National Technical University of Athens
School of Electrical and Computer Engineering
MSc Data Science & Machine Learning

Creating & Evaluating a Music Recommender System Without Access to Multiple User Data

Implementation and Testing on the Spotify Platform

Diploma Thesis

of

GEORGIOS PIPILIS

Supervisor: Stefanos Kollias
Professor

Athens, September 2022

National Technical University of Athens
School of Electrical and Computer Engineering
MSc Data Science & Machine Learning

Creating & Evaluating a Music Recommender System Without Access to Multiple User Data

Implementation and Testing on the Spotify Platform

Diploma Thesis

of

GEORGIOS PIPILIS

Supervisor: Stefanos Kollias
Professor

Approved by the examination committee on March 2023.

(Signature)	*(Signature)*	*(Signature)*
.....	.....	.....
Stefanos Kollias	Georgios Stamou	Athanasios Voulodimos
Professor	Professor	Associate Professor

Athens, September 2022

National Technical University of Athens
School of Electrical and Computer Engineering
MSc Data Science & Machine Learning

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

(Signature)

.....
Georgios Pipilis

March 2023

Abstract

A recommender system is a type of algorithm that provides personalized recommendations to users based on either their past behaviors or preferences or the properties of the content they consume. It is commonly used in e-commerce, streaming services, social media platforms, and other applications to enhance user experience and engagement. When it comes to music, recommender systems usually rely on vast amounts on user or track data in order to generate suggestions.

This diploma thesis aims to explore the creation and evaluation of a full recommender system pipeline that does not rely on data from multiple users or bleeding edge computing resources in order to function. This is done through the exploration of a listener's Spotify music history. The final algorithm, as well as the methods in which it is evaluated, will be compared to Spotify in order to evaluate how far one can reach without the need for extra resources.

Keywords

Recommender system, content based filtering, music recommendation, Spotify

To my friends Alyssa, Eva, Nikos & Sneha who supported me through thick and thin

Acknowledgements

This work would not have been possible without the help of my supervisors Paraskevi Tzouveli and Stefanos Kollias. I am grateful to them, as well as the entire faculty of ECE, NTUA and the DSML master which helped me form the necessary knowledge toolkit required to complete this thesis.

Athens, Summer 2022

Georgios Pipilis

Table of Contents

List of Tables

Chapter 1

Introduction

1.1 English

Over the last decade, the music industry has undergone a tremendous transformation. The widespread adoption of smartphones and high speed internet has led to the rapid digitisation of one of the biggest industries on the planet, with streaming services making up 65% of the global recorded music revenue in 2022 [1]. In comparison, only a decade earlier, back in 2012, the streaming service market share was merely 6.8% [2, 3]. One of the companies that spearheaded this shift in listening and market habits has been Spotify, a Swedish audio streaming provider that boasts 433 million active users as of the end of Q2, 2022 [4].

As a result of this digitisation, millions of unique tracks, albums and artists are now available, usually for free, to anyone who has access to an internet connection. The sheer volume of options available makes the task of discovering new music that fits the tastes of the user a challenging task. In contrast to problems such as recommending a new film or book, music recommendation systems deal with a larger variety of available options (the number of published songs is much higher than the number of published films and books) and a higher frequency of requests for recommendations. After all, when a user finds a suitable film or book, they can occupy themselves for multiple hours by only utilising that single recommendation whereas, a user searching for new music, might often look for more than one tracks.

The primary method used for tackling all of these problems, whether it's recommending music, books or any other digital or physical product, is a family of information filtering systems called recommender systems (RS). RS combine different data mining and filtering methodologies in order to help users select an item from possibly millions of different choices. Their applications range from suggesting digital content to users of streaming services (videos, songs, podcasts) [5, 6, 7] to recommending news articles and user groups to users of social media applications. There are three main approaches to recommender systems: collaborative filtering, content-based filtering and hybrid methods, which combine the aforementioned techniques in order to achieve better results.

Collaborative filtering (CF) [8] revolves around the idea of selecting items suitable for a user based on the habits of users with similar tastes. This is based on the assumption that users who largely have matching preferences about certain products will continue to

do so in the future for most new products. The similarity of tastes is usually quantified by a metric such as a product score or an up-vote/down-vote system. Generating a recommendation using CF requires data from multiple users but generates predictions specific to each user.

On the other hand, content-based filtering (CBF) systems [9] recommend new items based on the different features of each product and the history of the user's preferences when it comes to said features. As an example, in CBF film RS, features might include the genre of the film (Horror, Comedy, Drama), the cast of actors, the director etc. By utilising the user's rating and viewing history, the RS can determine whether a Horror film is a good recommendation, whether the user will enjoy a film with a specific actor starring and so on.

Naturally, hybrid systems refer to RS that utilise a combination of CF and CBF. Nowadays, hybrid algorithms are the go-to approach for most RS since they can offer the best of both worlds, without any significant drawbacks.

However, while selecting an appropriate RS approach might be straightforward in a multitude of cases, evaluating an RS is a much more difficult process. This arises from the fact that, when it comes to human preferences in music, books or any other product, there is no unique, absolute truth or single item that can be selected as the optimal recommendation for every user. Each human has their own, unique tastes and preferences. This has transformed the field from a purely computer science focused area of science to one that attracts research from mathematicians, psychologists and lawyers, among others.

Unfortunately, these facts make creating and evaluating an RS highly reliant on the existence and acquisition of data from multiple, sometimes million, users. This reliance on user data often leads to biased recommendations. In fact, research has shown [10, 11] that RS are often biased towards popular items, which leads to significant misrepresentation of unpopular items and usually limits the range of possible recommendations to a very small number. This is very evident in music streaming services [12], where users are often exposed to repeated recommendations of mainstream artists and suffer markedly worse quality recommendations if they are not interested in mainstream music.

When it comes to making recommendations, Spotify uses a hybrid system that relies both on the data of millions of users and content related features. While none of the user data is available to the general public, a significant amount of song and artist information is accessible through the platform's API. At the same time, any Spotify user can manually request a detailed copy of their own listening history from the past 365 days.

The goal of this thesis is to utilise the user's Spotify listening history and any content related features provided by the Spotify API in order to examine the plausibility and effectiveness of a music RS that is not based on the existence of large datasets that are comprised of millions of users. Furthermore, we aim to explore RS evaluation methods that do not rely on metrics such as song popularity or artist following, thus providing equal representation to artists, regardless of whether they are considered mainstream or not.

1.2 Ελληνικά

Κατά την τελευταία δεκαετία, η μουσική βιομηχανία έχει υποστεί μια τεράστια μεταμόρφωση. Η ευρεία υιοθέτηση των smartphones και του διαδικτύου υψηλών ταχυτήτων οδήγησε στην ταχεία ψηφιοποίηση μιας από τις μεγαλύτερες βιομηχανίες του πλανήτη, με τις υπηρεσίες streaming να αποτελούν το 65 % των παγκόσμιων εσόδων από την ηχογραφημένη μουσική το 2022. Συγκριτικά, μόλις μια δεκαετία νωρίτερα, το 2012, το μερίδιο αγοράς των υπηρεσιών streaming ήταν μόλις 6,8%. Μία από τις εταιρείες που πρωτοστάτησαν σε αυτή τη μετατόπιση των συνηθειών ακρόασης και αγοράς ήταν το Spotify, ένας σουηδικός πάροχος ροής ήχου που διαθέτει 433 εκατομμύρια ενεργούς χρήστες (με βάση τα στατιστικά έως και το τέλος του δεύτερου τριμήνου του 2022).

Ως αποτέλεσμα αυτής της ψηφιοποίησης, εκατομμύρια μοναδικά κομμάτια, άλμπουμ και καλλιτέχνες είναι πλέον διαθέσιμα, συνήθως δωρεάν, σε οποιονδήποτε έχει πρόσβαση σε σύνδεση στο διαδίκτυο. Ο τεράστιος όγκος των διαθέσιμων επιλογών καθιστά το έργο της ανακάλυψης νέας μουσικής που ταιριάζει στα γούστα του χρήστη δύσκολο. Σε αντίθεση με προβλήματα όπως η σύσταση μιας νέας ταινίας ή ενός βιβλίου, τα συστήματα μουσικών συστάσεων καλούνται να διαχειριστούν μεγαλύτερη ποικιλία διαθέσιμων επιλογών (ο αριθμός των δημοσιευμένων τραγουδιών είναι πολύ μεγαλύτερος από τον αριθμό των δημοσιευμένων ταινιών και βιβλίων) και μεγαλύτερη συχνότητα αιτημάτων για συστάσεις. Εξάλλου, όταν ένας χρήστης βρίσκει μια κατάλληλη ταινία ή ένα βιβλίο, μπορεί να απασχοληθεί για πολλές ώρες χρησιμοποιώντας μόνο αυτή τη μία σύσταση, ενώ ένας χρήστης που αναζητά νέα μουσική, μπορεί συχνά να αναζητήσει περισσότερα από ένα κομμάτια.

Η κύρια μέθοδος που χρησιμοποιείται για την αντιμετώπιση όλων αυτών των προβλημάτων, είτε πρόκειται για συστάσεις μουσικής, βιβλίων ή οποιουδήποτε άλλου ψηφιακού ή φυσικού προϊόντος, είναι μια οικογένεια συστημάτων φιλτραρίσματος πληροφοριών που ονομάζονται συστήματα συστάσεων (Recommender Systems - RS). Τα RS συνδυάζουν διαφορετικές μεθοδολογίες εξόρυξης δεδομένων και φιλτραρίσματος προκειμένου να βοηθήσουν τους χρήστες να επιλέξουν ένα αντικείμενο από ενδεχομένως εκατομμύρια διαφορετικές επιλογές. Οι εφαρμογές τους κυμαίνονται από την πρόταση ψηφιακού περιεχομένου στους χρήστες υπηρεσιών ροής (βίντεο, τραγούδια, podcasts) έως τη σύσταση άρθρων ειδήσεων και ομάδων χρηστών (user groups) στα μέλη εφαρμογών κοινωνικής δικτύωσης. Υπάρχουν τρεις κύριες προσεγγίσεις στα συστήματα συστάσεων: το συνεργατικό φιλτράρισμα, το φιλτράρισμα βάσει περιεχομένου και οι υβριδικές μέθοδοι, οι οποίες συνδυάζουν τις προαναφερθείσες τεχνικές προκειμένου να επιτύχουν καλύτερα αποτελέσματα.

