x_{\max}$ could represent pop music. Suppose that a user liked these two music genres specifically and no other. Representing that user on the genre popularity axis by averaging their two liked genres would place them somewhere in the middle, where other music genres are also located, which would be misleading as it makes it impossible to retrieve the user's favorite genres.

This dramatic scenario isn't specific to 1-dimensional embeddings. A similar behavior, although less extreme, could be observed if a high number of musical genres were represented in two dimensions; averaging dissimilar embeddings would create an aggregate representation lying in an area where other musical genres are already represented. Fortunately, we prove in Section 4.3 that this problem can be alleviated by the use of higher-dimensional embeddings. To illustrate this, let's first study another embedding configuration.

One-hot encoding: One-hot encoding is the case when:

- $d = N$
- $\forall (i, j) \in \llbracket 1, N \rrbracket^2, \langle X_i, X_j \rangle = \delta_{ij}$

with $\delta_{i,j}$ the Kronecker delta defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \quad (4.8)$$

The following proposition holds:

Proposition 4.2. If $\mathcal{X}$ is the one-hot representation of the data, then:

$$\forall k \in \llbracket 2, N - 1 \rrbracket, \text{Consistency}_k(\mathcal{X}) = 1. \quad (4.9)$$

Proof. Let $k \in \llbracket 2, N - 1 \rrbracket$. For any $\mathcal{U}_k \in \mathcal{X}_k$, let's call $\mu_{\mathcal{U}_k} = \sum_{X \in \mathcal{U}_k} \frac{X}{k}$ the average of its elements. Then

$$\forall X \in \mathcal{X}, \langle \mu_{\mathcal{U}_k}, X \rangle = \begin{cases} \frac{1}{k} & \text{if } X \in \mathcal{U}_k, \\ 0 & \text{otherwise.} \end{cases} \quad (4.10)$$

The above equation means that the k -nearest neighbours of $\mu_{\mathcal{U}_k}$ are exactly the elements of $\mathcal{U}_k$. Since this is true for every $\mathcal{U}_k \in \mathcal{X}_k$, we thus proved that:

$$\forall k \in \llbracket 2, N - 1 \rrbracket, \text{Consistency}_k(\mathcal{X}) = 1. \quad (4.11)$$

□

Proposition 4.2 states a somewhat surprising result; if N items are represented with one-hot encoding, then the average of a subset of k items is guaranteed to have exactly that same subset as its k -nearest neighbors. Moreover, this holds for any natural integer N greater than 2, and any k smaller than N . This contrasts with Proposition 4.1 and the intuition built from observing low-dimensional embeddings.

Taking again the example of musical genres, Proposition 4.2 implies that a user who only enjoys pop and classical music can safely be represented by the average of their two (one-hot encoded) embeddings, since the resulting vector can be used to retrieve exactly these two original genres using a nearest neighbour search. One might wonder, given than the embeddings of pop and

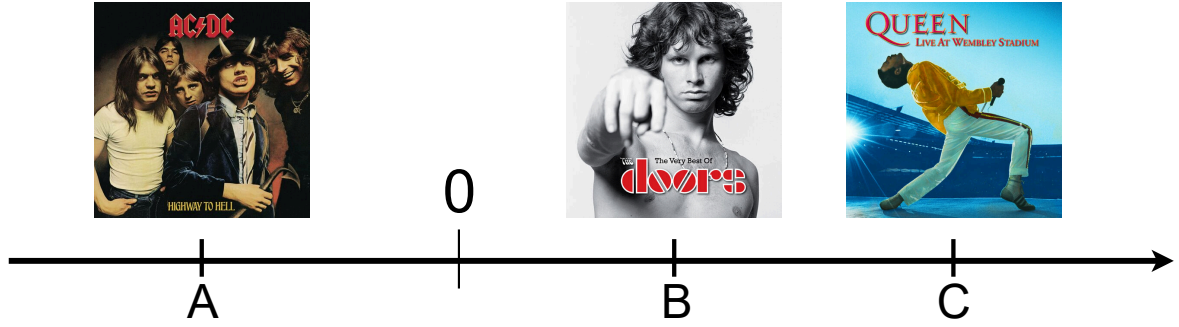


Figure 4.4: Illustration of 3 embeddings of dimension 1. The middle point between A and C necessarily ends too close to B.

classical music in this example have a similarity of zero, how their average does manage to stay close to both of them. The answer, as stated by Equation 4.10, is that it doesn't. Indeed, the average vector only has a low similarity with the elements of the initial sequence. However it suffices for them to be nearest neighbours, since the rest of the item set has a similarity of exactly zero. This surprising observation can be understood as a (fortunate) example of the *curse of dimensionality*. Since one-hot representations lie in $\mathbb{R}^N$, and there are "only" N items, this means that the space is mainly empty of embeddings. Hence when the average of several embeddings is computed, it does not fall close to an unrelated item just out of "bad luck" as it does in low-dimensions.

This example illustrates how intuitions based on visual examples can be misleading, as they do not always generalize to mathematical objects of higher dimension. Note however that increasing the dimension of embeddings is not a sufficient condition, if their configuration does not make use of the additional space. This is illustrated in Figure 4.5, where a set of points in the 2 dimensional plan all lie in the same straight line. Of course, embedding techniques were developed for recommendation to alleviate the issues encountered when using one-hot encoding, and a typical value for d in real-world embeddings lies between a few dozens up to a thousand. This raises the interesting research question of the behavior of averages of embeddings in this span of values, especially given that the above analysis of very low and very high values have outlined perfectly opposite behaviors. We develop a partial answer to this question in the next section, where we consider the case of realistic values of d , and instead of assuming specific layouts as we did in the examples above, we adopt a probabilistic approach where embeddings are independent identically distributed (i.i.d) random vectors.

The remainder of this chapter provides an analysis of $\text{Consistency}_k(\mathcal{X})$ in a theoretical setting with assumptions on the distribution of embeddings in $\mathcal{X}$, and studies its empirical behavior on real-world embeddings.

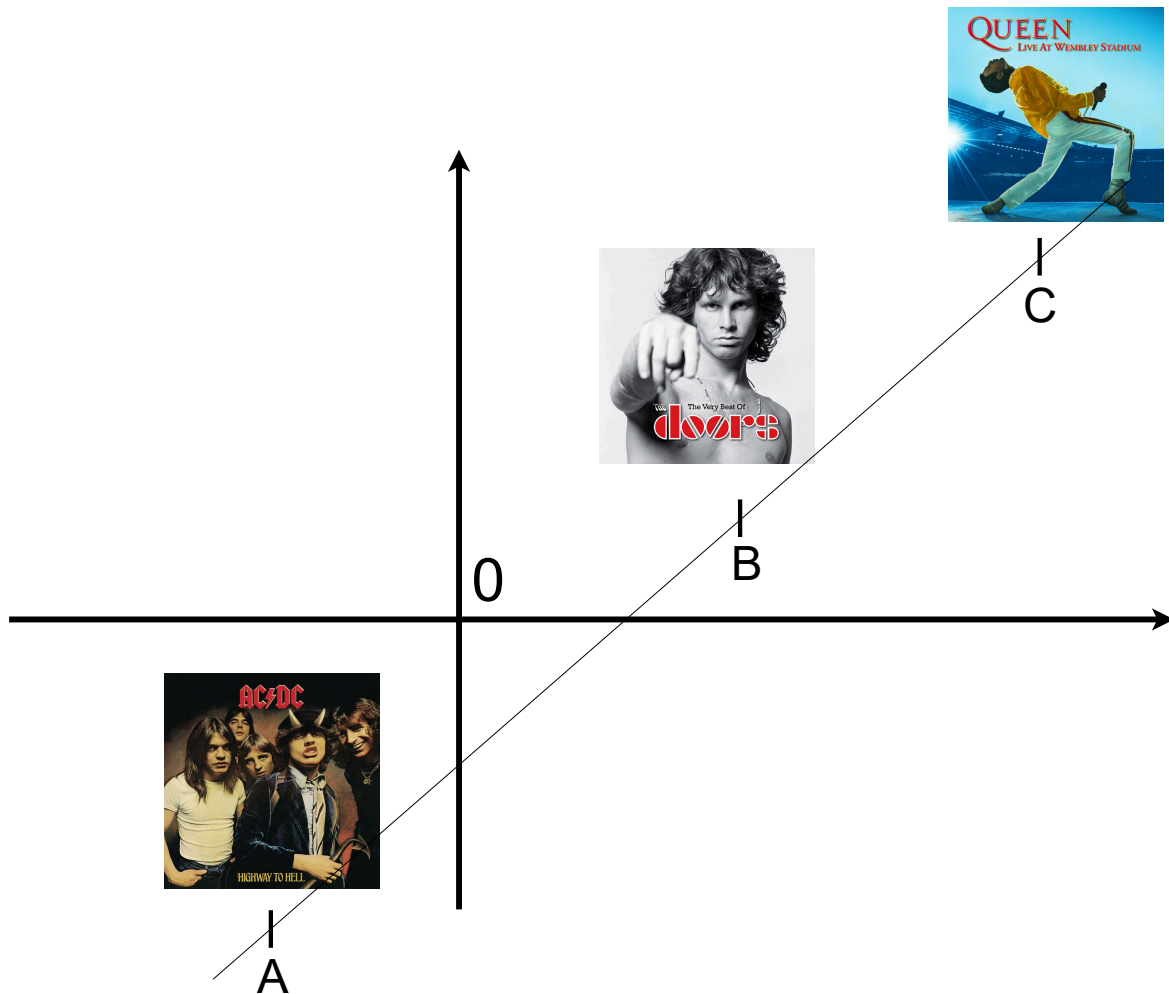


Figure 4.5: Illustration of 3 embeddings of dimension 2, all lying in the same 1-dimensional subspace. The Consistency metric for this set of embeddings is the one as the set of Figure 4.4

4.3 A Theoretical Analysis of Consistency_k($\mathcal{X}$)

We dedicate Section 4.3 to our theoretical analysis of Consistency_k($\mathcal{X}$). For clarity, we only present our setting, main results, and interpretation of these results in this section. We report all mathematical proofs in Appendices 4.A and 4.B.

4.3.1 Setting and Assumptions

We focus on the setting where $\mathcal{X}$ is a set of independent and identically distributed (i.i.d.) multi-dimensional random variables (r.v.). For every $X_i \in \mathcal{X}$, the elements $\{X_{i,j}\}_{1 \leq j \leq d}$ form a set of i.i.d uni-dimensional r.v. and we denote by μ , σ^2 , γ , and κ the mean, variance, skewness, and kurtosis of their distribution, respectively. We assume that these moments are finite. s is the prevalent *inner product similarity*: $\forall (x, y) \in \mathbb{R}^d \times \mathbb{R}^d, s(x, y) = x^\top y = \sum_{i=1}^d x_i y_i$.

4.3.2 Main Results and Interpretations

In this theoretical setting, we obtain the following results on average embeddings.

Proposition 4.3. Let $k \in \{2, \dots, N\}$, $\mathcal{U} \in \mathcal{X}_k$, $u_{\text{in}} \in \mathcal{U}$, and $u_{\text{out}} \in \bar{\mathcal{U}}$. Then, under the hypotheses of Section 4.3.1, both $s_{\text{in}} = s(u_{\text{in}}, \mu_{\mathcal{U}})$ and $s_{\text{out}} = s(u_{\text{out}}, \mu_{\mathcal{U}})$ converge in probability to normal distributions as d increases. Their respective means and variances are provided in the proof of Appendix 4.A. Moreover, under such distributions, we have:

$$\mathbb{P}(s(u_{\text{in}}, \mu_{\mathcal{U}}) > s(u_{\text{out}}, \mu_{\mathcal{U}})) = \frac{1}{2} \left(1 + \text{erf} \left(\sqrt{\frac{d\sigma^2}{2((2(k-1) + \kappa)\sigma^2 + 2k\gamma\mu\sigma + 2k^2\mu^2)}} \right) \right), \quad (4.12)$$

where erf denotes the Gauss error function (Andrews, 1998): $\text{erf} : x \mapsto \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

Proposition 4.3 is a consequence of the *Central Limit Theorem* (CLT) (Fischer, 2011) applied to inner product similarities. Figure 4.7 illustrates that, for dimension values such as $d = 128$ (a common choice in recommendation applications), the normal distribution faithfully approximates the distribution of similarities, regardless of the original embedding distribution. Hence, Equation (4.12) provides a reliable approximation of the probability that an element from $\mathcal{U}$ will be closer to its center $\mu_{\mathcal{U}}$ than a random point from $\bar{\mathcal{U}}$. It is worth noting that the probability is guaranteed to be greater than 0.5, as the expression within the erf function is positive. We also observe that the probability increases with d , illustrating that, as the dimension increases, the chances for a vector from $\bar{\mathcal{U}}$ to have a stronger similarity with $\mu_{\mathcal{U}}$ than elements of $\mathcal{U}$ diminishes.

The case of centered embeddings ($\mu = 0$) is of particular interest. Indeed, for $\mu = 0$ and a fixed d , the probability becomes independent of σ and γ , and a decreasing function of k and κ . In essence, as k increases, it becomes increasingly difficult for $\mu_{\mathcal{U}}$ to remain similar to all $\mathcal{U}$ items, while remaining dissimilar to all $\bar{\mathcal{U}}$ items. The decrease with respect to the kurtosis κ relates to its interpretation as a measure of the propensity of a distribution to produce outliers (Westfall, 2014). Intuitively, the presence of outliers would impact the ability for $\mu_{\mathcal{U}}$ to remain similar to

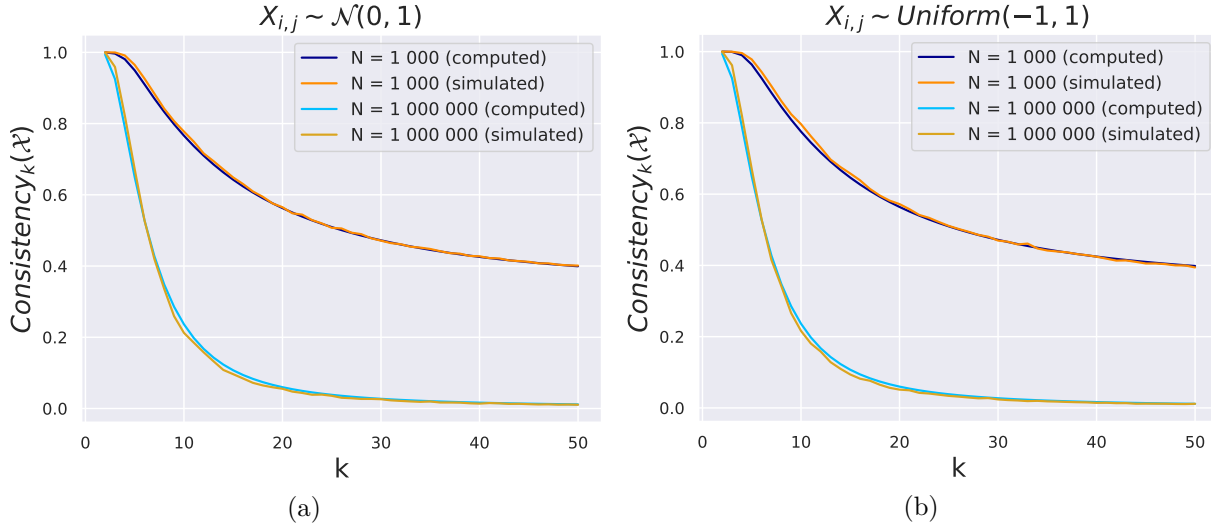


Figure 4.6: For these distributions, for small values of k , $\text{Consistency}_k(\mathcal{X})$ remains close to 1 even for $N = 1\,000\,000$. Therefore, even with a large catalog of millions of items, averaging item embeddings appears as a consistent way to faithfully represent collections of a few items.

all $\mathcal{U}$ items and dissimilar to all $\overline{\mathcal{U}}$ items². In the following, we continue to focus on $\mu = 0$, using results from Proposition 4.3 to express $\text{Consistency}_k(\mathcal{X})$.

Proposition 4.4. Under the hypotheses of Section 4.3.1 and approximated distributions of Proposition 4.3 with $\mu = 0$:

$$\text{Consistency}_k(\mathcal{X}) = \frac{1}{k} \sum_{i=1}^k \int_{-\infty}^{\infty} f_{\text{in},(i)}(x) \times F_{\text{out},(k-i+1)}(x) \, dx, \quad (4.13)$$

where $f_{\text{in},(i)}$ is the *Probability Density Function* (PDF) of the i^{th} order statistic (Nagaraja et al., 2003) of the set of random variables $\{s(u_{\text{in}}, \mu_{\mathcal{U}}), u_{\text{in}} \in \mathcal{U}\}$ and $F_{\text{out},(k-i+1)}$ is the *Cumulative Distribution Function* (CDF) of the $(k - i + 1)^{\text{th}}$ order statistic of the set of random variables $\{s(u_{\text{out}}, \mu_{\mathcal{U}}), u_{\text{out}} \in \overline{\mathcal{U}}\}$. Explicit formulas for $f_{\text{in},(i)}$ and $F_{\text{out},(k-i+1)}$ are provided in the proof of Appendix 4.B.

Proposition 4.4 provides a useful approximated analytical expression of $\text{Consistency}_k(\mathcal{X})$. In Figure 4.6, we assess the accuracy of this expression by comparing, for various $X_{i,j}$ distributions and values of k , the $\text{Consistency}_k(\mathcal{X})$ score obtained from Equation (4.13) - the values of the integrals are computed using the Python library `scipy` - to the one estimated via Monte-Carlo simulations. Our computed expression systematically coincides with the simulated score, validating the correctness of Proposition 4.4 and the relevance of our approximations.

Overall, we observe that $\text{Consistency}_k(\mathcal{X})$ decreases with k and the number of items $N = |\mathcal{X}|$. Regarding k , this result is coherent with our above interpretation of Proposition 4.3. Regarding

²On the contrary, Equation (4.12) would be maximized by distributions with minimal kurtosis, such as a re-centered Bernoulli(0.5) or a Rademacher distribution ($X_{i,j} = 1$ or -1 with probability 0.5 each). We emphasize that these distributions are *discrete*, and might therefore allow for less precise similarity computations between vectors. This brings to light an interesting *trade-off* between similarity precision and average embedding consistency.

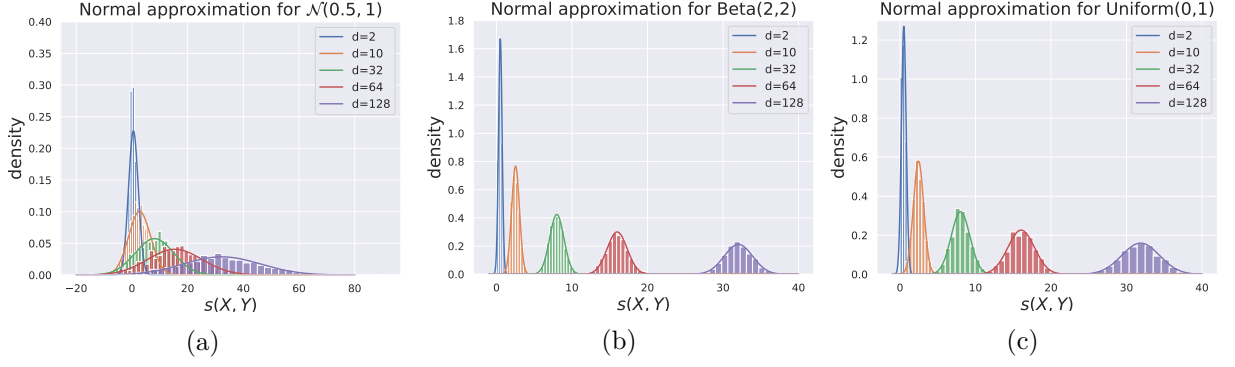


Figure 4.7: Histograms of inner product similarity values between 1 000 d -dimensional vectors with entries randomly drawn from $\mathcal{N}(0.5, 1)$ (Figure 4.7a), Beta(2, 2) (Figure 4.7b), or Uniform(0, 1) (Figure 4.7c) distributions, with $d \in \{2, 10, 32, 64, 128\}$. Histograms undergo a rightward shift as d increases, since similarity computations involve summing more elements (see Section 4.3.1). Curves correspond to normal approximations of similarity distributions. We observe that, for the largest values of d , the normal distribution faithfully approximates all similarity distributions, an important result to validate the approximations of Propositions 4.3 and 4.4.

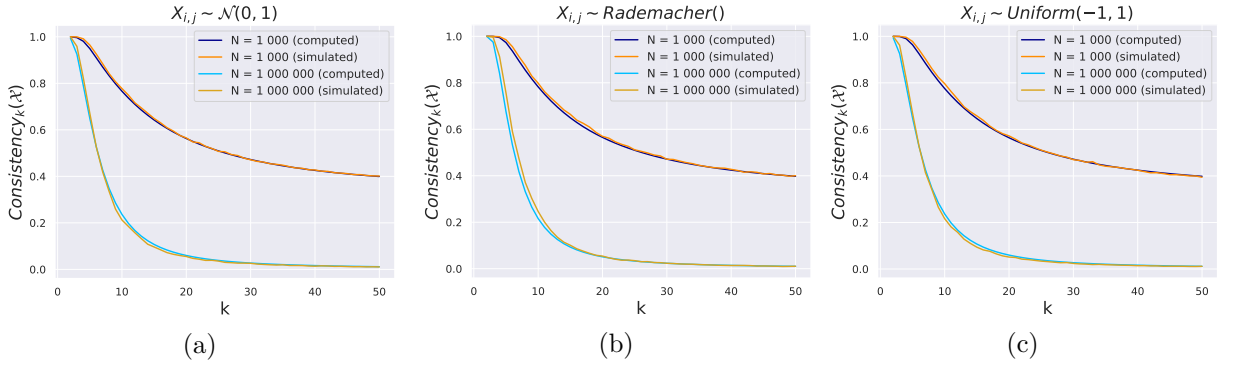


Figure 4.8: Comparison of the Consistency $_k(\mathcal{X})$ scores obtained by a direct *computation* of Equation (4.13) to the ones estimated via numerical *simulations*, for $k \in \{2, \dots, 50\}$, with $d = 128$, $N = 1\,000$ or $1\,000\,000$, and $X_{i,j} \sim \mathcal{N}(0, 1)$ (Figure 4.8a), $X_{i,j} \sim \text{Rademacher}()$ (Figure 4.8b), or $X_{i,j} \sim \text{Uniform}(-1, 1)$ (Figure 4.8c). We used the Python library *scipy* to compute integrals from Equation (4.13). For numerical simulations, we sampled N vectors from the above three distributions. Then, for each k , we randomly picked a subset of k vectors and computed the precision of this subset (Equation 4.2). We repeated this operation 1 000 times for each value of k and reported averaged scores on figures.

N , such a result is also unsurprising. Indeed, the more vectors in $\mathcal{X}$, the greater the likelihood of some unrelated item embeddings becoming similar to $\mu_{\mathcal{U}}$ by chance.

Yet, in Figure 4.6, all scores remain quite high for $N = 1\,000$ (e.g., around 0.4 for $k = 50$). Importantly, for small values of k , the consistency of average embeddings remains close to 1 even for $N = 1\,000\,000$. Therefore, even with a large catalog of millions of items, as in music streaming services, averaging item embeddings appears as a consistent way to faithfully represent collections of a few items, provided that these embeddings comply with our theoretical assumptions.

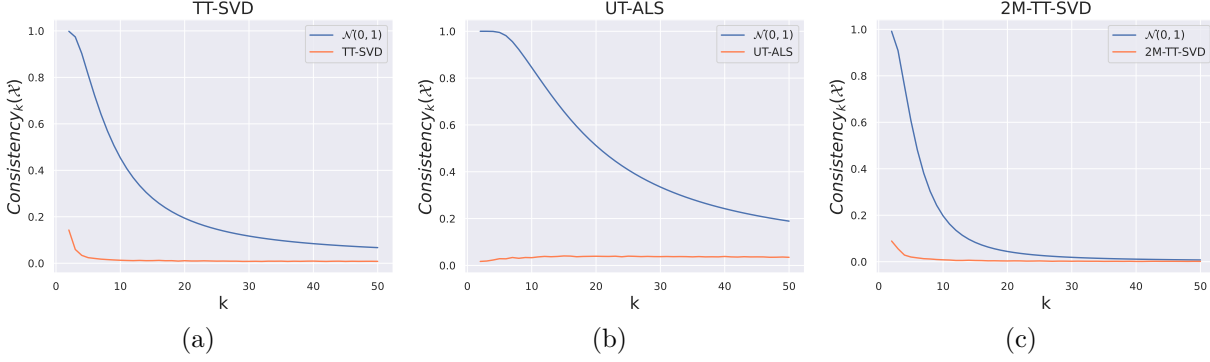


Figure 4.9: $\text{Consistency}_k(\mathcal{X})$ scores of centered versions of TT-SVD (Figure 4.9a), UT-ALS (Figure 4.9b), and 2M-TT-SVD embeddings (4.9c), for $k \in \{2, \dots, 50\}$ and the inner product similarity s . For comparison, figures also display the $\text{Consistency}_k(\mathcal{X})$ scores of embeddings generated from normal distributions, with the same dimension and number of items as the real-world embeddings under consideration. Average embeddings of real-world data are less consistent than those of data complying with our theoretical setting from Section 4.3.

4.4 From Theory to Practice

4.4.1 Experimental Setting

In this section, we analyze the empirical behavior of $\text{Consistency}_k(\mathcal{X})$ on real-world data, with the aim of discussing potential discrepancies with our results from Section 4.3. Our experiments focus on three *music track embeddings* datasets. As illustrated in this chapter, music recommendation is especially prone to high-order concept learning: album, playlist, session, music genre, and user embeddings can all be obtained by averaging music track embeddings (Briand et al., 2021a; Hansen et al., 2020; Schedl et al., 2018; Bendada, Salha-Galvan, Bouabça, et al., 2023).

Firstly, we consider two variants³ of 50 000 music track embeddings publicly released by Deezer (Briand et al., 2021a). The first ones, denoted *TT-SVD embeddings*, consist of 128-dimensional vectors obtained by factorizing a track-track (TT) pointwise mutual information matrix computing track co-occurrences in Deezer playlists, using singular value decomposition (SVD) (Briand et al., 2021a). The second ones, denoted *UT-ALS embeddings*, are 256-dimensional vectors obtained by factorizing a user-track (UT) interaction matrix, using alternating least squares (ALS) (Briand et al., 2021a).

Besides, we report results on *2M-TT-SVD embeddings*, a private dataset of two million 128-dimensional track embeddings, extracted from Deezer’s production environment. These embeddings are computed using SVD on a co-occurrence matrix comparable to the TT-SVD one. We voluntarily omit technical details for confidentiality reasons.

4.4.2 Results and Discussion

Figure 4.9 reports our evaluation of $\text{Consistency}_k(\mathcal{X})$ scores for the three music track embeddings under consideration. Our experiments show that average embeddings of real-world data are less consistent for recommendation than those computed from embedding data explicitly complying with our theoretical setting from Section 4.3.

Specifically, one can not ensure that the average embedding of a collection of items will remain similar to the items present in this collection. Even for low values of k , the consistency of average embeddings does not surpass 14%, 6%, and 2 % for TT-SVD, 2M-TT-SVD, and UT-ALS embeddings, respectively. Consequently, even the average of two randomly selected embeddings would likely result in a vector whose two most similar neighbors will not be the selected embeddings themselves.

Regarding SVD-based embeddings (TT-SVD, 2M-TT-SVD), we observe that consistency scores drop as k increases, as in Figure 4.8. On the contrary, scores remain steady at around 2% for UT-ALS embeddings. This phenomenon highlights the dissimilarity in distributions of embeddings generated by different representation learning algorithms (SVD, ALS). In particular, the steady consistency of UT-ALS suggests that these embeddings might be more suitable for downstream applications involving average operations on large collections, while TT-SVD embeddings might be preferable for applications with low values of k , although more studies would be required in future work for confirmation.

