UTRECHT UNIVERSITY

Department of Information and Computing Science

Artificial Intelligence Master's Thesis

Interactive evolution: interactive genetic algorithms for addressing popularity bias in music recommender systems

First examiner:
Eelco Herder

Second examiner:
Jeroen Ooge

Candidate:
Jip Sierksma

Student number:
6564275

Version: September 25, 2024

Abstract

Within recommender systems, there is a well-known bias called popularity bias, where these systems tend to recommend more popular items, over items in the so-called long-tail, which refers to the vast number of less popular items that collectively make up a significant portion of the total data. Popularity bias causes problems for both users and artists, such as limited exposure for niche or undiscovered artists and a lack of variety in users' music consumption habits. To address this issue, we developed an interactive genetic algorithm (IGA) for music recommendations, which evolves a population of recommendations based on user feedback. Our method improves on previous approaches by incorporating mutation, as well as dynamic crossover and mutation rates. We benchmarked our method against a previous approach using simulated users. Results show that our method shows similar feedback scores across all users as the benchmark. However, the convergence rate was higher, meaning optimal solutions were found more quickly. Moreover, our method improves feedback scores significantly for users with more niche interests, showing a 150.85% improvement from the initial to the final generation, whereas the benchmark shows a statistically lower increase of 107.27%. For users with traditional preferences, our system showed similar performance to the benchmark. The results suggest potential for the real-world applications of IGAs for music recommendations, as well as show the impact of incorporating mutation and dynamic genetic operation rates into IGAs.

Contents

1. Introduction

Before the internet, music consumption was limited by the medium it was consumed in. Vinyl, cassette tapes, CDs and other physical formats defined the way people accessed and listened to music. For music discovery, people relied on the radio, recommendations from family and friends, or serendipitous encounters at record stores. The constraints that physical mediums bring, not only shaped the listening experience, but also the width of music exposure. When online stores came along, the music consumption landscape changed. Online stores still sold the same physical formats, but consumers were no longer limited by the amount and diversity available in their local record store. However, having a large amount of items available, as is possible in these online stores, can cause information overload for users[1]. This can result in difficulties in selecting items for consumption.

Music recommendation systems

With the increasing availability of personalized music purchasing and listening, starting with online stores that specialize in selling single songs, such as iTunes, and followed by music streaming services such as Spotify and Apple Music, usage of music recommendation systems has skyrocketed[2]. Research has shown that consumers today are facing a problem of information overload for music data[1], and this is the case with modern methods of consumption such as streaming as well[3]. This overload occurs when individuals are presented with an overabundance of music choices, making it difficult for them to make informed decisions about what to listen to. This information overload is why the need and desire for music recommendation algorithms exists. People want to find music that is tailored to their tastes, and in a world where there is more and more music every day, music recommendation algorithms help information overload, by curating music that a user is expected to enjoy.

In 1999, the theory of Information Foraging was introduced by Pirolli and Card[4]. This theory is an approach of understanding how people seek and consume information. The theory states that people will, when possible, (try to) maximize their rate of gaining information. Individuals, much like animals foraging in the wild, will seek out an environment where their so-called "information scent" can be maximized. The information scent is the perceived value of the information compared to the cost of obtaining it. In the context of music consumption, this theory suggests that users engage in behavior that allows them to discover music that aligns with their tastes while minimizing the effort required to find it. Music recommendation systems help with this, by maximizing the information scent, lowering the cost of finding information (music), and having the information be of a higher perceived value since it aligns with their taste.

However, as music consumption keeps evolving, so do the challenges associated with information overload and recommendation algorithms. While recommendation systems can help with the information overload found in modern music services[3] by offering personalized suggestions, they can also contribute to filter bubbles and echo chambers[5], where users are only exposed to a limited range of content that aligns with their existing preferences.

Music recommendation systems (MRS) are a subfield of recommender systems. The objective of a music recommender system is to provide users with suggestions for music based on their preferences, which are inferred from the users' history and other factors such as the user's listening history and how the user's history compares with that of others, as well as contextual information such as time of day and season, and, in the case of some recommenders, even mood and emotion[6].

Genetic algorithms

One promising new avenue for personalized recommendation is genetic algorithms. These algorithms are based on principles of evolution in nature. Items are selected for reproduction to the next generation based on their fit-

ness. This fitness is a measure of how well the output of the system fits the goals of the genetic algorithm.

One of the goals a recommender system can have is accuracy, or matching the users' tastes as closely as possible. However, having this goal might not always be desirable. It has been shown that satisfaction in recommender systems is also influenced by the diversity of recommendations[7]. It is easy to imagine that repeatedly getting extremely similar items recommended would not lead to a high satisfaction rate, even though it might be highly accurate. Because of the deviation from focusing on accuracy, literature in the recommender space has developed several different metrics to measure performance. This allows the recommender system to improve metrics that account for both accuracy and diversity, as well as other metrics for measuring implied satisfaction such as novelty, or how much the item differs from the user's history. This can lead to more balanced recommendations that better satisfy users' needs for diversity in their recommendation.

Since recommendation is therefore no longer only focused on optimizing accuracy, recommendation has become a multi-objective optimization problem (MOP). Genetic algorithms are able to find so-called Pareto optimal solutions to these multi-objective optimization problems[8]. This means that the algorithm tries to find a solution that scores well on all metrics. Selecting the items that have the highest fitness for the next generation allows these algorithms to "evolve" solutions to problems that satisfy multiple criteria, such as is the case for recommender systems.

This is also the case for interactive genetic algorithms (IGAs), which are a type of genetic algorithm, where the fitness is determined by a user. This means that there are no explicit goals for the algorithm to try and reach, but the outcome is determined by choices users make during the process of the run. This solves a problem with genetic algorithms, where the output of the algorithm is very sensitive to what fitness function is chosen.[9][10] The fitness function in a genetic algorithm is the function that is used to calculate the fitness. Interactive genetic algorithms do not have a clear fitness function, since user feedback is what steers the algorithm in the right direction

instead of a fitness function. In the IGA, the user's preferences shape the outcomes, since the feedback determines in which direction the algorithm evolves. This means that the algorithm is able to follow any goals that the user thinks are important, including getting diverse recommendations.

Navigating music as a recommendation item

Music recommendation can be considered a special case within recommendation systems. Music is very personal, and there exist a lot of types of music. The success of a recommender system lies in its ability to capture the nuances of individual taste, considering factors such as genre preferences, mood and cultural influences. Compared to systems for movies or TV, music recommender systems experience much higher interaction rates, as users engage with more individual tracks than they do individual movies or TV shows. Another factor that needs to be considered is a user's preference for more mainstream or more obscure music. Temporal dynamics can also play a role in shaping the preferences a user has, as user preferences can change over time, based on the time of year (holiday music, summer tunes, etc.) and even the time of day and the activity (relaxing music in the evening, energetic music for exercise, etc.). Interactive genetic algorithms can respond to these constantly changing preferences by taking current user preferences into account and evolving a population of music to suit the users' current needs.

Bias within recommender systems

As mentioned previously, music recommender systems have the task of accurately modeling a user's individual taste. This does however not come without additional challenges. One of these challenges is popularity bias. This bias has been shown to affect recommender systems in general. It refers to the phenomenon where recommender systems tend to favor popular items over less popular ones. This bias can lead to an over-representation in the output of these systems of already popular hits or established artists, overshadowing emerging artists or genres that might better align with a user's preferences.

Within the framework of Information Foraging Theory, popularity bias is a distortion in the information-seeking of users, skewing their exploration towards more popular music at the expense of less popular songs. Using a recommendation algorithm decreases the cost of obtaining new information for the user, and therefore increases the information scent of this information. However, this ignores the idea that outside of the popular mainstream, there is information that will provide the user with stronger increases in the information scent, which is not shown by the recommendation algorithm. Therefore, information from recommender systems that have a larger degree of popularity bias has the possibility of increasing the information cost for users, in comparison to information from algorithms with a lesser degree of popularity bias.

This work will focus on the mitigation of the popularity bias through the use of interactive genetic algorithms. User interaction is what shapes the outcomes in an IGA. By incorporating user feedback into the system, the user is able to "steer" the system towards music that fits their tastes. Abdollahpouri et al. (2019)[11] discussed that user's preference for popularity is reflected in the kind of items they want to interact with. Having users influence their own recommendations with feedback allows them to receive more personalized recommendations by actively steering the selection toward items that align with their preferences.

To overcome the popularity bias and give satisfactory recommendations, the IGA needs to dynamically adjust to user feedback. This will allow the user to create their own recommendation list, which should match their tastes in music. To measure the popularity bias, we intend to explore evaluation measures such as long-tail coverage. Moreover, to assess the effectiveness of our method, we will compare it to preexisting methods, looking at changes in average feedback scores for multiple users.

The rest of this thesis is structured as follows: Chapter 2 contains a review of the existing literature on recommender systems and genetic algorithms, as well as on popularity bias and its mitigation. Chapter 3 contains the motivations, as well as outlining the methodology used for the devel-

opment and testing of the IGA. Chapter 4 presents the results of the experiments, and chapter 5 discusses the findings, limitations and future work. Finally, chapter 6 contains a conclusion to the research by summarizing the contributions and implications.

2. Literature review

This section will provide the necessary background to understand the problem the thesis is going to address. By looking at the existing knowledge, this section aims to identify gaps in the current understanding and lay out the basis for the proposed research. Firstly, previous work in recommendation will be discussed, including methodologies for content-based and collaborative filtering. Then, genetic and evolutionary algorithms will be discussed. Thereafter, the popularity bias will be discussed and how it can be mitigated.

2.1 Previous work in recommendation

In this section, previous approaches to recommendation will be discussed. As discussed in the introduction, consumers today are facing an information overload of music data[1]. This challenge is where music recommendation algorithms come into play. These algorithms are helpful to their users, as they present them with items that match their tastes. It has been shown that recommender systems are able to help people find the items they need effectively[12], thereby possibly enhancing user satisfaction with services that employ them.

2.1.1 Content-based filtering

The earliest music recommendation algorithms were basic rule-based systems that often relied on genres, pre-defined artist similarities, and user ratings, i.e. so-called "*content-based filtering*". These content-based filtering algorithms would select items for user's consumption based on the correlation between the user's history, and the content of items[13]. As an example, a user could like the song "Holiday" by Madonna. Based on this preference, a content-based filtering algorithm might recommend other songs such as

"Girls Just Want to Have Fun" by Cyndi Lauper due to their similarities in genre, musical style, and other metadata such as release year. One advantage of using this system is that content-based recommender systems are able to recommend items for which there is no rating data. This is because the system only looks at the content of the items, and whether the user has enjoyed them, and does not need data about other users[14]. This has the benefit of being able to recommend even items that are not well known, or more niche. These content-based approaches have limits however, as they struggle to capture the nuances of users' preferences, as well as having the limitation of needing to have the content of the items be in some way machine-parseable[13]. Content-based filtering will also not result in serendipitous discoveries when the algorithm works as intended. Due to the similarity-based nature of the algorithm, the recommended items will always be similar to the users' history. To address these limitations, collaborative filtering recommendation algorithms emerged, which analyze user behavior, and draw insights from a larger user group, instead of just the users' individual preferences.

2.1.2 Collaborative filtering

There exist different versions of collaborative filtering. User-based collaborative filtering algorithms compare the preferences of a target user to the preferences of users who share similar tastes. Item-based collaborative filtering recommends items based on their similarity to those that the user has shown interest in. There also exist hybrid versions of these two. These forms of collaborative filtering fall under the umbrella of neighborhood-based models, which are based on computing the relationships between users or items directly from the raw data. The other form is model-based collaborative filtering, which uses machine learning (ML) to predict missing ratings in the user-item matrix.

2.1.2.1 User-based collaborative filtering

One of the earliest implementations of collaborative filtering-based recommender systems is Tapestry by Goldberg et al. (1992)[15]. This system relied

on the explicit opinions of a group of people known by the users, which were used in tandem with the user's preferences to generate the recommendations. However, recommender systems employed in large systems, such as music recommenders within large music streaming services, cannot depend on all users knowing each other.

In 1995, Shardanand and Maes[16] introduced *Ringo*, a recommendation algorithm that specializes in recommending music albums and artists. This system is one of the first music recommendation systems. The system gathers user preferences through email, after which it compares them to other users of the system, to generate personalized suggestions based on the users' preferences. Various functions are used to calculate similarities between users for the purposes of recommendation, including mean squared difference, Pearson correlation and constrained Pearson correlation. These similarity scores are then used to recommend artists and albums.

However, user-based collaborative filtering approaches to recommendation, such as Ringo, come with some drawbacks. If a new item appears in the database, like is often the case with new music, there is no way it can be recommended to users until another user interacts with it. Thus, if the number of users is small (like it is with Ringo) or if the volume of information is high (like it is in modern music streaming services), there is a danger of the coverage of ratings becoming very sparse. This thins the collection of recommendable items[17]. Secondly, if a user's tastes do not align closely with the rest of the population, there will not be any users who are similar, leading to poor recommendations.[17] This sparsity problem is therefore a significant challenge for user-based collaborative filtering recommenders.

2.1.2.2 Item-based collaborative filtering

Sarwar et al. (2001)[18] evaluated several collaborative filtering algorithms. According to them, in addition to the sparsity problem, user-based collaborative filtering algorithms are also difficult to scale. Limits in computing power made it so that scaling to a website that recommends items to millions of users proved to be difficult. Moreover, the authors note that it was

also proving to be a challenge to improve the quality of recommendations for users. These two challenges are therefore in some ways at odds with each other, as the less computing power used by a user-based collaborative filtering algorithm to look for neighbors, the more scalable it will be, but this will most likely also negatively affect the quality of the recommendation. Therefore, they propose item-based collaborative filtering algorithms, which avoid these two challenges by recommending items based on items that are similar to other items the user has liked. Item-based collaborative filtering therefore has a reduced computational overhead, as the number of comparisons for user-based collaborative filtering grows exponentially as the number of users grows. Item-based collaborative filtering doesn't have this problem, as it only needs to compare items and not users. It also has better performance, as in real-world scenarios, user-item interaction data is sparse, as users interact with only a subset of all available items. Item-based collaborative filtering doesn't have this data sparsity problem, as it focuses on similarities between items, which makes it perform better. However, this can still lead to data sparsity in other ways. Users interact with only a small subset of items, causing the model to miss out on items that don't have large similarities with users' previous choices. Consequently, users end up receiving items that are similar to those they have already chosen, decreasing satisfaction.

Item-based collaborative filtering and content-based filtering are related, but different. Content-based recommendation, as discussed, focuses on the similarity between items, returning items that are similar to those the user has liked. Item-based collaborative filtering focuses on the similarity between user interactions with items, drawing on user data to infer relevant items. As an example, consider an online store selling technology, where a customer is looking at an iPad. A content-based recommender would analyze the specific details of the product, such as its functionality as a tablet. Based on these details, it might recommend a different iPad or Samsung Galaxy Tab, as these are also tablets with similar features. In contrast, an item-based collaborative filtering approach would look at purchasing patterns for users who have bought iPads. As a result, the item-based collab-

orative filtering recommender could recommend screen protectors, cases or Apple Pencils, as other users who bought iPads also bought these items. It could also recommend other Apple products like iPhones or MacBooks, or even seemingly unrelated items that are popular among iPad buyers, such as specific headphones or smart home devices.

2.1.2.3 Hybrid collaborative filtering

Wang et al (2006)[19] also found methods to unify these two collaborative filtering algorithms. According to the authors, this addresses the issues of data sparsity found in both user-based and item-based collaborative filtering, by reformulating the collaborative filtering problem within a "generative probabilistic framework". This means that the individual user-item ratings are used as indicators for predicting missing ratings. The final rating is estimated by combining predictions from three sources; predictions based on ratings of the same item by other users (user-based collaborative filtering), predictions based on different item ratings made by the same user (item-based collaborative filtering) and ratings predicted based on data from other but similar users rating other but similar items (combination of user- and item-based collaborative filtering). Like other collaborative filtering methods, however, this method still has issues. One of these is the "cold start" problem, as discussed by Camacho and Souza (2018)[20], where the recommender has difficulties making recommendations when there is insufficient data available. This arises for new users who have not rated or interacted with enough items, or for new items that have not yet been interacted with by users.

Cano et al. (2006)[21] investigated the topology of several music recommendation networks. The authors showed that despite sharing certain characteristics such as small-world properties, diverse network features emerged such as the link degree distribution among different recommendation networks. The link degree distribution refers to the distribution of the number of connections (links) that each node in the network has. In simpler terms, it describes how many connections each item has within the network. This difference is notable, as this means that the way a recommender system is

built influences the recommendations. The authors specifically note that the recommendations can be biased towards popular items in systems that are based on collaboration, such as collaborative filtering. This shows us that the architecture of a network can influence its bias towards popular items.

2.1.2.4 Model-based collaborative filtering

Model-based collaborative filtering is a technique that uses machine learning methods to create individual predictions. In the previously seen neighbourhood-based models, the predictions are specific to individual instances. In contrast, model-based methods create summarized models upfront and separate the training phase from prediction[22]. This allows for a representation of the rating matrix in a much more compact way, resulting in a higher prediction speed and thus better scalability. Several methodologies explore different machine learning models to use for this task, which show higher accuracies than neighbourhood-based models[23]. Furthermore, model-based collaborative filtering algorithms have several notable advantages over neighborhood-based approaches. Firstly, they exhibit superior space efficiency by generating significantly smaller learned models compared to the original rating matrix. This leads to reduced storage requirements, a crucial factor, especially in large-scale systems[24]. Additionally, in terms of training and prediction speed, model-based systems outperform their neighborhood counterparts. The pre-processing phase in neighborhood methods often scales exponentially with the number of users or items, whereas model-based systems streamline this process, resulting in faster training and prediction[24]. Model-based collaborative filtering can however suffer from lessened interpretability, as the created summarized model is a complex abstraction of user interactions and item characteristics, making it difficult to interpret what is driving the recommendations.[25]. Moreover, due to their capacity to model patterns in the data, these models can overfit the data. This means the algorithm can become too specific to the training data, capturing specific patterns that do not generalize well to unseen data.

Since the demand for music recommender systems is so high, as discussed in the introduction, there have been numerous more modern de-

velopments in the space of content-recommender systems, as well as music recommender systems specifically. In their research, Saini and Singh (2024)[26] propose a recommender system that combines stacked long short-term memory (LSTM) and an attention-based auto-encoder. They note that the results of the proposed method are more accurate than other approaches on the validation set, therefore theoretically increasing accuracy in real-world recommendations. This technique combines the strengths of LSTM networks in capturing sequential dependencies and the attention mechanism's ability to focus on relevant parts of the input. By using these components, the model can better learn patterns within the data. However, despite the advancements in the accuracy of recommendation algorithms, popularity bias remains a significant challenge. Most state-of-the-art systems, including those using LSTM and attention mechanisms, still tend to favor popular items due to the underlying data distributions they are trained on. Lin et al. (2024)[27] showed how dimension collapse, as the creation of a summarized model in model-based methods is also called, can exacerbate biases in data. They found that this collapse emphasizes the principal spectrum, referring to the most significant parts of the data, capturing the most significant variations in the dataset. This tends to disproportionately represent popular items, as the dimension reduction process prioritizes the most prominent features, increasing the prominence of popular items. As a result, the recommendation outputs of model-based recommendation systems become skewed towards popular items.

2.2 Evolutionary computing and genetic algorithms

Recent methods have also shown the use of evolutionary computing to solve several problems in an efficient and (near) optimal way. These algorithms are inspired by biology and natural selection. In this section, we will discuss some evolutionary computing methods, and look at the use of genetic algorithms in research on recommender systems, as well as look into one specific form of a genetic algorithm, an interactive genetic algorithm (IGA).

