Abstract

The recommender system has become one of the most essential roles in recent e-commerce business models. However, it still faces significant challenges, such as data sparsity and the cold start problem. This study aims to compare the performance difference between traditional and modern movie recommender system models, focusing on their ability to cope with sparsity and cold start issues using a limited dataset (user-movie rating matrix as input only). A series of experiments are performed to compare various recommenders from the conventional content-based recommender to the matrix completion model and neural network models. Performance was evaluated using quantitative metrics such as accuracy, precision, recall, F1 Score, and RMSE. The results indicate that given limited data, a matrix factorization neural network model representing modern deep learning models and user-based collaborative filtering recommender systems representing traditional methods can both provide good recommenders facing data sparsity and cold start issues. These findings suggest that while deep learning models such as a matrix factorization neural network offer better performance especially in a cold start user case, conventional methods such as user-based collaborative filtering can still be a viable option and have the potential to be implemented into hybrid models for even better performance. With limited data resources, both recommender systems can provide reasonable predictions. However, building a model with neural networks can help shorten the runtime when making future predictions while conventional method may require starting the computation from scratch again. The result for the study is particularly useful for smaller companies and startups that want to know recommender systems from basic and can only collect or handle limited data. The project provides good reviews on recommender systems and suggests suitable models tailored for such scenarios.

Index Terms: Recommender systems, Collaborative Filtering, Matrix Completion Recommender (SVT, ALS), Matrix Factorization Recommender, Neural Network Recommender (MLP, Autoencoder)

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Recommender Systems (the term "RS" will sometimes be used to represent a recommender system in this project.) are everywhere now. We see them on Spotify, YouTube, Amazon, Netflix, and other platforms where user ratings or viewing history are used to generate a list of recommended products or services that users might be interested in. It all started with Chris Anderson's (2007) "*The Long Tail Theory*" [2], which stated that less popular products that are in lower demand could actually increase in profitability. Therefore, companies started to recommend tons of products and services to their customers in order to make use of them.

However, companies soon began to notice a decline in customer satisfaction and overall sales, and they realized that something had gone wrong. According to research conducted by *Iyengar and Lepper (2000)* [18], even though an increase in the number of available options to consumers may initially seem appealing, it can actually reduce their motivation to buy a product later on. One possible reason is that customers might be feeling overwhelmed by the large number of available options, which makes it hard for users to make decisions effectively. This phenomenon is also known as *"Information Overload"* early mentioned in the study by Edmunds and Morris (2000) [10]. And it has been amplified by the rapid improvement in information technology. Therefore, to deal with information overload and find potential buyers for such niche products, a well-designed recommender system should be integrated into the original business model. A recommender system will filter out only the most relevant products for particular users, which can effectively narrow down the number of total recommendations and make the choice more obvious to customers.

People start to put lots of efforts on this area especially those who heavily rely on e-commerce or provide service through Internet. Despite the widespread use of recommender systems and their gradual emergence as one of the most important roles in today's e-commerce environment, they still face lots of challenges. To cope with those challenges, it has been through lots of improvements over the years, evolving from an early collaborative model to a matrix factorization model as well as the most recent models that utilize deep learning methods. In this dissertation, we will heavily discuss how we can effectively overcome the most common challenge that forced the recommender system to be developed in this way.

1.2 Scope and Research Problem

This project will be of particular interest in studying the overall performance comparison between conventional recommender systems and modern approaches, such as matrix completion and deep learning models using the user-movie utility matrix as input only. In recent studies, most of the cutting edge models are built with complex hybrid approaches that take multiple inputs for predictions. For example, in a modern movie recommender, the keywords that users use in searching particular movies or genres, the viewing time for the movies, and the comments left in the recommender systems can all be used as inputs. Some RSs even categorize the movie using a Convolutional Neural Network to analyse the movie frame by frame. All of the above data are stored and used as input to the hybrid complex models. However, this is not the case for small companies or startups where collecting and storing such databases may not be viable. A simple but effective model which can give reliable predictions with minimal input will be more helpful for such companies.

The project aim to answer the question if models such as matrix completion and deep learning methods can solve the data sparsity issue and the cold start user issue when it comes to using a limited dataset and outperform the conventional methods where only user-movie ratings are given and no additional features nor supporting data are provided and to give model suggestions for the readers accordingly.

1.3 Objectives and Contributions

This project aims to fulfil the following objectives.

- To provide a comprehensive analysis of different recommender systems, including both traditional methods (e.g., content-based filtering, user-based collaborative filtering)

and modern simple approaches (e.g., matrix completion, neural network models) with their theoretical frameworks.

- To explore model variations, for example (ALS and SVT in matrix completion) to identify the best configurations.
- To conduct experiments on different parameters and architectural configurations of recommender systems, aiming to identify key factors and fine-tune the recommender models accordingly.
- To compare the performance of different recommender models across different scenarios, evaluating them based on metrics such as accuracy, precision, recall, F1 Score and RMSE, and suggesting suitable models for readers.

By fulfilling above objections, this project will give the readers a comprehensive review on recommender systems. Additionally, the findings from this study could be particularly useful for small businesses or startups that have difficulty collecting or storing complex data to feed the recommender systems because the study is conducted using limited data particularly.

Chapter 2

Literature Review

2.1 Background Research

This section covered the background research, including the basic understanding of how different types of recommender systems work and the challenges they face. Then, in the related research section, we will introduce the concept of deep learning by explaining how it can possibly fill up the gap between conventional recommender system and their limitation.

2.2 What is a Recommender System

A recommender system is no different from any machine learning model where users pass an input X, and the system returns a prediction Y. The input X can represent the user's ratings or preferences for some products, or it can also be features that describe users (such as age, gender, occupation, among others.) The system then outputs a prediction Y, that is, the suggested products in the database. See Schematic Figure 2.1.

More formally, we can express the recommendation problem as follows: Consider C as the set of all users and S as items that can be potentially recommended (books, movies). U is a utility function that computes the utility of item s to user c, that is to say, u: C x S $\rightarrow$ R, where R is an ordered set (normally, we see it in a non-negative real number within some range, such as rating scores). Next, for every c $\in$ C, the system that computes such item s' $\in$ S that can maximize $u(c, s)$. [6]

$$\forall c \in C, \ \ s'_c = \arg\max_{s \in S} u(c, s) \tag{2.1}$$

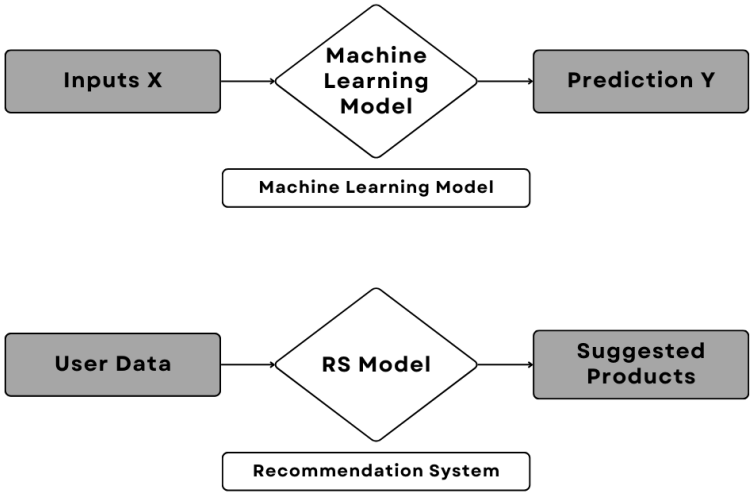


Figure 2.1: A Basic Recommender System

The utility of an item can be generally seen in a rating form. For example, we can form a user-to-movie matrix, which is also known as a utility matrix, and the entries indicate the degree to which the user likes the movie. This matrix is the core data that should be collected in most cases. To explain what a recommendation does most simply is to fill in the missing value in the utility matrix (Figure 2.2).

Figure 2.2: Utility Matrix (image resource: Google image)

Though we usually use ratings for utility, it can be any arbitrary function defined case by case. The user space C can also be defined differently. For example, in a content-based filtering system, we can use age, gender, and language spoken in this space, while we can use UserID only in a collaborative filtering system. The concept applies in item space S as well. As long as the recommender system can correctly approximate a good utility matrix, then making a good recommendation can be a straightforward task by suggesting the top-N rated items that give the highest utility to particular users. In the later subsection, we will discuss different types of recommender and their limitations.

2.2.1 Content-Based Filtering

The recommendation can be traced back to the year 1979, when Rich, Elaine mentioned a system in her study (1979)"*"User modelling via stereotypes"*" [30]. The study created a scenario, as shown below. *"Someone walks into a library and asks for a recommendation on books that are related to China. The reasons that cause one to demand such a book might differ. Therefore, before giving out a good personal recommendation, information about the users should be collected, and the book should be well categorized"* A system named "Grundy" was then developed and used as a book recommender using "Stereotype," a term that was used in the study to describe user characteristics.

Inherited from the concept of the former recommender system, a content-based filtering system started to develop. This system makes recommendations based on the description of the item (movie genre, year of release) and the profile of the user's interests. That is, the utility $u(c, s)$ of item s from user c is calculated by the utility $u(c, s')$ of similar item s'. In our movie recommender system, for example, the system captured the similar features that kept appearing in those highly rated movies by user c in the past and suggested movies with similar features.

Let us make a more concrete example for better understanding. Suppose that a user has rated three movies and all movies are categorized well in a one-hot encoding form (where 1 means the movie contains the feature, and 0 otherwise) as shown in Figure 2.3 ($a, b, c > 0$). We then use the feature matrix to calculate the weighted score of each movie for the user, as shown in Figure 2.4. By normalizing the weighted Matrix, we can obtain the normalized user profile matrix, as shown in Figure 2.5. Finally, the recommender system uses the profile of the user's interest combined with item features to compute utilities for all other movies existing in the database and then recommend movies accordingly.

User Ratings	
Movie	Ratings
s1	a
s2	b
s3	c

Features One-Hot Encoding Matrix				
Movie	Feature 1	Feature 2	Feature 3	Feature 4
s1	0	1	1	0
s2	1	1	1	1
s3	1	0	1	0

Figure 2.3: User Ratings and Item Features Matrix

Weighted Feature Matrix				
Movie	Feature 1	Feature 2	Feature 3	Feature 4
s1	0	a	a	0
s2	b	b	b	b
s3	c	0	c	0

Figure 2.4: Weighted Matrix for 3 Movies

Normalized User Profile				
Movie	Feaure 1	Feature 2	Feature 3	Feature 4
s1	(b+c) / n	(a+b) /n	(a+b+c) /n	(b) /n

Figure 2.5: Normalized Profile of User's Interest($n = 2a + 4b + 2c$)

Challenges:

The Above study provided a great concept of how a recommendation should look like. It also explained how profiles of user's interests and features of items are important inputs and how they affect the final recommendation. One of the advantages of using a content-based filtering system is that it does not rely on much data from another user, which makes it a good system when the customer scale is large but the item is limited. However, there are some drawbacks as well.

Firstly, we believe there is still a chance that the *user profile* might be collected wrongly because the users might not have a good understanding of themselves or they simply do not want to reveal their genuine opinion about their taste, causing them to pass wrong descriptions to the system. From the research done by Rasinski et al. (1999) [26], we know that people might give fake answers when it comes to sensitive questions. For example, a 40-year-old man who loves cartoons might want to keep his liking private from the system when being surveyed. Therefore, the output suggestions might not fit the user's actual demands since the input was initially wrong.

Also, defining the features of the items requires hand-crafted work to some extent. Therefore, the quality of the output might be significantly influenced by how the system categorizes the items. Another issue is that the model can only provide recommendations according to users' existing interests, which means it is hard for the system to help expand the user's interest in the future but stick to similar products all the time.

To overcome the user profile issue, a study done by Van et al. (2000) [34] shows that

we can still update the *user profile* from users' interaction with the items. For example, viewing time can be a great data representing the interaction. The system can use those data to reconstruct user profiles instead of directly asking user questions. That is a reason why collaborative filtering techniques and other hybrid models, such as a neural network system, can be helpful because they also use another user to help give out recommendations, which helps expand the interest of all users and can help reduce the need for hand-craft feature design.

In this dissertation, we want to explore modern recommender systems more. As a system based on collaborative filtering and matrix factorization, not only does it focus on interactions between users and items, but also be able to capture hidden relationships or patterns. We expect it to be helpful to provide solutions to the drawbacks of the content-based filtering system.

2.2.2 Collaborative Filtering Systems - User Based

At a similar stage, the Collaborative Filtering Systems was introduced. The term "*Collaborative Filtering*" was introduced in the study by Goldberg et al. (1992) [11] in their work and then taken up by Amazon (2003) for further research and development. [22]. MIT Media-Lab also published a study that uses collaborative filtering techniques to build up personalized recommendations from any arbitrary database to a user based on similarities between the interest profile of that user and those of other users. [33]

They build up the model based on the assumption that similar people love similar products. Therefore, Collaborative Filtering recommender systems predict a specific user's utility for an item (movie) based on the ratings given by others. This method has been widely used in academic or industrial areas, such as GroupLens [20, 29], Spotify, Video Recommender [15], and a lot more. A movie recommender system built with a collaborative filtering model no longer focuses on the profile of items that are mentioned in content-based filtering but rather on the similarities between users.

To formally describe the above task, we will again use the utility function. The utility u(user c, movie s) is based on the utilities $u(c, s)$. And $u(c', s)$ is decided by the interactions between user $c' \in C$, who is considered similar to user c. So, a movie recommender system using collaborative filtering techniques starts by calculating the similarities between user c and $c' \in C$. The system then can recommend the movies that are most liked by the filtered similar users to user c [1].

CF: Memory-Based Methods

According to the studies from Breese et al. (2013) and Delgado et al. (1999) [5, 9], we can group the collaborative filtering method into two different groups. One is Memory-Based methods, and Model-Based methods are the other.

