# Recommender System using Reinforcement Learning

by

Rushikesh Bapu Sargar

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2020 by the
Graduate Supervisory Committee:

Robert Atkinson, Co-Chair
Yinong Chen, Co-Chair
Maria Elena Chavez-Echeagaray

ARIZONA STATE UNIVERSITY

May 2020

ABSTRACT

Currently, recommender systems are used extensively to find the right audience with the "right" content over various platforms. Recommendations generated by these systems aim to offer relevant items to users. Different approaches have been suggested to solve this problem mainly by using the rating history of the user or by identifying the preferences of similar users. Most of the existing recommendation systems are formulated in an identical fashion, where a model is trained to capture the underlying preferences of users over different kinds of items. Once it is deployed, the model suggests personalized recommendations precisely, and it is assumed that the preferences of users are perfectly reflected by the historical data. However, such user data might be limited in practice, and the characteristics of users may constantly evolve during their intensive interaction between recommendation systems.

Moreover, most of these recommender systems suffer from the cold-start problems where insufficient data for new users or products results in reduced overall recommendation output. In the current study, we have built a recommender system to recommend movies to users. Biclustering algorithm is used to cluster the users and movies simultaneously at the beginning to generate explainable recommendations, and these biclusters are used to form a gridworld where Q-Learning is used to learn the policy to traverse through the grid. The reward function uses the Jaccard Index, which is a measure of common users between two biclusters. Demographic details of new users are used to generate recommendations that solve the cold-start problem too.

Lastly, the implemented algorithm is examined with a real-world dataset against the widely used recommendation algorithm and the performance for the cold-start cases.

*To Aai and Anna*

# ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my thesis mentor Dr. Robert Atkinson for allowing me to work on this topic.

I want to thank my parents for their love, caring, and sacrifices for educating and preparing me for my future. I would like to acknowledge my grandparents and family; I will forever be grateful for the knowledge and values they instilled in me. I would like to acknowledge my late uncle, who taught me vital life lessons, which has helped me to come this far in life. I am thankful to my brother, sister in law, and my niece for always being their supporting and cheering me. A special thanks to George Boben and my numerous friends for enduring this journey with me and making this experience enjoyable.

Lastly, but perhaps most importantly, I am incredibly grateful to Richa Shah for always motivating and encouraging me through tough times.

I thank you all.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

# Chapter 1

# INTRODUCTION

Recommender systems were originally defined as ones in which "people provide recommendations as inputs, which the systems then aggregates and directs to appropriate recipient" [1], with the evolution of the internet and increased competition importance of the recommender system, has been increased drastically, and the term has a broader meaning, describing any system that produces individualized customer-centric recommendations as output. Now the concept has been extended from recommending products to recommending friends, movies, songs, and efforts are increasing to know more about an individual's personal preferences more clearly and precisely. Leading companies, most notably Amazon, YouTube, and Netflix, have definitively demonstrated their value and have radically transformed what customers expect from any digital experience. The utility and return of investments (RoI) of recommendations are unquestionable. Amazon, for example, directly attributes an estimated 35% of sales to their recommender system.[2] Netflix's VP of Product Innovation Carlos Gomez-Uribe and Chief Product Officer mention in a research paper [3] that the combined effect of personalization and recommendations save more than $1B per year. In addition, they also mention that a typical Netflix user loses interest in 60 to 90 seconds after reviewing 20-30 titles (only 3 in detail). This emphasizes the importance of a recommender system to a business. High-quality recommendations generated by

such systems can transform the user experience from annoying to delightful while also building long term trust and loyalty [4]

## 1.1 Recommendation Models

The basic models for recommender systems work with two kinds of data, which are (i) the user-item interactions, such as ratings or buying behavior, and (ii) the attribute information about the users and items such as textual profiles or relevant keywords.[5] Methods that use the former are referred to as *collaborative filtering* methods, whereas methods that use the latter are referred to as *content-based recommender*[5], there are other methods as well such as *knowledge-based* where recommendations are based on explicitly specified user requirements. Instead of using historical data, external knowledge and constraints are used. Some recommender systems combine these different aspects and create *hybrid recommender systems* where strengths of different systems are combined [5].Table 1.1 summarizes different recommendation models. [6]

### 1.1.1 Collaborative Filtering

Collaborative filtering models use the collaborative power of the ratings provided by multiple users to make recommendations. The basic idea of collaborative filtering methods is that these unspecified ratings can be imputed because the observed ratings are often highly correlated across various users and items. For example, consider two users named Alice and Bob, who have very similar tastes. If the ratings, which both have specified, are very similar, then their similarity can be identified by the

| Technique | Background | Input | Process |
|---|---|---|---|
| **Collaborative** | Ratings from Users U of Items I | Ratings from u of items in I. | Identify users in U similar to u, and extrapolate from their ratings of i. |
| **Content-Based** | Features of items in I | u's ratings of items in I | Generate a classifier that fits u's rating behavior and use it on i. |
| **Demographic** | Demographic information about U and their ratings of items in I | Demographic information about u. | Identify users that are demographically similar to u, and extrapolate from their ratings of i. |
| **Utility-Based** | Features of items in I. | A utility function over items in I that describes u's preferences. | Apply the function to the items and determine i's rank. |
| **Knowledge-Based** | Features of items in I. Knowledge of how these items meet a user's needs. | A description of u's needs or interests. | Infer a match between i and u's need. |

Table 1.1: Recommendation Models

underlying algorithm. In such cases, it is very likely that the ratings in which only one of them has specified a value, are also likely to be similar. This similarity can be used to make inferences about incompletely specified values. Most of the models for collaborative filtering focus on leveraging either inter-item correlations or inter-user correlations for the prediction process. It is the most familiar and widely used technology.

### 1.1.2 Content-Based Recommender Systems

In the content-based recommender systems, the features of the items, such as color, size, design, category, and other descriptive words are used to make the recommendations. For example, Bob rated a blue colored cotton shirt highly, and the system does not have ratings from many users. However, the item description of the blue colored cotton shirt contains similar keywords with some other products. In such cases, those products can be recommended to Bob. Content-based methods are preferred when the items do not have sufficient rating data available. The items with similar descriptors might have been rated by the active user. Therefore, the model will be able to use the ratings as well as the item descriptors to recommend the item. Some of the disadvantages of the content-based methods are[5]:

- This method may not suggest different items which do not share any similar keywords with the previously purchased items by the user. All the recommendations become obvious and unanticipated, but more relevant items are never recommended.

- Though content-based systems are considered better for the recommendation of new items, it is not effective for providing recommendations to the new users since very few ratings would be available for the user, and not many items can be recommended.

### 1.1.3 Knowledge-Based Recommender Systems

Knowledge-based recommender systems recommend items based on user's needs and requirements; it is commonly used for items that have less flow, that people do not buy as frequently as other products such as a book, cloth, office supplies, movies, Such as land, apartments, luxury items. Knowledge-based systems have functional knowledge about how a particular item could meet the needs of a particular user. In such cases, ratings may not be available abundantly; also, consumer preferences evolve. For example, a model of a car may evolve significantly over the years, and thus it is difficult to track the user's interests entirely with the help of past ratings. Knowledge-based recommender systems are closely related to content-based systems, but domain knowledge is something that plays a vital role in the case of knowledge-based recommendations. Also, knowledge-based systems are useful where multiple features are essential to driving the decision of the user, and it is rare to have the ratings available based on each of that feature. The process is facilitated with the use of knowledge bases, which contain data about rules and similarity functions to use during the retrieval process, In both collaborative and content-based systems, recommendations are decided entirely by either the user's past actions/ratings, the action/ratings of her peers, or a combination of the two. Knowledge-based systems are unique in that they allow the users to specify what they want explicitly. Knowledge-based systems also have sub-types[5]:

- Constraint-based recommender system: In which users specifies the certain features or domain specific rules and items qualifying those features are selected

for recommendation as seen in Figure 1.1, user is asked to enter the preferences to suggest precise recommendations.

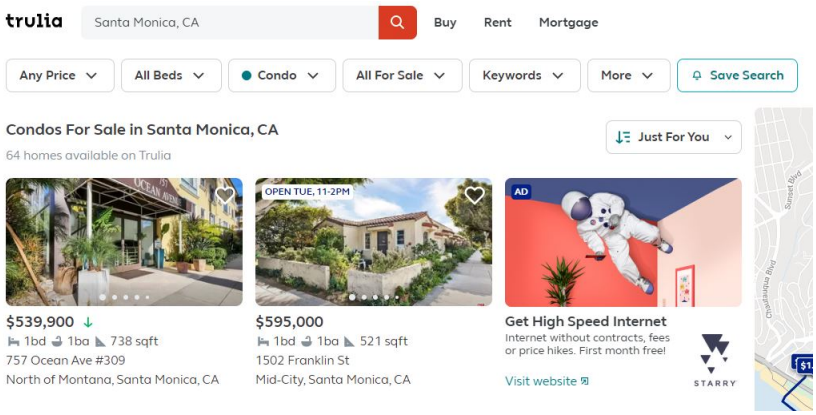



Figure 1.1: Example of Constraint Based Recommender System

- Case-based recommender system: In which the user specifies a case or item and attributes that items are then extracted, similar items are recommended. Similarity metrics are defined on the item attributes to retrieve similar items to these cases

### 1.1.4 Demographics-Based Recommender Systems

In demographic recommender systems, the demographic information about the user is used to map the preferences of the users. Patterns of the users' likes and dislikes can be studied to recommend items based on the ratings for items in a particular area, age, gender, occupation, and other demographic information[7]. Geographical location, Climate, Regional culture are also some factor which affects the preferences of the users, and hence such demographic information is a useful way to understand those patterns. For example, Certain brand of athletic shoes is being liked and rated

highly by young males between the age 15-21 in the western part of LA, with such insights, it is effective to recommend items of the same brand for that particular group. Although demographic recommender systems do not usually provide the best results on a stand-alone basis, they add significantly to the power of other recommender systems as a component of hybrid or ensemble models

### 1.1.5 Hybrid Recommender Systems

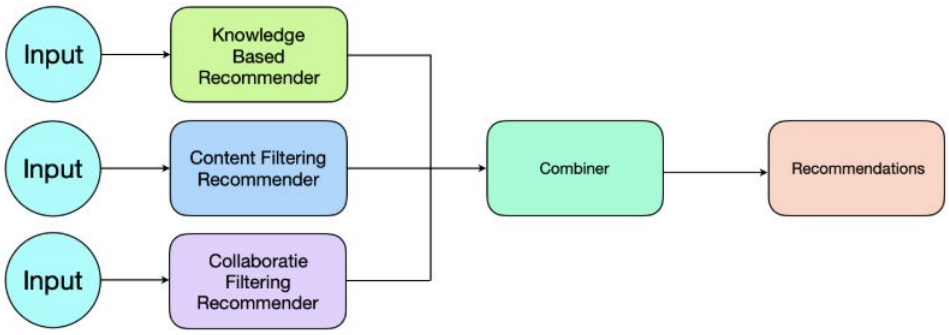



Figure 1.2: Hybrid Recommender System

Each recommender system has its advantages and disadvantages, and they may work well in different scenarios. For example, collaborative filtering depends on ratings, content-based methods depend on description and attributes, and knowledge-based systems rely on the context of the knowledge bases. Similarly, demographic systems use the demographic profiles of the users to make recommendations. Some recommender systems, such as knowledge-based systems, are more effective in cold-start settings where a significant amount of data is not available. Other recommender systems, such as collaborative methods, are more effective when much data is avail-

able.

As seen in Figure 1.2 (Source: packtpub.com), In Hybrid recommender systems, recommender systems are combined to create a more robust model, it not only uses multiple data sources, but they are also helping to improve the effectiveness of particular recommender system, e.g., combining multiple types of collaborative filtering methods. For Example, Collaborative filtering and Content-based approach can be combined and the weighted average of both can be calculated. The rank of each item being recommended could be the measure for the weight. The top items with the high weighted average can be recommended to the user.

## 1.2 Use of Deep Learning and Machine Learning

Though the underlying concepts of using models described previous section to recommend items is common, the approaches used to implement those have improved a lot with the improvement in the technology. Matrix factorization based methods are used to reduce the dimensions of the ratings matrix and approximate it by two or more small matrices with k latent components.[5] Decision trees are frequently used in data classification, The decision tree forms a predictive model which maps the input to a predicted value based on the input's attributes.[8]

A neural network based approach uses the concept of preceptron learning to predict the missing rating of the particular item[5]. Figure 1.3[5] shows how ratings for known movies are used to predict the missing rating.