Overall, our experiments also pave the way for future work to align real-world embeddings with the setting from Section 4.3, to improve the consistency of real-world averages. For instance, one could consider alternating SVD or ALS matrix reconstruction optimization steps with projections within the set of distributions complying with assumptions from Section 4.3. One could also examine adding a regularization term to the optimized loss during training, e.g., the Kullback-Leibler divergence of embeddings with a pre-selected complying distribution. In particular, these strategies could help to enforce the identical distribution of embedding dimensions. In addition, we note that, in Figure 4.9, we computed scores on *centered* embeddings. Besides being in line with Section 4.3, our tests revealed that this centering operation slightly improves the consistency of TT-SVD, UT-ALS, and 2M-TT-SVD average embeddings (albeit modifying initial similarities). Our future research will aim to further understand the impact of centering embeddings on consistencies.

4.5 Conclusion

This chapter proposed a rigorous study of the common practice consisting of averaging item embeddings in recommender systems. We provided a mathematical analysis of the consistency of these averaging operations in a general theoretical setting, as well as an empirical evaluation on real-world data. Our results revealed that real-world averages were less consistent than those computed in our theoretical setting. This sets the stage for future research directions, discussed

³We release our source code on GitHub to ensure the reproducibility of our experimental analysis on these two Deezer public datasets: <https://github.com/deezer/consistency>.

in this chapter, toward better aligning real-world data with our theoretical assumptions. Due to the prevalence of the embedding averaging practice in industry-oriented research and applications, we believe our study and proposed directions will be insightful for the recommendation community, and could eventually lead to the improvement of embedding-based recommender systems leveraging average embeddings for representation learning.

Appendix

In this supplementary section, we report the mathematical proofs of our Propositions 4.3 and 4.4 from Section 4.3.

4.A Proof of Proposition 4.3

4.A.1 Preliminaries

Let X , Y , and Z be d -dimensional random variables (r.v.) composed of d independent and identically distributed (i.i.d.) uni-dimensional r.v. as elements. Distributions might differ between X, Y , and Z . Let $\mu_X, \sigma_X^2, \gamma_X, \kappa_X$ be the finite mean, variance, skewness, and kurtosis of the distribution of elements of X , respectively. Let $\mu_Y, \sigma_Y^2, \mu_Z, \sigma_Z^2$ be the finite mean and variance of the elements of Y and Z , respectively. When d increases, the following approximations hold:

$$\begin{aligned} s(X, Y) &\sim \mathcal{N}(\mu_{XY}, \sigma_{XY}^2), \\ s(X, X) &\sim \mathcal{N}(\mu_{XX}, \sigma_{XX}^2), \end{aligned} \tag{4.14}$$

with:

$$\begin{aligned} \mu_{XY} &= d \times \mu_X \mu_Y, \\ \sigma_{XY}^2 &= d \times ((\sigma_X^2 + \mu_X^2)(\sigma_Y^2 + \mu_Y^2) - \mu_X^2 \mu_Y^2), \\ \mu_{XX} &= d \times (\mu_X^2 + \sigma_X^2), \\ \sigma_{XX}^2 &= d \times (4\mu_X^2 \sigma_X^2 + 4\mu_X \gamma_X \sigma_X^3 + (\kappa_X - 1)\sigma_X^4). \end{aligned}$$

Indeed, we have $s(X, Y) = \sum_{i=1}^d X_i Y_i$. Each r.v. $X_i Y_i$ verifies $\mathbb{E}[X_i Y_i] = \mu_X \mu_Y$ and $\text{Var}(X_i Y_i) = (\sigma_X^2 + \mu_X^2)(\sigma_Y^2 + \mu_Y^2) - \mu_X^2 \mu_Y^2$. $s(X, Y)$ being the sum of d i.i.d. r.v., according to the central limit theorem (Fischer, 2011), it can be approximated by a normal distribution of expectation $d \times \mathbb{E}[X_i Y_i]$ and variance $d \times \text{Var}(X_i Y_i)$ for large values of d , leading to the approximation for $s(X, Y)$ in Equation (4.14). A similar reasoning leads to the approximation for $s(X, X)$ in this same Equation (4.14).

Moreover, regarding covariances of these similarities, we have:

$$\begin{aligned} \text{Cov}(s(X, Y), s(X, Z)) &= d \times \sigma_X^2 \mu_Y \mu_Z, \\ \text{Cov}(s(X, X), s(X, Y)) &= d \times \mu_Y (\gamma_X \sigma_X^3 + 2\mu_X \sigma_X^2). \end{aligned} \tag{4.15}$$

Indeed:

$$\begin{aligned}
 \text{Cov}(s(X, Y), s(X, Z)) &= \mathbb{E}[s(X, Y)s(X, Z)] - \mathbb{E}[s(X, Y)]\mathbb{E}[s(X, Z)] \\
 &= \mathbb{E}\left[\left(\sum_{i=1}^d X_i Y_i\right)\left(\sum_{j=1}^d X_j Z_j\right)\right] - d \times \mu_X \mu_Y \times d \times \mu_X \mu_Z \\
 &= \mathbb{E}\left[\sum_{i=1}^d (X_i Y_i X_i Z_i + \sum_{\substack{j=1 \\ j \neq i}}^d X_i Y_i X_j Z_j)\right] - d^2 \times \mu_X^2 \mu_Y \mu_Z \\
 &= \sum_{i=1}^d (\mathbb{E}[X_i^2] \mathbb{E}[Y_i] \mathbb{E}[Z_i] + \sum_{\substack{j=1 \\ j \neq i}}^d (\mathbb{E}[X_i] \mathbb{E}[Y_i] \mathbb{E}[X_j] \mathbb{E}[Z_j])) - d^2 \times \mu_X^2 \mu_Y \mu_Z \\
 &= d \times ((\sigma_X^2 + \mu_X^2) \mu_Y \mu_Z + (d-1)(\mu_X^2 \mu_Y \mu_Z)) - d^2 \times \mu_X^2 \mu_Y \mu_Z \\
 &= d \times \sigma_X^2 \mu_Y \mu_Z.
 \end{aligned}$$

We obtain the second covariance of Equation (4.15) with similar computations. Notice how $\mu_Y = 0$ or $\mu_Z = 0$ implies that similarities are uncorrelated and, therefore, also independent when they follow multivariate normal distributions (Jacod et al., 2004).

4.A.2 Distribution of $s_{\text{in}} = s(u_{\text{in}}, \mu_{\mathcal{U}})$ and $s_{\text{out}} = s(u_{\text{out}}, \mu_{\mathcal{U}})$

Using the distributive property of the inner product s , we have:

$$s_{\text{out}} = \frac{\sum_{i=1}^k s(u_{\text{out}}, u_i)}{k},$$

i.e., the sum of k correlated identically distributed normal distributions. The expectation of the sum is thus:

$$\mathbb{E}[s_{\text{out}}] = d \times \mu^2.$$

We compute its variance using Bienaymé's identity (Loève, 1977) stating that, for k r.v. $(A_i)_{1 \leq i \leq k}$, then:

$$\text{Var}\left(\sum_{i=1}^k A_i\right) = \sum_{i=1}^k \text{Var}(A_i) + \sum_{\substack{i,j=1 \\ i \neq j}}^k \text{Cov}(A_i, A_j) = \sum_{i,j=1}^k \text{Cov}(A_i, A_j).$$

Also considering $A_i = s(u_{\text{out}}, u_i)$, and using the first half of both Equations (4.14) and (4.15), we obtain:

$$\begin{aligned}
 \text{Var}(s_{\text{out}}) &= \frac{kd(\sigma^4 + 2\sigma^2\mu^2) + k(k-1)d\sigma^2\mu^2}{k^2} \\
 &= d \frac{(\sigma^4 + (k+1)\sigma^2\mu^2)}{k}.
 \end{aligned}$$

Besides, under preliminary approximations:

$$s_{\text{in}} = \frac{s(u_{\text{in}}, u_{\text{in}}) + \sum_{i \neq \text{in}} s(u_{\text{in}}, u_i)}{k}$$

is the sum of k correlated normal distributions, i.e., a normal distribution with:

$$\mathbb{E}[s_{\text{in}}] = d(\mu^2 + \frac{\sigma^2}{k}).$$

Using Bienaymé's identity and Equations (4.14) and (4.15), we get:

$$\begin{aligned} \text{Var}(s_{\text{in}}) &= d \times \frac{4\mu^2\sigma^2 + 4\mu\gamma\sigma^3 + (\kappa - 1)\sigma^4 + (k - 1)(\sigma^4 + 2\sigma^2\mu^2) + 2(k - 1)\mu(\gamma\sigma^3 + 2\mu\sigma^2) + (k - 1)(k - 2)\sigma^2\mu^2}{k^2} \\ &= \frac{d \times (k(k + 3)\mu^2\sigma^2 + 4\mu\gamma\sigma^3 + (\kappa + k - 2)\sigma^4)}{k^2}. \end{aligned}$$

4.A.3 Distribution of $s_{\text{diff}} = s_{\text{in}} - s_{\text{out}}$

Using preliminary approximations, and since the difference of two normally distributed r.v. is also normally distributed (Jacod et al., 2004), we obtain that $s_{\text{diff}} = s(u_{\text{in}}, \mu_{\mathcal{U}}) - s(u_{\text{out}}, \mu_{\mathcal{U}})$ follows a normal distribution with:

$$\begin{aligned} \mathbb{E}[s_{\text{diff}}] &= \mathbb{E}[s(u_{\text{in}}, \mu_{\mathcal{U}})] - \mathbb{E}[s(u_{\text{out}}, \mu_{\mathcal{U}})] \\ &= d \frac{\sigma^2}{k}. \end{aligned}$$

To compute its variance, we remark that:

$$\begin{aligned} s_{\text{diff}} &= s(u_{\text{in}}, \mu_{\mathcal{U}}) - s(u_{\text{out}}, \mu_{\mathcal{U}}) \\ &= \frac{1}{k} (s(u_{\text{in}}, u_{\text{in}}) + s(u_{\text{in}}, \sum_{j \neq \text{in}}^k u_j) - s(u_{\text{out}}, u_{\text{in}}) - s(u_{\text{out}}, \sum_{j \neq \text{in}} u_j)) \\ &= \frac{s(X, X) + s(X, Z) + s(Y, X) + s(Y, Z)}{k}, \end{aligned}$$

with $X = u_{\text{in}}$, $Y = -u_{\text{out}}$ and $Z = \sum_{j \neq \text{in}} u_j$. Since X , Y , and Z are three independent multidimensional normal distributions, for which dimensions are i.i.d. with expectation μ , $-\mu$, $(d - 1)\mu$ and variance σ^2 , σ^2 , $(d - 1)^2\sigma^2$, we once again use Bienaymé's identity with Equations (4.14) and (4.15) to obtain:

$$\text{Var}(s_{\text{diff}}) = d \frac{(2(k - 1) + \kappa)\sigma^4 + 2k\gamma\mu\sigma^3 + 2k^2\sigma^2\mu^2}{k^2}.$$

Finally, using the cumulative distribution function of normal distributions (Jacod et al., 2004), we get Equation (4.12):

$$\begin{aligned} \mathbb{P}(s_{\text{diff}} > 0) &= 1 - \mathbb{P}(s_{\text{diff}} \leq 0) = \frac{1}{2} \left(1 - \operatorname{erf} \left(-\frac{\mathbb{E}[s_{\text{diff}}]}{\sqrt{2\operatorname{Var}(s_{\text{diff}})}} \right) \right) \\ \iff \mathbb{P}(s(u_{\text{in}}, \mu_{\mathcal{U}}) > s(u_{\text{out}}, \mu_{\mathcal{U}})) &= \frac{1}{2} \left(1 + \operatorname{erf} \left(\sqrt{\frac{d\sigma^2}{2((2(k-1) + \kappa)\sigma^2 + 2k\gamma\mu\sigma + 2k^2\mu^2)}} \right) \right). \end{aligned}$$

4.B Proof of Proposition 4.4

This proof starts by evaluating $p_{(\frac{i}{k})} = \mathbb{P}(\text{Precision}_k(\mathcal{U})) = \frac{i}{k}$ for $i \in \{1, \dots, k\}$, and then computes $\text{Consistency}_k(\mathcal{X}) = \mathbb{E}[\text{Precision}_k(\mathcal{U})] = \sum_{i=1}^k \frac{i}{k} \times p_{(\frac{i}{k})}$ under the studied hypotheses. To begin with, we set:

$$p_{(\frac{i}{k})}^+ = \mathbb{P}(\text{Precision}_k(\mathcal{U}) \geq \frac{i}{k}), \text{ for } i \in \{1, \dots, k+1\}.$$

As i takes integer values:

$$p_{(\frac{i}{k})} = p_{(\frac{i}{k})}^+ - p_{(\frac{i+1}{k})}^+.$$

Assuming $p_{(\frac{k+1}{k})}^+ = 0$ by definition, we have:

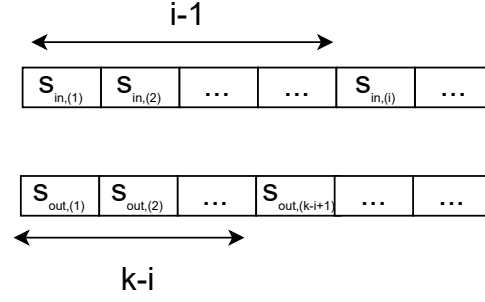
$$\mathbb{E}[\text{Precision}_k(\mathcal{U})] = \sum_{i=1}^k \frac{i}{k} \times (p_{(\frac{i}{k})}^+ - p_{(\frac{i+1}{k})}^+) = \frac{1}{k} \sum_{i=1}^k p_{(\frac{i}{k})}^+. \quad (4.16)$$

Let us denote by $s_{\text{in},(i)}$ the distribution of the i^{th} highest value of $S_{\text{in}} = \{s(u_{\text{in}}, \mu_{\mathcal{U}}), u_{\text{in}} \in \mathcal{U}\}$, and by $s_{\text{out},(k-i+1)}$ the $(k-i+1)^{\text{th}}$ highest value of $S_{\text{out}} = \{s(u_{\text{out}}, \mu_{\mathcal{U}}), u_{\text{out}} \in \bar{\mathcal{U}}\}$, for $i \in \{1, \dots, k\}$. Using this notation, we observe that:

$$p_{(\frac{i}{k})}^+ = \mathbb{P}(s_{\text{in},(i)} > s_{\text{out},(k-i+1)}).$$

Indeed, as illustrated in Figure 4.10, we can show that $s_{\text{in},(i)} > s_{\text{out},(k-i+1)} \iff \text{Precision}_k(\mathcal{U}) \geq \frac{i}{k}$:

- If $s_{\text{in},(i)} > s_{\text{out},(k-i+1)}$ then at most $k-i$ elements of S_{out} are greater than $s_{\text{in},(i)}$. Also, by definition, exactly $i-1$ elements of S_{in} are greater than $s_{\text{in},(i)}$. So, at most $k-1$ similarities are greater than $s_{\text{in},(i)}$ overall. Thus, $s_{\text{in},(i)}$ is one of the k highest values of $S_{\text{in}} \cup S_{\text{out}}$ and $\text{Precision}_k(\mathcal{U}) \geq \frac{i}{k}$.
- If $\text{Precision}_k(\mathcal{U}) \geq \frac{i}{k}$, then at least i elements of S_{in} are in the top- k of $S_{\text{in}} \cup S_{\text{out}}$, including $s_{\text{in},(i)}$. That leaves at most $k-i$ slots available in the top- k . Since, by definition, exactly $k-i$ elements of S_{out} are greater than $s_{\text{out},(k-i+1)}$, $s_{\text{out},(k-i+1)}$ is necessarily outside of the top- k elements of $S_{\text{in}} \cup S_{\text{out}}$, and so $s_{\text{in},(i)} > s_{\text{out},(k-i+1)}$.


 Figure 4.10: Ranking s_{in} and s_{out} statistics.

In our setting, we derive $s_{\text{in},(i)}$ by noticing that the distribution of the i^{th} highest value of S_{in} is also the distribution of its $(k-i)^{\text{th}}$ lowest value. Hence, $s_{\text{in},(i)}$ is the $(k-i)^{\text{th}}$ order statistic of S_{in} . As $\mu = 0$, we have that, given two distinct elements of $\mathcal{U}$, say u_1 and u_2 , $\text{Cov}(s(u_1, \mu_{\mathcal{U}}), s(u_2, \mu_{\mathcal{U}})) = 0$, which implies that all elements of S_{in} are i.i.d. and that the probability density function of the $(k-i)^{\text{th}}$ order statistic of S_{in} is (Nagaraja et al., 2003):

$$f_{\text{in},(k-i)}(x) = \frac{k!}{\sigma_{\text{in}} 2^{k-1} (k-i)! (i-1)!} \phi\left(\frac{x - \mu_{\text{in}}}{\sigma_{\text{in}}}\right) \left(1 + \text{erf}\left(-\frac{x - \mu_{\text{in}}}{\sqrt{2}\sigma_{\text{in}}}\right)\right)^{k-i} \left(1 - \text{erf}\left(-\frac{x - \mu_{\text{in}}}{\sqrt{2}\sigma_{\text{in}}}\right)\right)^{i-1},$$

with:

$$\begin{aligned} \mu_{\text{in}} &= d \frac{\sigma^2}{k}, \\ \sigma_{\text{in}} &= \sigma^2 \frac{\sqrt{d(\kappa + k - 2)}}{k}, \\ \phi : x &\mapsto \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \end{aligned}$$

Similarly, $s_{\text{out},(k-i+1)}$ is the $(N - 2k + i)^{\text{th}}$ order statistic of S_{out} and so its cumulative density function is (Nagaraja et al., 2003):

$$F_{\text{out},(k-i+1)}(x) = \sum_{j=N-2k+i}^{N-k} \binom{N-k}{j} \left(1 + \text{erf}\left(-\frac{x}{\sqrt{2}\sigma_{\text{out}}}\right)\right)^j \left(1 - \text{erf}\left(-\frac{x}{\sqrt{2}\sigma_{\text{out}}}\right)\right)^{N-k-j},$$

with:

$$\sigma_{\text{out}} = \sigma^2 \sqrt{\frac{d}{k}}.$$

Finally, we obtain:

$$\begin{aligned} p_{\left(\frac{i}{k}\right)}^+ &= \mathbb{P}(s_{\text{in},(i)} > s_{\text{out},(k-i+1)}) \\ &= \int_{-\infty}^{\infty} f_{\text{in},(i)}(x) \times F_{\text{out},(k-i+1)}(x) dx, \end{aligned} \tag{4.17}$$

and, by injecting Equation (4.17) into Equation (4.16), we retrieve the expression from Equation (4.13) of Proposition 4.4.

The past two chapters have outlined a recurring assumption in sequence modeling: to predict future values, one can attempt to learn the "rules" that were used to generate the past observed sequences, then use these rules to generate more sequences. A convenient mathematical tool to describe such rules is to consider that past observations were sampled from a probability distribution P_{data} . The assumption of a stochastic behavior to describe these rules allows to account for the unobservability of some of the factors that influence the data generating process. We refer the reader to Chapter 5 of Goodfellow et al., 2016 for more discussion on this assumption. In Chapter 4 we introduced the notion of consistency of an aggregator of embeddings. Building a consistent aggregation can be understood as finding a representation for a sequence of embeddings that maximizes the likelihood of each of these embeddings to be a part of the original sequence. Similarly, the models of Chapter 4 were trained to build representations of playlists that maximize the likelihood for songs in the playlist to be a part of it. In the case of automatic playlist continuation, the data generating function P_{data} is a probability distribution defined over the set of subsets of songs from a catalog. When models are trained to fit a dataset of user-made playlists, P_{data} can be interpreted as the decision making process of a single meta-user, responsible for all of the independent decisions made by each individual user of the streaming platform. Training on observations drawn from P_{data} , i.e fitting a dataset of users playlists, gathers in a single location the information defining "how are playlists made" which was previously dispatched between millions of users. Models can then recommend songs to each user by leveraging the aggregated knowledge of how millions of other users make their playlists. From a single user's perspective, getting access to a knowledge that was formerly distributed across the entire user base can be of significant value, explaining the success of sequence modeling approaches for APC, as observed in the live experiment on Deezer reported in Chapter 3.

In the next chapters, we will study the possibility to sample songs for playlists following a distribution that differs from the one observed in the training set, with the ultimate goal of increasing user satisfaction. To this end, Chapter 5 is dedicated to the description and analysis of an exploration method aiming at replacing the $\arg \max$ operation usually performed at the final stage of a song selection process (see Section 2.3 of Chapter 2) by a sampling operation that, although informed of which song maximizes the likelihood of following a given sequence—with that sequence being represented by an embedding vector—deliberately selects other songs. The introduction of exploration will result in a shift of the distribution of observed samples, which will benefit future approaches based on a *Reinforcement Learning* (RL), described in Chapter 6, that aim at directly optimizing the satisfaction that a model is expected to bring.

Chapter 5

Exploring Large Action Sets with Hyperspherical Embeddings using von Mises-Fisher Sampling

This chapter presents research conducted with Guillaume Salha-Galvan, Romain Hennequin, Théo Bontempelli, Thomas Bouabça, and Tristan Cazenave, and presented at the Workshop on Aligning Reinforcement Learning Experimentalists and Theorists, 41st International Conference on Machine Learning (ICML 2024) (Bendada, Salha-Galvan, Hennequin, Bontempelli, et al., 2024).

5.1 Introduction

In this chapter, we propose to address the important question of unrestricted exploration in large-scale decision making, an important challenge to overcome before adapting Reinforcement Learning (RL) approaches to sequential music recommendation. We postpone the formal introduction of some fundamental concepts of RL to the next chapter, as we currently simply make a single step towards RL by giving embedding-based recommendation models studied in the previous chapters the ability to select songs, or actions, in a stochastic manner. Given that our analysis goes beyond music recommendation, in this chapter we regularly refer to models as RL agents and to items as actions to select.

Our work focuses on the specific setting where actions are represented by embedding vectors of dimension $d \geq 2$ with unit norm, i.e., embedding vectors lying on the d -dimensional unit hypersphere. Our contributions are as follows:

- We introduce von Mises-Fisher exploration (vMF-exp), a scalable method for exploring large sets of actions represented by hyperspherical embedding vectors. vMF-exp involves

initially sampling a state embedding vector using a von Mises-Fisher distribution (Fisher, 1953), then exploring this representation’s nearest neighbors. Our proposed strategy scales to millions of candidate actions and, unlike two-stage approaches where action sampling is preceded by the selection of a small set of candidates, does not restrict exploration to a specific neighborhood.

- We provide a comprehensive analysis of vMF-exp, demonstrating that, under certain theoretical assumptions, it asymptotically maintains the same probability of exploring each action as the popular Boltzmann exploration (B-exp) method, while overcoming its scalability issues. Consequently, vMF-exp serves as a scalable alternative to B-exp for effectively exploring large action sets.
- While our analysis remains general, we also offer a real-world example of a vMF-exp usage. We describe how, in 2024, we have deployed vMF-exp at scale on Deezer to introduce randomness in the song selection process of the Track Mix feature presented in Chapter 3. This application, backed by successful A/B tests on millions of users, confirms the empirical relevance of vMF-exp.
- We release a Python implementation of vMF-exp on GitHub to encourage its future use.

5.2 Preliminaries

5.2.1 Problem Formulation

Notation In this chapter, we consider an RL agent sequentially selecting actions within a set $\mathcal{I}_n = \{1, 2, \dots, n\}$ of $n \in \mathbb{N}^*$ actions. Each action $i \in \mathcal{I}_n$ is represented by a distinct low-dimensional vectorial representation $X_i \in \mathbb{R}^d$, i.e., by an embedding vector or simply an embedding¹, for some fixed dimension $d \in \mathbb{N}$ with $d \geq 2$ and $d \ll n$. Additionally, we assume all vectors have a unit Euclidean norm, i.e., $\|X_i\|_2 = 1, \forall i \in \mathcal{I}_n$. They form a set of embeddings noted $\mathcal{X}_n = \{X_i, 1 \leq i \leq n\} \in (\mathcal{S}^{d-1})^n$, where $\mathcal{S}^{d-1}$ is the d -dimensional unit hypersphere (Fisher, 1953): $\mathcal{S}^{d-1} = \{x \in \mathbb{R}^d : \|x\|_2 = 1\}$.

We also assume the availability of an approximate nearest neighbor (ANN) (Johnson et al., 2019; Li et al., 2019) search engine. Using this engine, for any vector $V \in \mathcal{S}^{d-1}$, the nearest neighbor of V among $\mathcal{X}_n$ in terms of inner product similarity (equal to the cosine similarity, for unit vectors (Tan et al., 2016)), called $X_{i_V^*}$, can be retrieved in a sublinear time complexity with respect to n . Although ANN engines are parameterized based on a trade-off between efficiency and accuracy, we make the simplifying assumption that $X_{i_V^*}$ is the actual nearest neighbor of V , which we later discuss in Section 5.4.3. Formally, $i_V^* = \arg \max_{i \in \mathcal{I}_n} \langle V, X_i \rangle$.

On a music streaming service, $\mathcal{X}_n$ would represent embeddings associated with each song of the catalog $\mathcal{I}_n$. In this case, n would be on the order of several millions (Bendada, Salha, et al., 2020; Jacobson et al., 2016; Schedl et al., 2018). The RL agent would be the recommender system

¹At this stage, we do not make assumptions regarding the specific methods or data used to learn these embedding vectors, nor the precise interpretation of proximity between vectors in the embedding space.

sequentially recommending these songs to users. Normalizing embeddings, as is done in the recommendation system of Deezer (see Section 2.3.4 of Chapter 2) is a common practice in both academic and industrial recommender systems (Afchar et al., 2023; Bontempelli et al., 2022; D. Kim et al., 2023; Schedl et al., 2018) to mitigate popularity biases, as vector norms often encode popularity information on items (Afchar et al., 2023; J. Chen et al., 2023). Normalizing embeddings also prevents inner products from being unbounded, avoiding overflow and underflow numerical instabilities (LeCun et al., 2015). Recall that the assumption of unit-norm embeddings wasn't a prerequisite of the theoretical analysis of Section 4.3 of Chapter 4. This is because in this previous chapter we were interested in characterizing which properties of a distribution favor a better consistency of the average of embeddings sampled from that distribution, within a very broad family of distributions of vectors of a given dimensions, not necessarily similar to current real-world embeddings. Interestingly enough, we established that the most appropriate distribution when it comes to maximizing consistency was the Rademacher distribution, which samples fixed-norm embeddings also laying in a hypersphere.

At time t , the agent considers a state vector $V_t \in \mathcal{S}^{d-1}$, noted V for brevity. It selects the next action in $\mathcal{I}_n$, whose relevance is evaluated by a reward provided by the environment. In our example, the agent would recommend the next song to continue the playlist, based on the current state of the playlist that a function, for instance a neural network, has represented as an embedding vector V . In this case, the reward might be based on user feedback, such as liking or skipping the song (Bontempelli et al., 2022). The agent may select i_V^* , i.e., exploit i_V^* (Sutton et al., 2018). Alternatively, it may rely on an exploration strategy to select another $\mathcal{I}_n$ element. Formally, an exploration strategy P is a policy function (Sutton et al., 2018) that, given V , selects each action $i \in \mathcal{I}_n$ with a probability $P(i | V) \in [0, 1]$.

Objective Our goal in this chapter is to develop a suitable exploration strategy for our specific setting, where hyperspherical embedding vectors represent actions, and the number of actions can reach millions. Precisely, we aim to obtain an exploration scheme meeting the following properties:

- **Scalability (P1):** we consider an exploration scalable if the time required to sample actions given a vector V is at most the time needed for the ANN engine to retrieve the nearest neighbor, which is typically achieved in a sublinear time complexity with respect to n . Scalability is a mandatory requirement for exploring large action sets with millions of elements.
- **Unrestricted radius (P2):** $\text{Radius}(P | V)$ is the number of actions with a non-zero probability of being explored given a state V . While exploring actions too far from V might be suboptimal (e.g., resulting in poor recommendations), it is crucial that exploration is not restricted to a specific radius by construction. Such a restriction could prevent the agent from exploring relevant actions that lie beyond this radius. An unrestricted radius ensures that the exploration strategy remains flexible and capable of adapting to various contexts, allowing for the exploration of relevant actions regardless of their embedding position.

