Video Game Recommender Systems

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Rutwik Borole

School of Computer Science

University of Galway

Supervisor(s)

Dr. Séamus Hill

In partial fulfilment of the requirements for the degree of

MSc in Computer Science (Data Analytics)

DECLARATION

I, Rutwik Borole hereby declare that this thesis, titled Video Game Recommender Systems, and the work presented in it are entirely my own except where explicitly stated otherwise in the text, and that this work has not been previously submitted, in part or whole, to any university or institution for any degree, diploma, or other qualification.

Rutwik Borole

August 2023

Acknowledgements

I would like to take this opportunity to sincerely thank my project supervisor, Dr. Séamus Hill, for his support and guidance over the past few months. Without him, this work would not have been possible. I would like to acknowledge my fellow graduate students for their assistance throughout the academic year. Throughout this semester, you helped me develop and gain new knowledge. I would also like to thank my family and friends for their unwavering support and encouragement throughout this master's programme.

Abstract

Recommender systems have become a norm in recent times due to the availability of a vast number of products and services available to a daily user. Recommendation systems are considered to be an essential item for almost all big conglomerates. This thesis explores the need for a better video game recommendation system using deep learning. By making use of machine learning and collaborative filtering techniques, this project explores the performance of two prominent recommendation models: Collaborative filtering using Singular Value Decomposition and Neural Collaborative Filtering. Through experimentation on Steam dataset of user-game interactions, the study examines the predictive accuracy and recommendation performance of these two models. Additionally, this thesis also dives into the top ten video game recommendation generated by the models and draw insights from these recommendations. The findings shed a light on the strengths and limitations of both models. By using evaluation metrics like Root Mean Squared Error, we found that the Neural Collaborative filtering model outperforms the Singular Value Decomposition Model, but the recommendations generated by both of these models proved to have their own unique strengths.

Contents

List of Figures

Chapter 1

Introduction

The work entailed in this project describes the task of building a recommender system based on Steam Client data to get video game recommendations for users by implementing a simple Neural Collaborative Filtering (NCF) approach. The NCF model is a state-of-the-art recommendation algorithm that combines the power of matrix factorization techniques with neural networks. It is designed to handle both explicit and implicit feedback data, making it suitable for recommendation tasks with various types of user-item interactions .

1.1 Motivation

The European gaming marketplace has undergone considerable growth in recent years, as of 2022 the market saw over 277 million users engaging in video games across various platforms. With the start of the pandemic and following lockdown measures, more people turned to gaming as a source of entertainment, leading to a surge in user counts and market expansion. It is projected that the user count will rise to 313 million users by the end of 2023. The availability of a vast library of games across various platforms, such as consoles, PCs, and handheld devices, has contributed to this expansion of the gaming industry. In 2022 the industry grew by almost 8% in earnings reaching a whopping estimated value of $44.6 billion. This was possible due to the presence of diversity in games. We can now see how over the years the industry has evolved in terms of providing countless users with various platforms, a substantial number of game categories, and genres.

Nowadays, recommender systems are common, and they serve the function of automatically satisfying the user's desire to receive appropriate content [1]. Almost every online platform like Amazon, Netflix, Steam, and Epic Games Store, provides their customers with a customised recommendation system based on their specific purchases and interactions, the goal of these systems is to increase customer retention, provide more product variety, and to increase company profits by suggesting more similar products the users. Given the rapid growth and diversification of the European gaming market and the increasing challenge for users to navigate the overwhelming number of games and genres available across different platforms. Players have the difficult problem of choosing games that match their preferences and interests as the variety of games and genres on various platforms keeps expanding. As the gaming industry continues to expand, it becomes vital to provide users with personalized recommendations that align with their preferences and interests. User data and preferences can be used by recommender systems to deliver personalized and relevant game suggestions, which will enhance the overall gaming experience for the players.

The following recommendation paradigms will be explored in this thesis: collaborative filtering using SVD and Neural Collaborative filtering. Collaborative filtering leverages user-item interactions to make recommendations [10]. Neural Collaborative Filtering (NCF) is an innovative approach to making recommendations. It uses deep neural networks to learn how a user interacts with an item and to model complex, non-linear trends in the data. In contrast to more traditional forms of collaborative filtering, NCF uses neural embeddings to directly represent users and items to a mutual latent space, thus enabling the capture of complex user-item interactions [11].

The thesis will evaluate the performance of these recommendation paradigms by measuring their root mean squared error and mean squared error in generating game recommendations. A baseline recommendation paradigm that is collaborative filtering will be used for comparison. The findings of this project may contribute to the development and improvement of more efficient and personalized recommendation systems in the gaming industry.

Furthermore, this thesis may give a way to explore the potential impact of these recommendation systems on users and the gaming community. By providing relevant and personalized recommendations, these systems can help users discover new games, navigate the vast library of options, and enhance their overall gaming experience. Game developers can also benefit from these systems by gaining insights into user preferences, improving user engagement, and promoting new releases.

1.2 Research Questions

The research questions that will be explored in this project are as follows:

1. How does the performance of the Neural Collaborative Filtering model compare to Collaborative Filtering using Singular Value Decomposition? (RQ1)
2. Do both models generate appropriate recommendations for a given user? (RQ2)

1.3 Structure of Thesis

This thesis is detailed into 6 chapters in total. In Chapter 1 we discuss the motivation, pose our research questions. In Chapter 2, We provide background information and a review of the relevant research in the field of recommender systems. Chapter 3 describes about the dataset used in this project with its preprocessing and data analysis. Chapter 4 describes the research methodology used for the two recommender system models. Chapter 5 shows us the experimentation conducted and the results observed on those two models and answer the above-posed research questions. Finally, Chapter 6 serves as the conclusion for the thesis.

Chapter 2

Background and Literature Review

This chapter presents a general idea of the background material and existing research on the topic of recommender systems.

2.1 Introduction to Recommender Systems

There are countless options and decisions to be made every day, from what to eat to what TV show or movie to stream to what stocks to invest in. Decisions can be especially challenging for everybody because of the ever-increasing volume of choices available online and as well as the huge volume of information available on the internet.

For a long time, may it be a minor or major decision, people have always depended on advice and recommendations. It is the human tendency to seek opinions on different choices that are available at a given time to a person. Thus gave a start to recommender systems, a recommender system simply gives personalized recommendations (suggestions) to users based on the user's likes and dislikes. As a result, recommendation systems can be broken down into two primary classifications: collaborative filtering systems and content-based filtering systems.

In the subsequent section, we will discuss each of these categories in more detail. These categories are included in systems that are based on similarity measures [2]; however, after seeing the recent developments in artificial intelligence, people have moved on to more complex methodologies, such as machine learning and deep learning to build recommender systems for different avenues.

2.2 History

Recommender systems have been around since the dawn of information retrieval and filtering. The concept of making recommendations or suggestions to users may be traced back to the early efforts of different researchers and scientists. Different recommendation systems have been introduced by researchers over many years. One of the very early examples of recommendation systems may be found in library science, where librarians collected and recommended books to users based on their interests and preferences. With the development of computers and information technology, this manual approach evolved into more automated processes. One of the earliest recommender systems can be traced back to Computer Librarian Grundy by Elaine Rich, this system worked by interviewing users about their tastes and then recommending books to them. Based on the information gathered, the system assigned the user to a stereotyped group using a crude method, recommending the same books to everyone in the same group [7].

In the last decade or so there has been a huge spike in e-commerce and online platforms, and it became clear that people needed more and more personalised suggestions. Big conglomerates like Amazon and Netflix saw how important it was to give their users personalised ideas to improve their browsing and buying experiences. This resulted in the development of automated recommendation systems that used methods called collaborative filtering. " The term collaborative filtering was coined by the creators of a recommender system called Tapestry " [8]. Collaborative filtering looks at how users interact with items and finds patterns between users to make suggestions. The success of these first recommender systems made it possible for the field to keep getting better.

In the year 2017, Anwar et al. came up with a recommender system, they believed it to be beneficial for gamers. " They proposed a collaborative filtering system which divides games and then evaluate them based on the preference of other similar players. In order to anticipate and recommend new games to a particular player, the algorithm takes into account both the individual ratings of other players and the ratings of the games a chosen player enjoys " [3]. This system offered two types of recommendations: generalised (all games that are of a chosen genre with high

ratings) and personalised (the user is requested to rate at minimum 3 games from a list of genres).

2.3 Recommender Systems

Building a recommendation system can be done in a number of different ways, and these methods are typically categorised based on the requirements of the application. The basic classifications of recommendation systems include collaborative filtering, content-based filtering, and hybrid recommendation systems, despite the fact that there are a variety of different kinds of recommendation systems. Let's have a look at each of these categories in the following section.

2.3.1 Collaborative Filtering Recommender System

Collaborative filtering is one of the most common algorithms implemented to create a recommendation system. The fundamental concept underlying collaborative filtering is that, in order to provide recommendations, it is necessary to draw on the experience and insight of as many users as possible. The algorithm discovers similarities and patterns among users based on their previous behaviour, such as ratings, purchases, or click-through data. The algorithm is able to make recommendations for products that have received positive feedback from users who have preferences or interests which are similar to the target user, but the target user has not seen or interacted with such items before. The use of collaborative filtering can be further broken down into two primary categories: user-based and item-based filtering. User-based collaborative filtering looks for users who are similar to one another in order to make recommendations, while item-based collaborative filtering seeks related products in order to make suggestions to users [9]. Collaborative filtering techniques have been shown to be beneficial in many domains, including online marketplace, recommendations for films and music, and social networking platforms. These types of domains are ones in which user preferences and behaviour play an important part in driving the recommendation process.