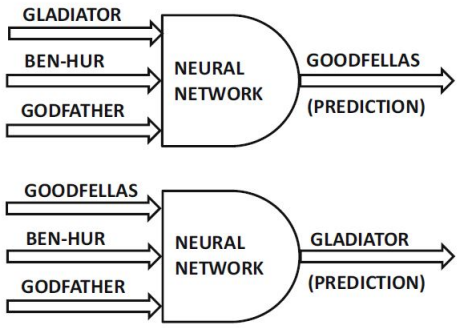



Figure 1.3: Use of Neural Network for Predicting the Rating of Missing Entry

## 1.3 Issues and Challenges

As there are many advantages of the mentioned recommender systems, none of them work perfectly in every situation and each has its own drawbacks [9]

### 1.3.1 Cold-Start Problem

The cold-Start problem can be simply understood as 'lack of data'[10], when a new user or new item is added to the system. In such cases, neither the preferences of the user can be predicted, not the new items be rated by users leading to poor recommendations. The cold-start problem can be solved easily by either asking users to rate some items at the beginning or asking them to explicitly state their taste. Other options are to consider the demographic information of the users to suggest initial items based on ratings from users sharing similar demographics, similar can be done with items where their descriptors and attributes can be used like in content-based systems. This approach has been proposed in this thesis to tackle the cold-start problem.

### 1.3.2 Grey Sheep Problem

In case of pure Collaborative Filtering where ratings given by user do not match with any group and therefore systems fail to provide good recommendations, Content-based recommender systems can solve this problem; such users can be identified and separated from other by applying offline clustering techniques like k-means [9]

### 1.3.3 Changing Data and Preferences

It is difficult to keep up with the trends; in the current world, trends change quickly, and data becomes stale rapidly. Clearly an algorithmic approach will find it difficult if not impossible to keep up with fashion trends. [10]

It is not completely useful to rely completely on the historical data; it becomes important to learn from users' current feedback and update the strategies accordingly. For Example, if a user has been ordering a particular brand of clothing and suddenly changes the brand due to historical ratings, the system may still recommend the previous brand if it is not able to capture the change.

## 1.4 Motivation

Growing online presence of the large population of the society has increased the importance of recommending the best things to users as per their needs and continuously evolving as they change. Recommender systems have been getting involved in every business, be it Groceries, Movies, Music, News Articles, Real Estates, or even life companions. Keeping this in mind having a robust approach to generate recommen-

dations is important, and research in this field has become crucial. Microsoft has recently published an article [11] stating the hot research topics in the field of recommender systems, which tries to suggest the future of recommender systems and how can the existing problems be addressed and what industries might be expecting from recommender systems in future. It focuses on several aspects, such as the application of deep learning, knowledge graph, reinforcement learning, user profiling, and explainable recommendations.

With the latest techniques in deep learning and machine learning scope of improving the recommender systems has increased since most of the established recommender systems are based on the historical data, and a specific type of supervised learning model is trained to capture the underlying preferences over the different kinds of items. Such models are generally static since they only focus on historical data. Fortunately, user feedback generated in such a process will not only complement any insufficiency of the historical data but also help to uncover user characteristics for the current stage. Reinforcement learning lays the technical foundation for utilizing user feedback for a recommender system The article asks for users to explore these opportunities and to work on many technical advancements in reinforcement learning-based recommendations. For one, helping reinforcement learning algorithms to adapt to limited data sets. Today, mainstream deep reinforcement learning algorithms try to avoid modeling the environment and instead try to learn policy directly from the user experience (model-free).[11] However, such a strategy requires a considerable amount of empirical data that is typically limited in scale and sparse in reward.

## 1.5 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.[12]. Though reinforcement learning and supervised learning, use the mapping between the input and output. The difference is unlike supervised learning where feedback provided to the agent is the correct set of actions for performing a task; reinforcement learning uses rewards and punishment as signals for positive and negative behavior.

As compared to unsupervised learning, reinforcement learning is different in terms of goals. While the goal in unsupervised learning is to find similarities and differences between data points, in reinforcement learning, the goal is to find a suitable action model that would maximize the total cumulative reward of the agent. The Figure 1.4 [13] represents the basic idea and elements involved in a reinforcement learning model.

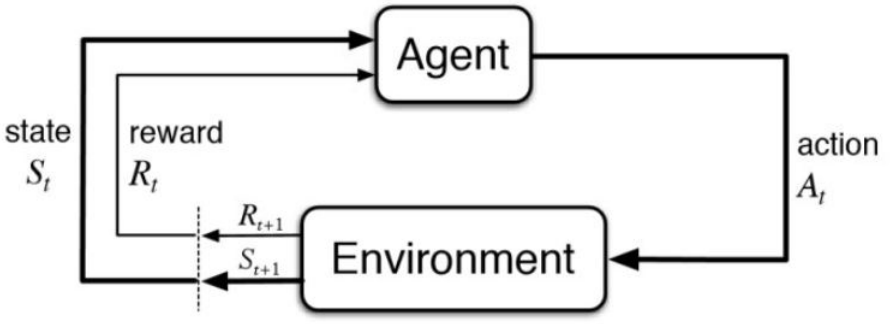



Figure 1.4: Reinforcement Learning

Some key terms that describe the elements of a Reinforcement Learning (RL) problem are:

**Environment:** Physical world in which the agent operates

**State:** Current situation of the agent

**reward:** feedback from the environment

**Policy:** Method to map the agent's state to actions

**Value:** Future reward that an agent would receive by taking action in a particular state

A Reinforcement Learning problem can be best explained through games. Let us take the game of PacMan as seen in Figure 1.5, where the goal of the agent (PacMan) is to eat the food in the grid while avoiding ghosts on its way. The grid world is the interactive environment for the agent. PacMan receives a reward for eating food and punishment if it gets killed by the ghost (loses the game). The states are the location of PacMan in the grid world and the total cumulative reward is PacMan winning the game.

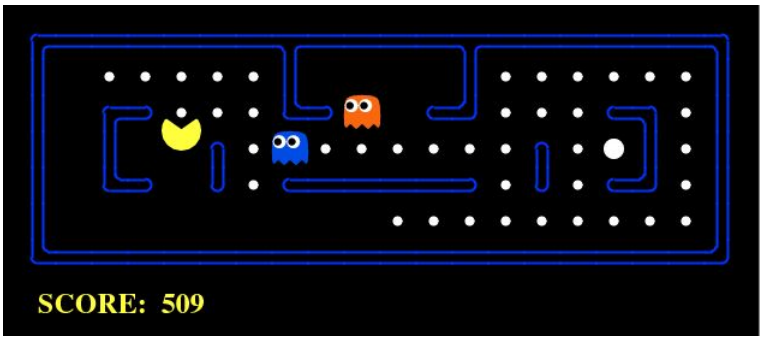



Figure 1.5: Pacman Game

In order to build an optimal policy, the agent faces the dilemma of exploring new

states while maximizing its reward at the same time. It is called Exploration vs. Exploitation trade-off.

### 1.5.1 Markov Decision Process

A Markov Decision Process (MDP) is a model for sequential stochastic decision problems [13] MDPs are mathematical frameworks to describe an environment in Reinforcement Learning (RL) and almost all RL problems can be formalized using MDPs. An MDP consists of a set of finite environment states $s$, a set of possible actions $A$ in each state, a real valued reward function and a transition function. The agent stays in particular state of particular time step $t$ $\epsilon${0,1,2,3..}, after choosing an action the agent moves to the next state $s_{t+1}$ by calling a function T($s_t$,$a_t$) and it receives a required $r_t$ from the environment by reward function R($s_t$ , $a_t$ , $s_{t+1}$). Based on a policy $\pi$(s), the action is selected in a specific state s. RL can solve MDP. RL aims to find the optimal policy $\pi^\star$ that maximizes the expected cumulative reward $G$ which is called return. In RL, the optimal policy can be learned by a state-action value function $Q_\pi$(s, a) which means the expected value of the return $G$ obtained from episodes starting from a certain state $s$ with the action a.$Q_\pi$(s, a) can be expressed as follows:

$$Q_\pi(s,a) = E_\pi\{G_t|s_t = s, a_t = a\} \tag{1.1}$$

$$= E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s, a_t = a\} \tag{1.2}$$

### 1.5.2 Q-Learning

Q-learning is a commonly used model-free approach that can be used for building a self-playing PacMan agent. It revolves around the notion of updating Q values, which denotes the value of doing action $a$ in state $s$. The value update rule is the core of the Q-learning algorithm. The Q-learning algorithm tries to learn a policy that selects the best action for every state in the state space. The algorithm tries to calculate the q value for every action at every state. Following are the variables involve in the Q-learning algorithm[14]:

- Learning Rate ($\alpha$): Learning rate, which is usually $\alpha_t$, decides how much the new information overrides the old one. Value of $\alpha_t$=1 means the algorithm only considers the latest information, and $\alpha_t$=0 does not learn anything new. In the deterministic environment, a learning rate $\alpha_t$=1 is optimal. In practice, a constant learning rate is used, such as $\alpha_t$=0.1 for all t.

- Discount Factor ($\gamma$): The discount factor $\gamma$ decides the role of future rewards. Value of discount factor $\gamma_t$=0 makes the algorithm short-sighted and considers only the current rewards; more the discount factor it will look for the long term high reward.

- Exploration co-efficient ($\epsilon$): This coefficient tries to balance the trade-off between exploration and exploitation. The value of coefficient states the probability of choosing a random action over the one which is being suggested by the maximum Q value.

## 1.6 Biclustering

Clustering is considered as an unsupervised method to group objects based on their features, unlike traditional clustering methods which considers all the features, biclustering algorithms try to find local patterns using the only subset of features, and it is one of the important factors of the designed recommender system. The biclustering algorithm tries to cluster the rows and columns of the data matrix simultaneously. These techniques are usually used in the analysis of the gene expression data since a gene can take part in several biological pathways that can be active under specific conditions only. There are several biclustering algorithms have been developed in recent years.

Kemal Eren provides an example in his blog to explain biclustering[15]; he considers a clustering problem with 20 samples and 20 features. With the original data arranged in the form of matrix does not help to observe the clusters immediately as seen in Figure 1.6[15]

Here is a scatterplot seen in Figure 1.7[15] of the first two dimensions, with cluster membership designated by color.

As seen in Figure 1.7[15] , the clustering approach considers only the first two dimensions, and as the number of dimensions increases clustering the data points gets tougher, Biclustering algorithms consider all the features, but data points in individual clusters share similarity on a subset of features.

By rearranging the rows of the matrix, the samples belonging to each cluster can be made contiguous. In the rearranged matrix, the correct partition is more obvious

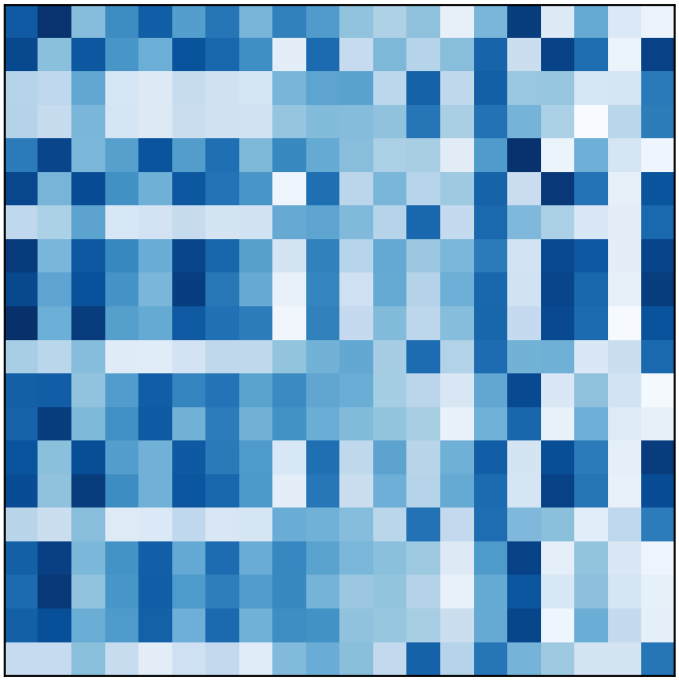


Figure 1.6: Matrix Without Clustering

1.8[15]: As observed, three biclusters can be easily observed, reshuffling the rows and columns of the matrix; this can also be seen as clustering both rows and columns simultaneously; The algorithm usually returns two boolean lists per bicluster, which tell which rows and columns are included in that particular biclusters. Reshuffling rows and columns make the cluster contiguous and easily visible in the matrix. To explain this simultaneous clustering Kemal Eren gives an example :

***Example : Throwing a Party***

*Bob is planning a housewarming party for his new three-room house. Each room has a separate sound system, so he wants to play different music in each room. As a conscientious host, Bob wants everyone to enjoy the music. Therefore, he needs to*