- **Order preservation (P3)**: order is preserved when the probability of selecting the action i given the state V is a strictly increasing function of $\langle V, X_i \rangle$. More formally, order preservation requires $\forall (i, j) \in \mathcal{I}_n, \langle V, X_i \rangle > \langle V, X_j \rangle \implies P(i | V) > P(j | V)$. Order preservation implies that the exploration strategy properly leverages the information captured in the embedding vectors to assess the relevance of an action given a state.

5.2.2 Limitations of Existing Exploration Strategies

Finding an exploration strategy that simultaneously meets these three properties is essential for effective exploration in RL problems with large action sets and embedding representations. Nonetheless, existing exploration strategies suffer from limitations that motivate our work in this chapter.

Random and ε -greedy Exploration The most straightforward example of an exploration strategy would be the random (uniform) policy, where $P_{\text{rand}}(i | V) = \frac{1}{n}, \forall i \in \mathcal{I}_n$. A popular variant is the ε -greedy strategy (Sutton et al., 2018). With a probability $\varepsilon \in [0, 1]$, the agent would choose the next action uniformly at random. With a probability $1 - \varepsilon$, it would exploit the most relevant action based on its knowledge. Random and ε -greedy exploration strategies are scalable (**P1**), as elements of $\mathcal{I}_n$ can be uniformly sampled in $\mathcal{O}(1)$ time (Cormen et al., 2022). Additionally, they verify **P2**. Indeed, $\text{Radius}(P_{\text{rand}}|V) = n$ since every action can be selected. However, these strategies ignore embeddings at the sampling phase and do not achieve order preservation (**P3**). This is a significant limitation, reinforced by the fact that these policies have a maximal radius. As explained in Section 5.1, in large action sets, many actions are often irrelevant, e.g., most songs from the musical catalog would constitute poor recommendations given an initial state (Tomasi et al., 2023). Exploring each action/song with equal probability, including inappropriate ones, could result in negative user feedback.

In Figure 5.1, we share some (anonymized) real examples of negative feedback that users wrote to Deezer following unsatisfying recommendations. Some of those feedback were the direct consequence of introducing stochasticity using an ε -greedy approach in a previously deterministic policy. Those negative feedback were sent to the customer support team of the company by independent unrelated users within the couple of days following the algorithmic changes. As can be seen, the negative consequences of those bad recommendations can hardly be captured by a finite scalar value. Indeed, how should a model value the threat of unsubscribing from the service to go to competitors, or the deliberate action of spreading bad publicity about the application on social media? Figure 5.2 displays an excerpt of negative reactions from Deezer users on X.com following unsatisfactory recommendations. The consequences of those reactions can unfortunately not be balanced by future good recommendations.

Boltzmann Exploration To address the limitations of random exploration, one can sample actions according to their embedding similarity with V . The prevalent approach in RL is *Boltzmann Exploration* (B-exp) (Amin et al., 2021; Cesa-Bianchi et al., 2017; M. Chen, B. Chang, et al., 2021; Sutton et al., 2018), which employs the Boltzmann distribution for action

sampling:

$$\forall i \in \mathcal{I}_n, P_{\text{B-exp}}(i \mid V, \mathcal{X}_n, \kappa) = \frac{e^{\kappa \langle V, X_i \rangle}}{\sum_{j=1}^n e^{\kappa \langle V, X_j \rangle}}, \quad (5.1)$$

where the hyperparameter $\kappa \in \mathbb{R}^+$ controls the entropy of the distribution. B-exp samples actions according to a strictly increasing function of their inner product similarity with V for $\kappa > 0$, guaranteeing order preservation (**P3**). By carefully tuning κ , one can ensure that irrelevant actions are practically never selected while maintaining a non-zero probability of recommending actions with less than maximal similarity, thereby indirectly controlling the radius of the policy (**P2**). Unfortunately, B-exp does not satisfy **P1**, i.e., it is not scalable to large action sets. Indeed, evaluating Equation (5.1) requires explicitly computing the probability of sampling each individual action before actually sampling from them, which is prohibitively expensive for large values of n (M. Chen, B. Chang, et al., 2021). Note that, while we focus on B-exp in this section, these scalability concerns would remain valid for any other sampling distribution requiring explicitly computing similarities and probabilities for each of the n actions (Amin et al., 2021).

Truncated Boltzmann Exploration Due to these scalability concerns, previous work on RL with large and embedded action sets often settled for a workaround consisting in sampling actions from a truncated version of the Boltzmann distribution (or another distribution) (M. Chen, B. Chang, et al., 2021). In this method, which we refer to as Truncated Boltzmann Exploration (TB-exp), a small number $m \ll n$ of candidate actions, usually around hundreds or thousands, is first retrieved using the ANN search engine, leading to a candidate action set $\mathcal{I}_m(V)$. The sampling step is subsequently performed only within $\mathcal{I}_m(V)$:

$$\forall i \in \mathcal{I}_m(V), P_{\text{TB-exp},m}(i \mid V, \mathcal{X}_n, \kappa) = \frac{e^{\kappa \langle V, X_i \rangle}}{\sum_{j \in \mathcal{I}_m(V)} e^{\kappa \langle V, X_j \rangle}}. \quad (5.2)$$

TB-exp performs action selection in a time that depends on m instead of n , and has been successfully deployed in production environments involving millions of actions (M. Chen, Beutel, et al., 2019; M. Chen, B. Chang, et al., 2021; M. Chen, C. Xu, et al., 2022). While it still satisfies **P3**, TB-exp also meets **P1** for small values of m . However, it no longer satisfies **P2**. This method restricts the radius, i.e., the number of candidate actions, based on technical considerations rather than exploration efficiency. This restriction can potentially hinder model convergence by neglecting the exploration of relevant actions beyond this fixed radius. In summary, the challenge of finding an exploration strategy that satisfies **P1**, **P2**, and **P3** simultaneously—in other words, an exploration scheme with properties similar to full Boltzmann exploration yet scalable—remains relatively open.

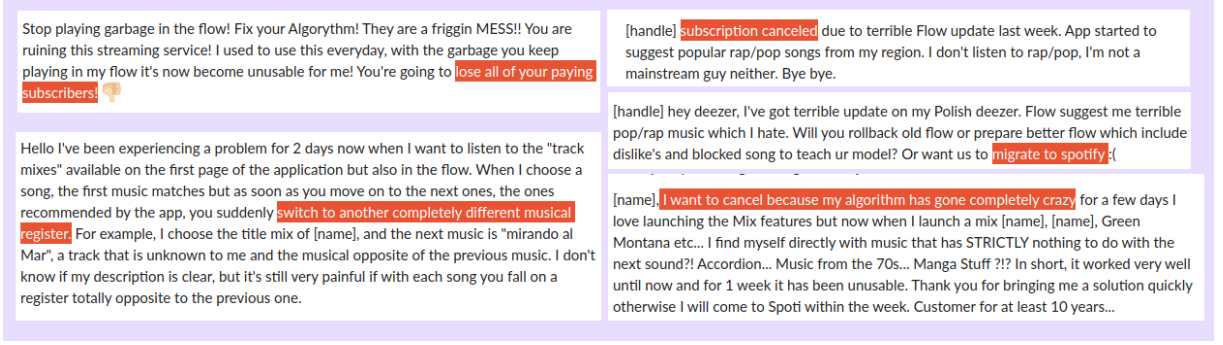


Figure 5.1: Messages sent to Deezer from users complaining about their recommendations.

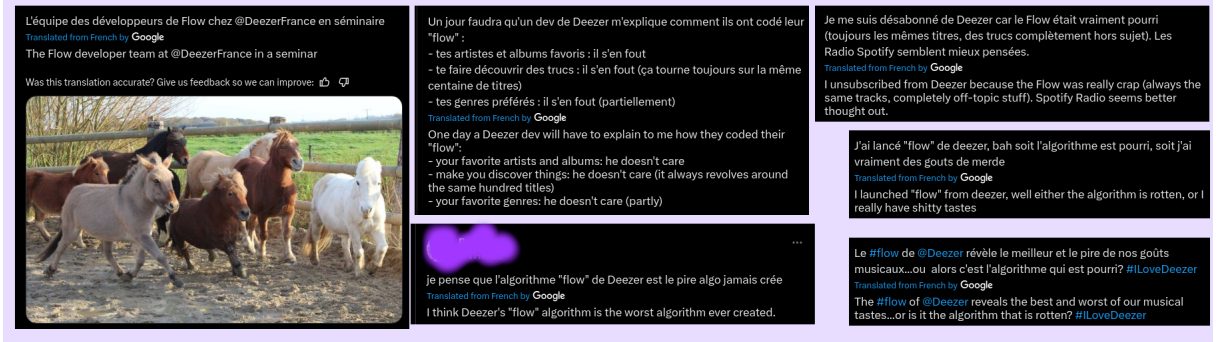


Figure 5.2: Public tweets from users complaining about their Deezer recommendations on X.

5.3 From Boltzmann to von Mises–Fisher (vMF) Exploration

In this section, we present our solution for exploring large action sets with hyperspherical embeddings.

5.3.1 von Mises–Fisher Exploration

The inability of B-exp to scale arises from its need to compute all n sampling probabilities explicitly. In this chapter, we propose *von-Mises Fisher* (vMF), an alternative exploration strategy that overcomes this constraint. Specifically, given an initial state vector V , vMF-exp consists in:

- Firstly, sampling a vector $\tilde{V}$ according to a vMF distribution (Fisher, 1953) centered on V .
- Secondly, selecting $\tilde{V}$'s nearest neighbor action in the embedding space for exploration.

In directional statistics, the vMF distribution (Fisher, 1953) is a continuous vector probability distribution defined on the unit hypersphere $\mathcal{S}^{d-1}$. It has recently been used in RL to assess the uncertainty of gradient directions (Zhu et al., 2024). For all $\tilde{V} \in \mathcal{S}^{d-1}$, its probability density function (PDF) is:

$$f_{\text{vMF}}(\tilde{V} \mid \kappa, V, d) = C_d(\kappa) e^{\kappa \langle V, \tilde{V} \rangle}, \text{ with } C_d(\kappa) = \frac{1}{\int_{\tilde{V} \in \mathcal{S}^{d-1}} e^{\kappa \langle V, \tilde{V} \rangle} d\tilde{V}} = \frac{\kappa^{\frac{d}{2}-1}}{(2\pi)^{\frac{d}{2}} I_{\frac{d}{2}-1}(\kappa)} \quad (5.3)$$

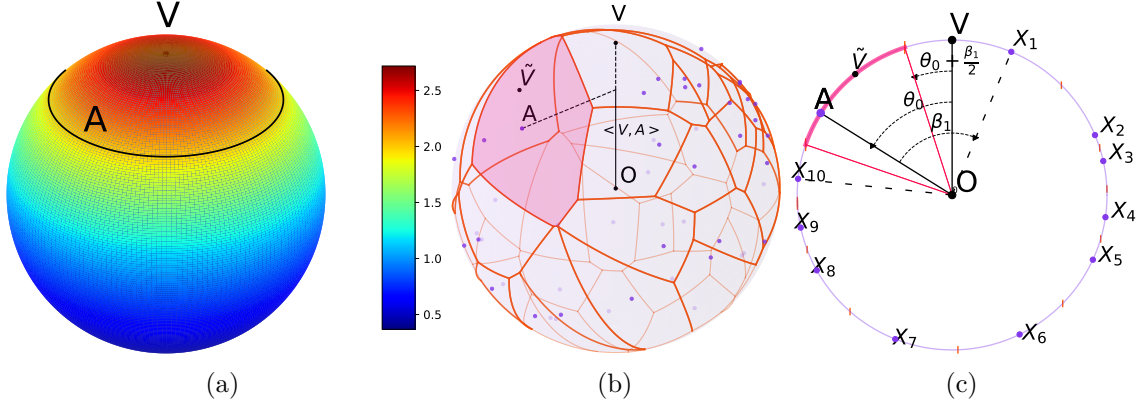


Figure 5.3: (a) PDF of a 3D vMF distribution. (b) vMF-exp explores the action A when the sampled vector $\tilde{V}$ lies in A 's Voronoï cell, shown in red in 3D. (c) Same as (b) in 2D.

with $\kappa \in \mathbb{R}^+$. The function $I_{\frac{d}{2}-1}$ designates the modified Bessel function of the first kind (Baricz, 2010) at order $d/2-1$. Figure 5.3a illustrates the PDF of a vMF distribution on the 3-dimensional unit sphere. For any $\tilde{V} \in \mathcal{S}^{d-1}$, $f_{\text{vMF}}(\tilde{V} \mid \kappa, V, d)$ is proportional to $e^{\kappa \langle V, \tilde{V} \rangle}$, which is reminiscent of the Boltzmann sampling probability of Equation (5.1). The hyperparameter κ controls the entropy of the distribution. In particular, for $\kappa = 0$, the vMF distribution boils down to the uniform distribution on $\mathcal{S}^{d-1}$.

5.3.2 Properties

P1 vMF-exp only requires sampling a d -dimensional vector instead of handling a discrete distribution with n parameters, allowing $\tilde{V}$ to be sampled in constant time with respect to n . Therefore, vMF-exp is a scalable exploration strategy. Efficient sampling algorithms for vMF distributions have been extensively studied (S. Kang et al., 2024; Pinzón et al., 2023). As shown in the following sections, we successfully explored sets of millions of actions without scalability issues, using the Python vMF sampler from (Pinzón et al., 2023) for simulations in Section 5.4 and our own variant implementation for A/B tests in Section 5.5 which steps are detailed in Appendix 5.G.

P2 The probability of sampling $i \in \mathcal{I}_n$ given V for exploration is the probability that X_i is the nearest neighbor of $\tilde{V}$ among $\mathcal{X}_n$ vectors, i.e., that $\tilde{V}$ lies in $\mathcal{S}_{\text{Voronoi}}(X_i \mid \mathcal{X}_n)$, the Voronoï cell of X_i in the Voronoï tessellation of $\mathcal{S}^{d-1}$ defined by $\mathcal{X}_n$ (Du, Faber, et al., 1999; Du, Gunzburger, et al., 2010) (see Figures 5.3b and 5.3c). We have: $\mathcal{S}_{\text{Voronoi}}(X_i \mid \mathcal{X}_n) = \{\tilde{V} \in \mathcal{S}^{d-1}, \forall j \in \mathcal{I}_n, \langle \tilde{V}, X_i \rangle \geq \langle \tilde{V}, X_j \rangle\}$, and $\bigcup_{i \in \mathcal{I}_n} \mathcal{S}_{\text{Voronoi}}(X_i \mid \mathcal{X}_n) = \mathcal{S}^{d-1}$. Using this notation, the probability of sampling the action i for exploration using vMF-exp is:

$$\forall i \in \mathcal{I}_n, P_{\text{vMF-exp}}(i \mid V, \mathcal{X}_n, \kappa) = \int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(X_i \mid \mathcal{X}_n)} f_{\text{vMF}}(\tilde{V} \mid \kappa, V, d) d\tilde{V}, \quad (5.4)$$

which is always strictly positive. Thus, vMF-exp verifies the unrestricted radius property. Like B-exp, tuning κ ensures that actions with low similarity have negligible sampling probabilities in practice.

P3 $P_{\text{vMF-exp}}(i \mid V, \mathcal{X}_n, \kappa)$ increases due to two factors. Firstly, the average $f_{\text{vMF}}(\tilde{V} \mid \kappa, V, d)$ value for $\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(X_i \mid \mathcal{X}_n)$, which is correlated to $\langle X_i, V \rangle$ and contributes to order preservation. Secondly, the surface area of $\mathcal{S}_{\text{Voronoi}}(X_i \mid \mathcal{X}_n)$, measuring how dissimilar X_i is from other $\mathcal{X}_n$ elements. Actions embedded in a low-density subspace of $\mathcal{S}^{d-1}$ will have an expanded Voronoi cell and may be selected more often than actions closer to V but located in a high-density subspace. Hence, vMF-exp favors actions that are both similar to V and dissimilar to other actions, and order preservation depends on the $\mathcal{X}_n$ distribution. Section 5.4 will focus on a setting where B-exp and vMF-exp asymptotically share similar probabilities. Consequently, vMF-exp, like B-exp, will verify order preservation (**P3**). In conclusion, in this setting, vMF-exp will verify **P1**, **P2**, and **P3** simultaneously.

5.4 Theoretical Comparison of vMF-exp and B-exp

We now provide a mathematical comparison of vMF-exp and B-exp. We focus on the theoretical setting presented in Section 5.4.1. We show that, in this setting, vMF-exp maintains the same probability of exploring each action as B-exp, while overcoming its scalability issues. As noted above, this implies that vMF-exp verifies **P1**, **P2**, and **P3** simultaneously and, therefore, acts as a scalable alternative to the popular but unscalable B-exp for exploring large action sets with hyperspherical embeddings.

5.4.1 Setting and Assumptions

We focus on the setting where embeddings are independent and identically distributed (i.i.d.) and follow a uniform distribution on the unit hypersphere, i.e., $\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})$. For convenience in our proofs, we consider the action set to be the union of $\mathcal{I}_n$, the set of n actions, and another action a with a known embedding $A \in \mathcal{S}^{d-1}$. The resulting entire action set $\mathcal{I}_{n+1}$ and embedding set $\mathcal{X}_{n+1}$ are defined as $\mathcal{I}_{n+1} = \mathcal{I}_n \cup \{a\}$ and $\mathcal{X}_{n+1} = \mathcal{X}_n \cup \{A\}$. In this section, we are interested in the probability of each exploration scheme, B-exp and vMF-exp, to sample a among all actions of $\mathcal{I}_{n+1}$ given a state embedding vector $V \in \mathcal{S}^{d-1}$. These probabilities are defined respectively as:

$$P_{\text{B-exp}}(a \mid n, d, V, \kappa) = \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [P_{\text{B-exp}}(a \mid V, \mathcal{X}_{n+1}, \kappa)], \quad (5.5)$$

$$P_{\text{vMF-exp}}(a \mid n, d, V, \kappa) = \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [P_{\text{vMF-exp}}(a \mid V, \mathcal{X}_{n+1}, \kappa)]. \quad (5.6)$$

5.4.2 Results

We now present and discuss our main theoretical results. For brevity, we report all intermediary lemmas and mathematical proofs in the Appendices 5.A to 5.D of this chapter. Our first and most general result links the asymptotic behavior of B-exp and vMF-exp as the action set grows.

Proposition 5.1. In the setting of Section 5.4.1, we have:

$$\lim_{n \rightarrow +\infty} \frac{P_{\text{B-exp}}(a \mid n, d, V, \kappa)}{P_{\text{vMF-exp}}(a \mid n, d, V, \kappa)} = 1. \quad (5.7)$$

Proposition 5.1 states that, for large values of n , the probability of selecting a for exploration is asymptotically the same using either B-exp or vMF-exp. This result follows from the respective asymptotic characterizations of $P_{\text{B-exp}}$ and $P_{\text{vMF-exp}}$, detailed below. Importantly, it implies that, for large values of n , vMF-exp shares the same properties as B-exp (**P2**, **P3**), including order preservation. However, as noted in Section 5.3, vMF-exp offers greater scalability since its implementation only requires sampling a vector of a fixed size d , an operation independent of the number of actions n (**P1**). Next, we give a common approximate expression for both methods, defined as $P_0(a | n, d, V, \kappa) = \frac{f_{\text{vMF}}(A|V, \kappa) \mathcal{A}(\mathcal{S}^{d-1})}{n}$, with $\mathcal{A}(\mathcal{S}^{d-1})$ denoting the surface area of the hypersphere $\mathcal{S}^{d-1}$, and describe the rate at which this asymptotic behavior is reached as n grows.

Proposition 5.2. In the setting of Section 5.4.1, we have:

$$P_{\text{B-exp}}(a | n, d, V, \kappa) = P_0(a | n, d, V, \kappa) + o\left(\frac{1}{n\sqrt{n}}\right). \quad (5.8)$$

Proposition 5.3. In the setting of Section 5.4.1, we have:

$$P_{\text{vMF-exp}}(a | n, d, V, \kappa) = P_0(a | n, d, V, \kappa) + \begin{cases} \mathcal{O}\left(\frac{1}{n^2}\right) & \text{if } d = 2, \\ \mathcal{O}\left(\frac{1}{n^{1+\frac{2}{d-1}}}\right) & \text{if } d > 2. \end{cases} \quad (5.9)$$

In essence, when n is large, the probability of sampling the action a can be approximated by the PDF of the vMF distribution evaluated at A multiplied by the average surface area of the Voronoï cell of A , for both exploration methods. As n grows, this Voronoï cell shrinks until f_{vMF} becomes nearly constant across its entire surface. Figure 5.4f illustrates this interpretation.

However, the rate at which both exploration methods reach their asymptotic behavior differs. The rate at which the Voronoi cell shrinks depends on the dimension of the hypersphere, which explains why the second term of Equation (5.9) depends on d . Note that this is not the case for B-exp. Consequently, for large values of d , one may require a higher number of actions n before the asymptotic behavior of Equation (5.7) is observed. For this reason, it is useful to obtain a more precise approximation of $P_{\text{vMF-exp}}(a | n, d, V, \kappa)$ when d increases, which we provide in the next section.

5.4.3 Discussion

High Dimension Following the above discussion, Proposition 5.4 offers a more precise expression of $P_{\text{vMF-exp}}(a | n, V, \kappa)$ to use when d increases (roughly, $d \geq 20$ in our experiments). It is obtained by studying the first two terms of the Taylor expansion (Abramowitz et al., 1948) of f_{vMF} near A , instead of only the zero-order term, the second term becoming more significant when d increases. Despite its apparent complexity, it can be interpreted simply. The negative sign before $\langle V, A \rangle$, indicates that, when A is similar to V , it is sampled less often than with B-exp for the same κ and d values. Conversely, when A is on the opposite side of the hypersphere, the term contributes positively to $P_{\text{vMF-exp}}(a | n, V, \kappa)$. To summarize, for larger d values, vMF-exp is expected to explore more than B-exp with the same κ .

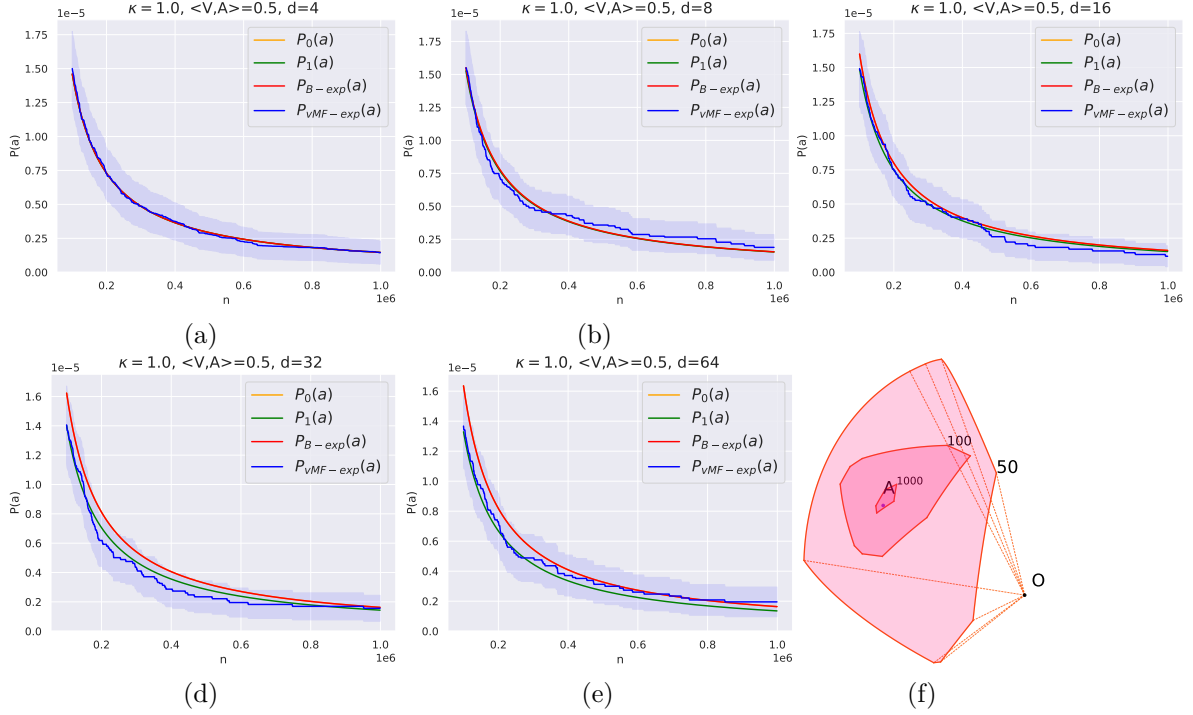


Figure 5.4: (a) to (e): Simulations of Section 5.4.3. (f) 3D Voronoi cell of A for $n \in \{50, 100, 1000\}$.

Proposition 5.4. Let $B : (z_1, z_2) \mapsto \int_0^1 t^{z_1-1} (1-t)^{z_2-1} dt$ denote the Beta function, and $\Gamma : z \mapsto \int_0^\infty t^{z-1} e^{-t} dt$ denote the Gamma function (Abramowitz et al., 1948). In the setting of Section 5.4.1 with $d \geq 3$, we have:

$$P_{\text{vMF-exp}}(a \mid n, V, \kappa) = P_1(a \mid n, V, \kappa) + \mathcal{O}\left(\frac{1}{n^{\frac{2}{d-1}}}\right), \text{ with:} \quad (5.10)$$

$$P_1(a \mid n, V, \kappa) = P_0(a \mid n, V, \kappa) - \frac{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1}) \kappa \langle V, A \rangle \Gamma\left(\frac{d+1}{2}\right)}{n} \left(\frac{(d-1)B\left(\frac{1}{2}, \frac{d-1}{2}\right)}{n} \right)^{\frac{2}{d-1}}.$$

The case $d = 2$ In 2 dimensions, Voronoi cells are arcs of a circle and are delimited by the perpendicular bisectors of two neighboring points, as shown in Figure 5.3c. Interestingly, in this specific case, $P_{\text{vMF-exp}}(a \mid n, d = 2, V, \kappa)$ can be computed using geometric arguments. We report a comprehensive analysis in Appendix 5.B, confirming that, when $d = 2$, vMF-exp approaches its asymptotic behavior faster than B-exp, as indicated by the $\mathcal{O}(\frac{1}{n^2})$ term in Proposition 5.3.