2.2.1 Evolutionary computing

Evolutionary computing is a branch of artificial intelligence, which encompasses a range of techniques. As previously discussed, this type of computing is inspired by principles of evolution in nature and is created to solve problems for which it is not possible to find an optimal solution or a solution within a bound of the optimum within a reasonable amount of time[28]. There exist several different techniques that fall under the umbrella of evolutionary computing. One example is Ant Colony Optimization (ACO), which mimics how ants find the shortest path to food by depositing pheromones and following paths with stronger pheromone concentrations to solve routing problems[29]. Another example is Artificial Immune Systems (AIS) which mimic the human immune system using mechanisms such as clonal selection, immune memory, and negative selection to solve pattern recognition problems, such as anomaly detection in network security[30]. As a final example, Particle Swarm Optimization (PSO) simulates birds or fish schooling behavior to find solutions to problems based on its experience and that of its neighbors[31] to find solutions to problems such as image segmentation, allowing for the extraction of features and analysis of the image data[32]. These techniques encompass a wide range of domains, showing the usefulness of evolutionary computing across a range of applications.

2.2.2 Genetic algorithms

One of the main uses of evolutionary computing is genetic algorithms (GAs). These algorithms are a form of evolutionary algorithm where populations evolve over generations. Genetic algorithms use mechanisms such as selection, crossover, and mutation to evolve solutions to problems. The basic version of the genetic algorithm is represented in figure 2.1. The algorithm starts with an initialization, which can either be random or guided by other methods such as historical data or representative sampling[33]. Then, a population of candidate solutions, often referred to as individuals, is evaluated based on a fitness function. After the evaluation, individuals are selected for reproduction based on their fitness, with fitter individuals having a higher

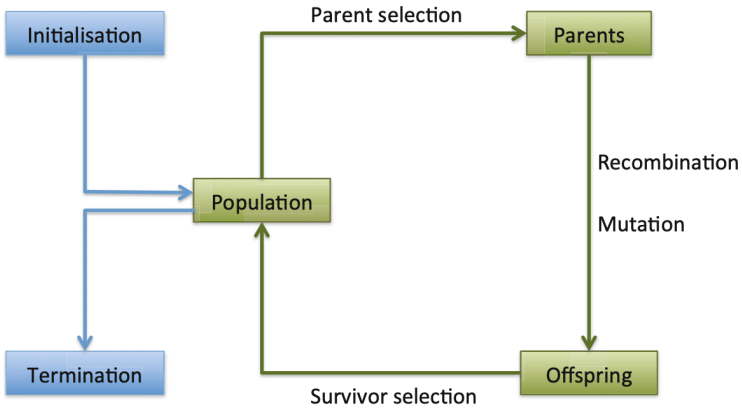


Figure 2.1: General scheme of an evolutionary algorithm[37]

chance of being selected. This selection process mimics the principle of "survival of the fittest" in natural evolution.[34] Once the individuals are selected, they undergo genetic operations to produce offspring solutions. These genetic operations help explore the solution space and introduce diversity into the population. This is done in two main ways: mutation and crossover. Crossover (or recombination) is the operation of combining two parent solutions to generate new offspring to produce potentially better solutions. Mutation is the operation of modification of individuals to help maintain diversity, and avoid getting stuck in local optima, to be able to generate more effective solutions over time.[35] These genetic operations typically have associated rates that control their frequency of application. The crossover rate determines how often crossover occurs, while the mutation rate determines how often and how much individual solutions are altered. These values are usually found through trial and error. Hong et al. (2002)[36] proposed a dynamic genetic algorithm (DGA) that utilizes multiple crossover and mutation rates. In their approach, the crossover and mutation rates are not static but change dynamically based on the evaluation results of the offspring in the subsequent generation. The newly created solutions are then evaluated, and this cycle continues until a termination condition is met, such as reaching a satisfactory solution, exceeding a predefined computational budget or reaching a set number of generations.

2.2.2.1 Fitness function

Genetic algorithms work on a population of candidate solutions. Each solution has a fitness value, that indicates its closeness to the optimal solution for the problem that the genetic algorithm is trying to solve. This fitness value is calculated using the task-specific fitness function. The solutions with a higher fitness value than others are selected for survival to the next generation. For this next generation, GAs produce better offspring by combining selected solutions, or mutating existing solutions.[37] However, genetic algorithms have been found to be sensitive to the selection of the fitness function. The performance of a GA is closely tied to the quality and design of this fitness function. As a writer on a book on genetic algorithms puts it: "the population's evolution will ruthlessly exploit all boundary conditions and subtle defects in the fitness function"[9]. The fitness function needs to be more nuanced than only classifying solutions as "good" or "bad". There should be an accurate score for the fitness across the spectrum, allowing for the distinction between a somewhat complete solution and a complete one.[9] A well-designed fitness function uses this sensitivity of the algorithm to guide towards optimal solutions for the problem. However, a poorly designed fitness function can lead to sub-optimal results. Even with well-designed fitness functions, the outcome of the resulting solution can still be over-specialized or overly adapted to the variables chosen in the fitness function. This is a risk that has been found in genetic algorithms, due to the algorithm's built-in sensitivity to the fitness evaluation[10].

2.2.2.2 Examples of genetic algorithms

The first genetic algorithms were designed to solve optimization problems. They were first discussed by J.H. Holland in 1975.[38] He introduced GAs to simulate the process of natural selection and evolution in order to find optimal solutions to problems. Since then, GAs have been applied to a wide range of fields, including engineering, economics, biology, and artificial intelligence.

One of the first practical examples of the application of GAs to real-life

problems is in the optimization of control systems in the 1970s. Here, researchers used GAs to optimize the parameters of controllers for complex systems where traditional methods struggled due to non-linearity or uncertainty in the models.[39] In the 1980s and 1990s, GAs were applied to a broader range of fields. For example, in economics, they were used to model adaptive behaviors in market participants and to solve complex financial optimization problems.[40] GAs were also employed in other artificial intelligence tasks to evolve neural networks and create adaptive learning algorithms.[41] They were also used for applications in bioinformatics for tasks such as protein structure prediction and DNA sequence alignment.[42]

Genetic algorithms have also been used to improve traditional recommender systems. They do this by re-evaluating the items the traditional system outputs, to focus on other metrics, such as diversity, novelty, coverage and serendipity[43]. For example, Alhijawi and Kilani (2020)[44] introduced the BLIGA system, which leverages a genetic algorithm to optimize recommendation lists, by evaluating entire lists of recommendations. The genetic algorithm represents each potential recommendation list as an individual within a population. These individuals are then evaluated using three distinct fitness functions. The first fitness function assesses the semantic similarity between items, ensuring that the items in the recommendation list are "semantically cohesive" and relevant. Semantic cohesion in this case refers to the idea that songs in a recommendation list should be related in terms of content, theme, or other meaningful characteristics that make them logically connected to one another. The second fitness function measures the similarity in satisfaction levels between users, aiming to match users with items that others with similar preferences have enjoyed. The third fitness function focuses on predicted ratings, selecting the list that is expected to yield the highest user satisfaction based on historical rating data. However, this methodology is still reliant on historical data, making it subjective to popularity bias within the recommended items.

Stitini et al. (2022)[45] discuss a recommender system, which is designed to mitigate a problem called the "over-specialisation problem". This is a problem where items that are recommended to a user are too similar to

their already existing profile, and therefore lack novelty. This is related but different from popularity bias, as in popularity bias, the recommender system tends to recommend popular items regardless of the user profile, while over-specialization focuses on items with strong correlations to a user's interaction history. Both result in a lack of diverse recommendations, but the origin is different. In their method, Stitini et al. use a GA to generate new recommendations for users based on their preferences and interactions with items. The genetic algorithm in their system focuses on optimizing the recommendation process based on existing user-item interactions and preferences. While the genetic algorithm approach proposed aims to mitigate the over-specialization problem by diversifying recommendations, it does not directly address the popularity bias. Since the genetic algorithm used in their system primarily focuses on user preferences and item interactions, items with strong correlations to the user's history can still be popular. Popular items are often interacted with by many users, increasing the chance of correlation with a user's profile, and therefore also increasing the chance that the item gets recommended.

2.2.3 Selection mechanisms for genetic algorithms

To determine which individuals continue on to the next generation, genetic algorithms can use several techniques for selection. These mechanisms introduce pressure to select individuals with higher fitness, which is how they mimic the process of natural selection.

Roulette wheel selection is built on the idea of a roulette wheel where each slice's size is proportional to an individual's fitness. A higher fitness gets a larger slice, giving it a higher chance of being selected for reproduction.[46]

In truncation selection, the candidate solutions are ordered by fitness, and some proportion of the fittest individuals are selected and reproduced. The higher the proportion, the more items are selected for reproduction.[47]

In tournament selection, a small group of individuals (e.g., 2 or 3) is chosen randomly, and the one with the highest fitness within that group becomes a parent. This process is repeated to select multiple parents.[48]

Rank-based selection has individuals ranked based on their fitness, with the best individuals receiving a higher rank. Selection probability is then assigned based on this ranking, with higher-ranked individuals having a greater chance of being chosen.[46]

Steady-state selection removes a small number of the least fit individuals in each generation, and these are replaced with new offspring from higher-fitness parents. The rest of the population remains unchanged.[46]

These selection mechanisms have been evaluated by researchers. For instance, Pandey et al. (2016)[49] compared roulette wheel selection and rank-based selection, and found that the rank-based method outperformed the roulette wheel in terms of the total number of generations required to reach the optimum. However, rank-based selection was found to be faster, more robust, and more certain towards the optimum. In the same study, tournament selection was also compared with roulette wheel selection, and it was found that the tournament selection method was superior as it provides better exploration and exploitation balance in the search space compared to roulette wheel selection. It is however important to note that the selection pressure, if not selected appropriately, can lead to slow convergence rate and premature convergence. Therefore, the choice of selection method needs to be made with consideration of the problem at hand.

2.2.4 Interactive genetic algorithms

For recommender systems, one option on how to use genetic algorithms is an algorithm that responds to user preferences. This style of genetic algorithm is called an interactive genetic algorithm (IGA)[50]. These algorithms are genetic algorithms where the fitness values are calculated based on the evaluations of users according to their own preferences. This allows users more freedom in their exploration of the available items, granting users the ability to discover new items they enjoy and get more. Moreover, this solves the problem of genetic algorithms being highly sensitive to the parameters that calculate fitness.

Kim et al.[51] (2010) used interactive genetic algorithms to recommend

music. The proposed algorithm lets users evaluate the fitness value of each music track, which is done in the way of assigning their rating scores according to subjective user preferences. This user evaluation data is used to recommend new items. The authors show that over time, the scores for each generation generally increase. While this method does incorporate user feedback, it's important to note that mitigating popularity bias wasn't a goal in this research, and therefore remains a gap in current knowledge. This research used BLX-α crossover. BLX-α crossover is a method for crossover where each item feature for a child is chosen as an interval between the two parents, with an α that determines how far outside the two parents' features the interval can be chosen. However, only using BLX-α crossover can lead to limited exploration, generating new recommendations that are only marginally improved, especially if the initial population has low-rated initial selections. This can result in slow convergence, potentially causing user frustration if satisfactory recommendations are not quickly found. Moreover, this research did not incorporate the common genetic operation of mutation. Developing a method for genetic mutation within an interactive genetic algorithm is therefore a potential area for future research. The reasoning given for this omission is that mutation causes deviation from the "common pattern discovered by the evolution process", and therefore it should be omitted. However, this reasoning overlooks the potential benefits of mutation in search space exploration within the recommendation space and enhancing diversity within the recommendation. Mutation introduces variations that can help escape local optima, where the algorithm might otherwise become stuck on a narrow set of similar items. By allowing for controlled deviations, mutation can discover new and potentially more suitable tracks that align with user preferences not yet fully captured by the existing recommendations.

The authors of Kim et al. (2010) expanded their system in subsequent works, including Kim et al. (2011)[52], Kim and Ahn (2012)[53], and Kim et al. (2014)[54], by incorporating additional music features and data grouping techniques to improve the quality and speed of their recommendations[55]. However, these improvements still do not address the possibility of popu-

larity bias in the recommendation, nor the lack of a mutation mechanism.

Similarly, the use of IGA by Kant and Bharadwaj (2013)[56], uses reclusive methods (RMs) and interactive genetic algorithms to recommend items. Their approach begins by providing users with a set of initially recommended items, ranked based on predicted relevance based on earlier user ratings. The user then evaluates these items by rating their relevance, and this feedback is used to update similarity scores and refine recommendations. The IGA evolves these recommendations through iterative processes, applying genetic operators (selection and crossover) to improve item relevance over generations. However, their research also did not explore the potential benefits of incorporating mutation operators, which could introduce greater diversity and mitigate issues such as avoiding local optima. The authors cite the same reason as Kim et al.(2010)[51] as to why mutation is unsuitable for IGAs, namely that it causes the recommendations to deviate from the common pattern discovered by the evolution process. Moreover, they also use the same selection method, as well as the same crossover method as Kim et al (2010)[51]. Their approach adds a fuzzy theoretic approach based on reclusive methods to match a newly created crossover track to an existing track in the dataset.

In a more recent paper, Saito and Sato-Shimokawara (2023)[57] also propose a music recommender system that utilizes an interactive genetic algorithm to tailor music recommendations based on user evaluations. They base their method on Kim et al. (2010)[51]. They note that their method also employs BLX-α crossover as an alternative to mutation, which impacts the exploration and convergence dynamics of the algorithm. However, similarly to Kim et al. (2010)[51], the use of BLX-α crossover and no mutation can lead to limited exploration, generating new recommendations that are only marginally improved. Additionally, without mutation, the algorithm risks falling into a local optimum of "least disliked" items, rather than discovering truly liked music. The lack of mutation may also lead to reduced diversity in recommendations, as variations are primarily constrained to the initial set of items. The difference between their method and Kim et al. (2010)[51] is that Saito and Sato-Shimokawara's approach incorporates multiple mu-

sic selection criteria, which considers changes in music selection criteria of users and utilizes them to recommend the most suitable music for users. Moreover, it uses a more diverse set of features from Spotify, which includes valence, energy, popularity and release date.

Future research could address these issues by incorporating mutation to enhance exploration and diversity, potentially improving the algorithm's ability to escape local optima and provide more diverse and satisfying recommendations.

The common theme in these previous works on interactive genetic algorithms in recommendation is therefore that none of them take popularity bias into account specifically, or do not address it. Moreover, no approaches have found suitable mutation operations for evolving new generations, or have deemed them irrelevant. This therefore remains a gap in the current knowledge.

2.3 Data sources for music recommender systems

A music recommender system can use multiple data sources, used to deduce preferences and generate music recommendations for users. Different data sources can be used for different types of recommender systems. This section will look at what data sources have been used in the literature, split up into three main categories; content-based data, user behavior data and contextual data.

2.3.0.1 Content-based data

Content-based data is a data source generated using the data from the music itself. One example of this is the use of audio features.[58] Using various methods, low-level features like tempo and pitch, as well as high-level features such as genre and energy are extracted from the music itself.[59] Moreover, metadata of the music can be used. This can include information such as artist details, album release year, and other song attributes.[60]

2.3.0.2 Contextual data

Contextual data as a source for music recommender systems refers to the information that is gathered about the user's environment or situation. This information can then be used to make more relevant recommendations, based on information such as their location, time of day, weather, and physical activity.[61] This type of data can be acquired explicitly, implicitly, or inferred through methods such as machine learning. For example, Lee et al. (2017) developed a smartphone-based system that uses machine learning to recognize human activity and then makes music recommendations based on this contextual information.[62] Contextual data can enhance the relevance of recommendations provided by a music recommender system, by considering the context in which a user consumes the music.

2.3.0.3 User behaviour data

User behavior data is another data source to infer a user's listening preferences from within music recommender systems. Music recommendation systems can tailor their recommendations based on user input, using both individual user preferences and the collective listening habits of similar users, as is the case for collaborative filtering recommenders. This can be done through explicit feedback, such as ratings, reviews and liking[63], or implicit feedback, such as listening history, skipped tracks, and time spent on a particular song or artist.[63] Recent methods have focused mostly on implicit feedback, as this is easier to obtain on a large scale, with users having to take no actions themselves to generate the feedback[64][65]. However, explicit feedback also still has its place. Music can be consumed as off-screen content, meaning the user can be doing something else with their time while listening to music. This means that an implicit metric such as uninterrupted consumption does not always have to correlate with enjoyment of the content. Therefore, explicit feedback remains valuable in capturing user preferences and improving the accuracy of music recommender systems. In interactive genetic algorithms, the feedback given by users is inherently explicit, as users give feedback on the previous generation. This

means the algorithm can infer what type of music the user wants in that specific run, meaning the user can take the contextual information into account that they deem to be relevant, such as selecting energetic music to create recommendations for exercise. The IGA can then use the indicated preferences to recommend music.

2.4 Popularity bias and the long-tail problem

Because of the quantity of music being produced now and the amount of music that has already been produced in the past, as a listener, you will never be able to listen to all music that has ever been made. This shows the value of music recommendation systems, where they can help with this information overload. Music as a medium is inherently social, as shown by Woodruff and Cross (2009)[66], and some music is more popular than others. This popularity causes even more popularity through means such as word-of-mouth, social media sharing and top charts. This leads to a mainstream that contains the most popular genres. Because of this, there will always be music that is listened to more than other music. This mainstreamness is hard to define, but several researchers have created measures to define what it means for music to be mainstream. This mainstreamness can then help analyze the items that a recommender system outputs, and draw conclusions from them.

Bauer and Schedl[67] describe quantitative measures to describe the proximity of a user's music preference to the music mainstream. They also describe the difference between a global music mainstream and a country-specific mainstream and define measurements at these two levels. This results in a framework of six measures to quantify what the mainstream is. The measures rely on two metrics for artist popularity; artist playcounts (APC) and artist listener counts (ALC). The measures are further divided into distribution-based and rank-based methods, which compare user preferences based on either the distribution of artist popularity or the rank-ordering of artist preferences, respectively. This results in 6 total measures, with 3 country-specific measures, and 3 global measures. The authors also

analyzed the difference between countries in terms of their mainstreamness, outliers in country-specific popularity and the difference between countries listening preferences related to popular music artists. Additionally, the authors show the applicability of their research to improve music recommendation systems, which they do by tailoring preferences for a user by using the six mainstreamness measures.

Müllner (2019)[68] used unsupervised clustering and classification to identify non-mainstream music styles. Subsequently, these styles were used to link users to a certain style, and therefore obtain different user groups of styles. They found that there are distinct preferences between user groups, as well as different demographics between user groups. The author also notes that there were distinct types of non-mainstream users, which they name, such as Festival or Complex listeners. Certain groups of users were found to listen to certain genres more than others, with festival listeners listening to punk and hard rock, and relax listeners listening to ambient and post-rock genres. This shows us that there are different types of music listeners, who have distinct preferences, and want to listen to music that aligns with their tastes.