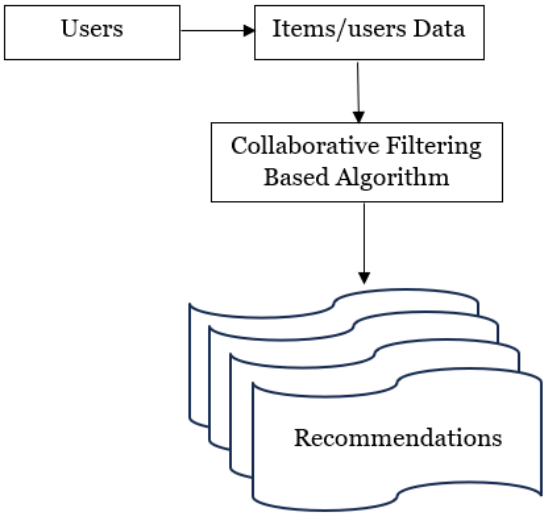

Figure 1: Collaborative Filtering Recommendation System

2.3.2 Content-Based Recommender System

A content-based filtering system is a sort of recommender system that utilizes the subject matter and user preferences to suggest items to users. In content-based filtering, the system examines the qualities or attributes of objects, such as text, photographs, or metadata, and generates a profile for each user based on their previous interactions or stated preferences. These profiles can be used to restrict the kind of content that the user sees. For the purpose of developing a user profile and making movie suggestions, a system that recommends films might, for instance, take into account characteristics such as the genre, director, actors, or storyline keywords. It would then propose films that share these characteristics. When user preferences may be successfully recorded through the analysis of the content of things, content-based filtering is very valuable. It is frequently used in fields like music, news, and article suggestions, all of which are examples of areas in which the content itself plays an important part in shaping the preferences of individual users. It is not necessary to collect users' explicit evaluations or preferences in order for content-based filtering to be able to generate personalised suggestions.

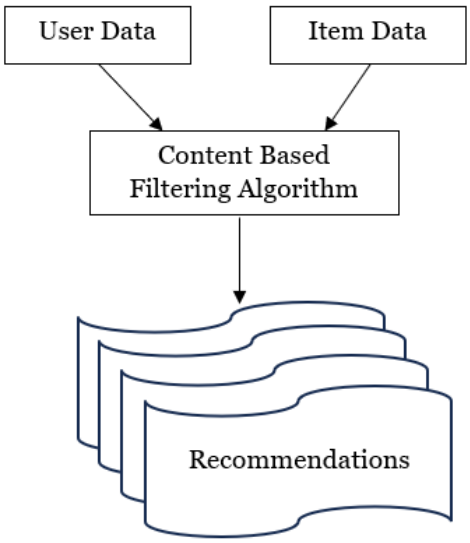

Figure 2: Content-Based Recommender System

2.3.3 Hybrid Recommender Systems

A hybrid recommender system is one that combines different recommendation algorithms, some of which are machine learning methods, to provide users with more precise and diverse recommendations. A hybrid system seeks to improve overall recommendation performance by overcoming the limits of individual recommendation techniques by focusing on the strengths that are offered by a variety of different methods. This is accomplished by leveraging the strengths that are offered by each particular method. Within the realm of machine learning, a hybrid recommender system can exploit user-item interactions, item attributes, and user preferences by including several methods such as collaborative filtering, content-based filtering, and matrix factorization, amongst others. Using past data, the system is able to discover patterns and associations, which then allows it to generate personalised recommendations. As a result, the hybrid system generates recommendations that are more thorough and efficient. In addition, techniques from machine learning can be utilised to dynamically change the recommendation models depending on the feedback from users and their ever-changing tastes. Because it is flexible, scalable, and has the potential to increase recommendation accuracy, the

hybrid method with machine learning is a promising technique in the field of recommender systems.

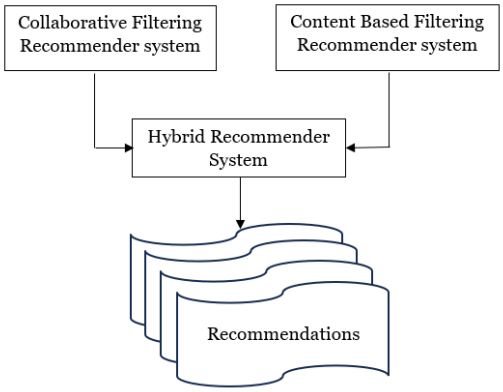

Figure 3: Hybrid Recommender System

2.4 Cold-Start Problem

The cold start problem is an important issue for recommendation systems to overcome, especially when working with new users or items with little to no prior feedback on them. It becomes difficult for a recommendation system to give out personalised recommendations when it is given with a “cold start” scenario for users, which occurs when there is insufficient data about the users' preferences and interactions with the products. There are two types of cold-start problems, the first being the User Cold-Start problem which occurs when very less or almost nothing is known about the user which makes it nearly impossible to make assumptions about them. And the other type is the Product Cold-Start problem, this issue occurs when there is so little information available about the product that it makes it difficult to recommend it.

Across the years many computer scientists have come up with various strategies to tackle the cold-start problem. Zi-Ke Zhang et al. suggested making use of social tags which are user-defined keywords which help in classifying the data into categories. “ It delivers more personalised recommendations when the supplied tags cover a wider range of topics.” [13]. Also, content-based filtering techniques are not

dependent on ratings that have been provided by other users, they can be utilised to provide suggestions for any and all things, provided that the qualities of the objects in question are accessible. Content-based predictions of other users who are similar to the active user can also be used to further improve recommendations for that user.[14]

2.5 Feedback

In the case of recommender systems, the term ‘feedback’ refers to any interaction that takes place between a given user and an item that provides us with important knowledge about the user's response to that particular item. The recorded feedback may be either positive (indicating that the item is liked/approved) or negative (indicating that the item is disliked/disapproved). Implicit feedback, explicit feedback, and hybrid feedback are the three main types of feedback that can be given.

1. Implicit feedback:

   User interactions that are indirectly related to the item are what make up implicit feedback. This includes using items, watching movies, adding items to a wish list/basket, subscribing to a channel, and other similar actions. This feedback cannot be used to assess the level of user satisfaction; nonetheless, it is very simple to get and simple to process. The amount of negative feedback that is recorded here is unusually low.

2. Explicit feedback:

   Explicit feedback refers to the direct interaction/rating given by the users in regard to the product in question. For example, liking or disliking a video, or offering ratings or reviews on different products. The level of satisfaction that a user has with an item can be easily recorded as either positive or negative, but it is tough to acquire as many users simply don’t bother to give explicit feedback.

3. Hybrid feedback:

   The combination of both implicit and explicit feedback is known as hybrid feedback. It is considered tough to acquire and handle this type of feedback. It

is possible to record both positive and negative reviews of an item based on how the consumer interacts with that item specifically. Every single one of today's e-commerce websites uses a hybrid kind of user input to power their respective recommendation algorithms. It majorly focuses on the sentiment expressed by the user for the given item.

2.6 Matrix Factorization

Matrix factorization is quite a popular and effective method that is utilised in a variety of domains, such as recommendation systems and data analysis. It is mainly used in the collaborative filtering techniques. It involves breaking down a given matrix into two or more lower-dimensional matrices in order to capture underlying patterns and relationships in the data. Matrix factorization is mainly used in collaborative filtering to create personalised item recommendations for users in the context of recommendation systems. This algorithm is very beneficial when dealing with sparse data, as not all users give feedback for all things. Furthermore, matrix factorization techniques such as Singular Value Decomposition (SVD) [10] and Alternating Least Squares (ALS) [12] have shown usefulness in boosting recommendation accuracy and resolving scalability issues.

2.7 Deep Learning in Recommender Systems

Deep Learning has recently become the state-of-the-art method to solve any and all complex problems regardless of the domain. Deep Learning is considered to be a small part of Machine Learning, like a subset of machine learning. Since the launch of ChatGPT in late 2022 by OpenAI [22], everyone has been hooked on the idea of using deep learning to create more intuitive solutions to a given problem. Most of the big tech companies like Google, Microsoft, YouTube, Amazon etc. have started implementing deep learning systems to provide better customer service experience.

In recent years, many industries have seen a steep rise in the implementation of deep learning in recommender systems. Making use of Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) has significantly improved the performance

of the recommendations as compared to the recommendations generated by the original methods like collaborative filtering and context-based filtering techniques. Deep learning-based systems take advantage of the artificial neurons to learn from the data and make appropriate recommendations to a given user.

Traditional recommender systems often make use of hand-crafted features [2, 3, 7], but deep learning models automatically learn hierarchical and abstract features through multiple layers of interconnected neurons [1]. This lets them find hidden connections and correlations that may not be clear from the raw data. Matrix factorization [10], autoencoders [23], and feedforward neural networks [22] are all common designs for recommender systems that use deep learning. These models are able to tackle problems like the cold start problem and the lack of feedback data available for a given area of interest. Deep learning models are also able to handle different types of input data, such as user-item interactions, textual descriptions, and images. Recommender systems using a neural network model work well because they can make highly personalised recommendations which are within context to any given user. This makes users more engaged and happier in a wide range of applications, from e-commerce platforms to content streaming services. However, their complexity and computational needs should be taken into account, and they need to be tuned carefully to find a good balance between speed and efficiency.

Chapter 3

Data Analysis

This chapter of the project describes how the dataset that will be used for training the recommender system. This section also addresses the preprocessing and formatting steps for the data to make it fit for the recommender models. I will also do some exploratory analysis of the data to get some insights into the video games and players involved in the dataset.

3.1 Steam Data

Steam platform is a very popular video game client which acts as a platform for huge library of PC video games. On this platform players can purchase games and in-game items (cosmetics items like weapon skins, avatars, clothes etc.) and play those game on said platform. The dataset that was used for my project was acquired from Julian McAuley's recommender systems datasets page [15]. This website is regarded as a reliable resource that is well-known for its reputation of supplying researchers with datasets of a high-quality. The data was acquired from the Steam API with the intention of making research and experimentation in the field of recommendation systems easier [16, 17, 18]. For my project I will be making use of two files first being the 'User-item_data.JSON' and 'Item_metadata.JSON'.