Validation Using Monte Carlo Simulations Using the Python sampler of (Pinzón et al., 2023), we repeatedly sampled vectors $\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})$ and $\tilde{V} \sim \text{vMF}(V, \kappa)$, for various d , κ , and $\langle V, A \rangle$. Figure 5.4 reports, for varying values of N , for $\kappa = 1.0$, $\langle V, A \rangle = 0.5$ and growing values of d , the $P_{\text{vMF-exp}}(a)$ sampling probability depending on the number of actions n , as well as $P_{B-\text{exp}}(a)$ with similar parameters and our approximations $P_0(a)$ and $P_1(a)$. We repeated all experiments 8 million times and reported 95% intervals. Our results are consistent with our theoretical findings. Firstly, in line with Proposition 5.2, $P_{B-\text{exp}}(a)$ and $P_0(a)$ are indistinguishable

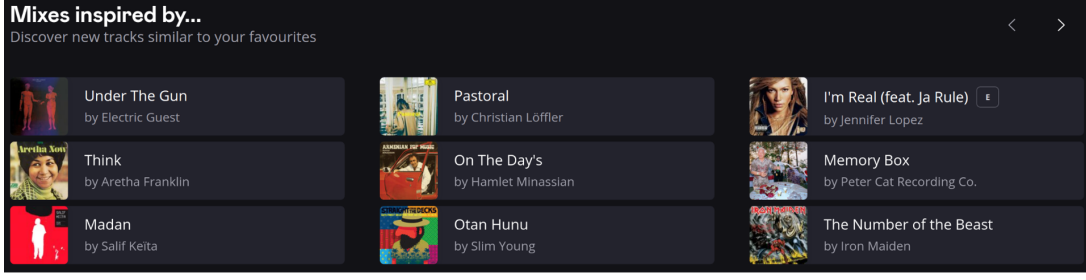


Figure 5.5: Interface of the “Mixes inspired by” recommender system on Deezer

for this range of n values. Secondly, for small d values (Figures 5.4a, 5.4b, 5.4c), $P_{\text{vMF-exp}}$ is also tightly aligned with $P_{\text{B-exp}}(a)$ and $P_0(a)$, consistently with Proposition 5.1 and 5.3. Note that the y-axis is on a $1\text{e-}5$ scale; hence, probabilities are extremely close. Thirdly, when $d \geq 16$ (Figures 5.4d, 5.4e), $P_1(a)$ becomes more distinguishable from $P_0(a)$ and constitutes a better approximation of $P_{\text{vMF-exp}}(a)$ than $P_0(a)$, as per Proposition 5.4. Lastly, since $\langle V, A \rangle > 0$, Proposition 5.4 predicts that $P_{\text{B-exp}}(a) \geq P_{\text{vMF-exp}}(a)$ for large d , which our experiments confirm. We provide comparable simulations with other $(d, \kappa, \langle V, A \rangle)$ combinations in Appendix 5.F. Our code is available online in the public GitHub repository mentioned in Chapter 1.

Link with Thompson Sampling One might draw interesting similarities between vMF-exp and bandit arm exploration using Linear Thompson Sampling (Chapelle et al., 2011). Appendix 5.E compares the two approaches.

Limitations and Future Work While we believe our study offers valuable insights into vMF-exp, several limitations must be acknowledged. Most notably, our theoretical guarantees are currently restricted to the setting of Section 5.4.1 where embeddings are i.i.d. and uniform vectors. Although, in practice, vMF-exp can be used with hyperspherical embeddings from other distributions, we do not yet provide guarantees in these cases. For instance, studying vMF-exp in clustered embedding settings, as is sometimes the case with music recommendation embeddings (Afchar et al., 2023) (where clusters can, e.g., summarize music genres (Salha-Galvan, Lutzeyer, et al., 2022)), could be insightful. Section 5.5 will demonstrate the practical value of vMF-exp on song embeddings that do not explicitly comply with Section 5.4.1, but further mathematical investigation would be warranted. In future work, we will also study the second-order term of Proposition 5.4, which could be relevant for large values of κ , and the impact of errors from the ANN engine. While we assumed this engine returns exact neighbors, this may not hold for very large action sets (Johnson et al., 2019) and, intuitively, could cause minor exploration perturbations.

5.5 Application to Large-Scale Music Recommendation

Our analysis of vMF-exp in Section 5.4 was intentionally general, as the method can be applied to various problem settings. In this Section 5.5, we showcase a real-world application of vMF-exp.

5.5.1 Experimental Setting

We consider again the same Track Mix feature presented in Section 3.6 of Chapter 3. As shown in Figure 5.5, it displays a personalized shortlist of songs, selected from those previously liked by each user. A click on a song generates a playlist of 40 songs “inspired by” the initial one, with the aim of helping users discover new music within a catalog including several millions of recommendable songs.

Remember that to generate playlists from a seed track, Deezer historically leverages a collaborative filtering model (Koren and Bell, 2015). This model learns unit norm song embedding representations of dimension $d = 128$ by factorizing a mutual information matrix based on song co-occurrences in various listening contexts, using singular value decomposition (SVD) (S. Banerjee et al., 2014). Inner product proximity in the resulting embedding space aims to reflect user preferences. When a user selects an initial song, the model retrieves its embedding, then (approximately) identifies its neighbors in the embedding space using the efficient Faiss library (Johnson et al., 2019) for ANN. Historically, Deezer would generate the entire playlist at once in production. The service is considering RL approaches to, instead, recommend songs one by one while adapting to user feedback on previous songs of the playlist (likes, skips, etc.). However, as explained in Chapter 2, adopting such approaches would require exploring millions of possible actions/songs, significantly increasing the complexity of this task. In this section, we continue generating Track Mix playlists all at once, but take a step towards RL by comparing three methods for exploring large action sets of millions of songs:

- vMF-exp: we use the embedding of the user’s selected song as the initial state V . We sample a random state embedding $\tilde{V}$ according to the vMF distribution, using the estimator of (A. Banerjee et al., 2005) to tune κ (see Equation (4) of (Sra, 2012)). Finally, we recommend the 40 nearest neighbors of $\tilde{V}$ in the embedding space according to the ANN engine.
- TB-exp: comparing vMF-exp to full B-exp is practically intractable at this scale. We compare vMF-exp to TB-exp with a similar κ . We first retrieve the $m = 500$ nearest neighbors of the initial song in the embedding space, according to the ANN engine. Then, we generate the playlist by sampling 40 songs from these 500 using a truncated Boltzmann distribution.
- Reference: we also compare vMF-exp to a baseline that retrieves the 500 nearest neighbors of the initial song using ANN, then shuffles them randomly to generate a playlist of 40 songs.

In early 2024, we conducted an online A/B test on Deezer to compare these exploration strategies in real conditions. The test involved millions of users worldwide, randomly split and unaware of the test.

5.5.2 Results

Firstly, it is important to highlight that we were able to successfully deploy vMF-exp in Deezer’s production environment, achieving a sampling latency of just a few milliseconds, comparable to the other methods. This industrial deployment on a service used by millions of users on a

daily basis confirms the claimed scalability of vMF-exp and its practical relevance for large-scale applications.

Using vMF-exp or TB-exp for exploration improved the daily number of recommended songs “liked” by users through “Mixes inspired by” (liking a song adds it to their list of favorites), compared to the reference baseline. For confidentiality, we do not report exact numbers of likes or users in each cohort, but present relative rates with respect to the reference. On average, users exposed to vMF-exp or TB-exp added 11% more recommended songs to their playlists than the reference cohort. These differences were statistically significant at the 1% level ($p\text{-value} < 0.01$). No apparent differences were observed between vMF-exp and TB-exp, showing that vMF-exp is competitive with TB-exp.

In addition, vMF-exp, which does not suffer from the restricted radius of TB-exp, recommended more diverse playlists. We measured the average Jaccard similarity (Tan et al., 2016) of playlists generated from the same initial selection, to assess how similar the songs sampled from the same state embedding were, for each method. Results reveal that TB-exp had an average Jaccard similarity 35% higher (less diverse playlists) than vMF-exp, a statistically significant difference at the 1% level ($p\text{-value} < 0.01$). Therefore, vMF-exp allowed for a more substantial exploration, without compromising performance.

At the time of writing, Deezer continues to use vMF-exp for Track Mix recommendations. Playlists are still generated at once, but our work equips this service with an effective strategy to explore their large and embedded action set of millions of songs. This opens interesting avenues for further investigation of RL for recommendation. In the near future, Deezer will launch tests involving actor-critic RL models (Konda et al., 1999; Sutton et al., 2018) to explore and generate songs sequentially based on user feedback.

5.6 Conclusion

In conclusion, the primary contribution of this chapter is the development of vMF-exp, a scalable method for exploring large action sets in RL problems where hyperspherical embedding vectors represent actions. We have shown that, under theoretical conditions, vMF-exp asymptotically maintains the same probability of exploring each action as the popular B-exp method while overcoming its scalability issues. Additionally, unlike the TB-exp workaround, which restricts exploration to a specific neighborhood, vMF-exp allows for unrestricted exploration. This makes vMF-exp a valuable tool for RL researchers and practitioners aiming to explore large action sets with hyperspherical embeddings, offering a suitable alternative to both B-exp and TB-exp. We have also discussed the limitations of our work, suggesting directions for future research. While our analysis has been general, the final part of this chapter has also provided a real-world application of vMF-exp. Specifically, we have successfully deployed vMF-exp on Deezer, where it has been used for months to better explore songs to recommend to millions of users. This application highlights the practical relevance of our work and will facilitate future RL research and large-scale experiments on Deezer.

Appendix

This appendix provides detailed proofs and discussions for all theoretical results presented in the chapter.

5.A Asymptotic Behavior of Boltzmann Exploration (Proof of Proposition 5.2)

We begin with the proof of Proposition 5.2 claiming that, in the setting of Section 5.4.1, we have:

$$P_{\text{B-exp}}(a \mid n, d, V, \kappa) = \underbrace{\frac{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1})}{n}}_{\text{denoted } P_0(a \mid n, d, V, \kappa)} + o\left(\frac{1}{n\sqrt{n}}\right), \quad (5.11)$$

with f_{vMF} the probability density function (PDF) of the von Mises-Fisher (vMF) (Fisher, 1953) distribution:

$$\forall A \in \mathcal{S}^{d-1}, f_{\text{vMF}}(A \mid V, \kappa) = C_d(\kappa) e^{\kappa \langle V, A \rangle}, \quad (5.12)$$

where $\mathcal{A}(\mathcal{S}^{d-1})$ is the surface area of $\mathcal{S}^{d-1}$, the d -dimensional unit hypersphere, and $C_d(\kappa)$ is the normalizing constant.

Proof. By definition,

$$\begin{aligned} P_{\text{B-exp}}(a \mid n, d, V, \kappa) &= \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\frac{e^{\kappa \langle V, A \rangle}}{e^{\kappa \langle V, A \rangle} + \sum_{i=1}^n e^{\kappa \langle V, X_i \rangle}} \right] \\ &= \frac{e^{\kappa \langle V, A \rangle}}{n} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\frac{1}{\frac{e^{\kappa \langle V, A \rangle}}{n} + \sum_{i=1}^n \frac{e^{\kappa \langle V, X_i \rangle}}{n}} \right] \\ &= \frac{e^{\kappa \langle V, A \rangle}}{n} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\frac{1}{D_n} \right]. \end{aligned} \quad (5.13)$$

We use D_n to denote the denominator of the expression inside the above expectation. D_n is the empirical average of n independent and identically distributed (i.i.d.) random variables (plus a constant). Therefore, by applying the *Central Limit Theorem (CLT)* (Fischer, 2011), we know that as n grows it will be asymptotically distributed according to a Normal distribution with the following expectation:

$$\begin{aligned} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [D_n] &= \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\frac{e^{\kappa \langle V, A \rangle}}{n} + \sum_{i=1}^n \frac{e^{\kappa \langle V, X_i \rangle}}{n} \right] \\ &= \frac{e^{\kappa \langle V, A \rangle}}{n} + \mathbb{E}_{X \sim \mathcal{U}(\mathcal{S}^{d-1})} [e^{\kappa \langle V, X \rangle}]. \end{aligned} \quad (5.14)$$

Moreover, we have:

$$\begin{aligned}\mathbb{E}_{X \sim \mathcal{U}(\mathcal{S}^{d-1})} [e^{\kappa \langle V, X \rangle}] &= \int_{X \in \mathcal{S}^{d-1}} \frac{e^{\kappa \langle V, X \rangle}}{\mathcal{A}(\mathcal{S}^{d-1})} dX \\ &= \frac{1}{\mathcal{A}(\mathcal{S}^{d-1}) C_d(\kappa)},\end{aligned}\tag{5.15}$$

using the fact that $C_d(\kappa)$ is the normalizing constant of a vMF distribution, ensuring that its PDF (Equation (5.12)) sums to 1 when integrated on the unit hypersphere.

Let us define $\sigma = \text{Var}_{X \sim \mathcal{U}(\mathcal{S}^{d-1})} [e^{\kappa \langle V, X \rangle}]$. Although we do not need an explicit expression for σ , we know it is finite. Additionally, let $g : x \mapsto \frac{1}{x}$ be the inverse function. The CLT ensures that:

$$\sqrt{n} \left[D_n - \frac{1}{\mathcal{A}(\mathcal{S}^{d-1}) C_d(\kappa)} \right] \xrightarrow{D} \mathcal{N}(0, \sigma^2),\tag{5.16}$$

where $\xrightarrow{D}$ denotes convergence in distribution (Jacod et al., 2004). Moreover, since g is a differentiable function on $\mathbb{R}_+^*$, we use the *Delta method* (Oehlert, 1992) to infer that:

$$\sqrt{n} [g(D_n) - g(\frac{1}{\mathcal{A}(\mathcal{S}^{d-1}) C_d(\kappa)})] \xrightarrow{D} \mathcal{N}(0, \sigma^2 [g'(\frac{1}{\mathcal{A}(\mathcal{S}^{d-1}) C_d(\kappa)})]^2).\tag{5.17}$$

Replacing g and g' by their respective values, we obtain:

$$\sqrt{n} \left[\frac{1}{D_n} - C_d(\kappa) \mathcal{A}(\mathcal{S}^{d-1}) \right] \xrightarrow{D} \mathcal{N}(0, \sigma^2 (\mathcal{A}(\mathcal{S}^{d-1}) C_d(\kappa))^4).\tag{5.18}$$

Furthermore, recall that if a sequence $Z_1, Z_2, \dots$ of random variables converges in distribution to a random variable Z , then for all bounded continuous function ϕ , $\lim_{n \rightarrow +\infty} \mathbb{E} [\phi(Z_n)] = \mathbb{E} [\phi(Z)]$ (Jacod et al., 2004). Since for every n the random variable $Z_n = \sqrt{n} [\frac{1}{D_n} - C_d(\kappa) \mathcal{A}(\mathcal{S}^{d-1})]$ has bounded values, we can simply chose the identity function for ϕ to conclude that :

$$\lim_{n \rightarrow +\infty} \mathbb{E}_{X_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\sqrt{n} \left[\frac{1}{D_n} - C_d(\kappa) \mathcal{A}(\mathcal{S}^{d-1}) \right] \right] = 0,\tag{5.19}$$

which is equivalent to:

$$\mathbb{E}_{X_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\frac{1}{D_n} \right] = C_d(\kappa) \mathcal{A}(\mathcal{S}^{d-1}) + o\left(\frac{1}{\sqrt{n}}\right).\tag{5.20}$$

Finally, by multiplying Equation (5.20) by $\frac{e^{\kappa \langle V, A \rangle}}{n}$, we obtain Equation (5.11), concluding the proof. $\square$

5.B Asymptotic Behavior of vMF Exploration in $d = 2$ dimensions (Proof of Proposition 5.3, Part 1)

We now prove Proposition 5.3 when $d = 2$. In 2 dimensions, the vMF distribution takes the special form of the von Mises (vM) distribution (Mardia et al., 2009) which, instead of describing

the distribution of the dot product between V and $\tilde{V}$, describes the distribution of their angle θ . The PDF of a von Mises distribution is defined as follows:

$$\forall \theta \in [-\pi, \pi], f_{\text{vM}}(\theta \mid \kappa) = \frac{e^{\kappa \cos(\theta)}}{2\pi I_0(\kappa)}. \quad (5.21)$$

Let us define θ_0 as the angle between V and A . In this section, we prove that:

$$P_{\text{vMF-exp}}(A \mid n, d = 2, \kappa) = \frac{e^{\kappa \cos(\theta_0)}}{n I_0(\kappa)} + \mathcal{O}\left(\frac{1}{n^2}\right). \quad (5.22)$$

Proof. By definition,

$$P_{\text{vMF-exp}}(A \mid n, d = 2, \kappa) = \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^1)} \left[\mathbb{P}(\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})) \right], \quad (5.23)$$

where $\mathcal{S}_{\text{Voronoi}}(X_i \mid \mathcal{X}_n) = \{\tilde{V} \in \mathcal{S}^{d-1}, \forall j \in \mathcal{I}_n, \langle \tilde{V}, X_i \rangle \geq \langle \tilde{V}, X_j \rangle\}$. Let us call $\mathcal{Y}_n = \{Y_i\}$ the result of the permutation of the indices of $\mathcal{X}_n$ such that the (signed) angles β_i between A and Y_i are sorted in increasing order. Since the $\{X_i\}$ are i.i.d. and uniformly distributed on the circle, then the angles between A and the $\{X_i\}$ are i.i.d. and uniformly distributed on $[0, 2\pi]$. Therefore, the set $\{\beta_i\}$ is the set of the order statistics of n i.i.d. random variables uniformly distributed on $[0, 2\pi]$. Consequently, the set $\{\frac{\beta_i}{2\pi}\}$ is the set of the order statistics of n i.i.d. random variables uniformly distributed on $[0, 1]$, which are known to follow Beta distributions (Gentle, 2009) defined as follows:

$$\forall 1 \leq i \leq n, \frac{\beta_i}{2\pi} \sim \text{Beta}(i, n+1-i). \quad (5.24)$$

As a consequence, we have:

$$\mathbb{E}[\beta_1] = \frac{2\pi}{n+1}, \quad (5.25)$$

$$\mathbb{E}[\beta_n] = \frac{2\pi n}{n+1}, \quad (5.26)$$

$$\text{Var}[\beta_1] = \text{Var}[\beta_n] = \frac{4\pi^2 n}{(n+1)^2(n+2)}. \quad (5.27)$$

Moreover, for given values of Y_i , we can see from Figure 5.6 that, in 2 dimensions, Voronoï cells are arcs of the circle and are delimited by perpendicular bisectors of two neighboring points. Specifically, the Voronoï cell of A is delimited by the perpendicular bisector of A and Y_1 on one side, and the perpendicular bisector of A and Y_n on the other side. By denoting θ the (signed) angle between V and $\tilde{V}$, we have:

$$\begin{aligned} \mathbb{P}(\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})) &= \mathbb{P}\left(\theta \in \left[\theta_0 + \frac{\beta_n - 2\pi}{2}, \theta_0 + \frac{\beta_1}{2}\right] \mid \theta \sim \text{vM}(0, \kappa), \beta_1, \beta_n\right) \\ &= \int_{\theta_0 + \frac{\beta_n - 2\pi}{2}}^{\theta_0 + \frac{\beta_1}{2}} f_{\text{vM}}(\theta \mid \kappa) d\theta. \end{aligned} \quad (5.28)$$

5.B.1 Zero-Order Estimate

Let us study the zero-order approximation of $f_{\text{vM}}(\theta \mid \kappa)$ near θ_0 :

$$\begin{aligned}
\mathbb{E}_{\beta_1, \beta_n} \left[\int_{\theta_0 + \frac{\beta_n - 2\pi}{2}}^{\theta_0 + \frac{\beta_1}{2}} f_{\text{vM}}(\theta \mid \kappa) d\theta \right] &= \mathbb{E}_{\beta_1, \beta_n} \left[f_{\text{vM}}(\theta_0 \mid \kappa) \left(\theta_0 + \frac{\beta_1}{2} - \left(\theta_0 + \frac{\beta_n - 2\pi}{2} \right) \right) \right] \\
&= \mathbb{E}_{\beta_1, \beta_n} \left[f_{\text{vM}}(\theta_0 \mid \kappa) \left(\pi - \frac{\beta_n - \beta_1}{2} \right) \right] \\
&= \pi f_{\text{vM}}(\theta_0 \mid \kappa) \mathbb{E}_{\beta_1, \beta_n} \left[1 - \frac{\beta_n - \beta_1}{2\pi} \right] \\
&= \frac{e^{\kappa \cos(\theta_0)}}{2I_0(\kappa)} \left(1 - \mathbb{E}_{\beta_1, \beta_n} \left[\frac{\beta_n}{2\pi} \right] + \mathbb{E}_{\beta_1, \beta_n} \left[\frac{\beta_1}{2\pi} \right] \right) \\
&= \frac{e^{\kappa \cos(\theta_0)}}{2I_0(\kappa)} \frac{n+1 - n+1}{n+1} \\
&= \frac{e^{\kappa \cos(\theta_0)}}{2I_0(\kappa)} \frac{2}{n+1} \\
&= \frac{e^{\kappa \cos(\theta_0)}}{(n+1)I_0(\kappa)} \\
&= \frac{e^{\kappa \cos(\theta_0)}}{nI_0(\kappa)} - \frac{e^{\kappa \cos(\theta_0)}}{n(n+1)I_0(\kappa)} \\
&= \frac{e^{\kappa \cos(\theta_0)}}{nI_0(\kappa)} + \mathcal{O}\left(\frac{1}{n^2}\right).
\end{aligned} \tag{5.31}$$

This proves that, asymptotically, the contribution of the zero-order term of f_{vM} to the probability of selecting A is equal to the probability of selecting A using B-exp with the same κ value.

To understand how fast vMF-exp reaches its asymptotic behavior, we now need to study $R_0(\theta)$, the remainder of the Taylor series expansion of f_{vM} around θ_0 .

5.B.2 Bounding of the Remainder Term

We start by computing the first derivative of f_{vM} :

$$\forall \theta \in [0, 2\pi], |f'_{\text{vM}}(\theta \mid \kappa)| = \frac{|\sin(\theta)| \kappa e^{\kappa \cos(\theta)}}{I_0(\kappa)}, \tag{5.32}$$

which is bounded² on $[0, 2\pi]$ by $M = \frac{\kappa e^\kappa}{I_0(\kappa)}$. According to the Taylor-Lagrange inequality (Abramowitz et al., 1948), this in turn bounds the remainder term as follows:

$$\forall \theta \in [0, 2\pi], |R_0(\theta)| \leq M |\theta - \theta_0|. \tag{5.33}$$

²We note that a tighter bound could be found by studying the second derivative, but will not be necessary for the purpose of this proof.

In particular, this inequality holds for every $\theta \in [\theta_0 + \frac{\beta_n - 2\pi}{2}, \theta_0 + \frac{\beta_1}{2}]$, and so:

$$\begin{aligned}
 \int_{\theta=\theta_0+\frac{\beta_n-2\pi}{2}}^{\theta_0+\frac{\beta_1}{2}} |R_0(\theta)| d\theta &\leq \int_{\theta=\theta_0+\frac{\beta_n-2\pi}{2}}^{\theta_0+\frac{\beta_1}{2}} M|\theta - \theta_0| d\theta \\
 &= \int_{\theta=\theta_0}^{\theta_0+\frac{\beta_1}{2}} M(\theta - \theta_0) d\theta + \int_{\theta=\theta_0+\frac{\beta_n-2\pi}{2}}^{\theta_0} M(\theta_0 - \theta) d\theta \\
 &= \int_{\theta=0}^{\frac{\beta_1}{2}} M\theta d\theta - \int_{\theta=\frac{\beta_n-2\pi}{2}}^0 M\theta d\theta \\
 &= M \frac{\beta_1^2 + (\beta_n - 2\pi)^2}{8}.
 \end{aligned} \tag{5.34}$$

The above inequality holds when considering the expected values over uniformly distributed X_i :

$$\begin{aligned}
 \mathbb{E}_{\beta_1, \beta_n} \left[\int_{\theta=\theta_0+\frac{\beta_n-2\pi}{2}}^{\theta_0+\frac{\beta_1}{2}} |R_0(\theta)| d\theta \right] &\leq M \frac{\mathbb{E}_{\beta_1, \beta_n} [\beta_1^2] + \mathbb{E}_{\beta_1, \beta_n} [(\beta_n - 2\pi)^2]}{8} \\
 &= M \frac{\text{Var}_{\beta_1, \beta_n} [\beta_1] + (\mathbb{E}_{\beta_1, \beta_n} [\beta_1])^2 + \text{Var}_{\beta_1, \beta_n} [(\beta_n - 2\pi)] + (\mathbb{E}_{\beta_1, \beta_n} [\beta_n - 2\pi])^2}{8} \\
 &= \frac{M}{8} \left(\frac{2 \times 4\pi^2 n}{(n+1)^2(n+2)} + \frac{2 \times 4\pi^2}{(n+1)^2} \right) \\
 &= \frac{M\pi^2}{(n+1)(n+2)} \\
 &= \mathcal{O}\left(\frac{1}{n^2}\right).
 \end{aligned} \tag{5.35}$$

Since $\left| \mathbb{E}_{\beta_1, \beta_n} \left[\int_{\theta=\theta_0+\frac{\beta_n-2\pi}{2}}^{\theta_0+\frac{\beta_1}{2}} R_0(\theta) d\theta \right] \right| \leq \mathbb{E}_{\beta_1, \beta_n} \left[\int_{\theta=\theta_0+\frac{\beta_n-2\pi}{2}}^{\theta_0+\frac{\beta_1}{2}} |R_0(\theta)| d\theta \right]$, we have shown:

$$\mathbb{E}_{\beta_1, \beta_n} \left[\int_{\theta=\theta_0+\frac{\beta_n-2\pi}{2}}^{\theta_0+\frac{\beta_1}{2}} R_0(\theta) d\theta \right] = \mathcal{O}\left(\frac{1}{n^2}\right). \tag{5.36}$$

In summary, when combining the asymptotic behavior of the zero-order term and the remainder term, we conclude that when $d = 2$ we have:

$$P_{\text{vMF-exp}}(A \mid n, d = 2, \kappa) = \frac{e^{\kappa \cos(\theta_0)}}{nI_0(\kappa)} + \mathcal{O}\left(\frac{1}{n^2}\right). \tag{5.37}$$

This proves Proposition 5.3 when $d = 2$. Note that, comparing the asymptotic expressions for $P_{\text{B-exp}}(A \mid n, d = 2, \kappa)$ and $P_{\text{vMF-exp}}(A \mid n, d = 2, \kappa)$, also gives us a proof for Proposition 5.1 when $d = 2$. $\square$

5.C Asymptotic Behavior of vMF Exploration in $d > 2$ dimensions (Proofs of Proposition 5.3, Part 2, and of Proposition 5.4)

We now prove Proposition 5.3 when $d > 2$, starting with a series of intermediary lemmas. We subsequently justify the approximate expression of Proposition 5.4.

5.C.1 Intermediary Lemmas

We introduce a series of lemmas regarding the properties of the Voronoï cell of A when $\mathcal{X}_n \sim \mathcal{U}^{d-1}$. We recall that, for a given set of embedding vectors $\mathcal{X}_n$, we use the notation $\mathcal{X}_{n+1} = \mathcal{X}_n \cup \{A\}$.

Lemma 5.1. Let $d \in \mathbb{N}, d \geq 2$, $A \in \mathcal{S}^{d-1}$ and $n \in \mathbb{N}^*$. As before, let $\mathcal{A}(\mathcal{S}^{d-1})$ denote the surface area of $\mathcal{S}^{d-1}$. Then:

$$\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1}))] = \frac{\mathcal{A}(\mathcal{S}^{d-1})}{n+1}. \quad (5.38)$$

Proof. To compute this expectation, one can notice that:

$$\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1}))] = \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(X_{n+1} \mid \mathcal{X}_{n+1})) \mid X_{n+1} = A]. \quad (5.39)$$

Indeed, considering that A is known is equivalent to considering A as a random vector $X_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})$ with the constraint $X_{n+1} = A$. We will now show that the right part of Equation (5.39) is actually independent of the value of A .