Το συνεργατικό φιλτράρισμα (Collaborative Filtering - CF) περιστρέφεται γύρω από την ιδέα της επιλογής στοιχείων κατάλληλων για έναν χρήστη με βάση τις συνήθειες χρηστών με παρόμοιες προτιμήσεις. Αυτό βασίζεται στην υπόθεση ότι οι χρήστες που έχουν σε μεγάλο βαθμό ταυτόσημες προτιμήσεις για ορισμένα προϊόντα θα συνεχίσουν να το κάνουν και στο μέλλον για τα περισσότερα νέα προϊόντα. Η ομοιότητα των προτιμήσεων ποσοτικοποιείται συνήθως με μια μετρική, όπως μια βαθμολογία προϊόντος ή ένα σύστημα ψήφων προς τα πάνω/κάτω. Η παραγωγή μιας σύστασης με χρήση CF απαιτεί δεδομένα από πολλούς χρήστες, αλλά παράγει προβλέψεις ειδικά για κάθε χρήστη.

Από την άλλη πλευρά, τα συστήματα φιλτραρίσματος βάσει περιεχομένου (Content Based

Filtering - CBF) συνιστούν νέα προϊόντα με βάση τα διαφορετικά χαρακτηριστικά κάθε προϊόντος και το ιστορικό των προτιμήσεων του χρήστη όσον αφορά τα εν λόγω χαρακτηριστικά. Για παράδειγμα, στα CBF κινηματογραφικά RS, τα χαρακτηριστικά μπορεί να περιλαμβάνουν το είδος της ταινίας (τρόμος, κωμωδία, δράμα), το καστ των ηθοποιών, τον σκηνοθέτη κ.λπ. Αξιοποιώντας τη βαθμολογία και το ιστορικό προβολής του χρήστη, το RS μπορεί να καθορίσει αν μια ταινία τρόμου αποτελεί καλή σύσταση, αν ο χρήστης θα απολαύσει μια ταινία με πρωταγωνιστή έναν συγκεκριμένο ηθοποιό κ.ο.κ.

Φυσικά, τα υβριδικά συστήματα αναφέρονται σε RS που χρησιμοποιούν έναν συνδυασμό CF και CBF. Στις μέρες μας, οι υβριδικοί αλγόριθμοι είναι η προσέγγιση που επιλέγεται για τα περισσότερα RS, καθώς μπορούν να προσφέρουν το καλύτερο και από τους δύο κόσμους, χωρίς σημαντικά μειονεκτήματα.

Ωστόσο, ενώ η επιλογή μιας κατάλληλης προσέγγισης RS μπορεί να είναι απλή σε μια πληθώρα περιπτώσεων, η αξιολόγηση ενός RS είναι μια πολύ πιο δύσκολη διαδικασία. Αυτό προκύπτει από το γεγονός ότι, όταν πρόκειται για τις ανθρώπινες προτιμήσεις στη μουσική, τα βιβλία ή οποιοδήποτε άλλο προϊόν, δεν υπάρχει μοναδική, απόλυτη αλήθεια ή ένα μόνο στοιχείο που μπορεί να επιλεγεί ως η βέλτιστη σύσταση για κάθε χρήστη. Κάθε άνθρωπος έχει τα δικά του, μοναδικά γούστα και προτιμήσεις. Αυτό έχει μετατρέψει το πεδίο από έναν τομέα της επιστήμης που επικεντρώνεται καθαρά στην επιστήμη των υπολογιστών σε έναν τομέα που προσελκύει την έρευνα από μαθηματικούς, ψυχολόγους και νομικούς, μεταξύ άλλων.

Δυστυχώς, τα γεγονότα αυτά καθιστούν τη δημιουργία και την αξιολόγηση ενός RS σε μεγάλο βαθμό εξαρτώμενη από την ύπαρξη και την απόκτηση δεδομένων από πολλούς, μερικές φορές εκατομμύρια, χρήστες. Αυτή η εξάρτηση από τα δεδομένα των χρηστών οδηγεί συχνά σε μεροληπτικές συστάσεις. Στην πραγματικότητα, η έρευνα έχει δείξει ότι τα RS είναι συχνά προκατειλημμένα προς τα δημοφιλή στοιχεία, γεγονός που οδηγεί σε σημαντική παραποίηση των μη δημοφιλών στοιχείων και συνήθως περιορίζει το εύρος των πιθανών συστάσεων σε έναν πολύ μικρό αριθμό. Αυτό είναι πολύ εμφανές στις υπηρεσίες ροής μουσικής, όπου οι χρήστες συχνά εκτίθενται σε επαναλαμβανόμενες συστάσεις mainstream καλλιτεχνών και υφίστανται σημαντικά χειρότερες ποιοτικά συστάσεις εάν δεν ενδιαφέρονται για mainstream μουσική.

Όταν πρόκειται να κάνει συστάσεις, το Spotify χρησιμοποιεί ένα υβριδικό σύστημα που βασίζεται τόσο στα δεδομένα εκατομμυρίων χρηστών όσο και σε χαρακτηριστικά που σχετίζονται με το περιεχόμενο. Ενώ κανένα από τα δεδομένα των χρηστών δεν είναι διαθέσιμο στο ευρύ κοινό, ένας σημαντικός αριθμός πληροφοριών για τραγούδια και καλλιτέχνες είναι προσβάσιμος μέσω του API της πλατφόρμας. Ταυτόχρονα, κάθε χρήστης του Spotify μπορεί να ζητήσει χειροκίνητα ένα λεπτομερές αντίγραφο του δικού του ιστορικού ακρόασης από τις τελευταίες 365 ημέρες.

Στόχος της παρούσας διπλωματικής εργασίας είναι να αξιοποιήσει το ιστορικό ακρόασης του χρήστη στο Spotify και τυχόν χαρακτηριστικά που σχετίζονται με το περιεχόμενο και παρέχονται από το API του Spotify, προκειμένου να εξετάσει την απόδοση και την αποτελεσματικότητα ενός μουσικού RS που δεν βασίζεται στην ύπαρξη μεγάλων συνόλων δεδομένων που αποτελούνται από εκατομμύρια χρήστες. Επιπλέον, στόχος μας είναι να διερευνήσουμε μεθόδους αξιολόγησης RS που δεν βασίζονται σε μετρικές όπως η δημοτικότητα των τρα-

γουδιών ή η παρακολούθηση καλλιτεχνών, παρέχοντας έτσι ισότιμη εκπροσώπηση στους καλλιτέχνες, ανεξάρτητα από το αν θεωρούνται mainstream ή όχι.

Chapter 2

Background

Before we can start the in depth discussion of our algorithm, we need to define a few key concepts that play a pivotal role in understanding the essence of how music recommendation works.

2.1 Recommender Systems

Recommender systems (also known as recommendation systems) are algorithms that belong in the family of information filtering techniques. They aim to provide personalised recommendations to users by utilising information such as their past activity, sociodemographic information, as well as the preferences of other users. These systems have gained widespread adoption in various domains, including but not limited to, music and video streaming services (recommendation of new content the user might enjoy), e-commerce websites (recommendation of products the user might be interested in), and social media platforms (recommendation of groups and pages the user might consider following).

Recommender systems are usually classified into the following categories, based on the technique the algorithm uses to generate recommendations:

2.1.1 Collaborative Filtering

Memory-based Collaborative Filtering

Memory-based collaborative filtering is a method of recommendation that relies on the past behavior and preferences of users in order to make recommendations. Memory-based collaborative filtering approaches can be further divided into two categories: user-item filtering and item-item filtering.

User-item filtering is based on the idea that similar users tend to have similar preferences. Given a target user, the algorithm will look for users with preferences similar to those of the target user and will, therefore, recommend items that those similar users have liked. Mathematically, this can be represented as:

$$\text{Prediction(user, item)} = \frac{\sum_{u \in U} sim(u, u_0) \cdot r_{u,i}}{\sum_{u \in U} |sim(u, u_0)|}$$

Where:

U is the set of all users $sim(u, u_0)$ is the similarity between users u and u_0 $r_{u,i}$ is the rating given by user u to item i

Item-item filtering, on the other hand, is based on the idea that users have an inclination towards items that are similar to ones they already rate highly. Given a target item, the system will look for similar items and recommend them - for example, if someone generally likes action movies item-item based RS will be biased towards suggesting new action movies rather than picking a film from a different genre. Mathematically, this can be represented as:

$$\text{Prediction(user, item)} = \frac{\sum_{i \in I} sim(i, i_0) \cdot r_{u,i}}{\sum_{i \in I} |sim(i, i_0)|}$$

Where:

I is the set of all items $sim(i, i_0)$ is the similarity between items i and i_0 $r_{u,i}$ is the rating given by user u to item i

A few different metrics can be used to calculate the similarity between items or users. One of the most popular measures of similarity used is Cosine similarity. Cosine similarity is defined as the cosine of the angle between two feature vectors in an inner product space. Mathematically, it can be written as:

$$\text{cosine similarity} = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}||\vec{B}|} = \cos(\partial)$$

Where A and B are the two feature vectors and $|A|$ and $|B|$ are the magnitudes of the vectors.

The dot product of the vectors, $A \cdot B$, measures the similarity between the two vectors, while the magnitudes of the vectors, $|A|$ and $|B|$, measure the magnitude of the vectors. The cosine similarity is then calculated by dividing the dot product of the vectors by the product of their magnitudes.

The resulting value will be between -1 and 1, with 1 indicating that the vectors are perfectly aligned (i.e., they point in the same direction), 0 indicating that the vectors are orthogonal (i.e., they are perpendicular), and -1 indicating that the vectors are opposite (i.e., they point in opposite directions).

In a recommendation system, cosine similarity can be used to measure the similarity between two users (or two items) based on their ratings (or other features) in a vector space. For example, if two users have rated a similar set of movies, their feature vectors will be more similar, and the cosine similarity between them will be closer to 1. On the other hand, if the two users have rated very different sets of movies, their feature vectors will be less similar, and the cosine similarity between them will be closer to 0.

