**UNIVERSITY OF**

**SCIENCE AND TECHNOLOGY**

**OF HANOI**

**Machine Learning and Data Mining 2**

**REPORT**

**Recommender systems**

**GROUP 5**

Phan Thanh Bình - BI12-059(\*)

Vũ Quỳnh Anh - BI12-019

Trần Thế Trung - BI12-452

Trần Minh Trung - BA11-093

Châu Phan Phương Mai - BI12-263

Nguyễn Thị Thảo - BI12-423

Nguyễn Hà Phương - BI12-359

**Table of contents**

[I. Introduction 2](#_Toc130422432)

[1. Overview 2](#_Toc130422433)

[2. Motivation 2](#_Toc130422434)

[3. Classifications of Recommendation Systems 3](#_Toc130422435)

[4. Contribution 5](#_Toc130422436)

[II. Project process 6](#_Toc130422437)

[1. Dataset 6](#_Toc130422438)

[2. Data Analysis 7](#_Toc130422439)

[3. Model 11](#_Toc130422440)

[a. Hybrid User-Item based Collaborative Filtering 11](#_Toc130422441)

[b. Matrix Factorization and Deep Learning 15](#_Toc130422442)

[c. Self-attention model 18](#_Toc130422443)

[4. Experiment design 22](#_Toc130422444)

[a. Evaluate metrics 22](#_Toc130422445)

[b. Hybrid User-Item based Collaborative Filtering 22](#_Toc130422446)

[c. Matrix Factorization and Deep Learning 24](#_Toc130422447)

[d. Self-attention model 26](#_Toc130422448)

[e. Compare three hybrid Collaborative Filtering model 27](#_Toc130422449)

[III. Conclusion 28](#_Toc130422451)

[IV. References 29](#_Toc130422452)

# Introduction

## Overview

A recommendation system is a tool that assists users in making choices by providing them with personalized suggestions. It is a technique for filtering information that tries to anticipate and recommend things that a person might be interested in. These systems have become increasingly popular due to the rapid growth of online content, which can make it difficult for users to navigate and find what they are looking for.

** Recommender systems have been applied in a wide range of domains, including e-commerce, entertainment, social networking, and online dating. Examples of popular recommendation systems include Netflix, which suggests movies and TV shows based on a user's viewing history, and Spotify, which recommends music based on a user's listening history.

Picture 2. Spotify's recommendations



Picture 1. Netflix's recommendations

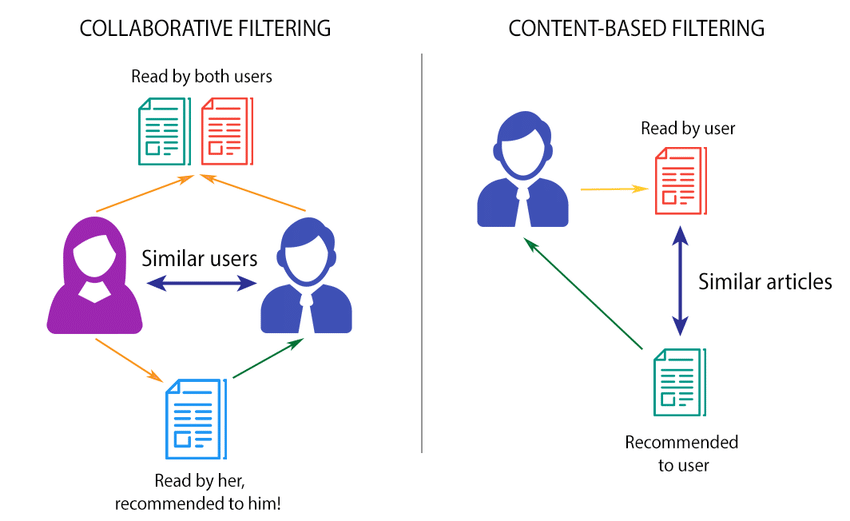
## Motivation

In today's era of Big Data, the sheer amount of available information makes it challenging for users to filter out relevant data from the overwhelming amount of irrelevant information. This leads to the problem of information overload, which can interfere with users' ability to make informed decisions. Recommender systems (RS) help to address this problem analyzing user behavior and recommend items that match their interests.