Consider any point $A' \in \mathcal{S}^{d-1}$. One can always define a (not necessarily unique) rotation $R_{A,A'}$ such that $R_{A,A'}(A) = A'$. Since rotations preserve inner products, they also preserve areas of Voronoi cells, which means that for a given set of vectors $\mathcal{X}_{n+1}$, we have:

$$\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(X_{n+1} \mid \mathcal{X}_{n+1})) = \mathcal{A}(\mathcal{S}_{\text{Voronoi}}(R_{A,A'}(X_{n+1}) \mid R_{A,A'}(\mathcal{X}_{n+1}))). \quad (5.40)$$

Moreover, the image of the rotation of a random vector uniformly distributed on the hypersphere is also uniformly distributed, which means that:

$$\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1}) \Leftrightarrow R_{A,A'}(\mathcal{X}_{n+1}) \sim \mathcal{U}(\mathcal{S}^{d-1}). \quad (5.41)$$

Therefore:

$$\begin{aligned} & \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(X_{n+1} \mid \mathcal{X}_{n+1})) \mid X_{n+1} = A] \\ &= \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(R_{A,A'}(X_{n+1}) \mid R_{A,A'}(\mathcal{X}_{n+1}))) \mid X_{n+1} = A] \\ &= \mathbb{E}_{R_{A,A'}(\mathcal{X}_{n+1}) \sim \mathcal{U}(\mathcal{S}^{d-1})} [\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(R_{A,A'}(X_{n+1}) \mid R_{A,A'}(\mathcal{X}_{n+1}))) \mid R_{A,A'}(X_{n+1}) = A'] \quad (5.42) \\ &= \mathbb{E}_{R_{A,A'}(\mathcal{X}_n) \sim \mathcal{U}(\mathcal{S}^{d-1})} [\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(A' \mid R_{A,A'}(\mathcal{X}_n)))] \\ &= \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(A' \mid \mathcal{X}_n))]. \end{aligned}$$

This result proves that $\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})}[\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1}))]$ is independent of A . Then, we use this information along with Equation (5.39) to obtain:

$$\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})}[\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1}))] = \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})}[\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(X_{n+1} \mid \mathcal{X}_{n+1}))]. \quad (5.43)$$

Since $\sum_{i=1}^{n+1} \mathcal{A}(\mathcal{S}_{\text{Voronoi}}(X_i \mid \mathcal{X}_{n+1})) = \mathcal{A}(\mathcal{S}^{d-1})$ (Du, Faber, et al., 1999; Du, Gunzburger, et al., 2010) and the X_i are i.i.d., we derive:

$$\mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})}[\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(X_{n+1} \mid \mathcal{X}_{n+1}))] = \frac{\mathcal{A}(\mathcal{S}^{d-1})}{n+1}. \quad (5.44)$$

Combining Equations (5.39) with Equation (5.44) leads to Equation (5.38), concluding the proof. $\square$

Lemma 5.2. Let $d \in \mathbb{N}, d \geq 2$, $A \in \mathcal{S}^{d-1}$ and $n \in \mathbb{N}^*$. Then:

$$\exists \lambda \in \mathbb{R}, \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})} \tilde{V} \, d\tilde{V} \right] = \lambda A. \quad (5.45)$$

Proof. We want to prove that the average normal vector of the Voronoï cell of A and A are collinear, as illustrated in Figure 5.7. To do so, we will show that this average normal vector is invariant to any rotation around A . For every $\theta \in [0, 2\pi]$, we define $R_{A,\theta}$ as the rotation around A of the angle θ . As discussed in the proof of Lemma 5.1, $\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1}) \Leftrightarrow R_{A,\theta}(\mathcal{X}_n) \sim \mathcal{U}(\mathcal{S}^{d-1})$. Moreover, $R_{A,\theta}(A) = A$. Let us denote:

$$N(A \mid n) = \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})} \tilde{V} \, d\tilde{V} \right], \quad (5.46)$$

the expected normal vector of the Voronoï cell of A . Its image by the rotation $R_{A,\theta}$ verifies:

$$\begin{aligned} R_{A,\theta}(N(A \mid n)) &= R_{A,\theta} \left(\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})} \tilde{V} \, d\tilde{V} \right] \right) \\ &= \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(R_{A,\theta}(A) \mid R_{A,\theta}(\mathcal{X}_{n+1}))} \tilde{V} \, d\tilde{V} \right] \\ &= \mathbb{E}_{R_{A,\theta}(\mathcal{X}_n) \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid R_{A,\theta}(\mathcal{X}_{n+1}))} \tilde{V} \, d\tilde{V} \right] \\ &= N(A \mid n). \end{aligned} \quad (5.47)$$

This proves that $N(A \mid n)$ and A are collinear. $\square$

Lemma 5.3. With the same hypotheses as Lemma 5.2:

$$\lambda = \frac{\mathcal{A}(\mathcal{S}^{d-1})}{n+1} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1}), \tilde{V} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\max_i \langle \tilde{V}, X_i \rangle \right]. \quad (5.48)$$

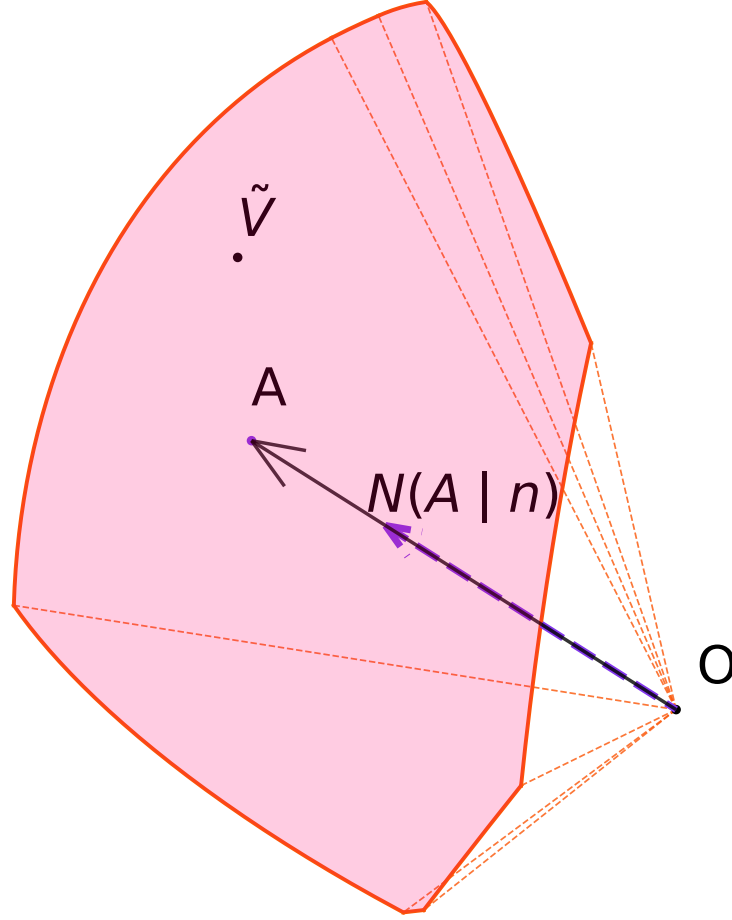


Figure 5.7: The Voronoi cell of A , $\mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})$, along with the average normal vector of the cell $N(A | n)$. On expectation, $(A | n)$ and A are collinear.

Proof. λ is defined as follows:

$$\begin{aligned}
 \lambda A &= \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} \tilde{V} \, d\tilde{V} \right] \\
 \implies \langle \lambda A, A \rangle &= \langle \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} \tilde{V} \, d\tilde{V} \right], A \rangle \\
 \Leftrightarrow \lambda &= \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} \langle \tilde{V}, A \rangle \, d\tilde{V} \right] \\
 \Leftrightarrow \lambda &= \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(X_{n+1} | \mathcal{X}_{n+1})} \langle \tilde{V}, X_{n+1} \rangle \, d\tilde{V} \mid X_{n+1} = A \right] \\
 \Leftrightarrow \lambda &= \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(X_{n+1} | \mathcal{X}_{n+1})} \max_i \langle \tilde{V}, X_i \rangle \, d\tilde{V} \mid X_{n+1} = A \right].
 \end{aligned} \tag{5.49}$$

Moreover, as done in the proof of Lemma 5.1, we can leverage the invariance by any rotation of the above expression to infer that the conditional expectation is actually independent of A :

$$\lambda = \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(X_{n+1} | \mathcal{X}_{n+1})} \max_i \langle \tilde{V}, X_i \rangle \, d\tilde{V} \right]. \tag{5.50}$$

Since, in the above equation, X_{n+1} has the same distribution as every element of $\mathcal{X}_{n+1}$, a similar expression for λ can be found using each $\mathcal{X}_{n+1}$ element. By summing them together, we obtain:

$$\begin{aligned}
(n+1)\lambda &= \sum_{j=1}^{n+1} \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(\mathcal{X}_j | \mathcal{X}_{n+1})} \max_i \langle \tilde{V}, X_i \rangle d\tilde{V} \right] \\
&= \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\sum_{j=1}^{n+1} \int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(\mathcal{X}_j | \mathcal{X}_{n+1})} \max_i \langle \tilde{V}, X_i \rangle d\tilde{V} \right] \\
&= \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}^{d-1}} \max_i \langle \tilde{V}, X_i \rangle d\tilde{V} \right] \\
&= \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}^{d-1}} \frac{\mathcal{A}(\mathcal{S}^{d-1}) \max_i \langle \tilde{V}, X_i \rangle}{\mathcal{A}(\mathcal{S}^{d-1})} d\tilde{V} \right] \\
&= \mathcal{A}(\mathcal{S}^{d-1}) \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\mathbb{E}_{\tilde{V} \sim \mathcal{U}(\mathcal{S}^{d-1})} [\max_i \langle \tilde{V}, X_i \rangle] \right],
\end{aligned} \tag{5.51}$$

which proves the lemma. $\square$

The last two lemmas are useful to describe the distribution of $\max_i \langle \tilde{V}, X_i \rangle$ when $\tilde{V}$ is fixed, $\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})$, and n is large.

Lemma 5.4. Let $B : (z_1, z_2) \mapsto \int_0^1 t^{z_1-1} (1-t)^{z_2-1} dt$ denote the Beta function. Let $d \geq 3$, $\tilde{V} \in \mathcal{S}^{d-1}$ and X be a random vector with $X \sim \mathcal{U}(\mathcal{S}^{d-1})$. Let F_{radial} be the cumulative distribution function (CDF) of $\langle \tilde{V}, X \rangle$. The Taylor series expansion of F_{radial} near 1 is:

$$F_{\text{radial}}(t) = 1 - \frac{2^{\frac{d-1}{2}}}{(d-1)B(\frac{1}{2}, \frac{d-1}{2})} (1-t)^{\frac{d-1}{2}} + o((1-t)^{\frac{d-1}{2}}). \tag{5.52}$$

Proof. The distribution of $\langle \tilde{V}, X \rangle$ has been studied in directional statistics (Mardia et al., 2009). Its PDF is known to be:

$$\begin{aligned}
f_{\text{radial}}(t) &= \frac{(1-t^2)^{\frac{d-1}{2}-1}}{B(\frac{1}{2}, \frac{d-1}{2})} \\
&= \frac{(1-t)^{\frac{d-1}{2}-1} (1+t)^{\frac{d-1}{2}-1}}{B(\frac{1}{2}, \frac{d-1}{2})} \\
&= \frac{(1-t)^{\frac{d-1}{2}-1} (2-(1-t))^{\frac{d-1}{2}-1}}{B(\frac{1}{2}, \frac{d-1}{2})} \\
&= \frac{2^{\frac{d-1}{2}-1} (1-t)^{\frac{d-1}{2}-1} (1 - \frac{(1-t)}{2})^{\frac{d-1}{2}-1}}{B(\frac{1}{2}, \frac{d-1}{2})} \\
&= \frac{2^{\frac{d-1}{2}-1} (1-t)^{\frac{d-1}{2}-1}}{B(\frac{1}{2}, \frac{d-1}{2})} \left(\sum_{i=0}^{\infty} \binom{\frac{d-1}{2}-1}{i} \left(\frac{1-t}{2} \right)^i \right).
\end{aligned} \tag{5.53}$$

The last line above was obtained using Newton's generalized binomial theorem for real exponent (Coolidge, 1949). It involves the term $\binom{\frac{d-1}{2}-1}{i} = \frac{(\frac{d-1}{2}-1)_i}{i!}$ with $(\cdot)_i$ the Pochhammer symbol used to designate a falling factorial (Abramowitz et al., 1948). We have obtained an expression of f_{radial} involving an infinite weighted sum of powers of $(1-t)$ with exponents greater or equal

to 0 since $d \geq 3$. Therefore, by uniqueness of the Taylor polynomial, we derive that the Taylor series expansion of f_{radial} near 1 is:

$$f_{\text{radial}}(t) = \frac{2^{\frac{d-1}{2}-1}(1-t)^{\frac{d-1}{2}-1}}{B(\frac{1}{2}, \frac{d-1}{2})} + o((1-t)^{\frac{d-1}{2}-1}). \quad (5.54)$$

Since by definition F_{radial} is the primitive of f_{radial} on $[-1, 1]$ and that $F_{\text{radial}}(1) = 1$, we can integrate the above equation to get:

$$\begin{aligned} F_{\text{radial}}(t) &= 1 - \frac{2}{d-1} \frac{2^{\frac{d-1}{2}-1}(1-t)^{\frac{d-1}{2}-1}}{B(\frac{1}{2}, \frac{d-1}{2})} + o((1-t)^{\frac{d-1}{2}-1}) \\ &= 1 - \frac{2^{\frac{d-1}{2}}(1-t)^{\frac{d-1}{2}}}{(d-1)B(\frac{1}{2}, \frac{d-1}{2})} + o((1-t)^{\frac{d-1}{2}}). \end{aligned} \quad (5.55)$$

Since this is exactly the Equation (5.52), this completes the proof. $\square$

Lemma 5.5. Let $d \geq 3$, $\tilde{V} \in \mathcal{S}^{d-1}$ and let F_{radial} be defined as in Lemma 5.4. For $n \in \mathbb{N}^*$, let $\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})$ be a set of n i.i.d. random vectors uniformly distributed on $\mathcal{S}^{d-1}$, and let F_n be the CDF of $\max_i \langle \tilde{V}, X_i \rangle$. Then, for $u \in [-1, 1]$:

$$\lim_{n \rightarrow +\infty} F_n(a_n u + b_n) = e^{(-1+\gamma u)^{\frac{-1}{\gamma}}}, \quad (5.56)$$

where $\gamma = -\frac{2}{d-1}$, $a_n = \frac{1}{2} \left(\frac{(d-1)B(\frac{1}{2}, \frac{d-1}{2})}{n} \right)^{\frac{2}{d-1}}$ with B the Beta function, and $b_n = 1 - \frac{2}{d-1} a_n$.

Proof. The proof relies on the Fisher–Tippett–Gnedenko theorem (Gnedenko, 1943) which states that if there exists a couple of sequences a_n and b_n such that the left term of Equation (5.56) converges, then its limit should be the CDF of a Generalized Extreme Value distribution (GEV) with shape parameter γ , which is the right term of Equation (5.56). Theorem 5 of (Gnedenko, 1943) provides a necessary and sufficient convergence condition for a random variable with maximal value $x_{\max}$ and CDF F , provided that $\gamma < 0$:

$$\lim_{t \rightarrow 0^+} \frac{1 - F(x_{\max} - u t)}{1 - F(x_{\max} - t)} = u^{\left(\frac{-1}{\gamma}\right)} \text{ for all } u > 0. \quad (5.57)$$

Recall that Lemma 5.4 gives us the Taylor expansion of F_{radial} near 1 : $F_{\text{radial}}(t) = 1 - K(1-t)^{\frac{d-1}{2}} + o((1-t)^{\frac{d-1}{2}})$ with $K = \frac{2^{\frac{d-1}{2}}}{(d-1)B(\frac{1}{2}, \frac{d-1}{2})}$. Knowing that $x_{\max} = 1$, we obtain that, $\forall u > 0$:

$$\begin{aligned} \lim_{t \rightarrow 0^+} \frac{1 - F_{\text{radial}}(1 - u t)}{1 - F_{\text{radial}}(1 - t)} &= \lim_{t \rightarrow 0^+} \frac{1 - (1 - K(ut)^{\frac{d-1}{2}}) + o((t)^{\frac{d-1}{2}})}{1 - (1 - K(t)^{\frac{d-1}{2}}) + o((t)^{\frac{d-1}{2}})} \\ &= \lim_{t \rightarrow 0^+} \frac{K(ut)^{\frac{d-1}{2}} + o((t)^{\frac{d-1}{2}})}{K(t)^{\frac{d-1}{2}} + o((t)^{\frac{d-1}{2}})} \\ &= u^{\left(\frac{d-1}{2}\right)}, \end{aligned} \quad (5.58)$$

which guarantees convergence and in the same time gives the value of $\gamma = -\frac{2}{d-1}$.

To find suitable sequences a_n and b_n , we can use the fact that $F_n(t) = F_{\text{radial}}(t)^n$ and study the behavior of $\ln F_n(t)$ near $t = 1$:

$$\begin{aligned} \ln F_n(t) &= \ln (F_{\text{radial}}(t)^n) \\ &= n \ln (F_{\text{radial}}(t)) \\ &= n(\ln(1 - K(1-t)^{\frac{-1}{\gamma}} + o((1-t)^{\frac{-1}{\gamma}}))) \text{ as } t \rightarrow 1^- \\ &= -nK((1-t)^{\frac{-1}{\gamma}} + o((1-t)^{\frac{-1}{\gamma}})) \text{ as } t \rightarrow 1^-. \end{aligned} \quad (5.59)$$

By defining $a_n = -\gamma(Kn)^\gamma$, $b_n = 1 - (Kn)^\gamma$ and doing the change of variable $u = \frac{t-b_n}{a_n}$, we see that:

$$\begin{aligned} t &= a_n u + b_n \\ &= 1 - (1 + \gamma u)(Kn)^\gamma. \end{aligned} \quad (5.60)$$

Since for every u , $\lim_{n \rightarrow +\infty} (1 + \gamma u)(Kn)^\gamma = 0$ (recall that $\gamma < 0$), the term $o((1-x)^{\frac{-1}{\gamma}})$ as $x \rightarrow 1^-$ is equivalent to $o(\frac{1}{n})$ as $n \rightarrow +\infty$. this means that:

$$\begin{aligned} \ln (F_n(a_n u + b_n)) &= -nK \left(((1 + \gamma u)(Kn)^\gamma)^{\frac{-1}{\gamma}} + o\left(\left(\frac{1}{n}\right)\right) \right) \text{ as } n \rightarrow +\infty. \\ &= -(1 + \gamma u)^{\frac{-1}{\gamma}} + o(1) \text{ as } n \rightarrow +\infty. \end{aligned} \quad (5.61)$$

We can now consider the exponential of the above expression to get our asymptotic maximum distribution:

$$\lim_{n \rightarrow +\infty} F_n(a_n u + b_n) = e^{-(1+\gamma u)^{\frac{-1}{\gamma}}}, \quad (5.62)$$

which concludes the proof. $\square$

Corollary 5.5. *With $\Gamma : z \mapsto \int_0^\infty t^{z-1} e^{-t} dt$ the Gamma function (Abramowitz et al., 1948), we have:*

$$\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\max_i \langle \tilde{V}, X_i \rangle \right] = 1 - \frac{\Gamma(\frac{d+1}{d-1})}{2} \left(\frac{(d-1)B(\frac{1}{2}, \frac{d-1}{2})}{n} \right)^{\frac{2}{d-1}} + o\left(\frac{1}{n^{\frac{2}{d-1}}}\right). \quad (5.63)$$

Proof. According to the Portmanteau theorem (Billingsley, 2013), Lemma 5.5 is equivalent to:

$$\frac{\max_i \langle \tilde{V}, X_i \rangle - b_n}{a_n} \xrightarrow{D} \text{GEV}(\gamma), \quad (5.64)$$

where $\text{GEV}(\gamma)$ is a generalized extreme value distribution with shape parameter γ (Gnedenko, 1943). Recall that if a sequence $Z_1, Z_2, \dots$ of random variables converges in distribution to a random variable Z , then for all bounded continuous function ϕ , $\lim_{n \rightarrow +\infty} \mathbb{E}[\phi(Z_n)] = \mathbb{E}[\phi(Z)]$. Since

$\frac{\max_i \langle \tilde{V}, X_i \rangle - b_n}{a_n}$ is bounded for every n , we can consider the identity function for ϕ and obtain:

$$\lim_{n \rightarrow +\infty} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\frac{\max_i \langle \tilde{V}, X_i \rangle - b_n}{a_n} \right] = \mathbb{E} [\text{GEV}(\gamma)] = \frac{\Gamma(1-\gamma) - 1}{\gamma}. \quad (5.65)$$

Replacing γ , a_n and b_n by their respective expressions, it implies that:

$$\begin{aligned} \lim_{n \rightarrow +\infty} \frac{\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [\max_i \langle \tilde{V}, X_i \rangle] - 1 + (Kn)^{-\frac{2}{d-1}}}{(Kn)^{-\frac{2}{d-1}}} + \Gamma\left(\frac{d-1}{d-1}\right) - 1 &= 0 \\ \implies \lim_{n \rightarrow +\infty} \frac{\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [\max_i \langle \tilde{V}, X_i \rangle] - 1 + K^{-\frac{2}{d-1}} \Gamma\left(\frac{d-1}{d-1}\right)}{n^{-\frac{2}{d-1}}} &= 0. \end{aligned} \quad (5.66)$$

Since $K^{-\frac{2}{d-1}} = \frac{1}{2} \left(\frac{(d-1)\text{B}(\frac{1}{2}, \frac{d-1}{2})}{n} \right)^{\frac{2}{d-1}}$, this is equivalent to writing:

$$\begin{aligned} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [\max_i \langle \tilde{V}, X_i \rangle] - 1 + \frac{1}{2} \left(\frac{(d-1)\text{B}(\frac{1}{2}, \frac{d-1}{2})}{n} \right)^{\frac{2}{d-1}} \Gamma\left(\frac{d-1}{d-1}\right) &= o\left(\frac{1}{n^{\frac{2}{d-1}}}\right) \\ \Leftrightarrow \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} [\max_i \langle \tilde{V}, X_i \rangle] &= 1 - \Gamma\left(\frac{d-1}{d-1}\right) \frac{1}{2} \left(\frac{(d-1)\text{B}(\frac{1}{2}, \frac{d-1}{2})}{n} \right)^{\frac{2}{d-1}} + o\left(\frac{1}{n^{\frac{2}{d-1}}}\right). \end{aligned} \quad (5.67)$$

We have thus obtained Equation (5.63), concluding the proof of the corollary. $\square$

5.C.2 Proof of Proposition 5.3

We now return to Proposition 5.3. In this section we consider the case of vMF-exp when $d > 2$ and X_i embeddings are uniformly distributed on $\mathcal{S}^{d-1}$. Under those assumptions:

$$P_{\text{vMF-exp}}(a \mid n, d, V, \kappa) = \frac{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1})}{n} + \mathcal{O}\left(\frac{1}{n^{1+\frac{2}{d-1}}}\right). \quad (5.68)$$

Proof. Similarly to the 2 dimensional case, the definition of $P_{\text{vMF-exp}}(a \mid n, d, V, \kappa)$ is:

$$P_{\text{vMF-exp}}(A \mid n, d, V, \kappa) = \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\mathbb{P}(\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1}) \mid \tilde{V} \sim \text{vMF}(V, \kappa)) \right], \quad (5.69)$$

which can be written using the PDF of the vMF distribution:

$$P_{\text{vMF-exp}}(A \mid n, d, V, \kappa) = \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})} f_{\text{vMF}}(\tilde{V} \mid V, \kappa) d\tilde{V} \right]. \quad (5.70)$$

As done in the 2D case, we study the Taylor expansion of f_{vMF} near A :

$$\begin{aligned}
 \forall \tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1}), f_{\text{vMF}}(\tilde{V} \mid \kappa, V) &= C_d(\kappa) e^{\kappa \langle V, \tilde{V} \rangle} \\
 &= C_d(\kappa) e^{\kappa \langle V, A \rangle} e^{\kappa \langle V, \tilde{V} - A \rangle} \\
 &= f_{\text{vMF}}(A \mid V, \kappa) \sum_{i=0}^{\infty} \frac{(\kappa \langle V, \tilde{V} - A \rangle)^i}{i!} \\
 &= f_{\text{vMF}}(A \mid V, \kappa) (1 + \kappa \langle V, \tilde{V} - A \rangle + R_1(\tilde{V})).
 \end{aligned} \tag{5.71}$$

with $R_1(\tilde{V}) = \sum_{i=2}^{\infty} \frac{(\kappa \langle V, \tilde{V} - A \rangle)^i}{i!}$. Leveraging the linearity property of both integration and expectation (Jacod et al., 2004), we can study $P_{\text{vMF-exp}}(A \mid n, d, V, \kappa)$ by assessing separately the contribution of the different terms of the expansion of f_{vMF} in:

$$\begin{aligned}
 P_{\text{vMF-exp}}(A \mid n, d, V, \kappa) &= \\
 \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})} f_{\text{vMF}}(A \mid V, \kappa) (1 + \kappa \langle V, \tilde{V} - A \rangle + R_1(\tilde{V})) d\tilde{V} \right].
 \end{aligned} \tag{5.72}$$

However, contrary to the 2D case where $\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})$ is always defined as the arc between 2 angles on the circle, for $d > 2$ the shape of $\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})$ is highly dependent of the layout of the elements of $\mathcal{X}_n$ that share a frontier with A . Figure 5.8 provides an illustration of the complexity and diversity of the shapes of Voronoi cells for uniformly sampled points on the 3D sphere.

As a consequence, expliciting the bounds of integration, as we did in the 2D case, can be somewhat tedious. Instead, we will leverage the geometrical properties of the problem at hand to estimate $P_{\text{vMF-exp}}(A \mid n, d, V, \kappa)$. We start with the zero-order term.

5.C.2.1 Zero-Order Term

Since the zero-order term is constant, its integral over $\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})$ can be expressed as:

$$\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})} f_{\text{vMF}}(A \mid V, \kappa) d\tilde{V} = f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})), \tag{5.73}$$

where $\mathcal{A}(\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1}))$ is the value of the surface area of $\mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})$. To assess the expected value of the above equation for uniformly distributed $\mathcal{X}_n$, we use Lemma 5.1 and obtain:

$$\begin{aligned}
 \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A \mid \mathcal{X}_{n+1})} f_{\text{vMF}}(A \mid V, \kappa) d\tilde{V} \right] &= \frac{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1})}{n+1} \\
 &= \frac{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1})}{n} + \mathcal{O}\left(\frac{1}{n^2}\right).
 \end{aligned} \tag{5.74}$$

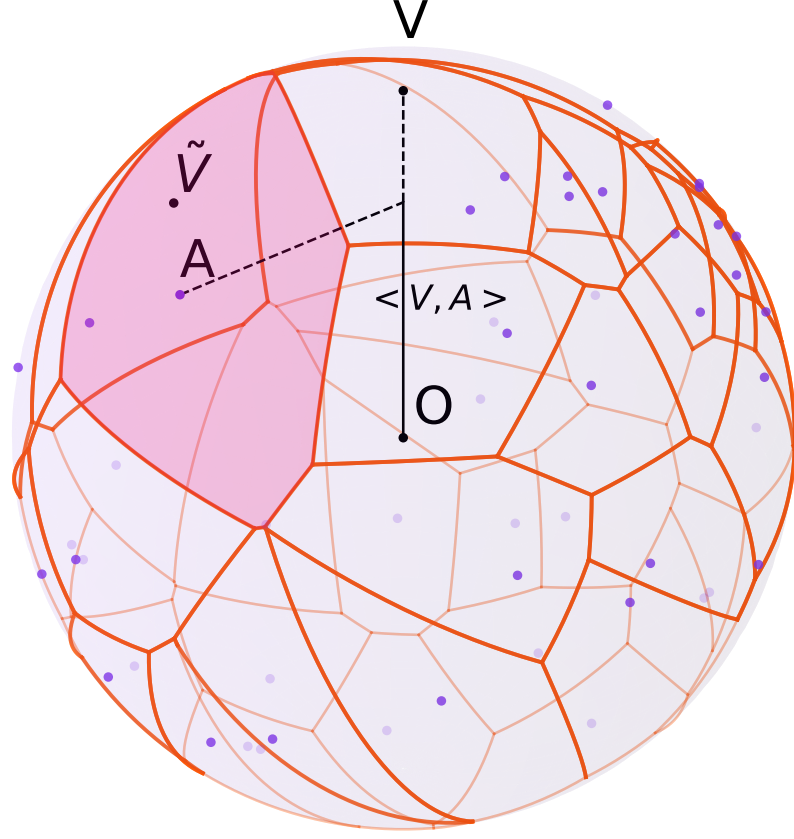


Figure 5.8: For $d = 3$: vMF-exp explores the action A when $\tilde{V}$ lies in its Voronoï cell, shown in red.