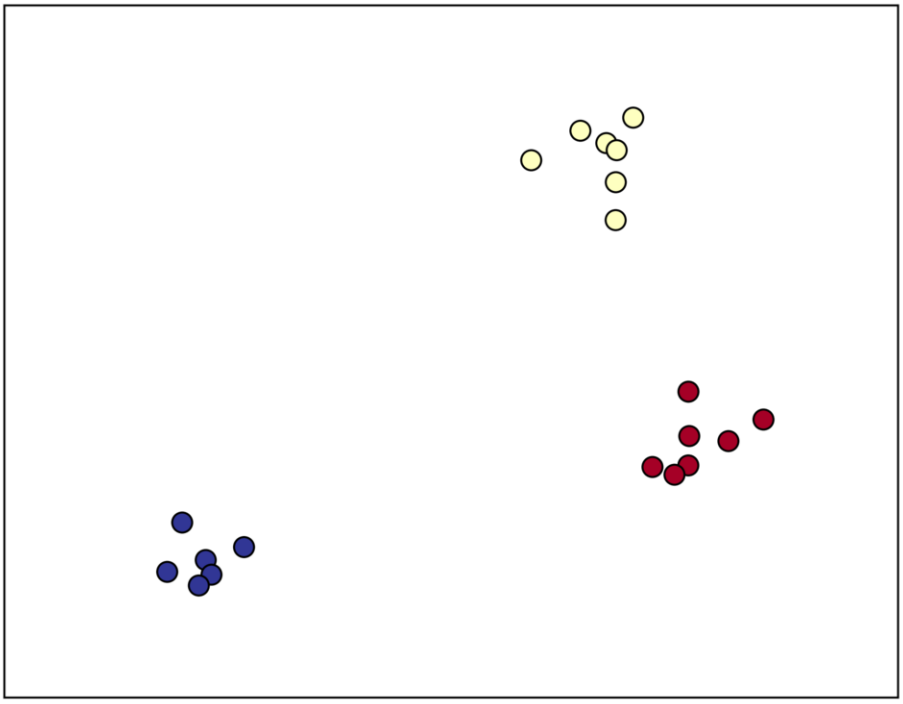


Figure 1.7: Scatterplot of the First Two Dimensions

*distribute albums and guests to each room in order to ensure that each guest hears their favorite songs.*

*Our host has invited fifty guests, and he owns thirty albums. He sends out a survey to each guest, asking if they like or dislike each album. After receiving their responses, he collects the data into a 50 X 30 binary matrix M, where* $M_{ij} = 1$ *if guest i likes album j*

*In addition to ensuring everyone is happy with the music, Bob wants to distribute people and albums evenly among the rooms of his house. All the guests will not fit in one room, and there should be enough albums in each room to avoid repetitions.*

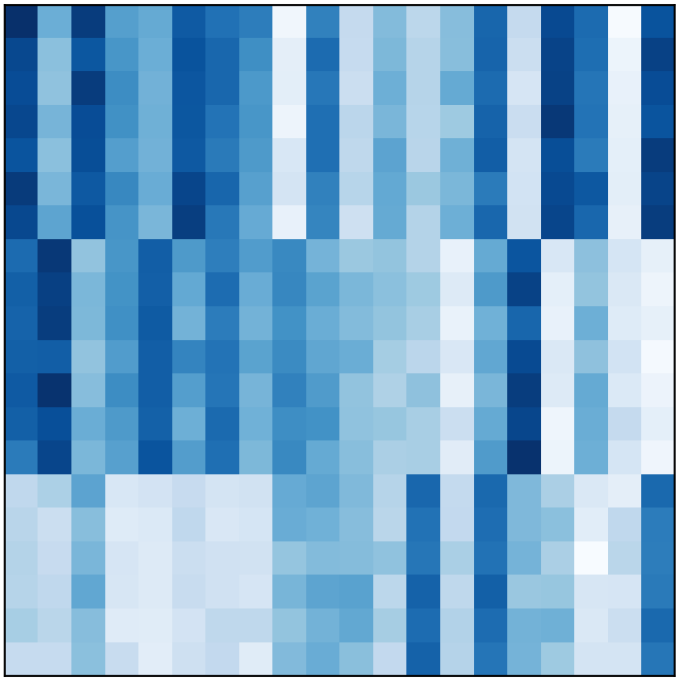


Figure 1.8: Biclusters in Matrix

*Therefore, Bob decides to bicluster his data maximize the following objective function:*

$$S(M, r, c) = b(r, c). \sum_{i,j,k} M_{ij} r_{ki} c_{kj}$$

*where $r_{ki}$ is an indicator variable for membership of guest $i$ in cluster $k$, $c_{kj}$ is an indicator variable for album membership, and $b \in [0,1]$ penalizes unbalanced solutions, i.e., those with biclusters of different sizes. The objective function tries to calculate the score for a particular arrangement in clusters. If user $i_1$ likes album $j_1$, its entry in M(1,1) will be 1. If both $i_1$ and $j_1$ are placed in the same cluster $k_1$, It will contribute a positive score to function. Similarly, membership of all users will add up to the objective function score of the particular setup. Setup with the highest score gives the best biclusters.*

*Bob uses the following algorithm to find his solution: starting with a random as-*

*signment of rows and columns to clusters, he reassigns row and columns to improve the objective function until convergence. In a simulated annealing fashion, he allows suboptimal reassignments to avoid local minima. However, this algorithm is not guaranteed to be optimal. Had Bob wanted the best solution, the naive approach would require trying every possible clustering, resulting in* $k^{n+p} = 3^{80}$ *candidate solutions. This suggests that Bob's problem is in a non-polynomial complexity class. Most formulations of biclustering problems are in NP-complete.After biclustering is applied, the room will consist of not only similar persons but also the albums over which they are similar.*

This example gives a good understanding of how biclustering algorithms work and what they try to achieve. A bicluster only considers a subset of rows and a subset of columns of the original matrix. After an exchange of rows and columns, the rows and columns define a contiguous submatrix of data matrix see Figure 1.9 Biclustering

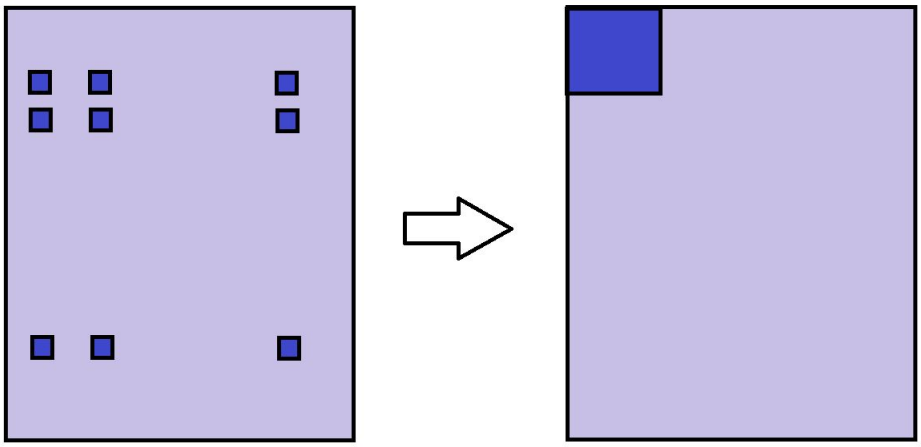
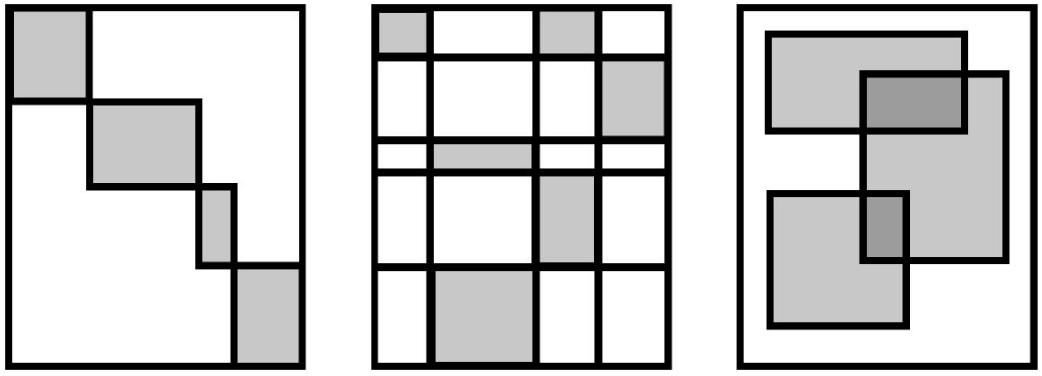


Figure 1.9: Contiguous Biclusters by Row and Column Exchanges

algorithms differ widely in their problem formulation. Also, the structure of the bicluster may differ; in some cases, the rows of biclusters share some relationship with some columns and not with remaining columns. In simple words, rows and columns will not be shared across the biclusters. See Figure 1.10(a) Another type of bicluster structure is checkerboard pattern-seeking tiles that meet the fitness criteria, in which two clusters do not share rows and columns simultaneously. see Figure 1.10(b). A third type of structure defines bicluster as any submatrix with some fitness property see Figure 1.10(c) [16]



(a) Partitioned structure: each row and each column assigned to exactly one bicluster.

(b) Checkerboard structure: entire data matrix partitioned; some tiles meet bicluster fitness criteria.

(c) Unrestricted structure: biclusters may overlap

Figure 1.10: Types of Bicluster Structures

There are different types of biclustering techniques:

### 1.6.1 Cheng and Church

It is a deterministic greedy approach that seeks to find biclusters with low variance, as defined by a value known as the Mean Squared Residue (MSR)

### 1.6.2 OPSM

It is the algorithm that is deterministic greedy that seeks biclusters with ordered rows. The OPSM model defines a bicluster as an order-preserving submatrix, in which there exists a linear ordering of the columns in which the expression values of all rows of that submatrix are strictly increasing from left to right [16]

### 1.6.3 BiMax

This technique seeks to bicluster 1's in the binary matrix; it uses the divide and conquer approach and recursively divides into checkerboard format since the algorithm works for binary data matrix, the dataset has to be converted into a binary representation. this process of binarization influences the performance of the technique.

## 1.7 Overview of Current Study

In this thesis, an effective method for the recommender system has been designed, which uses MovieLens dataset to generate movie recommendations for users. It involves biclustering of the movies' rating data and applies Q-learning upon arranging all the biclusters in the form of a squared grid[17]. The learned policy helps to recommend movies to the user. It considers the demographic information of users to handle the cold-start problem and also considers the fact that not all the users who rate the movies are the same, some are lenient while some are strict while rating the movie. In the next chapter, we will see how the discussed concepts are used together to design a meaningful recommender system.

# Chapter 2

# METHOD

This chapter briefly describes the proposed approach for the recommender system, In which the MovieLens100k dataset is preprocessed in the format such that the biclustering algorithm can ingest it. The design uses the BiMax algorithm to bicluster the data, which identifies all the biclusters. A fitness function based on Mean Squared Residue (MSR) is used to select and order the biclusters. Ordered biclusters are arranged in a 2-dimensional square grid, which acts as an environment for the Q-learning algorithm. It learns the policy to traverse through the grid. The learned policy helps to generated recommendations for a user; details of each process are defined in the following sections.

## 2.1 Problem Definition

Thesis formulates the problem of recommender system as MDP problem which can be represented in the form of gridworld.

As seen in Figure 2.1, a gridworld is usually a 2-dimensional environment made up using rows and columns where an agent can move, each unit of the grid is called a state, and the agent can move in 4 directions. With the use of reinforcement learning, the goal of the grid-world system is to maximize the reward received by the agent

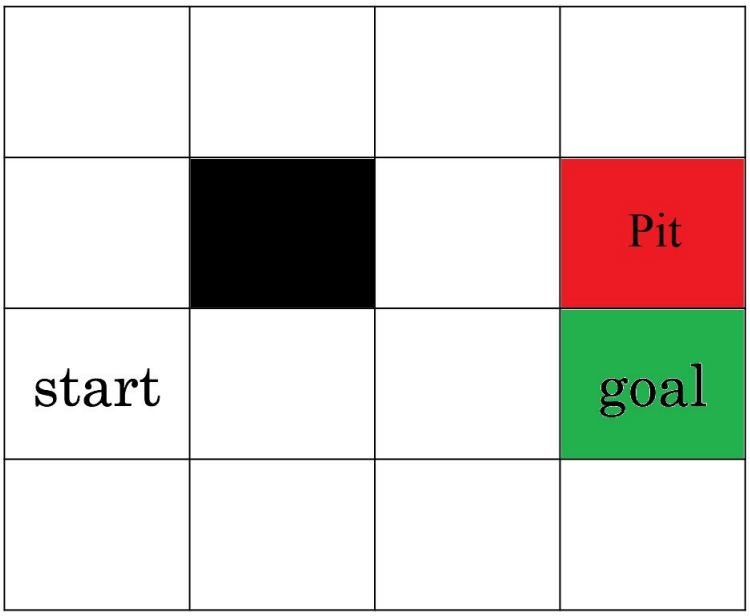



Figure 2.1: Example of Gridworld

while each action receives either a reward or gets penalized based on the reward function. User movement gathers recommendations, and reward can be considered analogous to the user satisfaction for the items suggested.

## 2.2 Proposed Approach

The proposed approach to solving this problem involves the data pre-processing, constructing the biclusters, Constructing the states of the gridworld, learning Q function, generating recommendation based on the q table, generating recommendations if no data is available for the user, comparing it with the state of the art systems to evaluate the performance.

The data which is being used for the implementation is Movielens100k.

## 2.3 Data Description

MovieLens100k is a relatively older data state, and the critical reason to select this particular dataset is it has the users' demographic details such as age, gender, occupation and zip, which can be leveraged in case there are no ratings for that particular user.