3.1.1 Dataset

1. User-Item data

   This file contains user-item interaction data that will be used as the foundation for building collaborative filtering model. The dataset is organized in a structured format. Key components of this file include unique user IDs, item counts (count

of games owned), steam IDs, user URLs and items(games). Each row corresponds to a user, and the ‘item_id’ list contains the games with their id in dictionary format that the user has bought.

2. Item Metadata

   Aside from the interaction data itself, the second file contains metadata for each game in the dataset. This metadata includes components such as game ID, game title, genre, release date, tags, price, reviews, and sentiments. This dataset will be helpful in retrieving the titles, rating, and game IDs to match and combine with the user-item file.

3.2 Preprocessing

In this subsection we will discuss about the data loading, formatting and preprocessing techniques used for prepping the dataset for recommender system.

3.2.1 Loading Data

The data for this project came from the files “australian_users_items.json” & “steam_games.json” which were downloaded from the Julian McAuley's website. It contains information about Australian users and the games they own and details about all the steam games until 2017. The files were in JSON format, the dataset was loaded by making use of the JSON built-in package in python. Then the data was converted into python dictionary and stored into a Pandas DataFrame.

3.2.2 Formatting Data

To begin, we created a new column to the user-item dataset named ‘uid’ in order to substitute the user IDs in the DataFrame, which weren't concise. This column was filled with consecutive integer values, one for each user in the dataset. Then the item IDs for each user were taken from the "items" column of the first DataFrame holding the user-item interaction data. With the help of a lambda function and Python's list comprehension, we added an extra field to the DataFrame labelled ‘item_id’ which

listed the item IDs for each user in a separate column in the original DataFrame. We also checked if there were any users who did not own any game by checking for null/missing values in the ‘items’ column and removed them from the dataset.

Finally, to streamline our models and to get a focus on only the essential data, we kept only the ‘uid’ and ‘item_id’ columns in the DataFrame and dropped all other columns which weren’t required for our specific goal. Then expanded the ‘item_id’ column into separate rows for each individual item per user. This helped in examining individual interaction between the user and items effectively.

Once we had the granular user-item interaction DataFrame, we loaded the second dataset file into a DataFrame and checked for null/missing values. Before dropping the other columns and keeping ‘user_id’, ‘item_id’, ‘title’, and ‘sentiment’ we made a separate DataFrame with all columns for conducting some data analysis. After that, we used the pandas merge function to combine the two DataFrames with the 'item_id' column as the key.

3.2.3 Sentiment Label Filtering

As the dataset doesn’t have explicit ratings in the appropriate format, we refined the dataset by applying sentiment label filtering and mapping the specific sentiment category to corresponding general numerical ratings ranging from 0 to 5. These rating were added onto a new column as ‘rating’ and the ‘sentiment’ column was dropped. The sentiment mapping for the dataset were assigned as follows:

1. Overwhelmingly Positive: 5
2. Very Positive: 4
3. Mostly Positive & Positive: 3
4. Mixed: 2
5. Mostly Negative & Negative: 1
6. Overwhelmingly Negative & Very Negative: 0

3.3 Data Visualization

Within this section, we will begin the process of visualising the data that is contained within our merged dataset. As we have enormous amount of data, it makes it difficult to interpret, comprehend, and get any meaningful insights from just by viewing it row by row.

First, we calculate the total number of unique values present in game, users, game publishers, total number of unique genre and the game developers. Table 1 give us an overall idea about how diverse our dataset is in terms of statistics. We made use of the tabulate package in python to create the summary statistics of the dataset.

Category	Count
Total Games	7677
Total Users	69277
Total Publishers	3522
Total Genres	569
Total Developers	4955

Table 1: Summary Statistics of Dataset

Next, we are going to concentrate on the years that the games were first made available to the public in order to obtain an idea of when the majority of the games that were purchased by customers occurred.

Figure 4 displays the amount of game releases between 2008 and 2018. The graph depicts the patterns in game release throughout the selected time period. The x-axis shows the years from 2008 to 2018, while the y-axis shows the number of games

launched in each year. The number of game releases varies throughout time. Particularly, 2018 saw the fewest game releases, due the original dataset being compiled in the beginning of that year. On the other hand, the year 2017 had the most game releases, indicating a robust expansion in the gaming business during that time period.

The bar graph efficiently visualises the fluctuations in game releases for almost a decade and provides significant insights into the growing gaming industry during this time.

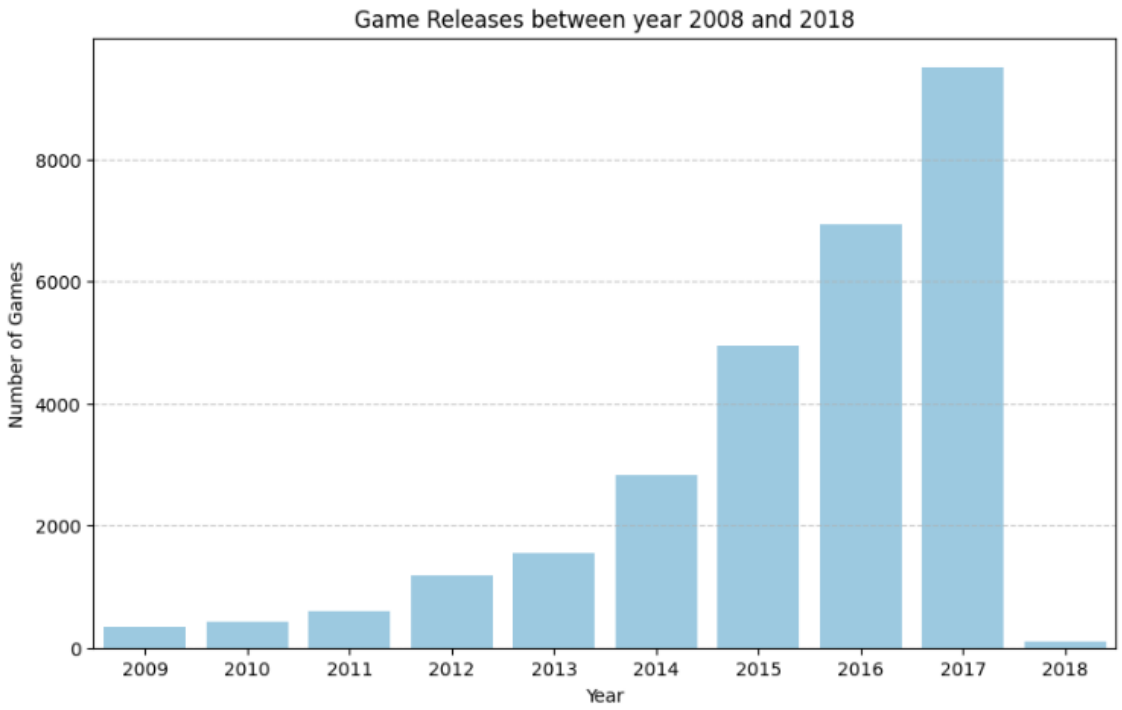

Figure 4: Number of Game Releases per Year (2008 - 2018)

The Figure 5 aggregates the game releases across all the available years in the dataset. The x-axis represents the months, ranging from January to December, and the y-axis indicates the count of game releases for each month.

Since the plot does not focus on a specific year but instead covers all available data in our DataFrame, it provides an overall view of the distribution of game releases across the months, irrespective of the individual years. The purpose of the plot is to show

the general pattern of game releases throughout the months, identifying any trends or seasonality in game launch patterns across different months.

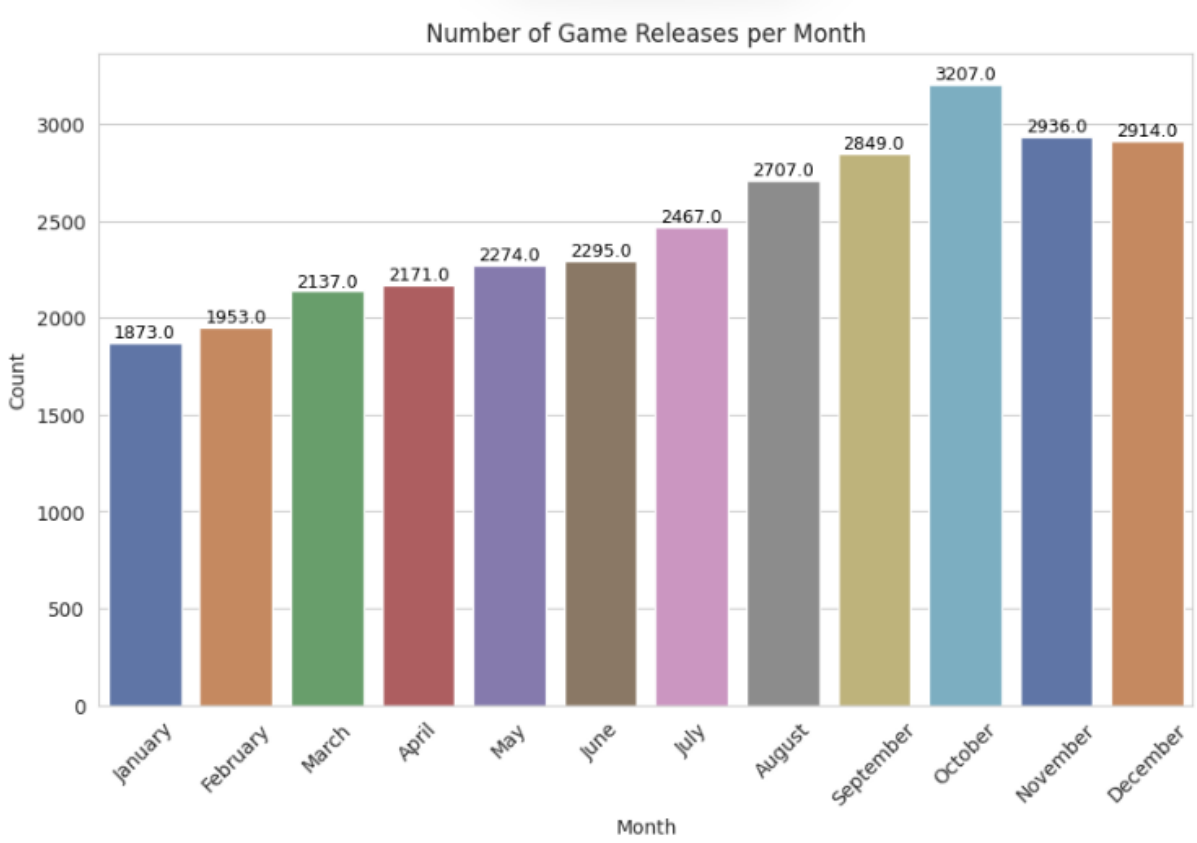

Figure 5: Aggregated Monthly Releases

3.4 Issues in the Steam Dataset

The steam user and meta item data available on the Julian McAuley's website was in JSON format, although when reading the data using the JSON library in python, we discovered that it was not valid JSON since it used single quotes instead of double quotes, which is required by the JSON linter in python. We considered several solutions, including switching to double quotes to get around this problem. But this caused further issues when a game's title contained an apostrophe, for example 'Assassin's Creed.' The python function 'ast.literal_eval()' helped in converting the data into python dictionary format and then we added the newly formatted data to an empty JSON file using JSON.dump() function and read the data correctly into a Pandas DataFrame.