5.C.2.2 First-Order Term

We want to estimate the value of:

$$\begin{aligned} & \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} f_{\text{vMF}}(A | V, \kappa) \kappa \langle V, \tilde{V} - A \rangle d\tilde{V} \right] \\ &= f_{\text{vMF}}(A | V, \kappa) \kappa \left(\langle V, \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} \tilde{V} d\tilde{V} \right] \rangle - \frac{\langle V, A \rangle \mathcal{A}(\mathcal{S}^{d-1})}{n} \right). \end{aligned} \quad (5.75)$$

Using Lemmas 5.2 and 5.3 as well as Corollary 5.5, the left term inside the parentheses is:

$$\begin{aligned} & \langle V, \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} \tilde{V} d\tilde{V} \right] \rangle \\ &= \langle V, A \rangle \frac{\mathcal{A}(\mathcal{S}^{d-1})}{n+1} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1}), \tilde{V} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\max_i \langle \tilde{V}, X_i \rangle \right] \\ &= \langle V, A \rangle \frac{\mathcal{A}(\mathcal{S}^{d-1})}{n+1} \left(1 - \frac{\Gamma(\frac{d+1}{d-1})}{2} \left(\frac{(d-1)B(\frac{1}{2}, \frac{d-1}{2})}{n} \right)^{\frac{2}{d-1}} + o\left(\frac{1}{n^{\frac{2}{d-1}}}\right) \right). \end{aligned} \quad (5.76)$$

Reinjecting this expression into Equation (5.75) gives the following expression for the contribution of the first-order term to the probability of sampling A :

$$\begin{aligned} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} & \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} f_{\text{vMF}}(A | V, \kappa) \kappa \langle V, \tilde{V} - A \rangle d\tilde{V} \right] \\ &= -f_{\text{vMF}}(A | V, \kappa) \frac{\mathcal{A}(\mathcal{S}^{d-1})}{n+1} \kappa \langle V, A \rangle \left(\frac{\Gamma(\frac{d+1}{d-1})}{2} \left(\frac{(d-1)\text{B}(\frac{1}{2}, \frac{d-1}{2})}{n} \right)^{\frac{2}{d-1}} + o\left(\frac{1}{n^{\frac{2}{d-1}}}\right) \right). \end{aligned} \quad (5.77)$$

5.C.2.3 Remainder Term

As done in the 2D proof, we leverage the Taylor-Lagrange inequality (Abramowitz et al., 1948). The second derivative of the function $f(x) = C_d(\kappa)e^{\kappa x}$ is $f(x)^{(2)} = \kappa^2 f(x)$, which is bounded on $x \in [-1, 1]$ by $M = \kappa^2 C_d(\kappa)e^{\kappa x}$. This implies that:

$$\begin{aligned} |R_1(\tilde{V})| &\leq \frac{M \langle V, \tilde{V} - A \rangle^2}{2} \\ &\leq \frac{M \|\tilde{V} - A\|_2^2}{2} \text{ (according to the Cauchy-Schwarz inequality (Jacod et al., 2004))} \\ &= M(1 - \langle \tilde{V}, A \rangle). \end{aligned} \quad (5.78)$$

This inequality holds for every $\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})$ when $\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})$, which means that:

$$\begin{aligned} \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} & \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} f_{\text{vMF}}(A | V, \kappa) |R_1(\tilde{V})| d\tilde{V} \right] \\ &\leq f_{\text{vMF}}(A | V, \kappa) \mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} M(1 - \langle \tilde{V}, A \rangle) d\tilde{V} \right] \\ &= f_{\text{vMF}}(A | V, \kappa) \frac{\mathcal{A}(\mathcal{S}^{d-1})}{n+1} M(1 - \mathbb{E}_{\mathcal{X}_{n+1} \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}^{d-1}} \max_i \langle \tilde{V}, X_i \rangle d\tilde{V} \right]) \\ &= f_{\text{vMF}}(A | V, \kappa) M \frac{\mathcal{A}(\mathcal{S}^{d-1})}{n+1} \left(\frac{\Gamma(\frac{d+1}{d-1})}{2} \left(\frac{(d-1)\text{B}(\frac{1}{2}, \frac{d-1}{2})}{n} \right)^{\frac{2}{d-1}} + o\left(\frac{1}{n^{\frac{2}{d-1}}}\right) \right) \\ &= O\left(\frac{1}{n^{1+\frac{2}{d-1}}}\right). \end{aligned} \quad (5.79)$$

We used Lemmas 5.2 and 5.3 to go from line 2 to 3, and Corollary 5.5 to go from line 3 to 4. In essence, we have bounded the contribution of $R_1(\tilde{V})$ to the probability of sampling A as follows:

$$\mathbb{E}_{\mathcal{X}_n \sim \mathcal{U}(\mathcal{S}^{d-1})} \left[\int_{\tilde{V} \in \mathcal{S}_{\text{Voronoi}}(A | \mathcal{X}_{n+1})} f_{\text{vMF}}(A | V, \kappa) |R_1(\tilde{V})| d\tilde{V} \right] = O\left(\frac{1}{n^{1+\frac{2}{d-1}}}\right) \quad (5.80)$$

Finally, adding up Equations (5.74), (5.78), and (5.80), we conclude the proof of Proposition 5.3 for $d \geq 3$ and (via the first-order term) simultaneously justify the approximate probability $P_1(a | n, V, \kappa)$ introduced in Proposition 5.4. $\square$

5.D Similar Asymptotic Behavior of B-exp and vMF-exp for Large Action Sets (Proof of Proposition 5.1)

Finally, Propositions 5.2 and 5.3 allow us to derive Proposition 5.1, i.e., that in the setting of Section 5.4.1, we have:

$$\lim_{n \rightarrow +\infty} \frac{P_{\text{B-exp}}(a \mid n, d, V, \kappa)}{P_{\text{vMF-exp}}(a \mid n, d, V, \kappa)} = 1. \quad (5.81)$$

Proof. According to Proposition 5.2, we have:

$$P_{\text{B-exp}}(a \mid n, d, V, \kappa) = \frac{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1})}{n} + o\left(\frac{1}{n\sqrt{n}}\right). \quad (5.82)$$

Moreover, according to Proposition 5.3, we have:

$$P_{\text{vMF-exp}}(a \mid n, d, V, \kappa) = \frac{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1})}{n} + \begin{cases} \mathcal{O}\left(\frac{1}{n^2}\right) & \text{if } d = 2, \\ \mathcal{O}\left(\frac{1}{n^{1+\frac{2}{d-1}}}\right) & \text{if } d > 2. \end{cases} \quad (5.83)$$

Therefore:

$$\begin{aligned} \lim_{n \rightarrow +\infty} \frac{P_{\text{B-exp}}(a \mid n, d, V, \kappa)}{P_{\text{vMF-exp}}(a \mid n, d, V, \kappa)} &= \lim_{n \rightarrow +\infty} \frac{n}{n} \frac{P_{\text{B-exp}}(a \mid n, d, V, \kappa)}{P_{\text{vMF-exp}}(a \mid n, d, V, \kappa)} \\ &= \frac{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1}) + 0}{f_{\text{vMF}}(A \mid V, \kappa) \mathcal{A}(\mathcal{S}^{d-1}) + 0} \\ &= 1. \end{aligned} \quad (5.84)$$

□

5.E Link with Thompson Sampling

At first glance, one might draw some similarities between vMF-exp and Thompson Sampling (TS) with Gaussian prior for contextual bandits (Chapelle et al., 2011). Admittedly, vMF-exp shares a common spirit with TS, where action selection is preceded by sampling individual weights according to a Normal distribution centered on an observed context/state vector. However, vMF-exp also presents two major differences:

- Firstly, in vMF-exp, vector sampling is performed according to a vMF hyperspherical distribution, centered on the state embedding vector V . This choice of distribution ensures that vectors with the same inner product with the state vector have the same probability of being sampled, as illustrated in Figure 5.3a. This aligns better with the similarity used to retrieve nearest neighbors and, as emphasized in this chapter, leads to probabilities of exploring actions asymptotically comparable to Boltzmann Exploration (with better scalability) under the theoretical assumptions of Section 5.4.1.
- Secondly, vMF-exp is not designed to maximize the expected reward of a policy in an RL or contextual bandit environment and does not impose any parameter update strategy.

Instead, it serves as an action selection tool for any scenario where policy updates cannot be performed regularly (as in the batch RL setting commonly found in industrial applications), yet broad exploration must still be guaranteed between consecutive updates.

5.F Monte Carlo Simulations

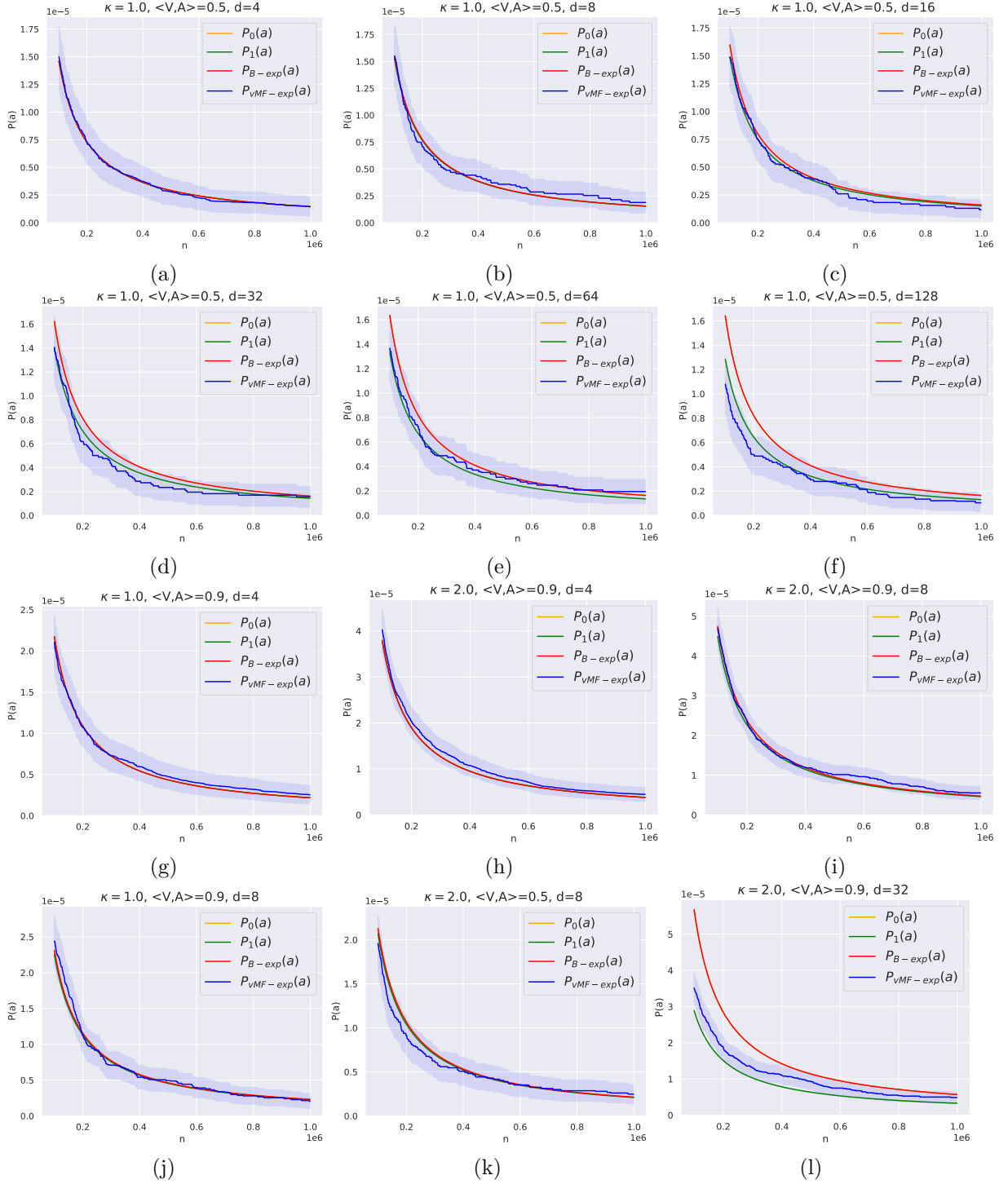


Figure 5.9: We report complete results for the Monte Carlo simulations presented and discussed in Section 5.4.3, involving more combinations of d , κ , and $\langle V, A \rangle$. We recall that $P_{B-exp}(a)$ and $P_0(a)$ are indistinguishable for this range of n values. We emphasize that the y-axis is on a 10^{-5} scale; hence, all probabilities are extremely close.

5.G Sampling from the von Mises-Fisher Distribution

5.G.1 Radial-tangent decomposition

Given a vector $\tilde{V} \in \mathcal{S}^{d-1}$ and a concentration $\kappa \in \mathbb{R}^+$, the algorithm described in (Pinzón et al., 2023) sample from a $\text{vMF}(V, \kappa)$ by leveraging the radial-tangent decomposition of the elements of $\mathcal{S}^{d-1}$. For any $\tilde{V} \in \mathcal{S}^{d-1}$, let's call $\tilde{t} = \langle V, \tilde{V} \rangle$. Then we have:

$$\tilde{V} = \tilde{t}V + \sqrt{1 - \tilde{t}^2}\tilde{V}_O \quad (5.85)$$

where the vector $\tilde{V}_O$ has a unit norm and is orthogonal to V .

5.G.2 vMF distribution

If, $\tilde{V} \sim \text{vMF}(V, \kappa)$, then :

- $\tilde{t}$ is a scalar valued random variable.
- $\tilde{V}_O$ is a random vector uniformly distributed on the $(d-2)$ dimensional sub-sphere that is centered at and perpendicular to V . For instance, for $d = 3$, this would mean a circle centered around V .
- $\tilde{t}$ and $\tilde{V}_O$ are independent.

Since the reciprocal is also true, t and $\tilde{V}_O$ can thus be separately sampled to obtain $\tilde{V}$.

5.G.3 Sampling $\tilde{t} = \langle V, \tilde{V} \rangle$

The PDF of $\tilde{t}$ is known (Fisher, 1953) and follows:

$$f_{\text{radial}}(t; \kappa, d) = \frac{(\kappa/2)^{\frac{d}{2}-1}}{\Gamma(\frac{1}{2})\Gamma(\frac{d-1}{2})I_{\frac{d}{2}-1}(\kappa)} e^{t\kappa} (1 - t^2)^{\frac{d-3}{2}} \quad (5.86)$$

This PDF can be used to sample t through rejection sampling (Gentle, 2009).

5.G.4 Sampling $\tilde{V}_O$

$\tilde{V}_O$ can be obtained by following the steps of algorithm 1.

Algorithm 1: Sample $\tilde{V}_O$

- 1 Sample vector U uniformly from $\mathcal{S}^{d-1}$;
 - 2 Compute projection of U on V : $W = \langle U, V \rangle V$;
 - 3 Subtract projection and normalize: $\tilde{V}_O = \frac{U-W}{\|U-W\|}$;
 - 4 **return** $\tilde{V}_O$
-

Note that a simple way to sample U uniformly on $\mathcal{S}^{d-1}$ is to sample d standard Gaussians independently (one for each dimension) and then normalize the resulting vector (Gentle, 2009).

5.G.5 Wrapping up

The vector $\tilde{V}$ can now be computed by summing the right term of Equation (5.86). Overall, we see that sampling $\tilde{V}$ from $\text{vMF}(V, \kappa)$ is **data-independent**, hence the scalability of the approach.

5.H Additional Experiments on a public dataset: Glove-25 word embeddings

While our main contributions in this work are theoretical, we aimed in the main paper to complement our key findings with experimental evaluations to illustrate the performance of vMF-exp. The main paper reported both Monte Carlo simulations, which involved reproducible synthetic data, and an online A/B test conducted on the XXX music streaming service, using real-world private data from this service. However, due to industrial constraints, we were unable to provide detailed information about the real-world data used in the A/B test or release our private dataset for reproducibility.

Understanding that some readers may wish to further explore our topic through reproducible experiments on public real-world data, we present an additional study in this section.

Specifically, in this section we report experiments comparing the behaviors of B-exp and vMF-exp for selecting tokens among the public Glove-25³ dataset of word embeddings (Pennington et al., 2014). All vectors represent a word token, and embeddings are learned using Word2Vec (Mikolov, K. Chen, et al., 2013) trained on 2 billion tweets from [X.com](https://x.com).

We use the public set of 1 million vectors of dimension $d = 25$, and as preprocessing we subtract to each vector the average of the entire set and divide it by its norm to ensure all vectors belong to the unit-sphere, leading to a set of unit-vectors of dimension 25, denoted $\mathcal{G}$.

The experiments follow the same protocol as the Monte Carlo estimations of Section 5.4.3, but instead of sampling $\mathcal{X}_n$ from the uniform spherical distribution, this time vectors are randomly sampled from $\mathcal{G}$. The vector V and A are also sampled from $\mathcal{G}$ so that $\langle V, A \rangle$ equals a fixed desired value. Given $\mathcal{X}_n$, we aim to assess the probability that A is selected when the current state vector is V , using either B-exp or vMF-exp. Specifically, we want to verify whether Propositions **P1**, **P2** and **P3** defined in Section 5.2 hold. We also want to assess to which extent the approximations obtained in Section 5.4 can anticipate the behavior of B-exp and vMF-exp, despite the fact that real-world embeddings are used instead of embeddings drawn from the uniform spherical distribution.

5.H.1 P1: Scalability

As explained earlier in the paper, the efficiency of B-exp is determined by the time required to compute $\langle V, X_i \rangle$ for all $X_i \in \mathcal{X}_n$ (Section 5.2.2), whereas for vMF-exp it is determined by the time required to find the approximate nearest neighbor (ANN) of $\tilde{V}$ within $\mathcal{X}_n$ (Section 5.3.2). We report in Table 5.1 the performances of popular ANN algorithms on the Glove-25 dataset, extracted from the benchmark⁴ of Aumüller et al. (2017). To be consistent with the standards of ANN literature (see for instance the report of the 2023 NeurIPS competition on ANN (Simhadri et al., 2024)), the performance metric used is the maximum *throughput*, measured in number of queries-per-second (QPS), for which the average recall of the exact top-10 neighbors is greater than 90%. The algorithms chosen are two implementations of hnsw (Malkov et al., 2016), one

³Available at <https://nlp.stanford.edu/projects/glove/>

⁴https://ann-benchmarks.com/glove-25-angular_10-angular.html

from the library faiss (Douze et al., 2024) and the other one from the library nmslib (Boytsov et al., 2013), as well as the algorithms scann (R. Guo et al., 2020; P. Sun et al., 2023) and NGT-qg (Iwasaki et al., 2018). We also report the throughput when using exhaustive search (named "bruteforce" in Table 5.1) which is informative of the limited efficiency of B-exp.

Table 5.1 shows that performing exhaustive computations on the Glove-25 dataset for nearest neighbors search leads to a throughput 2 to 3 orders of magnitude lower than using ANN methods. This observation directly impacts the relative efficiency of B-exp compared to vMF-exp on the Glove-25 dataset, leading us to conclude that vMF-exp is scalable and verifies **P1** while B-exp does not.

Table 5.1: Throughput of ANN algorithms on the Glove-25 dataset. The number of queries-per-second (QPS) is increased by 2 to 3 orders of magnitude compared to bruteforce.

Algorithm	bruteforce	hnwsm-faiss	hnsw-nmslib	scann	NGT-qg
QPS	34	6197	14080	23436	22733

5.H.2 P2,P3: Unrestricted Radius and Order Preservation

Figure 5.10 reports, for $\kappa = 1$, for $20000 \leq n \leq 100000$ and for $\langle V, A \rangle \in \{0.9, 0.3, 0, -0.3, -0.9\}$, the probabilities of sampling vector A when the current vector state is V , using either B-exp (Figure 5.10a) or vMF-exp (Figure 5.10b). Sampling is performed 30 million times to obtain a significant estimate of $P_{\text{vMF-exp}(a)}$.

Figure 5.10a illustrates the two properties that make Boltzmann exploration a popular choice for performing "soft" max on small action sets: the ability to select any action, even those with low similarity to V (**P2**), and the ordering of action selection probabilities based on the similarity to V .

Importantly, Figure 5.10b demonstrates that our method, vMF-exp, also satisfies these two important properties. Specifically, A always has a positive probability of being sampled, with this probability increasing as a function of $\langle V, A \rangle$. Thus, both B-exp and vMF-exp verify properties **P2** and **P3** on the Glove-25 dataset. Additionally, since vMF-exp is scalable (**P1**), this experiment demonstrates that vMF-exp is a relevant alternative to B-exp for discrete, very large action sets represented by embedding vectors.

5.H.3 Theoretical Approximations

In this section, we compare the probabilities of selecting A using B-exp and vMF-exp on the Glove-25 dataset with the probabilities predicted by the theoretical analysis in Section 5.4, as shown in Figure 5.11.

As observed in the subfigures, despite relaxing the uniform distribution assumption, the asymptotic expression from Proposition 5.2 holds across all configurations. Consequently, the yellow curve is indistinguishable from the red curve in every plot.

Also, $P_{\text{vMF-exp}}$ (blue) remains very close to $P_{\text{B-exp}}$ (red) for values of $\langle V, A \rangle$ near zero (Figures 5.11a, 5.11b, and 5.11c), as anticipated by Proposition 5.3 and 5.1. On Figure 5.11b and Figure

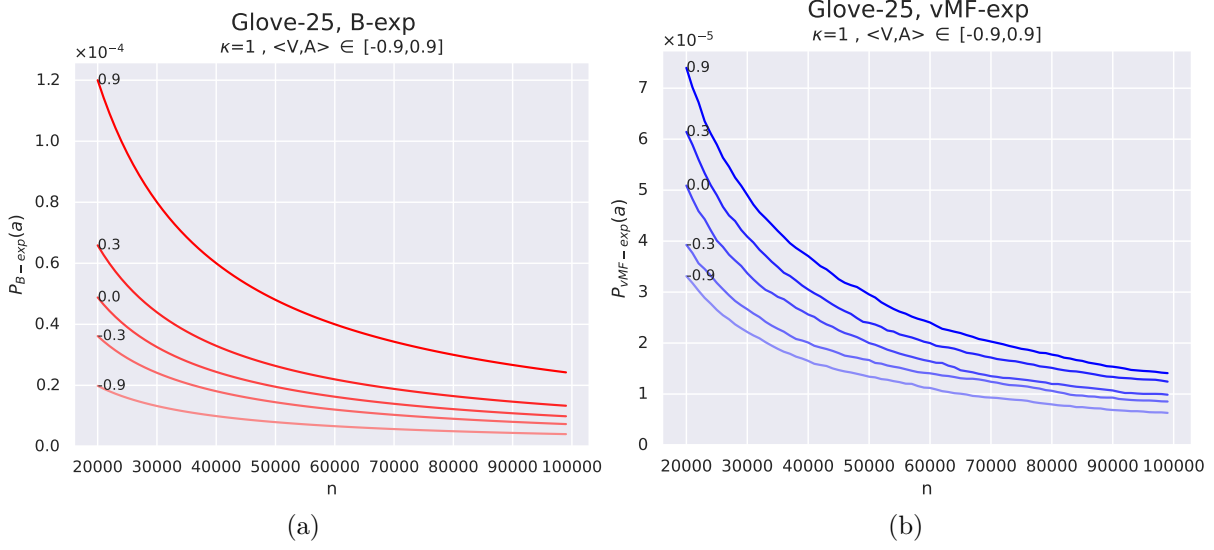


Figure 5.10: We report the observed probabilities of selecting vector A using B-exp (a) and vMF-exp (b), on the Glove-25 dataset, for $\kappa = 1$, $20000 \leq n \leq 100000$ and $\langle V, A \rangle \in \{0.9, 0.3, 0.0, -0.3, -0.9\}$. In this regime, for both methods, the probability of selecting A is strictly positive even for low values of $\langle V, A \rangle$ (**P2**) and is a strictly increasing function of $\langle V, A \rangle$ (**P3**).

5.11c, the small difference between $P_{vMF-\text{exp}}$ and $P_{B-\text{exp}}$ is anticipated by the alternate asymptotic expression (green) derived in Proposition 5.4. However, as $\langle V, A \rangle$ grows in absolute value (Figure 5.11d and Figure 5.11e), the difference between $P_{vMF-\text{exp}}$ and $P_{B-\text{exp}}$ increases faster than anticipated by Proposition 5.4, outlining the limits of the uniform distribution assumption.

Overall, the experiments show that the mathematical results obtained in Section 5.4 are useful to anticipate most of the respective behaviors of $P_{vMF-\text{exp}}$ and $P_{B-\text{exp}}$ on the Glove-25 dataset. To derive even more precise results and explain the behavior of vMF-exp on Figure 5.11d and Figure 5.11e, future work would benefit from modeling real-world embedding with other distributions to attempt to derive propositions equivalent to the ones presented in Section 5.4.

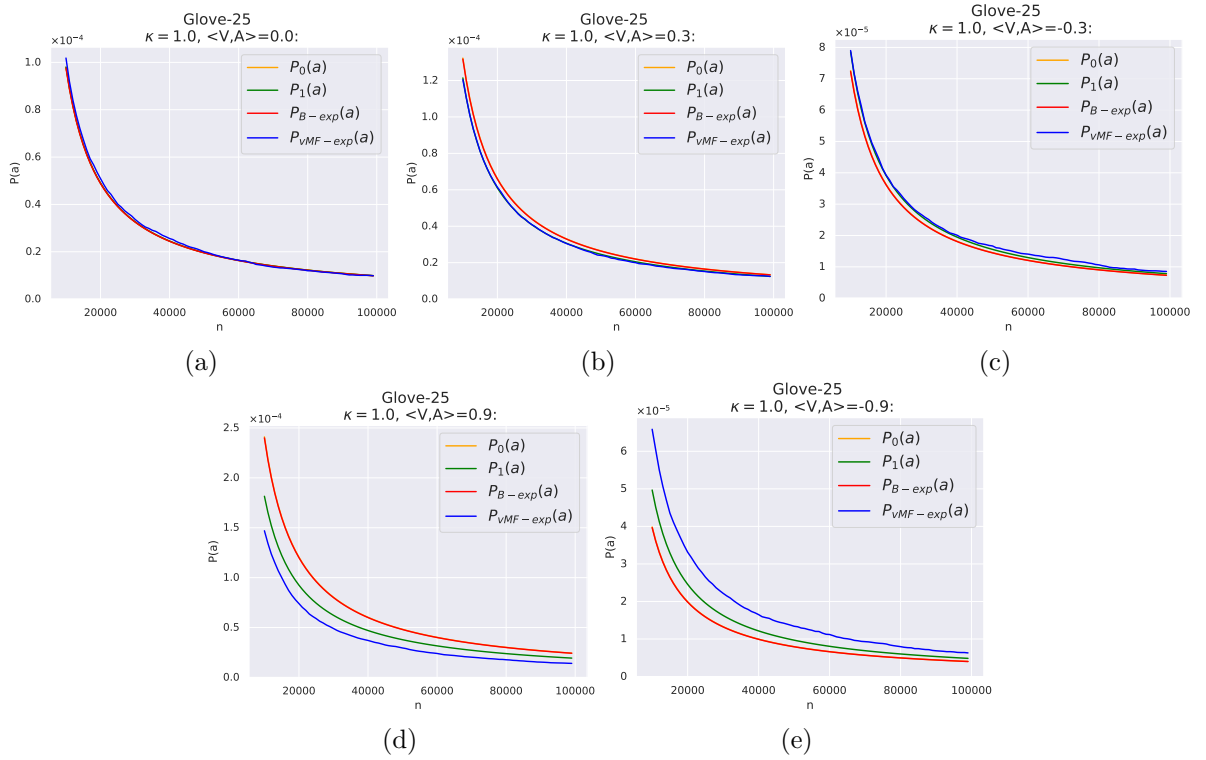


Figure 5.11: We compare the probabilities, on the Glove-25 dataset, of selecting A using B-exp and vMF-exp with the theoretical results obtained in Section 5.4. Despite the relaxation of the uniform distribution assumption, the expression from Proposition 5.2 holds for every configuration. Propositions 5.3 and 5.1 are also illustrated by (a), while (b) and (c) illustrate Proposition 5.4.