The data has 100,000 ratings by 943 users on 1682 movies, and each user has rated at least 20 movies. Along with the rating data, the dataset also has details about every movie e.g., release date, list of genres. The data set has the users' demographic information. Figure 2.2 shows the structure of the dataset.

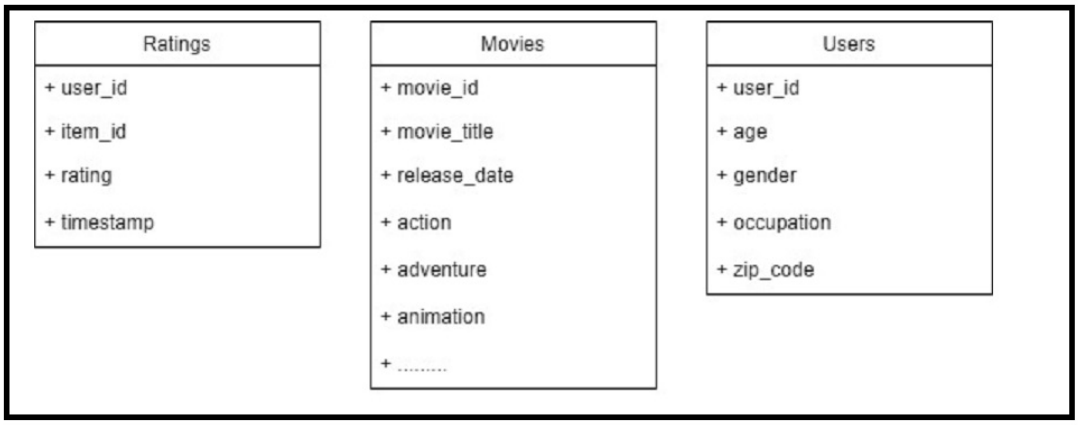



Figure 2.2: Structure of the Data

### 2.3.1 Quantifying the Demographic Details

Some of the demographic information may not be completely relevant in a particular case or can not be used in their original form, but some inferences can be made using that information, which can be useful in deriving decisions. In the demographic de-

tails, the occupation of the user does not provide details for driving decisions for the recommendations. On the other hand salary of the user can be helpful to understand what income group prefers what type of movies.[18] It also helps to create a relationship amongst different users on the same scale. Thus, the occupation of the user can be replaced by average annual income for that occupation Refer A.1 for details.

To have the flexibility all the demographic features are required to be quantified and gender code is also changed to 0, 1 values where 1 refers to male and 0 for female. Figure 2.3 shows the quantified data.

| user_id | age | annual_salary | gender_code | zip |
|---|---|---|---|---|
| 1 | 24 | 42000 | 1 | 85711 |
| 2 | 53 | 0 | 0 | 94043 |
| 3 | 23 | 57000 | 1 | 32067 |
| 4 | 24 | 42000 | 1 | 43537 |
| 5 | 33 | 0 | 0 | 15213 |
| 6 | 42 | 75000 | 1 | 98101 |
| 7 | 57 | 45000 | 1 | 91344 |
| 8 | 36 | 45000 | 1 | 05201 |
| 9 | 29 | 4000 | 1 | 01002 |
| 10 | 53 | 81000 | 1 | 90703 |
| 11 | 39 | 0 | 0 | 30329 |
| 12 | 28 | 0 | 0 | 06405 |

Figure 2.3: Example of Quantified Demographic Data

## 2.4 Data Pre-processing

Movielens100k dataset follows mentioned (Figure 2.2) structure, and it has to be converted into the format in which the program can ingest it. The biclustering approach can be applied to the 2D matrix; hence the data has to be pre-processed and converted into the 2D matrix format.

Following are the steps to get the data in the required format:

- Join Users information (i.e., user id, age, gender, occupation, zip) with occupation data, which gives approximate annual salary for each user.
- Join gender data to get the genders to have code in the resulting data; it is then converted to 1,0, where 1 refers to male and 0 to female.
- Join movie data file, which consists of information about the movie title, release date, and genre with rating file, which consists of ratings provided by users.
- After all these joins, the data will consist user's rating only for the movies which he has rated. In order to get the matrix, ratings for all the movies for all the users have to be entered. Thus the generated data set has to be pivoted with users in rows and movies in columns and replacing all the non-available ratings with 0. The biclustering followed in the implementation can not process matrix entries with no values; hence 'missing ratings' have to be replaced with zeros.

Generated data as seen in Figure 2.4 can be stored in pickle files to be picked by the biclustering algorithm to make biclusters out of it.

## 2.5 Biclusters Creation

As discussed in previous chapters, biclusters are simultaneous clustering of both rows and columns, where a subset of users get clusters based on some features. In the implemented approach, the BiMax technique has been used to create the biclusters from the data matrix.

| movie_id | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| user_id | | | | | | | | | | |
| 1 | 5.0 | 3.0 | 4.0 | 3.0 | 3.0 | 5.0 | 4.0 | 1.0 | 5.0 | 3.0 |
| 2 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 |
| 3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5 | 4.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 6 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 4.0 | 4.0 | 0.0 |
| 7 | 0.0 | 0.0 | 0.0 | 5.0 | 0.0 | 0.0 | 5.0 | 5.0 | 5.0 | 4.0 |
| 8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 | 0.0 |
| 9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 5.0 | 4.0 | 0.0 | 0.0 | 0.0 |
| 10 | 4.0 | 0.0 | 0.0 | 4.0 | 0.0 | 0.0 | 4.0 | 0.0 | 4.0 | 0.0 |

Figure 2.4: Example of the Preprocessed Data

As states in the previous chapter, The BiMax algorithm works on binary data, and it tries to find all biclusters consisting entirely of 1s and non-binary data can be converted to binary data in several ways, applying threshold being the simplest one. Following procedure is used to create biclusters [19] :

Bimax uses a recursive divide and conquer strategy to enumerate all the biclusters in m $\times$ n matrix M. The Figure 2.5 shows the initial matrix M[20]. To illustrate, consider the following data matrix. The process starts by choosing a data row containing 0s and 1s. If there is no such row, that means all the entries in the matrix are 1s or all are 0s. All 1s means the entire matrix is single bicluster. All 0s means there is no bicluster in the existing data matrix. Arbitrarily choose the first row $r_1$ of M to divide the matrix into two submatrices.

The submatrices can be found by dividing the columns C={1,2,3,....,n} into two sets, those for which row $r_1$ is 1, and those for which it is 0.

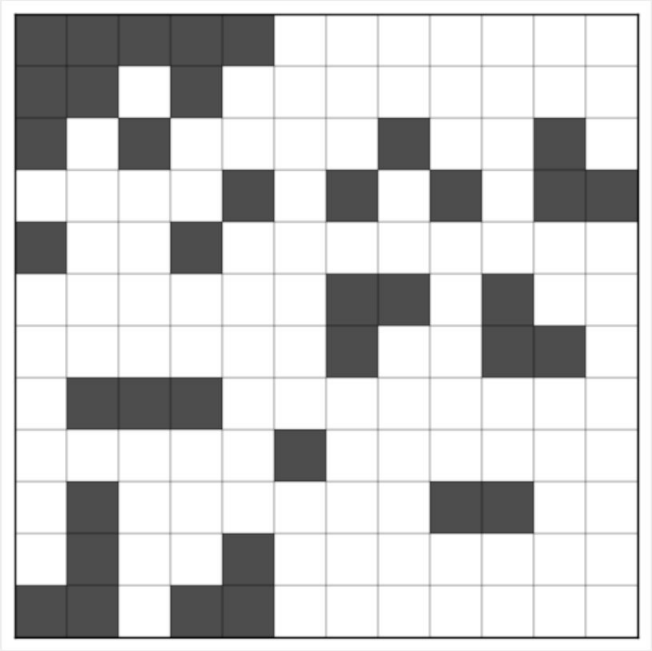


Figure 2.5: Sample Data Matrix

- $C_U = \{c : M[r_1, c] = 1\}$
- $C_V = C - C_U$

In the next step the m rows of M are divided into three sets:

- $R_U$: rows with 1s only in $C_U$
- $R_W$: rows with 1s in both $C_U$ and $C_V$
- $R_V$: rows with 1s only in $C_V$

After applying the above procedure, the group of columns in $C_U$ and $C_V$ may not be contagious, and the same goes for the sets of rows i.e., $R_U$,$R_W$ and $R_V$.

These sets can be made contagious by shuffling them e.g. in 10 column matrix, if 1,2,8,9 columns are part of $C_U$, then the 8th column can be exchanged with the 3rd

column and 9th with 4th column, which makes all columns in each set contagious.

After arranging the rows and columns of M, the matrix looks like Figure 2.6 [20]

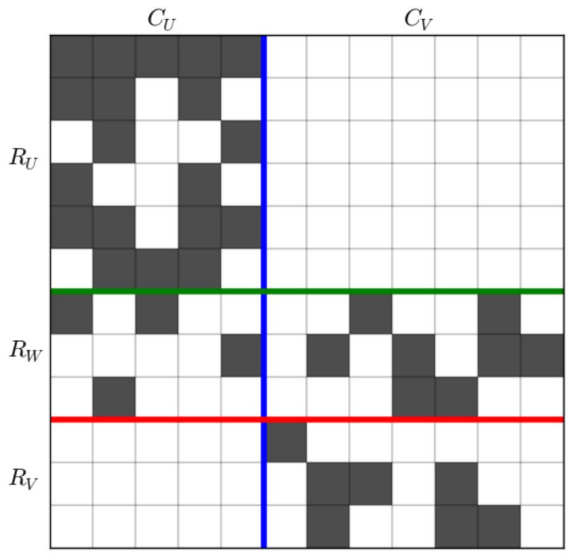



Figure 2.6: Data Matrix After Applying the Technique and Shuffling the Rows and Columns

Now since we have a better idea of how the divide and conquer process works for the BiMax algorithm. The submatrix formed by ($R_U$,$C_V$) is empty and can not contain any biclusters. The submatrix $U = (R_U \cup R_W, C_U)$ and $V = (R_V \cup R_W, C_U \cup C_V)$ contain all possible biclusters in M as seen in Figure 2.7[20]. The recursive process is performed in U and then in V.

## 2.6 Arranging Biclusters with Their Similarity

Created biclusters have to be arranged in the form of the grid and thus it becomes important to arrange them in specific order so that the nearer biclusters share max-

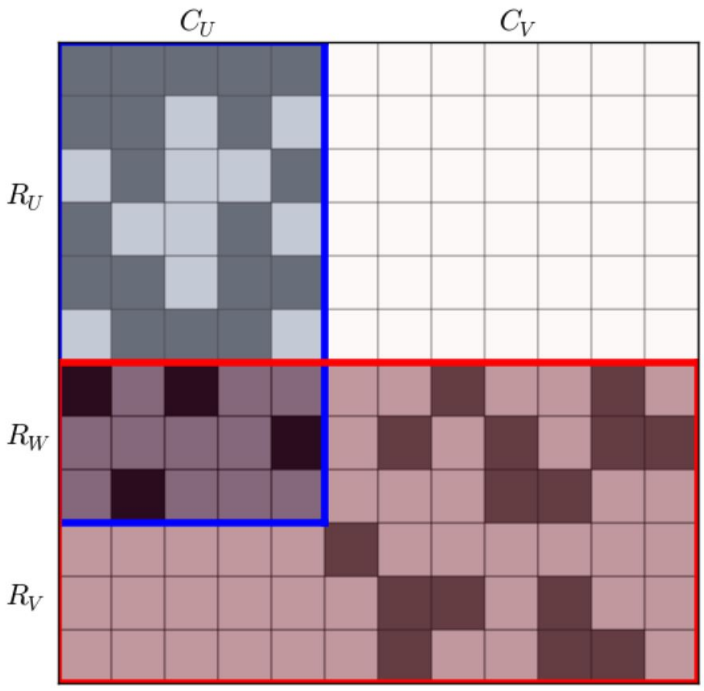



Figure 2.7: Matrix Showing U and V with Blue and Red Color Respectively

imum similarity. This helps the agent to get the maximum reward quickly since it has only four actions to take in the grid.

### 2.6.1 Selection of Biclusters

The proposed biclustering algorithm considers the binarized form of the data, and it identifies the every possible bicluster of the grid of various sizes and different value of intra-bicluster co-relation. In order to form the squared grid the number of biclusters has to be limited to certain value i.e $n^2$ Since the number of biclusters considered in the grid have to be limited several ways are followed to limit the number

- Select the biclusters with specific size, i.e. filtering the biclusters which have specific rows and columns in it. This also helps to increase the number of

recommendations when the bicluster is being preferred from a certain start state. Details will be discussed in upcoming sections.

- Select the biclusters which have high correlation amongst the included rows and columns. This can also be referred as fitness of the bicluster. This ensures that only the good quality of biclusters are considered in the grid.

### 2.6.2 Ordering the Biclusters

Filtered biclusters have to be arranged a certain way to have similarity with the nearer biclusters. This can be done naively since Q-learning can optimize it by learning a policy.

#### Using Fitness Values

All the filtered biclusters are sorted with their fitness values using by the fitness function[21], which is uses MSR (Mean Squared Residue)[22], which is lower for better biclusters and MSR threshold and importance of number of columns and rows.