Model-based Collaborative Filtering

Model-based collaborative filtering is a method based on creating a model that is learned from past user behavior to make predictions about a user's preferences for items. The model is typically a matrix factorization algorithm, such as singular value decomposition (SVD), that is trained on the past user-item interactions (e.g. ratings) to learn latent

representations of users and items. The learned latent representations are then used to make predictions about a user's preferences for items they have not yet interacted with.

The key idea behind matrix factorization is that the observed ratings can be approximated by a dot product of two low-rank matrices, one representing the users and the other representing the items. These low-rank matrices can be thought of as encoding the underlying preferences of the users and the characteristics of the items. Once the model is trained, it can be used to make predictions for any user-item pair by taking the dot product of the corresponding user and item representations. These predictions can then be used to rank items for a particular user, with the highest-ranked items being recommended to the user.

Model-based collaborative filtering can handle large datasets and can be more accurate than memory-based collaborative filtering, but it requires a lot of data to train and may not be able to model the dynamics of user preferences over time.

Naturally, approaches that combine model-based filtering with memory-based approaches are common.

2.1.2 Content-Based Filtering

On the other hand, content-based filtering provides recommendations based on the user's past behavior and the features (the content) of the items they have interacted with. This technique is based on the idea that if a user has liked or interacted with certain items in the past, they are likely to enjoy similar items in the future.

The general process for content-based filtering starts by representing each item in the dataset as a set of features or attributes. For example, in the case of movies, the features could be genre, actors, directors, and plot-related keywords. Subsequently, a user profile based on the user's past interactions with similar items is created. In the movie example provided above, that profile would include the user's ratings of different films. This profile is typically modelled as a vector, which represents the user's preferences for each of the item features.

Finally, the algorithm calculates the similarity of each single item in the dataset with the user's profile vector, essentially searching for items that closely match what the user enjoys on average. The items that display the highest similarity are then served to the user as recommendations. This similarity can be measured by using a variety of metrics, from cosine similarity to the Pearson correlation coefficient.

2.1.3 Hybrid Approaches

Nowadays, one of the most common approaches to recommender systems are hybrid approaches [13]. These combine collaborative filtering and content-based recommendations in order to bypass the drawbacks of each method and offer better recommendations.

One way to combine the two methods is by implementing them separately, generating predictions using each of them separately and then, finally combining the results of the two. The combination can be done either linearly or by using techniques such as voting,

where multiple recommender systems vote on the best recommendations with the majority deciding the item that will ultimately be recommended.

A different way to combine the two approaches is by creating user vectors (similarly to what is done in content-based filtering) for every user and then utilise these to provide recommendations based on user similarity, akin to what is done in collaborative filtering. This approach deals with certain sparsity issues that might arise in datasets where not many users have multiple items in common.

Alternatively, users can be grouped together based on collaborative filtering techniques. This is reminiscent of clustering and allows us to generate group profiles for each cluster instead of user profiles, which we can subsequently use for content-based filtering.

2.2 Recommending New Music

Music recommendation engines generate personalized music recommendations for users based on their past listening history and track information. These systems have been an important part of the music industry since the early '00s, when platforms such as Pandora and Last.fm introduced the first collaborative filtering-based recommendation engines.

Since then, music recommendation systems have undergone significant evolution, with the incorporation of more advanced techniques such as machine learning and the use of diverse data sources. For example, in 2015, the music streaming service Spotify introduced the Discover Weekly feature, which utilized collaborative filtering and natural language processing to recommend songs to users based on their listening history and the lyrics of the songs that they had listened to. Since then, mood detection based on lyrics and audio signals has been a cutting edge area of recommender system research [14, 15]

More recently, music recommendation systems have begun to incorporate deep learning techniques in order to analyze the audio features of songs and make recommendations based on the acoustic characteristics of the music.

2.3 The Spotify Recommendation Ecosystem

Naturally, the exact nature of the Spotify algorithm is not information that is available to the general public. This makes sense as Spotify is a for profit company, whose product largely relies on recommender systems for user retention and acquisition. And in today's landscape where competitor music streaming applications and services abound, leading the pack with cutting edge algorithms while, at the same time, protecting those algorithms is important. However, from various papers and presentations released over the years, we can get a brief glimpse under the hood of Spotify's recommendation engine.

As one might expect, collaborative filtering is part of the Spotify algorithm ecosystem [16]. With more than 433 million users, there is enough data granularity to allow almost perfect matches between clusters of users who listen to similar music. Similarly, Spotify

takes advantage of content based filtering [16], taking into account song lyrics (using Natural Language Processing for Sentiment Analysis), track features (a lot of which are publicly available through the service's API and will be used in the scope of this thesis) and raw track signal features among other things.

On top of that, according to various papers released by the Spotify Research & Development team, the company has experimented with pitch tracking and melody estimation [17], music recommendation using multi-armed bandit algorithms [18] and graph representation learning [19].

2.4 Bias in Music Recommendation Systems

Irrespective of the technology used, music recommendation systems have been found to be biased in various ways, thus perpetuating existing societal inequalities and limiting the diversity of music that is recommended to users.

One source of bias in music recommendation systems is the data used to train the system. For example, studies have found that music recommendation systems trained on data from predominantly white and/or male users tend to recommend music by white male artists more often [20, 21]. Similarly, music recommendation systems trained on data from Western countries tend to recommend Western music to users, while underrepresenting music from other cultural regions. These biases in the training data can result in a lack of diversity in the music that is recommended to users, subsequently making it more difficult for musicians from underrepresented groups to gain visibility.

Multiple research groups have found evidence of popularity bias in music recommendation systems [22, 10, 11, 12, 20, 21], indicating that recommendation systems are inclined towards recommending more popular artists, with users interested in unpopular items receiving worse recommendations overall [23].

When it comes to Spotify, the company's Research & Development team has started producing more and more research on the topic of recommender system & AI fairness with a focus on providing good recommendations without causing a "rich-get-richer" effect [24]. An example of this is the paper published by Aziz et al., which proposes utilising semantic information via means of knowledge graphs in order to recommend underserved podcasts to users who might be interested [25]. Unfortunately, the algorithmic implementations (and even a lot of the papers) are not publicly available.

Chapter 3

Methods

3.1 Model Overview

One of the purposes of this thesis is to explore the viability of music recommender systems (RS) which do not take advantage of data from millions of users. This is a limitation that naturally arises from the fact that user data from streaming services such as Spotify are not publicly available. As a consequence, the use of collaborative filtering (CF) for music recommendation is out of the picture.

Instead, we are going to explore possibility of using a single user's listening history in order to discover patterns about their listening habits. The listening history of any Spotify user can easily be obtained by the user themselves through the service's official website and includes a detailed timeline of any piece of music or podcast they streamed through Spotify in the past 365 days. From this file, we aim to create a "user profile" - that is to say a vector of features that best describes the music preferences of the person seeking recommendations.

The process of generating this vector starts by figuring out the user's top artists based on frequency and total time listened. Subsequently, these artists are separated into clusters that represent different styles of music. This clustering process is based on song and artist features that are openly available to anyone through the use of the Spotify API. These features provide a range of information for each track, from details of the musical composition (key, mode, tempo) to more abstract metrics calculated by Spotify such as the danceability of a song or its happiness levels. We average selected features for each artist while dealing with outliers and missing data and, afterwards, we generate a user vector by calculating the weighted average of the user's X top artists by listening time, weighed by total listening time. The amount of top artists X we use to create this vector is one of the hyperparameters of the model and, therefore, is not constant.

Now that he have a representation of the user's listening preferences, we aim to find artists that closely match this profile. In order to do this, we calculate artist vectors for every artist available on Spotify by averaging the features of their songs, after dealing with outliers and missing or problematic values. This allows us to use a variety of similarity and proximity measures in order to single out the artists that appear to fit the user's tastes. Naturally, before performing this operation, we remove the user's most listened to artists from the dataset of all Spotify artists in order to ensure there are no recommendations of

already favored artists as well as guarantee there is no contamination when evaluating the results of our RS.

In order for the process described above to work, we need to acquire two main datasets. First and foremost, we need a dataset of features for all possible artists on Spotify. This dataset will be used as a pool from which we can pick new recommendations but also as a source of information for any possible artist, whether it is an artist we are recommending or one that exists in the user's listening history.

Secondly, our method requires the existence of a data table that tracks information about the user's top artists. We single out these artists based on the person's listening history records and then we retrieve all features relevant to them from the aforementioned dataset of all Spotify songs.

3.2 The 8 Million Song Dataset

3.2.1 Data Acquisition

Spotify provides an openly accessible application programming interface (API) [26], through which anyone can query the service's databases in order to acquire various data. This includes data about any artist and track, as well as the audio features of said tracks. Unfortunately, the amount of tracks available on the platform is prohibitively large and would require querying the API hundreds of millions of times in order to scrape all of the information we need. Naturally, Spotify throttles large amounts of successive queries, thus making a scraping task like this prohibitively time and resource consuming.

Hence, instead of obtaining information on the entire platform catalogue by using API calls, we opted for the use of a public dataset found on Kaggle [27]. This dataset comes in the form of an SQLite database that provides information on 8 million of the most popular Spotify songs, including their audio features as well as artist, album and genre information. While this number is only a fraction of the total number of tracks available on Spotify (which is around 80 million), it is large enough for us to make recommendations since, as we'll observe later on, only a small fractions of songs on Spotify have any listeners. Of course, the fact a song has no plays doesn't make it unsuitable for recommendations but it is certainly not something an RS would recommend to a user.

It is worth noting that this dataset includes tracks that were released up to, and during, 2018.

3.2.2 Pre-Processing

Due to the size of the dataset (over 5.1 GB), it was provided to us in the form of an SQLite database. This database comprised of 9 different tables:

- Artists: Provides information about the popularity (a metric arbitrarily defined by Spotify) and following count of each artist, as well as their unique ID.
- Albums: Provides a unique ID for each album on Spotify, as well as some basic

information (album name, album type, release data, popularity). This information does not include the artist's name.