Chapter 6

Opening: Towards Reinforcement Learning for Sequential Music Recommendation

This brief chapter details our proposition to adapt the reinforcement learning paradigm to interactive music recommendation. It doesn't form a publication and does not present experimental results, but serves as a mean to share our reflections regarding future directions of research on large-scale music recommendation driven by contributions presented in this thesis.

6.1 Introduction

In Chapter 3 we studied the problem of scaling sequence modeling approaches to real-time playlist continuation on streaming platforms. We then showed that these models could be interpreted as probabilistic functions capturing the knowledge about how millions of users create their playlists. Chapter 5 studied the possibility of voluntarily sampling from a different distribution than the one learned, introducing a parameterized exploration method usable in real-time for large catalogs. An interesting perspective for future research is to study the possibility to learn distributions that outperform those fitted on organic user behavior.

RL is a machine learning framework where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards (Sutton et al., 2018). In the context of a music streaming platform, the agent would designate (part of) the recommendation system. Similarly to the way user engagement is computed, user feedback (see Figure 2.1 of Chapter 2) can be aggregated so that the agent's task boils down to maximizing a scalar value of particular interest. A sub-branch of RL, called *Imitation Learning* (Hussein et al., 2017), is dedicated to teaching models how to imitate human experts, highly successful at performing a task, through datasets of examples, much like we did in Chapter 3 when we assumed that user-made playlists were relevant sequences to predict. However, the majority of RL methods integrate observed reward

to continually models, further optimizing them for the task being solved. Observed reward hasn't been leveraged in the methods we developed in this thesis, and we believe that its proper consideration could lead to significantly increased performances in the future. For this reason, we introduce in this chapter elementary concepts of reinforcement learning, relate them to sequential music recommendation and propose some directions for future work on reinforcement learning based large-scale music recommender systems.

6.2 Background on Reinforcement Learning

This section introduces some elementary concepts of reinforcement learning to help practitioners of recommender systems understand how problems are handled with the RL paradigm.

6.2.1 Markov Decision Process

A *Markov Decision Process* (MDP) is a mathematical framework used to model decision-making in environments where outcomes are partly random and partly under the control of an agent (Sutton et al., 2018). Formally, a MDP is defined by a tuple (S, A, P, R, γ) , where S represents a set of states, A a set of actions, $P(s'|s, a)$ describes transition probabilities between states given an action, $R(s, a)$ is a reward function, possibly stochastic, defining the scalar valued reward that an agent will receive after performing action a when in state s , and γ a discount factor for future rewards. Action and state spaces in reinforcement learning can be either discrete, such as a robot selecting predefined movements or navigating through a grid (Brockman et al., 2016), or continuous, like controlling the precise speed and steering angle of a self-driving car (Kiran et al., 2021). An agent interacting with an MDP environment seeks to maximize long-term rewards accumulated from making sequential decisions.

The fundamental property of a MDP is the *Markov property* defined in Equation (6.1):

$$P(s_{t+1}, r_{t+1} \mid s_0, a_0, \dots, s_t, a_t) = P(s_{t+1}, r_{t+1} \mid s_t, a_t) \quad (6.1)$$

In plain words, it means that in MDP environments, the states and rewards observed are dependent on the states and actions taken at the time step immediately preceding, and independent of any older observation.

A sequence of chained interactions between an agent and the environment is called a *trajectory*, often named τ :

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_T, a_T, r_{T+1}) \quad (6.2)$$

The parameter T is called the horizon and determines the maximal number of steps that a trajectory may contain. The *return* of the trajectory τ can be defined for any time step $t \leq T$ as the expected cumulative reward that is gained between the step t and the step T . Formally :

$$G_t = \sum_{k=t}^T \gamma^{k-t} r_k \quad (6.3)$$

The discount factor γ is a mean of favoring reward obtained shortly after t in the computation of G_t . It also ensures the convenient mathematical property that G_t is defined even for an infinite value of the horizon T (provided that $\gamma < 1$ and that the reward is bounded), conveying a unified framework for environments with finite horizon, also called episodic environments, and those with infinite horizon.

6.2.2 Policy

To determine its actions based on the current state of the environment, an agent relies on a mapping from the set of states S to a probability distribution over the set of actions A called a *policy*. Formally, given a state s , the policy $\pi(a | s)$ provides the probability of taking action a in state s . When only a single action has a positive probability of 1 to be selected given a state, the policy is called *deterministic*, otherwise it is called *stochastic*.

In reinforcement learning, the goal is to find an *optimal policy* π^* , which maximizes the expected cumulative reward for the agent over time. This optimal policy dictates the best action to take in each state to achieve the highest long-term rewards.

6.3 Application to Interactive Track Mix

In the context of a recommender system, a MDP environment can be used to model the task of interactively recommending songs to a user, repeatedly leveraging feedback to update its predictions. We call this problem *interactive Track Mix*, which, similarly to what we did for automatic playlist continuation in Chapter 3, we formalize below.

6.3.1 Problem Setting

Let us consider a catalog $\mathcal{I} = \{1, \dots, N\}$ of $N \in \mathbb{N}^*$ songs available on a music streaming service. At time $t = 0$, a user may start a mix by selecting an initial song, and the recommender system must find additional songs to recommend recursively. The initial information available to the RS is the song that the user has selected, denoted i_0 . In reaction, the recommender system recommends a song i_1 and observes a feedback m_1 which is translated into a scalar reward r_1 according to some pre-defined function mapping feedback to reward. This process is repeated as long as the user keeps interacting with the recommender system, generating observations in the form of triplets $o_t = (i_t, m_t, r_t)$. The set of possible feedback $\mathcal{M}$ is defined by the actions listed in Table 2.1 of Chapter 2: $\mathcal{M} = \{stream, skip, collect, ban\}$.

The recommender system must maximize the overall reward of the Mix:

$$G_t = \sum_{k=0}^{+\infty} \gamma^k r_k \quad (6.4)$$

The task of interactive Track Mix can be seen as a variant of the Track Mix problem introduced in Chapter 3, where additional information about user feedback is available to the system after each recommendation, allowing it to update the predictions of future recommendations. This

more complex task, not studied in the thesis despite its interest for music streaming platforms, presents interesting research opportunities. We expose below how the work presented in this thesis could contribute to designing solutions for this problem and identifying areas open for research.

6.3.2 Our Approach

6.3.2.1 Presentation

We propose to model the interactive Track Mix task as a MDP where an RL agent must maximize its reward. At time t , We define the state of the environment in a recursive way:

$$s_t = \begin{cases} (i_t) & \text{if } t = 0 \\ \text{concat}(s_{t-1}, (i_t, m_t)) & \text{otherwise} \end{cases} \quad (6.5)$$

where "concat" designates the concatenation operation between two tuples.

The definition for the state s_t is a compromise between the Markov property (Equation (6.1), which imposes that r_{t+1} relies solely on the state s_t and the action a_{t+1} , and the sequence modeling assumption (Equation (2.7) in Chapter (2)) that the probability of observing a token at a given position is conditioned on the entire sequence preceding the token. In essence, this recursive definition of s_t translates the assumption that the song recommended at time $t + 1$ should depend on the list of songs previously listened in the mix $(i_0, \dots, i_t)$, as well as the feedback that the user has given so far $(m_1, \dots, m_t)$.

Regarding the definition of the action a_t , previous attempts at RL based large-scale item recommendation (M. Chen, Beutel, et al., 2019; M. Chen, B. Chang, et al., 2021; M. Chen, C. Xu, et al., 2022) have mainly considered that it should naturally be equal to the next item to recommend, making the action set equivalent to the catalog (i.e. $A = \mathcal{I}$). Instead, we propose to leverage the fact that the task is being solved by a recommender system already used in the production environment of an online service. We thus assume that it is equipped with a dictionary of song embeddings of dimension d , $\mathcal{E} = \{e_1, \dots, e_N\}$. We also assume that an approximate nearest neighbors system is available.

Consequently, we define the actions set as the vector space where the song embeddings lie, i.e. $A = \mathbb{R}^d$. At time t , the agent must select an action $a_t \in \mathbb{R}^d$ and give it to the production recommender system, which is in charge of selecting the exact song i_t to recommend.

To select an action a_t using the sequential state definition s_t , the agent can rely on a policy built based on our represent-then-aggregate (RTA) framework presented in Chapter 3. The RTA model can output a state embedding, treated as the mean direction of a von Mises-Fisher distribution of concentration parameter κ from which the actual action a_t can be sampled as described in Chapter 5. The exploration-exploitation trade-off of the policy is controlled by the parameter κ , leaving the possibility to describe a deterministic policy when $\kappa = +\infty$.

6.3.2.2 Analysis

The above modeling of the interactive Track Mix task as a Markov decision process makes it possible to leverage the work developed throughout this thesis to describe policies that are scalable, use relevant architectures and for which exploration-exploitation trade-off is efficiently parameterized. It leaves open, among others, the question of how to train those policies.

Although the discrete state set has a high cardinality due to the combinatorics involved when taking into account sequential information, the number of trainable parameters of the proposed family of policies is entirely controlled by the dimension d of the song embeddings. A similar state representation function was proposed in previous work (M. Chen, Beutel, et al., 2019; M. Chen, B. Chang, et al., 2021; M. Chen, C. Xu, et al., 2022), but it was seen as a preliminary step of the policy which eventually had to output sampling probabilities for each items of the catalog, treated as a large action set of millions of elements.

Because of the real-time imperatives mentioned in Chapter 5, this discrete definition of the action set forces stochastic policies to retrieve a small subset of candidate actions before computing probabilities, resulting in an effective probability of being recommend equal to zero for most items given a state. In contrast, our approach to exploration based on von Mises-Fisher sampling allows the number of different songs that can be recommended given a state to be determined by the concentration parameter κ . Given that models can only be updated by training batches comprising of millions of user sessions, we expect that the ability to explore more will help counter-balance the low frequency of model iterations and speed up convergence.

Another consequence of modeling the action space as a continuous embedding space is that the actual operation of selecting songs to recommend given an (action) embedding is now considered to be part of the environment’s behavior. Although this might seem counter-intuitive, this is in fact more aligned with some aspects of real-world production recommender systems, which are compounds made of several modules collaborating together to output recommendations visible to users. For instance, a RS system might have a module guaranteeing that a set of rules, such as preventing two songs from the same artist or the same album to be recommended one after another, is enforced. Another part of the RS might be responsible for intertwining recommendations proposed by models with sponsored songs for non-paying users, including the influence of these modules in the decision making process of the agent can prove conceptually tedious, whereas in our approach we consider that the agent is the module responsible for, given currently available information, computing a vector that will then query the ANN search system, while the rest of the recommender system is treated as part of the environment.

Additionally, to guarantee that user experience isn’t interrupted in case of technical shortcomings, such as the loss of internet connection on the client’s side, real-world RS are usually expected to send small batches of recommended songs, which may or may not be all displayed to the user, instead of recommending songs one by one. The case of several recommendations listened to in a row isn’t naturally handled by the discrete action set assumption, whereas it is compatible with our continuous action set definition, and can simply be modeled by trajectories where subsequent states may present an increment of more than one observation tuple.

In essence, our characterization of the environment is in line with the principles laid out in

(Sutton et al., 2018): "The general rule we follow is that anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment."

6.4 Future Work

In this chapter we modeled the task of interactive Track Mix as a Markov decision process with continuous action spaces and deep learning based policies. In our opinion, the future success of such models is conditioned by the following factors:

- **Training Procedure:** besides the concepts presented in Section 6.2, RL distinguishes itself from sequence modeling by its ability to consider observed reward when fitting the parameters of an agent’s policy. To this end, a considerable amount of algorithms have been proposed in recent years (Xu Wang et al., 2022) and we intend to study approaches compatible with our setting, i.e continuous action space and batch off-policy learning. Simple approaches like Advantage Weighted Regression (Peng et al., 2019) could be considered first, as well as some elements of the policy gradient family of algorithms (Xu Wang et al., 2022) such as soft actor-critic (Haarnoja et al., 2018). In this last case, an interesting line of work would be to challenge the commonly found Gaussian distribution used by the policy to model the probability of sampling a vector action, and replace it with a von Mises-Fisher distribution, thus tying the log-likelihood of an action being sampled from the distribution of the policy with the cosine similarity between the state representation given by the policy and the action selected, instead of the Euclidean distance between the two.
- **Embedding Construction:** the MDP definition we have proposed of the interactive Mix Generation task implies that the agent must navigate a multi-dimensional vector space in search of areas where the expected future cumulated reward will be high. The success of this navigation task should be facilitated when reward evolves smoothly in this space, which should depend on the ability of embedding methods to give close representations to songs holding similar values in the eyes of users.

To understand this, consider for instance methods for which training a policy relies on the use of a critic, i.e a neural network in charge of estimating the sum of future rewards that can be expected from selecting the vector action a when in state s . For actions coinciding with an actual song embedding e_i , a good critic should predict the value that this action holds for users when considering a given state, and interpolate between those anchor values. If these values drastically differ between songs closely embedded, this forces the critic to fit a complex reward estimation function with frequent oscillation, which can only be performed by critic networks with lots of parameters and fed with a high amount of data. For this reason, we expect embeddings distribution to highly contribute to the performances of such RL-based approaches. This could also benefit to the task of understanding properties of embedding spaces in the context of recommendation.

- **Reward Design:** in our approach the raw feedback observed from user must be translated into a scalar value. This transformation is done by heuristics aggregating heterogeneous

feedback. In the future, learning such transformations could help align training objectives of RL policies with the desired behaviour of a recommender system.

Conclusion and Future Work

In this thesis, we presented several contributions to industrial-scale music recommendation.

We introduced a general framework to build scalable automatic playlist continuation models meeting the expected requirements imposed by large-scale music streaming services. We provided a detailed overview of its capabilities and limitations, showing its versatility in incorporating a wide range of advanced representation learning and sequence modeling techniques. We demonstrated the empirical relevance of our framework through both offline experiments on the largest public dataset of playlists, and online experiments, sustainably improving automatic playlist continuation on the Deezer platform.

We then proposed a rigorous study of a practice commonly adopted for its convenience, which consists of averaging item embeddings in recommender systems. We provided a mathematical analysis of the consistency of these averaging operations in a general theoretical setting, as well as an empirical evaluation on real-world data. A subsequent comparison revealed that real-world averages are less consistent than those computed in our theoretical setting. Considering that averaging should continue to be widely used by recommender systems due to its ease of use and versatility, this result incentivizes the design of future embedding construction techniques that account for consistency during their training process to better handle sequences of embeddings. Finally, we introduced von Mises-Fisher exploration, a scalable approach to exploring large action sets in reinforcement learning problems where hyperspherical embedding vectors represent actions. We established a mathematical link between Boltzmann exploration and this new method, which should encourage its adoption by designers of large-scale recommender systems who wish to deviate in a precisely controlled manner from the predictions of production models. It has been successfully tested on Deezer and is now used to better explore songs to recommend to millions of users, facilitating future experiments on reinforcement learning-based music recommendation on the service.

For future work, we intend to explore the problem of continuously generating song recommendations for users within the paradigm of Markov decision processes, as detailed in the previous chapter. We expect that the main challenges to overcome will be including reward when training sequence models studied in this thesis as well finding representation for songs that accurately reflect the value they bring to a mix of tracks built for a user of a music streaming platform.

Bibliography

- Abramowitz, Milton and Irene A Stegun (1948). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Vol. 55. US Government Printing Office (cit. on pp. 76, 77, 84, 85, 90, 92, 96).
- Afchar, Darius, Romain Hennequin, and Vincent Guigue (2023). “Of Spiky SVDs and Music Recommendation”. *Proceedings of the 17th ACM Conference on Recommender Systems*, pp. 926–932 (cit. on pp. 20, 70, 78).
- Afsar, M. Mehdi, Trafford Crump, and Behrouz Far (2021). “Reinforcement learning based recommender systems: A survey” (cit. on p. 3).
- Akiba, Takuya, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama (2019). “Optuna: A Next-Generation Hyperparameter Optimization Framework”. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631 (cit. on p. 37).
- Amin, Susan, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup (2021). “A Survey of Exploration Methods in Reinforcement Learning”. *arXiv preprint arXiv:2109.00157* (cit. on pp. 71, 72).
- Andrews, Larry C (1998). *Special Functions of Mathematics for Engineers*. Vol. 49. Spie Press (cit. on p. 57).
- Arora, Sanjeev, Yingyu Liang, and Tengyu Ma (2017). “A Simple but Tough-To-Beat Baseline for Sentence Embeddings”. *Proceedings of the 5th International Conference on Learning Representations* (cit. on p. 48).
- Aumüller, Martin, Erik Bernhardsson, and Alexander Faithfull (2017). “ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms”. *International conference on similarity search and applications*. Springer, pp. 34–49 (cit. on p. 102).
- Banerjee, Arindam, Inderjit S Dhillon, Joydeep Ghosh, Suvrit Sra, and Greg Ridgeway (2005). “Clustering on the Unit Hypersphere using von Mises-Fisher Distributions.” *Journal of Machine Learning Research* 6.9 (cit. on p. 79).
- Banerjee, Sudipto and Anindya Roy (2014). *Linear Algebra and Matrix Analysis for Statistics*. CRC Press (cit. on pp. 19, 41, 79).
- Baricz, Árpád (2010). *Generalized Bessel Functions of the First Kind*. Springer (cit. on p. 74).
- Bendada, Walid, Théo Bontempelli, Mathieu Morlon, Benjamin Chapus, Thibault Cador, Thomas Bouabça, et al. (2023). “Track Mix Generation on Music Streaming Services using Transformers”. *Proceedings of the 17th ACM Conference on Recommender Systems*, pp. 112–115 (cit. on pp. 3, 26).
- Bendada, Walid, Guillaume Salha, and Théo Bontempelli (2020). “Carousel Personalization in Music Streaming Apps with Contextual Bandits”. *Proceedings of the 14th ACM Conference on Recommender Systems*, pp. 420–425 (cit. on pp. 1, 2, 11, 69).
- Bendada, Walid, Guillaume Salha-Galvan, Thomas Bouabça, and Tristan Cazenave (2023). “A Scalable Framework for Automatic Playlist Continuation on Music Streaming Services”. *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 464–474 (cit. on pp. 26, 60).
- Bendada, Walid, Guillaume Salha-Galvan, Romain Hennequin, Théo Bontempelli, Thomas Bouabça, and Tristan Cazenave (2024). “vMF-exp: von Mises-Fisher Exploration of Large Action Sets with Hyperspherical Embeddings”. *ICML 2024 Workshop: Aligning Reinforcement Learning Experimentalists and Theorists* (cit. on p. 68).
- Bendada, Walid, Guillaume Salha-Galvan, Romain Hennequin, Thomas Bouabça, and Tristan Cazenave (2023). “On the Consistency of Average Embeddings for Item Recommendation”. *Proceedings of the 17th ACM Conference on Recommender Systems*, pp. 833–839 (cit. on p. 47).
- Bennett, James, Stan Lanning, et al. (2007). “The netflix prize”. *Proceedings of KDD cup and workshop*. Vol. 2007. New York, p. 35 (cit. on pp. 2, 14).

- Billingsley, Patrick (2013). *Convergence of Probability Measures*. John Wiley & Sons (cit. on p. 92).
- Bobadilla, Jesús, Fernando Ortega, Antonio Hernando, and Jesús Bernal (2012). “A Collaborative Filtering Approach to Mitigate the New User Cold Start Problem”. *Knowledge-Based Systems* 26, pp. 225–238 (cit. on p. 3).
- Bobadilla, Jesús, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez (2013). “Recommender Systems Survey”. *Knowledge-Based Sys.* 46, pp. 109–132 (cit. on p. 1).
- Bokde, Dheeraj, Sheetal Girase, and Debajyoti Mukhopadhyay (2015). “Matrix Factorization Model in Collaborative Filtering Algorithms: A Survey”. *Procedia Computer Science* 49, pp. 136–146 (cit. on p. 41).
- Bonnin, Geoffray and Dietmar Jannach (2014). “Automated Generation of Music Playlists: Survey and Experiments”. *ACM Computing Surveys* 47.2, pp. 1–35 (cit. on p. 12).
- Bontempelli, Théo, Benjamin Chapus, François Rigaud, Mathieu Morlon, Marin Lorant, and Guillaume Salha-Galvan (2022). “Flow Moods: Recommending Music by Moods on Deezer”. *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 452–455 (cit. on pp. 3, 35, 70).
- Boytsov, Leonid and Bilegsaikhan Naidan (2013). “Engineering Efficient and Effective Non-metric Space Library”. *Similarity Search and Applications - 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings*. Ed. by Nieves R. Brisaboa, Oscar Pedreira, and Pavel Zezula. Vol. 8199. Lecture Notes in Computer Science. Springer, pp. 280–293 (cit. on p. 103).
- Briand, Léa, Guillaume Salha-Galvan, Walid Bendada, Mathieu Morlon, and Viet-Anh Tran (2021a). “A Semi-Personalized System for User Cold Start Recommendation on Music Streaming Apps”. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2601–2609 (cit. on pp. 6, 41, 44, 47, 48, 50, 60).
- (2021b). “A Semi-Personalized System for User Cold Start Recommendation on Music Streaming Apps”. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2601–2609 (cit. on pp. 11, 24).
- Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, et al. (2016). “Openai gym. arXiv”. *arXiv preprint arXiv:1606.01540* 10 (cit. on p. 107).
- Cesa-Bianchi, Nicolò, Claudio Gentile, Gábor Lugosi, and Gergely Neu (2017). “Boltzmann Exploration Done Right”. *Advances in Neural Information Processing Systems* 30 (cit. on p. 71).
- Chang, Jia-Wei, Ching-Yi Chiou, Jia-Yi Liao, Ying-Kai Hung, Chien-Che Huang, Kuan-Cheng Lin, et al. (2021). “Music recommender using deep embedding-based features and behavior-based reinforcement learning”. *Multimedia Tools and Applications*, pp. 1–28 (cit. on p. 3).
- Chapelle, Olivier and Lihong Li (2011). “An Empirical Evaluation of Thompson Sampling”. *Advances in Neural Information Processing Systems* 24 (cit. on pp. 78, 97).
- Chen, Ching-Wei, Paul Lamere, Markus Schedl, and Hamed Zamani (2018). “RecSys Challenge 2018: Automatic Music Playlist Continuation”. *Proceedings of the 12th ACM Conference on Recommender Systems*, pp. 527–528 (cit. on pp. 12, 23, 24, 27, 30, 35, 36, 39, 40).
- Chen, Jiawei, Junkang Wu, Jiancan Wu, Xuezhi Cao, Sheng Zhou, and Xiangnan He (2023). “Adap- τ : Adaptively Modulating Embedding Magnitude for Recommendation”. *Proceedings of the ACM Web Conference 2023*, pp. 1085–1096 (cit. on p. 70).
- Chen, Minmin, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi (2019). “Top-K Off-Policy Correction for a REINFORCE Recommender System”. *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*, pp. 456–464 (cit. on pp. 72, 109, 110).
- Chen, Minmin, Bo Chang, Can Xu, and Ed H Chi (2021). “User Response Models to Improve a REINFORCE Recommender System”. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pp. 121–129 (cit. on pp. 71, 72, 109, 110).
- Chen, Minmin, Can Xu, Vince Gatto, Devanshu Jain, Aviral Kumar, and Ed Chi (2022). “Off-Policy Actor-Critic for Recommender Systems”. *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 338–349 (cit. on pp. 72, 109, 110).
- Christakopoulou, Konstantina, Alex Beutel, Rui Li, Sagar Jain, and Ed H Chi (2018). “Q&R: A two-stage approach toward interactive recommendation”. *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 139–148 (cit. on p. 18).
- Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio (2014). “Empirical evaluation of gated recurrent neural networks on sequence modeling”. *arXiv preprint arXiv:1412.3555* (cit. on p. 22).

- Coleman, Ben (2020). *Why is it Okay to Average Embeddings?* Technical post on: <https://randorithms.com/2020/11/17/Adding-Embeddings.html> (cit. on p. 48).
- Coolidge, Julian L (1949). “The Story of the Binomial Theorem”. *The American Mathematical Monthly* 56.3, pp. 147–157 (cit. on p. 90).
- Cormen, Thomas H, Charles E Leiserson, Ronald L Rivest, and Clifford Stein (2022). *Introduction to Algorithms*. MIT Press (cit. on p. 71).
- Covington, Paul, Jay Adams, and Emre Sargin (2016a). “Deep Neural Networks for Youtube Recommendations”. *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 191–198 (cit. on pp. 1, 15, 31).
- (2016b). “Deep Neural Networks for Youtube Recommendations”. *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 191–198 (cit. on pp. 48, 50).
- Dang, Tran Khanh, Quang Phu Nguyen, and Van Sinh Nguyen (2020). “A Study of Deep Learning-Based Approaches for Session-Based Recommendation Systems”. *SN Computer Science* 1.4, pp. 1–13 (cit. on p. 21).
- Dauphin, Yann N, Angela Fan, Michael Auli, and David Grangier (2017). “Language Modeling with Gated Convolutional Networks”. *Proceedings of the 34th International Conference on Machine Learning*, pp. 933–941 (cit. on p. 31).
- Deezer (2024). <https://www.deezer.com> (cit. on p. 1).
- Donkers, Tim, Benedikt Loepp, and Jürgen Ziegler (2017). “Sequential User-Based Recurrent Neural Network Recommendations”. *Proceedings of the 11th ACM Conference on Recommender Systems*, pp. 152–160 (cit. on pp. 3, 21, 23).
- Douze, Matthijs, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, et al. (2024). “The faiss library”. *arXiv preprint arXiv:2401.08281* (cit. on p. 103).
- Du, Qiang, Vance Faber, and Max Gunzburger (1999). “Centroidal Voronoi Tessellations: Applications and Algorithms”. *SIAM Review* 41.4, pp. 637–676 (cit. on pp. 74, 88).
- Du, Qiang, Max Gunzburger, and Lili Ju (2010). “Advances in Studies and Applications of Centroidal Voronoi Tessellations”. *Numerical Mathematics: Theory, Methods and Applications* 3.2, pp. 119–142 (cit. on pp. 74, 88).
- Ek, Daniel (2024). *If you asked people what differentiates Spotify, one of the things they always cite is our playlists...* LinkedIn post on: https://www.linkedin.com/posts/daniel-ek-1b52093a_if-you-asked-people-what-differentiates-spotify-activity-7214371864054235137-cur4/ (cit. on p. 8).
- Epure, Elena V., Guillaume Salha, and Romain Hennequin (2020). “Multilingual Music Genre Embeddings for Effective Cross-Lingual Music Item Annotation”. *31st International Society for Music Information Retrieval Conference* (cit. on p. 1).
- Faggioli, Guglielmo, Mirko Polato, and Fabio Aiolli (2018). “Efficient Similarity Based Methods for the Playlist Continuation Task”. *Proceedings of the ACM Recommender Systems Challenge 2018*, pp. 1–6 (cit. on p. 22).
- Fano, Robert M (1949). *The transmission of information*. Vol. 65. Massachusetts Institute of Technology, Research Laboratory of Electronics ... (cit. on p. 19).
- Ferrari Dacrema, Maurizio, Paolo Cremonesi, and Dietmar Jannach (2019). “Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches”. *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 101–109 (cit. on p. 25).
- Ferraro, Andres, Dmitry Bogdanov, Jisang Yoon, KwangSeob Kim, and Xavier Serra (2018). “Automatic Playlist Continuation using a Hybrid Recommender System Combining Features from Text and Audio”. *Proceedings of the ACM Recommender Systems Challenge 2018*, pp. 1–5 (cit. on p. 22).
- Ferraro, Andres, Yuntae Kim, Soohyeon Lee, Biho Kim, Namjun Jo, Semi Lim, et al. (2021). “Melon Playlist Dataset: A Public Dataset for Audio-Based Playlist Generation and Music Tagging”. *Proceedings of the 2021 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 536–540 (cit. on p. 23).
- Fischer, Hans (2011). *A History of the Central Limit Theorem: from Classical to Modern Probability Theory*. Springer (cit. on pp. 57, 62, 81).
- Fisher, Ronald Aylmer (1953). “Dispersion on a Sphere”. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 217.1130, pp. 295–305 (cit. on pp. 69, 73, 81, 100).
- Garg, Diksha, Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff (2019). “Sequence and Time Aware Neighborhood for Session-Based Recommendations: STAN”. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1069–1072 (cit. on pp. 3, 38).