A general bicluster is represented by a matrix B of I rows (number of users ) and J columns (number of movies), where the element $b_{ij}$ is the rating of the movie j given by the user i. The MSR (Mean Squared Residue) value is calculated following these steps (Figure 2.8 summarizes the calculation):

1. Calculating the sum of values in each row and column i.e. $sum\_b_{iJ}$ *and* $sum\_b_{Ij}$ (as seen in Figure 2.8)

2. Calculation of the means $b_{iJ}$ of each row i. See equation 2.1

3. Calculation of the means $b_{Ij}$ of each column j. See equation 2.2

4. Calculation of the mean $b_{IJ}$ of the entire matrix. See equation 2.3

5. Calculation of the residue $r_{ij}$ of each matrix element. See equation 2.4

6. Calculation of MSR (Mean Squared Residue). See equation 2.5

$$b_{iJ}[i] = \sum_{j=0}^{J-1} b_{ij} = \frac{sum\_b_{iJi}}{J} \tag{2.1}$$

$$b_{Ij}[j] = \sum_{i=0}^{I-1} b_{ij} = \frac{sum\_b_{Ijj}}{I} \tag{2.2}$$

$$b_{IJ}[j] = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} b_{ij} = \frac{sum\_b_{IJ}}{I.J} \tag{2.3}$$

$$r_{ij} = b_{ij} - b_{iJi} - b_{Ijj} + b_{IJ} \tag{2.4}$$

$$MSR = \frac{\sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (r_{ij})^2}{I.J} \tag{2.5}$$

Fitness function[21] *F(I,J)* is defined as:

$$F = \frac{MSR}{\lambda} + \frac{w_c.\lambda}{J} + \frac{w_r.\lambda}{I} \tag{2.6}$$

where I and J are the set of rows and columns, respectively, in the bicluster, MSR is called the residue of a bicluster and is calculated as in Equation 2.5, $\lambda$ is a residue threshold (the maximum desired value for residue), $w_c$ is the importance of the number of columns, and $w_r$ the importance of the number of rows.

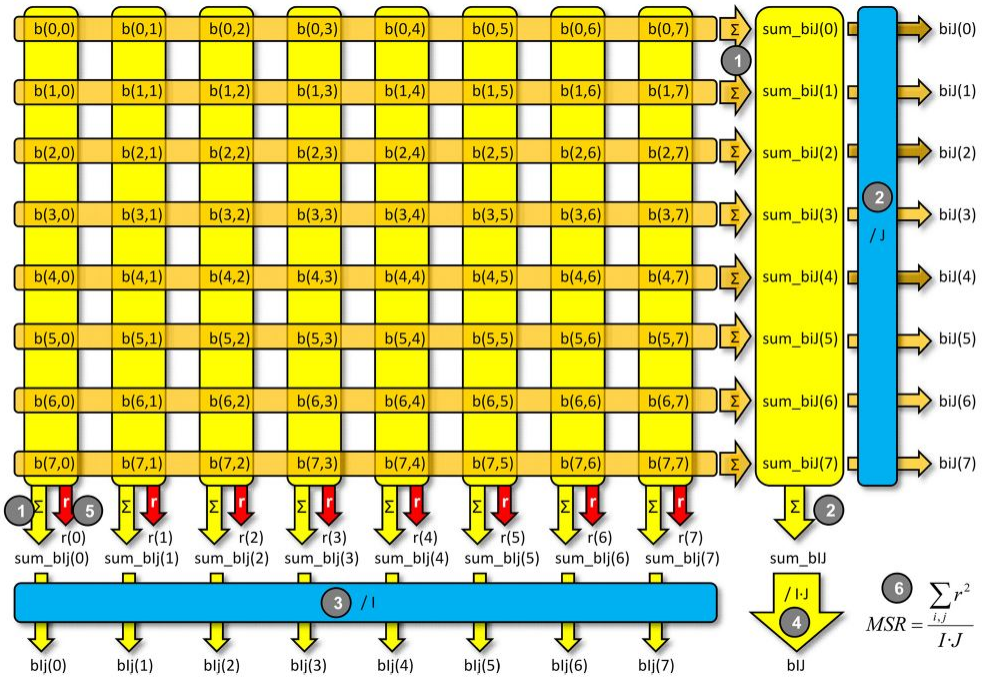



Figure 2.8: Calculation of Mean Squared Residue of Bicluster

## 2.7 Constructing the Grid

Once the biclusters are approximately sorted with their similarities with each other, they have to be arranged in a squared grid. See Figure 2.9

Challenge while arranging the sorted biclusters is to arrange it in such a way that every cluster has neighboring biclusters that share similarities with it. To achieve this concept, of the space-filling curve has been used.

### 2.7.1 Space Filling Curve

As the name suggests space-filling curves are curves that visit all possible points in multidimensional space [23] The most straightforward approach to do so is row-order

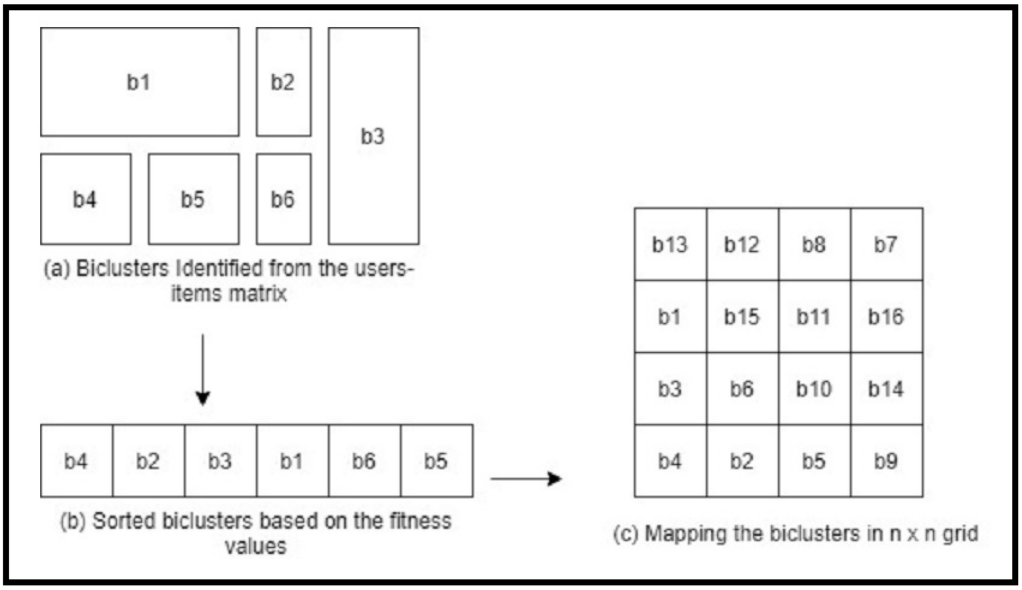



Figure 2.9: Arranging Biclusters in Grid

or column-order traversal of 2D space. Consider the example of column-order for 8×8 2D space. The value that the $C_{columnorder}$ takes for the input point is $< 1, 2 >$ can be computed as

$$C_{columnorder}(< 1, 2 >) = (1 \times 8^1) + (2 \times 8^0) = 10 \tag{2.7}$$

10th point of the curve will be in $1, 2$ of the grid, with this logic it is easy to show that $C_{columnorder}(< 1, 1 >) = 9$ and $C_{columnorder}(< 1, 3 >) = 11$. In other words, if the points in the space are neighbors along the y-axis, the column-order traversal as seen in Figure 2.10(b)[23] can place them on the traversal in such a way that they will be neighbors to each other. On the other hand, the same cannot be said about points that are neighbors to each other along the other dimensions[23].

This filling technique will place similar biclusters farther at the edges, and it would be difficult to get the maximum rewards for the agent immediately.

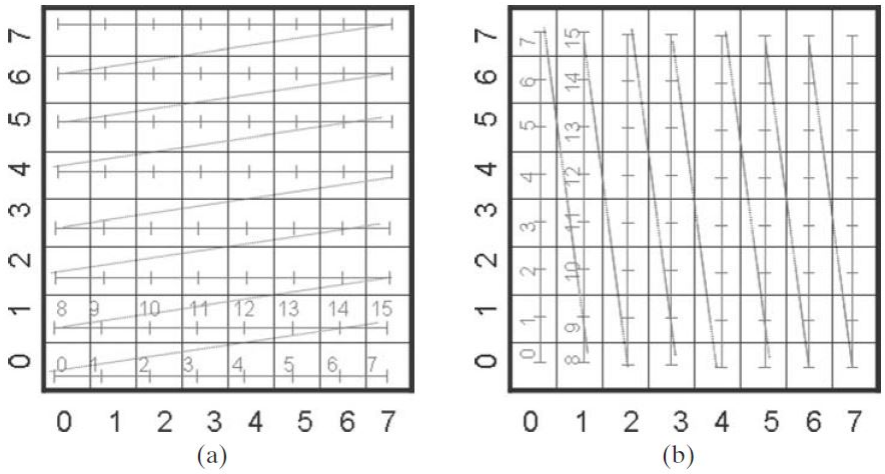



Figure 2.10: (a) Row (b) Column-Order Traversal of 2D Space

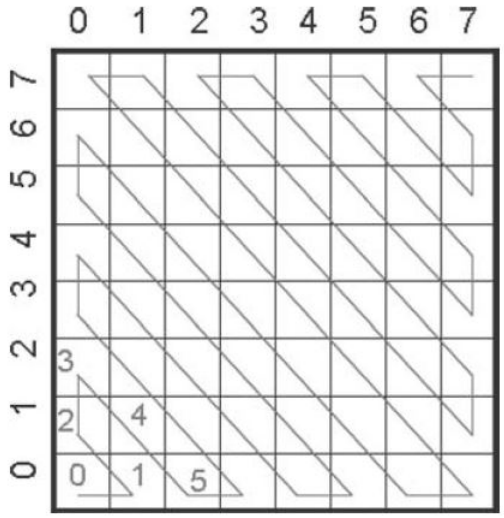



Figure 2.11: Contor-Diagonal-Order Traversal of 2D Space

To overcome this drawback, Contor-diagonal-order traversal as seen in Figure 2.11 approach, has been followed, which diagonally traverses the space, reducing the issue faced in column-order technique drastically but not completely.

## 2.8 Defining Q-learning for the Grid

With the set up of the grid, Q-learning problems can be defined. An agent can move around the grid by taking available actions and maximize the reward while switching the states. At first, let us specify the environment of the MDP. Gridworld of $n^2$ biclusters/states are considered, the environment is discussed in more detail below[17]:

### 2.8.1 State Space

Gridworld has $n \times n = n^2$ distinct states. Each state s = (U,I) is a bicluster consisting of some users U and some items I. The agent can choose any state as a start state and can move within the defined gridworld. Since some groups of the biclusters may not have any similarity with other biclusters, in such cases choosing only 1 start state may result in not exploring biclusters in such groups (Exploration coefficient becomes important here); hence Q-values can be updated again with all the state as start states one by one in iterations.

### 2.8.2 Action Space

Action space for the gridworld contains four actions to move around: up, down, left, right. It can only take one step in one action. Self-destructive actions from the extreme ends of the grid considered as dead state and particular iteration end their agent receives some negative reward.

### 2.8.3 Transition Function

Transition function $T(s_t, a_t)$ decides what would be the next state upon taking an action $a_t$ from the state $s_t$. In the deterministic gridworld, with agent moves in a state to the desired state with a 100% probability. For Example: In deterministic world if agent at state $< 1, 2 >$ takes action "left" the transition function $T(< 1, 2 >, ' left')$ would take the agent to $< 1, 1 >$. In this implementation, the transition function is considered to be not completely deterministic; in fact, it follows $\epsilon$-greedy policy while learning. $\epsilon$ greedy policy is a way of selecting random actions with uniform distribution from a set of available actions. Using this policy either random action with epsilon probability can be selected or action with the $1 - \epsilon$ probability that gives a maximum reward in a given state. For example, if an experiment is about to run ten times. This policy selects random actions in thrice if the value of epsilon is 0.3.

### 2.8.4 Reward Function

Reward function denoted by $R(s_t, a_t, s_{t+1})$ which depends on current state,action and new state reached. Here Jaccard index as the reward function has been proposed. In simple terms, it means how many users overlap between two states. More overlap gives a better reward. Since collaborative filtering tries to find similar users to suggest their preferred items to users, using a very similar concept, Jaccard index gives higher

reward for high overlap. Jaccard index can be defined as follows:

$$R(s_t, a_t, s_{t+1}) = JaccardIndex(U_{s_t}, U_{s_{t+1}}) \tag{2.8}$$

$$= \frac{\mid U_{s_t} \cap U_{s_{t+1}} \mid}{\mid U_{s_t} \cup U_{s_{t+1}} \mid} \tag{2.9}$$

The Jaccard index caps the reward between 0 to 1. When there are no common users between the two states, the reward becomes 0 and 1 when all the users are the same in both the states. The similarity between the items of the two states can also be considered, but that would put restrictions on the reward and very few items may get recommended to the user. Hence the reward function is made applied only based on the similarity of the users.