By analyzing users' historical behavior, RS can establish interest models that better match the users' interests. This means that RS does not require users to provide clear needs or requirements, instead, it analyzes users' behavior to make relevant recommendations. As a result, RS can significantly reduce consumer search costs and uncertainty associated with the purchase of unfamiliar products, leading to enhanced user loyalty and overall satisfaction.

Recommender systems play a vital role in cross-selling, as they enable companies to introduce customers to new products that they may not have discovered on their own. This can help businesses increase their revenue by ensuring that customers regularly discover new products that are of interest to them. For instance, Amazon and Netflix are two examples of companies that have leveraged recommender systems to drive their growth. Amazon's recommender system, which took over a decade to develop, increased their additional sales by 20% in 2002. Similarly, Netflix established a competition to improve the accuracy of its movie recommender system by 10% (Paschos), indicating the crucial role of recommender systems in the success of a company. [1]

## Classifications of Recommendation Systems



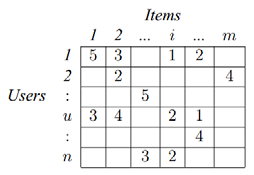
Picture 3. Classification of Recommendation System [2]

Recommendation systems can be divided into two branches: content-based and collaborative-based. Recommendations made by content-based recommendation systems are based on user profiles and item descriptions. To find trends and similarities in users' interests, this approach examines the content of items and users’ prior activities. The success of a content-based recommendation system depends heavily on the knowledge of the builders about features of the items to represent them correctly in latent space. In contrast, collaborative-based recommendation systems build their recommendations on past interactions and ratings of objects. This type of system identifies similarities between users' behavior and past interactions with items to suggest new items that they might like. Both types of recommendation systems have their advantages and limitations, and choosing the right type depends on the specific requirements and characteristics of the application. Content-based recommendation systems work well when there is a lot of available content data, while collaborative-based recommendation systems work better when there is a significant amount of user data available.

1. ***Collaborative filtering***

In this report, we will focus on Collaborative Filtering recommendation systems.

Collaborative filtering is a recommendation system technique that relies on past user interactions with items to suggest new items. Typically, a collaborative filtering approach will take into account the user-item ratings matrix like below where empty values indicate no interaction yet between users and items.



Picture 4. The user-item ratings matrix [3]

There are two most commonly used approaches to collaborative filtering: memory-based and model-based. In the memory-based approach, the Pearson:

or cosine similarity between user-vectors or item-vectors can be calculated based on the ratings matrix:

Then the rating of user U on item I can be determined by the ratings of U’s K-nearest neighbors on that item/ ratings of U on I’s K-nearest neighbors.

On the other hand, model-based collaborative filtering uses regression or classification machine learning models to predict the missing ratings values. This method can handle sparse data more skillfully and is more scalable than memory-based collaborative filtering.

1. ***Challenges of collaborative filtering recommender systems [4]***

* *Data Sparsity*

In practice, many commercial recommender systems are used to evaluate very large product sets. The user-item matrix used for collaborative filtering will thus be extremely sparse (usually > 99% of unknown values) and the performances of the predictions recommendations of the CF systems are challenged.

* *Scalability*

When numbers of existing users and items grow tremendously, traditional CF algorithms will suffer serious scalability problems, with computational resources going beyond practical or acceptable levels. For example, with tens of millions of customers (M) and millions of distinct catalog items (N), a CF algorithm with the complexity of O(n) is already too large. As well, many systems need to react immediately to online requirements and make recommendations for all users regardless of their purchases and ratings history, which demands a high scalability of a CF system.

To alleviate these problems, many hybrid models which ensemble different traditional models together have been proposed. The idea is that one approach’s pros will help other cons of approaches.