- Track: Provides a unique ID and basic information for every track on Spotify. This does not include the name of the corresponding artist or album.
- Audio Features: Provides a multitude of audio features for every unique track ID that exists on the "Track" table.
- Genres: Gives a unique ID to each genre descriptor that exists on Spotify.
- Albums ↔ Artists: Correlates each album ID with one or multiple artist IDs.
- Albums ↔ Tracks: Correlates each album ID with the track IDs of all of the songs in that album.
- Artist ↔ Genre: Correlates each artist ID with one or multiple genre IDs.
- Track ↔ Artist: Correlates each track ID with one or multiple artist IDs.

Evidently, the data is provided to us in a relational database format. In order to improve the efficiency and explainability of our model, we need to transform these tables into a format that is easier to manipulate and use for calculations. The format we selected for this implementation is that of Pandas DataFrames (DF).

The first step of this process is to export all of the SQLite database tables into separate comma-separated values (CSV) files, a file format that works brilliantly with Python and Pandas, the two main tools we are going to use for data processing. These CSV files are then loaded into Pandas DFs. For now, each of the DFs closely follow the structure of the corresponding SQL table. Our next step is to slowly start merging these DFs, while removing any duplicate data columns that might exist. The merging of the DFs happens on ID columns by taking into advantage the correlation tables outlined above. This process leads to a single DF with 11.8 million rows (the songs in our dataset) 16 columns: artist name, song name and duration, artist popularity and 12 music related features. It is apparent that the number of songs on this DF (11.8 million) does not match the "advertised" number of songs on the dataset (around 8 million). This is because any song that is a collaboration between multiple artists exists multiple times in the DF - one for each artist participating in the track. We are going to deal with this phenomenon in a later stage of the pre-processing process.

We also create a hash table (a Python dictionary in this case) that keeps track of the following count of each artist.

3.2.3 Audio Features

Since we are implementing a form of content-based filtering, we will need a set of features that describe the songs which are available for recommendation. In this case, this set of features mostly comes from the 12 music related features mentioned above. These are provided by Spotify for every track that is available on the service and play a pivotal role in describing any piece of music. The 12 audio features are:

- Acousticness: A confidence measure of whether the song is acoustic or not. Values range from 0.0 to 1.0, with higher values indicating higher confidence.

- Danceability: Describes how suitable a song is for dancing. Values range from 0.0 to 1.0, with higher values indicating a more "danceable" song. While Spotify doesn't provide an exact formula for this value, they mention that it is derived from a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity of the track.

- Energy: A measure of intensity and activity. Values range from 0.0 to 1.0, with higher values indicating more energetic tracks. According to Spotify, energetic tracks feel fast, loud, and noisy. An example of a track that scores high in energy would be a death metal track. On the contrary, a Bach track would score low on the scale. This is a purely perceptual measurement. Features contributing to it include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

- Instrumentalness: A confidence measure of whether the song is purely instrumental or not. Purely instrumental songs contain no vocals. Values range from 0.0 to 1.0, with values above 0.5 indicating an instrumental song. On the low end of the spectrum we would expect to find rap or spoken word tracks. It's worth mentioning that brief vocal sounds such as "ooh" and "aah" are treated as instrumental in this context.

- Liveness: A confidence measure of whether there is an audience present or not in the recording. Values again range from 0.0 to 1.0, with values above 0.8 indicating a high probability that the song is performed live.

- Loudness: The overall loudness of a track in decibels. According to the Spotify documentation, values are averaged across the entire track and are used when comparing the relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.

- Speechiness: This measure represents whether spoken words were detected in a song or not. Values range between 0.0 and 1.0, with higher values indicating recordings that are primarily speech-like, such as talk shows or poetry. According to Spotify, values above 0.66 strongly suggest a recording that is made almost entirely out of spoken words, while values between 0.33 and 0.66 point towards recordings that contain both speech and music and values below 0.33 suggest the absence of spoken words.

- Valence: Measures the musical positiveness of a song. While Spotify doesn't provide a detailed explanation about how it is calculated, we know that songs with higher valence sound more happy, cheerful and euphoric while tracks with low valence give off feelings of sadness, depression and anger. Values range from 0.0 to 1.0, with higher values representing more positive feelings.

Artist - Song	God is an Astronaut - Fragile
Acousticness	0.01
Danceability	0.30
Energy	0.48
Instrum.	0.82
Liveness	0.06
Loudness	0.73
Speechiness	0.03
Valence	0.20

Table 3.1. *Analysis of a Song from the Band "God is an Astronaut"*

- Key: The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C-sharp, 2 = D, and so on. If no key was detected, the value is -1.
- Mode: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
- Tempo: The estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the pace of a given piece and derives directly from the average beat duration.
- Time Signature: An estimated time signature. This is a notational convention to specify how many beats are in each bar. The time signature ranges from 3 to 7 indicating time signatures of "3/4", to "7/4".

Most of these features produce values that are already normalised. We normalise tempo and loudness values so they can produce results on the [0, 1] range, thus eliminating the huge differences in value scales that existed in our dataset. On one hand, this means that tempo and loudness values can no longer be interpreted at face value - while a loudness of -50 is thoroughly explainable and measurable in db, a loudness of 0.23 is something that is not explainable in and of itself and can only be used in comparisons. On the other hand, this change in scales will help us produce recommendations that are unbiased by loudness or tempo values.

Based on some simple data retrieval, we can see how these features can accurately describe a song or artist. Looking at table 3.1, we can see a track of the band "God is an Astronaut" which plays instrumental, rock music. An instrumentals of 0.82 confirms this is a track that does not include vocals, while a loudness of 0.73 and an energy value of 0.48 point towards rock music, with an acousticness of 0.01 confirming the track is performed with electrical instruments. Similarly, we can use this information to identify the traits of different performances of the same song. In table 3.2, we can see three different iterations of "House of the Rising Sun", with version 1 being the original version of the song and versions 2 and 3 being different, unidentified versions. Based on the

	Acoustic.	Dance.	Energy	Instrum.	Liveness	Loud.	Speech.	Valence
Version 1	0	0.31	0.48	0	0.09	0.76	0.03	0.29
Version 2	0.40	0.52	0.53	0.06	0.11	0.79	0.03	0.21
Version 3	0.09	0.22	0.67	0	0.71	0.75	0.07	0.24

Table 3.2. *Different Versions of "House of the Rising Sun"*

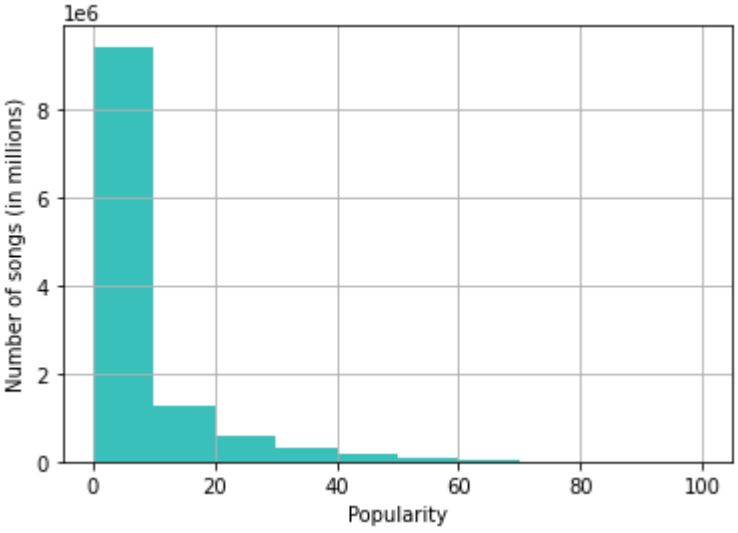
data at hand, we can tell version 2 is probably an acoustic performance of the song while version 3 is recorded live.

3.2.4 Artist Popularity & Following

Apart from the audio specific features mentioned above, we also take into account artist related features such as popularity and following. While a lot of the audio specific features are thoroughly explained by Spotify, popularity is analysed in a more obscure way. In Spotify's developer portal, with find this explanation:

"The popularity of a track is a value between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are. Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently. Artist and album popularity is derived mathematically from track popularity."

Even though this gives us a small glimpse of how the popularity metric works, we do not know how to interpret it or how it is distributed. For this reasons, we calculate a few basic metrics about it. The results are shown in table 3.3 and in the graph below. It is evident that we are dealing with a heavily skewed distribution where more than 50% of the artists in our dataset (that is to say, more than 4 million unique artists) score less than 1 out 100 in popularity and only a handful of artists score above 50/100.



The other metric we are going to explore is following. This is a more straightforward metric as it indicates the exact number of Spotify accounts that are following a particular

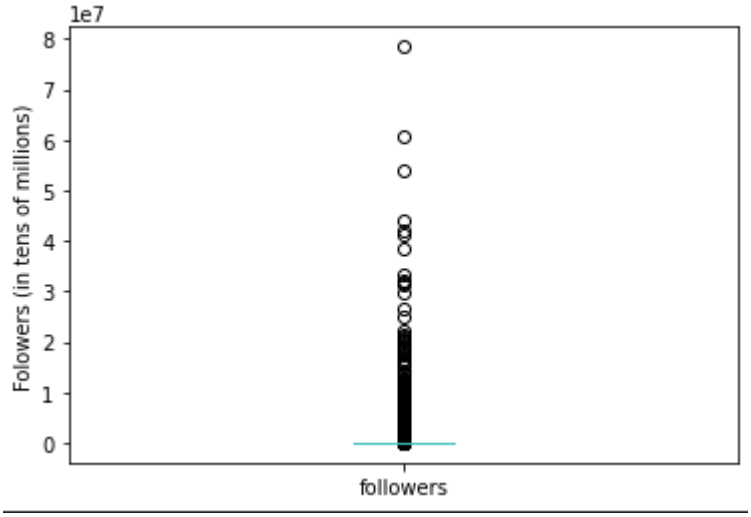
Mean (std)	25% (Q1)	50% (Median)	75% (Q3)
5.8 (10.4)	0	1	7

Table 3.3. *Popularity Score Distribution*

Mean (std)	25% (Q1)	50% (Median)	75% (Q3)	Max
7012.3 (208k)	0	0	101	78 Billion

Table 3.4. *Artist Following Distribution*

artist. When calculating a few basic statistics about artist following, we arrive at the results shown in table 3.4 and in the graph below. We can observe that more that 50% of all artists have 0 followers and that we need to get to the top 25% artists in order to start seeing follower counts that go into triple digits (>101).