Chapter 4

Research Methodology

This section of the project describes the two methodologies that will be used for building the recommender systems. Various recommendation systems have been in use for a long time now. Although there are several different algorithms and approaches to generate recommendations for users, many of them often fail to generalise appropriate recommendations. For this project, we will be using collaborative filtering model as a baseline for generating video game recommendations which will be compared with the recommendations from the Neural Collaborative Filtering Model.

4.1 Collaborative Filtering

The first recommender system will be based on the collaborative filtering technique which makes use of the user-item interaction dataset containing correlations with ratings. The collaborative filtering system used in this project is item-based, therefore it analyses the degree to which two objects have been rated similarly by the same user. The main purpose behind collaborative filtering is to map similar users that share common interest and the items rated by them are similar as well.

Neighbourhood approaches and latent factor models are the two main applications of collaborative filtering. The main focus of neighbourhood method is to calculate the interaction between items and users. According to ratings of "neighbouring" items given by the same user, item oriented method assesses a user's liking for a particular item. Products that receive generally comparable ratings when assessed by the same person are considered a product's neighbours[10]. For example, consider the game ‘DOTA 2’. Its neighbours might include similar Multiplayer Online Battle

Arena(MOBA) type of games like ‘Arena of Valor’ or the famous ‘League of Legends’ and few others. Figure 6 shows the user-item oriented approach, which identifies similar users who play and rate similar type of games.

Figure 6: User-item oriented Neighbourhood

4.1.1 Singular Value Decomposition

Singular Value Decomposition (SVD) is used as the matrix factorization technique to capture latent features from the user-item interaction matrix. “Matrix factorization algorithms decompose a matrix into components which can then be used for various purposes”[19]. When it comes to creating recommendations and reducing the value of Root Mean Squared Error (RMSE), our baseline recommendation system makes use of rating correlations in addition to singular value decomposition.

The SVD algorithm derives from linear algebra. It is mainly used as a technique to reduce the dimensionality in machine learning algorithms, which aids in reducing

the total amount of features in a chosen dataset by reducing the space dimension from the N to K (where K<N). As we know SVD is a matrix factorisation algorithm, in real time it breaks down a given user-item interaction matrix into three separate matrices, ***U, Σ*** and $\boldsymbol{V^T}$. "Given a ***m x n*** matrix ***A***, where ***m*** signifies the number of rows and ***n*** represents the number of columns, SVD can be represented as $\boldsymbol{A = U\Sigma V^T}$, where ***U*** is an ***m x m*** orthogonal matrix, ***Σ*** is an ***m x n*** diagonal matrix with non-negative real numbers on the diagonal (singular values), and $\boldsymbol{V^T}$ is the transpose of an ***n x n*** orthogonal matrix"[19].

$$\underset{m \times n}{\hat{X}} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix} \approx \underset{m \times r}{U} \begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix} \underset{r \times r}{S} \begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix} \underset{r \times n}{V^{\mathsf{T}}} \begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}$$

Figure 7: SVD Matrix Representation [20]

Figure 7 is an illustration of the SVD's matrix representation. Here, ***X*** denotes the utility matrix, and ***U*** represents the interaction between users and latent factors; it is a left singular matrix. Additionally, ***S*** is a diagonal matrix that describes the intensity of each latent factor, and finally, ***V transpose*** is a right singular matrix that represents the similarity between the items and latent factors [20]. The latent factors represent the features of the items, in our case the ratings of the games. SVD algorithm reduces the dimensionality of our utility matrix ***A*** by extracting its latent factors. Then it maps every user and every item into a ***r-dimensional*** latent space. This process gives way to a clear representation of the interaction and relationship between the users and items in our dataset.

Koren et al. illustrates the formation of the SVD approach by "mapping the users and items to a joint latent factor space of dimensionality ***f,*** such that user-item interactions are modelled as inner products in that space"[10]. We consider that each item is denoted by a vector $\boldsymbol{q_i}$ and each user is denoted by a vector $\boldsymbol{p_u}$. Therefore, the anticipated rating by a user ***u*** on an item ***i*** can be given as:

$$\hat{r}_{ui} = q_i^T \cdot p_u \quad (4.1)$$

In equation 4.1, $\hat{r}_{ui}$ is a type of factorisation in SVD algorithm. The vectors q_i and p_u are calculated so that the square error difference between their dot product and the predicted rating in the user-item matrix is minimised. A regularisation term is added to the equation in the form of a penalty to prevent the model from overfitting the training data and to ensure that it can generalise well. The algorithm also employs a bias term in order to limit the amount of error that occurs between predicted ratings and actual ratings. Koren et al. presented with a final equation as "For a given user-item pair *(u, i)*, μ is the average rating of all items, b_i is the average rating of item i minus μ and b_u is the average rating given by user u minus μ, the final equation after adding the penalty term and bias can be given as"[10]:

$$Min(q, p, b_i, b_u) \sum_{(u,i)\in K} (r_{ui} - q_i^T . p_u - b_i - b_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2 + b_i^2 + b_u^2) \quad (4.2)$$

The model is learned by the system by making adjustments to the previously predicted ratings. Nevertheless, the objective is to generalise those historical ratings in such a way that they may predict future ratings that are unknown. Therefore, the system should prevent itself from overfitting the predicted data by regularising the learned parameters. The value of the constant λ, which handles the level of regularisation, is often arrived at through the process of cross-validation.

4.1.2 Surprise – Python Library

We will be making use of the Surprise Python Library to carry out the collaborative filtering with the Singular Value Decomposition (SVD) approach. There are many Python libraries and packages available that enable users to perform collaborative filtering, but we selected the Surprise library due to its simplicity, efficiency, and specialized focus on recommendation systems. The Surprise library offers a comprehensive set of tools and algorithms tailored for recommendation tasks, streamlining the implementation of various collaborative filtering techniques [21]. The Surprise library has many useful built-in functions to execute recommender algorithms with ease. It even provides a way to read a custom dataset using its

Reader class and allows the user to choose any recommender algorithms included in their library.

We made use of the following inbuilt classes/modules to carry out the SVD algorithm:

1. Reader class: This class is used create a reader object that defines the explicit ratings present in our dataset (1 to 5).
2. Dataset Class: The surprise library offers this module to read a custom dataset into the appropriate format required for the SVD model.
3. Split Class: This module offers train_test_split() function for dividing the dataset into train set and test set.
4. SVD Algorithm: Here we initialise the SVD algorithm and set the number of epochs and learning rate with it.
5. Cross Validation: The library also provides with an inbuilt cross_validate() function. This helps in evaluating the SVD algorithm's performance using metrics like Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) with 5-fold cross-validation.

4.2 Neural Collaborative Filtering

Neural Collaborative Filtering (NCF) is an approach for using Deep Neural Networks (DNN) in order to perform collaborative filtering in recommender systems [11]. The NCF model shows the limitations of Matrix Factorization, which is a simple inner product of hidden features that demonstrates how a user interacts with an object and how the item impacts the user. [19].

4.2.1 Matrix Factorization

In the paper published by Xiangnan et al., a solution is found by combining Generalised Matrix Factorization (GMF) and Multi-layer Perceptron (MLP) into a single design [11]. They illustrated the limitations of matrix factorization wherein,

most of the time MF fails when dealing with huge amount of sparse data. As discussed in previous sections, Matrix Factorization implements collaborative filtering by calculating the dot product of two vectors. Consider, vector $\boldsymbol{p_u}$ and vector $\boldsymbol{q_i}$ which represent the users and items respectively and $\hat{\boldsymbol{y}}_{\boldsymbol{ui}}$ is the represents the predicted rating. The interaction function is as follows:

$$\hat{y}_{ui} = f(u, i|p_u^T, q_i) = p_u^T q_i = \sum_{K=1}^{K} p_{uk}\, q_{ik} \tag{4.3}$$

Equation 4.3 shows the calculation for predicted rating using Matrix Factorisation, where K is the dimension of the user and item latent feature vector. “MF models the two-way interaction of user and item latent factors, assuming each dimension of the latent space is independent of each other and linearly combing the with the same weight” [11].

Figure 8 demonstrates the limitations of matrix factorization; it informs us that, when calculating unit length latent vectors, we can also measure the cosine of the angle between two users' latent vectors to determine how similar they are. Only matrix factorization, which maps people and items to the same latent space, could make this possible. In case we choose to fix the problem by picking a higher-dimensional K, then it will lead to overfitting, which in turn will damage the model's ability to generalise, thus it is advised to not do that.