In music recommendation algorithms, the more popular an item is, the more people are listening to it, reviewing, or rating it, and generating data on it. This, in turn, results in more popular music having more occurrences in music-listening datasets, which are used for the creation of algorithms. This fact contributes to algorithms favoring the recommendation of the most popular items, while less popular items receive fewer recommendations. This discrepancy between more and less popular items is also called the "popularity bias". Items at the most popular level occur orders of magnitude more often than items that are more niche. This is also called the "long tail problem". This term was coined by Chris Anderson in an article in 2004[69]. When depicting the occurrence of items in a chart, sorted by the amount of occurrences, the most popular items have extremely high occurrences, while less popular items have exponentially fewer occurrences, leading to a figure that is large in the beginning, and drops off quickly. In Figure 2.2, a representation of the long-tail, shown in logarithmic scale, can be seen

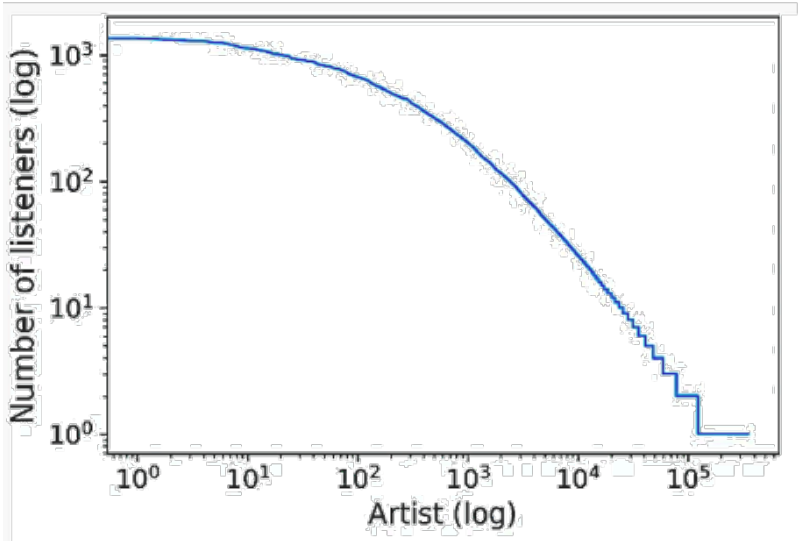


Figure 2.2: The long-tail problem illustrated in the LFM-1b dataset, with the artists ranked by the number of listeners on the x-axis, and the number of listeners for that artist on the y-axis (scaled logarithmically)[70]

based on a widely used music listening dataset, the LFM-1b dataset[70].

As can be seen, the most popular artists, also have the most listeners by far, while a few less popular artists have significantly fewer listeners. It has been found that listeners prefer variety and a mix of familiar and new music[71], implying that the recommendation algorithms need to give recommendations that match the users' tastes, while also introducing them to novel music.

The popularity bias in music recommender systems is a widely studied topic, with researchers focusing on causes, as well as mitigation techniques. People listen to music for all kinds of reasons, and music can carry an emotional load with it. This means that simply using collaborative filtering can be lacking, as this does not include this kind of data. This is also something that content-based models are better at addressing by focusing on features rather than musical history.

This leads to the paper by Song et al.[72] They describe how more user-centric approaches have been getting increasing attention, which are context-based and emotion-based models. In emotion-based models, the user's current emotional state is queried, and this is used for music recommendation. One example is Musicovery, which uses a 2D valence-arousal emotion

model, where the users locate their own perceived emotion in 2D space. Here, the users are therefore directly influencing the generation of their recommendations, by locating their own perceived emotions. The authors also discuss context-based information retrieval, which is recommendation based on public opinions, based on document mining to filter out important information to support problems like artist similarity, genre classification, emotion detection and semantic space. This work can offer potential ideas to address the popularity bias by considering user input to shape recommendations.

Regression towards the mean is a statistical phenomenon that describes the tendency of extreme values to move closer toward the average over time with subsequent measurements.[73] For music, this therefore means that music listeners who listen to more music from within the long tail, tend to drift towards more popular items. This tendency can be both strengthened by music recommender systems, by recommending more popular music and falling into the popularity bias, or the tendency can be weakened, by ensuring that recommendations are diverse, and serendipitous discoveries can be made. A study by Schedl et al.[74] warns of these exact effects in the context of music recommendation systems. The authors emphasize that while music recommender systems can effectively recommend songs based on historical behavior, there's a risk of homogeneity, where the recommender system recommends music that aligns with what the user or a group as a whole has already heard. This limits the diversity of suggestions and can cause users to not be exposed to novel songs and artists, therefore causing popularity bias.

2.5 Impact of the popularity bias

When the consumption of music through the internet was first getting started, many expected the creation of a more equitable marketplace for music. One of these is the author of the article that first coined the term "Long tail", Chris Anderson[69]. In retail stores, shelf spaces are limited and therefore a limited amount of physical music can be made available for consumption. In

online music stores and streaming services, there is almost unlimited space for music, and therefore these would create a more equitable marketplace for music.

However, it has been noted that this might be less the case than first hoped. Music magazine Rolling Stone noted that The top 1% of artists account for 90% of all music streams on Spotify, and the top 10% of artists for the top 99.4% of all music streams. When compared to physical album sales in the same period, the top 1% of all artists accounted for 54% of all physical album sales[75]. This means that currently, streaming is actually less equitable than physical album sales. There are many reasons that can be given for this inequity in streaming services, one of which is the subjectability of music recommender systems to the popularity bias. This has impact on both users and artists.

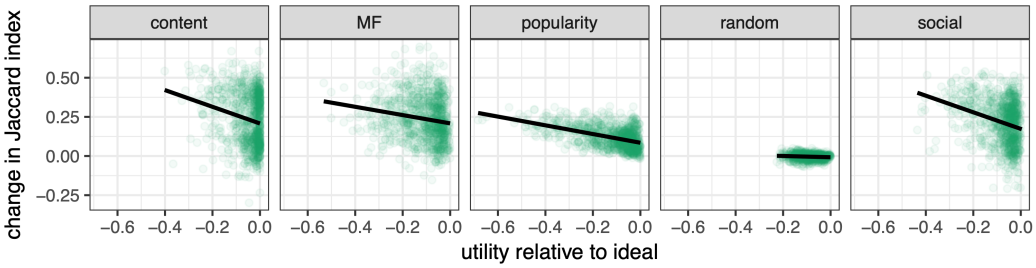


Figure 2.3: In repeated training, several algorithms are shown, highlighting the trend that users who experience losses in utility also have higher homogenization, as presented by Chaney et al.[76]

The impact of the popularity bias is even greater when the data from users' listening behavior that already has been exposed to algorithmic recommendations by recommender systems is used. Recommendation systems trained or evaluated on this "tainted" data can fall into a feedback loop, where algorithms are trained on data that has algorithmic bias, from which the data is again used to train a new algorithm. This increases homogeneity for the resulting recommendations. Using simulations, Chaney et al. (2018)[76] showed that this was the case, and warned that this "algorithmic confounding" does not correspond to increased utility for users. The authors also showed that these losses to utility are not distributed equally and that users whose true preferences are not captured well by user preferences,

i.e. users already less well served by the recommendation algorithm, experience lesser improvements and even decreases in utility when the homogenization occurs as a result from the feedback loop. This effect can be seen across algorithms, as shown in Figure 2.3, where the change in the Jaccard index (as an index for user behavior) is compared to the utility, as relative to the ideal utility. The Jaccard index measures the similarity between two sets. These sets here are user behavior patterns, indicating how homogenous user interactions become. Higher values of the Jaccard index suggest increased homogenization of user behavior.

This reinforcement effect could enhance biases already found in the original recommendations, which could create a feedback loop where these biases are amplified over time. This then could limit recommendation diversity and potentially create filter bubbles.

This reinforcement effect can possibly be mitigated using genetic algorithms. In IGAs, unlike in traditional recommender systems, the user is in direct control of the selection process. This user feedback then iteratively refines the recommendations, where user-liked items are prioritized. This iterative process allows the user to "steer" the recommendations away from unwanted effects, allowing for recommendations that are more in line with the preferences of the user.

2.5.1 Impact on artists

The popularity bias can have a large impact on artists. For artists, outside of a select few extremely popular ones, it is challenging to gain visibility and recognition. The unequal distribution of streams found by Blake (2021)[75], as discussed in section 2.5, means that less popular artists face difficulties breaking through and reaching a larger audience. The popularity bias found in recommender systems can worsen these difficulties for less popular artists, leading to economic consequences for artists who rely on music as their primary source of income.

Additionally, popularity bias can then influence the type of music artists produce. In a system where only a small number of popular artists domi-

nate streams, there may be motivation for artists to create music that is better suited to mainstream tastes to enhance their chances of visibility. This could cause a decline in musical creativity, as artists feel compelled to produce content that aligns with popular trends to reach larger audiences.

2.5.2 Impact on users

The popularity bias that is observed in recommendation systems can affect users in many different ways. A minority of artists has been found to attract the majority of streams.[75] This can result in users having a harder time finding artists and genres that align with their specific tastes. Users might turn to recommendation systems to find music that aligns with their tastes. However, the recommendation algorithms that are built into most streaming services tend to make this issue worse, as they are susceptible to popularity bias. This reinforces the cycle, which could cause musical tastes to become more similar and foster a reduced exploration of more niche genres.

The overall user experience is shaped by the concentration of streams on a select few extremely popular artists. This could limit the variety and more positive experience for users that a fairer recommendation system could offer.

Abdollahpouri et al.[11] studied the popularity bias and its inherent unfairness in their work. Their work focused on the user-side of the popularity bias, and how the popularity bias causes the recommendations of a music recommender system to deviate from what the users expect to get from their recommender system. They defined three distinct groups of users in accordance with their interest in popular items and showed the impact of the popularity bias on users of each group. The results of their method showed that many recommendation algorithms would recommend popular items, even if a user is interested in the long-tail and non-popular items, showing the strong tendency for recommendation algorithms to fall into the popularity bias. This is therefore something a recommendation system should address.

2.6 Popularity bias metrics

To define to what degree the output of a music recommendation algorithm could be subject to popularity bias, there are several metrics.

- **Delta Metrics** Lesota et al. (2021)[77] talk about Delta Metrics, which are metrics that show the percentage difference between the mean, median, variance, skewness, or kurtosis of the popularity distributions of the user's listening history and the recommendation list. However, when there is no previous user history to base this metric on, this metric is not suitable.
- **User Popularity Deviation** This metric was proposed by Abdollahpouri et al. (2021)[78], to quantify the deviations of the recommendations' popularity distributions from that of the user's listening history. This metric reflects the extent of miscalibration among users and the recommendations, serving as an indicator of how well recommendations align with the interest in popular items for users. A perfect value for this metric would therefore indicate that the recommendations mirror the users' preferences perfectly. When users submit their preferred popularity, it inherently aligns the recommendations with their popularity preferences, rendering this metric less critical. This user-submitted popularity constraint ensures that the recommended items correspond to the users' preferred popularity, contributing to fairer recommendations. Similarly to Delta Metrics, when there is no previous user history to base the metrics on, this metric cannot be used.
- **Average Percentage of Long Tail Items (APLT)** This metric, also used by Abdollahpouri et al. (2019)[79], measures the average percentage of the amount of long-tail items in the recommended list. APLT is calculated by taking the average fraction of long-tail items in each user's recommendation list and then averaging this fraction across all users. A higher APLT value indicates that the recommendation system is promoting higher music diversity by including more long-tail items,

thus mitigating popularity bias more.

- **Intra-List Similarity (ILS) Metric** Ziegler et al. (2005)[80] proposed the Intra-List Similarity (ILS) metric to assess the diversity of items within a recommendation list. The ILS metric measures the average pairwise similarity between all items in a user's recommendation list, where a lower ILS value indicates higher diversity.

The suitable metrics can be used to evaluate the performance of algorithms in their ability to recommend items from the long tail.

2.7 Investigating popularity bias in algorithms

Different algorithms have shown varying levels of subjectability to the popularity bias. For example, MultVAE[81] has been shown to yield high levels of popularity bias in its recommendations, while ItemKNN[18] shows relatively low levels of popularity bias in its recommendations[77].

Kowald et al. (2020)[70] investigated the popularity bias for six methods, including 3 baselines (Random, MostPopular and UserItemAvg and 3 to-be-evaluated methods, namely UserKNN, UserKNNAvg and Non-Negative Matrix Factorization (NMF). They found the popularity bias to differ between the algorithms, with the popularity bias being not as strong in the case of NMF, and being the strongest for UserKNN for all users. These results show us that the popularity bias can vary throughout different algorithms' recommendations, indicating the need for research into an algorithm that is less prone to popularity bias.

2.8 Mitigation of the popularity bias

As discussed, the popularity bias is a widespread issue within music recommender systems. Because of the prevalence of this issue, a lot of music recommender systems research has been done on the addressing of the issue.

2.8.1 Traditional popularity bias mitigation

Karboua et al. (2022)[82] did an empirical analysis of different mitigation techniques, to provide an overview of the state-of-the-art techniques for popularity bias mitigation. The authors discuss in-processing techniques, as well as post-processing techniques. These methods aim to address the popularity bias by modifying the recommender system's algorithm, or post-processing the output of the recommender to contain less popularity bias.

2.8.1.1 In-processing techniques

In popularity bias mitigation, in-processing techniques are techniques that are added to the recommender systems themselves, in the form of penalties or other steering mechanisms that have the final goal of helping the algorithm recommend more diverse items.

Karboua et al. (2022)[82] discuss these in-processing techniques. One example given is variational autoencoders (VAE) and adversarial training. VAEs are Multi-Layer Perceptron-based generative models that are a method for building Collaborative Filtering recommenders. Adversarial training is a technique where an adversary network creates a penalty for a base recommender model that steers the model in a certain direction. This method proposed by Borges et al.[83], proposes adding a penalty constraint to the decoder to boost the visibility of unpopular items. This penalty term has the effect of lowering the score of the most popular of all items while keeping the niche items at their initial score.

Krishnan et al.[84] propose a method where an adversary network learns the implicit relation structure of items through feedback data and, using this relation structure, correlates niche item recommendations of the base recommender with popular items in the user's history. The base recommender model is concurrently trained with the adversary network to replicate these associations while avoiding the adversarial penalty.

2.8.1.2 Post-processing techniques

Post-processing techniques for popularity bias mitigation take the output of a recommendation algorithm and use several methods to diversify the results.

One option for post-processing popularity bias mitigation is diversification. Antikacioglu and Ravi (2017)[85] discuss this method in their work. In their approach, they applied diversification after the recommendation algorithm had generated its results. This post-processing step involved optimizing the diversity and rating quality of the recommendations by selecting a subgraph from a pool of potential recommendations. By formulating the recommendation system design as a subgraph selection problem and using minimum-cost network flow methods, they were able to optimize for diversity while maintaining high rating quality. This post-processing technique allowed them to enhance the diversity of recommendations without altering the initial recommendation generation process, making it a valuable strategy for addressing popularity bias in recommendation systems.

Karboua et al.[82] also discuss post-processing techniques, one of which is the calibrated popularity, as proposed by Abdollahpouri et al. (2020)[86] Here, the distributional discrepancies in the groups to which items that are recommended belong are measured. The idea of calibrated popularity is that the recommended items should have the same percentage of popular and less-popular items as the to-recommend-to user has in their history. This is done by weighting relevance and calibration of a base recommended list and then taking the maximum. Multiple studies have been done on the use of calibrated popularity, with another study by Abdollahpouri et al. (2021)[78] using the technique to research the effect the popularity bias has affected users with various levels of interest regarding popular items. They found that, in addition to affecting all recommendations, popularity bias tends to impact users differently. Users with a lesser interest in popular items are more affected by this bias, and their recommendations deviate from the range of popularity levels they expect to receive.

Both in-processing and post-processing techniques have made signifi-

cant strides in the mitigation of popularity bias. However, they often rely on static data, and pre-determined metrics, which can limit their flexibility in changing user preferences. They also do not use the potential of real-time user interaction to adjust the recommendations. Abdollahpouri et al. (2021)[78] also discuss limitations of existing popularity bias mitigation techniques. They discuss how many of the existing metrics for evaluating popularity bias are mainly item-centered, and they ignore that different users have different degrees of interest in popular items. The calibrated popularity post-processing technique does account for this, taking the user's preference for popularity in their history into account. However, inferring the user's preference from their listening history could be a pitfall, as their listening history could contain more popular items than the user would actually like to listen to.

As discussed earlier, the majority of music streams go to a small amount of extremely popular artists[75]. However, some users have preferences for music that is less popular, as shown by Abdollahpouri et al. (2019)[11], and they might not be able to find music that suits their tastes. Therefore, using an algorithm where user feedback shapes the recommendations in an iterative cycle could help these groups of users find recommendations that are closer to their taste.

2.8.2 Popularity bias mitigation using Interactive Genetic Algorithms

Popularity bias mitigation can also be done through the use of interactive genetic algorithms. In comparison to traditional methods based on historical data, IGAs engage users directly in the recommendation process through their feedback. This can be useful for popularity bias mitigation, as users are able to actively seek out or avoid popular items. The recommendation in IGAs is mostly based on the content of the items, therefore avoiding any popularity bias that might come from collaborative data.

Existing IGA implementations, such as Kim et al. (2010)[51], Kant and Bharadwaj (2013)[56] and Saito and[57] lack a crucial element: the mutation

operator. Mutation introduces random variations within the recommendations, which can help escape local optima where recommendations become too focused on a set of similar items. By integrating mutation into the IGA, the recommender system could maintain diversity and adapt more effectively to unique user preferences, overcoming the popularity bias more robustly, as well as improving the chances for serendipitous discoveries.

Liang and Willemsen (2019)[87] focus on how recommender systems can support users in developing new preferences outside of their current tastes. Their approach explores how users can use recommendation algorithms to explore more niche music to expand their musical horizons. IGAs can be a good approach for discovery, as users can choose their own path through the recommendations, actively engaging in their own recommendations. Moreover, because of the feedback obtained from users in the IGA and the content-based recommendation style, these algorithms do not suffer from the same cold start problem as other recommendation systems do, as discussed in Camacho and Souza (2018)[20].

The proposed research therefore aims to explore the integration of mutation within an IGA for music recommendation. Moreover, we intend to include dynamic crossover and mutation rates, as discussed by Hong et al. (2002)[36]. This will enable user feedback to influence recommendations in real-time, enhancing both the diversity and personalization of recommendations as compared to other IGA methodologies, while directly addressing the limitations of traditional methodologies, such as the cold start problem.

3. Methodology and motivation

3.1 Motivation

Music recommender systems have revolutionized music consumption, offering personalized suggestions based on listening history and user preferences. However, one pervasive issue found within these systems is popularity bias, where a recommender system will tend to recommend the most popular items within the data. Popularity bias within recommender systems poses a significant challenge, as it can lead to a concentration around a popular mainstream, instead of aligning better with items that might better suit individual users. This bias perpetuates a popular mainstream obtained from training data, and therefore limits the exposure to varied music, creating market homogenization.

The motivation behind exploring popularity bias stems from the need to address the challenges associated with popularity bias within recommender systems. As we have seen, music consumption habits have evolved over time and the usage of recommender systems has grown. This growth in usage must also be paired with research into the mitigation of negative impacts of these systems.

The negative impact of popularity bias comes from different sources. For one, it narrows down the music users are exposed to, which might limit their chances to discover new genres and styles. Moreover, new artists can struggle to gain recognition and audience exposure, possibly hindering their creative and financial success. The problem of popularity bias and its implications and impacts have been studied widely, with several metrics and mitigation techniques having been studied. One metric, Average Percentage of Long Tail Items (APLT) as used by Abdollahpouri et al. (2019)[79] provides a valuable tool for quantifying this bias in recommendations. The

formula for the APLT can be found in equation 3.1. As mentioned earlier, this metric is a tool to measure the prevalence of popularity bias in recommendation systems. It specifically focuses on the proportion of long-tail items that appear in user recommendations. By analyzing the APLT, we can understand how well a recommendation system avoids popularity bias. A higher APLT indicates recommendations that are higher in long-tail items, therefore including less popular items.

$$APLT = \frac{|L \cap \Gamma|}{|\Gamma|} \tag{3.1}$$

where:

- (L) is the set of all long-tail items,
- (Γ) is the set of items recommended to the user,
- (L $\cap\Gamma$) represents the long-tail items in the recommendation set.