We are going to use both of these metrics in order to test different variations of our algorithm. As mentioned before, research has shown that RS recommendations are often skewed towards more popular artists and artists with higher following thus, being able to filter recommended artists based on these two metrics, will allow us to emulate different scenarios where the popularity/following of an artist plays - or does not - a role in determining the RS behavior.

3.2.5 Artist Vectors

In order to generate artist recommendations using a content-based RS, we are going to need a set of features that describe each artist. We generate this vector by averaging the audio features mentioned above, without key, mode and time signature, and appending artist following and popularity to it. While the discography of any artist is not always uniform in style or even genre, creating artist vectors instead of using track or album-specific ones better fits the experimental scope of the thesis, available resources and using someone's top artists in order to recommend new artists.

This is also done in order to combat the limitation of this dataset, which is that it includes only songs released up to, and including, 2018. We do not want to recommend

Index	End Time	Artist Name	Track Name	ms Played
1	2020-11-16 22:11	The Album Leaf	Another Day (Revised)	47831
2	2020-11-17 17:15	King Diamond	The Family Ghost	265724
3	2020-11-17 17:56	King Diamond	7th Day of July 1777	56995
4	2020-11-17 20:13	Mac Miller	Blue World	203831
5	2020-11-17 20:26	Mac Miller	Good News	342040

Table 3.5. *Example data from a Spotify listening history file.*

only older songs and there is no feasible path for updating our datasets, therefore we opt to provide artist recommendations instead of track recommendations in order to partially bypass this problem.

3.3 User Listening History

Acquiring a copy of one's listening history is a straightforward process that can be started for free through Spotify's automated "Download Your Data" tool [28]. This history includes every piece of music or podcast streamed by the user during the last year and is delivered in a .json format. By extracting data from that file, we arrive at something that looks like data table 3.5.

Each entry on this table is a unique event (a stream of a song or podcast) on the streaming timeline of the past one year and is described by 4 different parameters: the time when the playback stopped, the name of the artist, the name of the track and the amount of milliseconds the playback lasted. From these elementary features, we can derive a plethora of different statistics about the person seeking recommendations and their streaming habits. Before we do that however, we need to clean up the data.

3.3.1 Pre-Processing

The main limitation of our datasets lies in the fact that the 8 million song database includes songs that were released only up to, and including, 2018 and can not, in the scope and time frame of this thesis, be updated to include newer tracks. However, when it comes to user's streaming histories, this limitation does not apply. This gives rise to a significant inconsistency: some of the tracks that exist in a user's streaming history might not exist in our 8 million song dataset. Thankfully, even though we might be unable to update our entire dataset to include songs released after 2018, we can still retrieve all necessary information on the small sample of songs, released after 2018, that is relevant to the user and their listening history.

The first step towards dealing with this issue is singling out the unique songs which exist in our dataset. We do this since our data describe someone's music streaming history, it is highly probable that a song might appear multiple times - that would mean this specific song was played multiple times in the span of one year, which is an entirely reasonable assumption. After we have the list of all unique songs that exist in the user's history, we check to see how many of these exist in our 8 million song dataset. Any of the

songs that do not exist there are kept track of in a separate list, so they can be added to the dataset during the next step.

In order to do this, we need to use Spotify's API. After registering a developer account with the streaming service and setting up the authentication process and credentials flow, we query Spotify's database for information about every song missing. The information we retrieve is comprised of exactly the same audio and track/artist-specific features that are described in the 8 million song dataset section. Throughout this process, we are especially careful to handle any possible code exceptions with emphasis on the scenario where the desired song data does not exist on Spotify's current database at all. This might happen for one of two reasons:

1. The streaming item is a podcast and not a track. Podcasts might exist on a user's history file but neither we nor Spotify have any audio features about them. Hence, querying the API for more information about their audio features, returns an error. Of course, podcasts are outside the scope of this thesis, so at this stage we can safely delete the podcast entry from the user's history and proceed with the next entry. In other words, this is a way for us to filter out any possible non-song items, since only queries to the Spotify API about songs will yield any audio feature information whatsoever.

2. The streaming item is a song but does no longer exist within the Spotify database. That means that the track and artist existed in Spotify's database back in 2018 but has since been removed. In this case, we can safely delete this entry and move on to the next item. This deletion is performed in order to avoid any inconsistencies between the two datasets (current Spotify and 2018 Spotify) when it comes to artist existence.

After we retrieve all of the necessary information, we add these new songs to the 8 million song dataset.

Now that we have all track information for the user's listening history, we proceed to calculate the user's top artists. We do this by ordering every artist in the user's history by total listening time. We'll use only a certain amount of these artists in order to provide recommendations. The reasoning behind this is straightforward: throughout the course of the year, a person can be exposed to thousands of different artists. This is especially true on Spotify, where the platform provides you with tailor-made playlists and features like Spotify Radio, all aimed at discovering new songs, on a daily basis. Using all of these artists would unnecessarily increase our algorithm's processing time and would introduce a lot of noise to the model, even if we weighted the influence of each artist based on the percentage of total listening time they were listened to.

After singling out the subset of top artists, we subtract that from the dataset of all available artists. This is done in order to avoid suggesting artists the user already listens to on a regular basis as well as any possible data contamination.

Naturally, there might exist outliers in the top artists. This is not referring simply to a group of artists that share a genre different to the average genre of the listener but to

single artists that drastically differ from the rest and might significantly skew the results if taken into account during the recommendation process. In order to deal with them, we remove outliers from the top artists based on their z-score. A z-score is a measure of how many standard deviations an observation or data point is from the mean of a distribution. Mathematically, it is defined as:

$z = \frac{X-\mu}{\sigma}$

Where X is the value of the observation or data point, μ is the mean of the distribution, and σ is the standard deviation of the distribution. By removing any top artist that exhibits an absolute z-score above 3, we eliminate most of the outliers in this subset of our data. This happens because, assuming our data follows a normal distribution, 99.7% of data points will fall within 3 standard deviations of the mean. Therefore, any data point with an absolute z-score above 3, will most likely be an outlier.

An alternative option for outlier detection that is available to users, is removing outliers based on the interquartile range (also known as IQR - a measure of the spread of a dataset, defined as the difference between the 75th percentile and the 25th percentile) which yields similar results to the z-score method.

3.3.2 Personalised Statistics

Since we have full access to the user's listening history, we can generate a set of personalised user statistics. These include: total listening time per artist, total listening time per unique track, total and average listening time per hour of the day, total and average listening time per day of the week, total and average listening time per month of the year as well as combinations of the above (e.g. most listened to artist for the month of September).

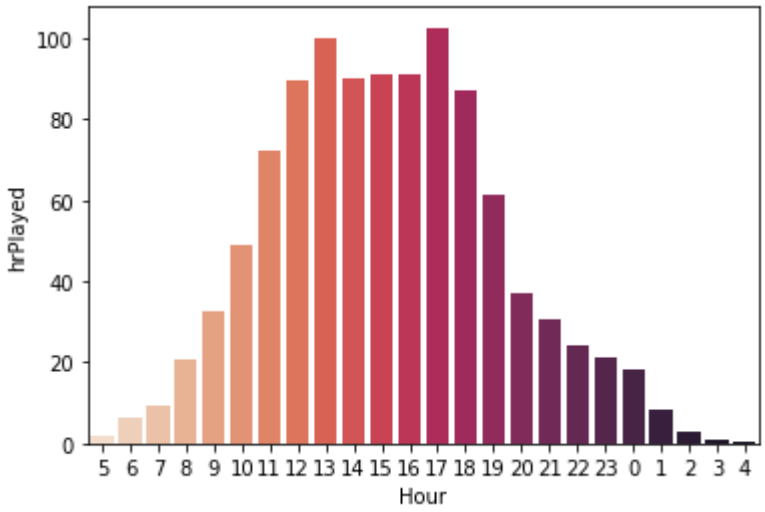


Figure 3.1. *Total Amount of Music Played at Each Hour of the Day*

The total listening time per artist will prove useful to us since we will utilise it in order to discover the user's top artists (ranked by listening time) and weigh our recommendations based on that. The rest of the statistics will be provided to the user together

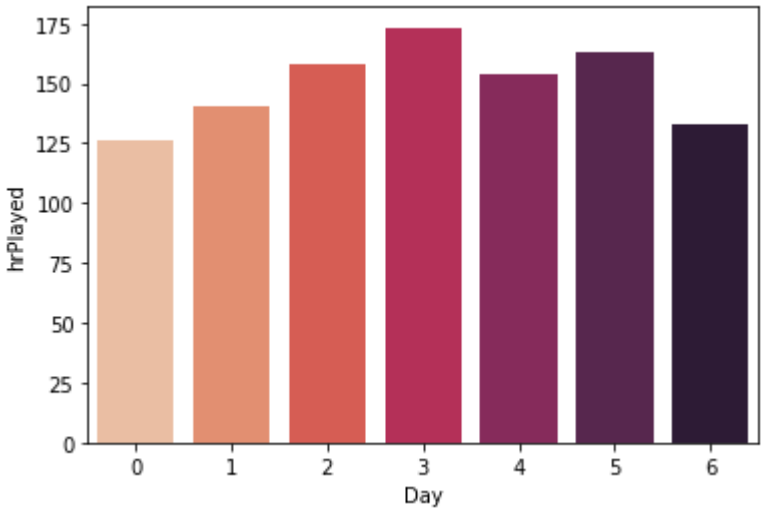


Figure 3.2. *Total Amount of Music Played at Each Day of the Month*

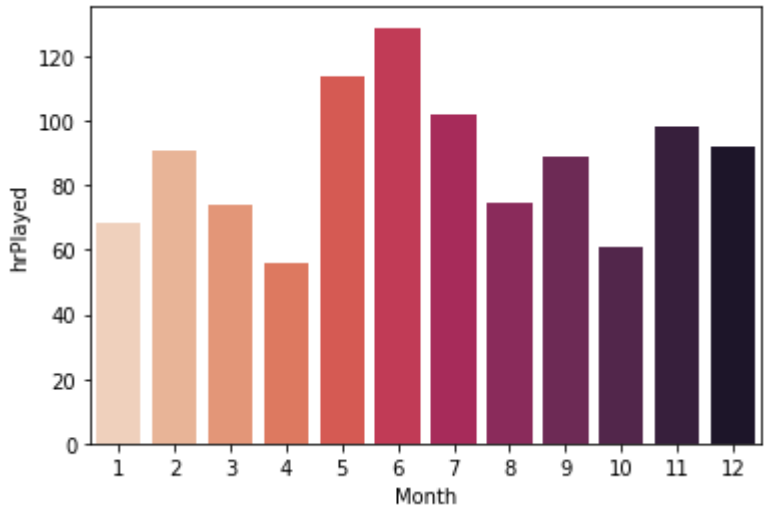


Figure 3.3. *Total Amount of Music Played at Each Month of the Year*

with the final song recommendations in the form of graphs, as a way to better visualise their listening habits. Some examples of these visualisations are 3.1, which shows the total amount of music played at each hour of the day during the past year, 3.1, which visualises the same statistic but for each day of the week and 3.3 which expands upon this and provides a visualisation for each month of the year.