## Contribution

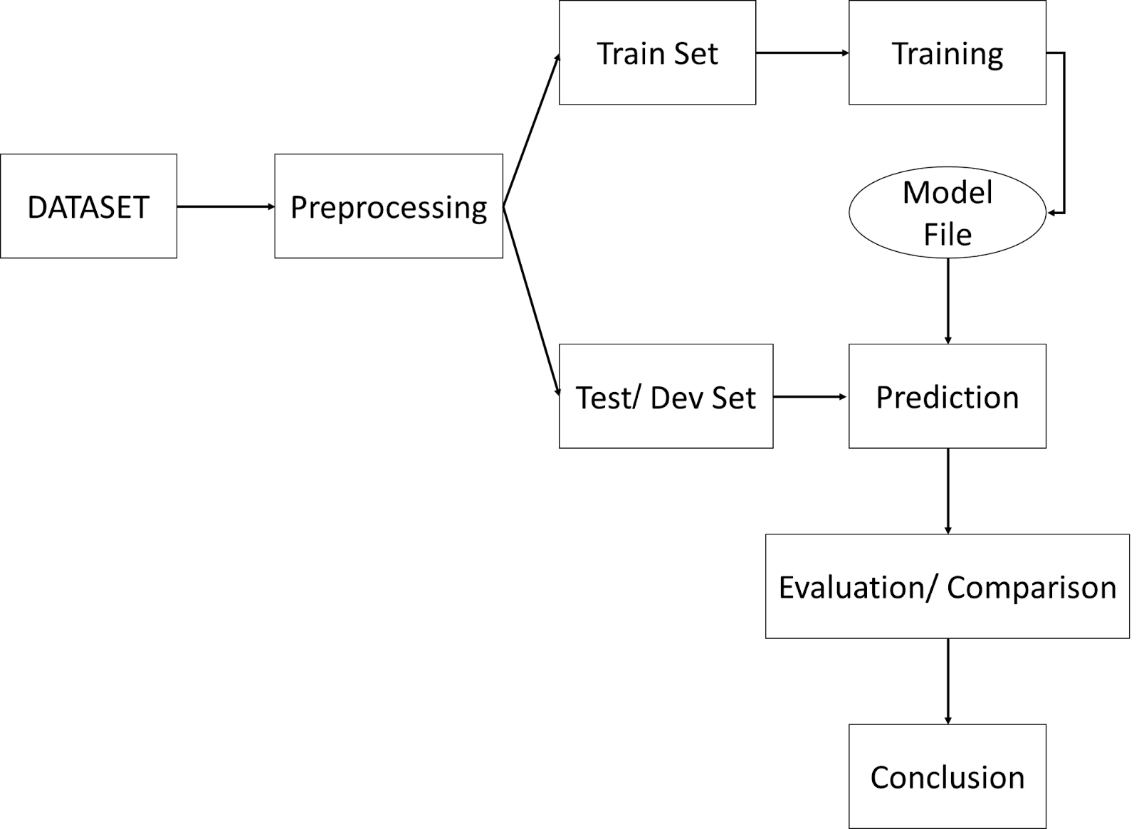
In this research, we investigate three different hybrid methods for developing a Recommendation System based on Collaborative Filtering:

* The first strategy is User-Item based Collaborative Filtering with clustering algorithms.
* The second method combines Matrix Factorization (MF) and Deep Learning.
* The final strategy is Self-attention Sequence-based Collaborative Filtering, which bases recommendations on sequential patterns and takes into account the temporal order of user-item interactions.

We plan to put these three various ways into practice in order to assess their effectiveness and determine which is best for the Recommender System. Through this study, we want to learn more about Collaborative Filtering methods and how they might be used in real-world Recommendation Systems.

The rest of this report us divided into the following sections. Section 2 describes our project process, provides some dataset information, analyzes and visualizes our data. We then introduce the techniques and modeling approaches employed in three hybrid CF-based recommender systems and their experiment designs. In Section 3, we summarize the conclusion and finally give project references in Section 4.