The APLT metric, therefore, provides a measure of how often long-tail items appear in the recommendations for the target user. To adapt this metric to our structure, the recommended long-tail items for each run will be calculated by calculating a popularity threshold. For each generation within a run, we determine the intersection of the recommended items and the long-tail set identified for that run. The precision is then calculated by computing the fraction of recommended long-tail items out of the total long-tail items for each generation. Finally, these results are aggregated to calculate the mean APLT across all runs.

Interactive genetic algorithms are uniquely positioned to address the issue of popularity bias. By having users express their preferences through an IGA, they can ensure that the recommendations are tailored to their specific tastes. The angle of interactive genetic algorithms for popularity bias mitigation has not been explored in the literature, and is, therefore, an interesting research avenue. Current methodologies, such as Kim et al. (2010)[51] focus on the improvement of feedback scores, which is something that our

system will also focus on. The ability of the system to mitigate popularity bias is enabled by the iterative interaction that is inherent to the interactive genetic algorithm, allowing the user to select items that are to their tastes. Moreover, research into IGAs has not explored the possibilities of mutation to enhance the quality and diversity of the recommendations. In previous methodologies, this has been omitted. The lack of a mutation mechanism could restrict the exploration capabilities of the algorithm, and we expect this will lead to more predictable, lower quality recommendations. When there is no mutation, the algorithm might fall into a local optimum of "least disliked" songs, instead of actually finding music the user actually enjoys. Moreover, by incorporating mutation into the methodology of the IGA, we believe we can enhance the chance of serendipitous discoveries. Mutation can also help overcome the convergence to popular items, helping counter the popularity bias that persists in many recommendation algorithms.

Another avenue where an IGA could be relevant and useful is in the exploration of new music genres. The authors of the paper "Personalized Recommendations for Music Genre Exploration" (Liang and Willemsen, 2019)[87], talk about how users might have the goal to develop new preferences away from their original ones. The interactive genetic algorithm would enable users to discover new music they might enjoy, by honing in on preferences they selected and expanding their view of that new avenue, enhancing the exploration of new genres.

3.2 Research questions

There are several unanswered questions that our methodology will seek to address. The main research question is as follows:

Does the use of mutation in an interactive genetic algorithm improve user music recommendations and address popularity bias?

This research question is supported by several sub-questions. These are as follows:

1. How effective is the interactive genetic algorithm recommender with

mutation in increasing feedback scores over generations in general?

2. How effective is the interactive genetic algorithm recommender with mutation in increasing the feedback scores for users with niche preferences?
3. How does the use of an interactive genetic algorithm with mutation impact the popularity bias of recommended music, as expressed in the Average Popularity of Long-Tail Items (APLT)?

3.3 Methodology

This research will be split up into two parts. The first part will be focused on the creation of the interactive genetic algorithm. This involves the integration of a rating system to determine fitness, the selection of songs for the next generation, as well as the integration of mutation and crossover. The second part will involve the testing of this algorithm, through the utilization of simulated user interactions and comparisons to other interactive genetic algorithms in the data.

3.3.1 Dataset

The dataset used for this research is a modified version of the LFM-1b dataset, created by Schedl (2016)[88]. This is a dataset that has been used extensively for music recommendation research, because of its scale and diversity. The LFM-1b dataset contains user listening histories gathered from Last.fm, a popular music streaming service. It includes a large number of tracks, which is useful for our cause of recommending music.

The user listening history that this dataset includes, can be valuable for developing collaborative filtering algorithms. However, using these events directly in our method, for the purpose of recommending music, could influence the IGA to choose items with more occurrences and therefore could lead to popularity bias in recommendations. This is because popular items are more likely to be present in a larger number of user histories, creating a higher probability of overlap between popular songs and a large number

of songs in the data. Because popular songs have a higher chance of overlap with other tracks, algorithms can recommend popular songs more often, even if those items might not align perfectly with a specific user's individual taste. Therefore, to mitigate popularity bias and ensure a diverse range of recommendations, we will incorporate audio features and genre information into our recommendation model. Audio features are quantitative measures that describe various characteristics of a song, such as danceability, energy, tempo, and valence. Incorporating these extra features means we have to expand the original LFM-1b dataset. This is done in two ways, which will be discussed shortly. Furthermore, occurrences within the LFM-1b dataset were used to quantify popularity. This approach involved calculating the popularity of each song based on its stream frequency relative to the maximum streams observed for a single song in the dataset. By normalizing the number of streams a song received against the highest stream count recorded across the entire dataset, we obtained a measure that reflects the relative popularity of each track within the context of the dataset's user listening histories.

In addition to the original LFM-1b dataset, which was used as the base of the dataset through the use of its tracks, an extension for this dataset was used, the LFM-1b User Genre Profile (UGP) dataset. This dataset builds upon the foundation of the LFM-1b dataset by incorporating additional information related to music genres. Specifically, it includes genre annotations for artists, encompassing both specific and broad genre categories. This dataset was created by Schedl and Ferwerda (2017)[89].

The LFM-1b UGP genres are derived from two sources, Allmusic and Freebase. Allmusic is a comprehensive online music database that provides information about artists, albums, songs, and genres. It is known for its genre categorization system, which covers a wide range of musical styles.[90] Freebase, on the other hand, is a large collaborative knowledge base that contains structured data on various topics, including music.[91] It offers a more granular and diverse set of genre labels compared to Allmusic, including niche genres and subgenres. In this dataset, only the broad genres were used. Through exploration of the dataset, it was found that the user-

annotated Freebase genres were more detailed in more popular artists, and not all artists had associated genres. Even though the methodology will not make use of these genres to recommend music, only the Allmusic genres were added, to later make testing of the algorithm more fair.

Moreover, the Spotify API was used to expand the dataset. The Spotify API is a set of tools and protocols that allow developers to interact with Spotify's vast music catalog and analyses. It provides access to information such as track details, artist information, audio features, and user playlists. By leveraging the Spotify API, we can enrich our dataset with additional metadata and features that are not available in the original dataset. The Spotify API was used to retrieve audio features for each track in the dataset. These features provide valuable insights into the musical style and mood of a track, and they can be used to enhance the accuracy and personalization of music recommendations. Features include danceability, valence and energy, which are algorithmically determined by Spotify.[92]

The process of expanding the dataset involved retrieving the Spotify track ID for each track in the LFM-1b dataset. This was done by querying the Spotify API using the track name and artist name as search terms. Once the Spotify track ID was obtained, we could then use the API to fetch the corresponding audio features in batches. Because of limitations in computing power, the number of tracks in the final was limited to 20,000. In table 3.1, the attributes per item of the dataset can be seen, and each attribute is described. Moreover, Figure 3.1 shows a comprehensive overview of the dataset creation process.

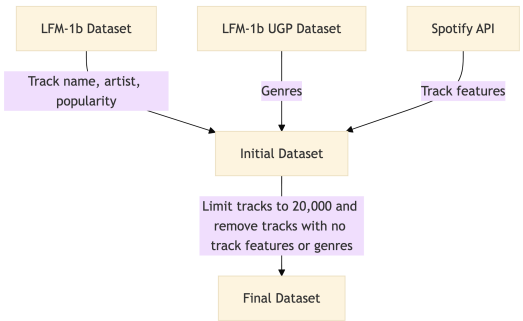


Figure 3.1: Comprehensive Overview of Dataset Creation

Table 3.1: Attributes of an item in the dataset

Name	Description
Artist Information	artist_id: Unique identifier in LFM-1b dataset (e.g., 8) artist: Name (e.g., Amon Amarth) genres: Associated genres from LFM-1b UGP dataset (e.g., 'heavy metal')
Song Information	song_title: Title (e.g., Where Silent Gods Stand Guard) track_id: Unique identifier in LFM-1b dataset (e.g., 3111) duration_ms: Length of the song in milliseconds time_signature: Estimated number of beats per measure (range: 3 to 7) popularity: Popularity of the track in LFM-1b dataset, based on occurrences (range: 0 to 1)
Audio Features[1]	danceability: How suitable a track is for dancing (range: 0 to 1) energy: Perceptual measure of intensity and activity (range: 0 to 1) key: The estimated main key of the track (0 = C, 1 = C#/D♭, etc., range: 0 to 11) loudness: The relative loudness of the track (range: -60 to 0 dB) mode: Indicates whether the track is major (1) or minor (0) (range: 0 to 1) speechiness: Presence of spoken words in the track (range: 0 to 1) acousticness: Measure of whether the track is acoustic (range: 0 to 1) instrumentalness: Predicts whether a track contains no vocals (range: 0 to 1) liveness: Detects the presence of an audience in the recording (range: 0 to 1) valence: Musical positiveness conveyed by a track (range: 0 to 1) tempo: The overall estimated tempo of the track in beats per minute
Spotify Metadata[1]	type: The object type, typically "audio_features". id: The Spotify ID for the track, different from the LFM-1b track ID. URI: The Spotify URI for the track. track_href: A link to the Web API endpoint providing full details of the track. spotify_id: Spotify ID, part of the URI, but does not identify the type of resource. analysis_url: An HTTP URL to access the full audio analysis of this track. preview_url: A link to a 30-second preview (MP3 format) of the track.

3.3.2 Algorithmic development

This section describes the algorithm's development and highlights choices made that could affect its performance. For this research, an interactive genetic algorithm (IGA) was created, with the purpose of recommending

[1]Spotify API Documentation can be found here

music in a way that is agnostic of the popularity of said music. This IGA iteratively refines a selection of music for a user, with subsequent generations having a more refined item set, meaning the recommendations are better fitted to the users' tastes.

3.3.2.1 Amount of items and generations

To understand the algorithm's mechanics, we first need to consider the structure of its evolutionary process. Each generation of the IGA contains a predetermined number of items. During initialization, the user or researcher can set the number of generations, directly impacting the duration of the refining process. A higher number of generations and items per generation typically leads to a longer refinement process, as more items need to be rated over more iterations.

3.3.2.2 Algorithm details

This section will detail the development of the algorithm, going into the functions that make up the algorithm, as well as the choices made. The pseudocode for the algorithm can be seen in Algorithm 1.

3.3.2.2.1 Initial generation The initial population for the IGA is generated by randomly sampling a number of individuals from the dataset, to create a population of tracks for the first generation. This random sampling of the dataset was used for two reasons. Firstly, it ensures that no specific songs or artists are favored from the start, as this could introduce a bias towards these songs. Representative sampling could favor certain songs in genres with smaller numbers of songs. As will be discussed later, not all genres are represented equally in the data. Moreover, if we were to use a more sophisticated methodology such as collaborative filtering as a starting point to generate the first generation, it could reinforce existing patterns in the data, as well as reinforcing existing user preferences. Collaborative filtering suffers from popularity bias, as we have seen, making it a bad fit for the initialization phase of the IGA. This brings us to the second reason. Random sampling in the initialization phase allows the user of

Algorithm 1 Interactive Genetic Algorithm for Music Recommendation

Require: *individuals*

function INTERACTIVEGENETICALGORITHM(*individuals*, *crossover_rate*, *mutation_rate*)
    population ← RANDOMSAMPLE(*individuals*, *pop_size*)
    feedback_history ← [], *population_history* ← []
    for *gen* ← 1 to *generations* **do**
        for *individual* in *population* **do**
            if *individual.rating* is None **then**
                individual.rating ← GETFEEDBACK(*individual*)
            feedback.append(*individual.rating*)
        feedback_history.append(*feedback*)
        population_history.append(*population*)
        if *gen* < *generations* **then**
            population ← CREATENEWGENERATION(*population*, *feedback*, *pop_size*, *crossover_rate*, *individuals*, *mutation_rate*)
    return *feedback_history*, *population_history*
function CREATENEWGENERATION(*population*, *feedback*, *pop_size*, *crossover_rate*, *individuals*, *mutation_rate*)
    parents ← SELECTPARENTS(*population*, *feedback*, *pop_size* × *crossover_rate*)
    new_population ← [], *used_songs* ← {}
    while |*new_population*| < *pop_size* **do**
        *parent*1, *parent*2 ← RANDOMSAMPLE(*parents*, 2)
        if RANDOM < *crossover_rate* **then**
            *child*1, *child*2 ← CROSSOVER(*parent*1, *parent*2, *individuals*, *used_songs*)
        else
            *child*1, *child*2 ← COPYPARENTS(*parent*1, *parent*2, *individuals*, *used_songs*)
        for *child* in {*child*1, *child*2} **do**
            if RANDOM < ADJUSTEDMUTATIONRATE(*mutation_rate*, *feedback*[*parent*]) **then**
                child ← MUTATE(*child*, *individuals*, *used_songs*)
            if |*new_population*| < *pop_size* **then**
                new_population.append(*child*)
                used_songs.add(*child*)
    return *new_population*
function CROSSOVER(*parent*1, *parent*2, *individuals*, *used_songs*)
    child_features ← BLENDFEATURES(*parent*1, *parent*2)
    return FINDSIMILARSONGS(*child_features*, *individuals*, *used_songs*)
function MUTATE(*individual*, *individuals*, *used_songs*)
    mutated_features ← APPLYRANDOMMUTATION(*individual.features*)
    return FINDSIMILARSONG(*mutated_features*, *individuals*, *used_songs*)

the IGA to discover avenues that they otherwise wouldn't have. This aligns with the goal of the IGA of mitigating popularity bias, as users consider all sorts of songs. In this context, an "individual" represents a music track characterized by various attributes such as artist, song title, genres, and track features obtained from the Spotify API. The function `generate_population` also ensures that the population size does not exceed the number of available individuals. This population will be rated by the user, after which a new generation will be created.

3.3.2.2.2 Fitness evaluation The first step in creating a new generation is evaluating the fitness of the individuals in the previous generation. Fitness evaluation is performed by gathering feedback from the user or a simulated user model. For the purposes of this research, simulated users were used, which will be discussed later. The feedback is in Likert-scale formatting, ranging from 1 (strongly dislike) to 5 (strongly like) for each individual in the population. This feedback is critical as it directly determines the fitness score of each individual. The fitness score, as discussed, is the value genetic algorithms use to determine how close a solution (in this case, a song) is to the optimum. The function `evaluate_fitness` sorts the individuals based on the received feedback, making sure higher-rated tracks are prioritized. Then parents are then selected from this sorted list. This method of selection is known as truncation-based selection, as discussed in the literature review. This ranking influences the selection of parents for the next generation, ensuring that only the best-performing individuals contribute to the next generation.

3.3.2.2.3 Creation of a new generation After the individuals have been sorted, the parents are selected for the next generation. The selection of parents is based on the fitness scores that were obtained from the user. The individuals in the population of the previous generation are ranked by their fitness. As discussed, this fitness is the score given by the user indicating how well they meet user preferences and is the same as the feedback score received. The number of parents is chosen by a predetermined base crossover rate. This crossover rate reflects the fraction of the previous generation that

will be selected as parents. The base crossover rate is then combined with the average feedback to create a final crossover rate. This final crossover rate is therefore higher when the average feedback is high, meaning a larger part of the previous generation is selected for the next generation. This was done to allow the algorithm to exploit good solutions, while reducing the exploitation when the population is bad. This is inspired by research done by Hong et al. (2002)[36], where the authors varied the crossover and mutation rates based on the evaluation results in each generation. The function to calculate the final crossover rate is as follows:

$$\text{crossover\_rate} = \text{base\_crossover\_rate} \times \left(\frac{\text{average\_feedback}}{5} \right)$$

Through preliminary testing, a base value of 0.7 was found to perform the best. Base crossover rates from 0 through 1, in steps of 0.1, were tested with simulated users, and a value of 0.7 consistently showed the most balanced performance in terms of generating high-quality songs while maintaining diversity within the population. Higher values inhibited diversity, while lower values had significant amounts of good solutions not contribute to the next generation. An analysis of the effects of different crossover rates will be discussed in the results. When feedback is high, a large portion of high-fitness individuals is retained, while still allowing space for more solutions. The new generation is then created by sampling from the selected individuals. Until the population size is met, two parents will be selected from the previous generation. The crossover rate will then determine whether crossover will take place. If it does not, the children in the new generation will be copied over directly from the parents to the next generation. If it does, similarly to Kim et al. (2010)[51], we make use of BLX-α crossover. This function combines the features of two parent songs to create new offspring. In BLX-α crossover, as discussed in the literature review, the value of a feature is calculated by taking a value in between the values of the parents. This non-existent song will then be passed onto the `find_similar_song` function, which will be discussed later, to find an ex-

isting song in the dataset that matches the newly created song. After the potential crossover, the mutation rate will be compared to a randomly generated number, and if this number is lower, mutation takes place. When this happens, the children will be mutated to find new individuals. This mutation will be discussed in a later section. During the creation of a new generation, a list of songs that are already in the new generation is updated continuously, to ensure that there are no songs that appear multiple times in a generation. Crossover ensures that parents are selected in such a way that desirable traits, characterized by high-performing songs, are retained in the next generations, leading to the user getting similar individuals with high-fitness traits recommended. Desirable traits in this context specifically refer to song characteristics that align closely with the user's indicated preferences, such as specific genres and audio feature preferences. These traits are considered desirable as they match individuals that the user finds appealing. For example, when a user indicates strong preferences for high energy and high danceability songs, individuals with these traits would be considered desirable, making high energy and high danceability desirable traits.

3.3.2.2.4 Similarity calculation To perform meaningful crossover and mutation, the algorithm needs to be able to find the most similar song for a certain input. This is done by calculating track feature-based similarities. The `find_similar_song` function is what is responsible for this operation. The function first extracts the track features from the individual, namely the danceability, the valence and the energy. While testing, these track features were found to most meaningfully differentiate different songs. Other track features had large clusters of individuals with extremely similar values, making them less suited, or did not have an effect on the overall sound of a song (e.g. duration). The distribution of track features will be shown later in the dataset analysis. The function then calculates the pairwise Euclidian distance between these features and the features of all songs in the data. This is to ensure that the function works for the purposes of crossover and mutation, where the song that is inputted does not exist, and a similar

song needs to be found. The function for calculating the distance looks like this:

$$\text{Euclidian Distance(p,q)} = \sqrt{(d_p - d_q)^2 + (v_p - v_q)^2 + (e_p - e_q)^2}$$

where d is the danceability, v the valence and e the energy of either individual p or q. The distance is then computed between the input and all individuals in the data. This is then sorted based on the most similar songs, and the outputted song is randomly chosen from the top 5 most similar songs. This ensures that the output matches the inputted song, while still creating some randomness for diversity. The use of track features for similarity calculations was in line with research by Barone et al. (2017)[93]. This research showed that the acoustic features of a listener's preferred genres significantly influence their track choices within non-preferred genres. To illustrate the effectiveness of the `find_similar_song` function, several songs by well-known artists and their similar songs are shown in table 3.2.