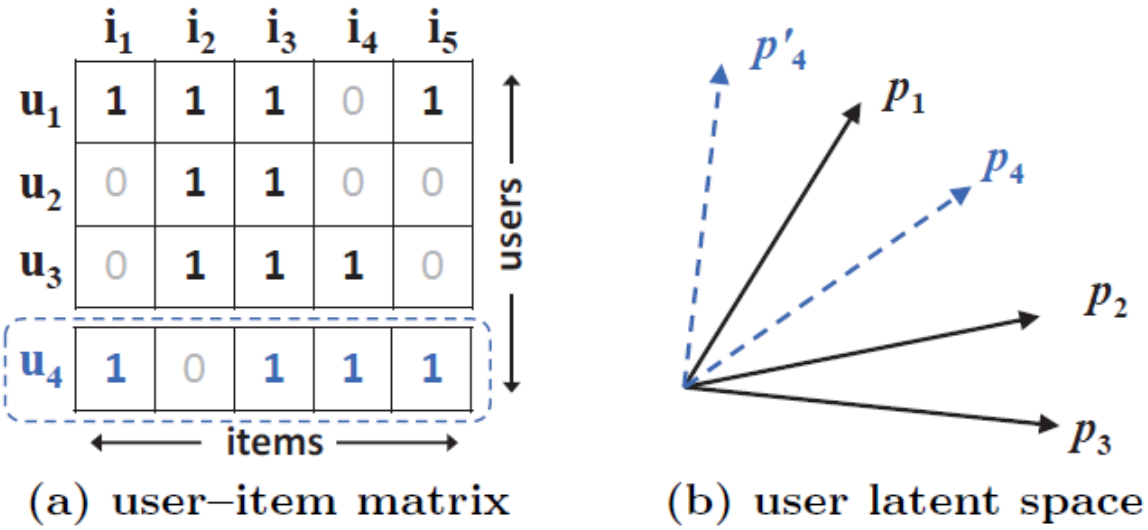

Figure 8: (a) User-Item Matrix & (b) User Latent Space [11]

4.2.2 NCF Architecture

The NCF model consists of four layers, first being the input player, followed by the embedding layer, the neural CF layer, and lastly the output layer. Figure 9 shows us the graphical representation of all the layers involved in the neural collaborative filtering framework, where the output from one layer is fed as the input into the next layer.

1. **Input Layer:** This is the first layer in the framework, which takes in input as one-hot encoded sparse item and user feature vectors v_u^U & v_i^I in binary format.
2. **Embedding Layers:** These layers are followed by the Input layer and accept the binarized sparse vectors from the previous layers as input. These layers map users and items to dense, continuous vectors in a lower-dimensional space, known as embeddings. Embeddings are learned through the process of optimization that takes place during training and they assist in capturing latent features that contribute to user-item interactions.
3. **Neural Network Structure:** The embeddings from the previous layer are fed into a multi-layer neural network (which is called as the neural collaborative filtering layers) that calculates the interaction probability or game rating predictions in our case. The neural network architecture can vary in terms of the number of layers, but it usually involves fully connected layers with activation functions.
4. **Output Layer:** This is the final layer of the neural collaborative filtering model; the function of this layer is to predict the target rating that is $\hat{y}_{ui}$. In the end, we do training at this layer by attempting to minimize the pointwise loss between the current value of $\hat{y}_{ui}$ and its corresponding target value y_{ui}.
5. **Loss Function:** The loss function calculates the difference between the predicted ratings and the actual ratings. We choose the loss functions based on our expected output which in our case is ratings between 1- 5, in the case of non-binary ratings, mean squared error (MSE) works best. The model

parameters, including the embeddings and neural network weights, are updated through backpropagation to minimize this loss.

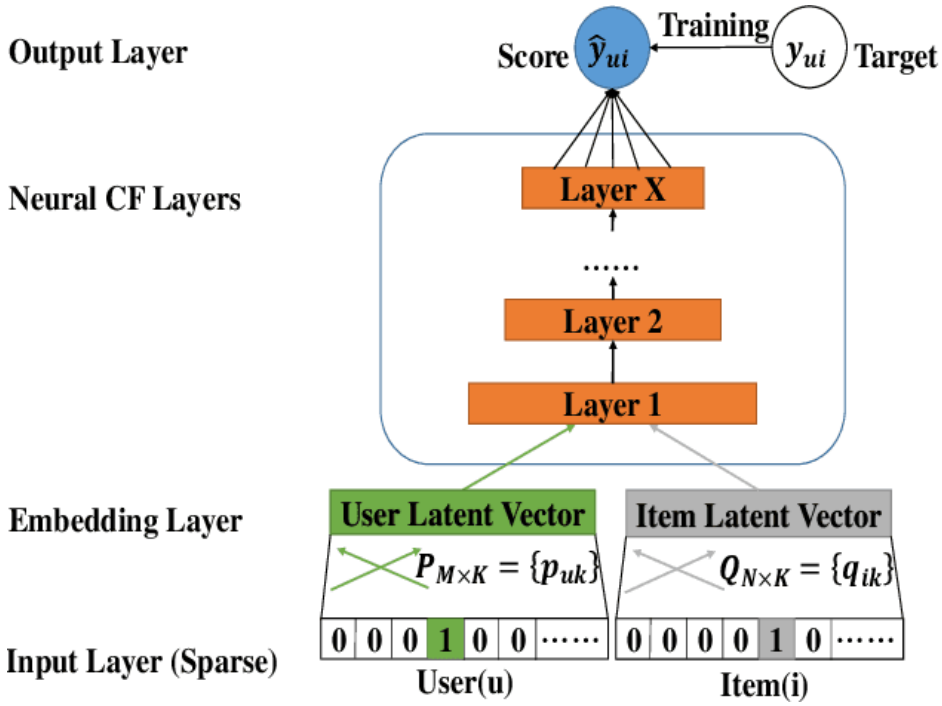

Figure 9: Neural Collaborative Filtering Framework [11]

4.2.3 TensorFlow – Keras

In the context of the NCF architecture outlined in the above section, we implemented the Neural Collaborative Filtering architecture by making use of TensorFlow's Keras library [25]. Keras is a Python-based open-source application programming interface (API) that enables the use of deep learning algorithms. The API offers a user-friendly and advanced interface for the purpose of constructing, designing, and training neural networks. Keras has gained significant attention and widespread adoption within the machine learning and artificial intelligence sectors due to its objective of enhancing accessibility and efficiency in deep learning.

Using TensorFlow's Keras library, we built a simple Neural Collaborative Filtering (NCF) model. We first started with preprocessing the dataset by mapping 'user_id'

and ‘game_id’ to integers, which gives the model an appropriate form of input to work with. The dataset is then split into train, validation, and test sets, which makes it possible to evaluate the model well. The NCF model is the most important part of the approach. First, we create the user and item embeddings and flatten them and then concatenate them before passing it to the first dense layer with 64 neurons/units followed by a dropout layer with a dropout rate of 0.2, In total we make use of 3 dense layers with 64, 32 and 18 neurons respectively with ‘relu’ activation function and L2 regularisation to make the model more general. These layers turn the original sparse binary vectors into dense continuous embeddings in an efficient way. These embeddings capture latent traits of how a user interacts with an item. Dropout layers in particular are added to make the model more resistant to overfitting.

4.3 Evaluation Metrics

For this project we will be making use of two evaluation metrics for both the recommendation systems. As machine learning and deep learning models which generate prediction or recommendations can be considered as a regression problem and are better evaluated using error rate then accuracy. As we cannot have ground truth in a recommender system dataset, we make use of error rates like Root Mean Squared Error and Mean Absolute Error to assess the performance of our models. We can calculate the difference between the actual ratings of the games given in the dataset and the predicted ratings from the models as a basis to compare and evaluate models.

4.3.1 Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) is a common way to measure the error rate in regression/prediction models. RMSE shows how much, on average, the expected values are different from the real values from the dataset. It calculates the square root of the squared aggregate mean of the difference between the actual and predicted values. The RMSE equation can be given as:

$$\text{RMSE} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{N}} \quad (4.4)$$

Where:

- y_i represents the actual ratings.
- $\hat{y}_i$represents the predicted ratings.
- N is the number of ratings in the dataset.

Results are more successful when the root-mean-square error (RMSE) between predicted and observed values is smaller. RMSE is especially useful for measuring how accurate prediction models are.

4.3.2 Mean Absolute Error (MAE)

Mean Absolute Error is also another way to evaluate the performance of a model. Unlike the previous metric MAE calculates the average of the absolute difference between the actual ratings predicted ratings. MAE is also known as L1 loss function and is considered to be one of the simplest loss functions for measuring loss.

The MAE equation can be given as:

$$\text{MAE} = \frac{\sum|y_i - \hat{y}_1|}{N} \quad (4.5)$$

Where:

- y_i represents the actual ratings.
- $\hat{y}_i$represents the predicted ratings.
- N is the number of ratings in the dataset

Just like RMSE , lower the MAE score, higher is the predictive performance of the model.

Chapter 5

Experiments and Results

In this chapter we discuss about experiments conducted on the steam dataset using singular value decomposition model and neural collaborative filtering model. This section also gives an overview of the results obtained from those models and its interpretation. We will also look at the top 10 video game recommendations given for a chosen user and draw insights from those recommendations.

5.1 Experimentation

In the previous chapter we discussed the methodology for the Singular value Decomposition Model and Neural Collaborative Filtering Model. In this section, we will perform experiments with the models to assess the performance of the two models.

5.1.1 SVD Model

In section 4.1.2 we laid out the classes that were used for implementing the SVD model. To make build our baseline recommender model, we first make a Reader object, which lets us parse the ratings present in our dataset by providing a rating scale of 1 to 5. Next, we use 'Dataset.load_from_df()' function to load the steam dataset, into the Surprise Dataset format.

To get better results from the model we set up the cross-validation process to figure out how well the SVD model works. Then we use the 'train_test_split()' method to split the data into training and testing sets, with a test size of 25%. Then we load the

SVD algorithm with certain hyperparameters, like the number of epochs (n_epochs), learning rate (lr_all) and regularisation term for all parameters (reg_all). These hyperparameters affect how the model is trained and how well it can find latent features. After setting that up, we use the 'cross_validate()' method to do a 5-fold cross-validation. In this process, the SVD algorithm is trained on a part of the data and then tested on data it has never seen before. Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) are used to measure how accurate the model is at making predictions and how well it can generalise. These metrics are calculated and added up across all folds.