- Gentle, James E (2009). *Computational Statistics*. Vol. 308. Springer (cit. on pp. 83, 100).
- Gnedenko, Boris (1943). “Sur la Distribution Limite du Terme Maximum d’une Serie Aleatoire”. *Annals of Mathematics*, pp. 423–453 (cit. on pp. 91, 92).
- Goldberg, Yoav and Omer Levy (2014). “word2vec Explained: Deriving Mikolov et al.’s Negative-Sampling Word-Embedding Method”. *arXiv:1402.3722* (cit. on p. 33).
- Gomez-Uribe, Carlos A and Neil Hunt (2015). “The Netflix recommender system: Algorithms, business value, and innovation”. *ACM Transactions on Management Information Systems (TMIS)* 6.4, pp. 1–19 (cit. on p. 1).
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT (cit. on pp. 21, 32, 34, 37, 67).
- Gope, Jyotirmoy and Sanjay Kumar Jain (2017). “A Survey on Solving Cold Start Problem in Recommender Systems”. *Proceedings of the 2017 International Conference on Computing, Communication and Automation*, pp. 133–138 (cit. on p. 22).
- Graves, Alex and Alex Graves (2012). “Long short-term memory”. *Supervised sequence labelling with recurrent neural networks*, pp. 37–45 (cit. on p. 22).
- Grbovic, Mihajlo and Haibin Cheng (2018). “Real-Time Personalization using Embeddings for Search Ranking at Airbnb”. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 311–320 (cit. on p. 47).
- Guillou, Frédéric, Romaric Gaudel, and Philippe Preux (2016). “Sequential Collaborative Ranking Using (No-)Click Implicit Feedback”, pp. 288–296 (cit. on p. 3).
- Guo, Lei, Hongzhi Yin, Qinyong Wang, Tong Chen, Alexander Zhou, and Nguyen Quoc Viet Hung (2019). “Streaming Session-Based Recommendation”. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1569–1577 (cit. on pp. 21, 23).
- Guo, Ruiqi, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, et al. (2020). “Accelerating Large-Scale Inference with Anisotropic Vector Quantization”. *International Conference on Machine Learning* (cit. on p. 103).
- Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. *International conference on machine learning*. PMLR, pp. 1861–1870 (cit. on p. 111).
- Hansen, Casper, Christian Hansen, Lucas Maystre, Rishabh Mehrotra, Brian Brost, Federico Tomasi, et al. (2020). “Contextual and Sequential User Embeddings for Large-Scale Music Recommendation”. *Proceedings of the 14th ACM Conference on Recommender Systems*, pp. 53–62 (cit. on pp. 47, 50, 60).
- Hariri, Negar, Bamshad Mobasher, and Robin Burke (2015). “Adapting to User Preference Changes in Interactive Recommendation”. *Proceedings of the 24th International Joint Conference on Artificial Intelligence* (cit. on p. 37).
- He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua (2017a). “Neural Collaborative Filtering”. *Proceedings of the 26th International Conference on World Wide Web*, pp. 173–182 (cit. on pp. 3, 30).
- (2017b). “Neural Collaborative Filtering”. *WWW*, pp. 173–182 (cit. on pp. 15, 33, 50).
- He, Xiangnan, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua (2016). “Fast Matrix Factorization for Online Recommendation with Implicit Feedback”. *39th Int. ACM SIGIR conf. on Research and Development in Information Retrieval* (cit. on p. 15).
- Hidasi, Balázs, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk (2015). “Session-Based Recommendations with Recurrent Neural Networks”. *Proceedings of the 4th International Conference on Learning Representations* (cit. on p. 50).
- (2016). “Session-Based Recommendations with Recurrent Neural Networks”. *Proceedings of the 4th International Conference on Learning Representations* (cit. on pp. 26, 32).
- Hinton, Geoffrey E and Ruslan R Salakhutdinov (2006). “Reducing the dimensionality of data with neural networks”. *science* 313.5786, pp. 504–507 (cit. on p. 51).
- Hu, Binbin, Chuan Shi, and Jian Liu (2017). “Playlist recommendation based on reinforcement learning”. *Intelligence Science I: Second IFIP TC 12 International Conference, ICIS 2017, Shanghai, China, October 25-28, 2017, Proceedings 2*. Springer, pp. 172–182 (cit. on p. 3).
- Hu, Yifan, Yehuda Koren, and Chris Volinsky (2008). “Collaborative Filtering for Implicit Feedback Datasets”. *Proceedings of the 8th IEEE International Conference on Data Mining*, pp. 263–272 (cit. on p. 2).

- Huang, Zhenhua, Yajun Liu, Choujun Zhan, Chen Lin, Weiwei Cai, and Yunwen Chen (2021). “A novel group recommendation model with two-stage deep learning”. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.9, pp. 5853–5864 (cit. on p. 18).
- Hussein, Ahmed, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne (2017). “Imitation learning: A survey of learning methods”. *ACM Computing Surveys (CSUR)* 50.2, pp. 1–35 (cit. on p. 106).
- IFPI, International Federation of the Phonographic Industry (2023). *Engaging With Music 2023 Report*. https://www.ifpi.org/wp-content/uploads/2023/12/IFPI-Engaging-With-Music-2023_full-report.pdf (cit. on pp. x, 9).
- Indyk, Piotr and Rajeev Motwani (1998). “Approximate nearest neighbors: towards removing the curse of dimensionality”. *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613 (cit. on p. 17).
- Iwasaki, Masajiro and Daisuke Miyazaki (2018). “Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data”. *arXiv preprint arXiv:1810.07355* (cit. on p. 103).
- Jacobson, Kurt, Vidhya Murali, Edward Newett, Brian Whitman, and Romain Yon (2016). “Music Personalization at Spotify”. *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 373–373 (cit. on pp. 1, 11, 69).
- Jacod, Jean and Philip Protter (2004). *Probability Essentials*. Springer Science & Business Media (cit. on pp. 63, 64, 82, 94, 96).
- Jakobsen, Laurie (2016). *PLAYLISTS OVERTAKE ALBUMS IN LISTENERSHIP, SAYS LOOP STUDY*. Blog post on: <https://musicbiz.org/news/playlists-overtake-albums-listenership-says-loop-study/> (cit. on pp. 1, 8).
- Jannach, Dietmar, Gabriel de Souza P. Moreira, and Even Oldridge (2020). “Why Are Deep Learning Models Not Consistently Winning Recommender Systems Competitions Yet? A Position Paper”. *Proceedings of the Recommender Systems Challenge 2020*, pp. 44–49 (cit. on p. 24).
- Järvelin, Kalervo and Jaana Kekäläinen (2002). “Cumulated gain-based evaluation of IR techniques”. *ACM Transactions on Information Systems (TOIS)* 20.4, pp. 422–446 (cit. on p. 36).
- Javed, Umair, Kamran Shaukat, Ibrahim A Hameed, Farhat Iqbal, Talha Mahboob Alam, and Suhuai Luo (2021). “A Review of Content-Based and Context-Based Recommendation Systems”. *International Journal of Emerging Technologies in Learning* 16.3, pp. 274–306 (cit. on p. 2).
- Jegou, Herve, Matthijs Douze, and Cordelia Schmid (2010). “Product quantization for nearest neighbor search”. *IEEE transactions on pattern analysis and machine intelligence* 33.1, pp. 117–128 (cit. on p. 17).
- Jing, How and Alexander J Smola (2017). “Neural Survival Recommender”. *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, pp. 515–524 (cit. on p. 23).
- Johnson, Jeff, Matthijs Douze, and Hervé Jégou (2019). “Billion-Scale Similarity Search with GPUs”. *IEEE Transactions on Big Data* 7.3, pp. 535–547 (cit. on pp. 41, 69, 78, 79).
- Kang, Seungwoo and Hee-Seok Oh (2024). “Novel Sampling Method for the von Mises–Fisher Distribution”. *Statistics and Computing* 34.3, p. 106 (cit. on p. 74).
- Kang, Wang-Cheng and Julian McAuley (2018). “Self-Attentive Sequential Recommendation”. *Proceedings of the 2018 IEEE International Conference on Data Mining*, pp. 197–206 (cit. on pp. 22, 25, 37).
- Katz, Helen (2022). *The media handbook: A complete guide to advertising media selection, planning, research, and buying*. Routledge (cit. on p. 8).
- Kim, Dain, Jinhyeok Park, and Dongwoo Kim (2023). “Test-Time Embedding Normalization for Popularity Bias Mitigation”. *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 4023–4027 (cit. on p. 70).
- Kiran, B Ravi, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, et al. (2021). “Deep reinforcement learning for autonomous driving: A survey”. *IEEE Transactions on Intelligent Transportation Systems* 23.6, pp. 4909–4926 (cit. on p. 107).
- Konda, Vijay and John Tsitsiklis (1999). “Actor-Critic Algorithms”. *Advances in Neural Information Processing Systems* 12 (cit. on p. 80).
- Koren, Yehuda and Robert Bell (2015). “Advances in Collaborative Filtering”. *Recommender Systems Handbook*, pp. 77–118 (cit. on pp. 14–16, 41, 49, 79).
- Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). “Matrix Factorization Techniques for Recommender Systems”. *Computer* 42.8, pp. 30–37 (cit. on pp. 3, 4, 15, 22, 33).

- Koren, Yehuda, Steffen Rendle, and Robert Bell (2021). “Advances in collaborative filtering”. *Recommender systems handbook*, pp. 91–142 (cit. on p. 3).
- Kowald, Dominik, Markus Schedl, and Elisabeth Lex (2020). “The Unfairness of Popularity Bias in Music Recommendation: A Reproducibility Study”. *Proceedings of the 42nd European Conference on Information Retrieval*, pp. 35–42 (cit. on p. 23).
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep Learning”. *Nature* 521.7553, pp. 436–444 (cit. on p. 70).
- Li, Wen, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, et al. (2019). “Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement”. *IEEE Transactions on Knowledge and Data Engineering* 32.8, pp. 1475–1488 (cit. on pp. 17, 41, 69).
- Loève, Michel (1977). “Probability Theory I”. *Graduate Texts in Mathematics* 4 (cit. on p. 63).
- Ludewig, Malte and Dietmar Jannach (2018). “Evaluation of Session-Based Recommendation Algorithms”. *User Modeling and User-Adapted Interaction* 28.4, pp. 331–390 (cit. on pp. 36, 38).
- Ludewig, Malte, Iman Kamehkhosh, Nick Landia, and Dietmar Jannach (2018). “Effective Nearest-Neighbor Music Recommendations”. *Proceedings of the ACM Recommender Systems Challenge 2018*, pp. 1–6 (cit. on p. 22).
- Ludewig, Malte, Noemi Mauro, Sara Latifi, and Dietmar Jannach (2021). “Empirical Analysis of Session-Based Recommendation Algorithms”. *User Modeling and User-Adapted Interaction* 31.1, pp. 149–181 (cit. on pp. 25, 37–39).
- Malkov, Yury A. and Dmitry A. Yashunin (2016). “Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs”. *CoRR* abs/1603.09320. arXiv: 1603.09320 (cit. on p. 102).
- Mardia, Kanti V and Peter E Jupp (2009). *Directional Statistics*. John Wiley & Sons (cit. on pp. 82, 90).
- Marlin, Benjamin M and Richard S Zemel (2009). “Collaborative Prediction and Ranking with Non-Random Missing Data”. *Proceedings of the 3rd ACM Conference on Recommender Systems*, pp. 5–12 (cit. on p. 2).
- McFee, Brian and Gert Lanckriet (2012). “Hypergraph Models of Playlist Dialects”. *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pp. 343–348 (cit. on p. 23).
- Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). “Efficient Estimation of Word Representations in Vector Space”. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings* (cit. on pp. 33, 102).
- Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). “Distributed Representations of Words and Phrases and their Compositionality”. *Advances in Neural Information Processing Systems* (cit. on p. 33).
- Mongia, A., N. Jhamb, E. Chouzenoux, and A. Majumdar (2020). “Deep Latent Factor Model for Collaborative Filtering”. *Signal Processing* 169, p. 107366 (cit. on p. 15).
- Mu, Ruihui (2018). “A Survey of Recommender Systems based on Deep Learning”. *IEEE Access* 6, pp. 69009–69022 (cit. on p. 1).
- Mulligan, Mark (2023). *Music subscriber market shares 2023: New momentum*. Blog post on: <https://www.midiaresearch.com/blog/music-subscriber-market-shares-2023-new-momentum> (cit. on pp. 8, 10).
- Nagaraja, HN and HA David (2003). *Order Statistics*. John Wiley & Sons, Inc (cit. on pp. 58, 66).
- Oehlert, Gary W (1992). “A Note on the Delta Method”. *The American Statistician* 46.1, pp. 27–29 (cit. on p. 82).
- Okura, Shumpei, Yukihiro Tagami, Shingo Ono, and Akira Tajima (2017). “Embedding-Based News Recommendation for Millions of Users”. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1933–1942 (cit. on p. 47).
- Pal, Aditya, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec (2020). “PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest”. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2311–2320 (cit. on p. 47).
- Park, Chanyoung, Donghyun Kim, Xing Xie, and Hwanjo Yu (2018). “Collaborative Translational Metric Learning”. *Proceedings of the 2018 IEEE International Conference on Data Mining*. IEEE, pp. 367–376 (cit. on p. 3).
- Paterek, Arkadiusz (2007). “Improving regularized singular value decomposition for collaborative filtering”. *Proceedings of KDD cup and workshop*. Vol. 2007. 2007, pp. 5–8 (cit. on pp. 4, 15).
- Pazzani, Michael J and Daniel Billsus (2007). “Content-based recommendation systems”. *The adaptive web: methods and strategies of web personalization*. Springer, pp. 325–341 (cit. on p. 2).

- Peng, Xue Bin, Aviral Kumar, Grace Zhang, and Sergey Levine (2019). “Advantage-weighted regression: Simple and scalable off-policy reinforcement learning”. *arXiv preprint arXiv:1910.00177* (cit. on p. 111).
- Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). “Glove: Global Vectors for Word Representation”. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543 (cit. on p. 102).
- Petrov, Aleksandr and Craig Macdonald (2022). “Effective and efficient training for sequential recommendation using recency sampling”. *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 81–91 (cit. on p. 45).
- Petrov, Aleksandr Vladimirovich and Craig Macdonald (2023). “gSASRec: reducing overconfidence in sequential recommendation trained with negative sampling”. *Proceedings of the 17th ACM Conference on Recommender Systems*, pp. 116–128 (cit. on p. 45).
- Pinzón, Carlos and Kangsoo Jung (2023). “Fast Python Sampler for the von Mises Fisher Distribution”. *HAL Id: hal-04004568* (cit. on pp. 74, 77, 100).
- Radford, Alec, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. (2018). “Improving Language Understanding by Generative Pre-Training”. *OpenAI* (cit. on p. 32).
- Rendle, Steffen (2010). “Factorization Machines”. *Proceedings of the 2010 IEEE International Conference on Data Mining*, pp. 995–1000 (cit. on pp. 22, 26, 31).
- Rendle, Steffen, Walid Krichene, Li Zhang, and John Anderson (2020). “Neural Collaborative Filtering vs. Matrix Factorization Revisited”. *Proceedings of the Fourteenth ACM Conference on Recommender Systems*, pp. 240–248 (cit. on pp. 25, 29, 30).
- Ricci, Francesco, Lior Rokach, and Bracha Shapira (2011). “Introduction to Recommender Systems Handbook”. *Recommender Systems Handbook*, pp. 1–35 (cit. on p. 29).
- Rubtsov, Vasilii, Mikhail Kamenshchikov, Ilya Valyaev, Vasilii Leksin, and Dmitry I Ignatov (2018). “A Hybrid Two-Stage Recommender System for Automatic Playlist Continuation”. *Proceedings of the ACM Recommender Systems Challenge 2018*, pp. 1–4 (cit. on pp. 22, 38).
- Sakurai, Keigo, Ren Togo, Takahiro Ogawa, and Miki Haseyama (2022). “Deep reinforcement learning-based music recommendation with knowledge graph using acoustic features”. *ITE Transactions on Media Technology and Applications* 10.1, pp. 8–17 (cit. on p. 3).
- Salha-Galvan, Guillaume, Romain Hennequin, Benjamin Chapus, Viet-Anh Tran, and Michalis Vazirgiannis (2021). “Cold Start Similar Artists Ranking with Gravity-Inspired Graph Autoencoders”. *15th ACM Conference on Recommender Systems*, pp. 443–452 (cit. on p. 24).
- Salha-Galvan, Guillaume, Johannes F Lutzeyer, George Dasoulas, Romain Hennequin, and Michalis Vazirgiannis (2022). “Modularity-Aware Graph Autoencoders for Joint Community Detection and Link Prediction”. *Neural Networks* 153, pp. 474–495 (cit. on p. 78).
- Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl (2001). “Item-Based Collaborative Filtering Recommendation Algorithms”. *Proceedings of the 10th International Conference on World Wide Web*, pp. 285–295 (cit. on p. 11).
- Schedl, Markus, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi (2018). “Current Challenges and Visions in Music Recommender Systems Research”. *Int. Journal of Multimedia Information Retrieval* 7.2, pp. 95–116 (cit. on pp. 1, 3, 11–13, 22, 23, 27, 36, 44, 60, 69, 70).
- Schütze, Hinrich, Christopher D Manning, and Prabhakar Raghavan (2008). *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge (cit. on p. 36).
- Sguerra, Bruno, Viet-Anh Tran, and Romain Hennequin (2023). “Ex2vec: Characterizing users and items from the mere exposure effect”. *Proceedings of the 17th ACM Conference on Recommender Systems*, pp. 971–977 (cit. on p. 18).
- Simhadri, Harsha Vardhan, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Dobson Manohar, et al. (2024). “Results of the Big ANN: NeurIPS’23 competition”. *arXiv preprint arXiv:2409.17424* (cit. on p. 102).
- Smith, Brent and Greg Linden (2017). “Two Decades of Recommender Systems at Amazon.com”. *IEEE Internet Computing* 21.3, pp. 12–18 (cit. on p. 1).
- Song, Weiping, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, et al. (2019). “AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks”. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1161–1170 (cit. on p. 31).

- Sra, Suvrit (2012). “A Short Note on Parameter Approximation for von Mises-Fisher Distributions: And a Fast Implementation of $\text{Is}(x)$ ”. *Computational Statistics* 27, pp. 177–190 (cit. on p. 79).
- Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). “Dropout: a Simple Way to Prevent Neural Networks from Overfitting”. *The Journal of Machine Learning Research* 15.1, pp. 1929–1958 (cit. on p. 37).
- Su, Xiaoyuan and Taghi M Khoshgoftaar (2009). “A Survey of Collaborative Filtering Techniques”. *Advances in Artificial Intelligence* 2009 (cit. on pp. 14–16).
- Sun, Fei, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, et al. (2019). “BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer”. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1441–1450 (cit. on pp. 22, 25).
- Sun, Philip, David Simcha, Dave Dopson, Ruiqi Guo, and Sanjiv Kumar (2023). “SOAR: Improved Indexing for Approximate Nearest Neighbor Search”. *Neural Information Processing Systems* (cit. on p. 103).
- Sutton, Richard S and Andrew G Barto (2018). *Reinforcement Learning: An Introduction*. MIT Press (cit. on pp. 3, 70, 71, 80, 106, 107, 111).
- Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar (2016). *Introduction to Data Mining*. Pearson Education India (cit. on pp. 69, 80).
- Tintarev, Nava, Christoph Lofi, and Cynthia C.S. Liem (2017). “Sequences of diverse song recommendations an exploratory study in a commercial system”. *UMAP 2017 - Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pp. 391–392 (cit. on p. 3).
- Tomasi, Federico, Joseph Cauteruccio, Surya Kanoria, Kamil Ciosek, Matteo Rinaldi, and Zhenwen Dai (2023). “Automatic Music Playlist Generation via Simulation-based Reinforcement Learning”. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4948–4957 (cit. on p. 71).
- Tsai, Yao-Hung Hubert, Shaojie Bai, Paul Pu Liang, J. Zico Kolter, Louis-Philippe Morency, and Ruslan Salakhutdinov (2019). “Multimodal Transformer for Unaligned Multimodal Language Sequences”. *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pp. 6558–6569 (cit. on p. 3).
- Turrin, Roberto, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi (2015). “30Music Listening and Playlists Dataset.” *RecSys Posters*, p. 75 (cit. on p. 23).
- Van Den Oord, A., S. Dieleman, and B. Schrauwen (2013). “Deep Content-Based Music Recommendation”. *Adv. in Neural Information Processing Systems*. Vol. 26 (cit. on p. 2).
- Vasile, Flavian, Elena Smirnova, and Alexis Conneau (2016). “Meta-Prod2vec: Product Embeddings using Side-Information for Recommendation”. *Proceedings of the 10th ACM conference on recommender systems*, pp. 225–232 (cit. on pp. 21, 23, 33).
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, et al. (2017). “Attention Is All You Need”. *Advances in Neural Information Processing Systems* 30 (cit. on pp. 22, 25, 26, 32).
- Volkovs, Maksims, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner (2018). “Two-Stage Model for Automatic Playlist Continuation at Scale”. *Proceedings of the ACM Recommender Systems Challenge 2018*, pp. 1–6 (cit. on pp. 12, 22, 31, 37, 38).
- Wang, Dongjing, Shuiguang Deng, and Guandong Xu (2018). “Sequence-based context-aware music recommendation”. *Information Retrieval Journal* 21.2-3, pp. 230–252 (cit. on p. 3).
- Wang, Dongjing, Shuiguang Deng, Xin Zhang, and Guandong Xu (2016). “Learning Music Embedding with Metadata for Context Aware Recommendation”. *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pp. 249–253 (cit. on pp. 21, 23, 33).
- Wang, Hao, Naiyan Wang, and Dit-Yan Yeung (2015). “Collaborative Deep Learning for Recommender Systems”. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1235–1244 (cit. on pp. 3, 15).
- Wang, Jizhe, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee (2018). “Billion-Scale Commodity Embedding for E-Commerce Recommendation in Alibaba”. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 839–848 (cit. on pp. 1, 47, 50).
- Wang, Mengzhao, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang (2021). “A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search”. *arXiv preprint arXiv:2101.12631* (cit. on p. 11).

- Wang, Shoujin, Liang Hu, Yan Wang, Longbing Cao, Quan Z Sheng, and Mehmet Orgun (2019). “Sequential Recommender Systems: Challenges, Progress and Prospects”. *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (cit. on pp. 3, 21).
- Wang, Xinxi, Yi Wang, David Hsu, and Ye Wang (2014). “Exploration in interactive personalized music recommendation: a reinforcement learning approach”. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 11.1, pp. 1–22 (cit. on p. 3).
- Wang, Xu, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, et al. (2022). “Deep reinforcement learning: A survey”. *IEEE Transactions on Neural Networks and Learning Systems* 35.4, pp. 5064–5078 (cit. on p. 111).
- Wang, Yining, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu (2013). “A Theoretical Analysis of NDCG Ranking Measures”. *26th Annual Conference on Learning Theory*. Vol. 8, p. 6 (cit. on p. 36).
- Westfall, Peter H (2014). “Kurtosis as peakedness, 1905–2014. RIP”. *The American Statistician* 68.3, pp. 191–195 (cit. on p. 57).
- Wu, Chen and Ming Yan (2017). “Session-Aware Information Embedding for E-Commerce Product Recommendation”. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 2379–2382 (cit. on p. 50).
- Yang, Hojin, Yoonki Jeong, Minjin Choi, and Jongwuk Lee (2018). “Mmcf: Multimodal collaborative filtering for automatic playlist continuation”. *Proceedings of the ACM Recommender Systems Challenge 2018*, pp. 1–6 (cit. on pp. 22, 38).
- Yuan, Fajie, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He (2019). “A Simple Convolutional Generative Network for Next Item Recommendation”. *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*, pp. 582–590 (cit. on pp. 21, 23).
- Zamani, Hamed, Markus Schedl, Paul Lamere, and Ching-Wei Chen (2019). “An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation”. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.5, pp. 1–21 (cit. on pp. 12, 24, 35, 36).
- Zangerle, Eva, Martin Pichl, Wolfgang Gassler, and Günther Specht (2014). “#nowplaying Music Dataset: Extracting Listening Behavior from Twitter”. *Proceedings of the 1st Int. Workshop on Internet-Scale Multimedia Management*, pp. 21–26 (cit. on p. 23).
- Zhang, Shuai, Lina Yao, Aixin Sun, and Yi Tay (2019). “Deep Learning Based Recommender System: A Survey and New Perspectives”. *ACM Computing Surveys* 52.1, pp. 1–38 (cit. on pp. 1, 49).
- Zhu, Yiwen, Jinyi Liu, Wenya Wei, Qianyi Fu, Yujing Hu, Zhou Fang, et al. (2024). “vMFER: von Mises-Fisher Experience Resampling Based on Uncertainty of Gradient Directions for Policy Improvement of Actor-Critic Algorithms”. *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp. 2621–2623 (cit. on p. 73).

RÉSUMÉ

Les systèmes de recommandation aident les utilisateurs à découvrir du contenu pertinent dans des catalogues en ligne trop vastes pour être explorés manuellement. Par exemple, sur les plateformes de streaming musical, ils suggèrent des chansons, albums ou artistes susceptibles de plaire à l'utilisateur. Historiquement, ces systèmes prédisaient les goûts individuels des utilisateurs pour chaque chanson, sans prendre en compte l'aspect séquentiel de l'écoute de musique ou l'immensité des catalogues disponibles. Cette thèse propose des solutions pour rendre les systèmes de recommandation plus adaptés aux plateformes musicales modernes, en adaptant des méthodes de prise de décision séquentielle aux contraintes posées par la taille des catalogues de musique. Elle introduit une méthode pour prolonger automatiquement des playlists ainsi qu'un nouveau schéma d'exploration, adoptés par la plateforme Deezer, pour explorer efficacement de vastes catalogues musicaux en temps réel.

MOTS CLÉS

Systèmes de recommandation, mise à l'échelle, prise de décision séquentielle, modèles probabilistes, apprentissage automatique, plateformes de streaming musical

ABSTRACT

Recommender systems help users discover relevant content in online catalogs that are too vast to be explored manually. For example, on music streaming platforms, they suggest songs, albums, or artists likely to appeal to the user. Historically, these systems predicted individual user preferences for each song without considering the sequential nature of music listening or the enormity of available catalogs. This thesis proposes solutions to make recommender systems better suited to modern music platforms by adapting sequential decision-making methods to the challenges posed by the size of music catalogs. It introduces a scalable framework to automatically extend playlists, as well as a new exploration scheme, adopted by Deezer, to efficiently explore large music catalogs in real-time.

KEYWORDS

Recommendation systems, scaling, sequential decision making, probabilistic models, machine learning, music streaming platforms