Title	Artist	Genres	Danceability	Energy	Valence
Blue Jeans	Lana Del Rey	pop, alternative, jazz, rock, blues, electronic, folk	0.553	0.828	0.504
The Missing	Deerhunter	rock, alternative	0.585	0.826	0.514
A Whisper	Coldplay	alternative, pop, electronic	0.277	0.797	0.260
No Cars Go	Arcade Fire	alternative, rock, folk, pop	0.324	0.772	0.291
English Chamber Orchestra - C	Wolfgang Amadeus Mozart	classical	0.175	0.058	0.040
Begluckt darf nun dich (Tannhauser)	Richard Wagner	classical	0.163	0.046	0.061
I Am the Walrus	The Beatles	rock, pop, alternative, folk	0.401	0.556	0.654
Plastic Fantastic Lover	Jefferson Airplane	rock, folk, alternative, blues	0.410	0.505	0.714
Halo	Beyoncé	pop, rap, jazz, rnb, electronic, alternative	0.508	0.720	0.472
Utakata	Kagrra,	rock, alternative, folk	0.479	0.714	0.502
Electioneering	Radiohead	rock, electronic, pop	0.201	0.888	0.329
Promise	R a p h a e l	rock	0.254	0.859	0.342
Goodbye Yellow Brick Road	Elton John	rock, easy listening, alternative, blues	0.559	0.473	0.397
I Want to Be the Boy to Warm Your Mother's Heart	The White Stripes	alternative, blues, punk, folk, pop	0.522	0.465	0.394

Table 3.2: Comparison of Original and Similar Songs

3.3.2.2.5 Mutation The mutation operation in this algorithm plays a large role in helping users find preferred songs and introducing diversity into the population. This part of the methodology is novel, as other implementations for IGAs for music recommendation do not include this. In biological mutation, small changes are made in an organism's DNA, which might make them better suited to their environment. Selection pressure then ensures the individuals best suited for their respective environments survive and reproduce. Since individuals in the population are individual songs,

mutation of the "DNA" of these tracks is not possible, as we cannot edit the genres and track features and obtain a new song.

Therefore, we must use a different methodology for obtaining new individuals. To do this, mutation is performed on an individual's track features, to create a song with modified features that do not exist in the data. This is done by modifying the track features to introduce variations. This mutation is done in a uniform manner over all track features, based on the mutation strength. The mutation strength is calculated as a function of both the mutation rate and the user feedback. This mutation rate was set to 0.5, as it was found to strike a balance between exploration and exploitation within the algorithm. The effect of different mutation rates will be discussed in the results section. The user feedback was used to strengthen the mutation when the feedback is low, and weaken it when feedback is high, similarly to Hong et al. (2002)[36]. This ensures that highly-rated songs are kept over time and low-rated songs are replaced with new ones, increasing the chance of finding a better match in the next generation. The formula for the mutation was based on the highest and lowest feedback scores and is as follows:

$$\text{Mutation Rate} = \text{Base Mutation Rate} \times \text{Feedback Factor}$$

The feedback factor scales from 0.95 when feedback is the lowest, to 0.05 when the feedback is the highest. After mutation is performed, the new non-existent individual's features are compared to the full dataset using the `find_similar_song` function, and a song that closely matches the non-existent individual created by mutation is chosen. This therefore results in a new individual, mutated from the original in such a way that it introduces diversity.

To illustrate how the mutation function works, an example will be given for the mutation of a single song. For this demonstration, mutated songs will be based on Wolfgang Amadeus Mozart's "Ave verum corpus, K. 618 (Arr. Spindler for English Horn, Strings and Organ)", with various feed-

back, to see what the outputted songs will be like. This song was chosen as a demonstration example. The track and its features in the dataset can be seen in Table 3.3. The full table of mutated songs for each feedback score can be seen in the Appendix. The idea is that when the feedback is high, similar songs will be chosen, while when feedback is low, very dissimilar songs will be returned.

Wolfgang Amadeus Mozart	Ave verum corpus, K. 618 (Arr. Spindler for English Horn, Strings and Organ)	["classical"]

danceability	energy	speechiness	acousticness	instrumentalness	liveness	valence	tempo
0.175	0.0578	0.0422	0.913	0.895	0.0906	0.0395	68.005

Table 3.3: Audio Features of Wolfgang Amadeus Mozart's 'Ave verum corpus, K. 618 (Arr. Spindler for English Horn, Strings and Organ)'

When the feedback rating is high, the mutate function returns songs that are expected to be similar to the Mozart song, such as "The Lake In The Moonlight" by Tchaikovsky. Both pieces share similarities in their classical instrumentation and structural qualities. However, one surprising item to come from the mutation of the Mozart song is the song "Eon Bleu Apocalypse" by Tool. At first glance, this song would seem like a bad fit for someone who rates the Mozart song highly. The categorized genres are "alternative", "rock" and "heavy metal". However, when looking at the audio analysis, the song is also a quiet song, with similar values for danceability, energy, speechiness, and acousticness to the Mozart piece. This shows the mutation function's focus on musical features, in line with the research by Barone et al.[93], which says musical feature preferences transfer over to other genres. The song is a relatively quiet instrumental song, meaning that users who like classical music, may also enjoy this track.

When the feedback rating is low, the mutate function returns songs dissimilar from the input song, to increase exploration within the data. When the feedback on the Mozart song is low, one example is the song "The world is my land" by Jupiter and Okwess International, a band from Kinshasha in the Democratic Republic of the Congo[94]. This choice shows the system's ability to broaden user exposure to diverse musical styles. Jupiter and

Okwess International are known for their unique fusion of traditional Congolese rhythms with modern rock influences[94], making it a song that is very dissimilar from the Mozart song inputted.

3.3.2.2.6 Final algorithm These functions are put together in a larger function, the `interactive_genetic_algorithm` function. This function is used to orchestrate the entire process of evolving a population of music according to user preference. The function begins by setting the number of generations and the population size, which are determined by the user or set to testing values for simulated users. The population is generated using the `generate_population` function. For each generation, several steps are performed. Firstly, the feedback is collected from the users. Once feedback is collected for all individuals, the algorithm will calculate the average feedback score over that generation, for later analysis. After this, a new generation is created using the `create_new_generation` function. This function selects the top-performing individuals from the current generation based on their fitness scores, and then creates new individuals by combining genetic material from these selected parents through the crossover and mutation operations discussed previously. This newly generated population will then again be rated, and this completes the cycle. This repeats until the preset number of generations is reached. Over time, the algorithm aims to converge towards a set of music recommendations that align closely with the user's preferences.

3.3.2.3 Simulated users

For the evaluation of the algorithm, both real users and simulated users could have been used. Due to time constraints, a real user evaluation of the interactive genetic algorithm was not feasible. However, simulated users were created to replace regular users in evaluating the individuals in the IGA. The simulated users were designed to emulate a user's music preferences and evaluate songs based on specific criteria, which will be discussed later. This evaluation is based on a consideration of both the genre of the song and its specific track features. This allows the simulated users to

closely mirror real user behavior.

To simulate a user, the function begins by initializing a neutral rating (3) for a given individual. This starting point is based on an assumption of indifference, reflecting that the simulated user has no prior bias towards or against any song. This is done such that later adjustments to this rating are effective and reflect human behavior. This rating is then adjusted based on several factors, starting with the song's genres. The simulated users have predefined genre preferences categorized into highly preferred, preferred, neutral, disliked, and highly disliked genres. If a song belongs to one of the highly preferred genres, the rating increases with 2. On the other hand, if the song is in a highly disliked genre, the rating decreases with 2. This ensures that the simulated user returns a rating that reflects its preferences. The change in rating is also done for the liked and disliked genres, with a less significant change of adding 1 or subtracting 1 respectively. In addition to genre preferences, the simulated user has preferences for specific audio features, which include danceability, energy, and valence. The features of an input song are compared to the preference. For instance, if a song's danceability exceeds the user's preferred level, the rating is increased by 0.5; otherwise, it is decreased by 0.5. Similar adjustments are made for energy and valence until finally a final rating is obtained for the song. The incremental adjustments reflect the idea that users' preferences typically manifest in subtle, rather than drastic, changes in their perception of a song. The choice of genres and audio features as primary factors is based on empirical studies, such as Karnop (2019)[95], which demonstrated the impact of personal factors on music selection behavior and the corresponding audio features of self-selected music, using the same audio features as were used in our system. As mentioned in the section 3.3.1, the algorithm does not make use of genre information to make recommendations. However, as is discussed in this section, the simulated users do. Track features were found to be good predictors of genre information, and this provided a more fair comparison to existing methods, also focused on track features, while providing interpretable simulated users.

As an example, we have created a person who likes to listen to classical

music. This was done by setting the highly preferred genre as "classical". All others were left empty, and the highly disliked genres were set to all other genres available. The track feature preferences were set by taking the median values of the musical features for all classical songs in the data. After these preferences were set, the simulated user rated the entire dataset. Tracks with a rating of 1, 3 and 5 were saved and were randomly sampled to obtain the tracks seen in table 3.4.

Rating	Track Name	Artist	Genres	Danceability, Valence, Energy
5	A Thousand Years	Sting	pop, jazz, alternative, easy listening, folk, blues, reggae, classical	0.605, 0.33, 0.437
5	Settler	Balmorhea	classical, electronic, folk	0.274, 0.244, 0.425
5	Fritz Lang	Chapelier Fou	classical, alternative	0.571, 0.392, 0.606
5	Wait For Me	Vangelis	new age, classical, easy listening	0.471, 0.172, 0.363
5	I Let You Down	365daband	pop	0.531, 0.308, 0.677
3	Drawing the Line	Porcupine Tree	rock, alternative	0.519, 0.45, 0.553
3	COLORS	-OZ-	rock, alternative	0.551, 0.171, 0.279
3	Scars	Blackfield	rock, alternative, pop	0.397, 0.494, 0.477
3	Last Conversation	Veronica Falls	pop, alternative, rock, folk	0.234, 0.412, 0.626
3	The Hour Of Need	Mind's Eye	rock	0.469, 0.292, 0.349
1	Content To Play Villain	dälek	rap, electronic, alternative	0.484, 0.107, 0.68
1	Slaughtered	Pantera	heavy metal, rock	0.239, 0.53, 0.964
1	Stay	MNDR	electronic, pop, alternative, world	0.688, 0.77, 0.907
1	Turquoise Hexagon Sun	Boards of Canada	electronic, alternative	0.567, 0.0445, 0.754
1	Iron	Ensiferum	folk, heavy metal	0.205, 0.387, 0.942

Table 3.4: Evaluation of Tracks by Simulated Classical Music Listener

3.3.3 Platform, environment and package details

This section will detail the platform and packages used to enhance reproducibility. The code for this research was written in Python, using the Google Colab platform. This platform provides a Jupyter notebook environment, allowing for easy testing and development of separate parts of code. Colab was chosen for its familiarity and its access to powerful computational resources.

3.3.3.1 Packages used in the initialization

Several packages were used for the development of the IGA. Firstly, the requests library was used to communicate with the Spotify API, using HTTP requests to get the token for API requests, as well as the track features and other relevant information such as the Spotify URI.

3.3.3.2 Packages used in the IGA

In the IGA itself, three well-known packages were used. Random was used for random sampling, such as in the sampling of the dataset to create the initial population, as well as with the creation of randomized preferences for simulated users. Moreover, the random library was used to randomize some parts of the genetic process itself, such as getting a random number to compare to the mutation and crossover rates, to figure out if mutation should happen, which is standard practice. If the mutation rate is higher, the chance of getting a random number that is smaller than this mutation rate gets lower. The math package is used for mathematical operations within the IGA. One example is calculating the Euclidean distance between two individuals' track features. The copy package is used to create copies of the track_features of an individual in the mutate function. This ensures that the original's features are not altered. In Python, when you assign a value to a variable, you are not creating an independent copy, you are instead creating a reference to that same data. Therefore, if we were to do `mutated_features = individual.track_features`, and then modify the `mutated_features`, the originals `track_features` will also be edited.

3.3.3.3 Packages used in the reporting and testing

In the reporting and testing of the algorithm, other packages were used. Some are well known, such as SciPy.stats for statistical tests, matplotlib for data visualization, including histograms, bar charts and pie charts, as well as NumPy for numerical operations. NumPy, for example, is used to calculate the mean, as well as the pairwise Euclidean distance in song similarity calculations.

3.3.4 Evaluation measures

To evaluate whether the IGA achieves its goals of increasing feedback scores over generations and mitigating popularity bias, evaluation measures are needed. Moreover, these feedback scores not only need to reflect increases in satisfaction through the increase of feedback scores for users. The evalua-

tion measures need to also compare the algorithms featured to each other on measures such as diversity and popularity bias. Here, we will summarize proposed measures for evaluating the IGA.

1. **Average feedback score**: This metric measures the mean feedback score across generations. An increasing trend here indicates that the algorithm is successfully learning user preferences and outputting more satisfactory results. Moreover, this allows the comparison between different sets of users, for example, those with niche interests.
2. **Average Percentage of Long Tail Items (APLT)**: Described in equation 3.1, this metric measures the proportion of long-tail items in recommendations. A higher APLT indicates better mitigation of popularity bias.
3. **Convergence Rate**: This measure tracks how quickly the algorithm converges to high-quality recommendations. It can be assessed by analyzing the rate of improvement in feedback scores across generations.
4. **Intra-List Diversity**: This metric, based on Ziegler et al. (2005)[80], evaluates how diverse the recommendations are in a recommendation list by looking at the pairwise similarity between tracks. Instead of comparing the diversity of the entire recommendation, we can use it for the evaluation of the IGA by looking at the diversity of songs within the first and within the final generation.

Using these metrics, a good overview of the performance of the IGA can be obtained, and compared to other existing methodologies.

4. Results

In this section, the outcomes of the interactive genetic algorithm for music recommendation are presented. Firstly, the dataset and experimental setup will be shown, along with choices made. Afterwards, the performance of the algorithm will be evaluated based on a variety of metrics, and the evolution of the performance across generations will be shown. The results are also compared to baseline methods, and the robustness of the algorithm across different simulated user preferences will be shown.

4.1 Dataset and Experimental Setup

4.1.1 Dataset Description

As described earlier, the dataset that was used for the interactive genetic algorithm is a combination of various different sources. The dataset is made up of tracks from the LFM-1b dataset. The information available for these tracks is then expanded using the Spotify API, as well as with the LFM-1b User Genre Profile dataset. The final dataset contains 18,953 items, after removing items for which genres or track features were not available. This section contains descriptions of the features of the data in the dataset, as well as showing the distribution of these features within the data.

The distribution of genres in the dataset is shown in Figure 4.1. The dataset contains 19 different genres, with the genres with the most occurrences being rock, alternative and pop. The genres with the least occurrences are spoken word, vocal and new age. The distribution of genres shows a skew towards certain genres while having fewer available items in others.

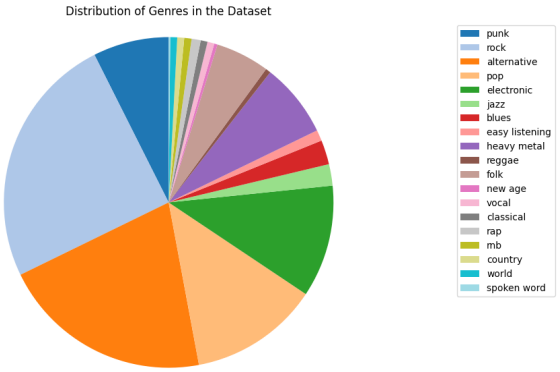


Figure 4.1: Distribution of Genres in the Dataset

The distribution of track popularity in the dataset is depicted in Figure 4.2. As can be seen, the popularity follows a long-tail distribution, indicating that a small number of tracks have very high popularity, while the majority of tracks have relatively low popularity. As described earlier, this is a pattern often observed in datasets for music recommendation systems.

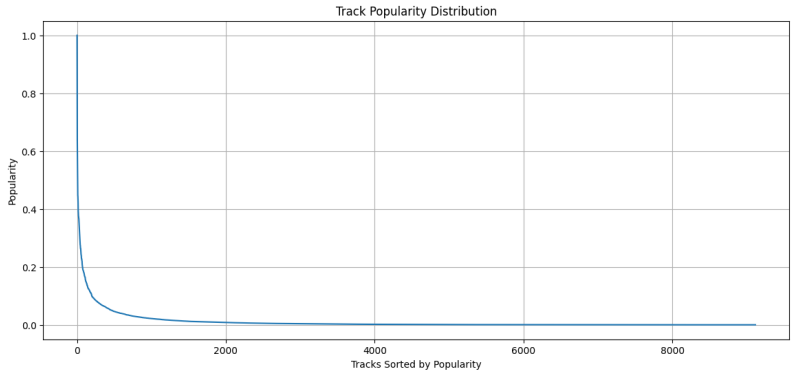


Figure 4.2: Track popularity in the dataset, sorted from high to low

In figure 4.3, the distribution of track features for songs in the dataset can be seen. One interesting thing to note here is the distribution of time signatures. As described by Hardman and Talarczyk (2021)[96], the most common time signature in Western music is 4/4, and we can indeed see that this is also the case for the songs in our dataset as well. Danceability follows roughly a normal distribution around 0.6, indicating moderate danceability for most songs. Energy is skewed towards higher values, indicating that there are many high-energy tracks in the dataset. Loudness shows a normal distribution peaking around -5 dB. This is consistent with the trend identi-

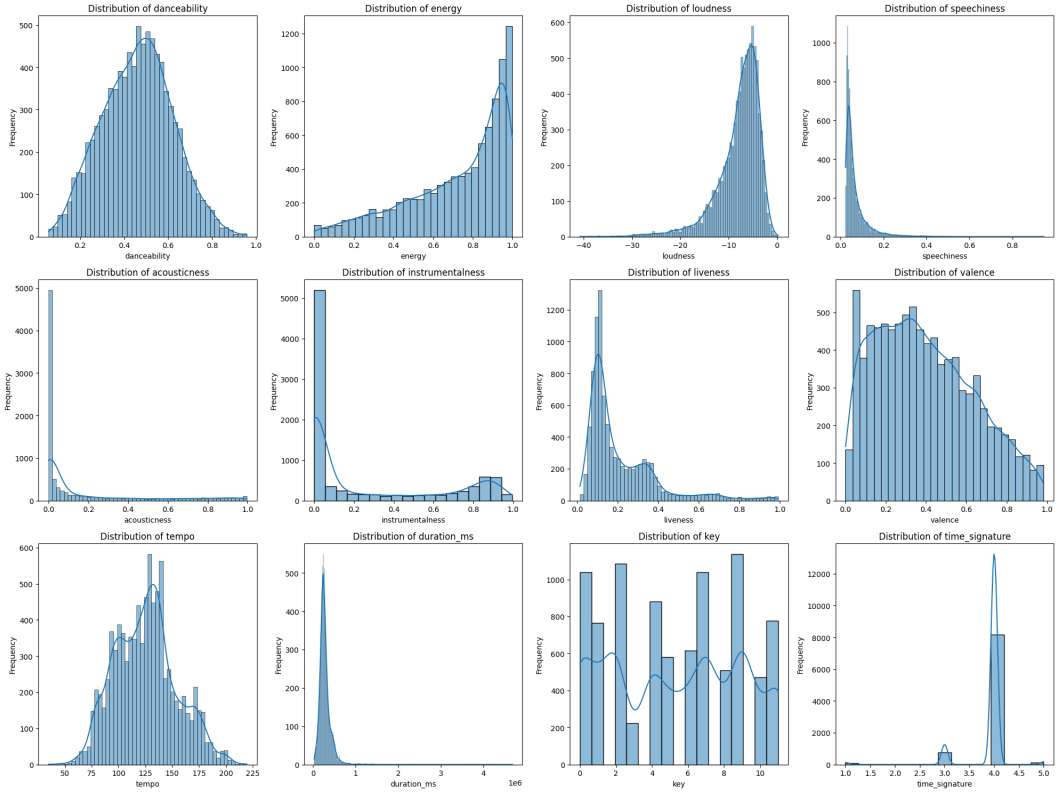


Figure 4.3: Distribution of track features within the dataset

fied by Haghbayan et al. (2019)[97], which shows that music has been getting progressively louder over time. Speechiness is strongly skewed towards lower values, reflecting minimal spoken words in most songs. Acousticness and instrumentalness are both highly skewed towards lower values, suggesting that the majority of tracks are non-acoustic and vocal, respectively. However, instrumentalness does show a small increase near the higher end of the plot, implying that there are some tracks that are totally instrumental. Liveness indicates low values for most tracks, implying most tracks are studio recordings. Valence distribution is relatively uniform with a slight peak in the middle, showing a wide range of emotional tones. Tempo follows a normal distribution centered around 120 BPM, which is common in music.[98] Duration peaks between 200,000 and 300,000 milliseconds (3 to 5 minutes), aligning with typical song lengths. Finally, the key distribution is relatively uniform across the 12 keys, with slight prevalence in some.