These statistics can help the user better understand their listening habits and might also be used in future work that derives from this thesis. For example, one could link patterns that arise in the listening times of a person (referring to if someone listens to music mostly at night or on Fridays or around Christmas etc) with the mood of the music they listen to in these specific times. This would would allow for more targeted recommendations based on mood or activities - maybe someone is listening to more intense music only when they work out (which would be a clear recurring pattern on the

stats) or calmer music when they study, etc.

3.3.3 Automated Clustering & User Vectors

After pre-processing is completed, it is time to generate a user vector. This refers to a set of features that describes what type of music the user enjoys on average, with values for danceability, acousticness, valence and all of the other audio features described earlier. Essentially, it is going to be the average vector of the user's most listened to artists. This will serve as a "user profile" and will be compared against other artists in the database in order to make recommendations.

As mentioned previously, we are using only a fraction of the listener's top artists in order to generate recommendations. In the current form of the algorithm, the default amount of top artists we use is 50. Obviously, there is no single value that is objectively ideal for every use case. Furthermore, generating recommendations based on 10 artists might yield significantly different results than generating recommendations using 100 artists. For this reason, we employ one of the ideas mentioned above: weighting each artist's influence on the user vector based on the time the user has listened to their music. Specifically, when we are creating the user profile, instead of simply averaging all top artist vectors, we calculate a weighted average. This means that if a user has spent 90% of his time listening to one artist, that artist will contribute roughly 90% (in practice it would be more than that, since 90% is the percentage of time the user listened to that artist as a percentage of the total amount listened to their *top* artists, not every artist in their listening history. The non-top artists of the listening history are removed in pre-processing, thus slightly increasing the overall listening time share of each top artist) to the weighted average and subsequently the user profile generated. In this way, we ensure that adding more artists to the "top artist" subset doesn't dramatically affect the consistency or the genre of the recommendations.

Of course, in most cases, a person's musical taste can not be averaged to a single set of characteristics that describes all of their favorite music. That arises from the fact that people rarely listen to exclusively only one style of music - even within genres with devout following such as Metal or Hip Hop, there are immense variations in sound and style between different sub-genres. This means that averaging all of a listener's top artists into one vector will likely not yield optimal results in case there are more than one genres (or sub-genres) present in the listening history. What happens in the case of a new parent that enjoys listening to Jazz on their own time but also, on the same Spotify account, frequently streams nursery rhymes for their newborn and instrumental music for car rides with the family? Averaging these vastly different genres will, in all likelihood, yield poor recommendations that belong neither in Jazz nor in the nursery rhyme or instrumental music genres.

Our solution to this is clustering the user's top artists into groups, creating averaged user profiles and then providing separate recommendations for each cluster. Obviously, every person's top artists are going to be clustered in a different amount of groups so, before we even deploy a clustering algorithm, we need to determine the appropriate amount

of clusters for that particular user. This process is fully automated and relies on the Calinski - Harabasz score, also known as the Variance Ratio Criterion. This is defined as the ratio of the sum of between-cluster dispersion and of within-cluster dispersion and is a measure of how similar an item is to its own cluster versus different clusters. Mathematically, this is defined as:

$$Calinski - Harabasz = \frac{\frac{\sum_{i=1}^{k} n_i(\mu i - \mu)(\mu i - \mu)^T}{k-1}}{\frac{\sum i=1^k \sum x \in C_i (x-\mu_i)(x-\mu_i)^T}{n-k}}$$

Where: n_i is the number of points in cluster i μ_i is the mean vector of cluster i μ is the overall mean vector of all data points C_i is the set of data points in cluster i x is a data point

The numerator of the formula represents the between-cluster variance and is calculated by summing up the squared distances between the cluster means and the overall mean, weighted by the number of data points in each cluster. The denominator of the formula represents the within-cluster variance and is calculated by summing up the squared distances between each data point and its corresponding cluster mean.

Once we have defined the appropriate number of clusters, we can allocate the top artists into them by utilising the K-Means clustering algorithm. Since, in the scope of this thesis and any future work done on it, we might encounter the need to cluster thousands of artists (in case the user wants to take into consideration every single artist they listened to in the past year) or millions of songs (in case in some future modification of this thesis user profiles are created based on unique songs and not artists), another option would be to utilise the Mini Batch version of the K-Means algorithm. Mini Batch K-Means aims to combat the increase in computation time that comes with bigger datasets by partitioning the input data in random, fixed size batches, thus removing the need to store the entirety of the dataset in memory. Every iteration of Mini Batch K-Means runs on a fresh random sample and updates the current clusters through a convex combination of the original clusters and the new data. The change in the learning rate is inversely proportional to the number of iterations, with the numerical value of the learning rate equating to the inverse of the number of sample points assigned to a cluster.

When it comes to the programmatic implementation of the thesis, both the regular and the Mini Batch version of K-Means are available to the user and can be easily alternated through a simple Boolean function parameter, with regular K-Means being the default option.

3.4 Making recommendations

Now that we have multiple clusters of artists, each corresponding to a different style of music the user listens to, we generate a "user profile" from each of them. As mentioned above, this is essentially the average feature vector of all artists in the cluster, weighted by the time the user spent listening to each of the artists. We will therefore generate recommendations for each of these vectors, using each of them as input, one by one.

The process of generating recommendations is based on using a similarity metric to discover artists whose average features most closely resemble the user vector in question.

The two main approaches available in our algorithm are K-Nearest Neighbors and Cosine similarity, with the latter being the default. The artists that have the highest similarity to the user vector are then suggested as recommendations to the listener.

3.4.1 Exploring new genres

In order to provide a solution to the recommendation bias that is evident in other music RS, we employ a variety of methods.

First of all, when receiving recommendations from the RS, the user has the option to tune a "max similarity" setting. This is a value that goes from 0 to 100 and acts as a similarity "cutoff". In other words, if the user sets max similarity to 80%, no artists with a cosine similarity above 0.8 will be recommended. Tuning this value allows the user to explore genres that fall outside of their usual preferences.

Additionally, we have implemented similar, adjustable cutoff values for popularity and following. Any artist with a popularity and / or following above the corresponding cutoff value will not be suggested by the RS. Modifying these settings allows the user to opt in for recommendations of less popular artists, thus avoiding any bias other RS might have towards more popular artists and allowing the user to explore artists that are new to Spotify.

Chapter 4

Results and Analysis

One of the most challenging aspects of designing a recommender system is that of evaluating the quality of its recommendations, as well as ensuring the lack of bias in their generation. This stems from the fact that there is no inherent, absolute, universally correct answer to questions such as what is the best song to recommend to a person, what is the most suitable item to recommend to a customer for purchase or what is the most appropriate article to recommend to a reader. Everyone has different tastes and preferences. This highlights the need for large amount of labeled data (in this case songs rated by users whose preferences are also documented and labeled) when evaluating recommender systems. This opens the door to evaluating new algorithms with the use of A/B testing.

A/B testing is a common technique used to evaluate the effectiveness of recommender systems [29]. In an A/B test, two versions of the recommender system are compared: the "control" version (the "tried and tested" system that has been in use for some time and has proven results) and the "experimental" version (the newer algorithm whose performance we want to evaluate). Users are then randomly assigned to one of these two versions and their interactions with the system are tracked and analyzed to determine which version performs better. In the context of music recommender systems, some interactions that might interest us would be: how much time the user spends listening to songs, how many songs on average does the user skip, how many of the recommended songs are being added to playlists or favorited, etc.

However, it is obvious that properly running an A/B test requires a large amount of user data, from a multitude of users. Since, in the scope of this thesis, we are trying to explore the life cycle of a recommender system without access to millions of user data, what are some other ways in which we can evaluate our algorithm? In this chapter, we explore a few alternatives used by this thesis. These methods are split in two categories: "Spotify-based" metrics and "user-based" metrics.

Spotify-based evaluation centers around the idea of using Spotify's recommendation engine and its recommendations for the user as our labelled, test dataset that evaluates the accuracy of our recommendations. Naturally, this would lead our algorithm to start behaving more and more like the Spotify algorithm which inherently leads to multiple problems, all of which we will explore later in this chapter.

On the other hand, user-based evaluation centers around the idea of having real users,

who have provided their Spotify history to us, participating in a single-blind study. In that study, they would be exposed to recommendations from multiple, different sources and they would have to rate them according to how much they enjoy them or not.

4.1 Spotify-based Evaluation

As previously discussed, Spotify-based evaluation centers around the idea of using the results of Spotify's recommendation engine as ground truth for the assessment of our algorithm's performance. We can leverage Spotify's engine through the service's API, which allows us to call the function "Get Recommendations". This function can take a variety of artists, genres or tracks as input and returns a list of recommendations generated by Spotify.

The way in which we take advantage of this API function for evaluating our recommender system is as follows. For every one of the artist clusters generated for a user, we select the top five artists. We then use these artists as the input to both our recommender system as well as the Spotify recommendation API. The reasoning behind selecting five artists is that the "Get Recommendations" function can take a maximum of 5 unique artists in consideration when generating recommendations. We instruct both our recommender system and Spotify's API to produce 100 recommendations. Our RS's performance can now be measured by using R-Precision.