In memory-based methods, the predictions are made entirely according to the previous utility matrix. That is to say, the unknown value $r_{c,s}$ for user c to movie s is decided by ratings given to movie s from the N most similar users.

$$r_{c,s} = \text{aggr } r_{c',s} \quad c' \in \hat{C}, \text{ where } \hat{C} \text{ is the set of N similar users to c.} \tag{2.2}$$

The aggregate function can be relatively simple; for example, we can use mean value (2.3) of $r_{c',s}$ where $c' \in \hat{C}$. However, this function did not consider the different similarities among each c'. Hence, another aggregate function (2.4) is introduced where we add k as a normalized parameter and similarities(sim function) as a weight parameter to compute the weighted average. It is evident that the higher the similarity between c' and c, the stronger the effect of $r_{c',s}$ on $r_{c,s}$. In most studies, we usually see aggregate function (2.4) implemented into the system.

$$r_{c,s} = \frac{1}{N} \sum_{c' \in \hat{C}} r_{c',s} \tag{2.3}$$

$$r_{c,s} = k \sum_{c' \in \hat{C}} \text{sim}(c, c') \times r_{c',s} \tag{2.4}$$

There are various methods to calculate the similarity $sim(c, c')$ between user c and c' in the system. However, the performance depends on the scenario and the data, according to the study. [19] Two popular methods are commonly used in collaborative filtering systems. One uses *Pearson Correlation Coefficient*(2.5), and the other uses *Cosine Similarity*(2.6).

$$\text{sim}(c, c') = \frac{\sum_{s \in S_{cc'}} (r_{c,s} - \bar{r}_c)(r_{c',s} - \bar{r}'_c)}{\sqrt{\sum_{s \in S_{cc'}} (r_{c,s} - \bar{r}_c)^2} \sqrt{\sum_{s \in S_{cc'}} (r_{c',s} - \bar{r}'_c)^2}} \tag{2.5}$$

$$\text{sim}(c, c') = \cos(\vec{c}, \vec{c'}) = \frac{\vec{c} \cdot \vec{c'}}{\|\vec{c}\|_2 \times \|\vec{c'}\|_2} = \frac{\sum_{s \in S_{cc'}} r_{c,s} r_{c',s}}{\sqrt{\sum_{s \in S_{cc'}} r_{c,s}^2} \sqrt{\sum_{s \in S_{cc'}} r_{c',s}^2}} \tag{2.6}$$

$r_{c,s}$ and $r_{c',s}$ mean the ratings that user c and c' give to particularly movie s respectively.
$\bar{r_c}$ and $\bar{r'_c}$ mean the average ratings that user c and c' give to all rated movies.
Finally, $S_{cc'}$ is the movie set that both users give ratings to.

CF: Model-Based Methods

Model-based methods use various techniques, ranging from pure statistics [16] to matrix factorization to the latest machine learning [14, 28]. Unlike memory-based methods, it uses historical data to build a prediction model. Whenever we want to get a prediction, we just input the data into the model without recalculating everything from the beginning. Once the model is trained, the runtime will outperform the memory-based methods, especially on an extensive database.

In memory-based methods, whenever we want to get a recommendation for a particular user, we have to calculate the similarities again and again. The process might be fast in a small dataset; however, when this system encounters enormous data that most companies face nowadays, it takes much more time to return outputs. A common practice is to assume that the relationship will not change enormously in a short period, so we can use the pre-calculated similarity during this period. [4]

Challenges: Unlike content-based filtering systems, collaborative filtering can be more flexible since the system only requires feedback without building up complicated user profiles or the features of the items. Also, the algorithm uses the similarity among users to suggest items, so it is more likely that the user will be recommended something he might like but never be exposed to, which would be helpful for the user to expand interest.

Even though collaborative filtering has its strengths, it has some downsides as well. The most common one that everyone brings up is the sparsity issue. The system uses the similarities to compute the utility. However, it can be hard to find common items that were rated by both users when the utility matrix is too sparse. "Cold start," a problem that occurs when new items or new users are added to the system, is another possible drawback. Since the new item has never been rated, it is likely not to be recommended to any users in the system despite its potential. For users who have never rated any movie, it is also hard for the recommender system to provide recommendations. We believe that is why, in common practice, lots of movie streaming platforms isolate a section called "New Movie" to avoid such cold start problems and require new users to give at least three movies that they

like when they first join the recommender system.

In this dissertation, we aim to find other solutions, such as matrix factorization or artificial neural networks in our recommender system. Matrix factorization can be seen as a model that uses hidden features that interact with users and items, whereas a neural network collaborative filtering system can be considered as a non-linear model that is good at capturing non-linear relationships. Both of them can complement the two conventional systems' strengths and weaknesses and capture more hidden features and non-linear relationships. Therefore, we believe they can be a powerful tool to conquer sparsity and cold start problems.

2.2.3 Matrix Factorization and SVD

Before entering the world of neural networks, we must introduce literature about matrix factorization done by Koren, Yehuda, Robert Bell, and Chris Volinsky (2009) [21], a concept that can change the way we view the conventional recommender system design. The basic explanation for matrix factorization is that we have a sparse utility matrix X_{mn} where m denotes the number of users and m is the number of items (movies). We aim to factor the utility matrix into two matrices, U and I, to build up a model that can estimate unknown values.

We then need some techniques to obtain an approximate $\hat{x}$ when not all values are known in the original X, as shown in Equation (2.7).

$$X_{mn} \approx U_{mk} \times I_{nk}^{T} = \hat{X} \tag{2.7}$$

X_{mn} denote the Utility Matrix in the database with the size $m \times n$
$\hat{X}$ means the approximation of X_{mn}
U_{mk} and I_{nk} mean User matrix with size $m \times k$ and Movie Matrix with size $n \times k$ respectively.

According to the above, we know each user, m, is related to a vector $q_m \in \mathbb{R}^k$, and each movie n is related to a vector $p_n \in \mathbb{R}^k$. And it is clear that for a particular movie and a particular user, the joint rating can be calculated by the dot product between user and movie vectors, that is, $\hat{r}_{mn} = p_n^T q_m$. We here use $\hat{r}$ to show that the rating we calculated is an approximation to the actual r. There are some approaches that we can use to obtain a good estimation. The most intuitive one is the well-known *singular value decomposition*

(SVD).

However, there is a critical issue that needs to be solved, which is the high portion of missing values in the original sparse utility matrix. As we all know, a conventional SVD is not defined when we have an incomplete matrix. Note that in the original sparse matrix, the missing value was *NULL* instead of *zero*. So, for a long time, people used zeros to fill in the missing blank. However, we argue that this method is questionable to some extent. The main reason is that the missing value in r_{ui} simply means the user has not yet seen the movie, but it does not necessarily mean the user dislikes the movie. We will talk more about how we fill the null value in our experiment in the methodology section.

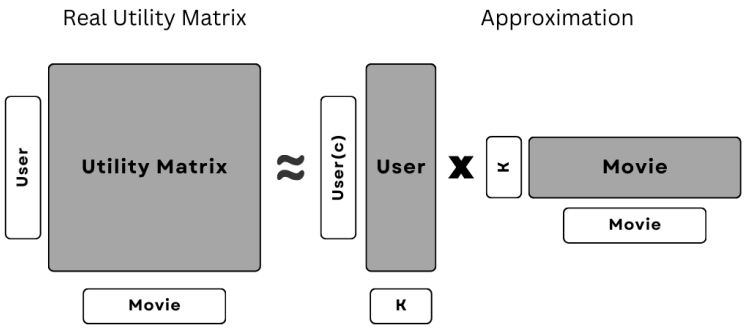


Figure 2.6: Matrix Factorization: The Concept

2.3 Related Works

2.3.1 TF-IDF

In the content-based filtering system, which is shown in Figure 2.3 to Figure 2.5, we demonstrated the most basic way to construct the profile of the user interest matrix. We encoded the movie genre features with a one-hot encoding method. However, there is a widely used algorithm called "term frequency/inverse document frequency," also known as (TF-IDF) which can be a better way to create movie feature vectors.

This concept was introduced early in the study by Salton et al. (1988). [31] and was

widely used in the natural language processing area to measure how important words are in a document. To compute the TF-IDF, we first compute the TF, the frequency of keyword k_i in document d_j, with the following Equation 2.8 where $f_{i,j}$ represents the number of times that word k_i appears in document d_j. The maximum is computed with frequencies $f_{z,j}$ of all words k_z that are present in the document d_j.

$$TF_{i,j} = \frac{f_{i,j}}{\max_z f_{z,j}} \tag{2.8}$$

Words that appear in several documents could be more effective in distinguishing between relevant and non-relevant ones. Hence, the inverse document frequency (IDF), shown in the Equation below, is usually combined with term frequency (TF). N denotes the total number of documents, and word k_i shows in n_i of these documents.

$$IDF_i = \log \frac{N}{n_i} \tag{2.9}$$

Finally, we can calculate the TF-IDF weight as below.

$$w_{i,j} = TF_{i,j} \times IDF_i \tag{2.10}$$

2.3.2 Neural Collaborative Filtering

In the research from He et al. (2017) [14] and Rawat et al. (2020) [27], the Multi-Layer Perceptron Model (MLP) was introduced and implemented into a recommender system. An MLP is a branch of artificial neural networks, a technique that is widely used in today's machine learning field. The basic idea comes from mimicking how neurons in the human brain work. The MLP is constructed by multiple layers containing input layers, where we insert the data; an output layer, where the system makes the predictions; and finally, multiple hidden layers, which represent the hidden relationship between the inputs and the outputs.

The magical and the most powerful parts of an MLP network lie in the hidden layers. We often use a combination of non-linear activation functions among layers, such as ReLU and its variations, Sigmoid and or Tanh to construct the architecture of neural networks. By using non-linear activation functions with deep layers of architecture, the model can now capture complex and non-linear user-item interaction patterns. Conventional recommender systems mentioned in the above sections, such as matrix factorization, even a more advanced system than the conventional content-based filtering and collaborative filtering, are still heavily

restricted by the assumption of linearity, which can sometimes be oversimplified and limit their performance to some extent.

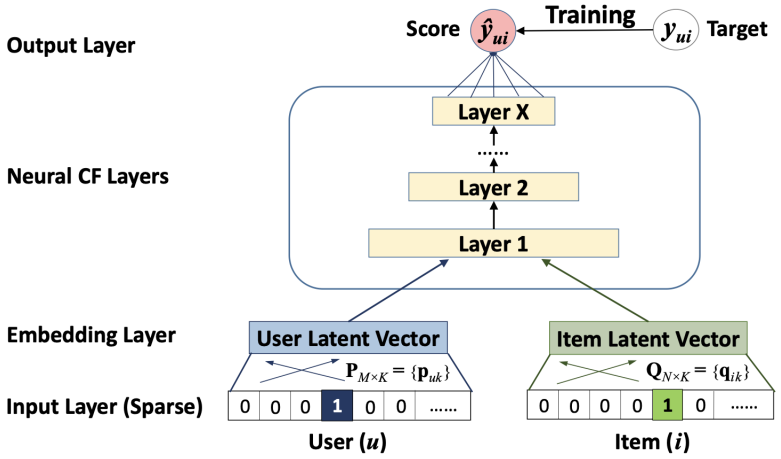


Figure 2.7: Collaborative Filtering Model using Neural Network, source: He 2017

2.3.3 AutoRec

The research from Sedhain (2015) proposed a model called AutoRec, an autoencoder neural networks model that is used in recommender systems [32]. In a rating-based collaborative filtering task, where we have a utility matrix R with size $m \times n$. m and n denote the number of users and movies respectively: $R \in \mathbb{R}^{m \times n}$.

For every user $u \in U = \{1, \ldots, m\}$, we can use a vector to represent its rating for all movies. $\mathbf{r}^{(u)} = (R_{u1}, \ldots, R_{un}) \in \mathbb{R}^n$. Also, a vector $\mathbf{r}^{(i)} = (R_{1i}, \ldots, R_{mi}) \in \mathbb{R}^m$ can be used to represent the movie ratings from all users for movie $i \in I = \{1, \ldots, n\}$. The task for the research is to build an autoencoder model that can take the $\mathbf{r}^{(i)}$ and $\mathbf{r}^{(u)}$, project it into a hidden space and use the autoencoder to reconstruct $\mathbf{r}^{(i)}$ and $\mathbf{r}^{(u)}$ back to the output layer to predict the missing value existing in original datasets. The task for the autoencoder is to solve the following Equation: 2.11.

$$\min_{\theta} \sum_{\mathbf{r} \in \mathbf{S}} \|\mathbf{r} - h(\mathbf{r}; \theta)\|_2^2 \tag{2.11}$$

In theory, we can either use item-based or user-based autoencoder. However, it turned out the item based performed better in the research. Therefore, in this project, we will be comparing these two approaches with other models as well.

Chapter 3

Methodology

Figure 3.1 illustrates the workflow of this project. The green box involved data prepossessing, which will be discussed in section 3.2. For different models, different data prepossessing should be done to be implemented into different algorithms. For example, we need to fill in the missing value to perform an SVT for matrix completion.

The blue area was about building up different recommender systems and using them to output predictions for movie ratings. The architecture and the algorithm behind each model will be discussed in section 3.3. A total of seven recommender systems will be discussed in this project. Two of them represent conventional RS, two using Matrix completion, and three using neural networks.

Finally, this project evaluated the result with metrics that are commonly used in recommendation systems illustrated in the yellow area. The aim is to evaluate the performance difference between conventional recommender systems and those built with neural networks. The metrics will be discussed and explained in section 3.4. In this project, we manipulated one particular user data into two different data. One with 30 % rated data being removed, which we use to represent a normal user entering the recommender system, and another having 100 % rated data being removed to mimic the scenario of a cold start user.