4.1.2 Experimental Setup

The experiments were conducted with a population size of 20, over 10 generations. These numbers were chosen to achieve a balance between realism and giving the algorithm the room to generate results. Using too small population sizes leads to less effective exploration of the solution space, whereas using too large population sizes makes the simulation less representative of real-world conditions, as well as being more computationally taxing. We will explore the impacts of different population sizes later in the results. The algorithm was run on the Python 3 Google Compute Engine backend, where one run of the algorithm, using a simulated user that rates 20 individuals over 10 generations took approximately 15-30 seconds, depending on the amount of crossover and mutation applied during the run. Because of the way the system is designed, the crossover and mutation rates are higher with lower ratings, and lower with higher ratings. Therefore, when there are low ratings, more calculations need to be made. Simulated user preferences were configured based on predefined genre categories and preferred audio features, randomized for each run, giving an accurate picture of the increase in feedback scores over time. The algorithm will be run repeatedly, to account for variability in mutation, simulated user preferences and initial population generation.

To compare our methodology with other methodologies, we will simulate a large number of runs of the algorithm, looking at the evolution of feedback scores over time. This gives a frame of reference for comparison to other methodologies, as well as decreasing the likelihood that increases in feedback scores are due to random chance. Specifically, we will compare the method of Kim et al. (2010)[51] to our methodology, looking at the various evaluation measures established earlier. This choice of point of comparison was chosen because the other existing IGAs for music recommendations base their methodology on this paper, making it a foundational work and good point of comparison. In the experimental framework, we will classify users based on their feedback ratings for the initial generation. The motivation for this is that users with more niche tastes will rate the first

generation of the algorithm more lowly, giving us a frame of reference for what users have more traditional preferences, as well as those having more niche preferences, and seeing how these users are able to improve their recommendations over time. This allows the comparison of the results of these groups of users, and draw conclusions from the comparison between the results of our method and other methodologies.

4.2 Algorithm performance

In this section, we will report the performance of our methodology, based on tests using randomized user preferences. The effectiveness of our algorithm will be tested using the performance indicators established earlier. First, we will give a general overview of the performance of the algorithm. After this, we will categorize two user groups, based on the feedback scores in the initial generation, and compare the performance across these to show the adaptability of the algorithm to both. After this, we will report in a similar manner on the performance of Kim et al. (2010)[51], given the same tests. We will then compare the performance of both, looking at the categorized user groups.

4.2.1 Performance over Generations

The performance of our algorithm was evaluated using simulated users. The feedback scores were averaged for all users for each generation, to obtain a representative overview of the evolution of feedback scores. Figure 4.4 shows the average feedback score progression over the 10 generations for all runs. 100 runs were conducted, with each having a new randomized simulated user profile.

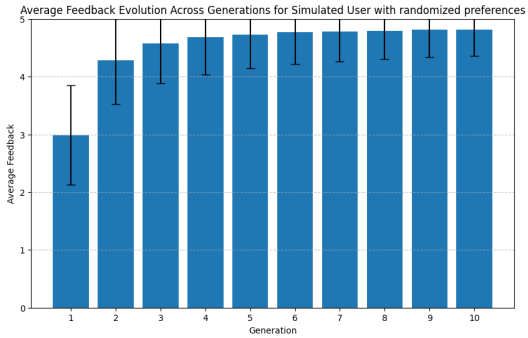


Figure 4.4: Average Feedback Evolution Across Generations for Simulated User with Randomized Preferences for Our Method

As can be seen in the graph, there is an upward trend in average feedback scores over the generations, meaning that the algorithm can successfully adapt to user preferences. The average feedback score increased from 2.99 in the initial generation to 4.82 in the final generation. This represents a 61.20% improvement from the first to the last generation. The standard deviation in the first generation was 0.86, while in the final generation it was 0.45. It should be noted that, from the third generation onwards, the increases as an average for all runs are quite low. In the third generation, the average feedback score is 4.58, representing a 5.24% increase to the final generation. However, the standard deviation in the third generation is 0.69, showing less consistency as compared to the final generation. Out of 25000 items over all generations and all runs, 23163 were considered to be part of the long tail. This means that the APLT is 92.65%. Items were considered to be in the long-tail when their popularity was under 0.1, meaning they received a maximum of 10% of the amount of listens than the most popular song. This was based on the distribution, as seen in Figure 4.2. A popularity value of 0.1 separates the "head" of the distribution (the most popular tracks) from the "tail" (the less popular tracks). The average intra-list diversity between the tracks in the first generation is 0.0949, whereas it is 0.0597 in the final generation. This means that there is a higher degree of similarity between the tracks in the final generation.

4.2.2 Performance for Different User Classifications

To assess the performance of the algorithm for different user types, users were categorized based on their initial feedback scores. Users with average feedback scores in the initial generation more than one standard deviation below the mean were classified as having "niche" preferences, while those with an average initial population feedback score one standard deviation above the mean were classified as having "traditional" preferences. Lower average initial scores indicate that that has preferences that don't line up with the randomized initial generation, meaning the preferences likely do not align with the dataset distribution. This means they are more likely to enjoy items that have preferences that occur in lower levels in the data, meaning they are more niche. The comparison between more niche and more traditional users can be seen in Figure 4.5. In this case, out of 100 runs, 23 were considered "niche", while 22 fell into the "traditional" category.

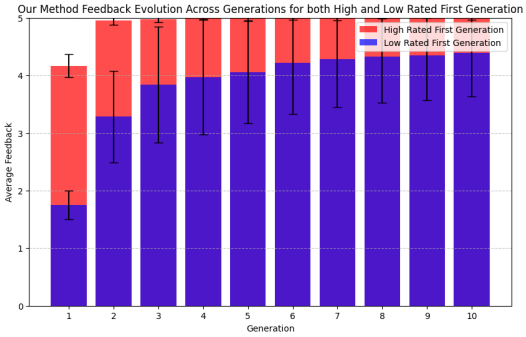


Figure 4.5: Average Feedback Evolution Across Generations for Simulated User with High and Low Rated First Generations

These results show that for users with traditional preferences, represented as a high-rated first generation, the algorithm starts with a relatively high baseline. This shows that the initial recommendations already align well with their tastes. The second generation nears perfect scores for these users. This continues in later generations. This can be attributed to the mechanism of dynamic mutation when feedback is high, leading to little mutation to songs in the previous generation to create the previous generation. The initial generation for this group of users had an average feedback score of 4.17, while the final generation had an average feedback score

of 4.99. This represents a 19.66% increase. This means that the increase is lower than the average for all users, but the overall scores are higher. This is logical, as starting from a higher score and increasing to close to the maximum, represents a lower increase than when starting with lower feedback scores. For the users with feedback scores 1 standard deviation lower than the mean initial feedback scores, the initial generation had a significantly lower average score of 1.75, meaning the average score for a song in the first generation for this group was very low. The final generation had a rating of 4.39, which is an increase of 150.85%. This is a much larger increase than for traditional users, more than doubling the original feedback score average.

4.3 Comparison to Other Methods

The algorithm proposed by Kim et al. (2010)[51] serves as a point of comparison to our methodology. Due to the unavailability of the code, the methodology was reconstructed for use in this paper using the methodology section of their paper. They specified several algorithmic parameters, such as a crossover chance of 0.5, which we adopted for consistency. To evaluate the performance of the method proposed by Kim et al. (2010)[51], we conducted similar experiments using simulated users with randomized preferences.

4.3.1 Performance over Generations

Figure 4.6 shows the average feedback of Kim et al.'s method. Similarly to our algorithm, the feedback scores for this method show an upward trend, indicating that their method also adapts to preferences over time.

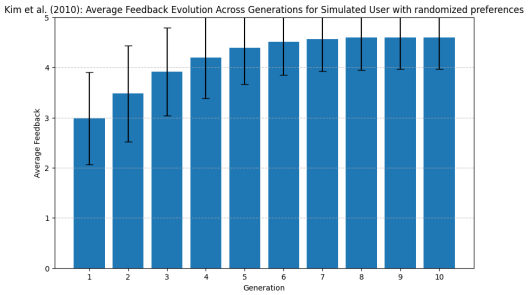


Figure 4.6: Average Feedback Evolution Across Generations for Simulated User with Randomized Preferences for Kim et al. (2010)

The initial average score for all simulated users for Kim et al.'s method is 2.99, which is similar to our method. The feedback score increased to an average of 4.60 in the last generation, meaning an increase of average scores of 53.85%. The error bars in this graph do show that the averages for Kim et al.'s method have a larger spread, meaning that the performance of their method is more variable across different simulated users. In the first generation, the standard deviation is 0.92, whereas it is 0.63 in the final generation. The APLT for this method was 90.34%, which is not a large deviation from our method, where it was 92.65%. The average intra-list diversity between the tracks in the first generation is 0.0796, whereas it is 0.0320 in the final generation. Similarly to our method, the intra-list diversity decreases, meaning that the tracks in the final generation are more closely related.

To statistically compare the feedback scores of the final generations of both methods, an independent t-test was considered. For this test, multiple assumptions need to be met. The first one is the independence assumption, where the samples from both methods need to be independent of each other. Since we are comparing feedback scores of simulated users, this assumption is met. Moreover, the data for both methods should be normally distributed. This can be checked using the Shapiro-Wilk test, which in our case results in a p-value of <0.05 for both results, meaning neither outcome is normally distributed. Additionally, we conducted a Levene's test to check if the variances between the two groups were not significantly different. The result of this test is a p-value of 0.8180, meaning the variances are not significantly different.

Because our results do not meet the normality assumption, we used a Mann-Whitney U test. The result of this test is a statistic of 91.0 and a p-value of 0.138. This shows us that the difference in performance between our method and that of Kim et al. is not statistically significant. Since the p-value is higher than the commonly used threshold of 0.05, we can state that our method does not significantly outperform Kim et al.'s method for all users when using simulated users with randomized preferences.

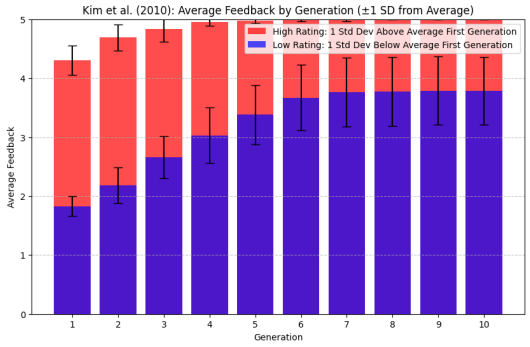


Figure 4.7: Kim et al. (2010)[51] Feedback Evolution for Low and High Initial Feedback Scores

4.3.2 Performance for Different User Classifications

User classification was identical here to the tests conducted using our method, meaning initial generations are considered to be niche when the average of the initial generation is at least one standard deviation lower than the average for all runs, and they are considered to be traditional when the feedback score average is higher than at least one standard deviation. The comparison between these can be seen in Figure 4.7.

As can be seen, the difference between the high and low-rated initial generation groups is higher than with our method. For the traditional users, the average feedback scores approach perfect scores in the final generation. The first generation for this group has an average feedback score of 4.05, which increases to 4.97 in the last generation. This is an increase of 20.62%. The standard deviation for this group is 0.31 in the initial generation, and 0.17 in the final generation. For the more niche users, the increase is noticeably worse. The initial population has an average feedback score of 1.83. This increases to 3.79 in the final generation, meaning the feedback increases by 107.27%. The standard deviation is 0.18 in the first generation, and 0.84 in the final generation.

While final results for all users are similar, as discussed in section 4.3.1, the results for users considered to be niche were higher for our method than with Kim et al.'s method, with the final generation having an average feedback rating of 4.39, as compared to 3.79 using Kim et al.'s method. To statistically test the difference, we used the same Mann-Whitney U test as in the

previous comparison. Here, we found a p-value of 0.0088, with a statistic of 85.0. This means that for users with niche tastes, the difference between feedback scores for our method and Kim et al.'s is statistically significant. To make our analysis complete, we can compare the results for traditional users, to determine if this increase in performance from our method comes at the cost of the results of the more traditional listeners. Our method had a final feedback score of 4.99, whereas Kim et al.'s method achieved a final score of 4.97. The Mann-Whitney U test resulted in a p-value of 0.8815, with a statistic of 45.0. This indicates that there is no significant difference between the two groups, and therefore for users that are considered traditional, there is no significant difference between feedback scores.

The convergence rate between both methods was also calculated. This is a measure that measures the difference in mean score each generation. This indicates the speed at which the algorithm is able to converge to well-performing solutions. Both algorithms have similar initial and final scores overall randomized simulated users, meaning that the convergence rate will most likely also be similar. This is indeed the case, with our method having an average convergence rate of 0.189, and Kim et al.'s method having an average convergence rate of 0.179. A plot of the convergence rate for each generation can be seen in Figure 4.8. Here, we can see the difference between both methodologies. For Kim et al.'s method, the maximum convergence rate was 0.489, whereas our method showed a much higher maximum convergence rate of 1.148.

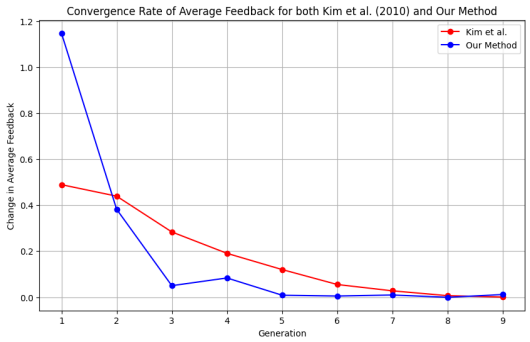


Figure 4.8: Convergence Rate of Average Feedback for both Kim et al. (2010) and Our Method

Metric	Our Method	Kim et al. (2010)
Average Feedback Improvement	61.20%	53.85%
Average Score in Final Generation	4.82	4.60
Standard Deviation in Final Generation	0.45	0.63
APLT	92.65%	90.34%
Convergence Rate in Initial Generation	1.148	0.489
Average Convergence Rate	0.189	0.179
Final Generation Intra-List Diversity	0.0597	0.0320

Table 4.1: Comparison of Key Metrics between our method and Kim et al. (2010)

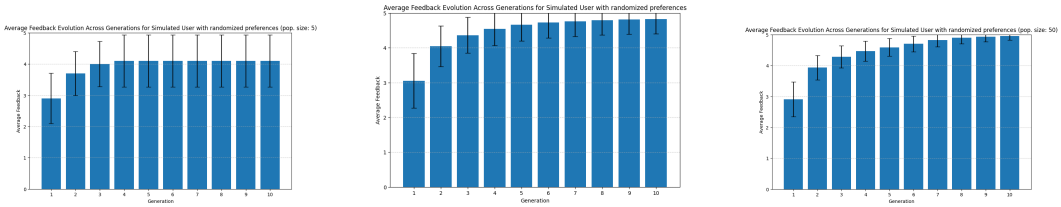
In Table 4.1 we show the differences between the method from Kim et al. (2010)[51] and our method. As can be seen, Kim et al. (2010) excels in catering to traditional users, but struggles with more niche users.

4.4 Comparison of different parameters

In this section, we will compare the performance of the algorithm under various parameters. In each section, we will report the results of a simulation using the same algorithm as discussed in section 4.2, while varying one parameter. This will provide a better understanding of how the parameters influence the evolution of the feedback scores.

4.4.1 Population size per generation

The population size of each generation determines the number of individuals in each generation. Varying this number can influence multiple factors that determine the outcome. When there are too few items per generation, the algorithm has lower chances of evolving the population to a state that is satisfactory to the user. With too many items in the generation, the algorithm could develop better, more sophisticated solutions, at the cost of the amount of items that need to be rated. When real users are considered, having many items per generation could lead to fatigue and make the user experience worse, as well as being more computationally intensive. It is therefore important to achieve a balance between both. In figure 4.9, three population sizes can be seen, 5, 20 and 50.



(a) Population size: 5 **(b)** Population size: 20 **(c)** Population size: 50

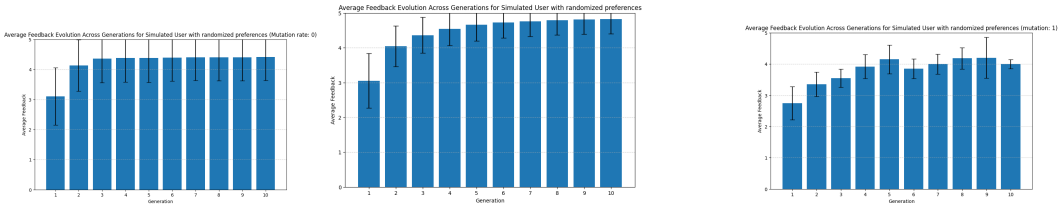
Figure 4.9: Comparison of different population sizes

The results indicate that a small population size can lead to evolutionary stagnation, as there are too few individuals to generate significant mutations or foster the evolution of better solutions using crossover. Moreover, having a very large population brings diminishing results, where the results for the feedback score are only slightly better than a smaller population size, but there are many more items to rate each generation. This large number of items is computationally costly when using simulated users, and unrealistic when using real users.

4.4.2 Mutation rate

The mutation rate is a parameter that controls the variation that is introduced in each generation by modifying the individuals. The mutation rate needs to be high enough to allow for an increase in exploration, while not disrupting convergence. When the mutation rate is too low, the algorithm could get stuck in local optima of "least disliked" individuals, instead of finding actual preferences. In figure 4.10, results of the algorithm with 3 mutation rates are shown, 0, 0.5 and 1. These were chosen to show low, optimal and high rates of mutation, respectively. A low mutation score is already represented by the results of Kim et al. (2010)[51], which performed relatively well for the average user, while having no mutation. However, our method provides other advantages, such as the addition of a dynamic crossover rate, which could still affect the outcome.

As can be seen in Figure 4.10, high mutation causes good solutions to be mutated away, therefore leading to less high scores for users. There is still a generally upward trend for this, but many optimal solutions are still mutated away, meaning the optimum is difficult to find for the algorithm.



(a) Mutation rate: 0 **(b)** Mutation rate: 0.5 **(c)** Mutation rate: 1

Figure 4.10: Comparison of different mutation rates

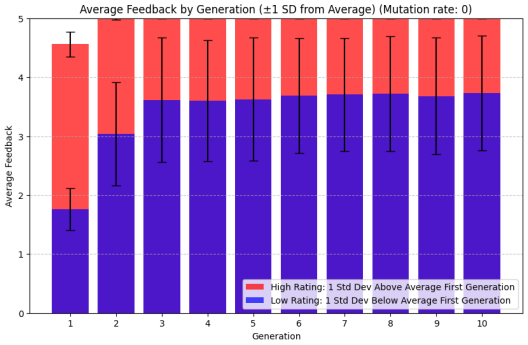


Figure 4.11: Mutation rate: 0 for Users with Traditional and Niche preferences

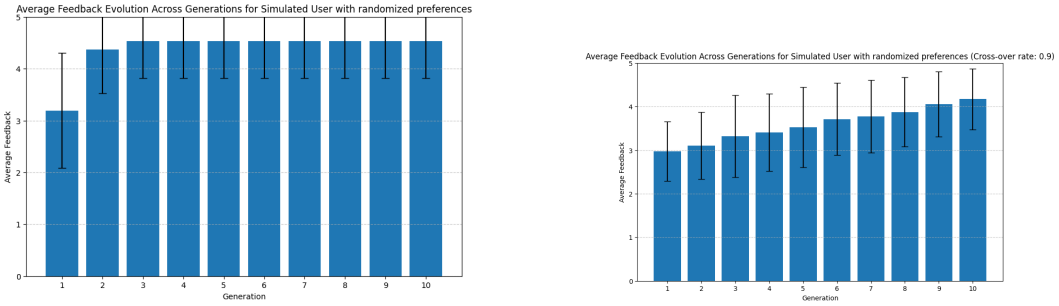
Low mutation, similarly to Kim et al. (2010)[51], causes the algorithm to plateau after a certain point. This plateau is more drastic for users with niche interests, as can be seen in Figure 4.11.