### 2.8.5 Goal

Q-learning usually has a goal state, which also a terminating state, where the particular iteration ends and the agent receives a maximum reward. In this case, agent getting off the grid or not getting any new items to recommend are the only terminating conditions, and the agent tries to maximize its reward before the iteration terminates.

## 2.9 Learning the Q-function

The output of the Q-learning algorithm is Q-table which has Q values for every action for every state.

Initially, the table is filled with arbitrary values for all the states (Qs), then, at


Figure 2.12: Example of Q-Learning in Gridworld

each time $t$ the agent selects an action $a_t$, observes a reward $r_t$, enters a new state $s_{t+1}$ generated using the reward function, and $Q$ is updated for the particular state. The core of the algorithm is a Bellman equation as seen in figure 2.13[14], a simple value iteration update, using the weighted average of the old value and the new information[14]:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}^{\text{old value}} \Big)}^{\text{temporal difference}}$$
$$\text{new value (temporal difference target)}$$

Figure 2.13: Bellman's Equation for Q-Learning

The Q value for a particular state and for particular action depends on the reward anticipated by the immediate next state, which is represented by the maximum Q

value of the next state which also maximizes the Q value of current state-action pair. Figure 2.12 shows an example how Q-values are populated for all actions in Gridworld

**Algorithm 1:** Learning Q function

**Input :** State-space S, Action-Space A, Transition function T, Agent

**Output :** A policy $\pi$

Select a random start state $s_1$;

**for** $i \leftarrow 1$ *to trials* **do**

  **while** *step* $<$ *max steps per episode or game over* **do**

    $a \leftarrow \epsilon$-greedy with action

    execute action a

    $s^{'} \leftarrow T(s, a)$

    $Q_{new}$=(1-$\alpha$)×$Q_{old}$ + $\alpha \times (r + \gamma \times$ maximum Q value in $s^{'}$)

    **if** *No new movie found or Agent moves off the grid* **then**
      game over = True;

    **else**
      s ←$s^{'}$

    **end**

  **end**

**end**

The learning algorithm takes a number of trials, maximum steps per episode as the input parameters. The implemented algorithm considers a random start state $s_t$ on the grid and keeps the note of the movies in that particular bicluster. The agent selects an action with $\epsilon$-greedy approach and shift to the new state $s_{t+1}$ and keeps note

of newly found movies which were not presented until the previous state; meanwhile it also receives the reward using the reward function $r_t$, following this it updates the $Q(s_t, a_t)$ value for the previous state for the taken action. It continues this process until it reaches the final state i.e., it either goes off the grid or does not discover a new movie in the new state. The algorithm also keeps track of the cumulative rewards obtained in every episode. We have already discussed the role of learning rate $\alpha$ and discount rate $\gamma$ in the previous chapter

## 2.10 Generating Recommendations

Once the Q table is learned, A policy is formed, which states the best action to take every state, which can result in the ultimate maximum reward. In this environment, any state can be the start state since the Q table has been learned in that way. To generate a recommendation for a particular user identifying the start state is vital as recommended movies are based on it too. To decide the start state, it is essential to find the closest bicluster of the user. There are multiple ways to find that. Calculating the jaccard index between the movies watched by the user and the set of movies in the bicluster could be one. This approach may not consider the ratings for movies and there may exist many biclusters sharing the same maximum value of jaccard index. The current study uses cosine similarity to calculate the similarity between the user and the biclusters. It uses the values of the features and also considers the fact that all users do not rate the movies in the same fashion, the details have been discussed in the next subsection,

**Algorithm 2:** Generating recommendations

**Input :**A policy $\pi$, User

**Output :** Recommended items

**if** *Ratings available for user* **then**
| Identify the start state s, using cosine similarity of rating vector

**else**
| Identify the start state s, using cosine similarity of demographic details

**end**

**while** *at least one item to recommend* **do**
| recommend items in s;
| $a \leftarrow \pi(s)$
| execute action a
| s $\leftarrow s^{'}$

**end**

### 2.10.1 Identifying the Start State

As discussed the start state for a particular user would be the most similar bicluster of the grid, the following steps are followed to calculate the most similar bicluster using cosine similarity as seen in Figure 2.14:

- Creating the vector containing avg. ratings of movies, i.e. (mean of ratings given by all the users for the particular movie) included in the particular bicluster

- Similarly creating a vector of the ratings of corresponding movies rated by the user

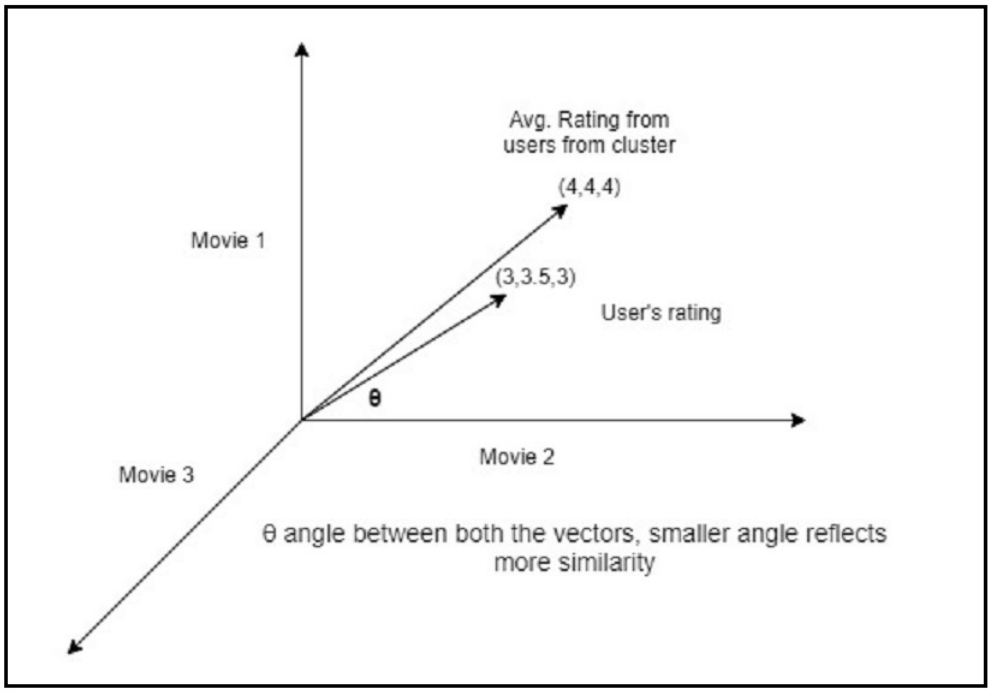



Figure 2.14: Implementation of Cosine Similarity

- Calculating the cosine similarity between both the vectors, the similarity gives the angle between both the vectors, and it does not consider the magnitude.

- Bicluster returning the maximum similarity with the user will be considered as the start state for that particular user.

The above approach is feasible when a user has already rated some items. In the case of a new user, no ratings will be available for the users, and it would be difficult to identify the start state. This case of the cold-start problem can be addressed by demographic information of the user.

## 2.11 Addressing the Cold-Start Problem

Demographic information of the user contains age,gender,location,occupation,educational qualification,marital status. To establish a similarity between a user and a group of users, it is important to quantify this information so that it can be used as a metric. As seen in the previous chapter, the data used for this implementation consists of age, gender, occupation, zip code details of the user and how it can be quantified to be considered as a similarity feature.

In the last section, an effective vector for bicluster was used to find the similarity with the user; it was feasible as all the features were movies and the values were between a fixed range. In this case, all the features are not of the same category and they have different ranges; hence a different approach has to be followed:

- Calculate the cosine similarity score between the considered user and each user within the bicluster.
- Average of all similarity scores can be considered as the effective similarity score between the user and the bicluster
- Bicluster with maximum score can be selected as the start state for a user with no ratings.

Since demographic information is inherently associated with the person and collected using the initial registration of all the platforms where recommender systems are used, This can be a good way to solve the cold-start problem.

After selecting the start state, the learned policy will be followed as the agent

keeps visiting new states non-rated movies from the visited states will be added as recommended items for the particular user.

## 2.12 Evaluation Process

Evaluation of the process should ideally depend on how satisfied the user is with the recommendations provided by the system.

This particular implementation has been evaluated against the state of the art system. In this study Singular Value Decomposition (SVD) [5], which internally uses matrix factorization. Recommendations are generated for the same data using the state of the art system and compared how the implemented system performs in comparison to state of the art. The setup of the experiment and the results have been discussed in the next chapter.

Chapter 3

# RESULTS

## 3.1 Performance of Implemented Algorithm

In an experiment where the implemented algorithm is trained for 100 movies and 100 users. Selected users and movies arranged in biclusters using 8×8 grid. Q-learning algorithm was executed for 150 trials.

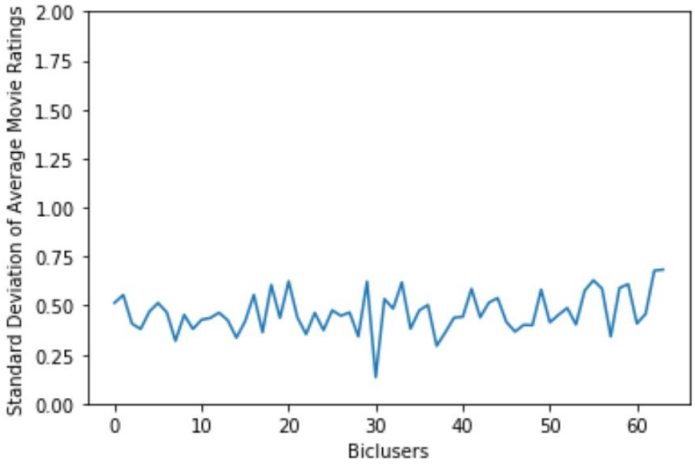



Figure 3.1: Standard Deviations of Average Movie Ratings for Biclusters

Figure 3.1 shows the standard deviation of average ratings for movies in each bicluster; it can be observed that the standard deviation was low with the mean value 0.46, which suggests that a maximum number of ratings are concentrated towards a particular value.

Figure 3.2 shows the length of each episode; it can be observed that most of the episodes run for about seven steps. i.e. means it explored seven biclusters. It was observed that Q-table contained updated values of 96% actions.

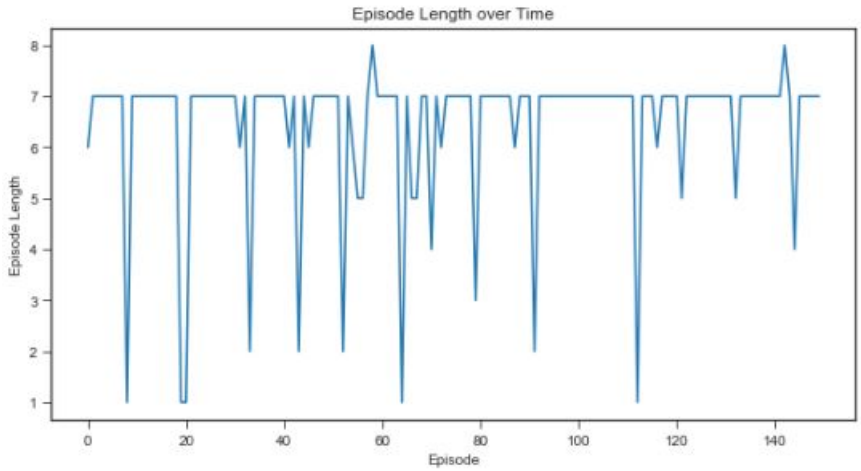



Figure 3.2: Length of Each Episode While Learning

Also, from Figure 3.3, we can understand the learning curve of the algorithm, which is plotted using the rewards received in each iteration by the algorithm. It shows an increasing learning curve in the early stages and flattens after a while, depicting that the policy has been learned.

## 3.2 State of the Art System

A state of the art system is refers to the highest level of general development, as of a device, technique, or scientific field achieved at a particular time. However, in some contexts it can also refer to a level of development reached at any particular time as a result of the common methodologies employed at the time.[24] To evaluate the

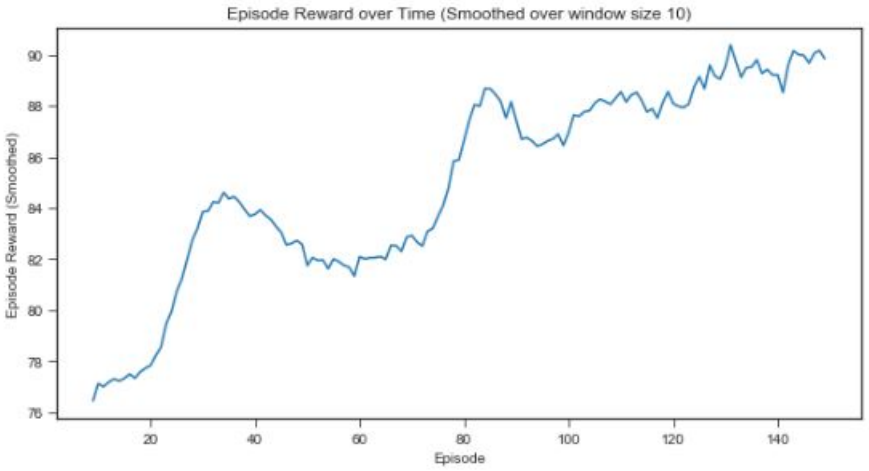



Figure 3.3: Cumulative Award Received While Learning

implemented system we are comparing its output against the state of the art system which are widely used in the real world applications. Here, underlying algorithms of widely used recommender systems have been used as the state of the art system. In the first experimental setup the output of the implemented against Singular Value Decomposition (SVD) Algorithm.