4.1.1 R-Precision

In 2018, Spotify, The University of Massachusetts, Amherst, and Johannes Kepler University, Linz ran a Machine Learning challenge [30] focused on music recommendation and, specifically, the enrichment of a user's playlists with new songs (given an initial set of tracks in a playlist, predict the subsequent tracks in said playlist). The dataset contained 1,000,000 playlists created by users on Spotify platform between January 2010 and October 2017. One of the ways the participants were evaluated, was through the use of a metric called "R-Precision". Specifically:

R-precision is the number of recommended, relevant tracks divided by the number of known relevant tracks:

R-precision = $\frac{|G \cap R_{1:|G|}|}{|G|}$

We are utilising a modified version of that metric in order to measure the performance of our RS. In our case, instead of taking note of relevant tracks, we are taking note of relevant artists. A known relevant artist, is an artist recommended by the Spotify algorithm. A recommended relevant artist is an artist recommended by our algorithm that is also recommended by Spotify's API when given the same input. Essentially, we are calculating the percentage in which our RS recommendations match those generated by Spotify's recommendation engine.

Follower Cut-Off	Mean R-Precision
75K	0.094432
50K	0.091044
25K	0.084729

Table 4.1. *Follower Cut-Off Relationship with R-Precision*

Popularity Cut-Off	Mean R-Precision
5	0.095200
1	0.09066
0	0.090309

Table 4.2. *Popularity Cut-Off Relationship with R-Precision*

4.1.2 Evaluating the Best Parameters for R-Precision Optimisation

Before we talk about some of the drawbacks the usage of R-Precision introduces to our method, we can use it to perform an initial grid search for optimising the hyperparameters of our model. More specifically, we are interested into how the cutoffs for follower and popularity count affect our results. While keeping other hyperparameters stable, we run multiple iterations of our algorithm on the data of 5 different users. Judging from tables 4.1 and 4.2, we can see there is a slight correlation between the cut-offs and an increase in R-Precision. However, it's worth noting that due to the small sample size of the experiment, we can not be certain that a statistically significant correlation exists between these hyperparameters and the way our algorithm performs.

4.1.3 Drawbacks of R-Precision

Even though R-Precision might give us an initial evaluation of our algorithm (after all, a recommender system that mimics Spotify's recommendation engine is, to a certain degree, an effective recommender system), there are some obvious drawbacks to it. First and foremost, trying to morph our algorithm into that of Spotify but with dramatically less data obviously leads us down a path where, no matter what we do, we can not offer better tailored recommendations than Spotify. That comes from the fact that, in that case, we would essentially be trying to "discover" the black box system of Spotify and run it ourselves but without having access to the vast amounts of user data or computing power that Spotify has. What is more, by trying to align to Spotify's algorithm, we are automatically "adopting" all of it's biases. This would defeat one of the purposes of this thesis - trying to propose new artists that are relatively unknown, without many track plays and possibly not completely aligned to the user's average listening habits.

4.2 User-based Evaluation

If R-Precision leads to all of the problems outlined above, then what could be a more accurate evaluation metric? Once again, we run into the problem of user data. Since, under the constraints of this thesis, we can not run large scale A/B testing (due to the lack

of unique users) we turn to personalised, single-blind surveys with single, uncorrelated users. In other words, we asked for users to provide us with their Spotify listening history file in order to be served personalised recommendations. These files were then pushed through our algorithm's pipeline and personalised recommendations were generated.

It's worth noting that the files of these unique users were pushed through the algorithmic pipeline in different instances of the code, all of which retained no information about the weights calculated for other users or the recommendations generated for them. This was done in line with the core idea of the thesis; trying to suggest new artists while having access only to the user's personal listening history and absolutely no other user data.

For every user (and therefore every listening history file), this process was repeated three different times, with three different sets of hyperparameters. In detail:

- Group 1: Our algorithm recommends the songs that closest match the average preferences of the user (similarity setting of 1), generates weighted user vectors, has a popularity cutoff of 1 and a following cutoff of 75 thousand. This is the best our algorithm has to offer, according to the R-Precision metric.

- Group 2: Same settings as above, with one key difference - the similarity setting is set to 0.98. This means that the algorithm will not suggest artists that are above a 98% match. This forces the algorithm to recommend artists outside the user's most common genres.

- Group 3: No follower or popularity cutoff. Similarity setting set to 1. This version of the algorithm doesn't follow the R-Precision way and instead is able to recommend literally any artist on the platform, even those with no recorded plays and/or no followers.

The same history file is also passed to the Spotify recommendation system via the API method mentioned above.

For all of the methods mentioned above and for each one of the resulting artist recommendations, their top Spotify song is then added to a list. The order of the songs in the list is then randomised and the list is sent to the users who rate each song from 1-5, based entirely on personal criteria. No instructions are given by the researcher, other than to rate each song based on how much the user likes it or not. The scores given by the users are then averaged. In case the same artist is recommended by multiple versions of the algorithm and/or Spotify, the score is factored into the average score of all algorithms involved.

The goals of this setup are:

1. Examine how close we can get to Spotify's RS performance when we use the hyperparameter values that give us the best possible R-Precision (Hence the inclusion of Group 1).

2. Identify how are ratings affected by recommending artists that are slightly outside the usual range of the user's preferences (Hence the inclusion of Group 2).

Algorithm	Mean Score	STD	Mode
Group 1 (simil = 1)	3.28	1.06	3
Group 2 (simil = 0.98)	2.83	1.13	3
Group 3 (no follower constraints)	2.71	1.21	4
Spotify	3.57	1.08	4

Table 4.3. *User-based evaluation results*

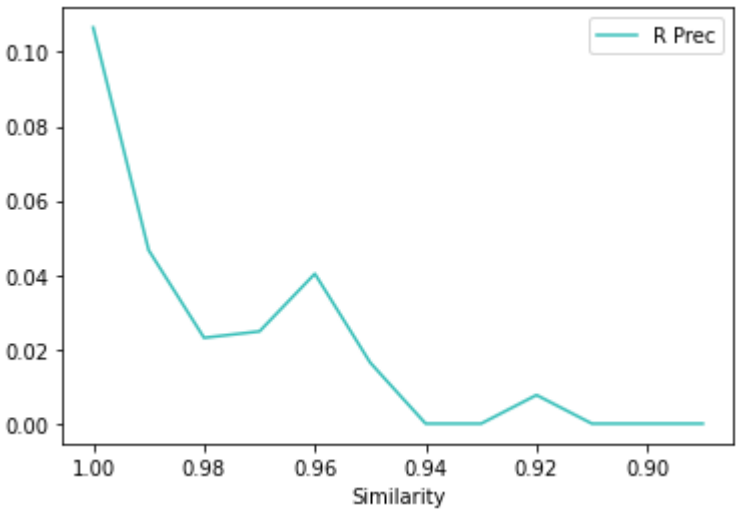
3. Evaluate the quality of recommendations a recommender system without follower/population biases generates (Hence the inclusion of Group 3).

All in all, five users participated in the experiment. Each one of them provided their personal Spotify history from the past year and evaluated a number of songs ranging from 60 to 100, based on how many artist clusters were generated for their user vector. The results of this experiment can be seen in table 4.3.

4.2.1 Results for Different Similarity Settings

From the results presented above it is obvious that decreasing the similarity setting results in an immediate penalty in average user rating. When users listened to artists that matched as close as 99.9% of their user vector, they gave an average score of 3.28/5 (with a standard deviation of 1.06 and a mode of 3). On the other hand, when they were exposed to songs capped at 98% similarity (while keeping all other experiment parameters constant) the average song rating dropped to 2.83/5 (std: 1.13, mode: 3).

Due to the amount of effort and time commitment needed by the experiment participants, it was not feasible to test a multitude of different values for the similarity hyperparameter. However, using the recommendations generated for each user, their listening histories and the R-Precision metric, we ran multiple, automated simulations where recommendations were generated and evaluated for all possible values of similarity, while always keeping the rest of the experiment variables steady. The results can be seen in graph below.



We see a steady drop in R-Precision the further away we go from similarity = 1.0. That being said, for future experiments that might include user-based ratings, it is worth experimenting with multiple similarity values that are higher than 0.98 but not quite 1.

4.2.2 Results That Ignore Artist Following

On the other hand, the results for Group 3 are more promising, even though the average score drops to 2.71/5 (std: 1.21). Surprisingly, the mode for this group was 4 (in contrast to groups 1 and 2 which had a mode of 3) with 2 users reporting higher average ratings for Group 3 than Group 2 and 1 user reporting higher ratings for Group 3 than Group 1.

It is obvious that when we remove the restrictions of popularity and following, there exist many more artists that are closer to the user vector than before, even if they have 0 followers and/or 0 Spotify plays. The problem is, however, that many times these artists are in different languages than the languages spoken by the user. According to the people participating in the study, this affected the ratings although we can not draw any conclusions just based on after the fact reports from users. This is an effect that would need to be studied in further experiments. It also points out to a possible need for filtering recommendations by language although this should be done with care as it might inadvertently introduce geolocation or ethnic biases to the algorithm.

4.2.3 Comparison With Spotify

When compared to Spotify's recommendation engine, no version of our algorithm manages to deliver recommendations that receive higher average rating. This is something that was partly expected due to the lack of algorithmic capabilities and data granularity that naturally arises from the scope of this thesis. However, the difference between our algorithm's results and those of Spotify's is narrow enough to conclude that a decently useful recommender system can be created without much data or computing power.

Chapter 5

Discussion

5.1 The Importance of User Data & the Cold Start Problem

Throughout the results of this thesis, it is evident that user data plays a significant role in the development of a recommender system. On one hand, the more data a recommender system has access to, the more accurate it can be in predicting user preferences and making recommendations. More data can help to identify patterns and trends that might be missed with smaller datasets, leading to more accurate and personalized recommendations. Furthermore, with more user data, a recommender system can provide recommendations for a broader range of music genres. This is because the system has a better understanding of user preferences and the relationship between the different genres, which allows it to recommend tracks that users might not have discovered otherwise.

Additionally, having access to an increased amount of user data allows the algorithm to provide recommendations that are more diverse and varied. This helps to prevent users from getting stuck in a "filter bubble" and can introduce them to new and unexpected songs or artists. It also helps make recommender systems more robust to changes in user behavior, preferences, or demographics. This means that the system can continue to provide accurate and relevant recommendations even as user behavior evolves over time.