	Fold 1	*Fold 2*	*Fold 3*	*Fold 4*	*Fold 5*	*Mean*	*Std*
RMSE	*0.1015*	*0.1005*	*0.1013*	*0.1013*	*0.1014*	*0.1012*	*0.0004*
MAE	*0.0638*	*0.0635*	*0.0636*	*0.0634*	*0.0637*	*0.0636*	*0.0001*
Fit time	*14.41*	*15.20*	*16.02*	*14.97*	*14.82*	*15.09*	*0.53*
Test time	*12.42*	*7.92*	*9.18*	*9.31*	*6.71*	*9.11*	*1.91*

Table 2: Cross Validation results on 5 Epochs

Table 2 and Table 3 shows the RMSE and MAE error rate across 5 folds. We conducted two experiments with the SVD model, Table 1 shows the performance of the cross validation on 5 epochs, learning rate = 0.01 and regularisation rate = 0.02. Whereas we can see the difference in the values in table 2 which displays performance on 10 epochs with the same learning rate and regularisation rate.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE	0.0780	0.0785	0.0784	0.0790	0.0788	0.0785	0.0003
MAE	0.0490	0.0491	0.0490	0.0490	0.0488	0.0490	0.0001
Fit time	25.74	27.93	28.16	27.71	30.06	27.92	1.37
Test time	6.87	10.00	10.04	6.90	10.91	8.92	1.69

Table 3: Cross Validation results on 10 Epochs

Figure 10 displays the Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) scores for the training data, which have been evaluated across 5 folds. The blue line represents the performance after 5 epochs, while the orange line represents

the performance after 10 epochs. It is evident that the model's performance exhibits slight improvement when trained for 10 epochs.

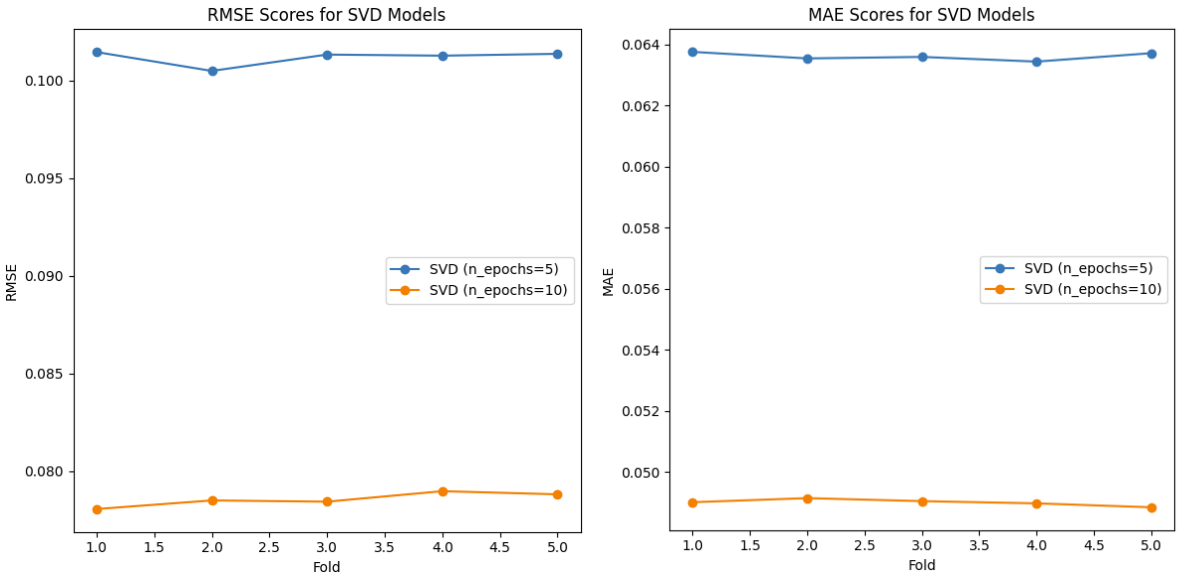

Figure 10: Comparison of RMSE and MAE Values for SVD

To get a better understanding of model performance we can take look onto figure 11 and 12 which shows us the learning curve for 5 and 10 epochs respectively. These learning curves indicate the relationship between the number of epochs and the corresponding RMSE values for both train and test sets.

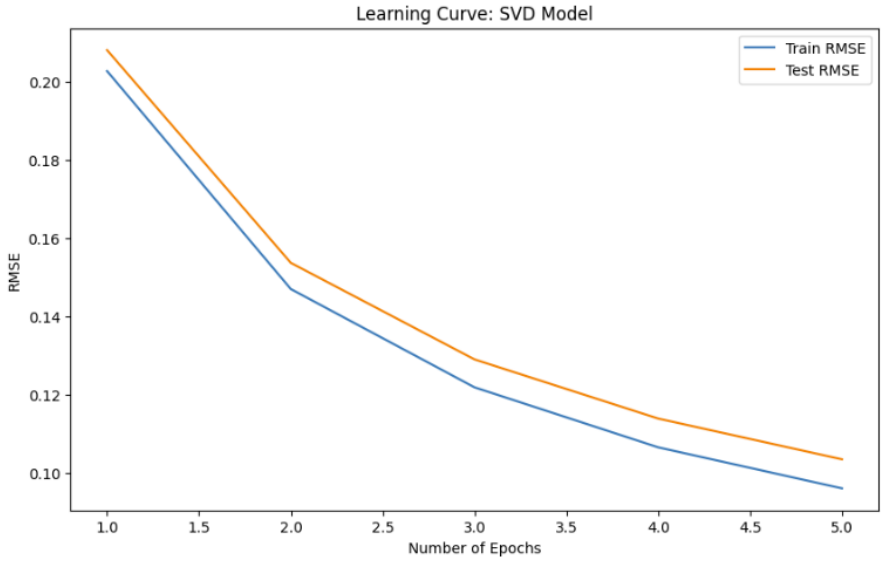

Figure 11: Leaning curve for the SVD Model for 5 Epochs

Upon examination of Figure 11, which shows the learning curve for 5 epochs, it becomes apparent that the model shows the ability to generalize well. Despite varying the number of epochs, the RMSE values for both the training and testing datasets remain consistent. This indicates that the model's performance remains stable across different epochs, suggesting that the model's predictive accuracy achieves a certain level of stability after a few epochs of training.

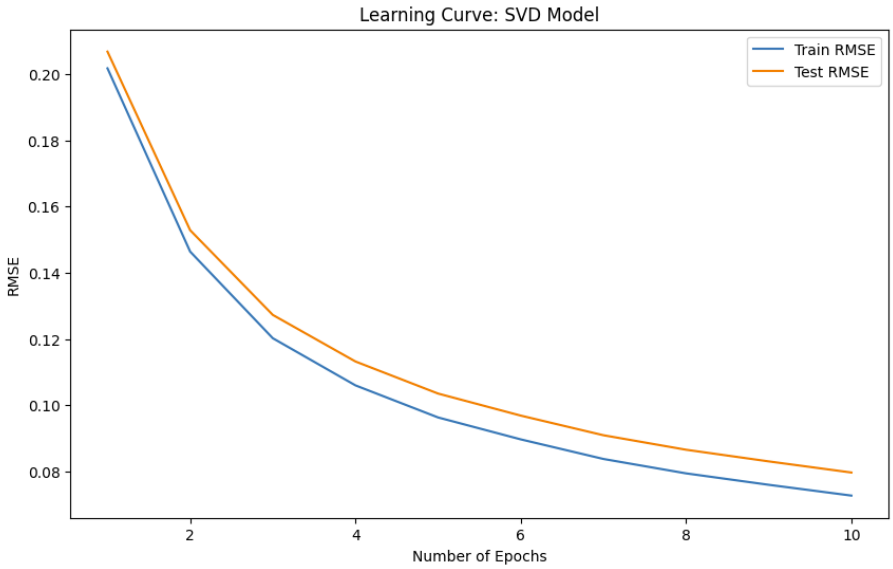

Figure 12: Leaning curve for the SVD Model for 10 Epochs

Looking at Figure 12, which showcases the learning curve for 10 epochs, it confirms the model's robustness and generalization capability. Similar to the learning curve for 5 epochs (figure 11), the RMSE values for both the training and testing datasets maintain a minimal difference. The consistent performance observed in the SVD model highlights its ability to capture underlying patterns in the data without overfitting. This is clear from the fact that even after more training, the model's performance on the test set stays consistent with its performance on the training set.

5.1.2 NCF Model

In section 4.2.2 and 4.2.3 we discussed the NCF architecture and Keras implementation of the NCF model. We trained the model on training set for different and evaluated with the validation set with Mean Squared Error as the loss function

for non-binary ratings. The model was trained with variations in the number of epochs and batch sizes, we had to limit the number of epochs to 10 due to hardware limitations. Table 4 displays the average MSE training and validation loss for each combination of epoch and batch size. As observed from the table, the average training loss decreases with the increase in the number of epochs, indicating that the model is learning to fit the training data more closely. Similarly, the validation loss also decreases, demonstrating that the model's performance improves on unseen validation data. It is noteworthy that a smaller batch size of 128 led to slightly lower average validation loss compared to a batch size of 256, possibly due to more frequent updates in the optimization process.

Epochs	Batch Size	Average Training Loss (MSE)	Average Validation Loss (MSE)
5	128	0.0080	0.0015
5	256	0.0132	0.0019
10	128	0.0043	0.0012
10	256	0.0109	0.0017

Table 4: MSE Values for NCF Model

Similarly, figures 13, 14, 15 and 16 show the learning curves for the fitting the model on different epochs and batch sizes.

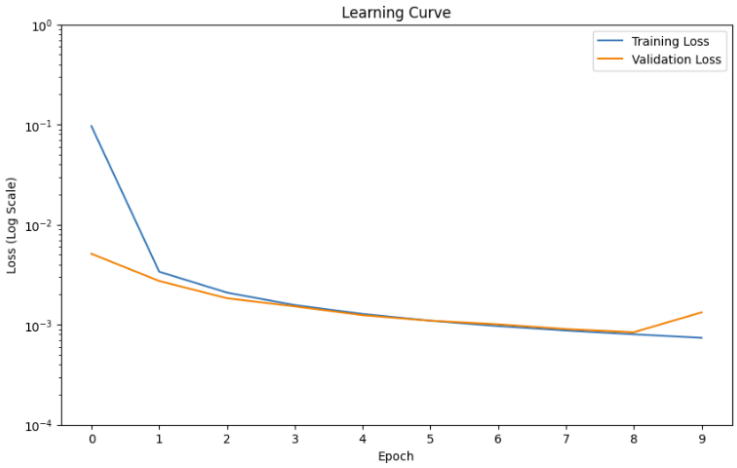

Figure 13: NCF Learning Curve on 10 epochs & 256 as batch size

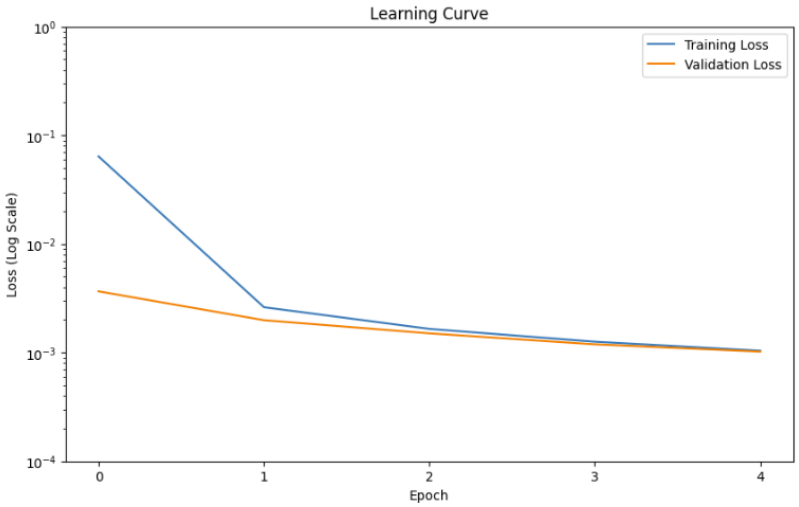

Figure 14: NCF Learning Curve on 5 epochs & 256 as batch size

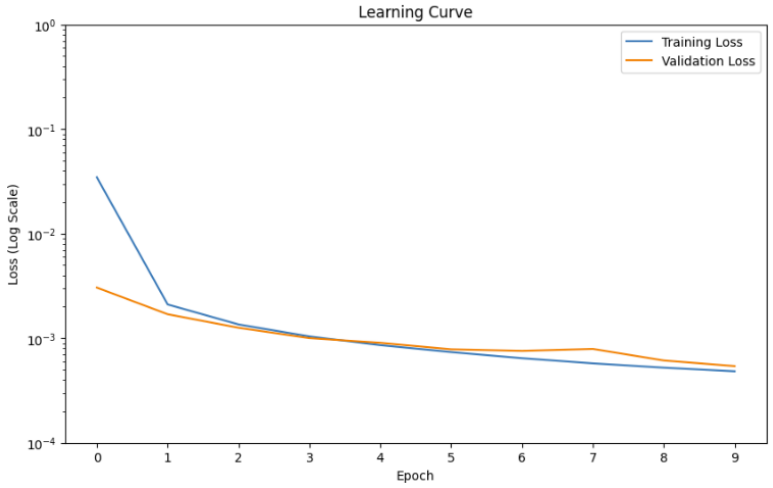

Figure 15: NCF Learning Curve on 10 epochs & 128 as batch size

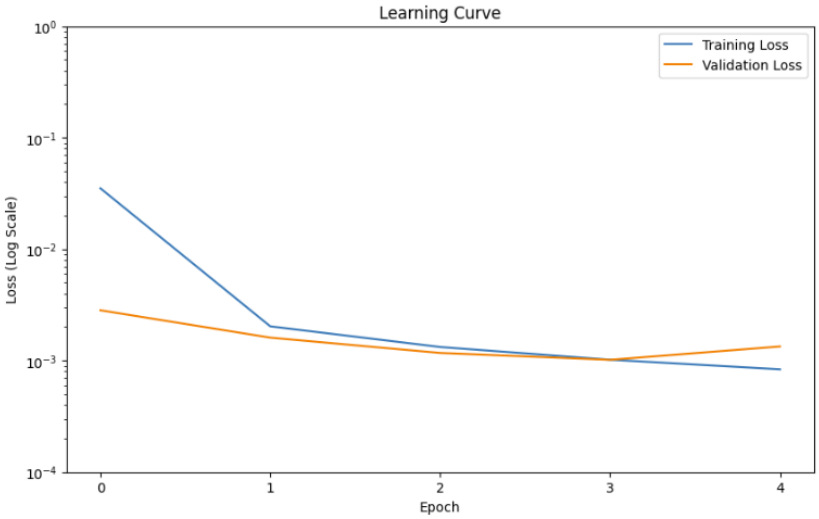

Figure 16:NCF Learning Curve on 5 epochs & 128 as batch size

5.2 Results

In the previous section we discussed the experimentation for the Singular value Decomposition Model and Neural Collaborative Filtering Model. In this section, we will talk about the results achieved from those experiments and answer the research question posed in section 1.2.

5.2.1 RQ1 – Model Comparison

To address RQ1, we evaluated the performance of two collaborative filtering models: the Neural Collaborative Filtering (NCF) model and the Collaborative Filtering model using Singular Value Decomposition (SVD). The evaluation was conducted using the root mean squared error (RMSE) metric.

Figure 17 shows the RMSE values obtained from both models, we can observe a significant difference in their performance. The Collaborative Filtering model based on Singular Value Decomposition yielded an RMSE value of 0.0785 on 10 epochs. In contrast, the Neural Collaborative Filtering model achieved a remarkably lower RMSE value of 0.0012 over 10 epochs. This big difference shows that the NCF model is better at making predictions than the SVD-based Collaborative Filtering.

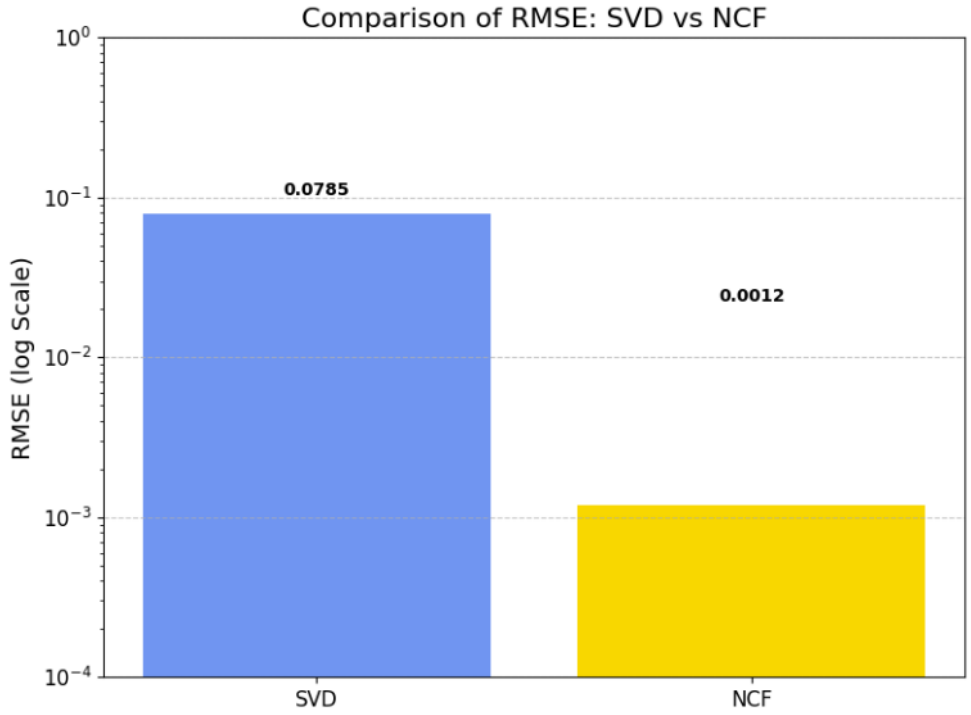

Figure 17: Comparison of SVD & NCF using RMSE

Due to its multi-layered neural network architecture, the NCF model is able to capture complex nonlinear relationships and latent features in user-item interactions. This enables the NCF model to generate highly accurate rating predictions and game recommendations, especially when dealing with sparse and diverse datasets, such as the one used in this project.

In conclusion, the result of our evaluation clearly answers our first research question that the NCF model outperforms the Collaborative Filtering using SVD in terms of predictive accuracy, as demonstrated by the significantly lower RMSE value.

5.2.2 RQ2 – Generating Game Recommendations

We generated top 10 game recommendation for a chosen user using both models. We also made sure that the models will recommend games which aren't rated by that specific user.

The SVD model recommended a set of games that exhibit genres like 'Action', 'Adventure', 'Indie', 'RPG', and 'Strategy' (Figure 18). The recommended titles include 'Crimzon Clover: World Ignition', 'Contradiction - Spot the Liar!', 'Planetarian: The Reverie of a Little Planet', 'SteamWorld Heist', 'Unreal Gold', and more. These games indicate that the user enjoys playing action/adventure games. Although these game titles are fairly considered to be games developed by small indie studios which don't usually reach everyone's eyes and are often given poor scores by the gaming communities.

	Game ID	Title	Genre	Original Rating	Predicted Rating
0	285440	Crimzon Clover WORLD IGNITION	['Action', 'Indie']	Not rated	5.000000
1	373390	Contradiction - Spot The Liar!	['Adventure', 'RPG', 'Strategy']	Not rated	5.000000
2	316720	planetarian ~the reverie of a little planet~	['Casual']	Not rated	5.000000
3	322190	SteamWorld Heist	['Action', 'Adventure', 'Indie', 'RPG', 'Strat...	Not rated	4.987908
4	13250	Unreal Gold	['Action']	Not rated	4.987515
5	413420	Danganronpa 2: Goodbye Despair	['Adventure']	Not rated	4.983036
6	250050	Life Goes On: Done to Death	['Action', 'Adventure', 'Casual', 'Indie']	Not rated	4.982937
7	411830	SENRAN KAGURA SHINOVI VERSUS	['Action']	Not rated	4.979972
8	368340	CrossCode	['RPG', 'Early Access']	Not rated	4.971272
9	70400	Recettear: An Item Shop's Tale	['RPG', 'Indie']	Not rated	4.968750

Figure 18: Top 10 Recommendations from SVD

On the other hand, the NCF model also recommends games with similar genres like 'Action', 'Adventure', 'Indie', 'Simulation', and 'Strategy' (Figure 19). The recommended titles include 'Counter-Strike', 'Valiant Hearts: The Great War', 'The Walking Dead: Season 2', 'Ori and the Blind Forest', 'Creeper World 3: Arc Eternal', and more. The NCF model, like the SVD model, gives these games a very high predicted rating (quite near to 5.0). This indicates that the NCF model considers these games to be engaging for the chosen user. Although by taking a quick look at the game titles, a seasoned gamer will immediately recognise that more than half of the recommendations are quite famous and are highly rated games in the online gaming community. The first game 'Counter Strike' was supposedly the highest played game on the steam platform few years back.

	Game ID	Title	Genre	Original Rating	Predicted Rating
0	10	Counter-Strike	['Action']	Not Rated	5.000000
1	260230	Valiant Hearts: The Great War™ / Soldats Incon...	['Adventure']	Not Rated	4.993598
2	261030	The Walking Dead: Season 2	['Adventure']	Not Rated	4.990521
3	261570	Ori and the Blind Forest	['Action']	Not Rated	4.987030
4	280220	Creeper World 3: Arc Eternal	['Indie', 'Simulation', 'Strategy']	Not Rated	4.983040
5	331470	Everlasting Summer	['Adventure', 'Casual', 'Free to Play', 'Indie']	Not Rated	4.980342
6	374570	Kung Fury	NaN	Not Rated	4.979933
7	13240	Unreal Tournament: Game of the Year Edition	['Action']	Not Rated	4.977120
8	13230	Unreal Tournament 2004: Editor's Choice Edition	['Action']	Not Rated	4.973420
9	245170	Skullgirls	['Action', 'Indie']	Not Rated	4.968434

Figure 19: Top 10 recommendations from NCF

5.2.3 Insights

The difference between the recommendations generated by the NCF model and the SVD model is a bit intriguing. The NCF model recommends games that have already gained a lot of popularity and contain significant player base on a global scale, these titles are usually a part of a popular series or have widespread popularity. The NCF model's ability to gather and use collective preferences from a wide range of users may explain why it tends to favour games that have gained large attention.

The SVD model, on the other hand, takes a different approach by recommending games that were made by independent developers with limited budgets. So, the SVD

model tends to recommend games that may not be well-known but have unique gameplay mechanics, new features, or intriguing stories. By focusing on games that might only appeal to a small group of people, the SVD model gives a diverse range of recommendations that fit a wider range of tastes. This shows that it can find hidden gems from the steam dataset. The NCF model tends to recommend games that are already quite popular and have a sizable player base around the world, whereas the SVD model favours games that were developed by smaller studios on a tighter budget, resulting in a more diverse set of recommendations.

Chapter 6

Conclusion

In this project, we have discussed about history of recommender system and the need for recommender systems in the gaming industry. It has yielded some valuable insights into the applications of video game recommendations.

Given the advancement in the field of Artificial Intelligence, the scope of this research is ever expanding. This thesis was aimed at comparing the performance of machine learning recommender system and deep learning recommender system and generating video game recommendation for a specific chosen user.

Throughout this study, we discussed and implemented two distinct recommendation models, first being the Collaborative Filtering using Singular Value Decomposition (SVD) using the Surprise python library and next being the Neural Collaborative Filtering (NCF) model using TensorFlow's Keras. We made use of the Julian McAuley's steam dataset and performed some quantitative data analysis to get an idea of the users and games throughout the years. The experimentation phase involved training and evaluating these models on the steam dataset of user-game interactions. The primary evaluation metric used was Root Mean Squared Error (RMSE), it was used to measure how well the models could predict the video game ratings. Notably, the NCF model always did better than the SVD model, with much lower RMSE values across different epochs and batch sizes. This performance advantage shows that the NCF model can capture complex user-item interactions and make better recommendations as a result.

The NCF model did a great job of recommending popular games that a lot of people rated because it was able to use user preferences as a whole. On the other hand, the SVD model was good at recommending less popular games with unique features that

fit niche tastes. These insights show how each model has the ability to meet the needs of different users, whether they want to find well-known games or try something new.

The experimentation conducted in this thesis paves a way for further research in the field of deep learning. Further studies could potentially investigate making use of the other fields in the dataset like genre, reviews, and price to generate more precise recommendations for a user.

Bibliography

[1] G. &. G. G. J. A. &. P. D. Cheuque, “Recommender Systems for Online Video Game Platforms: The Case of STEAM.,” in *Companion The 2019 World Wide Web Conference*, 2019.

[2] B. &. J. B. &. F. D. &. D. &. H. &. J. &. S. &. S. S. Schafer, “Collaborative Filtering Recommender Systems,” in *The Adaptive Web*, Springer, 2007, pp. 291-324.

[3] T. S. Z. S. R. K. a. M. M. S. M. Anwar, “A game recommender system using collaborative filtering (GAMBIT),” in *14th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, Islamabad, 2017.

[4] Y. K. a. C. V. Y. Hu, “Collaborative Filtering for Implicit Feedback Datasets,” in *Eighth IEEE International Conference on Data Mining*, Pisa, 2008.

[5] L. B. E. E. D. L. Y. R. a. J. B. H. Steck, “Deep Learning for Recommender Systems: A Netflix Case Study”,” in *AI Magazine*, 2021, pp. 7-18.

[6] B. S. a. G. Linden, “Two Decades of Recommender Systems at Amazon.com,” *IEEE Internet Computing,* vol. 21, pp. 12-18, 2017.

[7] E. Rich, “Stereotypes and User Modeling,” in *Springer-Verlag*, Berlin, 1989.

[8] D. N. B. M. O. a. D. T. David Goldberg, “Using collaborative filtering to weave an information tapestry,” *Communications of the ACM,* vol. 35, no. 0001-0782, pp. 61-70, 1992.

[9] S. N. N. W. M. B. M. H. M. A. K. Gosh, “Recommendation System for E-commerce Using Alternating Least Squares (ALS) on Apache Spark,” in *Springer*, Cham, 2021.

[10] Y. a. B. R. a. V. C. Koren, “Matrix Factorization Techniques for Recommender Systems,” *Computer,* vol. 42, pp. 30-37, 2009.

[11] X. a. L. L. a. Z. H. a. N. L. a. H. X. a. C. T.-S. He, “Neural Collaborative Filtering,” in *International World Wide Web Conferences Steering Committee*, Geneva, 2017.

[12] S. a. N. N. a. W. M. a. B. M. a. H. M. a. A. K. Ghosh, “Recommendation System for E-commerce Using Alternating Least Squares (ALS) on Apache Spark,” in *Springer International Publishing*, Cham, 2021.

[13] C. L. Y.-C. Z. T. Z. Zi-Ke Zhang, “Solving the cold-start problem in recommender systems with social tags,” *Europhysics Letters,* vol. 92, p. 28002, 2010.

[14] R. J. M. R. N. Prem Melville, “Content-boosted collaborative filtering for improved recommendations,” in *American Association for Artificial Intelligence*, Edmonton, Alberta,, 2002.

[15] J. McAuley, “Recommender Systems Datasets,” 2017. [Online]. Available: https://cseweb.ucsd.edu//~jmcauley/datasets.html#steam_data. [Accessed 28 06 2023].

[16] J. M. Wang-Cheng Kang, “Self-attentive sequential recommendation,” in *ICDM*, Singapore, 2018.

[17] J. M. Mengting Wan, “Item recommendation on monotonic behavior chains,” in *Association for Computing Machinery*, New York, 2018.

[18] K. G. J. M. Apurva Pathak, “Generating and personalizing bundle recommendations on Steam,” in *Association for Computing Machinery*, New York, 2017.

[19] S. Shah, “Introduction to matrix factorization for recommender systems,” 2018.

[20] B. S. L. E. S. A. S. T. Daniel Lew, “Recommender systems,” 2007. [Online]. Available: https://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/about.html. [Accessed 27 07 2023].

[21] N. Hug, “Surprise: A Python library for recommender systems,” *The Journal of Open Source Software,* vol. 5, no. 52, p. 2174, 2020.

[22] OpenAI, “Introducing ChatGPT,” 30 November 2022. [Online]. Available: https://openai.com/blog/chatgpt. [Accessed 09 08 2023].

[23] M. a. Q. H. a. Y. Z. a. L. L. a. L. Y. Fu, “A Novel Deep Learning-Based Collaborative Filtering Model for Recommendation System,” *IEEE Transactions on Cybernetics,* vol. 49, no. 3, pp. 1084-1096, 2019.

[24] Y. a. D. C. a. Z. A. X. a. E. M. Wu, “Collaborative Denoising Auto-Encoders for Top-N Recommender Systems,” in *Association for Computing Machinery*, New York, USA, 2016.

[25] S. Banerjee, “Collaborative Filtering for Movie Recommendations,” Keras, 24 05 2020. [Online]. Available: https://keras.io/examples/structured_data/collaborative_filtering_movielens/#show-top-10-movie-recommendations-to-a-user. [Accessed 13 08 2023].