This strengthens our findings that the inclusion of mutation helps especially these users, while not harming more traditional users. Here, the improvement of our method over Kim et al. (2010)[51] can also be seen, with regards to the convergence rate. Even though the feedback scores for niche users in the final generation are similar for their method versus ours without mutation (3.79 vs 3.73), our method achieves a relatively close score in the third generation, whereas their method does so in the sixth generation.

4.4.3 Crossover rate

The crossover rate defines how likely it is that two parent solutions are combined. Low crossover rates will make the population static, changing very little each generation and throwing away many individuals from the previous generation. A high crossover rate will cause too many sub-optimal solutions to be kept, leading to stagnation of the results. To compare the

effects of a differing crossover, it is necessary to also set mutation to 0. If this is not done, the mutation operator will still cause the algorithm to explore the solution space for optimal solutions, and the effects of crossover will not be clear. We demonstrate low and high crossover rates, 0.1 and 0.9 respectively. We used a crossover of 0, as a crossover of 0 and a mutation of 0 will lead to a static population where no new individuals are generated. The results of the experiment can be seen in figure 4.12.



(a) Crossover rate: 0.1 **(b)** Crossover rate: 0.9

Figure 4.12: Comparison of crossover rates

The results show that a crossover rate that is too high, will lead to a stifled growth rate. Each generation keeps a large percentage of individuals from the previous generation, having a high likelihood of keeping low-quality solutions. A low crossover rate seems to do relatively well for all users. However, when looking at the performance for users with low initial feedback scores, the improvements quickly plateau, as can be seen in Figure 4.13. This is because there are no good parent solutions to keep for the next generation or to cross over into new solutions.

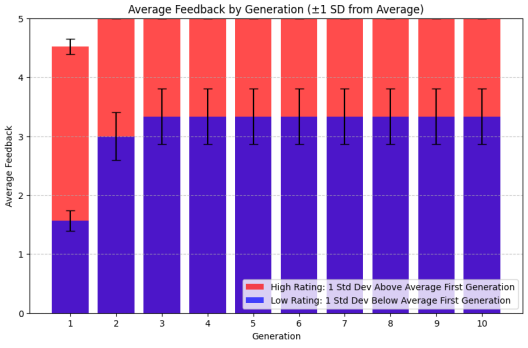


Figure 4.13: Crossover rate: 0.1 for Low and High Initial Feedback Scores

5. Discussion

In this section, we will discuss the implications of the results and answer the research questions based on the results. Moreover, we will put the findings in the broader context of the literature. Additionally, limitations to the study will be discussed that could have an effect on the interpretation of the result and future research directions will be proposed.

5.1 Interpretation of results

In this section, we will interpret the results found in the experiments performed. The results of our experiments indicate that our method is effective in improving recommendation quality over multiple generations. This is consistent with previous methods, where feedback scores also improved over time.

The average feedback score in our method increased from 2.99 to 4.82, which is a 61.2% increase. This increase suggests that our IGA is able to effectively learn and adapt to user preferences, as these were randomized. The decrease in standard deviation from 0.86 in the first generation to 0.45 in the final generation, representing the spread of averages of each generation across runs, indicates greater consistency for all users over time as the algorithm adapts to user preferences. This could be due to the inclusion of adaptive mutation and crossover rates, that vary based on the feedback scores. This is an improvement over previous methods.

Comparing our method to the baseline of Kim et al. (2010)[51], we observe that both methods improve recommendations over time. Kim et al.'s method performed generally on par for all users as compared to our method, with scores in the final generation of 4.60 as compared to 4.82 for our method. This is a difference of 4.78%. This implies that the inclusion of mutation does not significantly harm or improve recommendation quality for the average

user.

Our method achieved an APLT of 92.65%, which indicates an agnostic approach to recommendation. This is logical, as the methodology is based on the content of the items. Similarly to content-based recommenders, this ensures that the recommendations are not based on occurrences in the data, making sure that the recommendations are not affected by popularity bias. What an interactive genetic algorithm improves as compared to a regular content-based recommender is its ability to evolve and adapt to user feedback dynamically over multiple iterations. Moreover, the cold start problem, as discussed in the literature review, can be addressed using an IGA. IGAs can address this problem more effectively by rapidly adapting to initial feedback, whereas traditional content-based systems can struggle with a limited user history.

The convergence rate in the earlier generations was significantly higher in our method than in Kim et al.'s method. For our method, the initial generation improved with a score of 1.148, whereas Kim et al.'s method has a lower convergence rate of 0.489, where they both averaged roughly the same starting point. This could be attributed to the improvements made to the system, both in the addition of mutation functionality, as well as the dynamic crossover and mutation rates, helping exploit good solutions to recommend more effectively to users. Other experiments, where the mutation was set to 0, showed that this higher convergence rate was most likely due to the dynamic genetic operation rates. The results also show the advantage of our method in speeding up the initial improvements, without compromising long-term performance. This could make our method more suitable as an application to address the cold start problem than comparable methods.

One of the main contributions of our method is the increase in feedback scores for simulated users whose ratings in the initial generations are more than one standard deviation lower than the median. The feedback score for our method increased from 1.75 in the initial generation, to 4.39 in the final generation, which represents an increase of 150.85%. Kim et al.'s method in-

creased feedback scores from 1.83 in the initial generation to 3.79 in the final generation, which represents an increase of 107.27%. Our method therefore outperformed Kim et al.'s method, when looking at the final feedback scores. The difference between these final scores (4.39 vs 3.79) was found to be statistically significant with a p-value of 0.0088 using a Mann-Whitney U test. This implies that the contributions of our method are statistically significant in increasing the feedback scores of simulated users who rate the first generation lowly. Because this increase in feedback scores for niche users does not hold when the mutation component of our system is turned off, we can attribute the rise in feedback scores for niche users to the inclusion of mutation. Therefore, this could imply that for real users interested in more niche items, the additions of mutation and dynamic crossover and mutation rates could also improve the recommendation quality significantly.

It is notable to mention that these improvements for niche users did not come at the cost of users that are considered "traditional". The difference between the final feedback scores for this group using our method and Kim et al.'s method (4.99 vs 4.97, respectively) was not statistically significant with a p-value of 0.8815 using the same Mann-Whitney U test. Based on these results, we can say that our method offers significant advantages for niche users, without negatively impacting users with traditional interests. This highlights the capabilities of mutation in interactive genetic algorithms for recommendations.

Even though these performance gains for niche users were substantial, it should be noted that the APLT of the recommended sets of music is similar using both our method, as well as the baseline. Our method achieved an APLT of 92.65%, whereas Kim et al. (2010)[51] achieved an APLT of 90.34%. This means that, while both methods ensure a high level of agnosticism to popularity in their recommendations, the increase in feedback scores for niche users in our method was not due to an increase in the recommendation of popular songs. This is expected, as both methods are content-based recommenders that recommend songs based on the content, and not on collaborative data.

5.1.1 Research Questions

In this section, we will go over the research questions posed in Section 3.2, and connect findings found in the results to these questions to try and answer them.

How effective is the interactive genetic algorithm recommender with mutation in increasing feedback scores over generations in general?

The results of our experiments showed that the IGA with mutation is highly effective in increasing feedback scores over generations. The average feedback score across all users increased by 61.2%, from 2.99 in the initial generation to 4.82 in the final generation. This performance in increasing is on par with previous studies, which was confirmed by statistical tests. Additionally, the convergence rate of our method was found to be superior to that of the analyzed baseline, achieving faster improvements in earlier generations.

How effective is the interactive genetic algorithm recommender with mutation in increasing feedback scores over generations for users with niche preferences?

The results of our experiment show that the inclusion of mutation significantly benefits users with niche preferences. For these users, initial feedback scores were much lower for both our method and the baseline. However, by the final generation, the feedback score for this group of users, the feedback scores for our method increased by 150.85%, as compared to 107.27% in the baseline. The difference between the feedback scores for this group of users was statistically examined, and it was found to be statistically significant, indicating that mutation is a key component for enhancing recommendation quality for niche users. Further tests where the mutation rate was set to 0 for our method confirmed these findings, as the results were comparable to those of the baseline.

How does the use of an interactive genetic algorithm with mutation impact the popularity bias of recommended music, as expressed in the Average Popularity of Long-Tail Items (APLT)?

The interactive genetic algorithm with mutation was designed to potentially address the popularity bias in music recommendations. However, the use of the IGA with mutation did not lead to a decrease nor an increase in popularity bias in the recommendations, as is evident by the APLT values found for both our method and the baseline. The APLT for our method was 92.65%, whereas the baseline had a similar APLT of 90.34%.

5.2 Comparison to existing literature

In this section, we compare the findings of our methodology to the existing literature. Our main point of comparison is Kim et al. (2010)[51]. As discussed, both methods show the ability to improve recommendations over successive generations. However, our method contains multiple improvements over the previous approaches. The most notable difference is the inclusion of mutation in the methodology. This allows the algorithm to explore the solution space more effectively, leading to improvements for users with more niche interests. Moreover, the inclusion of adaptive mutation and crossover rates allows for a more flexible evolutionary process, exploiting good solutions while still allowing the exploration of less good solutions.

Another improvement our system made is in the convergence rate in the initial generations. Our method improved the feedback scores in the initial generation with more than Kim et al.'s method. One problem identified with recommender systems is the cold start problem. IGAs could serve as a good starting point for further refinement, leading to better recommendations. The enhanced performance in the initial generations in our method as compared to Kim et al., has implications for the potential user experience. When users receive more relevant recommendations from the start, they are more likely to trust and continue using the system, as shown by Shin (2020)[99]. This could lead to higher use rates of the system, which could

lead to the exposure of a wider range of artists because of the popularity-agnostic nature of the algorithm.

Moreover, our approach aligns well with the findings discussed in Liang and Willemsen (2019)[87]. They found that recommenders can facilitate the exploration of new genres. The use of interactive genetic algorithms for this purpose is promising, as users themselves can select avenues to drive the recommendations towards. While Liang and Willemsen emphasize mixed recommendation methods to enhance the user exploration of unfamiliar genres, our method uses interactive genetic algorithms to possibly enhance this user exploration. Our method, because of the use of mutation, can be particularly suitable for this "user preference development". By incorporating tracks that may initially seem less aligned with the established user preferences, our method could facilitate more serendipitous discoveries, allowing users to develop preferences away from their original ones.

5.3 Limitations of Study

While the results of this study are promising, several limitations should be considered when interpreting them. In this section, we will highlight identified limitations, and discuss what future studies could have improved.

The use of simulated users with randomized preferences is useful for the testing and evaluation of the algorithm. However, these simulated users cannot fully represent human behavior. Real users have tastes that are personal to them, and the reasoning is not always based on the musical components of the song, but also on the feelings towards the artist, the lyrics and the cultural context. Moreover, in real-world scenarios, user preferences can change over time. These dynamic preferences are something that the algorithm needs to adapt to. Future studies could evaluate either dynamic simulated users or have real users to evaluate each generation.

In addition, the track features obtained from the Spotify API that our methodology bases its recommendations on are predominantly calculated algorithmically. Due to the lack of transparency in the way Spotify calcu-

lates these values, it is difficult to determine whether this introduces any bias toward certain items in the dataset. According to Spotify, they utilize machine learning techniques to analyze audio, where a music expert classifies some sample songs, and these rules are then extended to other songs[92]. This process could lead to biases if sample songs are not representative of the entire music catalog.

Moreover, like Kim et al. (2010)[51], the analysis in our results focused mostly on feedback scores as the main performance indicator. This is a valuable metric to consider, as it provides quantifiable data on how well the algorithm fits the user's preferences. However, this metric alone can not fully reflect satisfaction. User engagement and enjoyment are also important in gaining an understanding of the user experience. Future research could consider evaluating multiple metrics to better capture user satisfaction.

The main point of comparison to our methodology was Kim et al. (2010)[51]. However, due to the original code not being available at the time of development or writing, the code was recreated. This lack of access to the original code raises the possibility of implementation errors, where our implementation of the methodology differs from their intent. Even though we took care to recreate the methodology as closely as possible, these possible variations in our approach might influence the comparability between the two methods.

Additionally, the dataset in this research is smaller than is used in other methods. This was done due to computational constraints, and the amount of testing that needs to be done when developing a new system. However, with larger datasets, the responsiveness of the algorithm could suffer. Especially the mutation and crossover functions cost large amounts of computational power. The non-existent song that results from these functions needs to be compared to all songs in the data to find one that is most closely aligned. This means that the number of comparisons that need to be done increases linearly with the number of songs in the dataset. When considering use with real users, this larger computational power required could

elongate the time it takes to create a new generation after rating. This could hinder the applicability of the method in real-world situations.

The simulated users in this research have randomized preferences. However, in real-life scenarios, users do not have an equal chance to have a certain preference over another. Genres such as pop are assumed to have larger audiences than more niche genres such as world. This means that the simulated users in this study could be less representative of real users than simulated users who account for varying audience sizes and preferences.

5.4 Future research directions

In this section, we will discuss directions that future research could take, based on this research. With this, we identify key areas that could benefit from further exploration to expand the understanding of the findings.

One direction for future research could be incorporating user feedback from real users. This could provide insights into how the algorithm performs outside of simulated users. Conducting an evaluation of the method in a real-world environment with actual users would be beneficial in obtaining data regarding the algorithm's practical efficacy. This research used simulated users, which cannot capture the full complexity of human behavior. When using real users, the IGA's performance under real conditions can be evaluated.

Moreover, applying the IGA to other datasets, including those from different music streaming platforms or more user demographics than the LFM-1b dataset, could validate its generalisability and robustness. This would also provide insights into how the algorithm performs across various contexts. This could help identify any necessary adaptations for specific datasets or user groups. Moreover, the algorithm could also be expanded to recommend items other than music, for example for product recommendation purposes on e-commerce websites or in social media recommendation on platforms such as YouTube and TikTok, or other platforms with recommendation.

Another potentially interesting avenue for future research is the explainability of the method. With our current implementation, real users would not get any feedback on why certain recommendations were made. Developing methods that explain the reasoning behind specific recommendations could possibly enhance satisfaction, as well as increase user trust.

Finally, future research could look into a scalability analysis. Since this research was based on a relatively small dataset, an analysis of the scalability of our method could help in real-world applicability. This could be in the form of investigating how the performance changes as the dataset size increases, as well as possible optimizations of the current method, such as a more efficient data structure in the dataset, or parallel processing for the creation of a new generation.

6. Conclusion

In this paper, we presented a novel approach to an interactive genetic algorithm (IGA) for music recommendation, by including mutation in the genetic operations, as well as adding dynamic cross-over and mutation rates based on feedback. We hypothesized that these inclusions would improve the adaptability to user preferences and enhance recommendations for users with more interest in niche items. This could help overcome popularity bias and help regular users explore their tastes as well.

Our method was benchmarked against a methodology proposed by Kim et al. (2010)[51]. Our results show that the proposed IGA consistently improves feedback scores for all users, in a similar manner as the benchmark. Our method had a 61.20% increase in feedback scores, while the benchmark achieved 53.85%. However, the method is particularly effective in improving recommendations for users with interests that are considered "niche". For these users, which were categorized based on whether their feedback in the initial generation was 1 standard deviation lower than the mean, the method improved feedback scores by 150.85%, while the benchmark increased them by 107.27%. For "traditional" users, categorized based on whether the initial feedback scores were one standard deviation higher than the mean, our method did not increase nor decrease the feedback scores. Statistical analyses confirmed that, while overall the performance of both methods was similar, our method outperformed the benchmark in increasing feedback scores for "niche" users, while "traditional" user feedback scores were not significantly affected.

These results are promising, but several limitations need to be considered. Firstly, our method used simulated users to analyze the performance of our method, as well as the baseline. However, the use of simulated users does not fully capture human behavior and is an oversimplification of real-world conditions. Future work should analyze the effects of mutation in

an IGA using real users. Additionally, we focused on feedback scores as the primary metric for comparison, whereas other additional metrics might better represent user satisfaction. Additionally, the baseline used as a comparison had no code available at the time of writing. For this reason, the code was recreated, but this could bring small variations in implementation that could affect comparability. Moreover, the dataset in this study was relatively small in the context of other recommender systems research. Larger datasets could increase the time it takes to generate a new generation, as the mutation function is especially computationally intensive. Future research could look into a scalability analysis to test the effects of larger datasets on the performance of the proposed IGA.

A. Appendix

A.1 Tables

Table A.1: Mutated songs based on feedback on English Chamber Orchestra

Feedback	Title	Artist	Genres	Danceability	Energy	Valence
5	The Lake In The Moonlight (Swan Lake)	Pyotr Ilyich Tchaikovsky	['classical']	0.193	0.080	0.0384
5	Eon Blue Apocalypse	Tool	['alternative', 'rock', 'heavy metal']	0.158	0.045	0.0699
5	Mother	Believe	['rock', 'alternative']	0.201	0.037	0.0353
4	The Funeral Of A Friend	Tuxedomoon	['electronic', 'alternative', 'jazz', 'punk', 'rock']	0.285	0.250	0.0385
4	Peace	Paul Kelly	['folk', 'rock', 'pop', 'country', 'jazz', 'alternative']	0.469	0.043	0.202
4	Anak	Xuefei Yang	['classical', 'world']	0.402	0.094	0.201
3	East Hastings	Godspeed You! Black Emperor	['alternative', 'rock', 'electronic']	0.152	0.368	0.0448
3	Requiem	99RadioService	['rock']	0.088	0.0034	0.334
3	Degree Zero of Liberty	Porcupine Tree	['rock', 'alternative']	0.314	0.013	0.0861
2	Alive	Daft Punk	['alternative', 'rock', 'pop']	0.723	0.716	0.0748
2	VI. (Ysobel) - Andantino	Sir Edward Elgar	['classical']	0.150	0.0044	0.0393
2	Safeway Cart	Neil Young	['rock', 'folk', 'country', 'blues', 'alternative', 'pop']	0.703	0.224	0.135
1	The Yellow Windows of the Evening Train	Porcupine Tree	['rock', 'alternative']	0.103	0.034	0.036
1	The world is my land	Jupiter & Okwess International	['world']	0.101	0.016	0.162
1	Wildlife Analysis	Boards of Canada	['electronic', 'alternative']	0.155	0.020	0.278

Bibliography

[1] Y. Hijikata, K. Iwahama, and S. Nishida, "Content-based music filtering system with editable user profile," in *Proceedings of the 2006 ACM symposium on Applied computing*, 2006, pp. 1050–1057.

[2] P. Darshna, "Music recommendation based on content and collaborative approach & reducing cold start problem," in *2018 2nd international conference on inventive systems and control (ICISC)*, IEEE, 2018, pp. 1033–1037.

[3] H.-C. Wang, S.-W. Syu, and P. Wongchaisuwat, "A method of music autotagging based on audio and lyrics," *Multimedia Tools and Applications*, vol. 80, pp. 15 511–15 539, 2021. DOI: 10.1007/s11042-020-10381-y.

[4] P. Pirolli and S. Card, "Information foraging.," *Psychological review*, vol. 106, no. 4, p. 643, 1999.

[5] Q. Areeb, M. Nadeem, S. S. Sohail, *et al.*, "Filter bubbles in recommender systems: Fact or fallacy—a systematic review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 13, 2023. DOI: 10.1002/widm.1512.

[6] R. D. Prisco, A. Guarino, D. Malandrino, and R. Zaccagnino, "Induced emotion-based music recommendation through reinforcement learning," *Applied Sciences*, 2022. DOI: 10.3390/app122111209.

[7] T. Likhitha and S. Pulari, "Diversity in recommender systems: A closer look," *Journal of Computational and Theoretical Nanoscience*, vol. 17, pp. 222–227, 2020. DOI: 10.1166/jctn.2020.8654.

[8] S. B. Selçuklu, "Multi-objective genetic algorithms," in *Handbook of Formal Optimization*, A. J. Kulkarni and A. H. Gandomi, Eds. Singapore: Springer Nature Singapore, 2023, pp. 1–37, ISBN: 978-981-19-8851-6. DOI: 10.1007/978-981-19-8851-6_31-1. [Online]. Available: https://doi.org/10.1007/978-981-19-8851-6_31-1.

[9] K. E. Kinnear Jr, "A perspective on the work in this book," *Advances in genetic programming*, pp. 3–19, 1994.

[10] J. Bradshaw and J. Miles, "Using standard fitnesses with genetic algorithms," *Advances in Engineering Software*, vol. 28, pp. 425–435, 1997. DOI: 10.1016/S0965-9978(97)00016-1.

[11] H. Abdollahpouri, M. Mansoury, R. Burke, and B. Mobasher, "The unfairness of popularity bias in recommendation," *arXiv preprint arXiv:1907.13286*, 2019.

[12] A. Borchers, J. L. Herlocker, and J. Riedl, "Ganging up on information overload," *Computer*, vol. 31, pp. 106–108, 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:10056380.

[13] U. Shardanand, "Social information filtering for music recommendation," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.

[14] *Content-based filtering advantages & disadvantages*, Jul. 2022. [Online]. Available: https://developers.google.com/machine-learning/recommendation/content-based/summary.

[15] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Communications of the ACM*, vol. 35, no. 12, pp. 61–70, 1992.

[16] U. Shardanand and P. Maes, "Social information filtering: Algorithms for automating "word of mouth"," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1995, pp. 210–217.

[17] M. Balabanović and Y. Shoham, "Fab: Content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.

[18] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.

[19] J. Wang, A. P. De Vries, and M. J. Reinders, "Unifying user-based and item-based collaborative filtering approaches by similarity fusion," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006, pp. 501–508.

[20] L. Camacho and S. N. A. Souza, "Social network data to alleviate cold-start in recommender system: A systematic review," *Inf. Process. Manag.*, vol. 54, pp. 529–544, 2018. DOI: 10.1016/j.ipm.2018.03.004.

[21] P. Cano, O. Celma, M. Koppenberger, and J. M. Buldu, "Topology of music recommendation networks," *Chaos: An interdisciplinary journal of nonlinear science*, vol. 16, no. 1, 2006.

[22] C. C. Aggarwal, *Recommender Systems*. Jan. 2016. DOI: 10.1007/978-3-319-29659-3. [Online]. Available: https://doi.org/10.1007/978-3-319-29659-3.

[23] M. Fu, H. Qu, D. Moges, and L. Lu, "Attention based collaborative filtering," *Neurocomputing*, vol. 311, pp. 88–98, 2018. DOI: 10.1016/j.neucom.2018.05.049.

[24] C. C. Aggarwal *et al.*, *Recommender systems*. Springer, 2016, vol. 1.

[25] B. Abdollahi and O. Nasraoui, "Using explainability for constrained matrix factorization," in *Proceedings of the eleventh ACM conference on recommender systems*, 2017, pp. 79–83.

[26] K. Saini and A. Singh, "A content-based recommender system using stacked lstm and an attention-based autoencoder," *Measurement: Sensors*, vol. 31, p. 100 975, 2024.

[27] S. Lin, C. Gao, J. Chen, *et al.*, *How do recommendation models amplify popularity bias? an analysis from the spectral perspective*, 2024. arXiv:

2404.12008 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2404.12008.

[28] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, vol. 237, pp. 82–117, 2013. [Online]. Available: %5E1%5E.

[29] J. E. Bell and P. McMullen, "Ant colony optimization techniques for the vehicle routing problem," *Adv. Eng. Informatics*, vol. 18, pp. 41–48, 2004. DOI: 10.1016/j.aei.2004.07.001.

[30] P. Saurabh and B. Verma, "An efficient proactive artificial immune system based anomaly detection and prevention system," *Expert Syst. Appl.*, vol. 60, pp. 311–320, 2016. DOI: 10.1016/j.eswa.2016.03.042.

[31] M. H. Yar, V. Rahmati, and H. D. Oskouei, "A survey on evolutionary computation: Methods and their applications in engineering," *Mathematical Models and Methods in Applied Sciences*, vol. 10, p. 131, 2016. DOI: 10.5539/MAS.V10N11P131.

[32] H. Li, H. He, and Y. Wen, "Dynamic particle swarm optimization and k-means clustering algorithm for image segmentation," *Optik*, vol. 126, no. 24, pp. 4817–4822, 2015, ISSN: 0030-4026. DOI: https://doi.org/10.1016/j.ijleo.2015.09.127. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030402615011961.

[33] H. Maaranen, K. Miettinen, and A. Penttinen, "On initial populations of a genetic algorithm for continuous optimization problems," *Journal of Global Optimization*, vol. 37, pp. 405–436, 2007. DOI: 10.1007/S10898-006-9056-6.

[34] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia tools and applications*, vol. 80, pp. 8091–8126, 2021.

[35] A. Dockhorn and S. Lucas, "Choosing representation, mutation, and crossover in genetic algorithms," *IEEE Computational Intelligence Magazine*, vol. 17, no. 4, pp. 52–53, 2022.

[36] T.-P. Hong, H.-S. Wang, W.-Y. Lin, and W.-Y. Lee, "Evolution of appropriate crossover and mutation operators in a genetic process," *Applied intelligence*, vol. 16, pp. 7–17, 2002.

[37] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015.

[38] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1975.

[39] C. Onnen, R. Babuška, U. Kaymak, J. Sousa, H. Verbruggen, and R. Isermann, "Genetic algorithms for optimization in predictive control," *Control Engineering Practice*, vol. 5, pp. 1363–1372, 1997. DOI: 10.1016/S0967-0661(97)00133-0.

[40] F. Allen and R. Karjalainen, "Using genetic algorithms to find technical trading rules," *Journal of financial Economics*, vol. 51, no. 2, pp. 245–271, 1999.

[41] A. Abraham, "Meta learning evolutionary artificial neural networks," *Neurocomputing*, vol. 56, pp. 1–38, 2004.

[42] S. B. Rout, S. Mishra, and D. K. Swain, "Application of genetic algorithm in various bioinformatics problems," *International Journal of Innovative Research in Technology*, vol. 4, no. 9, 2018.

[43] R. Vanhaesebroeck, "Music recommendation using genetic programming," 2020.

[44] B. Alhijawi and Y. Kilani, "A collaborative filtering recommender system using genetic algorithm," *Inf. Process. Manag.*, vol. 57, p. 102 310, 2020. DOI: 10.1016/j.ipm.2020.102310.

[45] O. Stitini, S. Kaloun, and O. Bencharef, "An improved recommender system solution to mitigate the over-specialization problem using genetic algorithms," *Electronics*, vol. 11, no. 2, 2022, ISSN: 2079-9292. DOI: 10.3390/electronics11020242. [Online]. Available: https://www.mdpi.com/2079-9292/11/2/242.

[46] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*, vol. 1, Elsevier, 1991, pp. 69–93.

[47] W. Wieczorek and Z. Czech, "Selection schemes in evolutionary algorithms," in *Intelligent Information Systems 2002. Advances in Soft Computing* (Advances in Soft Computing), M. Kłopotek, S. Wierzchoń, and M. Michalewicz, Eds., Advances in Soft Computing. Heidelberg: Physica, 2002, vol. 17. DOI: https://doi.org/10.1007/978-3-7908-1777-5_19.

[48] J. Zhong, X. Hu, J. Zhang, and M. Gu, "Comparison of performance between different selection strategies on simple genetic algorithms," in *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)*, IEEE, vol. 2, 2005, pp. 1115–1121.

[49] H. M. Pandey, A. Shukla, A. Chaudhary, and D. Mehrotra, "Evaluation of genetic algorithm's selection methods," in *Information Systems Design and Intelligent Applications: Proceedings of Third International Conference INDIA 2016, Volume 2*, Springer, 2016, pp. 731–738.

[50] S.-B. Cho, "Towards creative evolutionary systems with interactive genetic algorithm," *Applied Intelligence*, vol. 16, pp. 129–138, 2002.

[51] H.-T. Kim, E. Kim, J.-H. Lee, and C. W. Ahn, "A recommender system based on genetic algorithm for music data," in *2010 2nd International Conference on Computer Engineering and Technology*, IEEE, vol. 6, 2010, pp. V6–414.

[52] H.-T. Kim, J.-H. Lee, and C. W. Ahn, "A recommender system based on interactive evolutionary computation with data grouping," *Procedia Computer Science*, vol. 3, pp. 611–616, 2011.

[53] H.-T. Kim and C. W. Ahn, "An interactive evolutionary approach to designing novel recommender systems," *Int. J. Physical Sciences*, vol. 7, no. 15, pp. 2327–2338, 2012.

[54] H.-T. Kim, J. An, and C. W. Ahn, "A new evolutionary approach to recommender systems," *IEICE TRANSACTIONS on Information and Systems*, vol. 97, no. 3, pp. 622–625, 2014.

[55] T. Horváth and A. C. de Carvalho, "Evolutionary computing in recommender systems: A review of recent research," *Natural Computing*, vol. 16, pp. 441–462, 2017.

[56] V. Kant and K. K. Bharadwaj, "A user-oriented content based recommender system based on reclusive methods and interactive genetic algorithm," in *Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012) Volume 1*, Springer, 2013, pp. 543–554.

[57] T. Saito and E. Sato-Shimokawara, "Music recommender system considering the variations in music selection criterion using an interactive genetic algorithm," in *International Conference on Computer Information Systems and Industrial Management*, Springer, 2023, pp. 382–393.

[58] Q. Li, S.-H. Myaeng, and B.-M. Kim, "A probabilistic music recommender considering user opinions and audio features," *Inf. Process. Manag.*, vol. 43, pp. 473–487, 2007. DOI: 10.1016/j.ipm.2006.07.005.

[59] A. Jamdar, J. Abraham, K. Khanna, and R. Dubey, "Emotion analysis of songs based on lyrical and audio features," *ArXiv*, vol. abs/1506.05012, 2015. DOI: 10.5121/ijaia.2015.6304.

[60] D. Wang, S. Deng, X. Zhang, and G. Xu, "Learning music embedding with metadata for context aware recommendation," in *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, 2016, pp. 249–253.

[61] A. Lozano Murciego, D. M. Jiménez-Bravo, A. Valera Roman, J. F. De Paz Santana, and M. N. Moreno-García, "Context-aware recommender systems in the music domain: A systematic literature review," *Electronics*, vol. 10, no. 13, p. 1555, 2021.

[62] W.-P. Lee, C.-T. Chen, J.-Y. Huang, and J.-Y. Liang, "A smartphone-based activity-aware system for music streaming recommendation," *Knowl. Based Syst.*, vol. 131, pp. 70–82, 2017. DOI: 10.1016/j.knosys.2017.06.002.

[63] G. Jawaheer, M. Szomszor, and P. Kostkova, "Comparison of implicit and explicit feedback from an online music recommendation service," in *proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems*, 2010, pp. 47–51.

[64] R. Katarya and O. P. Verma, "Efficient music recommender system using context graph and particle swarm," *Multimedia Tools and Applications*, vol. 77, pp. 2673–2687, 2018.

[65] K. Gupta, N. Sachdeva, and V. Pudi, "Explicit modelling of the implicit short term user preferences for music recommendation," in *Advances in Information Retrieval: 40th European Conference on IR Research, ECIR 2018, Grenoble, France, March 26-29, 2018, Proceedings 40*, Springer, 2018, pp. 333–344.

[66] I. Cross and G. E. Woodruff, "77Music as a communicative medium," in *The Prehistory of Language*, Oxford University Press, Apr. 2009, ISBN: 9780199545872. DOI: 10.1093/acprof:oso/9780199545872.003.0005. eprint: https://academic.oup.com/book/0/chapter/162797023/chapter-ag-pdf/44910607/book_12719_section_162797023.ag.pdf. [Online]. Available: https://doi.org/10.1093/acprof:oso/9780199545872.003.0005.

[67] C. Bauer and M. Schedl, "Global and country-specific mainstreaminess measures: Definitions, analysis, and usage for improving personalized music recommendation systems," *PloS one*, vol. 14, no. 6, e0217389, 2019.

[68] P. Müllner, "Studying non-mainstream music listening behavior for fair music recommendations," Ph.D. dissertation, Ph. D. Dissertation. Graz University of Technology, 2019.

[69] C. Anderson, "The long tail," en-US, *WIRED*, 2004. [Online]. Available: https://www.wired.com/2004/10/tail/.

[70] D. Kowald, M. Schedl, and E. Lex, "The unfairness of popularity bias in music recommendation: A reproducibility study," in *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part II 42*, Springer, 2020, pp. 35–42.

[71] J. H. Lee, B. Bare, and G. Meek, "How similar is too similar?: Exploring users' perceptions of similarity in playlist evaluation.," in *ISMIR*, vol. 11, 2011, pp. 109–114.

[72] Y. Song, S. Dixon, and M. Pearce, "A survey of music recommendation systems and future perspectives," in *9th international symposium on computer music modeling and retrieval*, Citeseer, vol. 4, 2012, pp. 395–410.

[73] J. Bland and D. G. Altman, "Statistics notes: Some examples of regression towards the mean," *BMJ*, vol. 309, p. 780, 1994. DOI: 10.1136/BMJ.309.6957.780.

[74] M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, and M. Elahi, "Current challenges and visions in music recommender systems research," *International Journal of Multimedia Information Retrieval*, vol. 7, pp. 95–116, 2018.

[75] E. Blake, "Data shows 90 percent of streams go to the top 1 percent of artists," en-US, *Rolling Stone*, Sep. 2021. [Online]. Available: https://www.rollingstone.com/pro/news/top-1-percent-streaming-1055005/.

[76] A. J. Chaney, B. M. Stewart, and B. E. Engelhardt, "How algorithmic confounding in recommendation systems increases homogene-

ity and decreases utility," in *Proceedings of the 12th ACM conference on recommender systems*, 2018, pp. 224–232.

[77] O. Lesota, A. Melchiorre, N. Rekabsaz, *et al.*, "Analyzing item popularity bias of music recommender systems: Are different genders equally affected?" In *Proceedings of the 15th ACM Conference on Recommender Systems*, 2021, pp. 601–606.

[78] H. Abdollahpouri, M. Mansoury, R. Burke, B. Mobasher, and E. Malthouse, "User-centered evaluation of popularity bias in recommender systems," in *Proceedings of the 29th ACM Conference on User Modeling, Adaptation and Personalization*, 2021, pp. 119–129.

[79] H. Abdollahpouri, R. Burke, and B. Mobasher, "Managing popularity bias in recommender systems with personalized re-ranking," *arXiv preprint arXiv:1901.07555*, 2019.

[80] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *Proceedings of the 14th international conference on World Wide Web*, 2005, pp. 22–32.

[81] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 689–698.

[82] S. Karboua, F. Harrag, F. Meziane, and A. Boutadjine, "Mitigation of popularity bias in recommendation systems," in *Tunisian-Algerian Joint Conference on Applied Computing*, 2022.

[83] R. Borges and K. Stefanidis, "On mitigating popularity bias in recommendations via variational autoencoders," in *Proceedings of the 36th annual ACM symposium on applied computing*, 2021, pp. 1383–1389.

[84] A. Krishnan, A. Sharma, A. Sankar, and H. Sundaram, "An adversarial approach to improve long-tail performance in neural collaborative filtering," in *Proceedings of the 27th ACM International Conference on information and knowledge management*, 2018, pp. 1491–1494.

[85] A. Antikacioglu and R. Ravi, "Post processing recommender systems for diversity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 707–716.

[86] H. Abdollahpouri, M. Mansoury, R. Burke, and B. Mobasher, "Addressing the multistakeholder impact of popularity bias in recommendation through calibration," *arXiv preprint arXiv:2007.12230*, 2020.

[87] Y. Liang and M. C. Willemsen, "Personalized recommendations for music genre exploration," in *Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization*, ser. UMAP '19, Larnaca, Cyprus: Association for Computing Machinery, 2019, pp. 276–284, ISBN: 9781450360210. DOI: 10.1145/3320435.3320455. [Online]. Available: https://doi.org/10.1145/3320435.3320455.

[88] M. Schedl, "The lfm-1b dataset for music retrieval and recommendation," in *Proceedings of the 2016 ACM on international conference on multimedia retrieval*, 2016, pp. 103–110.

[89] M. Schedl and B. Ferwerda, "Large-scale analysis of group-specific music genre taste from collaborative tags," in *2017 IEEE International Symposium on Multimedia (ISM)*, IEEE, 2017, pp. 479–482.

[90] D. Nosowitz, "The story of allmusic, which predates the world wide web," en, *Vice*, Jan. 2015. [Online]. Available: https://www.vice.com/en/article/ypwezy/the-internets-most-complete-guide-to-music.

[91] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann, "Easy access to the freebase dataset," in *Proceedings of the 23rd international conference on World Wide Web*, 2014, pp. 95–98.

[92] E. V. Buskirk, *Plotting music's emotional valence, 1950-2013*, Spotify Insights, Accessed: 2024-07-28, 2013. [Online]. Available: https://web.archive.org/web/20160324094747/https://insights.spotify.com/us/2013/11/05/musics-emotion-over-time/.

[93] M. D. Barone, J. Bansal, and M. Woolhouse, "Acoustic features influence musical choices across multiple genres," *Frontiers in Psychology*, vol. 8, 2017. DOI: 10.3389/fpsyg.2017.00931.

[94] Jupiter & Okwess, *Official website: Biography (english)*, n.d. [Online]. Available: https://jupiterandokwess.fr/#bio.

[95] C. Karnop, "Prediction of audio features of self-selected music by situational and person-related factors," 2019.

[96] G. Hardman and P. R. Talarczyk, "The effects of odd time signatures on pop song enjoyment," *Journal of Student Research*, 2021. DOI: 10.47611/jsrhs.v10i4.1968.

[97] H. Haghbayan, E. Coomes, and D. Curran, "Temporal trends in the loudness of popular music over six decades," *Journal of General Internal Medicine*, vol. 35, pp. 394–395, 2019. DOI: 10.1007/s11606-019-05210-4.

[98] Aug. 2021. [Online]. Available: https://www.masterclass.com/articles/music-101-what-is-tempo-how-is-tempo-used-in-music.

[99] D. Shin, "How do users interact with algorithm recommender systems? the interaction of users, algorithms, and performance," *Computers in human behavior*, vol. 109, p. 106 344, 2020.