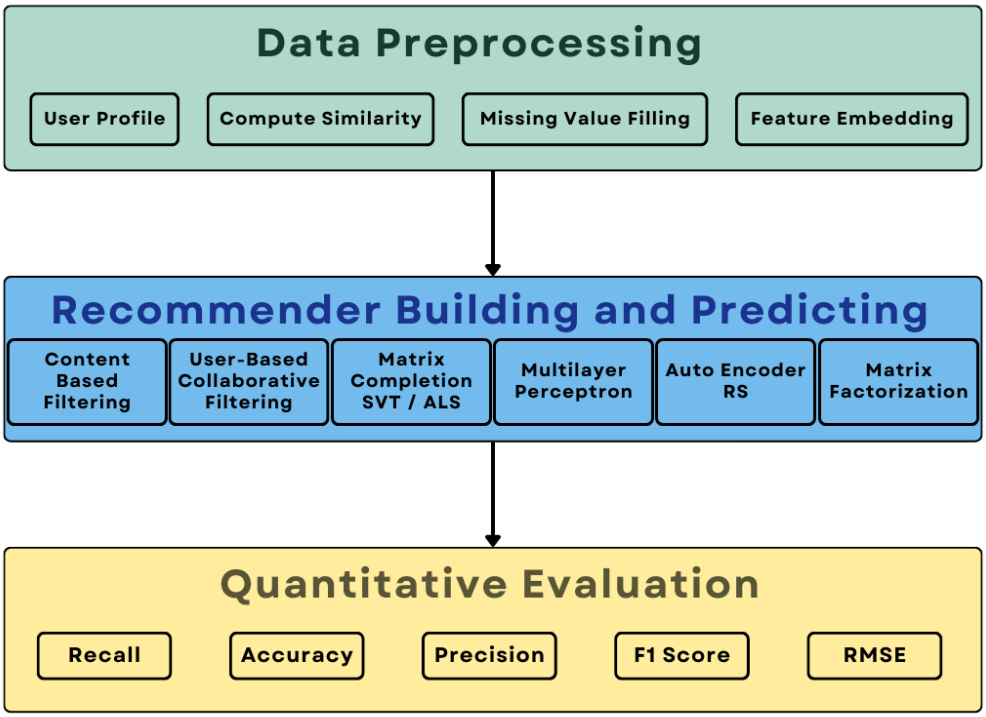


Figure 3.1: Illustration of the methodology and the workflow of this project

3.1 Data Introduction

The data used in this project is from the classic MovieLens 20M dataset, which contains a total of 20,000,263 ratings that were rated by 138,493 users to 27,278 movies with a utility matrix density of 0.5%. The sparsity of the utility matrix accurately reflects reality because, in reality, the number of movies in any database should be higher than the number of rated movies for any arbitrary user. The dataset is publicly provided by the University of Minnesota or the GroupLens Research Group. Because of its scale, authenticity, and diversity, it became one of the most popular datasets when it comes to research in recommender systems. [12]

3.2 Data Preprocessing

There are some data preprocessing procedures listed below that need to be done so that the recommender system can perform normally.

1. Inactive User Filtering

To ensure the original data only consists of active users so that the recommender systems can precisely show the performance when we test the cold start user in the later experiments, we implemented a mask that can filter out users that have less than five movie ratings.

2. Error Value Filtering
   In this dataset, the ratings in the datasets should be within a range from 1 to 5. It is crucial that we filter out ratings higher than five or lower than one so that the recommender system will not be influenced by those error data.

3. Fill in Missing Values
   Some recommender systems, such as matrix completion with Singular Value Thresholding (SVT), can only be done when the matrix contains no missing values. Therefore, before performing this algorithm, missing values should be filled. However, this is not necessary for some other algorithms where we can ignore the missing value and use the existing rating to get predictions.

4. Construct Sparse Matrix from Raw Data
   We use a sparse matrix to handle the utility matrix from raw data, because using numpy array from pandas will lower the memory efficiency as well as the computation efficiency.

5. Data Simulation for Special Scenarios
   In this project, we selected a user and treated the data as the ground truth. We then simulated two scenarios: in the first one, we removed 30% of the user's rated data to simulate a normal user joining the recommender systems, and the second scenario involves removing all ratings to simulate a cold-start user where the recommender system has zero information about his liking. These two data will then be considered as two virtual users and be fed into the recommender system to predict movie ratings. Finally, we then compared the result against the ground truth to evaluate the performance among different recommender systems. See Figure 3.2.

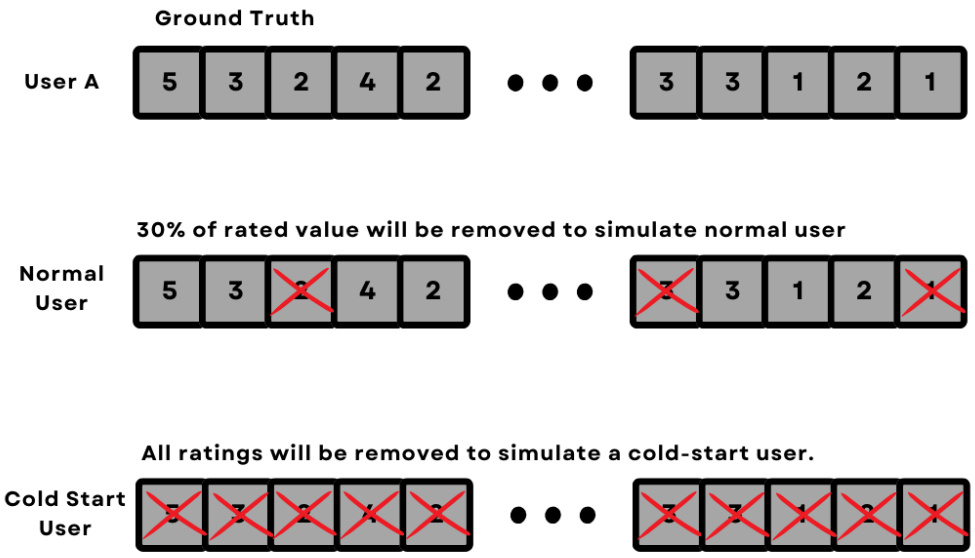


Figure 3.2: Data Simulation for Special Scenarios

3.3 Recommender Systems Built in the Project

In this section, we will introduce all the recommender systems used in this project. There are two conventional recommender systems: Content-Based Filtering RS and User-Based Collaborative Filtering RS. For comparison, five other recommender systems will also be introduced, two of which are built using Matrix Completion (SVT and ALS) and three with neural networks, a Multi-Layer Perceptron RS, an Autoencoder RS and a Matrix Factorization RS, respectively.

3.3.1 Content-Based Filtering

In the content-based filtering recommender systems, we used the movie feature to build up the profile of the user's interest. TF-IDF algorithm was used to transform movie features, which are in text form, into vector form. Let us make an easy example to illustrate the idea.

Assuming there are three movies to be seen by a particular user (Table 3.1), we then can calculate, for each movie, the TF-IDF weight in each genre (Table 3.2).

movieId	genres
1	Action, Adventure
2	Action, Comedy
3	Comedy, Drama

Table 3.1: Movie IDs and Genres

	Action	Adventure	Comedy	Drama	Ratings
Movie 1	0.5	0.5	0	0	4
Movie 2	0.5	0	0.5	0	3
Movie 3	0	0	0.5	0.5	5

Table 3.2: TF-IDF Feature Matrix for Movies with Ratings

By combining the given ratings, also shown in Table 3.2, we can calculate the profile of user interest by normalizing the summation of the score for each genre, as shown in Table 3.3.

Action	Adventure	Comedy	Drama
0.2917	0.1667	0.3333	0.2083

Table 3.3: Profile of User's Interest

To be noted, when we calculate the Profile of User's Interest, we only calculate with rated data. In a utility matrix, a missing value does not mean the user dislikes the movie. It simply means that the movie is still unseen or unrated by the user. So, it is crucial that we remove this part of the data from the recommender system to compute the profile of the user's interest correctly. Once the profile of the user's interest is established(in a vector form denoted V_u for user $u \in U$), then computing the prediction can be simple as $Prediction = V_u \cdot V_i$, where V_i denotes the TF-IDF vector for movie $i \in I$.

This model is rather simple and does not need other supporting data, for example, other user ratings. However, this characteristic became a major drawback when it comes to cold start users. Without other supporting data, by only receiving empty data from the cold start user, it is not possible for the RS to calculate the profile of the user's interest. Therefore, in this project, we used the average global user profile to replace the missing piece.

3.3.2 Collaborative Filtering - User Based

The collaborative filtering used in this project is based on user similarity. The solution relies heavily on the assumption that similar people will love similar items. In the algorithm,

we first compute the user similarity using Cosine Similarity = $\text{sim}(c, c')$ among different users, where $(c, c') \in C$ (a set of all users). If we have a utility matrix with shape $m \times n$, where m represents the number of users and n represents the number of movies, we can define a vector $c \in \mathbb{R}^n$ which contains all movie ratings rated by user c, and a vector $c' \in \mathbb{R}^n$ contains all movie ratings rated by user c'. Non-rated movies are considered to have a rating of zero at this point.

Therefore, we can see the user ratings as a vector existing in a n dimensional space, and the similarity between users can be computed by cosθ value, where θ is the angle between 2 vectors c, c' as shown in equation 3.1.

$$\text{sim}(c, c') = \cos(\vec{c}, \vec{c'}) = \frac{\vec{c} \cdot \vec{c'}}{\|\vec{c}\|_2 \times \|\vec{c'}\|_2} \tag{3.1}$$

After all the similarities among selected user c and other $c' \in C$ are computed, we adopted a concept from the K nearest neighbours (KNN) algorithm. We sorted and selected the top K nearest neighbours with the highest similarity scores. For all movies $\in S$, we predicted the output score by computing the average score provided by the K nearest neighbours. If this particular movie s has not yet been rated by nearest neighbours, then we used the average movie ratings between all users $r_{\bar{C},s}$. For movies that are not yet rated by any of the users, a cold start item in RS, we gave a default score in the project.

As for cold start user prediction where the user has zero ratings for all movies, there is no better way to calculate the cosine similarities since the vector of this cold start users will be an all zero vectors $[0, 0, \ldots 0, 0]$ under this content-based model. So, we tried to use the average movie ratings between all users to fill in the missing piece.

Other Method to Compute Similarities

In a user-based Collaborative Filtering RS, we tried to use the cosine similarities to select the K nearest neighbours. However, some adjustments can be made. For example, initially, we considered putting the user rating vector c in an n-dimensional space, $c \in \mathbb{R}^n$ where n = numbers of all movies . We might ignore the fact that some movies are not yet seen by the users, which means that even if both of the users have not seen the movie yet, it does not necessarily mean that they will rate this particular movie similarly after watching it. However, because of using zero for unrated ratings, the computed similarities can be biased.

Other approaches are used to adjust the way we compute the similarities. One of them considers computing the cosine similarities based on commonly rated movies only. Considering we have a movie set $S_{cc'}$ denote the movie set that both users give ratings to, we only compute the $\text{sim}(c, c')$ that is based on it. Also, we can set up a parameter to adjust the minimum common movies between users. This method can better avoid the problem caused by unrated movies and is assumed to bring better overall performance.

Another approach is to use the Pearson Correlation Coefficient, as shown in equation 3.2. The correlation coefficient can be only calculated based on commonly rated items initially. Therefore, the unrated movies do not influence the similarities calculated by this method.

$$\text{sim}(c, c') = \frac{\sum_{s \in S_{cc'}} (r_{c,s} - \bar{r}_c)(r_{c',s} - \bar{r}'_c)}{\sqrt{\sum_{s \in S_{cc'}} (r_{c,s} - \bar{r}_c)^2} \sqrt{\sum_{s \in S_{cc'}} (r_{c',s} - \bar{r}'_c)^2}} \tag{3.2}$$

Even though we used different similarities to predict the movie ratings for normal users, the solution in this model for the cold start user problem remained the same. Since we do not have the rating data from the cold start user, we can only compute the similarities by either replacing the empty value with a global mean or another selected value. This is another reason why we wanted to build other models that we hope can perform better when it comes to cold start user issues.

3.3.3 Matrix Completion

This is another approach that can be implemented into a recommender system and can shift the way we see it. The approach is to see the rating prediction problem as a matrix completion task where the missing value of the matrix can be seen as noise, and the target is to denoise the given utility matrix and try to find the best approximation of it.

Singular Value Thresholding (SVT)

One of the methods we used in the project was Singular Value Thresholding (SVT). The concept is very similar to a matrix factorization process [21], which is also a commonly used method in the recommender system area. Assuming we have a sparse utility matrix M with shape $D \times N$ and the task is to find a lower rank matrix X such that both X and M have the same projection on Ω. Omega is the observed indices in M, as shown in equation 3.3.

$$\min_X \tau\|X\|_* + \frac{1}{2}\|X\|_F^2 \quad \text{subject to} \quad \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \tag{3.3}$$

$$\mathcal{P}_\Omega : \mathbb{R}^{D\times N} \to \mathbb{R}^{D\times N} = \text{Projection on } \Omega \tag{3.4}$$

$$\mathcal{P}_\Omega(X) = \begin{cases} X_{ij} & \text{if } (i,j) \in \Omega, \\ 0 & \text{if } (i,j) \notin \Omega. \end{cases} \tag{3.5}$$

The solution can be done by following steps.

- Initialize: $\mathbf{Y} = 0(D \times N \text{ zero matrix})$, τ a thresholding parameter, β a step parameter
- Calculate: $\mathbf{X}^{(k+1)} = D_\tau(\mathbf{Y}^{(k)})$, where $\mathcal{D}_\tau$ denotes the singular thresholding operator
- Update: $\mathbf{Y}^{(k+1)} = \mathbf{Y}^{(k)} + \beta\mathcal{P}_\Omega(\mathbf{M} - \mathbf{X}^{(k+1)})$
- Iteration Stops after k steps when: $\frac{\|\mathcal{P}_\Omega(\mathbf{M}) - \mathcal{P}_\Omega(\mathbf{X}^{(k)})\|_F}{\|\mathcal{P}_\Omega(\mathbf{M})\|_F} \leq$ tolerance

This method relies on using singular value decomposition with singular value thresholding to shrinkage the singular value to obtain a lower rank matrix approximation. This method is helpful to capture the hidden patterns existing in our utility matrix and use the pattern to get the possible prediction on movie ratings. But, there is a crucial step that needs to be finished before performing SVD. That is, the missing values in the original utility matrix should be filled.

A common practice that was widely used in the past was filling the missing value with all zeros. However, this concept is again biased since the missing value in the matrix only means that the movie is not yet rated, but it does not necessarily mean that it is disliked. If we use all zero to fill in the super low-density utility matrix, the approximated matrix will be highly influenced by those values, which will lead to bad predictions. In this model, we filled the missing value for a particular user with a Gaussian distribution around the user's average movie ratings, naturally assuming for such extensive movie data, the distribution of the ratings will be approximately close to a Gaussian distribution.

Alternating Least Squares (ALS)

This is another matrix completion algorithm that can be helpful when it comes to doing a matrix completion task on a sparse matrix. The ALS algorithm also uses the concept of matrix factorization so that it can effectively handle sparse matrices. The core concept for

ALS is to decompose the utility matrix R into two factors, U and V, such that $R \approx UV^T$. Since this method is used in the context of recommender systems, the matrices U and V can be called the user factor and item factor matrix, respectively. u_i denotes the ith column of the user factor matrix, and v_i denotes the ith column of the item factor matrix. The ratings for user i to movie j denote R_{ij}. The target of ALS is to minimize equation 3.6, shown below.

$$\arg\min_{U,V} \sum_{(i,j)\in\Omega} (R_{ij} - u_i^T v_j)^2 + \lambda(\|U\|_F^2 + \|V\|_F^2) \tag{3.6}$$

- U the User Factor Matrix $(m \times K)$
- V the Movie Factor Matrix $(n \times K)$
- K numbers of hidden features
- Ω the observed rating set
- λ the regularization factor
- $\|\cdot\|_F$ the Frobenius norm

The solution can be obtained by alternately fixing one of the matrices U or V and updating the other one. The update rules are derived by taking the derivative of the function shown in Equation 3.6 with respect to U_i and V_j and setting them to zero. The best part of the ALS is that it can handle the missing value naturally because when we update U_i and V_j, we only consider the observed value and ignore the missing ones. The algorithm will converge when the observed error is below our assigned threshold.

3.3.4 Multi-Layer Perceptron RS

A Multi-Layer Perceptron Neural Network (MLP NN) is a feed-forward artificial neural network. It is one of the most basic models built with an input layer, multiple hidden layers, and an output layer. All layers are fully connected with the neighbouring layers. For neurons in layer $L+1$, each of them receives an input of $f(z)$ where $f(.)$ is an activation that introduced non-linearity, enabling the network to learn complex patterns and $z = W \times x + b$ is a linear combination with different weights and bias from previous layer L. By evaluating the standard MSE loss between the prediction value and the validation value, we can train our model with the optimal weight and bias via backpropagation.

The architecture that we used in the project can be illustrated in Figure 3.3. We adopt the concept also from the utility matrix factorization and the study from He et al. (2017) [14].

For every user i and movie j, we projected them into a k dimensional latent space $k = 64$ similar to the latent factors introduced in the conventional matrix factorization method. We concatenated the user latent vector with the movie latent vector into an embedded vector, which was then used as an input layer in our MLP-NN. Therefore, the input layer will contain a total of $k \times 2$ neurons. The purpose of using feature embedding with an MLP-NN is that we do not have other features given by the original datasets, only the utility matrix. By assuming there are hidden non-linear relationships existing in the interactions between users and movies, we use embedding techniques combined with an MLP-NN to capture them.

In the architecture, we used five hidden layers with decreasing neurons in each layer for the basic model. By gradually reducing the number of neurons as the layers get deeper, we want our MLP-NN to compress the information further. That is to say, the network extracts and condenses features and obtains the most useful information before inserting the result into our output layers. By doing so, we can reduce the chance of over-fitting. By getting more general features, we can thereby improve the generalization ability of our model. We also use dropout in between different hidden layers with a rate of 0.5 and ℓ_2 regularization $\lambda = 0.01$.

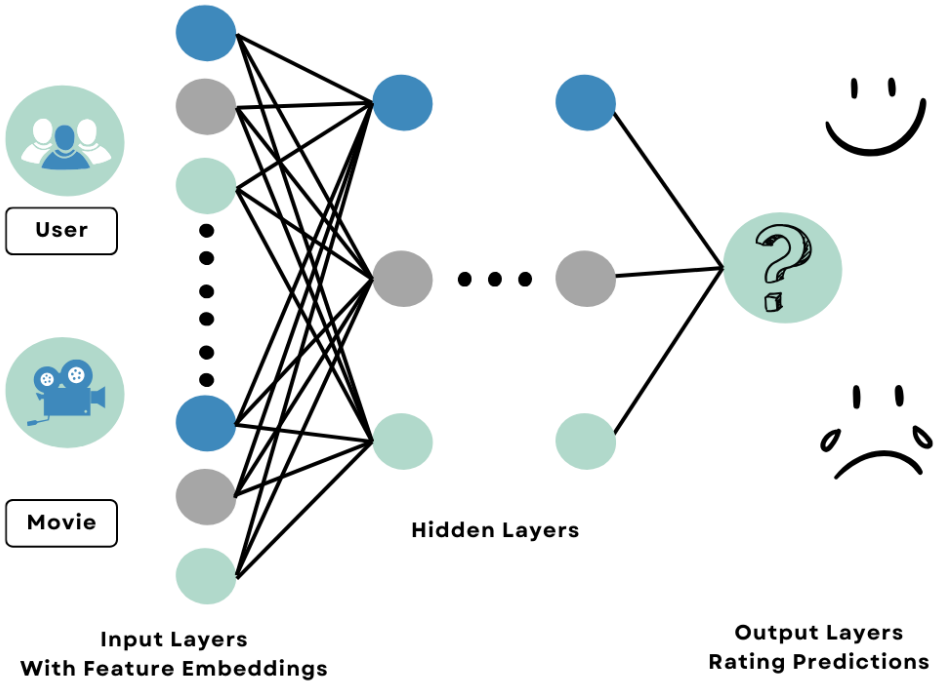


Figure 3.3: Illustration of a Multi-Layer Perceptron Neural Network

3.3.5 Autoencoder RS

An autoencoder is a neural network that is used to encode the input into some latent space and can be reconstructed from such representation into the original space. A classic autoencoder is built upon three types of layers, just like other common neural networks: an input layer, multiple hidden layers that construct an encoder, a decoder, a hidden space that represents the learned features, and an output layer. During the learning phase, the system trains the encoder part and decoder part. The encoder aims to find a mapping that maps the input layer to the hidden layer, while the decoder tries to reconstruct an output from the hidden layer, as shown in the Figure 3.4. [3]

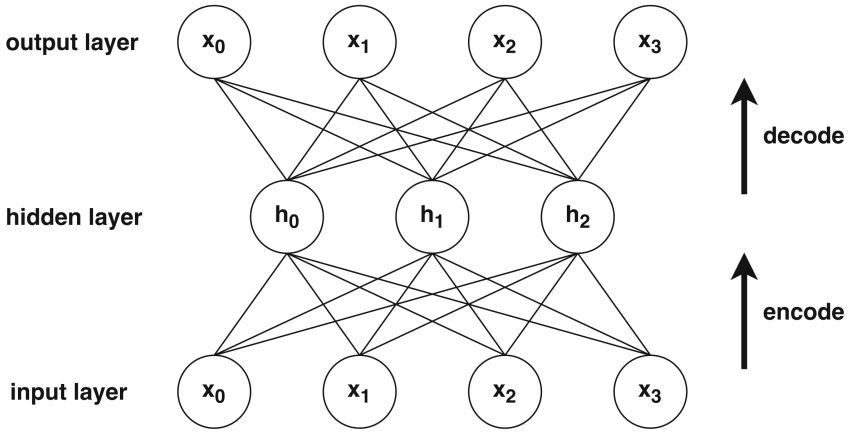


Figure 3.4: Illustration for an Autoencoder, source: Batmaz 2019

As we want to compare the performance between different neural networks, in the architecture of our autoencoder, we set the numbers of neurons in the hidden space to be the same as the embedding dimension in our MLP model. To increase the feature-extracting ability and have a better generalization, the encoder and the decoder were each built with multiple hidden layers with a symmetric architecture and a hidden space in between. The hidden space will receive the output from the encoder and transfer it to the decoder. Then, the loss will be calculated by the difference between the input data and the output from the decoder.

An autoencoder can also be trained in two different ways; the concept is similar to the conventional content-based and user-based models. We can choose the autoencoder to learn how similar users rate the movie or how similar movies are rated by changing the input of the RS to be a user-movie vector or a movie-user vector. Even though they share the same architecture, when we change the input vector, the autoencoder actually learns(extracts

features to latent space) differently and can possibly lead to different predictions. In this project, we will build both models and compare the results after experiments.

3.3.6 Matrix Factorization Networks

After trying to build recommender systems with MLP and autoencoder, we started to question how the performance would compare if we returned back to the basic idea of a recommender system using pure matrix factorization, where predictions are made by the dot product between two latent vectors (representing the hidden relationship between users and movies). And how closely the performance of this approach would be compared to an MLP model with complex architectures.

The most common combination of two latent vectors is the dot product shown below in Equation 3.7, where p and q can represent the user and movie latent vector.

$$\phi^{\text{dot}}(\mathbf{p}, \mathbf{q}) := \langle \mathbf{p}, \mathbf{q} \rangle = \mathbf{p}^T \mathbf{q} = \sum_{i=1}^{d} p_i q_i \tag{3.7}$$

To improve the overall performance for a simple matrix factorization, a common trick is to add biases for global bias, user bias, and item bias, as shown below in equation 3.8. b denotes the global bias and p_1 and q_1 denote the user bias and item bias respectively. In a matrix factorization model in a recommender system, we sometimes encounter situations where some users give higher scores despite their liking, and similarly, an item might be rated badly somehow. To overcome this issue, the user bias, item bias, and global bias terms are introduced in the model. This approach was also found effective in the study done by Rendle et al. (2020) [28].

$$\phi^{\text{dot}}(\mathbf{p}, \mathbf{q}) := b + p_1 + q_1 + \langle \mathbf{p}_{[2,\ldots,d]}, \mathbf{q}_{[2,\ldots,d]} \rangle \tag{3.8}$$

By combining the above concepts, the architecture of our matrix factorization neural networks is built. We first set up embeddings for users and movies. At the same time, we also initialized the global bias, item bias, user bias, and weights. The predictions are computed based on the dot products from the latent space plus biases. We use a weighted MSE to compute the loss, and we set up a threshold of 3.5, see equation 3.9 and 3.10. The reason is that we want our model to be more sensitive toward the high-rated items.

$$L = \frac{1}{N} \sum_{i} w_i (\hat{r}_i - r_i)^2 \tag{3.9}$$

$$w_i = \begin{cases} 1.2 & \text{if } (r_i) > 3.5 \\ 1 & \text{otherwise} \end{cases} \tag{3.10}$$

After the loss is computed, the model uses the backpropagation approach to update the weights and biases. By splitting the training data into the validation dataset and training dataset, we can perform an early stopping to minimize our training loss. The illustration of the architecture is shown below in Figure 3.5.

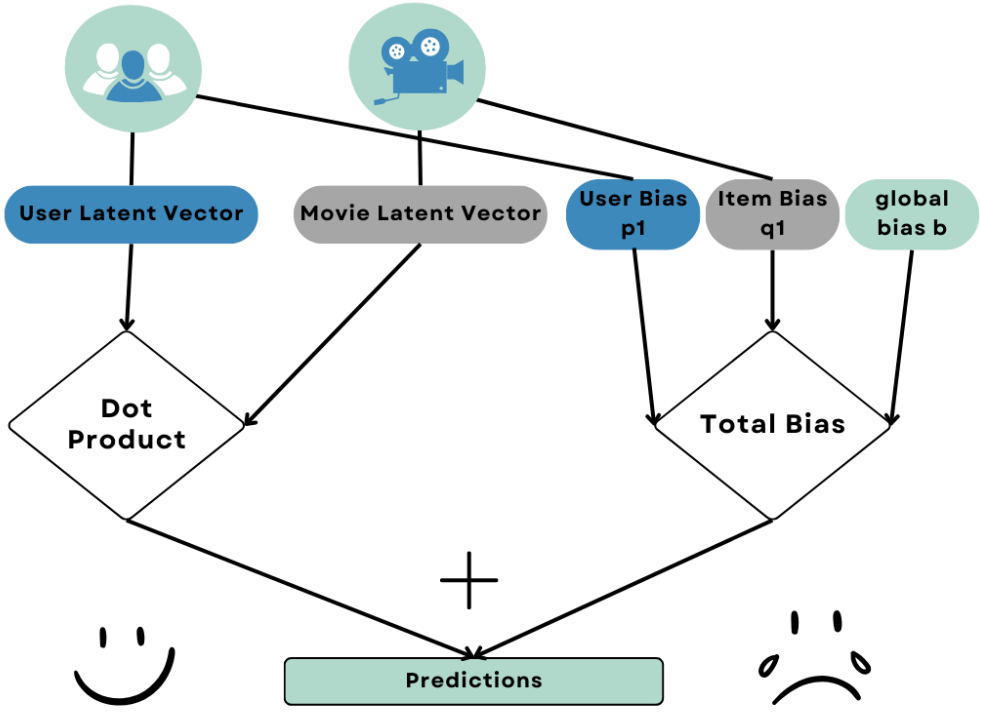


Figure 3.5: Illustration for a Matrix Factorization Model with Bias

3.4 Quantitative Evaluation

Concerning the evaluation method, there were several metrics that we used to evaluate and compare the models in the project: accuracy, precision, recall, F1 Score, and RMSE, respectively. To evaluate a recommender system, we normally try to group the predicted score into two categories with a threshold k. (In this project, k is set to be 3.5, a typical threshold value). When the predicted score is higher than k, we put this item into the "Liked group" and others into the "Disliked group." By doing so, we can create a confusion matrix (Table 3.4) that can be used to calculate the accuracy, precision, recall, and F1 score.

Table 3.4: Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

- TP True Positives: Numbers of predicted Liked and Liked in the ground truth
- TN True Negatives: Numbers of predicted Disliked and Disliked in ground truth
- FP False Positives: Numbers of predicted Liked but actually Disliked in ground truth
- FN False Negatives: Numbers of predicted Disliked but actually Liked in ground truth

3.4.1 Accuracy

Accuracy measures the overall successful prediction rate. We compare the prediction with the ground truth and compute the accuracy using the equation below.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.11}$$

3.4.2 Precision

Precision measures the proportion of instances that are actually positive among those that were predicted as positive. The metric reflects how well the model predicts the Liked movies. This metric is essential when False positives bring higher costs. For example, a customer might tolerate more when he has not suggested the movies that he might like than being suggested the movie that he did not like because the latter scenario wastes more time for the customers.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.12}$$

3.4.3 Recall

Recall measures the proportion of actual positive instances that have been correctly predicted as positive. Recall reflects how sensitive the model is in predicting liked movies. Recall can be important when a False-negative case brings higher costs.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.13}$$

3.4.4 F1 Score

F1 Score is the harmonic mean of Precision and Recall and can be used to evaluate the performance of the model comprehensively. The F1 score aims to strike a balance between precision and recall. As seen from the recall and precision section, we know that they focus on different scenarios, with precision being more critical when false positives are costly and recall being crucial when false negatives are costly. Therefore, to get a more balanced evaluation on both precision and recall, we sometimes use an F1 score that can take two metrics under consideration.

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.14}$$

3.4.5 Root Mean Squared Error (RMSE)

For the above-mentioned metrics, the higher, the better. Now, we introduce another metric called root mean squared error (RMSE), which is another standard metric used to evaluate the model without using the confusion metrics. It is calculated only by computing the root mean squared error between the predicted ratings and the ground truth utility metrics. The lower the RMSE, the better a model performs. It can be defined as below:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.15}$$

- n numbers of observed value, we do not calculate the error for unrated items even when we are provided with a predicted score by the model
- y_i the ith observed non-zero ground truth
- $\hat{y}_i$ the corresponding ith predictions

We tried to use multiple metrics to evaluate our model because the performance of the models can be easily misjudged by reviewing the accuracy only, especially when the distribution of the ratings from both ground truth and the predictions are not balanced. Also, for some scenarios where we are more concerned about the false positive than the false negative, multiple metrics can be helpful for some special occasions. By considering more metrics, we can have a more balanced evaluation of the performance of a model.

3.5 Summary

The methodology combined data prepossessing, implementing multiple recommender systems for comparison, including conventional RSs and other modern models, and critical quantitative evaluations gives a comprehensive evaluation. After all the recommender systems were properly built or trained, they were fed with two different user simulations, one with a 30% rating removed to simulate a normal user joining the system and the other with all ratings removed to simulate a cold start user. We set up evaluation thresholds of 3.5 to decide whether the movie is defined as liked or disliked by the user in order to compute the evaluation metrics that are widely used to evaluate RSs. Finally, the models are evaluated using the RMSE, Recall, Accuracy, Precision, and F1 score by iterating through all users in the data base and the averages metrics of them will be taken as the quantitative evaluations, and the results will be discussed thoroughly in the results and discussions chapter.

Chapter 4

Result and Discussions

In this chapter, we will dig into the details of the experiments done in the project and not only compare different models but also discuss how different hyper-parameters influence the performance of each model. The original data comes from MovieLens [12] with a total of 199,829 ratings rated by 1,384 users to 27,272 movies after data cleansing. The density for our utility matrix is 0.5%, as shown in Table 4.1. The dataset is particularly useful because it reflects a real-world challenge that all recommender systems have to face: a data sparsity issue. With this experiment conducted using this data, we can have a good understanding of how different models perform with such sparse data.

Utility matrix shape	(1,384 x 26,722)
Non-zero elements	199,829
Matrix density	0.5403%

Table 4.1: Utility Matrix Characteristics

In the experiment, we first set the original utility matrix as the ground truth, and then we picked a random user who will be used to create two manipulated data to simulate a normal user and cold start user joining the recommender system. We then use these two manipulated data with the built recommender to predict the new utility matrix for all users. By doing multiple iterations for each user $u \in U$ (a Set of all users), we can evaluate the model performance in different scenarios (normal case and cold stat user case) by computing the average value for all metrics (Average Accuracy, Average Recall ..., Average RMSE).

4.1 Model Hyper Parameters and Architectures

We will first discuss how different parameters influence the overall performance and how different architectures affect the neural networks in this section. After the parameters and

architectures for all models are decided, we will discuss the performance comparison between all models in the latter section.

4.1.1 Content-Based Filtering

This model is considered as the earliest conventional recommendation system that makes prediction under the assumption of people tend to love similar product. The prediction is made by calculating user profile using TF-IDF movie features. This model will be as the base model to be compared as a benchmark.

4.1.2 User-Based Collaborative Filtering: User Similarity

In a user-based collaborative filtering recommender system, the prediction is made by calculating the average ratings from similar users. And the most crucial part of the system is how we define the similarity between users. In the project, we first use the cosine similarity, where we project the paired user rating vector into k-dimensional space since it was widely used in other studies. $k =$ numbers of all movies, $u_i, u_j \in \mathbb{R}^k$. However, we found that this method should be adjusted by considering known values only because the similarity will be heavily influenced by the sparsity of the utility matrix.

Therefore, we first use the adjusted cosine similarity, which we consider the similarity based on commonly rated movies. We put paired user vector into d dimensional space where $d =$ numbers of commonly rated movies, $u_i, u_j \in \mathbb{R}^d$. By projecting into a lower dimensional space ($d \ll k$), the adjusted cosine similarity can be focused more on the rated portion. Another way to define similarity is using the Pearson Correlation Coefficient because it is initially defined to compute with common ratings only.

Model-2 Evaluation:

From the result shown in Figure 4.1, we can see that for adjusted cosine similarity, ignoring the zeros in the utility matrix did improve the overall performance in the collaborative filtering RS compare to the original cosine similarity and Pearson Correlation Coefficient. Additionally, RS using cosine similarity runs faster compare to the one using Pearson Correlation Coefficient, as seen in Table 4.2. One reason that causes cosine similarity to work better is that we only consider the angle between vectors but not the length. It will still capture similar behavioural patterns even if two users have different rating standards. Therefore, in most cases, causing cosine similarity is more suitable for movie recommender systems.

As for the cold start user in our dataset, changing the approach of computing similarity will not affect the performance of our conventional user-based collaborative filtering RS. Because a cold start user can not provide any useful information for RS to compute its similarities compared to other users. So in three recommender systems, we all used the global mean user profile to represent the cold start user profile.

Similarity	**Runtime**
Cosine Similarity	1m 20s
Pearson Correlation Coefficient	3m 52s

Table 4.2: Run Time Comparison for Different Similarities

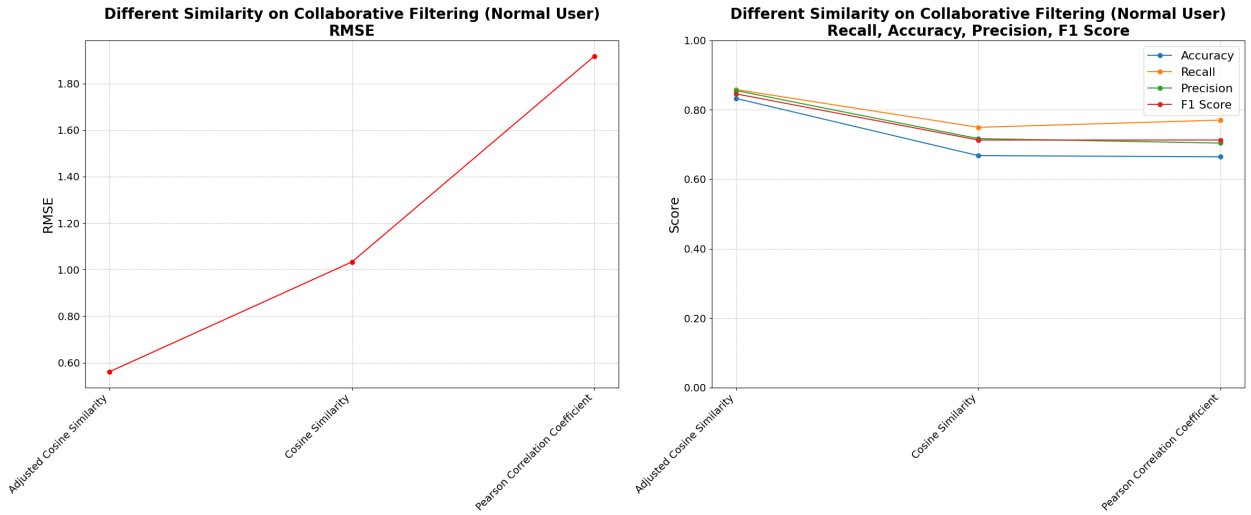


Figure 4.1: Comparison Between Different Similarities in User-Based Collaborative Filtering

4.1.3 Matrix Completion SVT: Tau and Beta

For matrix completion using singular value thresholding, there are two parameters, τ, a thresholding parameter, and β, a step parameter, that will affect the performance of the overall performance.

τ is the threshold parameter that decides how singular values are shrunk. In other words, it decides what singular value should be kept and what should be left during the update process in each iteration. Only the singular value that is higher than τ will be kept, and this is the part where we denoise the matrix and use it to obtain a low-rank approximation. So, if the τ is set to be too low, the noise might not be removed enough as we expected. In other words, too much useless information will be kept. On the other hand,

if the τ is set to be too high, lots of useful information might be removed so that it is hard to obtain an approximation with enough details.

β is used to control the weight of how the matrix will be updated. In many cases, if a β value is set too high, the unstable change might lead to an extreme change and eventually cause a bad output. However, if β is set to be a very low value, the training cost will be much higher as the RS may need more iterations to obtain a satisfying result.

Special note: Approaches to find suitable β and τ values should be done by performing multiple experiments for optimal parameter selections. However, it will cost too much time to run Matrix Completion many times. Also, in other models, the evaluation was made by getting the average metrics from iteration over all users existing in the data; it will be too costly to do so in a matrix completion RS. Because in matrix completion, it actually did not build any model but rather reconstructed a utility matrix approximation every time a simulated user joins. Therefore, in the evaluation for the matrix completion section, we will only evaluate the performance of 2 simulated users without taking the average for all users. The above issue will be left in future works.

Model-3 Evaluation:

First, we assign the τ to be 1,000 with β to be 0.3 (common range for β is from 0.1 $\rightarrow$ 1). We build up the second model by computing the SVD for the utility matrix (with missing value already prefilled) and set up the τ to be around the 25th percentile of the singular value Σ matrix, in this case 100. From Figure 4.2, we notice that in the normal user case, by lowering the τ threshold, the RMSE decreased while overall performance increased. Additionally, raising the β from 0.3 $\rightarrow$ 1 brought a small increase in overall performance and lowered the RMSE. Interestingly, in Figure 4.3, we notice that the parameter change had an opposite effect on the normal user case and cold start user case, respectively.

One possible reason for the above result can be caused by the prefilling process of the utility matrix. As we all know, a singular value decomposition (SVD) can only be performed when no missing value exists. To achieve the matrix completion with SVT, we first need to fill in all the missing values in the utility matrix. However, the value filled in the method might not be accurate, in other words, in a cold start user's case, the filled value might be considered as noise in the first place. Using higher τ, we might coincidentally remove more noise, which leads to a lower RMSE and better performance. On the other hand, using such high τ in a normal user case might remove too much useful information, as a lower percentage of missing values was filled initially compared to cold start cases. It is worth studying this

phenomenon in order to improve the algorithm. Another possible reason might be caused by the way we evaluate the models. As mentioned in the above special note section, the cost for running iteration through all users in Matrix Completion is too high, so by only evaluating two simulated users, the performance might be biased. This particular evaluation method for matrix completion, includes SVT and ALS, should be improved as well. Above mentioned point should also be considered to be done in future works.

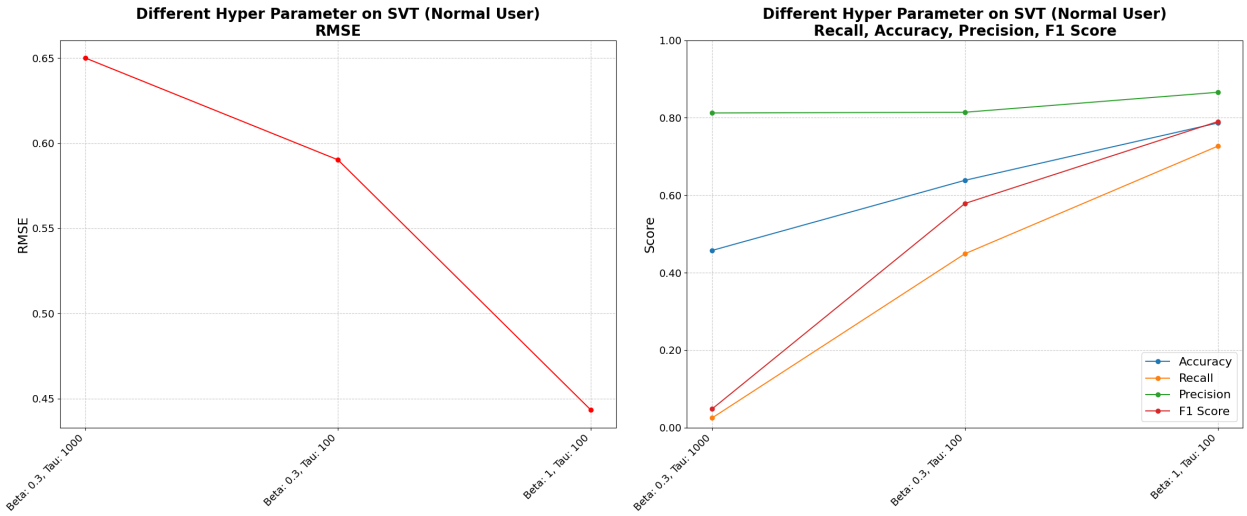


Figure 4.2: Comparison Between Different Parameters in SVT: Normal Case

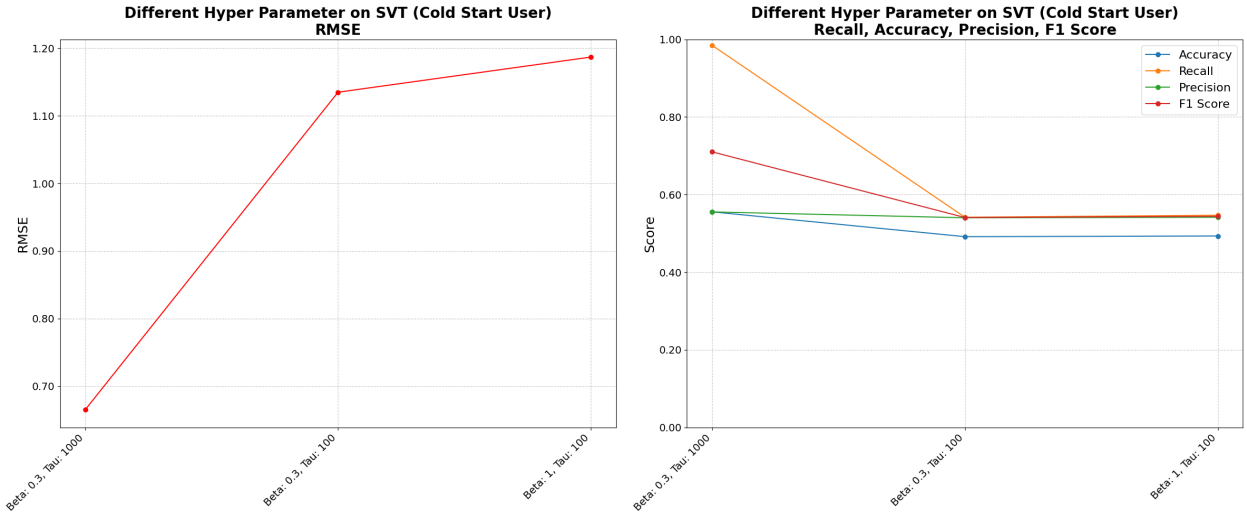


Figure 4.3: Comparison Between Different Parameters in SVT: Cold Start Case

4.1.4 Matrix Completion ALS

The original thought for introducing the Alternating Least Squares (ALS) method to solve a matrix completion problem is that it does not rely on prefilling all the missing utility matrices. Unlike performing a low-rank approximation in the previous SVT method, ALS compares the error between observed ratings and the predicted ratings and then alternately updates the U and V^T, which is considered to be a suitable method when it comes to sparsity issues.

Model-4 Evaluation:

Firstly, we would like to discuss how the RS performs in the cold user scenario because the algorithm should be good at dealing with sparse data by its design. Although the solution seems to be very promising at first, it still can not solve the problem caused by cold start users nor perform well under this circumstance. From Figure 4.4, the precision and the recall were all zeros, which makes the F1 Score to be zero as well. The problem is caused by how the ALS algorithm is designed. The core concept of the algorithm is to ignore missing values in the first place in order to handle a sparse matrix completion, and by doing so, it can not capture the pattern of the cold start user. Therefore, the matrix approximation process can not get any information from it, which leads to the inability to provide good predictions for this particular cold start scenario effectively. By prefilling the utility matrix, we successfully lower the RMSE and improve the overall performance.

The result for a normal case can be seen in Figure 4.5. We found that RMSE also decreased after filling out the initial utility matrix. Surprisingly, other evaluation metrics decreased at the same time. This is not a common thing because we usually expect to see a negative correlation between overall performance and RMSE. However, unlike RMSE, which cares about the absolute difference between prediction scores and real scores, other metrics focus more on how well a binary classification is done. Therefore, an increase in RMSE does not necessarily mean that the Accuracy, Recall, and other binary classification metrics will be worse, vice versa. All in all, even though ALS is good for dealing sparsity matrix, in order to perform better in both cold start user case or normal user case, a more accurate fill-in method should be implemented in the data preprocessing phase. Same as what we discovered in SVT, if the quality of fill-in method is not good enough, then it might not help the algorithm as we initially thought and bring more noise to the original data and lower the overall precision.

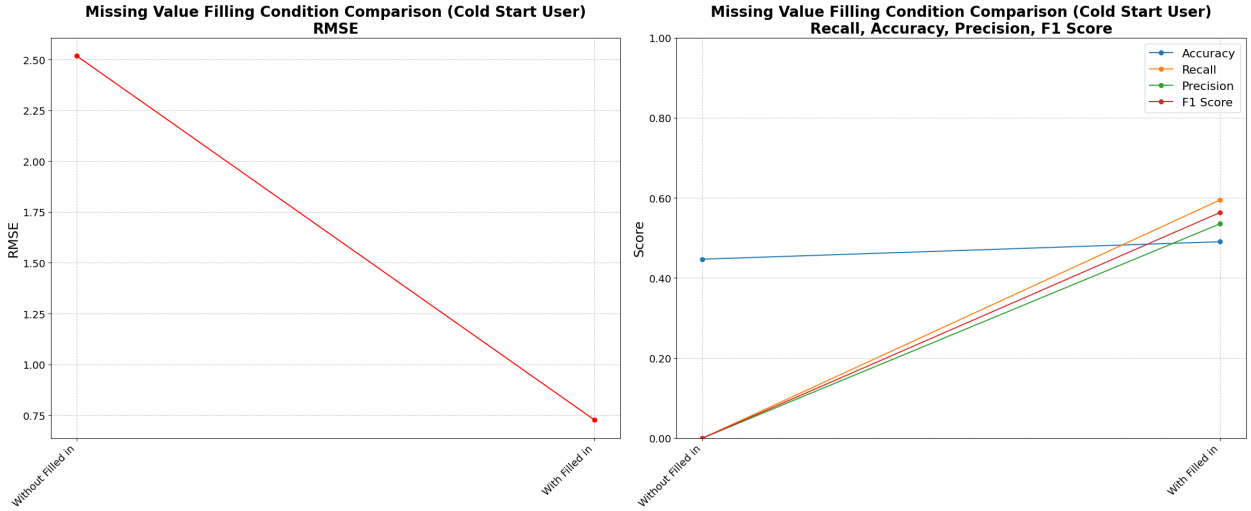


Figure 4.4: Difference between Unfilled/ Filled Utility Matrix: Cold Start Case

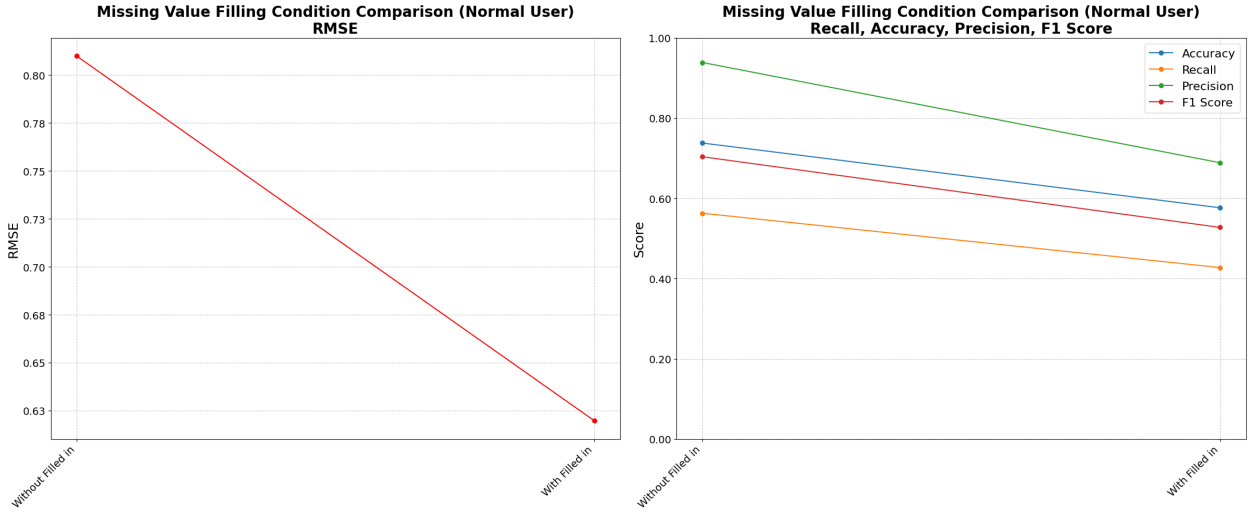


Figure 4.5: Difference between Unfilled/ Filled Utility Matrix: Normal User Case

4.1.5 Multi-Layer Perceptron

Throughout all three types of Neural Network models built in this project, Multi-Layer Perceptron, autoencoder and matrix factorization, respectively, we all start with the exact dimension of latent space set as 64. Other architectures then be built for comparison. Below is the introduction for the models built in the MLP section. Neuron drops out every layer except the last hidden layer before output with a dropout rate of 0.5. We use dropout and ℓ_2 regularization of λ = 1e-4 to prevent overfitting.

- MLP1 : Embedding Feature Layer 32×2 (User 32 + Item 32)
  Input: [64] $\rightarrow$ Hidden Layers: [32, 16, 8, 4, 2] $\rightarrow$ Output: [1]
  This is the base model built to be compared.

- MLP2 : Embedding Feature Layer 32×2 (User 32 + Item 32)
  Input: [64] $\rightarrow$ Hidden Layers: [32, 8, 2] $\rightarrow$ Output: [1]
  By reducing the depth of the network, we aim to reduce the complexity of the model and see how it will affect the overall performance.

- MLP3 : Embedding Feature Layer 258×2 (User 258 + Item 258)
  Input: [512] $\rightarrow$ Hidden Layers: [256, 128, 64, 32, 16, 8, 4, 2] $\rightarrow$ Output: [1]
  A simplified version of the deep and wide model from the study done by Cheng et al. (2016) [7]. The concept of deep and wide architecture uses a linear regression model at the input to capture and memorize the pattern and users' special behaviour, while outputs to a deep neural network enhance the ability of generalization and capture non-linear features. Instead of using hybrid models, we continue using the MLP model but change the input to relatively wide embeddings with deep hidden layers.

Model-5 Evaluation:

We first discuss the normal user case. From Figure 4.6, we can see that changing the depth of the hidden layers does not bring too much impact in terms of binary classification metrics (from MLP1 $\rightarrow$ MLP2). On the other hand, changing the width of the input layers, which are the embedding feature layers (from MLP1/ MLP2 $\rightarrow$ MLP3), brought a bigger change in the overall performance, especially on the binary classification metrics. We noticed that MLP3 (having the widest Embedding Layers with the deepest hidden layers) has the highest precision despite having the highest RMSE and lowest recall. When it comes to evaluating a recommender system, precision should be heavily considered. As mentioned in the methodology, precision is influenced by the false positive case, which is considered to be an important scenario that we want to avoid. Viewers tend to be more accepting of not being recommended the movie that they might like than being recommended something they do not like. The former can barely be noticed sometimes, but the latter scenario can affect the viewers' overall experience. Thus, we choose MLP-3 as it has the best precision compared with MLP-1 and MLP-2.

As for the cold start user shown in Figure 4.7, the MLP-1 and MLP-3 has very close performance in terms of binary classification task, but MLP-1 performs better in RMSE. These two model raise their precision by being conservative to give lower recalls, however

being conservative in a recommender system can sometimes beneficial to viewers as they avoid more false positive case as well.

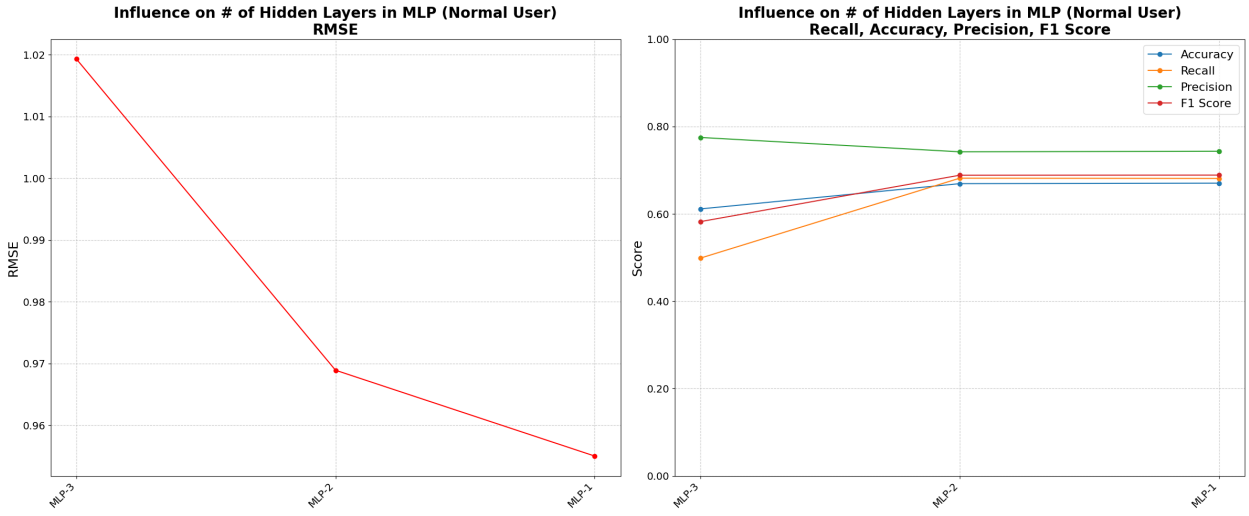


Figure 4.6: Different Architecture MLP: Normal Case

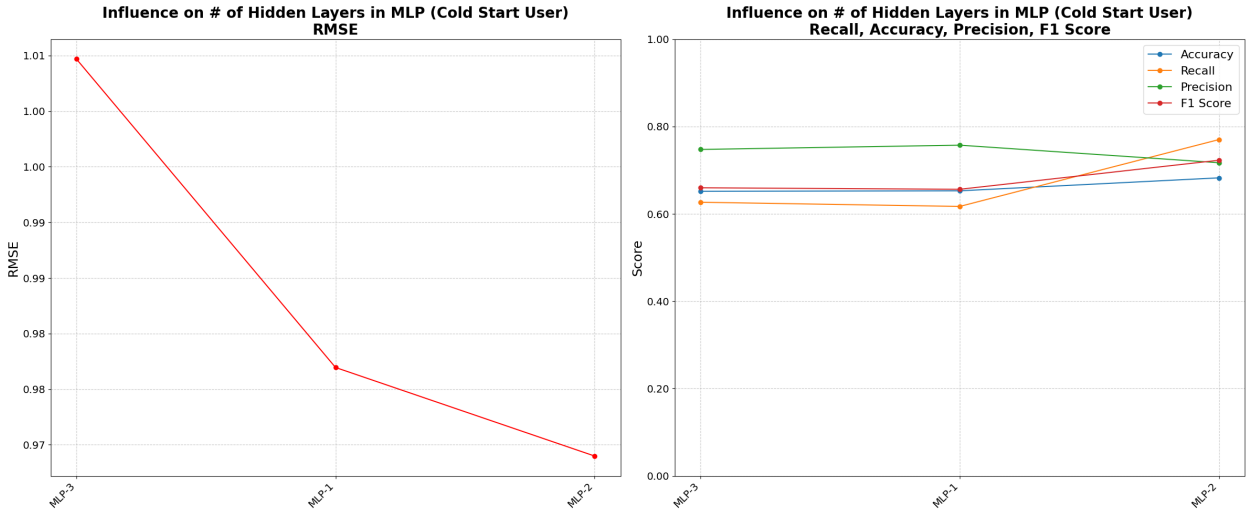


Figure 4.7: Different Architecture MLP: Cold Start Case

4.1.6 AutoEncoder: User-Based vs Item-Based

In this section, we still set up the dimension of the latent space to be 64, the same as other neural networks with a dropout rate of 0.5 and ℓ_2 regularization of λ = 5e-4 to prevent overfitting. By theory, an autoencoder RS can accept both the user-item rating vector

and the item-user rating vector as inputs. This concept is very similar to what a content-based filtering RS or a user-based collaborative filtering RS will do. By changing the input, the model changes how it makes predictions by considering user behaviour patterns or the similarity between movies.

- User-Based M1 : Latent Space: 64
  Input: [1,384 Users] → Encoder: [256, 128] → Latent Space:[64] → Decoder: [256, 128] → Output: [1,384 Users]
  This is the base model built to be compared.

- User-Based M2 : Latent Space: 64
  Input: [1,384 Users] → Encoder: [1,024, 512, 256, 128] → Latent Space:[64] → Decoder: [128, 256, 512, 1,024] → Output: [1384 Users]
  By increasing the depth of the encoder and decoder, we aim to explore whether adding more layers can capture additional details, potentially leading to an overall improvement in performance.

- Item-Based M3 : Latent Space: 64
  Input: [26,722 Movies] → Encoder: [256, 128] → Latent Space:[64] → Decoder: [128, 256] → Output: [26,722 Movies]
  Change from input user-movie vector to movie-user vector. Other architecture remains the same compared to the M1 model.

Model-6 Evaluation:

Again, we started by evaluating the normal user case. From Figure 4.8, we found that changing the architecture will bring a tiny decrease in RMSE, recall and corresponding F1 Score (from M1 → M2). However, if we change the input from using a user-movie vector to a movie-user vector (M1/M2 → M3), then the overall performance drops significantly, especially recall and the corresponding F1 score. This result is convincing because we observed similar patterns in the content-based filtering and user-based collaborative filtering sections, where the user-based algorithm performed better than the item-based algorithm using the MovieLens dataset.

See Figure 4.9 for the cold start user case. By increasing the depth of the encoder and decoder sections, the RMSE and accuracy are almost identical, with a little decrease on recall and corresponding F1-Score. At the same time the precision raises slightly, which, again, is something we would love to see. By increasing the depth of the architecture, the

model seems to give more conservative predictions, which leads to a lower recall but a higher precision. Another thing that should be noticed is that in the cold start user case, the item-based architecture remains to be the worst model having largest RMSE, and lowest recall, precision, accuracy and F1-score. Therefore, we prefer the user-based model 2 in this section.

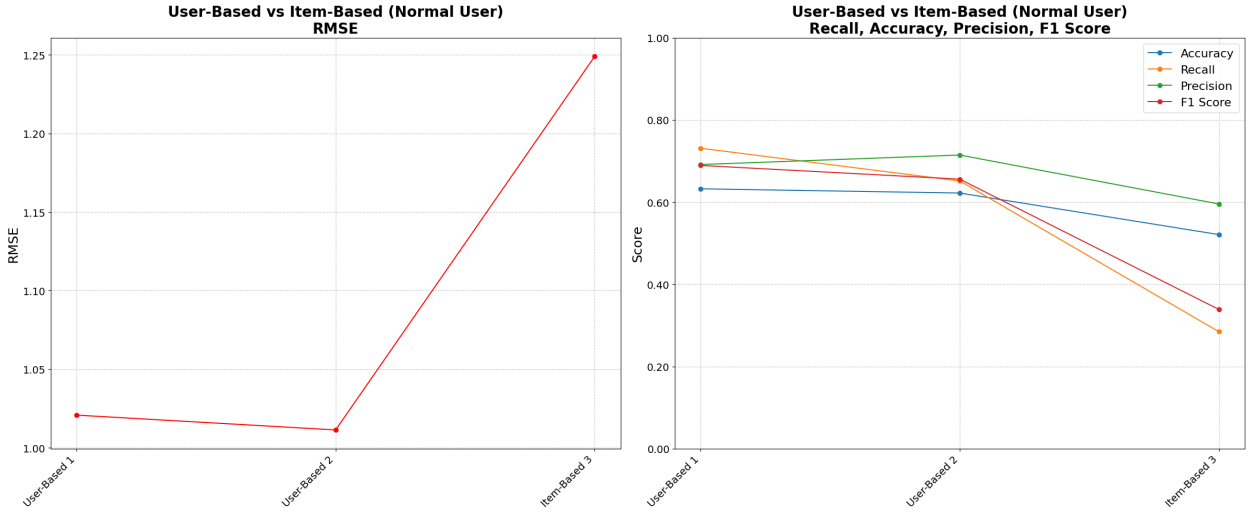


Figure 4.8: Different Architecture in AutoEncoder: Normal Case

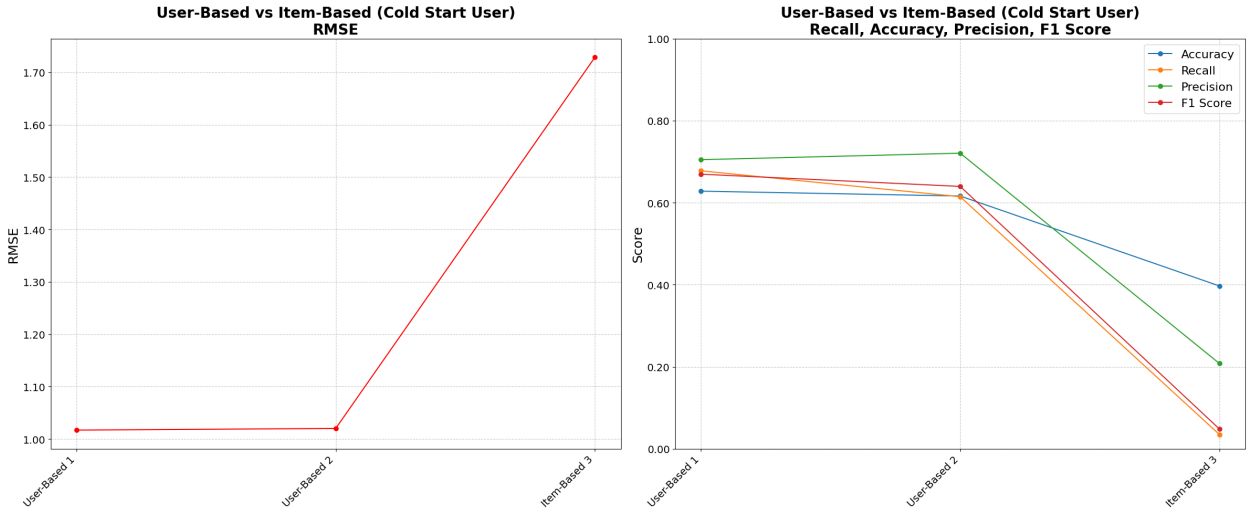


Figure 4.9: Different Architecture in AutoEncoder: Cold Start Case

4.1.7 Matrix Factorization: Change of Embedding Size

Though MLP and autoencoder seem to be an excellent neural network tool for solving recommender system tasks, we started to wonder if there is another way we can build a recommender system from the basic definition. That is why we tried to build a matrix factorization network simply considering the ratings, an interaction between the user and the movie, as a dot product of the user's latent vector and the movie's latent vector. We project the user and movie into their own latent space, representing the hidden relations between users and movies. In this section, we still set up the Embedding dimension for the user and movie to be 64, the same as other neural networks with a dropout rate of 0.5 and ℓ_2 regularization of λ = 5e-3 to prevent overfitting.

- MF 1: Embedding Dimension: 64
  Input: [Latent Vector U, Latent Vector V] → Dot Product: [64] + Bias → Output: [1]
  Latent Vector U represents the user latent vector from the user-embedding matrix, and Latent Vector V represents the movie latent vector from the movie-embedding matrix. This is the base model built to be compared.

- MF 2: Embedding Dimension: 16
  Input: [Latent Vector U, Latent Vector V] → Dot Product: [16] + Bias → Output: [1]
  We will form a model MF2 with a smaller embedding dimension and a model MF3 with a larger embedding dimension for comparison.

- MF 3: Embedding Dimension: 1024
  Input: [Latent Vector U, Latent Vector V] → Dot Product: [1024] + Bias → Output: [1]

Model-7 Evaluation:

In a normal user case, as shown in Figure 4.10, as the number of the embedding dimension increases from 16 to 1024, the RMSE decreases. Recall, accuracy and F1 Score drops firstly when the embedding dimension goes from 16 to 64, then raises when the embedding dimension goes from 64 to 1024. Despite the unusual change in recall, accuracy and F1 Score, the precision increased when the embedding dimension increased.

In Figure 4.11, a similar phenomenon happens in the cold start user as well. As the embedding increased, the recall and F1 score first dropped and then raised. Overall, the model results from the cold start user case and normal user case are very similar and almost synchronized from embedding dimensions 64 to 1024. The model also did well in both

scenarios, with little difference in binary classification evaluation metric and RMSE, which is rare throughout our project. Therefore, this is a good model to be used in the movie recommender. To optimize the architecture, more experiments should be done by changing the embedding dimensions with a higher range of options and different combinations of learning rates and regularizers, and we will leave this part for future work.

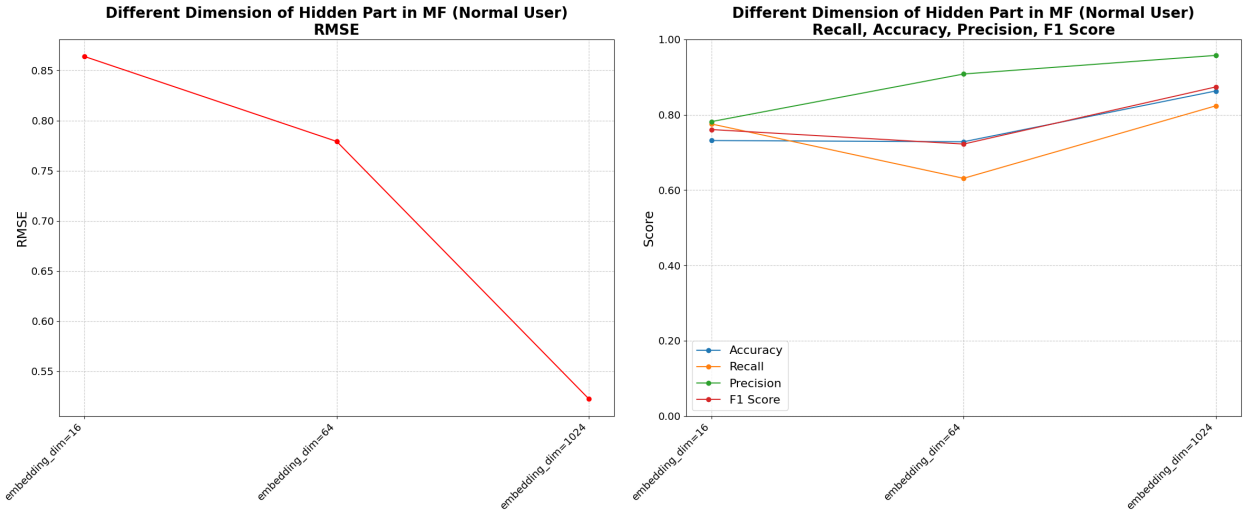


Figure 4.10: Different Architecture in Matrix Factorization: Normal Case

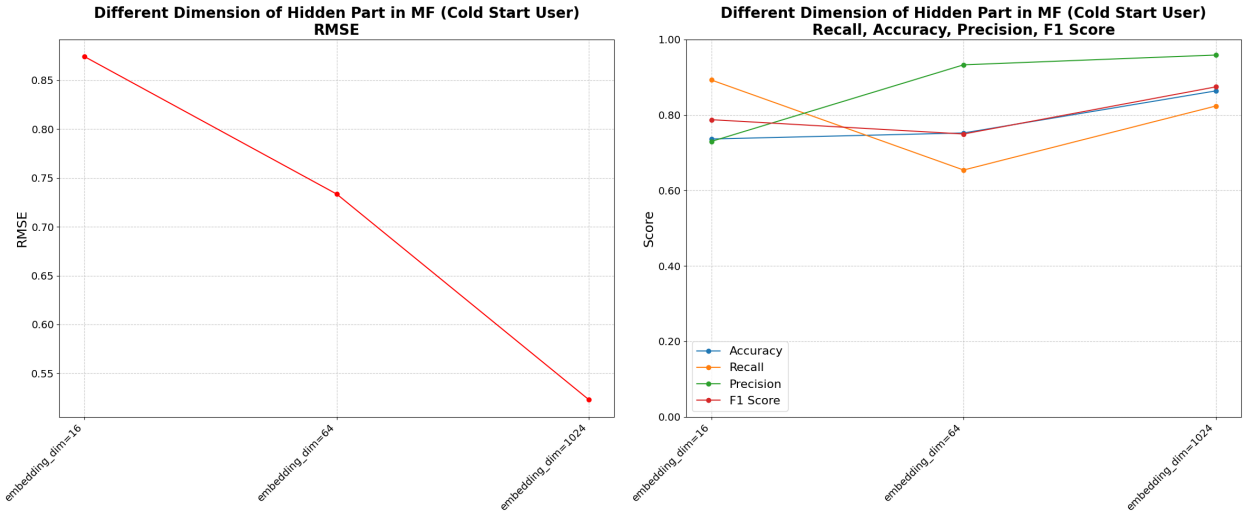


Figure 4.11: Different Architecture in Matrix Factorization: Cold Stat Case

4.2 Model Comparisons: Conventional RS vs Matrix Completion RS and Neural Network RS

After the ideal models for each algorithm are selected, we will compare how different RSs perform in this project. We will start from evaluating the normal user case then to cold start user case.

4.2.1 Normal User Scenario Discussion

In terms of RMSE, as shown in Figure 4.12(a), we can clearly group the result into three tiers. The smaller the RMSE, the better the performance.

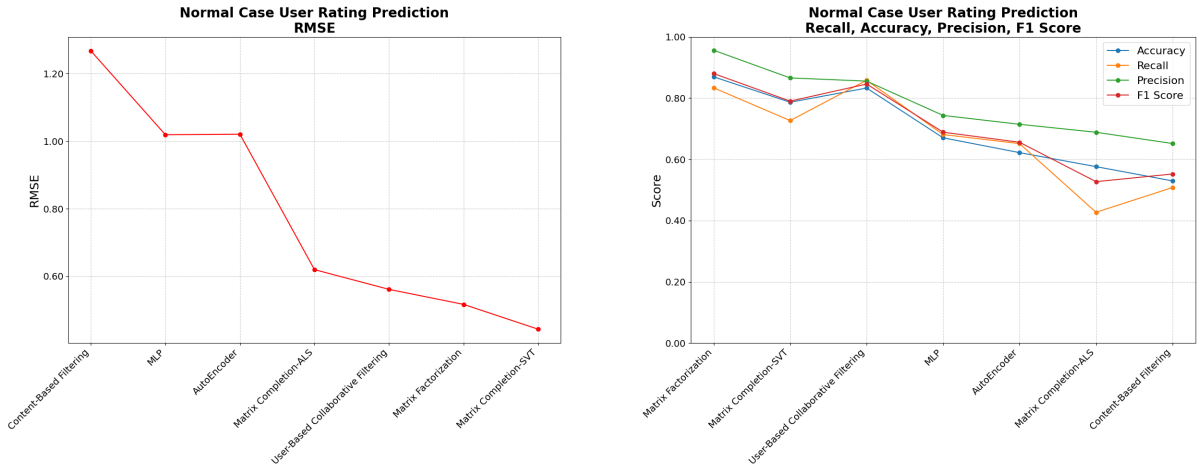
- Tier1: Matrix Completion SVT > Matrix Factorization > User-Based Collaborative Filtering > Matrix Completion ALS
- Tier2: Autoencoder ≈ MLP
- Tier3: Content-Based Filtering

We can clearly see that the conventional content-based filtering RS perform the worst when evaluated by RMSE. By only considering the profile of the user's interest built by the TF-IDF movie feature vector, the predictions were not as good as other models because movie genres might not be well categorized and limited. As mentioned in the literature review, the main challenge that content-based filtering faces is that it relies heavily on hand-crafted features. With limited features given by the dataset, the profile of users' interests can not be built properly, which leads to poor results. Surprisingly, the other conventional model, user-based collaborative filtering, ranked third in terms of RMSE. By adjusting the cosine similarities, the algorithm did a good job of giving a low RMSE prediction. We will evaluate further using other metrics to give final verdicts to the models in normal user cases.

We can not evaluate a model by only looking at RMSE. Other metrics, especially precision, should be considered as well. From Figure 4.12(b) we can see following:
Matrix Factorization > Matrix Completion SVT ≈ User-Based Collaborative Filtering > MLP > Matrix Completion ALS > Autoencoder > Content-Based Filtering

The top three models in terms of precision are Matrix Factorization, Matrix Completion SVT, and User-Based Collaborative Filtering, which are also the top three models in terms of RMSE. Above result shows that different type of algorithms (conventional RS, matrix completion and neural network) can all shine in a normal user case with a good choice of

architecture and hyper parameters. The matrix factorization model and user-based collaborative filtering have all four binary classification metrics above 80%, indicating that they are good models in normal user scenarios. Unlike MLP or Autoencoder, which seems to be black boxes for someone who is not familiar with how neural networks work, the matrix factorization expresses the interaction between user and movie through the dot product of the user latent vector and movie latent vector, making a better explanation. User-based collaborative filtering is also convincing, using the concept that similar people will like similar stuff. Even though it can not perform as well as the modern matrix factorization neural network model, it is still good enough for normal user scenarios, even under the limited features provided. Also, Matrix Completion using SVT is a good method, by transforming a utility matrix-solving problem into a matrix completion problem also gives a good result. But, it relies heavily on how we prefill the missing value, and we hope to make more improvements in future works.



(a) RMSE

(b) Evaluation Metrics: Ordered by Precision

Figure 4.12: Model Comparison: Normal Case

4.2.2 Cold Start User Scenario Discussion

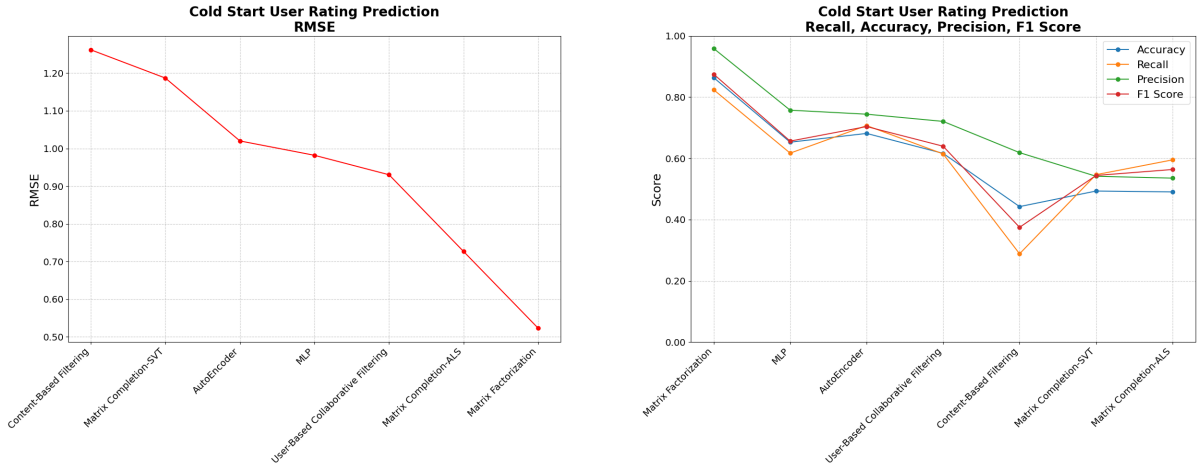
In terms of RMSE, From Figure 4.13(a), we can also group the cold start user result into three tiers.

- Tier1: Matrix Factorization > Matrix Completion ALS
- Tier2: User-Based Collaborative Filtering > MLP > Autoencoder
- Tier3: Matrix Completion SVT > Content-Based Filtering

Again, we see the Matrix Factorization model remaining in the first tier and Content-Based filtering being in the last tier. However, this time, User-Based Collaborative Filtering drops to tier 2 because we can not properly compute user similarities with empty user data input. As for Matrix Completion, we can see that ALS works better than SVT in terms of the RMSE. We think that it might be caused by SVT being poorly prefilled in a cold start user scenario. By ignoring missing values, ALS actually performed better. We will evaluate the model more using more metrics since it is hard to judge a model fully without considering precision.

From Figure 4.13(b), the ranking in terms of precision is as following:
Matrix Factorization > MLP ≈ Autoencoder ≈ User-Based Collaborative Filtering > Content-Based Filtering > Matrix Completion SVT ≈ Matrix Completion ALS

It is evident that in the cold start user case, three neural network models (Matrix Factorization, MLP and Autoencoder) performed well in the binary classification task. They are the top three models in terms of precision. The result meets our expectations because we initially assumed that the neural network model would perform better in a cold start user case since neural networks can handle non-linear relationships and learn from latent factors, which conventional models are incapable of. Matrix Factorization also performed well in cold start user and normal user scenarios. Learning the hidden feature in latent space overcomes the cold start user problem to some extent. Not only does it have good quality predictions in terms of RMSE, but it also has high precision. At the same time, the overall performance difference between the cold start user case and the normal user case for the matrix factorization model is small, showing good stability among different scenarios. Although MLP and Autoencoder did not perform well in terms of RMSE, they did an excellent job in terms of precision with relatively high recall and accuracy. User-based collaborative filtering RS can be promising even though, as a conventional RS, it does not deliver high-quality predictions like neural network models in the cold start user scenario. The performance gap between user-based collaborative filtering RS and neural network models are smaller than we expected. User-based collaborative filtering RS also performed well in both scenarios and had strong interpretability, explaining how the algorithm works. Matrix completion performed poorly in the cold start user scenario. We believe that too many missing values cause the problem and that we did not prefill the matrix well enough. Although ALS is designed to deal with sparsity problems, it fits poorly in a cold start user case because it ignores all the information from the end user and leads to poor predictions.



(a) RMSE (b) Evaluation Metrics: Ordered by Precision

Figure 4.13: Model Comparison: Cold Start User Case

Chapter 5

Conclusion and Recommendation

5.1 Conclusion

This project successfully achieved the aim and fulfilled the objectives by comprehensively reviewing multiple recommender systems, testing models with different parameters and architectures with detailed evaluation, and finally, giving model suggestions to the viewers.

We conclude from the result that, with limited input features from the raw data, good recommender systems can be built with conventional or modern deep learning models. In the project, a matrix factorization neural network and a user-based collaborative filtering RS did a great job in the usual user case. In a cold start user case, all three neural network models perform better than the conventional models and matrix completion RS, which meets our assumption that neural network models can benefit from their ability to capture non-linear patterns and can learn from latent factors, causing them to perform better in a cold start user scenario. Despite neural networks having such an advantage, user-based collaborative filtering was just a little behind. Even if we can not directly compute the user similarity between a cold start user and other users in the conventional collaborative filtering RS, we still have different methods to replace the missing value to increase the overall performance.

The two recommender systems (Matrix Factorization and User-Based Collaborative Filtering) should be good options for small companies and startups that can only handle small data. However, in terms of the data, the more, the better. As the company grows, it will start to collect more data for users and items as well as increase the features used in the recommender system. When more movie or user features are introduced, we believe a model-based hybrid recommender will be more suitable because it can complement conventional and

modern model's strengths and weaknesses. Additionally, the scalability of the model-based RS can decrease the overall runtime when the data becomes more extensive because users can use the trained weight directly to predict the ratings, but the conventional memory-based RS has to compute almost everything from scratch.

5.2 Suggestions of future works

There are some improvements can be done in future works according to the result, from data collecting improvements to algorithm improvements.

(1) More data should be collected and tested. By adding more features, we would like to see if a simple model, such as a user-based collaborative filtering model, can still perform similarly compared with a complex hybrid models that take multiple inputs into training.

(2) In matrix completion sections, the evaluation method should be improved. Because it took too many time to perform matrix completions iterate through all users. Therefore, some other evaluation methods should be considered to achieve a more accurate and efficient assessment.

(3) Other metrics can be used for evaluation, such as Normalized Discounted Cumulative Gain (NDCG). The RMSE only calculate the difference between prediction ratings and real ratings but neglects the binary classification. However, it is also important not only to consider the binary classification (Liked or Disliked) but also to consider the order of the recommendations. NDCG is a good way to evaluate it.

(4) A better filling method in the data pre-processing phase should be done. We can see from the result that a good filling method for the utility metrics can increase the overall performance of the recommender systems. Additionally, matrix completion methods rely heavily on this procedure. If we can make a better matrix prefilling, we can truly make the matrix completion method more powerful.

(5) A recommender system needs extensive hyper-parameter tuning, which is a common problem for machine learning in general. More experiments should be done in order to optimize the parameters.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[2] Chris Anderson. *The long tail: How endless choice is creating unlimited demand*. Random House, 2007.

[3] Zeynep Batmaz, Ali Yurekli, Alper Bilge, and Cihan Kaleli. A review on deep learning for recommender systems: challenges and remedies. *Artificial Intelligence Review*, 52:1–37, 2019.

[4] Fjolla Berisha and Eliot Bytyçi. Addressing cold start in recommender systems with neural networks: a literature survey. *International Journal of Computers and Applications*, 45(7-8):485–496, 2023.

[5] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *arXiv preprint arXiv:1301.7363*, 2013.

[6] Robin Burke, Alexander Felfernig, and Mehmet H Göker. Recommender systems: An overview. *Ai Magazine*, 32(3):13–18, 2011.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[8] Casper da Costa-Luis. Tqdm: A fast, extensible progress meter for python and cli. *Journal of Open Source Software*, 4(37):1277, 2019.

[9] Joaquin Delgado and Naohiro Ishii. Memory-based weighted majority prediction. In *SIGIR Workshop Recomm. Syst. Citeseer*, page 85. Citeseer, 1999.

[10] Angela Edmunds and Anne Morris. The problem of information overload in business organisations: a review of the literature. *International journal of information management*, 20(1):17–28, 2000.

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

[12] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[13] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[15] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201, 1995.

[16] William H Hsu, Prashanth Boddhireddy, and Roby Joehanes. Using probabilistic relational models for collaborative filtering. In *SRL2003 IJCAI 2003 Workshop on Learning Statistical Models from Relational Data*, page 9, 2003.

[17] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[18] Sheena S Iyengar and Mark R Lepper. When choice is demotivating: Can one desire too much of a good thing? *Journal of personality and social psychology*, 79(6):995, 2000.

[19] Gourav Jain, Tripti Mahara, and Kuldeep Narayan Tripathi. A survey of similarity measures for collaborative filtering-based recommender system. In *Soft Computing: Theories and Applications: Proceedings of SoCTA 2018*, pages 343–352. Springer, 2020.

[20] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[21] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[22] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[23] Wes McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51–56, 2010.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Alexander Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Shubho Chilamkurthy, Boris Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035, 2019.

[25] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, M Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Kenneth A Rasinski, Gordon B Willis, Alison K Baldwin, Wenchi Yeh, and Lisa Lee. Methods of data collection, perceptions of risks and losses, and motivation to give truthful answers to sensitive survey questions. *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, 13(5):465–484, 1999.

[27] Ram Murti Rawat, Vikrant Tomar, and Vinay Kumar. An embedding-based deep learning approach for movie recommendation. In *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pages 1145–1150. IEEE, 2020.

[28] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 240–248, 2020.

[29] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, 1994.

[30] Elaine Rich. User modeling via stereotypes. *Cognitive science*, 3(4):329–354, 1979.

[31] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[32] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, pages 111–112, 2015.

[33] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, 1995.

[34] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*, volume 30, pages 47–56. Barcelona, 2000.

[35] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J van der Walt, Michael Brett, Joshua Wilson, K Jarrod Millman, Nikolay Mayorov, Andrew R J Nelson, Eric Jones, Robert Kern, Eric Larson, Christopher J Carey, İlhan Polat, Yu Feng, Eric W Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E A Quintero, Charles R Harris, Anne M Archibald, Antônio H Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020.

The experiments done in the project are conducted using python with following libraries: Pandas [23], Numpy [13], Scikit-learn [25], SciPy [35], PyTorch [24], Matplotlib [17], Tqdm [8]