The state of the art system used to compare the outputs is collaborative filtering implemented using Singular Value Decomposition (SVD), which internally uses matrix factorization techniques. Matrix factorization techniques are one of the trusted methods in the field of collaborative filtering. It was also used by the algorithm which bested the Netflix Prize competition where the challenge was to come up with the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films [25].

### 3.2.1 Matrix Factorization

Matrix factorization involves a concept of reducing the matrix into its constituent parts.[26]. With this concept, the rating matrix, which includes users and movies with ratings for certain movies can be reduced to 2 separate matrices i.e., User matrix and Movie matrix. See Figure 3.4[27]. The individual matrices can then be used to predict the rating for a particular user—movie combination. Singular Value Decomposition (SVD) is used for matrix factorization in the considered state of the art system.

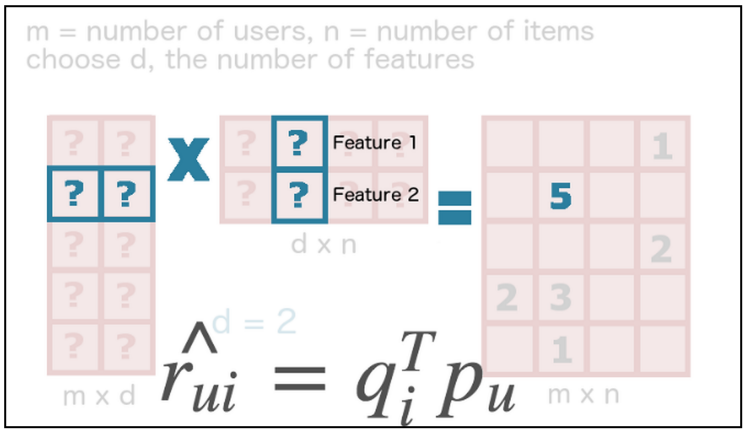



Figure 3.4: Matrix Factorization

## 3.3 Experimental Setup and Results

The implemented algorithm is being evaluated for a couple of cases. In first, recommendations are generated for a user who has rated some movies already. Second setup is for the cold-start case where a user without any ratings is considered, recommendations for that user are evaluated.

### 3.3.1 Comparing Against State of the Art System

The experimental setup to evaluate the proposed algorithm compares the output of the proposed algorithm and the state of the art systems with the same set of users and movies. As the experimental approach returns, rating for the unrated movies for the user comparison is being made in the top newly rated movies by the algorithm with the movies suggested by the proposed algorithm. As seen in Table 3.1, Results for a couple of users have been mentioned. The algorithm generated 12 recommendations for User1, and 41% of those movies were among the top 30 movies suggested by the state of the art system. 58% of those movies were above the average concerning the ratings.

| **Test User #** | **No. of Movies Recommended by Implemented Algorithm** | **% Overlap with Top 30 Movies Rated by State of the Art** | **% Movies Above Average Rating** |
|---|---|---|---|
| **User1** | 12 | 41% | 58% |
| **User2** | 15 | 46% | 67% |

Table 3.1: Comparison with State of The Art System

### 3.3.2 Performance for Cold-Start Problem

This experimental setup checks how the proposed algorithm performs when there is no relevant data available for a particular user; this has been tested by introducing a user from the test set i.e., not considering any of his ratings but only the demographic features. Furthermore, comparing the results given by the algorithm with the actual top rated movies by the user.

$M_P$ represents movies recommended by the implemented algorithm and $M_T$ represent the movies algorithm should have represented The precision is calculated as

$$Precision = \frac{|M_P \cap M_T|}{M_P} \tag{3.1}$$

$$Recall = \frac{|M_P \cap M_T|}{M_T} \tag{3.2}$$

100 test users were tested for the cold-start problem; following are the results which show the average precision value and recall value

| **No. of Users** | **Average Precision** | **Average Recall** |
|---|---|---|
| 100 | 0.35 | 0.29 |

Table 3.2: Results for Cold-Start Condition for 100 Users

Here are some examples of how the implemented algorithm performs for cold start cases, As seen in Table 3.2 for the User1 was introduced to the algorithm as a user without any rating, and his demographic details were considered, and the algorithm recommended 28 movies for that user.

| **Test User #** | **# Movies Recommended by the Algorithm** | **% Overlap with Top 30 Movies Rated by the User** |
|---|---|---|
| **User1** | 28 | 39% |
| **User2** | 24 | 46% |

Table 3.3: Results for Cold-Start Cases

These movies were compared with the actual ratings given by the user and 39%, movies overlapped with the top 30 movies rated by that user. For a similar scenario, User2 has shown a 46% overlap.

### 3.3.3 Comparison Against Algorithms with Metrics

After analyzing the dataset, a long-tail plot was observed. In statistics and business, a long tail of some distributions of numbers is the portion of the distribution having many occurrences far from the "head" or central part of the distribution.[28] It has very few occurrences of very large events, and very many occurrences of minimal events, which gives the graph a "long tail. In this experimental setup to avoid the sparsity, only the users with at least 50 ratings were selected. The long plot for the dataset can be seen in Figure 3.5 Usually, there are popular movies, and it is not tricky

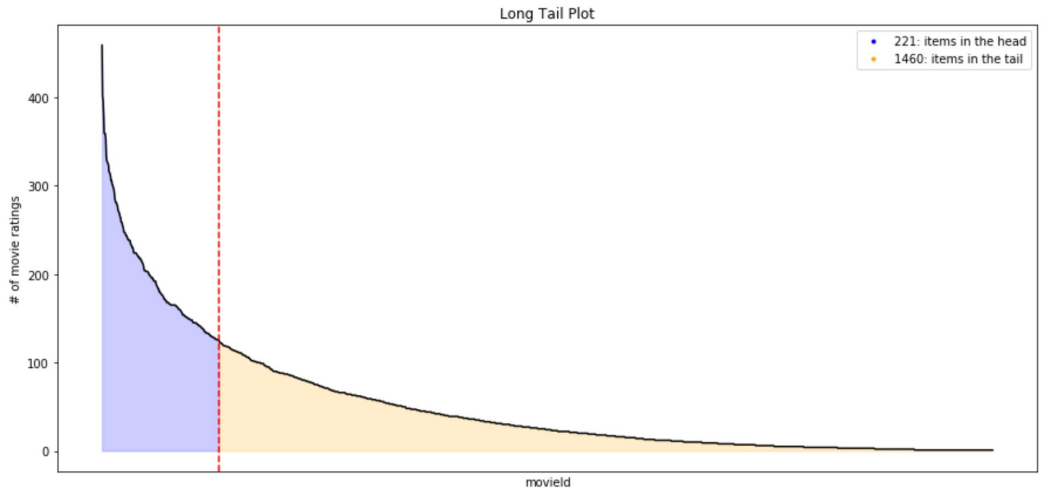



Figure 3.5: Long Tail Observed for Current Database

for recommender systems to learn to suggest those movies. Relevant recommendations are defined as recommendations of items that the user has rated positively in the test data. The metrics identified here provide methods for evaluating both the relevancy and usefulness of recommendations. There are several algorithms which are used for comparison; these are some of the most used algorithms for recommender systems

available in built-in packages for recommender systems. Here are the setups used to compare the results.

1. Considering only 100 movies and 100 users from the dataset where all users have rated at least 50 movies, such that the matrix is densely populated with ratings

2. Train and test dataset were split n in 67-33 ratio

3. Square grid with side eight and having a minimum four movies and four users in each bicluster.

The following are the metrics using which the various recommender system algorithms are compared.

**Root Mean Squared Error (RMSE)**

RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It is the square root of the average of squared differences between prediction and actual observation.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2} \tag{3.3}$$

**Mean Absolute Error (MAE)**

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It is the average over the test sample of the absolute differences between prediction and actual observation where all individual differences

have equal weight.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \tag{3.4}$$

**Latency**

The algorithms are compared by the time taken to generated the suggestions

**Coverage**

Coverage is the percent of items in the training data the model can recommend on a test set. For example, random recommendations will have 100% coverage; on the other hand, fixed recommendations will have low coverage.

**Personalization**

Personalization is a great way to assess if a model recommends many of the same items to different users. It is the dissimilarity (1- cosine similarity) between the user's lists of recommendations.

**Intra-List Similarity**

The intra-list similarity is the average cosine similarity of all items in a list of recommendations. This calculation uses features of the recommended items (such as movie genre) to calculate the similarity.

To generalize the model 3-fold cross-validation technique is used for all the algorithms and average RMSE, MAE scores are calculated, and time to fit the model and test the data has been logged. Since the implemented model does not predict

the rating, the RMSE score has not been calculated for it. From Table 3.4, it can be understood that SVD shows the relatively best performance. Lower the value of RMSE indicates a better fit. From Table 3.6, it can be seen that the other algorithms

| **Algortihm** | **fit_time** | **test_mae** | **test_rmse** | **test_time** |
|---|---|---|---|---|
| **SVD** | 0.18258 | 1.050492 | 1.449563 | 0.010981 |
| **KNNWithMeans** | 0.005332 | 1.026321 | 1.449685 | 0.107712 |
| **KNNWithZScore** | 0.008644 | 1.006323 | 1.452916 | 0.118542 |
| **KNNBaseline** | 0.005319 | 1.035848 | 1.455233 | 0.152299 |
| **SVDpp** | 2.522272 | 1.010394 | 1.462099 | 0.103466 |
| **SlopeOne** | 0.00659 | 1.043908 | 1.466868 | 0.065182 |
| **BaselineOnly** | 0.003324 | 1.118044 | 1.482259 | 0.00964 |
| **CoClustering** | 0.111029 | 1.036734 | 1.511059 | 0.010306 |
| **KNNBasic** | 0.003314 | 1.100995 | 1.552553 | 0.08843 |
| **NormalPredictor** | 0.003996 | 1.67664 | 2.234489 | 0.010285 |

Table 3.4: Results of Cross Validation for 100 Movies and 100 Users

are efficient for time, and they get executed within a few seconds.

Table 3.5 shows the time taken by the implemented algorithm to recommend movies to all users in the test data. Since the fewer data was considered for this particular experimental setup, the biclusters were identified in 2.48seconds. Q-learning algorithm took 3.90 secs to learn the policy, and then to find recommendations to all the users took 1.22 secs. Though it is slower than other algorithms, it is quick enough for a small size of data considering the processes it goes through.

| Step | Time Taken |
|---|---|
| Biclustering | 2.48 secs |
| Fit | 3.90 secs |
| Test | 1.22 secs |
| **Total** | **7.61 secs** |

Table 3.5: Time Taken by Implemented Algorithm

| Algorithm | RMSE | Time |
|---|---|---|
| **SVD** | 1.4615 | 0.2962 |
| **SVDpp** | 1.4407 | 3.3469 |
| **SlopeOne** | 1.4765 | 0.0588 |
| **NormalPredicor** | 2.2141 | 0.0139 |
| **KNNBaseline** | 1.4639 | 0.1261 |
| **KNNBasic** | 1.5742 | 0.0757 |
| **KNNwith Means** | 1.4541 | 0.0877 |
| **KNNwithZScore** | 1.4556 | 0.0977 |
| **BaselineOnly** | 1.4848 | 0.0099 |
| **CoClusering** | 1.499 | 0.0857 |

Table 3.6: Predicting on Testset

The next Table 3.7, compares the coverage of each algorithm. Coverage states the extent to which the algorithm recommends a variety of movies. Recommending the same set of movies to all users will never recommend other movies from the dataset, thus resulting in low coverage.

It can be seen that the coverage of the implemented algorithm is comparatively low, which means it recommends a limited set of movies to most of the users.

The next metric is personalization score 3.8; a high personalization score indicates the user's recommendations are different, meaning the model is offering a personalized experience to each user. The implemented algorithms have a low score compared to other algorithms. As the algorithm suggests movies by traveling through biclusters,

| **Algorithm** | **Coverage** |
|---|---|
| BaselineOnly | 2.8 |
| COClustering | 3.09 |
| KNNBaseline | 2.97 |
| KNNBasic | 2.97 |
| KNNWithMeans | 2.8 |
| KNNWithZScore | 2.8 |
| NormalPredictor | 5.89 |
| SVD | 3.87 |
| SVDpp | 4.16 |
| SlopeOne | 2.86 |
| Actual | 5.95 |
| **Algorithm** | **2.02** |

Table 3.7: Coverage Score Comparison

| **Algorithm** | **Personalization** |
|---|---|
| BaselineOnly | 0.713927291 |
| COClustering | 0.746185356 |
| KNNBaseline | 0.717204301 |
| KNNBasic | 0.716487455 |
| KNNWithMeans | 0.718637993 |
| KNNWithZScore | 0.718842806 |
| NormalPredictor | 0.902201741 |
| SVD | 0.756118792 |
| SVDpp | 0.782642089 |
| SlopeOne | 0.714900154 |
| **Algorithm** | **0.313824885** |

Table 3.8: Personalization Score Comparison

it collects similar movies, hence it may lack the high personalization score.