However, one of the most important problems that becomes more apparent when working with limited or sparse data, is the cold start problem. The cold start problem refers to the difficulty of providing accurate recommendations for new users or items that lack sufficient interaction data. This challenge arises due to data sparsity, where there is insufficient information available about the users or items to make accurate predictions. Additionally, the problem may involve data heterogeneity, where the available data is too diverse or inconsistent to provide meaningful recommendations (for example, an edge case could be trying to recommend new music to a user that's listening to a new, different, unique band every day).

5.2 Testing Recommender Systems

Surprisingly, the biggest challenge faced was evaluating the recommender system created. How can we confirm that we are serving the best possible recommendations

when music preferences are a topic where no absolute truth exists? Different artists and genres might appeal to different people. And, as a thought experiment, even if we create a recommender system that suggests 10 new songs and users report liking 8/10 of them, how can we be certain that there doesn't exist a different set of hyperparameters that results in a 9/10 score?

As mentioned earlier, we don't have a user base that would allow us to perform neither a proper A/B test nor a user survey. As a matter of fact, none of the popular recommendation system evaluation methods used in the industry are suitable for this thesis. For example:

One widely used approach for testing recommender systems is to use historical data to simulate real-world usage scenarios. This approach involves splitting the available data into training and testing sets, and using the training set to train the recommender model and the testing set to evaluate its performance. Common metrics used to evaluate recommender systems include accuracy measures such as precision, recall, and F1-score, as well as ranking measures such as mean average precision (MAP) and normalized discounted cumulative gain (NDCG).

Another approach for testing recommender systems is to conduct user studies and gather feedback from actual people. This approach involves recruiting a representative sample of users, presenting them with recommendations generated by the system, and gathering feedback on the relevance and usefulness of the recommendations. User studies can provide valuable insights into the effectiveness of the system, as well as the user experience and user satisfaction.

In recent years, there has been growing interest in using simulation techniques to test and evaluate recommender systems. Simulations can provide a more controlled and reproducible environment for testing, as well as the ability to test the system under a wide range of scenarios and conditions. For example, one recent study used agent-based simulation to evaluate the performance of a recommender system in a simulated e-commerce environment.

However, even in the industry, there are challenges to performing these types of tests. Some of these challenges include the lack of ground truth data, the difficulty of defining appropriate evaluation metrics, and the need to balance competing objectives such as accuracy, diversity, and serendipity.

5.3 Conclusions of the Thesis & Future Work

The bottom line of this thesis is that a simplistic recommender system which doesn't have access to data of more than one user might not be able to reach the quality levels of Spotify's algorithm or deal that well with edge cases (users with almost no data, users with extremely varied preferences), however it performs surprisingly well and can still be a strong source of discovering new music, without requiring the technology or data that Spotify has.

Despite the hurdles that arise from the lack of user data, the algorithm produced by this thesis can still play an important role in improving a user's exploration of music.

This is primarily achieved because of our algorithm's ability to generate more obscure and less popular recommendations on demand, allowing the user to freely explore the music suggested to them without the recommendations being affected by any societal factors or biases. Another area where the thesis, in its current state, can prove useful is that of detailed user statistics generation.

The functionality of this thesis can be expanded on multiple different fronts. On one hand, music recommendations can be further personalised and improved if we provide users with the ability to rate the recommendations generated by the RS, as well as any song they listen to. We could then use said ratings to fine-tune our RS. The drawback of this approach is that our algorithm would now require some kind of non-volatile memory in order to store user data.

On the other hand, the thesis can be expanded on the user interface / user experience (UI/UX) front, by building a web interface from which the user can control the different parameter of the RS. Within the same UI, we can provide the functionality of personal statistics discussed earlier in this thesis and showcase a variety of interactive graphs generated in order to showcase patterns in the user's listening history.

On a more academic level, the application of cutting edge techniques such as zero-shot learning might bring this thesis closer, performance-wise, to a commercial music RS. The way in which the recommender system is tested should also be re-evaluated, with a bigger group of users participating in the experiment.

Bibliography

[1] International Federation of the Phonographic Industry. *Global Music Report 2022 - State of the Industry*. https://www.ifpi.org/wp-content/uploads/2022/04/IFPI_Global_Music_Report_2022-State_of_the_Industry.pdf, 2022.

[2] International Federation of the Phonographic Industry. *Global Music Report 2012*. https://www.musikindustrie.de/fileadmin/bvmi/upload/06_Publikationen/DMR/ifpi_digital-music-report-2012.pdf, 2012.

[3] Ulrich Dolata. *The digital transformation of the music industry. The second decade: From download to streaming*. SOI Discussion Paper 2020-04, Stuttgart, 2020.

[4] Spotify. *Spotify Q2 2022 Earnings Update*. https://s29.q4cdn.com/175625835/files/doc_presentation/Q2-2022-Shareholder-Deck-FINAL.pdf, 2022.

[5] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas και Domonkos Tikk. *Session-based Recommendations with Recurrent Neural Networks*, 2015.

[6] Maxim Naumov, Dheevatsa Mudigere, Hao Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong και Misha Smelyanskiy. *Deep Learning Recommendation Model for Personalization and Recommendation Systems*, 2019.

[7] Paul Covington, Jay Adams και Emre Sargin. *Deep Neural Networks for YouTube Recommendations*. *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[8] David Goldberg, David Nichols, Brian M. Oki και Douglas Terry. *Using Collaborative Filtering to Weave an Information Tapestry*. *Commun. ACM*, 35(12):61–70, 1992.

[9] Robinvan Meteren. *Using Content-Based Filtering for Recommendation*. 2000.

[10] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang και Xiangnan He. *Bias and Debias in Recommender System: A Survey and Future Directions*, 2020.

[11] Himan Abdollahpouri, Masoud Mansoury, Robin Burke και Bamshad Mobasher. *The Unfairness of Popularity Bias in Recommendation*, 2019.

[12] Alessandro B. Melchiorre, Eva Zangerle και Markus Schedl. *Personality Bias of Music Recommendation Algorithms*. σελίδα 533–538, 2020.

[13] Robin Burke. *Hybrid Recommender Systems: Survey and Experiments*. *User Modeling and User-Adapted Interaction*, 12, 2002.

[14] K. Pyrovolakis, P. Tzouveli και G. *"Stamou (2021), "Mood detection analyzing lyrics and audio signal based on deep learning architectures, " 2020 25th International Conference on Pattern Recognition (ICPR)"*. *2021*, 10.:9363–9370, 2021.

[15] "K Pyrovolakis και G Stamou" P Tzouveli. *Multi-Modal Song Mood Detection with Deep Learning*. *Sensors 22 (3)*, 1065(3):35–43, 2009.

[16] Christopher C. Johnson. *Logistic Matrix Factorization for Implicit Feedback Data*. 2014.

[17] Rachel M. Bittner και Juan J. Bosch. *Generalized Metrics for Single-f0 Estimation Evaluation*. *International Society for Music Information Retrieval Conference*, 2019.

[18] James McInerney, Benjamin Lacker, Samantha Hansen, Karl Higley, Hugues Bouchard, Alois Gruson και Rishabh Mehrotra. *Explore, Exploit, and Explain: Personalizing Explainable Recommendations with Bandits*. *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, σελίδα 31–39, New York, NY, USA, 2018. Association for Computing Machinery.

[19] Antonia Saravanou, Federico Tomasi, Rishabh Mehrotra και Mounia Lalmas. *Multi-Task Learning of Graph-based Inductive Representations of Music Content*. *Proceedings of the 22nd International Society for Music Information Retrieval Conference*, σελίδες 602–609, Online, 2021. ISMIR.

[20] Alessandro B. Melchiorre, Navid Rekabsaz, Emilia Parada-Cabaleiro, Stefan Brandl, Oleg Lesota και Markus Schedl. *Investigating gender fairness of recommendation algorithms in the music domain*. *Information Processing & Management*, 58(5):102666, 2021.

[21] Andres Ferraro, Xavier Serra και Christine Bauer. *Break the Loop: Gender Imbalance in Music Recommenders*. *Proceedings of the 2021 Conference on Human Information Interaction and Retrieval*, σελίδα 249–254, New York, NY, USA, 2021. Association for Computing Machinery.

[22] Oleg Lesota, Alessandro Melchiorre, Navid Rekabsaz, Stefan Brandl, Dominik Kowald, Elisabeth Lex και Markus Schedl. *Analyzing Item Popularity Bias of Music Recommender Systems: Are Different Genders Equally Affected?* *Fifteenth ACM Conference on Recommender Systems*. ACM, 2021.

[23] Dominik Kowald, Markus Schedl και Elisabeth Lex. *The Unfairness of Popularity Bias in Music Recommendation: A Reproducibility Study*, σελίδες 35–42. 2020.

[24] Lex Beattie, Dan Taber και Henriette Cramer. *Challenges in Translating Research to Practice for Evaluating Fairness and Bias in Recommendation Systems*. *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, σελίδα 528–530, New York, NY, USA, 2022. Association for Computing Machinery.

[25] Maryam Aziz, Alice Wang, Aasish Pappu, Hugues Bouchard, Yu Zhao, Benjamin Carterette και Mounia Lalmas. *Leveraging Semantic Information to Facilitate the Discovery of Underserved Podcasts*. *Proceedings of the 30th ACM International Conference on Information &; Knowledge Management*, CIKM '21, σελίδα 3707–3716, New York, NY, USA, 2021. Association for Computing Machinery.

[26] *Spotify API*. https://developer.spotify.com/documentation/web-api/.

[27] *Kaggle.com - 8+ M. Spotify Tracks, Genre, Audio Features*. https://www.kaggle.com/datasets/maltegrosse/8-m-spotify-tracks-genre-audio-features.

[28] *Spotify Data Rights and Privacy Settings*. https://support.spotify.com/us/article/data-rights-and-privacy-settings/.

[29] Preetam Nandy, Divya Venugopalan, Chun Lo και Shaunak Chatterjee. *A/B Testing for Recommender Systems in a Two-sided Marketplace*, 2021.

[30] *AI Crowd - Spotify Million Playlist Dataset Challenge*. https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge.