A Genetic Algorithm Approach to Recommender System Cold Start Problem

by

Sanjeevan Sivapalan

Bachelor of Science, Ryerson University, 2011

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2015

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my dissertation may be made electronically available to the public.

A Genetic Algorithm Approach to Recommender System Cold Start Problem

Master of Science 2015

Sanjeevan Sivapalan

Computer Science

Ryerson University

Abstract

Recommender systems (RS) are ubiquitous and used in many systems to augment user experience to improve usability and they achieve this by helping users discover new products to consume. They, however, suffer from cold-start problem which occurs when there is not enough information to generate recommendations to a user. Cold-start occurs when a new user enters the system that we don't know about. We have proposed a novel algorithm to make recommendations to new users by recommending outside of their preferences. We also propose a genetic algorithm based solution to make recommendations when we lack information about user and a transitive algorithm to form neighbourhood. Altogether, we developed three algorithms and tested them using they MovieLens dataset. We have found that all of our algorithms performed well during our testing using the offline-evaluation method.

Acknowledgements

Finally I'm done!

First and foremost, I would like to thank my supervisor, Dr. Alireza Sadeghian, for all his help and guidance he has provided me with during my thesis. Without his help, I could not have completed my thesis. Dr. Sadeghian was always supportive of my ideas and he worked with me to ensure the successful application of these methodologies whenever possible.

I'm grateful for the support my family gave me during my studies. They were instrumental in me starting my MSc thesis and I can't thank them enough.

I would like to thank my committee members Dr. Ding, Dr. Hamelin and Dr. Woungang for taking their time to make sure my thesis was free of errors. Their questions and comments have helped to create a better thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem and Motivation

The cold start problem is one of the common popular problems in the field of recommender systems. The problem occurs when a new user is introduced into the system. The system gathers user preferences by ratings and item consumption patterns. However, in the beginning this information is not available and as a consequence, the recommendation process fails. There are various solutions designed to solve this issue but they give very generalized recommendations. Most solutions categorize the user based on demographics or recommend top items from the system. This is acceptable if the user does not have any activities in the system. However, most solutions take this approach even if the user has shown some interest in particular field by filling out a survey or giving ratings. They also tend to over-specialize and recommend the same type of items to the user. This prevents in the user actively searching for novel items which defeats the purpose of recommendation systems.

Our approach is to make versatile recommendation system that can recommend to users based on a single rating. We have implemented algorithms based on the notion that recommender systems should make it easier for users to discover items that are outside their immediate preferences. That is, we make recommendations based on known preferences of the user, but fall outside their preferred areas. This will allow us to make more recommendations early and learn more about the user's preferences. Our algorithm will learn and evolve the user preferences easily in the early stages. We also wish to alleviate over-specialization in the beginning due to lack of preference data about the user.

1.2 Methodology

Our novel algorithm is focused on automatically generating recommendations after user rates one item. We analyze the item rated to find the categories it belongs to. We then analyze the rating that user has given. We apply one of three different algorithms depending on the rating given. The first algorithm takes effect if the user has given a high rating to a popular item. That is, we have other users in the system who gave the item similar ratings. These users form the neighbours of the active users. Afterwards, all the highly rated items of the neighbours are taken. The items are then filtered to remove anything that might fall in the same category as the item the active user rated and presented.

The second algorithm is used when the user gives a high rating to an unpopular item where users have given it low rating. We first categorize the items and form a neighbourhood of items that fall in the same category. Then, we find user who have rated them and use them as the neighbours. The last step is to find items rated by the neighbours which are not in the same category and presented to the user.

The third algorithm is an evolutionary algorithm which is applied where the user gives a low rating to a popular item. In order to handle this rare case, we create a preference vector for the active user. The vector is then mutated and evolved through many generations. This would help us to make it easier to form neighbourhood. After the neighbourhood formation, we make recommendations without items from the same category as the item user rated.

1.3 Significance

The significance of our algorithms and ideas lie in its ability to automatically generate recommendations for a new user. We take away the hassle of user having to answer a questionnaire which is both time consuming and counters the idea of recommendation engines. The user should not tell the system what they want, rather the system should figure that out. We accomplish this through a myriad of approaches. We also do not fall in the routine of making very generic recommendations based on the user demographics. The algorithms we have developed personalizes the recommendations to user's preferences. We also avoid giving top N recommendations that most systems tend to do for new users. Also, we handle very specific case where user gives negative ratings to an item which most systems do not handle.

1.4 Contributions and Outline

We have developed a novel idea to make recommendations to user using only one rating during the early days of the user in the system.

Our main contributions is the novel algorithm which recommends users outside their areas of interest in the early stages. We make recommendations to the user by forming neighbourhood of users based on an item they like but we remove items that are part of user's interest area. We also implemented three algorithms to handle the user as well.

The three algorithms we have developed to solve cold-start problem for new users are:

1. Generate recommendations of items rated by users who have rated an item similarly to active user
2. Generate recommendations of items rated by users how have rated similar items as the active user's rated item
3. Generate recommendations of items rated by users who have similar preference as the active user

We also have two other contributions to help tackle the scenarios where user does not have immediate neighbours. The first contribution is the transitive approach to forming neighbours. The algorithm we have developed first categories the items rated. Then it finds items which also falls into any of the categories. Then, neighbourhood is formed containing the users who rated the items. These users are used to then make recommendations. The second contribution is the evolutionary algorithm to form neighbourhood. In the case where we cannot reliably form neighbourhood due to taste differences, we construct preference vector for the active user. The vector is mutated throughout generations to match the preferences of other users in system. Then, we get users who have similar preference to form the neighbourhood.

1.5 Outline

The thesis is divided into 5 main chapters. The first chapter gives a basic overview of the thesis. In the second chapter, we give a brief introduction to recommendation systems. We describe various recommending algorithms and different categories of recommendations that can be made. We also give brief introduction to genetic algorithms and their role in recommendation systems. In the third chapter,

we focus on our methodology and explain our algorithms in details with giving examples. The fourth chapter contains the implementation and experiment results. We run our algorithms and evaluate their performance using well known datasets and evaluation methods. Then in the fifth chapter, we conclude with mentioning the limitations of the system and give insights on future work.

Chapter 2

Literature Review

Recommender systems allow automating the task of recommending or exposing new and existing items to a user who has not viewed them before[1]. There are different algorithms and techniques recommender systems use to generate recommendations. The most popular ones are collaborative filtering which takes other users in the system in consideration and content based filtering which relies solely on the items.

2.1 Background

2.1.1 Data utilized

Recommender systems require data to make informed decisions and to generate recommendations. Information that is often used to make recommendation decisions include user demographics, item attributes, and user preferences[1, 60]. A table of common properties used in the process are given in table 2.1. User demographics are attributes of users in general which can affect the results of recommendations that are made. This information includes amongst other things such as likes and dislikes of a particular gender, age group, occupation, income level, and hobbies. Item attributes are classified as either extrinsic or intrinsic. Extrinsic features cannot be easily identified by analyzing the contents automatically. Intrinsic features on the other hand are easily obtainable from the contents. Intrinsic features include behaviour data and how the user uses the system without consuming any items or giving ratings. Sometimes, features of items are obtained from the description of the items themselves in addition of analyzing the

Table 2.1: Data types used in recommendation systems [60]

Data Type	Description
Rating Data	rating scores, such as discrete multi-levels ratings and continuous rating; and latent comments, such as best, good, bad, worse
Behaviour Pattern Data	duration of browsing, click times, the links of webs; save, print, scroll, delete, pen, close, refresh of webs; selection, edition, search, copy, paste, bookmark and even download of web content
Transaction Data	purchasing date, purchase quantity, price, discounting
Production Data	for movies or music, it means actor or singer, topic, release time, price, brand and so on, while for webs or documents, it means content description using key words, the links to others, the viewed times, the topic

item. This is evident in cases such as news articles or web pages. User preferences are either a presence score such as likes it or dislikes, or numerical score indicating how much the user likes the item. In some cases, they are explicit ratings provided by users when they are asked to rate the item. The amount of time a user spends on a particular page, reading and analyzing are implicit indicators of the users ratings [10]. Implicit indicators are more difficult to gather because it is hard to determine the user's preference automatically, but they offer more information about the user that the user otherwise might not provide.

2.1.2 Knowledge Discovery

Knowledge discovery in databases (KDD) also referred to as Data Mining is used to describe the process of extraction of useful information, patterns or knowledge from a dataset [41, 48]. The process of knowledge extraction from raw data follows a series of data transformation and analysis as shown in figure 2.1. The raw data gathered are aggregated from many sources into a data warehouse. The information gathered might be in multiple different formats and do not need to have a defined structure. The next step is to clean up the data, and to format the data in a way that is easily understood by the system and is manageable. This involves creating new data structures and models to represent them. The third step involves applying pattern analysis, and knowledge extraction techniques on the transformed data set. This yields information that is of interest. Association rules are a popular method of finding relationships

in data mining and recommender systems. They follow an if-then pattern to make predict what can happen based on previous data set. This is very useful in recommenders because it allows us to predict what a user might like based on their previous actions[35].

KDD techniques are used to find ways to improve efficiency of platform which leads to money saving, and to find new ways to provide more items to customers to consume [46, 50]. Companies that utilize KDD can find patterns in user usage behaviours such as what time of year certain items are more likely to be bound and make recommendations on that resulting in increased revenue [8]. One of the most prominent ways algorithms used in KDD which is used in recommendation engines is the association rules. The rules try to associate a set of items to a different set of items in such way that the presence of one item from a set implies there is a high chance of another item being in the same set [46]. Well known associate algorithms include Apriori, Direct Hashing and Pruning (based on Apriori), Tree Projection algorithms, and FP-tree algorithms.

2.1.3 Types of Recommendations

Recommender systems can be categorized as personalized, non-personalized, attribute-based, item-to-item correlation, and people-to-people correlation. Recommendations are either short-lived or long-lived depending on the implementation. The system is considered automatic if it requires minimal or no input from the active user and manual if it requires some work [50].

Personalized recommendations are automatic and based on the users preferences such as favourite colour, movie genre and music group. They are often compared against hand-picked items by content-providers and experts for users preferences and tastes to provide recommendations [51].

Non-personalized recommenders generate recommendations based only on item ratings from other users of the system [51]. These recommendations are straight forward since they require very little effort to produce and considered automatic since user input is not required [50]. These recommendations are not short-lived since they can be applied to a variety of users.

Items can be described using various features, and attributes which are used to generate recommendations. This method is considered manual since the user must explicitly search for a certain type of item to base the recommendations on [50, 51]. These recommendations can be short-lived or not depending on how long the system remembers users preferences for.

Item-to-item correlation recommenders recommend items based other items the user has displayed

interest in. These recommendations are prevalent in e-commerce sites where new items are recommended based on what the user has in their shopping cart [50]. These recommendations are manual since user must have a non-empty cart, and short-lived because the user does not have full shopping carts. Association rules are most often used in this system [3].

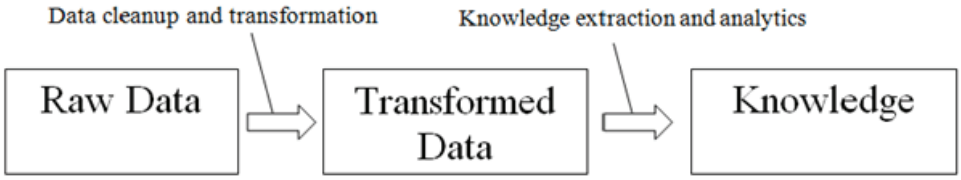

Figure 2.1: Overview of KDD process [48]

2.2 Collaborative Filtering

Collaborative filtering approach uses customer details, ratings, and reviews aggregated from all the users in the system to build recommendations for a user [1, 9, 41]. It was first introduced in 1992 with the idea that getting user feedback when making suggestions is effective[17]. The strength of this approach is that it analyzes existing active users with similar preferences and characteristics of the current user to build the recommendations. Collaborative filtering uses ratings users have given in the past to items using shown in equation 2.1 [2] using a rating function *f()* to make predictions for the future. The parameters *user* and *item* are passed to the function to get the rating the user gave to the item. The filtering method is achieved through a heuristic-based, a model-based method, or a hybrid model that combines characteristics from both heuristic and model-based approaches [60, 46, 5].

$$rating = f(user, item) \tag{2.1}$$

2.2.1 Heuristic based filtering

The heuristic model also called memory-based collaborative filtering model takes in rating information and how often the item was viewed (or not viewed) to make recommendations[60, 9]. The model uses the active user's information to find similar uses. This is done by selecting all the users who are neighbours of

the current user using similarity measures including personal information, rating information, and other user demographics [60]. Then, utilizing k-nearest neighbour classification method, prediction value is computed for each item that current user has not viewed but the other users have and rated. With the newly calculated set, recommendation is created based on items with the highest scores. There are many different algorithms and technique that can be used in heuristic based collaborative filtering includes k-nearest neighbour algorithm, decision trees, graph theory, and support vector machines [60].

As shown in figure 2.3 which describes the process, in the representation stage, a matrix R of size NxM is constructed for N users and M items in the database where $R_{i,j}$ is one if the i-th customer bought j-th item and zero otherwise. The matrix is called original representation [46]. Collaborative filtering has challenges with sparsity, scalability and synonymy. Synonymy occurs because similar items are labelled with different names in real life, and the recommender systems cannot always make association between these items and treat them as different. A reduced dimensional representation is constructed to alleviate the weaknesses. A matrix of size n x k is constructed where all the values in the matrix are non-zero, which implies that each user has had an association with the k item. Due to the decreased size, it also helps alleviate the problem with synonymy.

The neighbourhood formation, the second step, forms the heart of the recommendation system. In this step, the similarities between users are computed and used to create proximity based neighbourhood between the target customer and like-minded users [47]. For each user u and N users where $N = \{N_1, N_2, ..., N_l\}$, the user u does not belong to set of N and the similarity $sim(u, N_k)$ is greater than $sim(u, N_{k+1})$ with $sim(u, N_1)$ being the maximum. The function $sim(u, N_k)$ is a any similarity function that can be used to calculate the similarity(or proximity) between a user u and another user *Nk*. Proximity measures of two users can be calculated using Pearson correlation as shown in equation 2.2 or Cosine measure equation 2.3.

$$corr_{ab} = \frac{\sum_i (r_{ai} - \bar{r_a})(r_{bi} - \bar{r_b})}{\sqrt{\sum_i (r_{ai} - \bar{r_a})^2}\sqrt{\sum_i (r_{bi} - \bar{r_b})^2}} \tag{2.2}$$

In equation 2.2, using the Pearson correlation we can calculate the correlation between two different variables in terms of how the variables are related. Pearson correlation shows how well two data sets fit on a chart [52]. This is very useful to see the similarities and differences between two different users. The correlation between user a and b is defined as the summation over the items for which both user a

and b have voted [9, 47]. The notations r_{ai} and r_{bi} represent the rating given to i-th item by user A, and user B respectively. In the equation, r_a and r_b represent the averages. The result is between -1 and 1 with -1 being a perfect negative correlation. However, one of the limitations of Pearson correlation is that it is limited to the users having rated same item which is not always available at any given time [30].

$$\cos(\theta) = \frac{\overrightarrow{\mathbf{A}} \cdot \overrightarrow{\mathbf{B}}}{||\overrightarrow{\mathbf{A}}|| \times ||\overrightarrow{\mathbf{B}}||} \tag{2.3}$$

Alternatively, we can also measure the similarity between two users. To find the similarity of first and second user, we can use equation 2.3 to calculate cosine measure where a is the first user, and b is the second user. By treating each user ratings as a vector in NXN dimensional plane of N users and N item ratings given by the user, we can calculate the cosine of the angle between them. The lower the angle, the more similar the users are [56]. Cosine measure calculates the angle between two vectors. In this case, a vector describes all the ratings a user has given. By calculating the angle between them, we can see how similar they are. The closer the angle between the vectors is to zero, the closer the preferences of both users are as shown in figure 2.2.

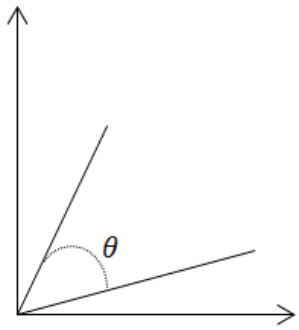

Figure 2.2: Angle between two user vectors

For n users , a similarity matrix S of size $n \times m$ is computed using either one of the proximity measures where each cell indicates the level of similarity between both users.

There are two methods to forming a neighbourhood: centre-based and aggregate neighbourhood [46]. Centre based techniques form a neighbourhood for a user c of size k by selecting l nearest users where both k and l are chosen arbitrary based on the number fo users to select from. Aggregate neighbourhood

creates a neighbourhood of size l for a user c by selecting the closest user by preference. The rest of the $l-1$ neighbours are selected similarly. At a certain point j, when there are j neighbours in N and $j < l$, the centroid of N, C is calculated using equation 2.4. Then a new user w who is not in N is selected as the j+1th if w is the closest to the centroid C. The centroid is then recomputed for $j+1$ neighbour and continues until the number of neighbours in N is l.

$$\overrightarrow{\mathbf{C}} = \frac{1}{j} \sum_{\overrightarrow{v} \in N} \overrightarrow{V} \tag{2.4}$$

The final part of the recommendation system is to make the actual recommendations which is to calculate top m recommendations from the computed neighbourhood of customers. Two prominent techniques that are used are most-frequent item recommendation, and association rule-based recommendations [46].

The first method of generating recommendations is the most frequent item generation. Neighbourhood N is scanned frequency count of purchases is calculated for each neighbour. All the items are then sorted according to the frequency and m most frequently bought items that is not purchased by the current customer are recommended [46]. The second approach uses association rules to generate the recommendations, L neighbours taken into account while using association rules to generation recommendations. Association rules work by recommending an item that a neighbour bought with the presence of another item[30]. However, having a limited number of neighbours to work limits the effectiveness of the recommendations made [46].

2.2.2 Model based filtering

The model based collaborative filtering method uses training data such as the active users ratings and reviews to build a model using different data mining and machine learning algorithms [60, 9]. The model is then validated using the testing data and list of items and rating is predicted for them if customers have not given any rating to it yet or been exposed to it. While the heuristics based model uses the entire database and the customers to create recommendations for the active customer, the model based approach only relies on the active customers information as the input. Techniques and algorithms from fields such as Bayesian model, clustering, association rules, artificial neural networks, linear regression, maximum entropy, latent semantic analysis, and Markov process can be used [60].

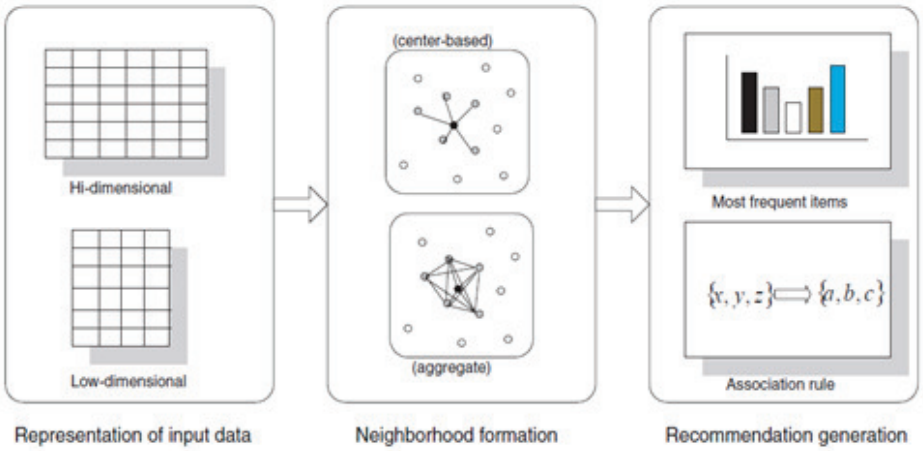

Figure 2.3: Process of generating recommendations [46]

2.2.3 Limitations

Collaborative filtering has a major disadvantage since it requires data to exist in order to be useful. It has three major limitations which are sparsity, scalability, and cold start [46, 60]. Sparsity occurs in large systems where not all items have been viewed or not all users have any ratings[41, 47]. In large e-commerce sites like Amazon and CDNow, active customers cannot easily purchase items such that they buy even 1% percent of the stores items. A recommender system that uses nearest neighbour algorithms is ill suited to make recommendations for an active user in those sites. This is commonly known as reduced coverage. It also leads to poor recommendations due to lack of enough data [46]. Nearest neighbour algorithms grow with the number of customers and items available, thus leading to scalability issues. Collaborative filtering tends to recommend items that are popular due to them having a much higher ratings than other items ignoring less popular or new items [1, 19]. In a recommender system, popular items are recommended more, and acquire better ratings. This leads to lack of diversity in the system because newer or items with fewer ratings are usually not recommended over the popular ones. Since CF works by finding users who have similar preference as the active user, it also suffers from cold start. This occurs when there is not much information available to predict the preferences of the user and make recommendations[1, 57].

2.3 Content-based filtering

Content-based filtering is based on being able to analyze items and find similarity with active user to recommend items. Unlike collaborative filtering or association rules, this method does not require an active database of purchase history.

2.3.1 Technique

Content based filtering approach is based on information retrieval, analysis and filtering [1, 60]. This approach is used mainly in places where content can be read or analyzed such as news articles, movies and anything with metadata attached. It also gives recommendations based on items the user has viewed in the past. The contents can be described using labels and the labels are given a weight of how well they describe the article. Using these labels and user preferences, nearest neighbour or clustering algorithms can be used to recommend other articles to the active user. However, new users with limited information and limited number of labels pose a challenge to this method [25]. Common algorithms that are applicable include k-nearest neighbour, clustering, Bayesian, and artificial neural networks [60].

Content-based filtering is conceptualized from information retrieval and filtering [43, 7]. Information filtering systems are usually used with structured data that can be easily analyzed to gain insights. Vast amounts of data are usually analyzed by filtering systems to give recommendations because it is per user profile [1, 7]. The user profiles are obtained explicitly through questionnaires and forms or implicitly using behavioural information [11]. A set of attributes describing a user is computed and they are then used to make recommendations to the user. The attributes are compared with keywords describing the recommendations as mentioned.

Keywords used to make recommendations are weighted using term frequency/inverse document frequency (TF-IDF) method to measure importance [32, 43]. Term frequency TF is calculated from N items that could be potentially recommended to user as shown in equation 2.5 where $f_{i,j}$ is the number of times keyword k_i appears in document d_j and computed maximum $f_{z,j}$ is the frequencies of all keywords k_z that appear in document d_j. Keywords that appear in many different documents are not useful when distinguishing between relevant and irrelevant documents. To do that, inverse document frequency is

used [1].

$$TF_{i,j} = \frac{f_{i,j}}{max_{z,f_{z,j}}} \tag{2.5}$$

Inverse document frequency IDF is calculated for keyword ki as shown in equation 2.6

$$IDF_i = log(\frac{N}{n_i}) \tag{2.6}$$

Then we can simply get the weight $w_{i,j}$ for keyword k_i in document d_j as shown in equation 2.7

$$w_{i,j} = TF_{i,j} \times IDF_i \tag{2.7}$$

Content-based filtering systems also recommend new items based on what the user had liked previously [1]. A content based profile can be constructed for a user from their previously liked items, ratings, search keywords, and other behavioural data. This information is aggregated to create a profile for the user.

2.3.2 Limitations

Content base systems are highly dependent on the items being easy to analyze. In order for recommender systems to be able to generate recommendations, content must be structured and easy to parse. If this is not, then the item must be described manually [1, 32]. Another problem is being able to differentiate between a bad item and a good item based on retrieved information. A bad item using same keywords as good item will also get recommended.

Another two major drawbacks are lack of information about a user, and overspecialization. When a new user is introduced into the system, their preferences and profiles are not aggregated. The user would not have given enough ratings, and reviews to items and leads to insufficient information to generate recommendations [1, 42]. When the system is only able to recommend certain items based on users profile, it leads to overspecialization. This is due to the user having rated a specific item; the recommender system is only able to provide recommendations for similar items. This also leads to the user never being recommended outside of their previous ratings [1, 43]. In such cases, genetic algorithms which evolve information filtering agents to provide recommendations have been proposed. This is done

by using an iterative method where previous output is used to learn and adapt dynamically [55, 33]. Some recommender systems also filter out items that maybe very similar to previous items the user has viewed in the past by comparing the recommended items against the users history of items rated [42]. Diversity is an appreciated feature of recommender systems. It allows the user to find new items that they may otherwise have not found by chance.

2.4 Hybrid filtering

To avoid problems that exist in both content-based and collaborative filtering systems have, hybrid solutions have been proposed [1, 60]. Solutions include: implement both filtering separately and combine the results, incorporating characteristics of content-based filtering to collaborative adding characteristics of collaborative filtering to content-based filtering systems and new algorithms that incorporate both systems techniques.

2.4.1 Combining different recommender systems

This approach involves building two different recommender systems based on collaborative-based and content-based approaches. The recommendations can be generated as either a combined linearly [10]. The algorithm works by assigning by a weight to the generated recommendations per user based on how much it is relevant to the user. The recommendations are then added in order to be presented to user. The second method is to use the level of confidence each system produced for the results that are more consistent with the users past ratings and provide them to the user [59].

2.4.2 Add content-based characteristics to collaborative-based systems

Many recommender systems are implemented using collaborative-based approach with content-based user profiles generated through content-based approach [1]. The profiles are then used to find similarity between users rather than items which help the system overcome some of the sparsity-related limitations. Recommendations can be generated through collaborative filtering first. They are then compared against current user profile to determine if its interesting to the user or not and to present it [33].

Recommender systems are suited for recommending new things that the use has not rated but they also perform inefficiently when faced with lack of ratings from the user. The two problems mainly faces in

this case are cold start users, and cold start items where the user has not rated anything and an item has not received any ratings respectively [25, 57]. This limits the system for making quality recommendations. The approach first builds a user model based on the users personal interest and preferences independent of all other users in the system by analysing the items the user has rated positively in the past using a minimum support. The next step incorporates neighbourhood formation techniques from collaborative filtering by using cosine measures to calculate similarity between users which calculates the similarity as the angle between two user vectors [56, 25]. The user model is then enhanced by adding features of closest neighbours. The recommendations are generated using newly added items that match best to the new user model. This approach tries to solve new item problem and new user problem by enhancing the user model. This approach also tries overcome problems with overspecialisation where the user is recommended the same items again [32].

2.4.3 Add collaborative-based characteristics to content-based systems

Curse of dimensionality occurs when a lot of features exist per item that makes it difficult to cluster or compare them [49]. The most common approach is to use dimensionality reduction algorithm on a group of content based profiles [1]. This allows performance improvements since it reduces the amount of preferences/features that must be compared to generate the recommendations.

2.5 Evolutionary programming and algorithms

Evolutionary programming (EP) takes its roots from natural selection and evolution[27] where organisms and species evolve and adapt to their surroundings. Evolutionary programming is a way of programming in a way such that programs automatically solve problems without constantly telling the computer how to accomplish it. It also plays an integral role in machine learning and artificial intelligence because it allows computers to behave in way that is distinguishable from humans. If a computer has an activity that EP has, it would seem that humans have used intelligence to solve the problem[45]. This is very promising in a recommender system because we would get the same behaviour as getting recommended by a person using a computer.

Evolutionary programming is useful in the fields of data mining, and knowledge discovery[4, 15, 61]. Attribute selection in data mining is a difficult task due to accuracy and cost of classification and GA

has been seen as a natural solution for the problem. Genetic algorithms are also employed in solving problems in the field of machine learning[18, 24] with rule extraction in the early stages when it is not known about the problem to solve. Rules in field of knowledge discovery help us to make association with different sets of data.

2.5.1 Background information

Genetic algorithm tries to solve the problem of how can a computer solve a problem without explicitly telling how to solve[44]. Genetic algorithms take inspiration from evolution and natural genetics and tries to apply them to computing. This allows us to rather specify the structure to solving problem, we let GP create the structure[58] to solve. Organisms in nature are always fighting for resources, adapting to environments, and improving how they live in harmony with their environment to ensure and increase their chances of survival. The way the organisms adapt and pass on to their offspring is using genes. Genes form chromosomes which control how an organism looks and behaves.

In nature, a phenomenon called "survival of the fittest" occurs where individual organisms fight for resources and the ones that are best fit survive. When reproducing, these organism pass on their genes to the offspring. When two different genes are combined in the process of reproducing, the offspring creates a new gene pool[6, 58]. The new combinations of chromosomes that are created as a result of mixing parents' chromosomes are called a crossover. In some cases, the two parents' genes form a new combination that allows offspring an advantage. The repeating occurrence of this process allows the species to survive better in the environment than their competition who do not have the same advantage.

The concept of genetic algorithms were first proposed in early 1970s by Holland[6, 22, 58] to create computer program that models and mimics nature's evolutionary process. Genetic algorithms work the same way as GP but instead of working with whole programs, GAs only work with structures that need to be optimized. Genetic algorithms convert a population of potential solutions to an optimization problem. The algorithms work based on the characteristics of the problems rather than on the problem directly. This is very similar to living things where the genes are modified instead of the entire organism. Each solution is then associated with a fitness value to indicate how good it is. The values of the solutions are compared against each other, and higher the value, the higher the chance that solution has on to moving on to the next stages. When moving on to the next stage or iteration, a crossover is simulated to create new combinations of genetic materials.

2.5.2 Recommendation Systems

Recommendation systems need to constantly adapt to user preferences and behaviour overtime to generate relevant recommendations. One of the method is to constantly keep producing new recommendations and ask the user to rate them and produce new ones interactively [26]. In their work, Hyun-Tae et al recommend music to users utilizing an IGA. They extract tempo, pitch and other features from music files to form their solutions. Using the extracted features, they find similar music using content based filtering. Their fitness function is determined by the user who gives rating to each music.

Genetic algorithms can also be employed to form neighbours in collaborative filtering to make recommendations. In their work, Pan et al represent each user in a RS as vector of product information, customer profile information and their transaction data to form solutions to be used in GA[23]. After that, they use this information to find the closest neighbour to the active user and then generate recommendations. After testing the recommendations against actual data, they mutate the profile at each iteration and make recommendation at each stage. This model promises to make better recommendations than normal approaches.

GA has been shown with promising results to solve sparsity and cold start problems in RS. This is approached by constructing feature vectors using available data and filling in missing data with user preferences[13]. Fong et al try to solve lack information in their items by aggregating all available user preference information. Their GA mutates the preference vector at each round by giving different weights to a feature after testing it. By emphasizing on known information about the user, they are able to make better recommendations.

2.5.3 Algorithm overview

Although genetic algorithm does not have a set standard of how it should be laid out[34], Genetic algorithms have five major aspects them [6, 58] which are:

1. Encoding - Each solution/organism have to be encoded in a manner that is easy to represent it using a binary string

2. Fitness function - Each solution has to go through a fitness function which assigns a value indicating how strong the solution is

3. Reproduction - Solutions are selected by their strength and compared to produced a new entity which will share characteristics from the parent species

4. Mutation - Eventually all the organisms that are left over will be mutated until they are very similar and converge to have the same/similar set of characteristics...

2.5.4 Encoding

The heart of every GA algorithm is an solution described by its chromosomes which will be later passed down to their offspring. It is common to use a binary string to represent each solution in GA where each bit indicates whether the solution possess that specific characteristic or not and to use numbers to indicate the strength of a feature in an solution[34, 39]. Using numbers have an advantage over using simple bits to indicate presence of a feature because it allows us to define an solution as stronger than others by using more than just presence of the feature. New features can easily be added by introducing new property and assigning a value, and using zero to indicate the solution does not have that feature. In equation 2.8 we represent a possible solution by describing the strengths of each feature. There are 6 features available and the higher a value, the stronger that feature is.

$$\begin{bmatrix} 4 & 9 & 1 & 3 & 4 & 7 \end{bmatrix} \tag{2.8}$$

2.5.5 Fitness function

The fitness function is a function to rate a given individual and assign a score to it. An individual that shows a lot of promise to be closer to ideal solution we are looking for will have a higher score than others. The fitness function is very critical in GA because they decide whether a solution is good or not and a poorly designed function will lead to less than ideal solutions in the end. Fitness algorithms can vary from being utility function or simple mathematical functions such as sum and average are applied[12, 58]. Utility functions measures the usefulness of something to a user rather than simply plugging into an equation.

2.5.6 Selection and Reproduction

In GP, the fitness function is applied to each and every solution. The assigned fitness value is used to select an individual probabilistically where individuals with higher fitness value have higher chance of being selected[37, 39] in a process called tournament selection. The algorithms used to select individuals also give individuals with lower fitness score a fighting chance to ensure their characteristics are passed on as well to create diversity. If we have two solutions which we consider parent solutions that are described using two vectors [4, 9, 1, 3, 4, 7] and [4 9 1 3 4 8], we can make a child of those solutions such as [4, 9, 1, 3, 4, 8]. In this example, the new solution inherits characteristics of both solutions. A particular characteristic from one parent is probabilistically selected over the other one.

2.5.7 Mutation

Earlier in reproduction, we saw that offspring takes it value from the parents. However, this is not very useful to us when finding the optimal solution. Therefore mutation is the primary way of obtain new solutions to work with in GA. All the solutions in GA are provided with the same probability of mutating with no favouritism. The mutation changes a value in the offspring to be something different than from both of its parents, introducing new possibilities. For example, if we have two solutions [4, 9, 1, 3, 4, 7] and [4, 9, 1, 3, 4, 8] which we consider to be the parents, child can be [4, 9, 5, 3, 4, 7]. In this case, the child not only inherited characteristic of the parents, but the third characteristic is mutated to be different than both parents. This approach has been met with criticism because they exists a chance good solutions will become bad. The solutions that are bad will become worse with the same mutation but the act of assigning the mutation to all also ends up degrading the quality of the good solutions[28, 39]. An approach to this includes reducing the mutation done to favourable solutions with higher fitness scores.

2.6 Cold Start Problem

In order for the recommender systems to predict well, information about the user is needed. There is little work done to alleviate the problem when a new user is introduced to the system in both collaborative filtering and content based filtering systems. Both systems fail to efficiently and accurately predict items due to lack of knowledge of the users preferences, activities, and ratings of items [40, 31]. A recommender

searches the system to for new items and decision theory is proven to be productive in sorting items in terms of their usefulness [40, 16]. In early stages, users can be recommended very similar items due to the lack of information. But as the user looks for more items, their needs change with the items they want [16].

2.6.1 Related work

One approach is to assign the user default set of preferences. The system would learn and adjust the users preferences using a utility function as the user becomes more active [31]. This model is not ideal in the beginning stages of the system where the user has not given enough information due to its tendency to give very non-personalized generic recommendations. Another way to learn more about the user is to assign the new user to a small of set of categories by presenting the user with a small questionnaire after which the system would learn from the user [20]. This method avoids making any assumptions about the user preference at the expense of requiring some initial work from the users. Another proposed methodology for solving for new users includes calculating the demographics for all the users in the system. The new user is compared against the calculated demographic to classify them per preferences which is used to find nearest neighbours [29]. However, this method is not very personalized and ill-suited for many applications.

In order make new items appear more prominent, techniques using filter-bots are utilized [19]. Utilizing bots to analyse documents, and assigning them score based spelling is effective in attacking cold start problem for items [49]. Filter-bots go through the system for new items that have not been exposed to users or have any ratings, and give them a rating by analysing the content using term frequency-invert document frequency (TFI-DF) of the items with promising results [19]. However, the work does not cover when a new user is entered into the system.

Chapter 3

Methodology

In this chapter, we outline our problem that we're trying to solve, our approaches to tackle the problem and how we are going to accomplish it. We will also give information pertaining the tools, packages and any data sets we will be using. Throughout the chapter, as we introduce new methodologies and algorithms, we will also show a brief demo using sample data to give better understanding of how we expect the algorithms to behave.

3.1 Cold-start solution

All recommendation systems will inevitably face cold-start problem. The cold-start problem is defined as having lack of a lot of knowledge and data to make recommendations. This happens when a new user or item is introduced to the system. The system lacks enough information about the user including what kind of items they are looking for, their preferences and characteristics. This makes it incredibly difficult to recommend anything useful and end up making recommendations that are not very personalized.

3.1.1 Problems with current solutions

We must first realize that when there is absolutely no information about the user, we cannot make any recommendations that can be personalized nor can we say are useful to them. Recommendation engines try to characterize users based on the information they know about the user. This includes using geographical information to place the user in a certain region or country, using gender to group them

or using age to place them into different age groups. These systems take away the personalization and treat each user as a generic user. Some systems also users extra questions in the early stage to better understand them but that is cumbersome to the user. The time can be better utilized rather than filling a questionnaire. Many systems in the absence of user preferences, use the very basic information have they use to recommend the same type of content to the user again and again leading to overspecialization. Overspecialisation occurs when the recommendations are are from the same feature space. This is because they do not have enough information to give recommendations from a variety of items.

3.2 Our solution

Our approach in its core is a very simple counter-intuitive idea. In the early stages, when we know very little information about the user, we generate recommendations that we do not know whether the user will like or not. When the user first uses the system, they might rate a single item describing when they like that item or not. We cannot fully describe the user using that one single rating. However, we can generate a lot more to the user from the areas of the system where the item does not fit. For example, if the user likes a movie in the horror genre, we can recommend to the user movies other horror movie lovers have rated highly from different genres such as sports and actions. This lets us learn more about the user's preferences

3.2.1 Reasoning

In early stages of a user's interactions with the system, we will not know all the information. At this step, the user is randomly recommended an item. The user then rate that item whether it was good or not. From this item rating and comparing it with what we know about the item and how others have rated item, we can gain some important perspective of the user. From the rating, we can figure out how much the person likes or hates the category the item belongs to and a little of their tastes towards that category. Then based on that, we can recommend items outside of the category.

For example, if the user has rated an action movie highly, then we can conclude the user has indicated some preference in action movies. Most systems would then recommend more action movies. However, an important point to note is that user will find action movies regardless of whether we recommend them or not just by browsing. The recommender system has nothing to gain by recommending more action

movies to the user. We would like to recommend outside of the range because that is the information we're lacking. In our case, since the user has shown interest in action movies, we will recommend them a drama or comedy film. This will introduce new things to watch that the user might not otherwise find on their own.

The benefits of this approach are many. First, we recommend new thing that the user will not discover themselves out of curiosity. That is what a recommender system should do. Secondly, we now have more chances of learning the user better. By introducing new categories (and genres in the case of movies), we can learn more about the user's tastes and preferences. This will lead to better recommendations later on. Thirdly, we also avoid overspecialisation. Overspecialisation occurs when the user is recommended items of the same category. We both avoid that and recommend new items to user with this approach.

When a user is new to system and has rated their first item, we have three scenarios on which we can build up on and recommend. The first case is that user has rated highly and all the other users have also rated highly. The second case is the user has rated highly but none of the users have rated highly. The third scenario is where user has rated lowly. We do not take into consideration what other users have rated when the rating is low unlike when rating is high. We will get into details as to why later when we discuss how we approach each case.

3.3 Use Cases

The first item the user rates in the system can have different rating values. We have created three generalized cases, and algorithms to handle them. The first case is when the user rates an item highly and other also rate that item highly. The second case occurs when the user gives a high rating to an item that everybody else rated lowly. The case happens when the user gives low rating to a popular item. We have three different algorithms to handle these situations. The first algorithm forms immediate neighbours based on the ratings. However, it fails when no user gave the movie a high rating. We apply the second algorithm to solve that problem. Then, we have a situation where the user gave low rating to a popular movie. The first and second algorithms fail to address this issue. Hence, we develop a third algorithm to fix this. We will talk in more details on how each algorithm works in later sections.

3.4 Dataset and Tools used

Before we begin our methodologies, we have to define the tools we will be using to implement our solution and the dataset we will be using.

3.4.1 Dataset

MovieLens[36] is a dataset collected by a research group from University of Minnesota called GroupLens for conducting research in the field of recommender systems. The data is anonymized to protect user identity. The data contains movies, users the ratings users have given to movies. The specific dataset is chosen due to its wide popularity in the field for benchmarking recommender systems[38]. The dataset consists of approximately 6,040 users, 3,900 movies and 100,000,209 ratings. The possible values for ratings are 1,2,3,4 and 5 with 5 indicating the user liked the movie the most and 1 being the user liked the movie the least.

3.4.2 Programming Language

For our implementation, we will be using Python. Python is an interpreted programming language well suited for doing data analytics, processing and database interactions. We will use Python for importing, transforming and extracting information from our MovieLens dataset.

3.4.3 Libraries and Utilities

We will be working with a large dataset so we will utilize Microsoft SQL Server Express 2014 to hold our data. The data from MovieLens will be imported into SQL Server to make analysing and querying information much easier than reading from files directly. Our research will also include some mathematics which we will implement in code for speed and efficiency. For that need, we will use NumPy, a Python library well suited for mathematics. In order to implement some of our methods using evolutionary algorithms, we will use a library called Distributed Evolutionary Algorithms in Python (DEAP)[14]. The library ships with many common strategies used in evolutionary algorithms implemented thus saving us the trouble of writing very genetic code.

3.5 High rating

The first case we have to tackle is when the user has given a high rating to an item and other users in the system also gave this item a high rating. This is far the best scenario because it gives us information about the user's preference. We can see what kind of items the user likes based on this rating. Since the rating is aligned with how other users have mostly rated as well, we can easily find neighbours to make recommendations.

3.5.1 Technique

We start with a basic recommendation engine that just lists all the movies sorted by their average ratings. This will do the basic recommendation just to expose what is available to the user so they can rate. After the user has given their first rating, we can started on making recommendations. After the recommendations are generated, we will filter out the ones we don't need. An overview of the flow is given on figure 3.1 illustrating the entire process.

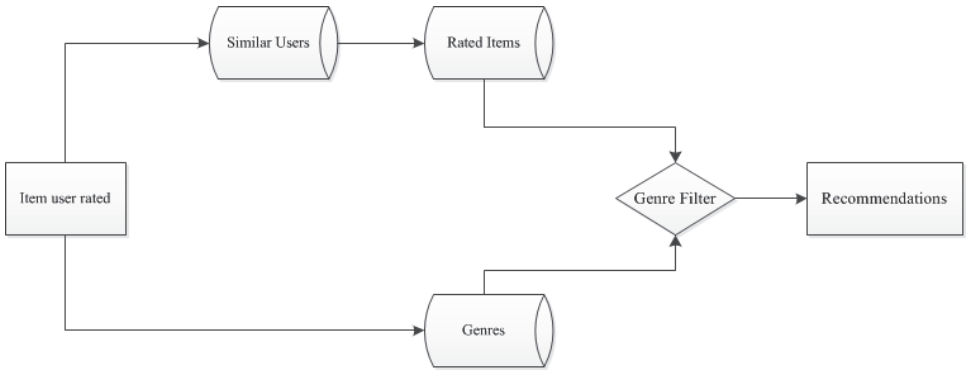

Figure 3.1: Overview of recommending for highly rated movie with immediate neighbours

We will begin by gathering information about the movie the user has rated. The information will include all the genres the specific the movie belongs to and the individual ratings given to the movie by each user. The individual ratings will exclude the active user's rating since it cannot be useful in anyway. Then using the user's rating, we find all the users who have rated the movie with similar rating. We will call these the neighbours since they have some similar preference. The next step will be to find all the movies the neighbours have rated highly. Of the movies gathered, we will remove all the movies that fall in the same genre as the user's rated item. This will give us a list of highly rated movies that

user will not find normally.

A small note to make is that we only get highly rated items from the neighbours. We do not get items with high average. This is because the average is influenced by the community whereas individual ratings reflect each rater's perspective and that is what we would like. We also recommend only the movies the user rated highly. It is true that one user's low rated item can be another's high rated but to avoid making assumptions about the user, we only rate high rated items. Another thing to consider is that when we include items of all types of rating, we saturate the result set. A smaller dataset allows the user to quickly go through. A larger set will cause problems with ordering since we cannot order in any specific way that is meaningful to the user.

In order to accomplish this task, we will need two methods which will be used to gather movie's genre information and to make actual recommendations. The first method will be straight forward which is shown in algorithm 1. The method will simply accept a movie and return a set which will contain all the genres the movie belongs to. We first go through all the genres in our dataset using a for loop and a variable called Genre to keep track at each iteration. If the current movie is part the current genre, we add it to the GenreSet variable. After the loop is completed, we return the set.

Algorithm 1 Getting genre names and ids for a given movie from database and returning them as a set

```
function GetGenres(Movie)
    set GenreSet to {}
    for each Genre in Genres do
        if Movie ⊂ Genre then
            Add Genre to GenreSet
        end if
    end for
    return GenreSet
end function
```

In table 3.1 we have a list of genres and the films. Each genre has a set of films associated to it. The same film can belong to multiple genres. If we are to apply our algorithm to select the genres of Toy Story, we would get back a set that is Children, Animation since it belongs to both of them. The retrieval of genres borrow from the idea of Content-based filtering where we analyze each item to get more information about it.

After we have obtained genre information, the first step will be finding neighbours of the active user. We will use the rating and movie information we have to find the neighbours. The neighbourhood will

Table 3.1: Movies and genres they belong to problem 1

Genre	**Movies**
Children	{Toy Story, Free Willy, ..., Santa Claus}
Horror	{Alien, Exorcism of Emily Rose, ..., Heavy Metal}
Animation	{Aladdin, ..., Toy Story}
Adventure	{Avatar, ..., Rambo}

formed by selecting all the users in the system who have also rated the movie with the same rating as shown an algorithm 2. This will ensure that all the neighbours have indicated similar preference towards that movie. In order to find neighbours, we go through all the ratings of all movies in the system. Then for each rating, we compare the metadata(movie, the user who rated it and the rating itself) associated to it. In this scenario we make the assumption there is at least one user who has given the movie with the same rating. This may not always be true and we will handle that case later on.

Algorithm 2 Get users who gave same rating to a specific movie

```
function GetUsers(Movie, Rating)
    set AllUsers to []
    for each MovieRating in MovieRatings do
        if Movie = MovieRating.Movie then
            if Rating = MovieRating.Rating then
                Add MovieRating.RaterId to AllUsers
            end if
        end if
    end for
    return Users
end function
```

Suppose we have a table of rating as shown in table 3.2 with all the ratings. Each cell in the table has an object representing the movie the rating belongs to, the id of the user who rated it and the actual rating given. If we use the function to find list of people who have rated Toy Story with rating of 5, we will get back a list [56]. This indicates that a user with id 56 has rated the movie 5. All the other ratings will be ignored.

After forming neighbourhood, the next step is to actually make recommendations as shown in figure 3.1. This is done simply by getting all the movies each neighbour has seen and rated as shown in

Table 3.2: Movie rating information for algorithm 1

Ratings
{Movie: Toy Story, RaterId: 1, Rating: 3}
{Movie: Free Willy, RaterId: 56, Rating: 5}
{Movie: Alladin, RaterId: 4, Rating: 5}
....
{Movie: Toy Story, RaterId: 56, Rating: 5}
{Movie: Alladin, RaterId: 56, Rating: 5}
{Movie: Avatar, RaterId: 56, Rating: 5}
{Movie: Exorcism of Emily Rose, RaterId: 543, Rating: 4}
{Movie: Avatar, RaterId: 56, Rating: 5}
.....

algorithm 3. When we make recommendations, we do not exclude all the items the current user has rated since it will be removed later when it is filtered for genres. This is to avoid recommending items that user has already seen. The rating used is a parameter that can be varied to increase the number of results. The lower the rating, the higher the number of items returned.

Algorithm 3 Generate recommendations from neighbours

```
function GenerateRecommendations(Rating, Neighbours)
    initialize Items = {}
    for each MovieRating in MovieRatings do
        if MovieRating.RaterId ⊂ Neighrbours then
            if Rating <= MovieRating.Rating then
                Add MovieRating.ItemId to Items
            end if
        end if
    end for
    return Items
end function
```

The method will be called with a high rating of 4 or 5 initially. The main reason is that we want only movies that have been highly rated and liked by users. By choosing a lower value, we increase the number of recommendations generated but those cannot easily be identified as good or bad. One approach will be to take the average of all the rating the movie has received however that doesn't reflect

the view of the neighbours. It will include ratings from users who are the farthest so we avoid that. The farthest users are users in the system who can have very different preferences. If however, we cannot make any recommendations with a value of 5, we will reduce by one to a minimum of 1. After each deduction, we will generate new recommendations.

Using the user obtained from the ratings of Toy Story, we can now generate recommendations. As we have shown, in our sample dataset the user closely matching our active user is 56. The use has rated Alladin, Toy Story and Avatar with high ratings. Running our algorithm would return us a set of movies containing all three of these including Toy Story, a movie we already watched. The set would be Toy Story, Aladdin, Avatar, Free Willy. We will remove Toy Story later on.

The most crucial step in our recommendation algorithm is the filtering of movies based on their genre. In this stage, we use a function shown in algorithm 4 to remove any movie that fits in any one of the given genre. When we use this function along with the genre of the movie user has rated, we will get movies that have very different genre. The first parameter of the function will be the set of genre from algorithm 1 and the second parameter is the movies from algorithm 3. The result will be a set of movies whose genres are mutually exclusive from the movie the user rated. During the filtering stage, we also remove any recommended movie that the user has already watched.

Algorithm 4 Filter out movies if they match any of the genre of a given set

```
function FilterMovies(Movies, Genres, WatchedMovie)
    initialize FilteredMovies to {}
    for each Movie in Movies do
        if Movie ⊂ WatchedMovies then
            continue
        end if
        set MovieGenres = GetGenres(Movie)
        if MovieGenres ∩ Genres = {} then
            Add Movie to FilteredMovies
        end if
    end for
    return FilteredMovies
end function
```

To illustrate this, we will use our set of movies obtained earlier. Since the active user rated Toy Story, we already know that it belongs to set of genre that is Children, Animation. Our filter method will remove all movies from our recommendation step that are part of either one of these genres. Alladin also falls into the same two genre so it will be filtered out. Then we look at Avatar. Avatar falls under

the category of Adventure but not Childen nor Animation so we keep that movie. Then Toy Story automatically gets filtered out. Our last movie in the list is Free Willy. Free Willy falls in the Children genre. Since it seems that children movies are of interest to the user, we also filter out Free Willy. This gives us a recommendation consisting of Avatar. This is a movie that user hasn't watched yet and it is in a new genre.

Recommending Avatar gives us the potential to learn more about their user's preference. Now we have movies that user will highly like outside the area of interest we know about. Based on the rating the user gives to Avatar, we can further learn more about the user and make better predictions.

3.6 High Rating with Different Taste

There is a scenario where a user might like a movie very much but no one else will. This can happen very frequently where a user would give a movie a rating of 5 but no other user in the system will. This makes it very difficult to form neighbours since the user's preference does not match any other user. To handle this tricky situation, we use an item-to-item-to-user similarity to find neighbours.

3.6.1 Technique

In order to find users when there is no user who has rated the item similarly, we use a transitivity approach as shown in figure 3.2. Due to the fact we cant find similar users to active users directly, we take a transitive route through the item rated. First we gather the genre of the movie rated. This is done using the same algorithm 1 as earlier. After obtaining the genre set, we apply a new algorithm on the set to find all movies from the system that are categorized into those genres. This would give us more items the active user has interest in. This also produces more items that have the potential to be rated highly by other users as well, helping to form neighbours.

Let's say we have the ratings for just Toy Story as shown in table 3.3 where none of the ratings are 5. In this case, our active user gives Toy Story of 5. Using the methods we have described above, we cannot form neighbours. Instead, we make basic assumption that the user is into the genre that Toy Story falls into. This is evident because the active user is the only who has rated it highly. This indicates the user is more into the genre than the actual movie. This is because the user watched a bad movie. The only information that can captivate the user to watch the movie is the genre, and not the storyline. To begin

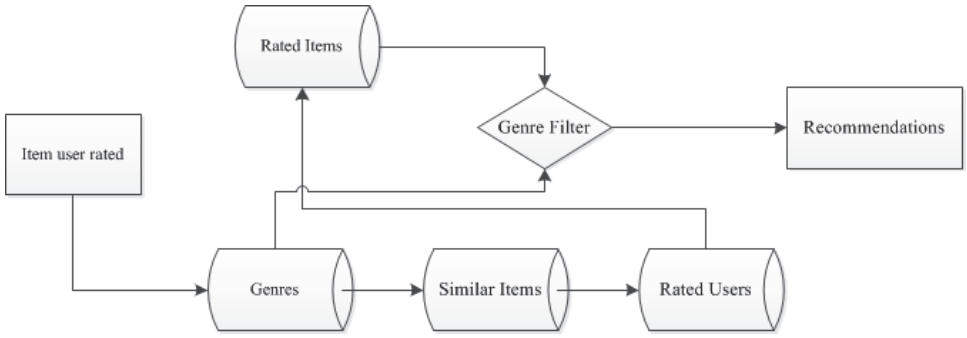

Figure 3.2: Overview of reecommending for highly rated movie with no immediate neighbours

Table 3.3: Movie ratings for algorithm 2

Ratings
{Movie: Toy Story, RaterId: 1, Rating: 3}
{Movie: Toy Story, RaterId: 56, Rating: 2}
{Movie: Toy Story, RaterId: 4, Rating: 1}
{Movie: Toy Story, RaterId: 352, Rating: 4}
{Movie: Toy Story, RaterId: 187, Rating: 3}

neighbours, we first get the genres Toy Story belongs to, which would be Children, Animation.

After obtaining the genre, we find all movies that also fall in the same genres. In this scenario, we find all movies that fall into all of the genre and have average rating above a certain threshold. We use average rating here because we want highly rated movies from that genre. We also use the average rating as a paramter to filter out movies so we can control how many movies we can find as neighbours for current movie as shown in algorithm 5.

To illustrate how this works, let's use a more detailed ratings as shown in table 3.2. After obtaining the genre for Toy Story, we run our neighbourhood formation algorithm for items. This is a item to item filtering. We go through all the movies in the system using an average rating of 3.0 and find that Aladdin is the only movie that satisfiesour constraint. Now we only have one movie. That is not much to recommend to the user. It also falls in the same genre as Toy Story which we want to avoid recommending so the next step will be find new users who might be neighbours.

We will be doing this by finding users who rated the movie of neighbours highly. We will define a

Algorithm 5 Find movies that are in the same genre as a given movie

```
function FindMovies(CurrentMovie, AllMovies, Rating)
    initialize FilteredMovies to {}
    initialize CurrentMovieGenre = GetGenres(CurrentMovie)
    for each Movie in AllMovies do
        if CurrentMovie = Movie then
            continue
        end if
        initialize MovieGenre = GetGenres(Movie)
        if CurrentMovieGenre = MovieGenre and Movie.AvgRating ≥ Rating then
            Add Movie to FilteredMovies
        end if
    end for
    return FilteredMovies
end function
```

function as shown in algorithm 6 that will accept three parameters, a rating, all the movies that are neighbours and all the ratings in the system. The rating will be determined when calling the function. The algorithm will go through all the ratings in the system. For each rating, it will check if that rating belongs to one of the neighbour movies. Neighbour movie is a movie that is in the same genre or has a genre in common. If it does, then we check the actual rating the user has given. If the rating is above the value we pass, we add that user to list of users and we return the entire user list at the end. We will use a rating of 3 to get rating as much as users as possible to form neighbours while still retaining highly rated movies. If we choose a lower rating, we will get more neighbours. By using a rating of 3, we will get users who actually liked those movies and perhaps the genres those movies fall into. This will help us form a better neighbour of users.

Algorithm 6 Find users who rated set of movies highly

```
function FindMovieRaters(Rating, Movies, MovieRatings)
    initialize Users = {}
    for each Movie in Movies do
        for each MovieRating in MovieRatings do
            if MovieRating.Movie = Movie then
                if Rating <= MovieRating.Rating then
                    Add MovieRating.RaterId to Users
                end if
            end if
        end for
    end for
    return Users
end function
```

Applying algorithm 6 will give us users who gave high ratings for the movie Aladdin we obtained earlier. The users wouldbe {4, 56} from the table 3.2. We assume these users to be the closest neighbours of the active user given the constraints such as the rating and movie rated. After we have gotten the users, we can easily make recommendations as outlined in algorithm 3. We will be using a rating value of 5. That is, we will only select items rated 5 by the users to recommend to the active user to make sure we only recommend movies that were seen as beneficial by the neighbour. This will give us a list of {Avatar, Aladdin, Toy Story, Free Willy}.

After having a list of movies, the final step is to use algorithm 4 to filter out movies that belong to the same genre. Since Aladdin, Toy Story, and Free Willy fall in the same genre of either Children or Animation we filter them out of the list. The remaining movie Avatar is then recommended to the user. This again is of a different genre so we have much better recommendation.

3.7 Low Rating

We handled the two scenarios that can happen when the user gives a high rating to a movie. The first case where everybody agrees with the user is relatively is to handle while the second case where no other user in the system gave the same high rating was a bit more difficult. However, using a transitive approach, we also handled that case elegantly. The third scenario that can happen is where the user gives a movie a very low rating and this is a much more difficult problem to solve. There can be two main reasons why a user can give a movie a low rating.

3.7.1 Bad Movie

First example that occurs when a movie is badly rated is that the movie is bad. We can easily confirm this by comparing the ratings the other users gave. If the average rating of the movie falls towards the lower end of 1-5 rating spectrum, we can confirm that the movie is indeed a bad movie. However, we have to note that the user was willing to watch the movie based on the genre. This indicates the use has shown interest in the movie.

To illustrate this point, we take a look at tables 3.4 and 3.5 which contain the rating data and the genre data for the movie The Last Airbender respectively. If our active user gives the movie a rating of 1, we can compare that against other ratings the system. We will gather the average of the ratings,

Table 3.4: Movies and genres they belong to problem 2

Ratings
{Movie: The Last Airbender, RaterId: 1, Rating: 1}
{Movie: The Last Airbender, RaterId: 56, Rating: 2}
{Movie: The Last Airbender, RaterId: 4, Rating: 1}
{Movie: The Last Airbender, RaterId: 352, Rating: 1}
{Movie: The Last Airbender, RaterId: 187, Rating: 2}

Table 3.5: Movies and their genre information problem 3

Genre	**Movies**
Children	{Toy Story, Free Willy, ..., The Last Airbender}
Animation	{Aladdin, ..., Toy The Last Airbender}
Adventure	{Avatar, ..., The Last Airbender}

which will be 1.4 in our scenario. This is evident that the movie is terrible but the user still showed interest in either Children, Animation or Adventure movies.

3.7.2 Good Movie

The second scenario occurs when everyone in the system gave the movie high rating. This also leads to two potential cases. The first situation occurs where the user actually dislikes the movie but likes the genre. This can occur if a particular plot is not very favourable to the user so they decide to give it a low rating. This does not tell us much about the user's preference because we cannot say with guarantee that user likes or hates the genre. If user dislikes Toy Story, we cannot say with confidence they hate the Children or Animation genre.

The second situation as to why the user rated it lowly is because they do not like the genre. For example, if they rated Toy Story with a low rating, it could indicate that the user does not like Children nor Animation. Even though this is a possibility, we have to take into consideration that if the user

did not like the genre, they would not have opted out to watch or rate Toy Story as the first movie. Therefore, we can assume that user did like the genre of Children and Animation.

By analysing all the possible outcomes, we can conclude the user did in fact like the genre when they give a bad rating. If only they did not like the movie, they gave it low rating unlike other users indicating they wanted to give the genre a try. The other case where users also rated lowly still indicates they wanted a movie from the genre. Using these findings, we can continue to form neighbours and generate recommendations.

3.7.3 Genetic Algorithms

To aid us in solving this problem, we have a few ways of using evolutionary algorithms. We have the option to implement using a genetic programming method or a genetic algorithm approach. We opt out to a genetic algorithm because we don't have enough information to implement various strategies required from genetic programming. In GP, we evolve programs to solve our problem. However, with limited knowledge of the user, we can not implement it.

As discussed earlier, evolutionary algorithms (EA) are an optimal solution finding algorithms. That means throughout mutation and crossover at each generation, they find the optimal solution that fits our requirements. Another characteristic of EA is that towards the end of the cycle, the solutions converge to be very similar. This will aid in making better recommendations because it will fulfil our criteria for finding the optimal recommendations.

Unlike our scenarios with high rating, we do not find similar users or items using the mechanisms described earlier but use a GA approach. This is because we want to avoid neighbours formed on the basis of negative ratings. The reasoning is that users have preferences in terms of what kind of movies they like and the genres they are into. When we have a high rating, it is easy to form neighbourhood for like minded users since the amount of genres that will be similar are very low. However out of the possible genres, the users will dislike more genres than they actually like. When we form a neighbour using this approach, we dilute our approach with less than optimal data which will lead to recommendations of poor quality.

In order to make recommendations, we will have to find a best vector that can match the users. In other words, we will find an individual that will have the best chance of finding neighbours who highly rate movies. We will first construct a preference vector for each user by averaging their ratings per each

genre of movie. Using GA, we will then find a best individual solution that matches users by comparing the user's preferences against our current individual. Then by utilizing that solution and forming a neighbourhood, we will generate recommendations filtering out movies in the same genre.

Before we start implementing our Genetic Algorithmic approach, we must define how our individual solutions will look like, how they are represented and what information they will provide.

3.7.4 Individual Representation

In our MovieLens data, the movies are categorized into 18 different genres. We will use a vector of size 18 where each value directly corresponds to each of the movie genre to represent each individual. Each value will be in the range of 0.0 - 1.0 inclusively. The values will be an indication of how much they directly correlate to the genre. That is, a value closer to 1.0 will indicate a strong preference towards that genre while a value closer to 0 indicates dislike towards that genre. We will use this vector to find users who have similar tastes and preferences in terms of genres. This representation is very useful at mutation because at each iteration, the mutation will either cause the individual to prefer some genres more or dislike them more.

We choose preference vector of different genres to represent the user because of its versatility. We can easily implement a fitness function of our choosing. It is also easy to use in the selection and crossover process. We can easily select individuals based on their certain genre values and crossover by easily mapping to same preferences of another individual. Also having real number in the range of 0 to 1.0 gives us greater range of mutation.

3.7.5 Sample initialization

In our genetic approach, we need a sample population to begin with. Under normal circumstances, in GA, the sample population is selected as N random solutions from the entire population. This allows the algorithm to begin with actual data and find the best solutions. However, this is not the case with our scenario. We have an initial population which is the preferences of each individual user in the system. However, our can not form a sample population from these preferences because it does not work for the active user. Our initial population should not be based on existing users since we will need to find neighbours. In other words, it is not worthwhile if we construct the initial population using the same

users who we are trying to match against.

To circumvent this problem, we will construct the initial population randomly. The size of the initial population will be 18. We will have a population of 18 individuals where each individual have preferences for the 18 different genres in the system. The reason for choosing a population of 18 is that we want each individual to highly correlate towards a specific genre. For example, the first individual will like Children most, second will like Animation the most and so on. All the other values are randomly selected but will be less than the genre the individual prefers the most.

The reasoning behind choosing our population in such manner is that our ultimate goal is to find the best individual who will be able find neighbours who have same preferences and also have highly rated movies. The reason we choose 18 individuals mapping to different genre instead of one individual which likes all genres is that it is both impossible and cannot be used in GAs. What we mean by impossible is that, we will not have a user who likes all the genres equally.

One of the most important aspect of each individual is that, it will also have a strong preference towards the genres of the movie rated. The value will not be higher than the genre the individual represent but it will not be lower than any of the other genres. This is very crucial because this is the only information we have about the user. This will aid in forming neighbours who likes the genre the user showed interest in and like other genres as well.

To demonstrate how our initial population will look, we'll use individuals from table 3.6. The table only covers three different genre for this purpose. The individual's representation will be of format [Children, Animation, Adventure] to indicate the different genres that are possible in the system. Each value will indicate preference level. In our example, we assume the active user liked a movie from the genre of Children. The first individual which favours Children, does not have high values for any of the other genres. However, the other solutions have a high value to indicate preference for Children movie and their corresponding genre. The second individual favours Animation mover Adventure while the third individual prefers Adventure over Animation. They both equally prefer Children genre.

3.7.6 Selection and crossover

We have to determine the mechanisms used to select parent individuals at each iteration to be used for generating offspring. For this task, we will be using tournament selection process that is prevalent in the field. Tournament selection method selects top N individuals who score highly using the fitness function.

Table 3.6: Sample initial population of three genres

Genre	**Individual**
Children	[1.0, 0.435, 0.443]
Animation	[1.0, 0.652, 0.286]
Adventure	[1.0, 0.158, 0.751]

The selection process is random giving higher chances to individuals with higher scores. This enables us to select individuals with higher potential to be used in the next generation and to produce offspring. We will use N=3 to limit the number of individuals selected. If this however proves inadequate, we can increase it later up to maximum of 18 to suit our needs.

Crossover step of the algorithm is where the offspring is produced with characteristics of both parents. This allows the offspring to be created with higher preference values for some genres. This will allow the individual to highly correlate to some users in the system. For this step, we will be using two point crossover method. The two point crossover splits both parents into several segments. Each parent's segment will have a corresponding segment from the other parent of the same length and size. The child will then have randomly selected segments from either one of the parents. The selected segments are joined to create the offspring. The offspring will then have the same length of both parents but different features from both parents. We can control how much of each parent to select for generating of the offspring. We can choose to select 40% of genetic material from one parent and 60% of genetic material from the second parent to create the offspring. However, to give all individuals an equal chance of passing their genetic material over, we choose a value of 0.5 or 50%. This will make sure we get half genetic material from each parent.

One of the interesting thing to note is the reasoning behind crossover for algorithm. We had initialized our individuals with preferences towards a specific genre. However during crossover, this all break down and each solution might have a lot of genres to like. At the same time, the crossover has the chances to create offspring who will dislike all the genres. This allows us to overcome our problem we had during initialization. We did not have enough information to create diverse individuals. However, with crossover we overcome that problem.

In table 3.7 we have a small list of individuals, and the fitness scores associated to them. We will

Table 3.7: Fitness scores for individuals

Fitness score	**Individual**
0.75	[0.422, 0.435, 0.443]
0.65	[0.751, 0.652, 0.286]
0.34	[0.404, 0.158, 0.751]
0.83	[0.407, 0.652, 0.286]

discuss about the fitness function later. The selection process will randomly select individuals with higher fitness values to create offsprings. In our example, the first two individuals with fitness values of 0.75 and 0.83 will be selected because out of the entire population, they have the highest values. After selecting, their offspring will contain half the genetic material from each parent. Since, our individuals have an odd number of preferences, we will randomly choose one parent and take an extra preference from them. For example, the offpsring might look like [0.422 0.652 0.286]. The first value was obtained from the first parent and the other two values are obtained from the second parent.

3.7.7 Mutation

The mutation process is where we introduce a little variation into the population. Mutation is where we modify the offspring created using existing solutions to hopefully introduce new and useful features while eliminating the features that are not worth while nor beneficial. However, this is not guaranteed to always produce beneficial features. Hence, we go through various generations to find solutions that have beneficial features. For mutation, we will be using the Gaussian method.

The Guassian method adds or removes a random value to each vector. The vector to mutate is selected using a probability so not every value in the vector is mutated. For the mutation probability, we will use 0.2 as seed value. This is a parameter that we can later modify to affect the result and speed of the algorithm. We select a random value generated using the seed. If the value is greater than 0.5, then the a specific preference of the individual is mutated. If it is less than or equal to 0.5, then we leave the value as it is. This makes sure we do not completely change then entire solution during mutation. We choose a value of 0.2 because it is low enough that every preference in individual have a chance of mutation.

One of the advantages of Guassian method is that if the value of a preference after mutation falls outside of the lower or upper range, it will be reduced to bring it down within the boundaries. The reason we choose the Guassian method is because it automatically takes care of generating random value, adding it to current value and making sure the final result is acceptable. One of the most important things to note is that after mutation, the solution might not still respect the preference it was initialized for. If we have a solution representing Children's genre with high value in the Chidren's spot, after mutation it might be lower than all the other genres. This is one of the drawbacks since it makes solution prefers the genre less. However, the mutation also gives it the opportunity to correlate to Children more. The mutation also affects other values as well. This could lead to a solution designed to prefer Children genre to prefer Comedy after generations of mutations because it is well suited.

3.7.8 Fitness Function

In every GA, the fitness function forms the heart of the algorithm. It also is the most personalized part of any genetic algorithm. We have to define our fitness function that will measure each individual and assign it a score. A higher value indicates the individual is preferred while lower value is not preferred. We will define our fitness function based on how well it can be utilized to make recommendations. Our ultimate goal is to find the best individual to make recommendations. Therefore, we evaluate each individual's ability to generate recommendation and assign a score based on the ratings of the recommendations.

Our individuals have values indicating how well they correlate to each genre in the movie database. The first step is to find users who are considered top N neighbours of the individuals. In order to find neighbours, we use cosine similarity measure. We compare the individual against all the users in the system. Each user will have their preferences based on their ratings and which genres and movies the rating applies to. We will use that information to sort through users and find top N neighbours who are the closest. Closest neighbours will have an angle closer to 0.0 when using running through cosine measure. We will randomly choose N = 10 to find 10 neighbours for each individual.

3.7.9 Algorithm overview

We have described each component in the genetic approach as shown in the 3.3. However, the actual recommendations are generated only after the genetic algorithm has finished running. Our algorithm

runs for a specified number of generations rather than running until it meets a certain goal. The reason is that we do not know in advance the exact result of what we would like to achieve. Therefore, it is difficult if not impossible to keep the program running for many generations. Therefore, we will only run our algorithm for 50 generations for each user. This limit would make sure we do not run forever. It will also end our GA within realistic times. Our algorithm does not have a objective function that can be used to

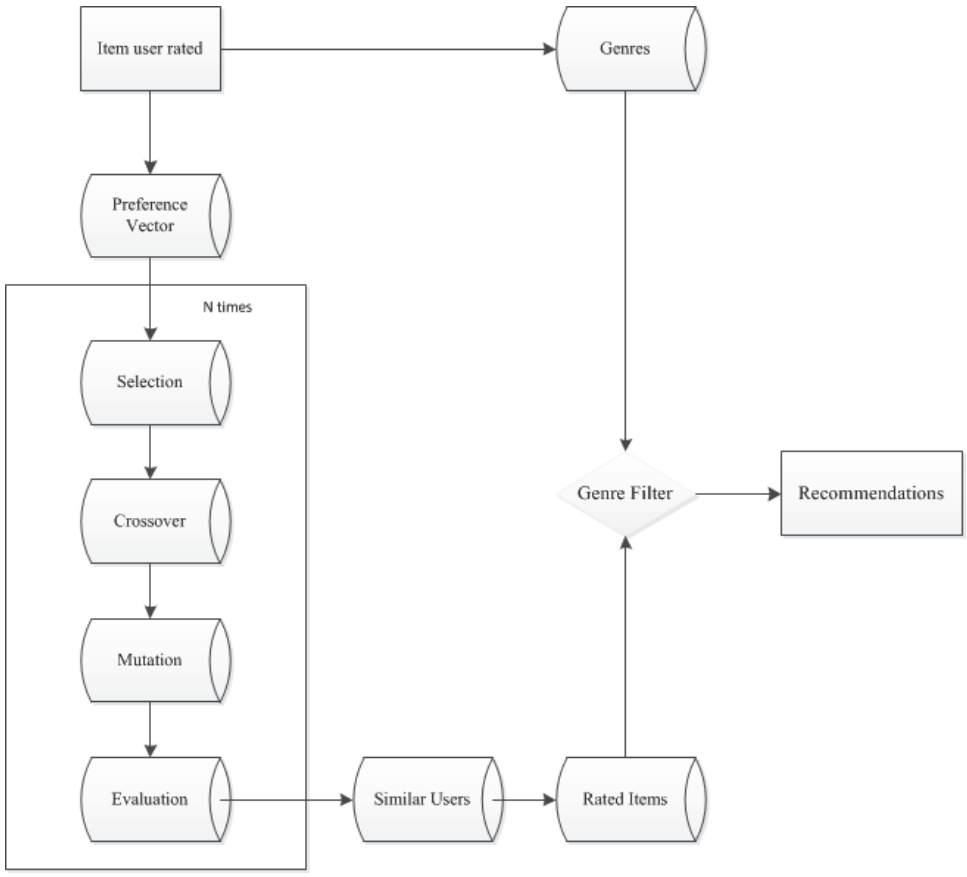

Figure 3.3: Overview of recommending using Genetic Algorithm

After GA, we have the best optimal individual. We choose the most optimal individual obtained by running the algorithm. Then, we do a cosine measure against all the other users again to find users who match the individual well. In our scenario, the optimal individual is able to generate recommendations of highly rated items by users of many genres rather than one specific.

After the formation of neighbours, we select top N neighbours again. We randomly choose N = 10

for our intended purposes. This can be changed to increase the number of recommendations or lower them. We can also use it to speed up our algorithm by lowering the value if we notice any performance issues. Then we get all the items neighbours have rated highly with a score of at least 4. We then filter out movies that belong to the same genre as the current movie. This would give us new movies outside the range of what we know about the user to recommend.

The real benefit of using the Genetic Algorithm approach is that we do not need to have lots of information about the user to make personalized recommendations. Earlier when the user gave high ratings, we knew the user liked the movie and the genre. However, with low rating we cannot argue with certainty the use hates the movie or genre. So we use genetic approach we tries to find vectors based on current user that will find neighbours. The genetic approach adds variety to make sure we can get diverse neighbours who have different genre preferences.

Chapter 4

Experiments

In order to prove that our methodology works, we have to implement it in a feasible manner, run, generate results and compare them against a set of values we know. We also have to use methods that are well accepted to evaluate and criticize the effectiveness of algorithms and approaches.

4.1 Evaluation

Evaluating recommendation systems includes addressing a problem, and applying an evaluation method to check whether the problem is addressed or not[53]. In order for recommender systems to be useful, they must provide a solution to problem. The problem must be well defined so it is measurable whether the problem was solved or not. There are three distinct ways of experimenting, and validating the efficiency of recommendation system. They are off-line experiments, user studies and on-line experiments[54]. The evaluation approach that we will be utilizing for our system is the off-line evaluation method.

4.1.1 User selection

In more details for our experiments, we will select N random users. We will then select a random user from that sample list. We will introduce that user to the system as a new user. The data set will have all the ratings of that user removed. Then we will randomly select an item that user has actually rated to represent the initial item the user has rated. Then we will run our algorithms accordingly to generate recommendations. We will do this for all the users in the random sample. When we generate, we will

utilize all the ratings and users except the active user.

To test all three of the scenarios, we need to extract some users from the dataset. We will designate 100 users from the entire dataset to be used for testing. When selecting the 100 users, we have to make sure that they full-fill the requirements of each scenario we are trying to tackle. The first set of users must have rated a movie with high rating and that movie must also be rated similarly by other users. The second set of users who need to have rated the movie high but no other user in system should have given the same rating. The third set is where the users must have given a low rating while no other user in the system has given the same movie a low rating.

4.1.2 Item selection

The second step to gathering the data required to test our method is the item that will be used as the entry point by the user. All our scenarios are based on assumption that the active user rated at least one of the items in the system. Each scenario will have the user rate a specific item that full-fills the requirements. In the first case, we need the user to rate an item highly which is also rated highly by other users. For the second scenario, we need the user to have rated an item highly where no other user has given a high rating to that item. The third scenario requires the user has given a low rating to an item while others have given higher ratings. We already have users who fill these requirements but the important step is to find the rated item that places them in the specific group.

4.1.3 Precision and Recall

An important metric in comparing the results generated and the actual items is precision and recall[21]. It is important in recommender systems and mainly offline systems because it lets us test whether the items we have generated are relevant to the user. Precision is the measure that indicates how relevant an item is to the user. Recall is the measure that gives the probability that a relevant item will generated. Precision is calculated as shown equation 4.1 and recall is calculated as shown in equation 4.2.

Calculating precision takes the number of items generated which are also rated by the user previously. It is then divided by the actual number of items generated. This will indicate how well the algorithm generate items that closely resemble what the user has already rated or viewed. The higher the intersection of the two sets, the higher the precision. Precision is only useful to use in the contexts of movies

that fall outside the genre of the item user has rated. This is because we can only compare similar sets otherwise we would end up using datasets with different characteristics.

$$Precision = \frac{items\ generated\ which\ also\ exists\ in\ items\ rated}{total\ items\ generated} \tag{4.1}$$

Recall is calculated very similarly by taking the number of items generated which are also rated by the user previously. It is then divided by the number of actual items that user rated. This gives the percentage of items that are relevant and are generated by the system for the user. A higher value indicates that we are able to generate better recommendations. As in precision, we will only consider movies that fall outside the genre of the movie the user has rated. In our cases, the total relevant items are movies that that user has watched that are not part of a given set of genres.

$$Recall = \frac{items\ generated\ which\ also\ exists\ in\ items\ rated}{total\ relevant\ items} \tag{4.2}$$

4.1.4 Offline evaluation approach

Offline evaluation method is very attractive to our approach because it lets us test our algorithm without user input and interaction[54]. This is ideal in situations where we already have data about users, items and the ratings. This method takes in the data to simulate real users and test the algorithm. A small number of users are first selected to represent the set of users to test the algorithm on. They are then introduced to system as new users. The set of results produced by our recommender is compared against the actual values to see how well we perform.

To evaluate our algorithm, we make recommendation for each user from each scenario utilizing the algorithm that we have devised the scenario. The recommendations are then compared against the user's actual rating data to see how well the fare. One of the method we will be following is to compare the ratings the user has given to the items compared to the items that were generated. A higher rating from the recommended result is promising because it indicates the user prefers the recommendations more than the other movies we have discarded as non-relevant. A lower rating for recommended result means we were unsuccessful in making good recommendations.

To illustrate our point, let's assume user has ratings as shown in table 4.1. The table 4.2 has items generated to the user. The list of generated movies explicitly excludes Children genre. The first thing

Table 4.1: Sample ratings for user

Ratings
{Movie: Toy Story, Rating: 3}
{Movie: Lord of The Rings, Rating: 5}
{Movie: Blade Runner, Rating: 1}
{Movie: Hunger Games, Rating: 4}
{Movie: Twelve Monkeys, Rating: 3}

we do calculate the average of the user, 3.20, from the rating data as shown in table 4.1. Then we calculate the average of movies that were generated for the user as shown in equation 4.3. Then we compare both averages. The average of the recommended items is higher than the actual average of the user therefore we can say the quality of recommendations is favourable. A key thing to note is that the recommendations include a movie, Exorcism of Emily Rose which the user has not viewed nor rated. We cannot reliably use that movie in our rating prediction whether it may affect our result positively or negatively. We eliminate any uncertainty from our results.

The limitation of off-line method is that we cannot verify movies the user has not given a rating to. This limits us the number of items we can utilize. Offline method is not ideal in places where novel items to be recommended that try solve sparsity problems[21]. This is not case with our approach since our goal is to make recommendations of any sort to the user filtering out some items in the process.

$$\begin{aligned} Avg &= \frac{3+5+1+4+3}{5} \\ &= \frac{16}{5} \\ &= 3.20 \end{aligned} \tag{4.3}$$

$$\begin{aligned} Avg &= \frac{5+1+4}{3} \\ &= \frac{10}{3} \\ &= 3.33 \end{aligned} \tag{4.4}$$

Table 4.2: Sample recommendations generated for user

Recommendations
{Movie: Lord of The Rings}
{Movie: Blade Runner}
{Movie: Hunger Games}
{Movie: Exorcism of Emily Rose}

4.2 Evaluating High Rating

4.2.1 General results

In figure 4.1 we have a plot of the actual ratings of the user compared against the ratings of the generated items. The actual ratings refer to the rating data in the MovieLens dataset for the user. The chart shows that the ratings are more or less on par with each other. The actual ratings are higher for some users while the generated items' ratings are higher for other users. Taking a closer look, we can see that on average, the ratings of the generated items are higher than the actual ratings. This is an indication that our algorithm is well suited for generating quality movies to recommend in the absence of information about the user. The results would have been acceptable even if the lines were equal due to the fact that we were able to generate recommendations that are equivalent to what the user would have discovered and watched with minimal information.

An important point to note is that couple of times the generated ratings were much smaller than the actual ratings and the generated ratings are much higher than the actual ratings. These anomalies cannot be meaningfully interpreted as to whether the algorithm is really good nor indicates that the algorithm needs work. The reason spikes occur is because of the lack of information to fully understand the user. The algorithm makes some prediction to discard movies belonging to the genre of rated movie. These movies are either rated highly or lowly by the user. When they are removed, we see the random spikes.

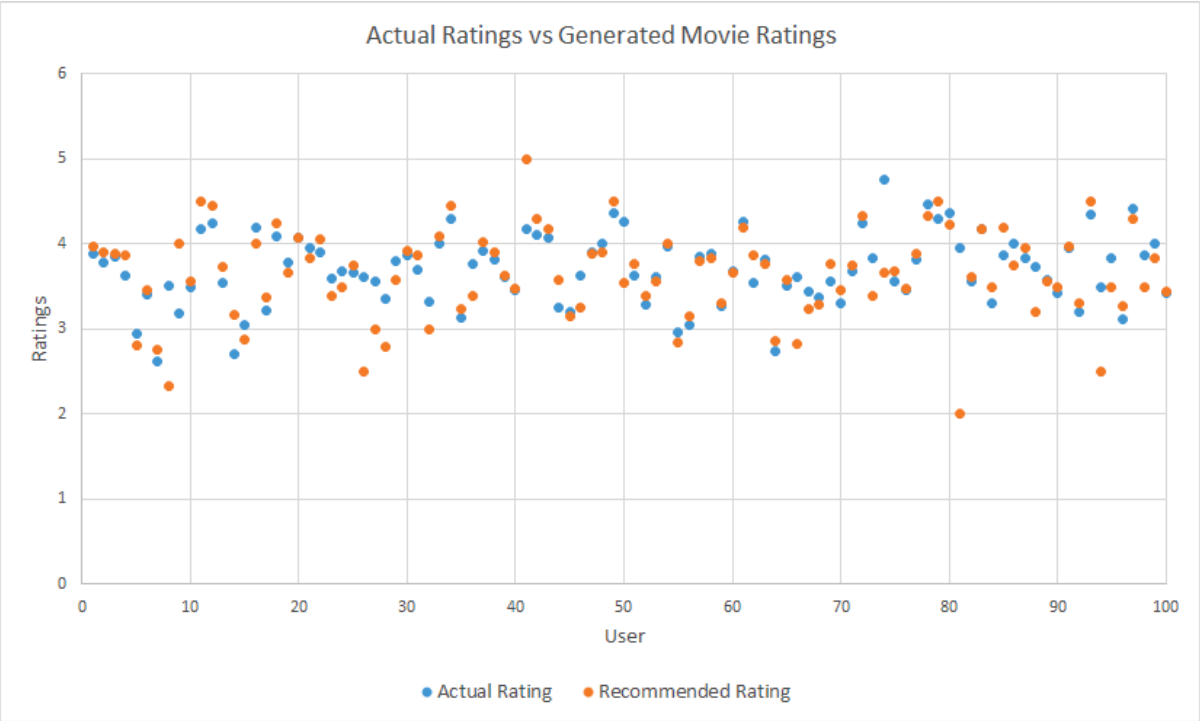

Figure 4.1: Overview of ratings of 100 users for first scenario

4.2.2 Percentage of ratings

An important metric in comparing actual versus generated ratings is the percentage of users who had actual rating higher than the percentage who had generated ratings higher. This will show a better picture of whether actual or generated ratings is overall higher with clarity as shown in figure 4.2. We can see that more users have higher number of generated ratings than actual ratings. That is, 56 users' actual rating are less than the ratings of the items that were generated. Only 44 users have actual ratings which are higher. The 12% difference is a lot considering that the recommendations were generated without any additional information from the user and with only one rating.

We cannot prove or disprove our algorithm by running over a small sample. Therefore we ran the test 10 more times with different users who are chosen randomly. The results of the ten tries are given in table 4.3. We can see from the table that the actual ratings are lower than the generated ratings in most cases even if the differences are not very significant. In some cases, the actual rating are higher but by very little. In rare cases, there was a difference of 24% between actual and generated ratings.

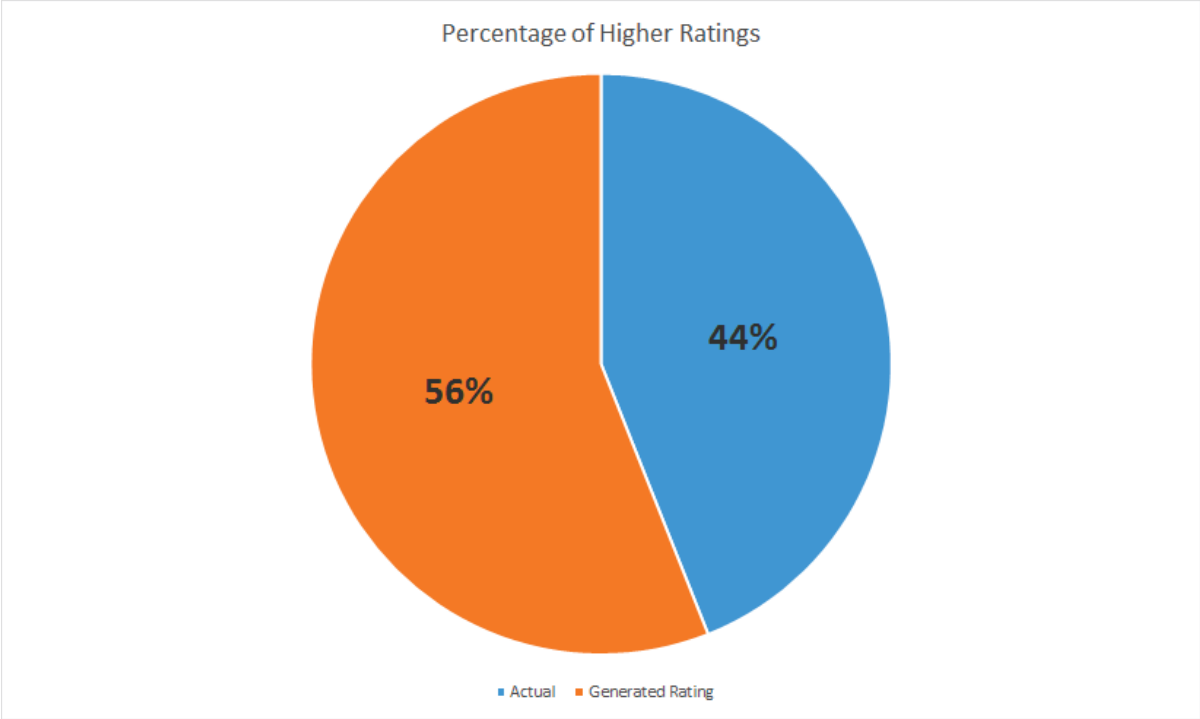

Figure 4.2: Overview of actual ratings vs generated ratings for first scenario

4.3 Evaluating High Rating with Different Taste

In the second scenario we are trying to make recommendations when the user has given a movie high rating but no other user has given it a high rating. We selected 100 different users who full-fill this criterion to run our experiments and gather results.

4.3.1 General Results

The results of running for 100 users is shown in the figure 4.3 where we plot the actual ratings against the ratings the user would have given the items recommended through our algorithm. We see that almost every user has given high rating to the generated movies than their overall movies. This is very different from our results from the first scenario where the results were more or less on par. This probably due to the various criteria that must be filled in while we are forming neighbours during the transitive approach using the rated movie.

Table 4.3: Results of running first algorithm ten times for first scenario

Actual ratings	**Generated Ratings**
49%	51%
44%	56%
51%	49%
48%	52%
50%	50%
45%	55%
43%	57%
39%	61%
43%	57%
52%	48%

We select items that are rated highly which also belong the same genre as the rated movie. Then we form neighbourhood using these highly rated items. During this process, we minimize the movies we can utilize to form neighbours and we end up eliminating low ranked movies. Thus, we tend to form neighbourhood of users who give higher ratings. Then at the end when make recommendations, the movies tend to be from the higher rated end, producing the effect we see. This is very ideal to our problem we're trying to solve. We could not form neighbours because the active user's preference did not match any of the user in the system. However, through an item-to-items-to-raters transitive approach we are able to make recommendations.

4.3.2 Rating distribution

In figure 4.4 we can see the distribution from our run of the second algorithm. Earlier we saw that both ratings were hovering around the same values. However, it turns out that generated ratings have higher value than the actual values more often. This indicates that our algorithm works well. Particularly, we see that there is a difference of 26% between the ratings. Our second algorithm is much more promising than our first algorithm when it comes making recommendations.

The next step to verify that our results can be obtained if we repeat the experiments. We ran the

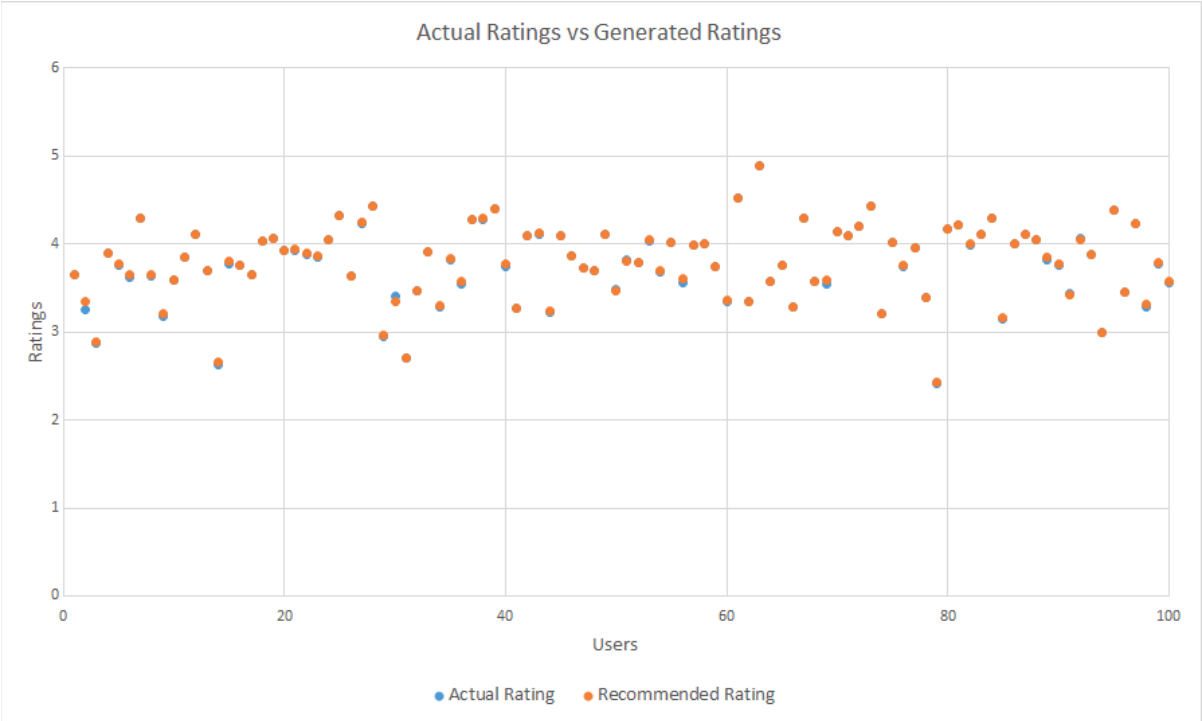

Figure 4.3: Overview of actual ratings vs generated ratings for second scenario

tests for 10 additional times to confirm our experiments. The results are shown in table 4.4. We have found that the results tend to stay within same range of both actual ratings and generated ratings. The difference between these are also hovering around 25% with the generating ratings taking the lead. This shows that we can apply this algorithm and expect to make good recommendations even when the user has different tastes.

4.4 Evaluating Low Rating

The last scenario we wish to evaluate is when the user gives a low rating. For this, we implemented the algorithm using an evolutionary approach to evolve the user's preferences to match against other users to form neighbours.

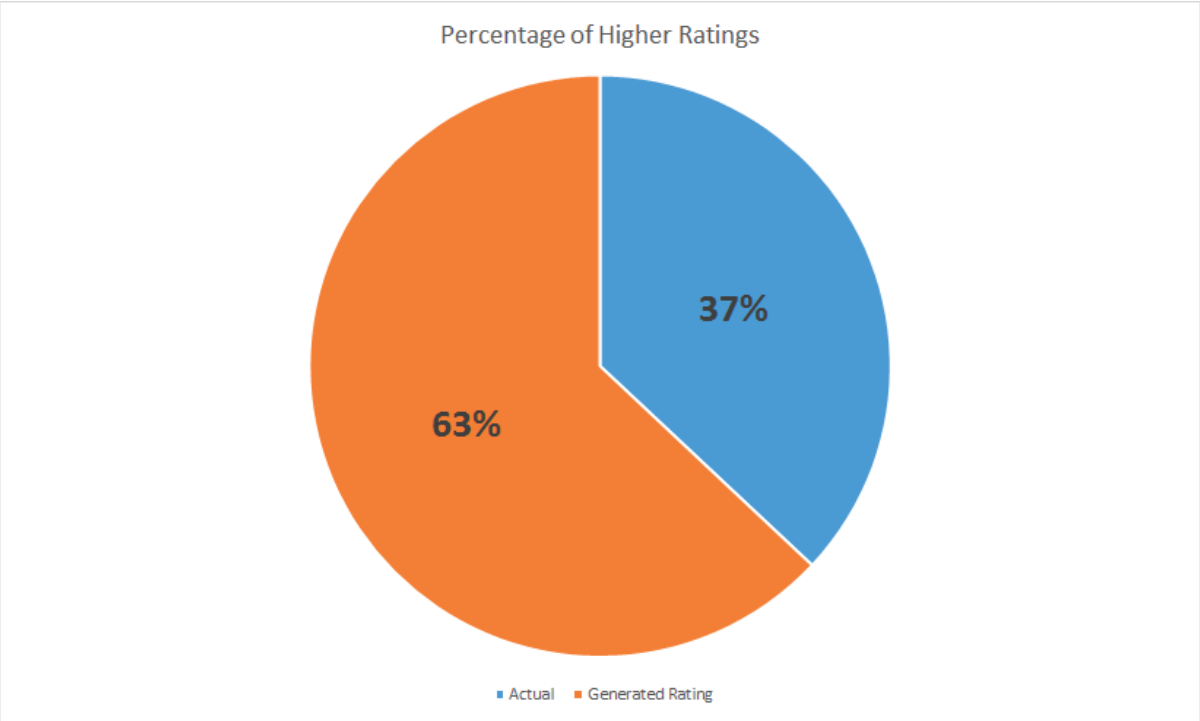

Figure 4.4: Overview of actual ratings vs generated ratings for second scenario

4.4.1 General Results

In figure 4.5 we have the result of generating recommendations for 100 users using the genetic algorithm. We can see that the ratings for the generated recommendations are higher than the actual ratings more often than the ratings from the first two algorithms. The most probable reason we have such a rating is because of our evolutionary approach. In each iteration, the preference vector gets mutated to find users who will be the best neighbours. Thus the end result tend to have users who are the most likely to be the match of the active user and have similar tastes.

The algorithm ran based on a negative rating. This coupled with the fact no other user gave the movie such a negative rating makes it difficult to form a neighbourhood. The fact that we were able to make quality recommendations indicates that this algorithm is well suited for neighbourhood formation without user interaction.

Table 4.4: Results of running first algorithm ten times for second scenario

Actual ratings	**Generated Ratings**
38%	62%
47%	53%
39%	61%
36%	64%
40%	60%
41%	59%
38%	62%
43%	57%
34%	66%
39%	61%

4.4.2 Rating Distribution

After getting an overview of the algorithm's prowess, we ran it ten more times to see how well it fares when we repeat the tests. The results are shown in figure 4.6 we have the distribution of how many users had higher generated ratings than actual ratings. We can see that we had higher ratings for the generated items for 83 users out of the 100 users we ran the tests for. This is an excellent result because it shows our algorithm works even under the worst conditions.

The next step is to run the test for ten more times so we can validate whether our first run was a fluke or our algorithm actually performs well. The results are shown in table 4.5 for 100 different users for 10 runs. We can see that we consistently have ratings for the movies we've recommended higher than the actual ratings of the items user has rated. An impressive feat since that about 80% users have the ratings higher while only about 20% have the actual ratings to be higher.

Though the algorithm is very promising, it is not optimized for speed and can take a while to complete depending on many factors. We chose to run the algorithm for only 10 generations to limit the running time. The running time increases with increasing values for generations, the number of users to compare against and the number of genres we have to consider.

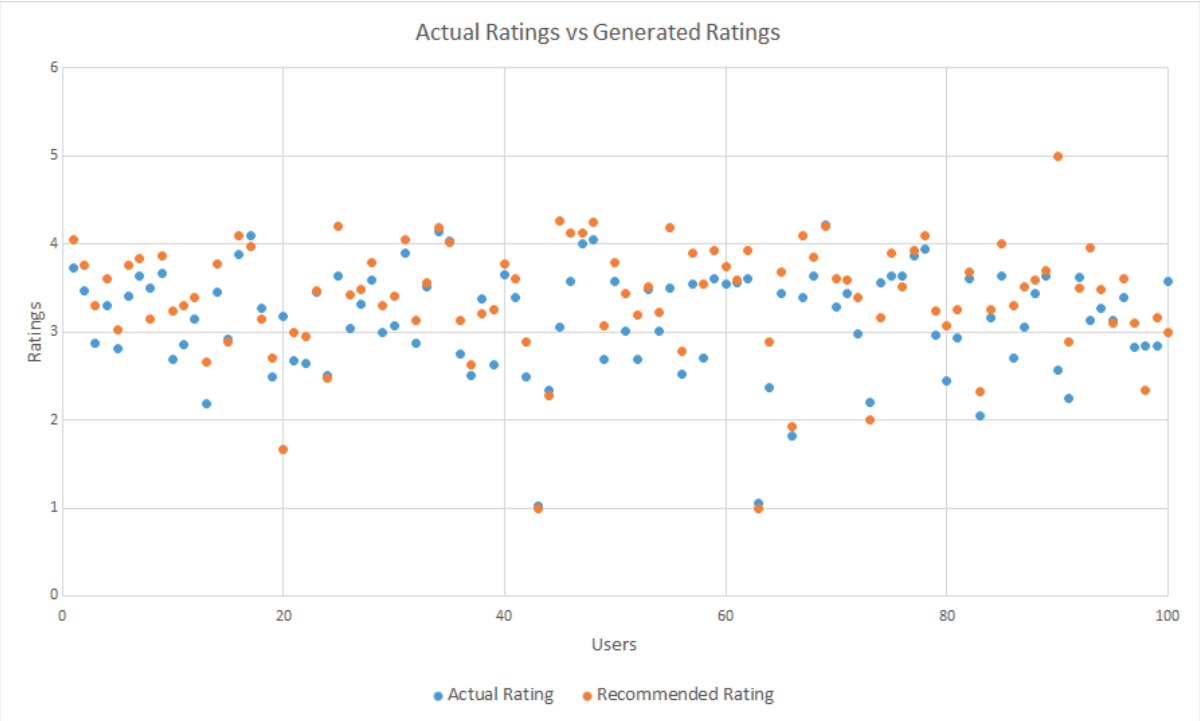

Figure 4.5: Overview of actual ratings vs generated ratings for third scenario

4.5 Precision and recall

Precision and recall is useful in recommendation systems because they tell us how successful we are at able to retrieve items that user has found useful in the past. We can use the same metrics to see if we are able to make recommendations that user will enjoy.

4.5.1 Algorithm 1

In table 4.6 we have calculated precision and recall of 100 users. The table covers all the three scenario we are tackling. Looking at the first scenario's precision and recall, we see that the precision of our recommendations varies from around 0.0008 for user 7 to as high as 0.20 for user 23. The recall is nearly around 1.0 indicating that of the items we generate, a lot of them are part of the movies the user rated.

The reason for such low precision in some cases is because we are able to generate more recommendations that the user has not rated. Due to the fact we're limiting the testing set of the user's movies to

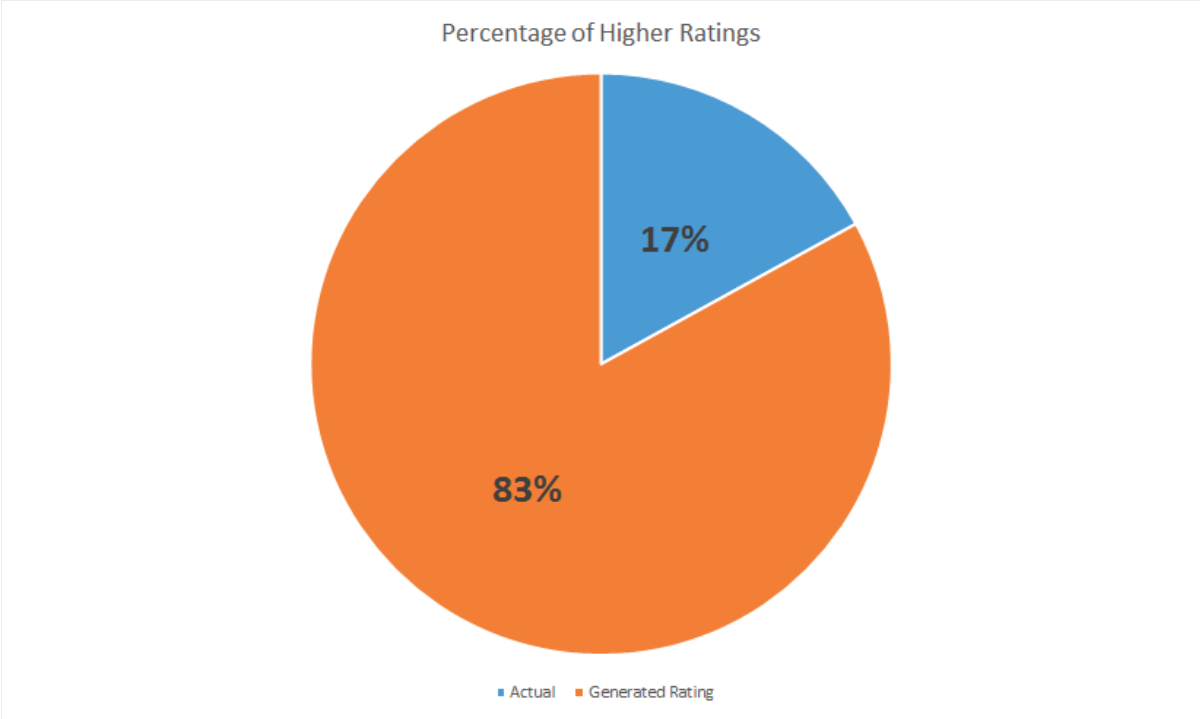

Figure 4.6: Overview of actual ratings vs generated ratings for third scenario

exclude a few genres limit the number of ratings we have. However, we have many neighbours to make recommendations from. This also coincides with the higher recall values because due to the fact we can generate a lot of recommendations, we can recall more relevant items to recommend to the user. The higher recall values indicates that of the movies we recommended, a lot of the movies the user actually watched.

4.5.2 Algorithm 2

We can refer to table 4.6 to see the precision and recall values when generating recommendations for 100 different users for the second algorithm. It is interesting to note that both the precision and recall values are very similar to the first algorithm. It indicates that both algorithms are able to make very similar recommendations in terms of what the user prefers. The recall values are also very high just like the first algorithm. This shows that we able to make recommendations that are very favourable to the user.

Table 4.5: Results of running first algorithm ten times for third scenario

Actual ratings	**Generated Ratings**
16%	84%
15%	85%
10%	90%
20%	80%
10%	90%
16%	84%
16%	84%
14%	86%
20%	80%
17%	83%
12%	88%

4.5.3 Algorithm 3

The third column of table 4.6 shows the precision and recall values when generating recommendations for 100 different users for the third algorithm. Compared to the other two algorithms, the precision is very high. That is we were able to recommend movies the user has already seen. This also is likely the cause of the higher ratings we saw earlier. We also see that our recall went down compare to the other algorithms. The reason for this is that our recommended movie set is fairly large. Thus, the chances of that a movie the user has watched being in our generated set is also high.

	Scenario 1		**Scenario 2**		**Scenario 3**	
User	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
1	0.00320	1.0	0.04953	0.99378	0.67848	0.41117
2	0.17202	0.94137	0.07680	0.99595	0.44258	0.69163
3	0.02758	1.0	0.02420	0.98734	0.04381	0.43037
4	0.24860	0.69531	0.01423	1.0	0.31554	0.48477

Continued on next page

Table 4.6 – continued from previous page

	Scenario 1		Scenario 2		Scenario 3	
User	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
5	0.08108	0.90196	0.06162	0.96601	0.68642	0.64610
6	0.04929	0.91304	0.01857	0.98360	0.31418	0.64114
7	0.00080	0.5	0.04214	1.0	0.22370	0.78723
8	0.06187	0.94186	0.07492	0.97975	0.03039	0.72413
9	0.00839	1.0	0.06842	0.99103	0.08812	0.68316
10	0.00876	0.90909	0.04023	1.0	0.21015	0.41108
11	0.03982	0.98214	0.06530	1.0	0.15865	0.41904
12	0.00310	1.0	0.02940	1.0	0.10344	0.61983
13	0.00241	1.0	0.11919	0.99740	0.21658	0.51732
14	0.01817	0.91176	0.05447	0.95652	0.00773	0.45
15	0.03326	1.0	0.16563	0.98165	0.3168	0.72794
16	0.00184	1.0	0.12768	1.0	0.25915	0.77966
17	0.02646	0.90476	0.06530	0.98139	0.12256	0.61764
18	0.10722	0.94117	0.00247	1.0	0.04755	0.55384
19	0.02535	0.97142	0.03869	0.98425	0.55577	0.39630
20	0.05234	0.99065	0.16068	0.99807	0.01382	0.375
21	0.16666	0.92810	0.03220	0.98113	0.16891	0.49833
22	0.01163	0.96153	0.15696	0.98065	0.29176	0.48818
23	0.20345	0.86206	0.05294	0.98843	0.03231	0.38
24	0.00837	1.0	0.05634	1.0	0.00414	0.66666
25	0.00697	1.0	0.01454	1.0	0.17069	0.34890
26	0.04734	0.97014	0.01314	1.0	0.33921	0.51641
27	0.00203	1.0	0.00030	1.0	0.24125	0.68928
28	0.00160	1.0	0.00588	1.0	0.23602	0.75886
29	0.11002	0.9875	0.04271	0.98571	0.21661	0.66858

Continued on next page

Table 4.6 – continued from previous page

	Scenario 1		Scenario 2		Scenario 3	
User	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
30	0.13623	0.95	0.01187	1.0	0.45758	0.59077
31	0.01055	1.0	0.00433	1.0	0.06979	0.59782
32	0.13692	0.98429	0.08018	0.98106	0.50452	0.63621
33	0.04980	0.95121	0.09377	1.0	0.33944	0.68265
34	0.01207	0.92307	0.14948	0.98170	0.10682	0.51552
35	0.16437	0.80782	0.21313	0.99421	0.15674	0.632
36	0.01967	1.0	0.01498	1.0	0.26853	0.57040
37	0.00431	0.875	0.01052	1.0	0.09577	0.67543
38	0.01819	1.0	0.22995	0.95979	0.18232	0.80487
39	0.00512	1.0	0.01771	1.0	0.31834	0.52394
40	0.01342	1.0	0.13622	0.97995	0.06290	0.66101
41	0.09261	0.89839	0.07056	0.99563	0.27805	0.63584
42	0.01776	1.0	0.00650	1.0	0.34132	0.50733
43	0.01701	1.0	0.01578	1.0	0.01716	0.22857
44	0.00819	0.92857	0.04797	0.95679	0.39350	0.49616
45	0.13580	0.93856	0.00835	1.0	0.02847	0.47272
46	0.02360	0.98181	0.05913	0.99479	0.25645	0.68240
47	0.16490	0.94642	0.04613	0.99333	0.18930	0.64485
48	0.01087	0.89473	0.09936	0.99664	0.06618	0.64566
49	0.08673	0.99009	0.01869	1.0	0.10994	0.60617
50	0.04411	0.97058	0.04365	0.99295	0.42302	0.54766
51	0.02081	1.0	0.11617	0.98425	0.37005	0.53230
52	0.07349	0.99275	0.06594	0.99069	0.03354	0.46428
53	0.00436	1.0	0.16037	0.98479	0.22237	0.69469
54	0.03085	0.93548	0.04769	1.0	0.33986	0.59558

Continued on next page

Table 4.6 – continued from previous page

	Scenario 1		Scenario 2		Scenario 3	
User	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
55	0.00413	1.0	0.03839	1.0	0.12908	0.56862
56	0.32081	0.84702	0.05851	0.96428	0.25625	0.53947
57	0.06220	0.82812	0.01672	1.0	0.81278	0.28640
58	0.02589	1.0	0.00928	1.0	0.52859	0.42299
59	0.10995	0.97797	0.04554	1.0	0.29946	0.64739
60	0.00536	0.83333	0.30721	0.98696	0.03409	0.66666
61	0.00629	1.0	0.03003	1.0	0.09002	0.52112
62	0.01294	1.0	0.08485	0.99636	0.50791	0.54314
63	0.07107	0.95505	0.00216	1.0	0.01311	0.39285
64	0.04474	0.96774	0.04056	0.99242	0.02390	0.53125
65	0.05689	0.97	0.16640	0.98154	0.17690	0.80508
66	0.05087	0.88709	0.07647	0.99196	0.07387	0.65079
67	0.02012	0.76315	0.00278	1.0	0.12857	0.47368
68	0.00976	0.84615	0.01609	0.98113	0.18390	0.41450
69	0.00567	1.0	0.36586	0.95012	0.11066	0.88297
70	0.00899	0.85714	0.30876	0.97841	0.15799	0.52614
71	0.0008	0.5	0.08492	1.0	0.26629	0.83541
72	0.00388	0.8	0.05139	1.0	0.39077	0.67132
73	0.01748	0.97560	0.03434	1.0	0.02371	0.26086
74	0.00551	1.0	0.05541	1.0	0.00993	1.0
75	0.15822	0.86692	0.00804	1.0	0.23543	0.62580
76	0.00425	1.0	0.06811	0.99547	0.07654	0.6
77	0.03025	0.95	0.00340	1.0	0.15629	0.87732
78	0.06271	0.94805	0.04798	0.98726	0.22161	0.78378
79	0.02306	0.97727	0.01145	0.97368	0.18312	0.31338

Continued on next page

Table 4.6 – continued from previous page

	Scenario 1		Scenario 2		Scenario 3	
User	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
80	0.00821	0.5	0.00061	1.0	0.13597	0.49773
81	0.01966	1.0	0.05049	0.99390	0.13351	0.74874
82	0.04354	0.98275	0.01083	0.97222	0.05203	0.71153
83	0.00402	0.8	0.04642	1.0	0.22287	0.72903
84	0.09105	0.96638	0.02603	1.0	0.01197	0.38095
85	0.02124	0.96153	0.00866	1.0	0.25637	0.67058
86	0.02844	0.92982	0.00123	0.66666	0.19829	0.42670
87	0.04935	1.0	0.03077	1.0	0.57318	0.55192
88	0.37219	0.87062	0.15559	0.99207	0.18342	0.75
89	0.03947	0.96428	0.01392	1.0	0.28925	0.7
90	0.04130	0.89534	0.11730	0.97680	0.00196	0.14285
91	0.04623	1.0	0.00154	1.0	0.01291	0.52941
92	0.05236	0.43661	0.02105	1.0	0.18442	0.60810
93	0.06263	0.97727	0.07027	1.0	0.08529	0.71212
94	0.15361	0.99038	0.21539	0.99426	0.02846	0.65714
95	0.02079	0.91666	0.02198	1.0	0.08037	0.59090
96	0.02621	1.0	0.00773	1.0	0.10745	0.86734
97	0.04798	1.0	0.06904	0.99111	0.5	0.44889
98	0.08876	0.97916	0.21331	0.98288	0.00425	0.375
99	0.00540	0.875	0.04272	1.0	0.51590	0.58647
100	0.02087	1.0	0.13281	0.97945	0.01401	0.46153

Table 4.6: Precision and recall of 100 users for all the scenarios

4.6 Summary

We implemented our algorithms to attack three different scenarios a user might fall in when they are introduced to a recommendation system and to make recommendations using minimal information. This was done to overcome the cold start problem caused by lack of data. We have found that all of our algorithms full-fill the criteria they are designed for. The first algorithm was able to make recommendations using the simple case where the user gave a popular movie high rating. The second algorithm was able to make recommendations when the user gave high rating to a low rated movie. This was achieved through a transitive approach. The movie the user rated was used to find similar movies. The movies were then used to form neighbourhood of users who are then used to make predictions. The last step was where the user rated a movie with low rating. This is very difficult and unlikely case. However, using an evolutionary algorithm, we were able to make recommendations again. The algorithm fared better than the first two, although at the cost of speed.

Chapter 5

Conclusion

A common problem in recommender systems is the cold start problem. This occurs when a new item or user is introduced to the system. The lack of information about the user and their preferences makes it difficult to predict what kind of items will be of interest to them. In our approach, we category the user into three different classes based on the rating they give to the first item. The cases are that the user likes a popular movie, likes a unpopular movie, or dislikes a popular movie. We developed three different solutions to these cases and used a movie database to test our algorithms using the off-line evaluation method.

The first algorithm had a straight forward approach. The user has liked a very popular movie that others have also shown interest in. This would form our base for creating neighbours. We can easily find users who have given high rating to the same movie and they would instantly form the neighbour of the current user. We find movies the neighbours have rated highly and recommend them to the user that do not fall in the genre of the movie used to form the neighbourhood. We see how many of the generated movies is actually rated by the user and find the ratings they would have given to them. We compare that rating against all of the user's ratings. We found that the ratings were more or less on par. However, the ratings of the items were slightly better thus showing us the algorithm works.

The second algorithm was a bit more difficult because the user has shown interest in a movie but no other user in the system has given same rating. We implemented the algorithm using a transitive approach. We take the movie the use has rated. We then find all the genres the movie belongs is part of and find other movies that also follow in any of the genres. The movies are then filtered out to remove

low rated movies. The next step was to get users who have rated these movies to form neighbourhood. We then recommend all the movies the neighbours have rated highly except the ones that fall in the same genre as the initial movie user rated. After running our tests and evaluating the results, we found the results to be better than our first algorithm. This is because our neighbourhood formation improved a little bit. We only have highly rated movies to form neighbours.

The third algorithm was the most difficult to setup. We had the user rate a popular movie with low rating. This is very difficult to form a neighbourhood because all the other users have given the movie high rating. We devise a genetic algorithm to solve this. We create preference vector for the active user and evolve it through generations. This causes the preference vector to be more aligned with other users in the system. Then we find users with similar preferences. The movies the users rated highly that do not fall into the genres of the movie active user rated are recommended to the user. We saw that the genetic approach had the best results out of all the algorithms we have developed for the different scenarios. This is due to the fact that the preference vector is able to find users who are very similar.

5.1 Contributions

We have developed a novel algorithm to make recommendations to user using only one rating during the early days of the user in the system.

Our main contributions are the novel algorithm which recommends users outside their areas of interest in the early stages and the genetic algorithm that can help with forming neighbourhood for the active user. We make recommendations to the user by forming neighbourhood of users based on an item they like but we remove items that are part of user's interest area. We also implemented three algorithms to handle the user as well.

The three specific algorithms we have developed to solve cold-start problem for new users are based on their first ratings:

1. Generate recommendations of items rated by users who have rated an item similarly to active user
2. Generate recommendations of items rated by users who have rated similar items as the active user's rated item using transitive approach to form neighbours
3. Generate recommendations of items rated by users who have similar preference as the active user

using GA approach

We have developed two strategies to form neighbourhood for the second and their algorith mentioned above. The first contribution is the transitive approach to forming neighbours. The algorithm we have developed first categories the items rated. Then it finds items which also falls into any of the categories. Then, neighbourhood is formed containing the users who rated the items. These users are used to then make recommendations. The second contribution is the evolutionary algorithm to form neighbourhood. In the case where we cannot reliably form neighbourhood due to taste differences, we construct preference vector for the active user. The vector is mutated throughout generations to match the preferences of other users in system. Then, we get users who have similar preference to form the neighbourhood.

5.2 Future work

There are many possibilities to further continue to improve the algorithm to increase accuracy of the results and efficiency. One of the approaches is to apply the genetic algorithm in the first scenarios to see how well it fares. We can also try out different user similarity algorithms rather than cosine measure such as Pearson correlation. The ratings our testing data contains are discrete values from 1 to 5 inclusive. A limitation of the system is that it is not built to handle half-star ratings.

There are many research areas that we conduct to further advance our solution. One of the fields we have not tested is genetic programming. That is to make programs which are evolved to solve the cold start problem. We research and apply how to introduce genetic programming into recommender systems. The GP approach would modify parts of program to help alleviate the cold-start problem. It can also evolve programs to minimize the information needed about the user. Another area of interest is to apply the GA in an interactive environment to see how well it works without mutating through probability.

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. pages 217–253, 2011.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[4] Vahab Akbarzadeh, Alireza Sadeghian, and Marcus dos Santos. Derivation of relational fuzzy classification rules using evolutionary computation. In *Fuzzy Systems, 2008. FUZZ-IEEE 2008. (IEEE World Congress on Computational Intelligence). IEEE International Conference on*, pages 1689–1693, June 2008.

[5] Xavier Amatriain, Alejandro Jaimes*, Nuria Oliver, and JosepM Pujol. Data mining methods for recommender systems. pages 39–71, 2011.

[6] David Beasley, RR Martin, and DR Bull. An overview of genetic algorithms: Part 1. fundamentals. *University computing*, 15:58–58, 1993.

[7] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Commun.ACM*, 35(12):29–38, dec 1992.

[8] Ronald J. Brachman, Tom Khabaza, Willi Kloesgen, Gregory Piatetsky-Shapiro, and Evangelos Simoudis. Mining business databases. *Commun.ACM*, 39(11):42–48, nov 1996.

[9] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[10] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper, 1999.

[11] Mark Claypool, Phong Le, Makoto Wased, and David Brown. Implicit interest indicators. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, IUI '01, pages 33–40, New York, NY, USA, 2001. ACM.

[12] Weiguo Fan, Edward A Fox, Praveen Pathak, and Harris Wu. The effects of fitness functions on genetic programming-based ranking discovery for web search. *Journal of the American Society for Information Science and Technology*, 55(7):628–636, 2004.

[13] Simon Fong, Yvonne Ho, and Yang Hang. Using genetic algorithm for hybrid modes of collaborative filtering in online recommenders. In *Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on*, pages 174–179. IEEE, 2008.

[14] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[15] Alex Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. In *Advances in evolutionary computing*, pages 819–845. Springer, 2003.

[16] Eric J. Glover and William P. Birmingham. Using decision theory to order documents. In *Proceedings of the Third ACM Conference on Digital Libraries*, DL '98, pages 285–286, New York, NY, USA, 1998. ACM.

[17] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[18] David E Goldberg and John Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[19] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[20] Vu Ha and Peter Haddawy. Toward case-based preference elicitation: Similarity measures on preference structures. In *In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 193–201, 1998.

[21] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[22] John H Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[23] Chein-Shung Hwang, Yi-Ching Su, and Kuo-Cheng Tseng. Using genetic algorithms for personalized recommendation. In Jeng-Shyang Pan, Shyi-Ming Chen, and NgocThanh Nguyen, editors, *Computational Collective Intelligence. Technologies and Applications*, volume 6422 of *Lecture Notes in Computer Science*, pages 104–112. Springer Berlin Heidelberg, 2010.

[24] Cezary Z Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13(2-3):189–228, 1993.

[25] Heung-Nam Kim, Inay Ha, Kee-Sung Lee, Geun-Sik Jo, and Abdulmotaleb El-Saddik. Collaborative user modeling for enhanced content filtering in recommender systems. *Decision Support Systems*, 51(4):772, 2011.

[26] Hyun-Tae Kim, Eungyeong Kim, Jong-Hyun Lee, and Chang Wook Ahn. A recommender system based on genetic algorithm for music data. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 6, pages V6–414–V6–417, April 2010.

[27] William B Langdon, Riccardo Poli, Nicholas F McPhee, and John R Koza. Genetic programming: An introduction and tutorial, with a survey of techniques and applications. In *Computational Intelligence: A Compendium*, pages 927–1028. Springer, 2008.

[28] S Marsili Libelli and P Alba. Adaptive mutation in genetic algorithms. *Soft Computing*, 4(2):76–80, 2000.

[29] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Syst.Appl.*, 41(4):2065–2073, mar 2014.

[30] Weiyang Lin, Sergio A. Alvarez, and Carolina Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data Min.Knowl.Discov.*, 6(1):83–105, jan 2002.

[31] Greg Linden, Steve Hanks, and Neal Lesh. Interactive assessment of user preference models: The automated travel assistant. In *In Proceedings of the Sixth International Conference on User Modeling*, pages 67–78. Springer, 1997.

[32] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. pages 73–105, 2011.

[33] Balabanović Marko and Yoav Shoham. Fab: Content-based, collaborative recommendation. *Commun.ACM*, 40(3):66–72, mar 1997.

[34] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[35] Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Proceedings of the 3rd International Workshop on Web Information and Data Management*, WIDM '01, pages 9–15, New York, NY, USA, 2001. ACM.

[36] *MovieLens dataset, http://grouplens.org/datasets/movielens/*, as of 2014.

[37] Heinz Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In *PPSN*, volume 92, pages 15–25, 1992.

[38] Verónika Peralta. Extraction and integration of movielens and imdb data. Technical report, Technical Report, Laboratoire PRiSM, Université de Versailles, France, 2007.

[39] Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. *A field guide to genetic programming*. Lulu. com, 2008.

[40] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, and John Riedl. Getting to know you: Learning new user preferences in recommender systems. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, IUI '02, pages 127–134, New York, NY, USA, 2002. ACM.

[41] Namitha Ann Regi and Rebecca Sandra. A survey on recommendation techniques in e-commerce. *International Journal of Engineering Research & Technology*, 2(2):1586–1590, 2013.

[42] James Salter and Nick Antonopoulos. Cinemascreen recommender agent: Combining collaborative and content-based filtering. *IEEE Intelligent Systems*, 21(1):35–41, jan 2006.

[43] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1989.

[44] A L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206–226, Jan 2000.

[45] Arthur L Samuel. Ai, where it has been and where it is going. In *IJCAI*, pages 1152–1157, 1983.

[46] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2Nd ACM Conference on Electronic Commerce*, EC '00, pages 158–167, New York, NY, USA, 2000. ACM.

[47] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[48] Badrul Sarwar, Joseph Konstan, Al Borchers, and John Riedl. Applying knowledge from kdd to recommender systems. Technical report, 1999.

[49] Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, Jon Herlocker, Brad Miller, and John Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, CSCW '98, pages 345–354, New York, NY, USA, 1998. ACM.

[50] J. Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[51] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Min.Knowl.Discov.*, 5(1-2):115–153, jan 2001.

[52] Toby Segaran. *Programming collective intelligence: building smart web 2.0 applications.* " O'Reilly Media, Inc.", 2007.

[53] Guy Shani. Tutorial on evaluating recommender systems. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 1–1, New York, NY, USA, 2010. ACM.

[54] Guy Shani and Asela Gunawardana. Evaluating recommender systems. Technical Report MSR-TR-2009-159, November 2009.

[55] B. Sheth and Pattie Maes. Evolving agents for personalized information filtering. In *Artificial Intelligence for Applications, 1993. Proceedings., Ninth Conference on*, pages 345–352, 1993.

[56] Amit Singhal. Modern information retrieval: a brief overview. *BULLETIN OF THE IEEE COMPUTER SOCIETY TECHNICAL COMMITTEE ON DATA ENGINEERING*, 24:2001, 2001.

[57] Sanjeevan Sivapalan, Alireza Sadeghian, Hossein Rahnama, and Asad M. Madni. Recommender systems in e-commerce. In *World Automation Congress (WAC), 2014*, pages 179–184, Aug 2014.

[58] Maloth Srinivas and Lalit Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, June 1994.

[59] Thomas Tran and Robin Cohen. Hybrid recommender systems for electronic commerce. In *In Knowledge-Based Electronic Markets, Papers from the AAAI Workshop, AAAI Technical Report WS-00-04*, pages 78–83. AAAI Press, 2000.

[60] Kangning Wei, Jinghua Huang, and Shaohong Fu. A survey of e-commerce recommender systems. In *Service Systems and Service Management, 2007 International Conference on*, pages 1–5, 2007.

[61] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection*, pages 117–136. Springer, 1998.