Lastly, as seen in table 3.9, This calculation uses features of the recommended items (such as movie genre) to calculate the similarity, in this particular setup, featured vector considers only three genres i.e., Action, Comedy, Romance. If a recommender system is recommending lists of very similar items to single users, then the intra-list similarity will be high.

| **Algorithm** | **Intra-List Similarity** |
|---|---|
| BaselineOnly | 0.068077601 |
| COClustering | 0.08324515 |
| KNNBaseline | 0.068077601 |
| KNNBasic | 0.067372134 |
| KNNWithMeans | 0.067724868 |
| KNNWithZScore | 0.06984127 |
| NormalPredictor | 0.090299824 |
| SVD | 0.076190476 |
| SVDpp | 0.076895944 |
| SlopeOne | 0.06984127 |
| **Implemented Algorithm** | **0.02680776** |

Table 3.9: Intra-List Similarity Comparison

Chapter 4

# DISCUSSION AND CONCLUSION

Considering the existing recommender systems and growing market, the need for having more context-aware system is increasing, Chapters 3, 4 have described the system developed to generate recommendations and how it performs in various cases. In this chapter, implications, limitations, and future work for the implemented algorithm has been discussed.

## 4.1 Discussion

The designed system uses several vital components such as biclusters, Q-learning, similarity metrics. It was designed to address the cold-start problem and generate explainable recommendations. Biclustering the users and the movies in the early stage binds the similar items together since the generated recommendations come through traversed biclusters, each one can be explained. Another critical aspect of the implemented algorithm to identify the start state it uses cosine similarity as a measure that does not get affected by the magnitude. As the system was built by considering two cases, the testing strategy was built accordingly. Generally, recommender systems predict the missing ratings of the movies, which enable them to evaluate the test data. In the current study, the algorithm recommends movies and does not predict the

rating; hence comparing it against the state of the art system, which is widely used, was a viable option. It can not be assumed that state of the art gives the best results, having a right amount of similarity braces the approach. For the cold-start problem, it is difficult to evaluate the output if the user does not have any rating, Users with ratings were used for testing by hiding their ratings, and the results were compared against the actual rating. It was observed that most of the recommended movies were among the top-rated by the user, which suggests that demographic information is an excellent measure to solve the cold-start problem.

Concerning the issues discussed in Chapter 1, the Implemented algorithm addresses the cold-start problem, It also addresses the critical issue of grey sheep problem where the user is not matched with any other group of users and the system fails to generate useful recommendations. The strategy discussed in this study to find start states i.e., closest biclusters can solve this problem. This approach may not solve other problems of changing data preferences, but the online implementation of this algorithm lt, which considers feedback of the users, can be built to solve this problem; it is discussed in detail in the future work section.

During the implementation of the algorithm, key things were observed. Identifying biclusters is a complex task, and it is done using a heuristic; thus, performance depends heavily on how quickly and accurately, best biclusters can be identified. Arranging the biclusters in the grid is also done heuristically using the Contor-diagonal-order traversal, The performance of q-learning depends on how well the biclusters are arranged in the grid, a better filling algorithm may place the biclusters in such a way that every cluster will have its most similar biclusters as neighbors. The closeness

between the biclusters is identified using the fitness value, which approximates the closeness and performance may depend on that, a better metric may result in finding that closeness more accurately but may also hamper the complexity and performance.

Biclustering has been in use for a while in the field of bioinformatics [29]. Recently it has been used in the field of recommender systems. Biclustering clusters users and items together and tries to keep the variance low, and it binds similar items together in early stages, and thus in recent works, recommender systems with biclustering approach have shown excellent results.A research paper [17] uses a reinforcement learning approach along with biclustering for a recommender system. Another research paper [21] uses the biclustering approach to implement a hybrid recommender system. The approach implemented in this study is unique in the way it filters the biclusters from all the created biclusters and arranges them in the grid. Earlier work claims to address the cold-start problem just with the help of biclustering as similar movies are clustered together, but it randomly chooses the start state. The current study explicitly addresses the problem of cold-start by using demographic features and finds the appropriate start state for the cold start case. The results for cold-start cases support the claim.

## 4.2 Implications

Our main of this study is to propose a recommendation strategy which works well by using reinforcement learning and addresses the cold-start problem, with the observed results the proposed algorithm performs well in comparison with the current state

of the art systems and also stands up to the mark for the accuracy. In the other experiment, it was observed that it was able to suggest relevant movies to the user who did not have any ratings in the past, and it implies that the proposed algorithm works well in case of the cold start situations. The implemented algorithm funnels movies with the use of the biclustering approach at the beginning, which binds similar items together at the very early stage of the algorithm have eventually made the system more robust. With the observed results, it can be implied that the mentioned strategy works very well for this type of problem.

The results depicting low standard deviation of average movie ratings for biclusters and the learning curve of the q-learning algorithm confirm that biclusters are indeed clustering similar movies together and the Q-learning algorithm tries to maximize the reward over a period and then flattens when the learning has been optimized, it implies that the algorithm is indeed learning based on the reward model used.

The evaluation metric such as coverage, personalization score and intra-list similarity show comparatively low values and it takes more time to get executed than the widely used algorithms too. These are the important facts to know and more research could be done on this in future to improve the overall coverage and intra-list similarity.

Overall based on the results and comparisons with the state of the art system, the implemented algorithm suitably explains the recommendations suggested for the user, At the same time the implemented algorithm also has some drawback which can be improved, and the algorithm can be made more accurate and acceptable for the real-world applications. The topic needs further research and improvement, the

limitations of the algorithm and future steps are discussed in the next section.

## 4.3 Limitations

Though the results of the proposed systems are satisfactory, it has certain limitations that have to be addressed and research can be done on those topics in the future. Traditional recommender system uses collaborative filtering, which internally uses matrix factorization techniques; these techniques are efficient and faster to execute. Though the biclustering technique used in the early stage of the algorithm binds similar items and users together, which helps to explain the recommendations suggested for the user, there is the downside of these techniques; these are heavy computation techniques and takes time to come up with the biclusters.

The other limitation of the algorithm is that it uses heuristics in many places such as to identify the biclusters and even to arrange them in the grid, the performance heavily depends on the accuracy of the heuristic. The one used for arranging biclusters in the grid fails to arrange all biclusters in such a way that all neighboring biclusters share high levels of similarity. Along with one of the diagonals of the grid, the neighboring biclusters may not share a good amount of similarity.

The biclustering algorithm considered in this study works on binary data; hence the unavailable ratings are replaced with 0. It loses the information if any of the users have rated a movie 0.

## 4.4 Future Directions

It is clear by now how important and effective recommender systems are for businesses nowadays and there is a huge scope of improvement in the proposed algorithm as well. Currently, the proposed model is being trained on a fixed amount of movies and users and there is no way to include new items. Online implementation of the algorithm where whenever a set of movies from a bicluster is recommended to user feedback can be taken from the user about how useful these recommendations are. If the feedback is positive, the user can then be added to the user set of that particular bicluster. Similarly, an implementation of the proposed algorithm which will view the system from the movie's point of view where a set of users are identified for a particular movie, and based on the recommended users' feedback, it can be decided whether to add the movie in the bicluster or not. Cold start problems in the case of movies can be solved with the help of the metadata of the movie, such as release date, genre, lead actors, time duration.

As discussed earlier, to enhance the performance and accuracy of the algorithm as a whole, the grid could be implemented as multi-dimensional with more than four actions covering the corners too. This would give the algorithm more flexibility to learn the better policy and generate more suggestions, this would be heavy computation tasks, but that trade-off can be topic to be researched in the future.

Currently, the grid is arranged by using Contor-diagonal-order traversal which has some limitations as discussed in the previous section. Calculating exact similarity and arranging them to ensure that similar items remain closer in the grid could be NP-

Hard problem, In future, a better heuristic can be identified which will be efficient as well.

The implemented algorithm uses the jaccard index as the reward function, which favors a set of movies only based on how many users in the biclusters overlap. It does not specifically consider the rating while rewarding since ratings are handled while creating the biclusters. Along with overlap, it can be researched in the future to understand which reward model works better in this kind of setup. Additionally, to make the reward function more context-aware reward function considers demographic information of the users in the biclusters as well as the details of the movies too.

## REFERENCES


[1] P. Resnick and H. R. Varian, *Recommender Systesm*. Communications of the ACM, 1997.

[2] I. MacKenzie, C. Meyer, and S. Noble, *How retailers can keep up with consumers*. McKinsey and Company.

[3] C. A. Gomez-Uribe and N. Hunt, *The Netflix Recommender System: Algorithms, Business Value, and Innovation*. Netflix.Inc, 2015.

[4] Blueshift, *Evolution of Recommender Systems*. Medium.com, 2017.

[5] C. Aggarwal, *Recommender Systesm*. Springer, 2016.

[6] R. Burke, *Hybrid Recommender Systems: Survey and Experiments*. California State University, Fullerton.

[7] F. Ricci, L. Rokach, and B. Shapira, *Introduction to Recommender Systems Handbook*. Springer US, 2011.

[8] A. Gershman, A. Meisels, K.-H. Luke, L. Rokach, A. Schclar, and A. Sturm, *A Decision Tree Based Recommender System*.

[9] S. Khusro, Z. Ali, and I. Ullah, *Recommender Systems: Issues, Challenges, and Research Opportunities*.

[10] R. Macmanus, *5 Problems of Recommender Systems*. https://readwrite.com/, 2009.

[11] X. Xie, J. Lian, Z. Liu, X. Wang, H. W. Fangzhao Wu, and Z. Chen, *Personalized Recommendation Systems: Five Hot Research Topics You Must Know*. Microsoft Research Lab-Asia, 2018.

[12] *Reinforcement learning*. Wikipedia: The Free Encyclopedia.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press.

[14] *Q-Learning*. Wikipedia: The Free Encyclopedia.

[15] K. Eren, *Introduction to Biclustering*. 2013.

[16] K. Eren, *Application of Biclustering Algorithms to Biological data*. Ohio State University, 2012.

[17] S. Choi, H. Ha, U. Hwang, C. Kim, J.-W. Ha, and S. Yoon, *Reinforcement Learning based Recommender System using Biclustering Technique*. IFUP, 2018.

[18] A. Liu and J. Callvik, *Using Demographic Information to Reduce the New User Problem in Recommender Systems*. KTH Royal Institute of Technology, 2017.

[19] A. Prelic, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, , and E. Zitzler, *A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data*. Bioinformatics, 2006.

[20] K. Eren, *The Bimax Algorithm*. 2013.

[21] P. A. D. de Castro, F. O. de França, H. M. Ferreira, and F. J. V. Zuben, *Applying Biclustering to Perform Collaborative Filtering*. University of Campinas.

[22] J. A. Gomez-Pulido, J. L. Cerrada-Barrios, S. Trinidad-Amado, J. M. Lanza-Gutierrez, R. A. Fernandez-Diaz, B. Crawford, and R. Soto, *Fine-grained parallelization of fitness functions in bioinformatics optimization problems: gene selection for cancer classification and biclustering of gene expression data*. BMC Informatics, 2016.

[23] K. S. Candan and M. L. Sapino, *Data Management for Multimedia Retrieval*. Cambridge University Press, 2010.

[24] *State of the Art*. Wikipedia: The Free Encyclopedia.

[25] *Netflix Prize*. Wikipedia: The Free Encyclopedia.

[26] *A Gentle Introduction to Matrix Factorization for Machine Learning*. Machine Learning Mastery.

[27] S. Chatterjee, *Overview of Matrix Factorisation Techniques using Python*. towardsdatascience.com.

[28] *Long Tail*. Wikipedia: The Free Encyclopedia.

[29] Y. Cheng and G. M. Church, *Biclustering of Expression Data*. 2000.

# APPENDIX A

# EXTRACTING SALARY INFORMATION FROM OCCUPATION

Approximate salaries from http://www.payscale.com/

| Occupation | Average Annual Salary |
|---|---|
| Administrator | $45,000 |
| Artist | $44,000 |
| Doctor | $169,000 |
| Educator | $64,000 |
| Engineer | $71,000 |
| Entertainment | $58,000 |
| Executive | $75,000 |
| Healthcare | $65,000 |
| Homemaker | $19,000 |
| Lawyer | $81,000 |
| Librarian | $49,000 |
| Marketing | $62,000 |
| None/Other | $0 |
| Programmer | $61,000 |
| Retired | $16,000 |
| Salesman | $30,000 |
| Scientist | $77,000 |
| Student | $4,000 |
| Technician | $42,000 |
| Writer | $57,000 |