# Project process

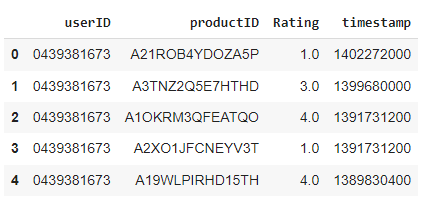


Picture 5. Project Process

## Dataset

In this study, the Video game dataset of Amazon [5] is used for evaluating the proposed algorithms.

This dataset includes four features: userID, productID, rating, and timestamp. There are 2489395 ratings provided by 71982 users on 1540618 items.



Picture 6. Video game dataset

## Data Analysis

Having the dataset, our first step is to clean and preprocess the data in this project before we can analyze and visualize it. We found no missing values but there are 75,954 duplicates user-item pairs in the dataset, which means that a user can rate the same item at different times. The data is sorted by timestamp, duplicates are removed, and the most recent interaction for each user-item pair is taken.

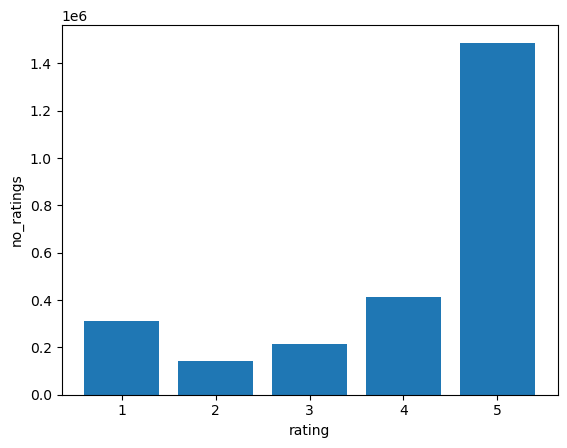


Figure 1. The distribution of the ratings

The figure above examines the distribution of the ratings in the dataset after cleaning the data. This bar chart shows the overall tastes of users, which are shown here are pretty high since more than half of the ratings are 5 stars which might affect explicit models since it will be imbalance.

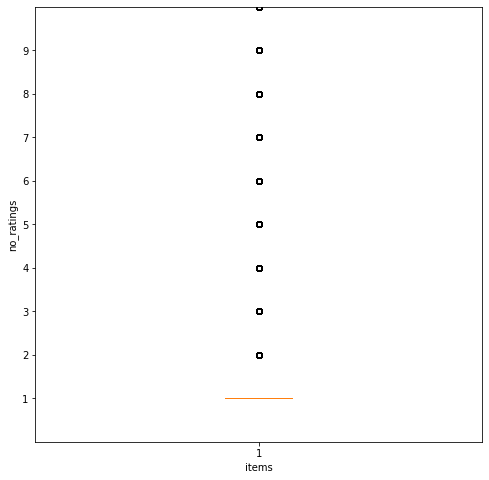
* *Represent the dataset by boxplot*

Figure 3. The number of ratings of one item

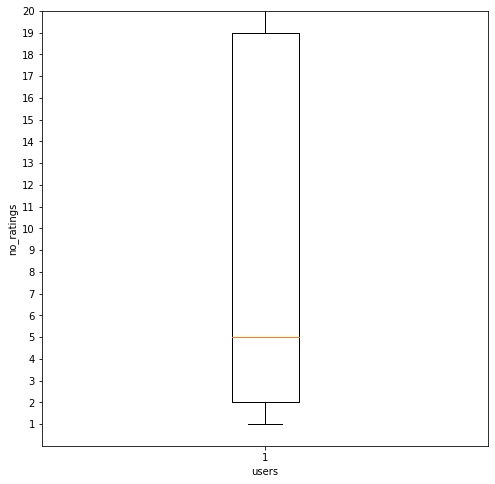


Figure 2. The number of ratings of one user

The boxplot in figure 2 reveals that the 25% of users have rated less than 3 items, as indicated by the 25th percentile values. This finding suggests that a significant portion of the user base may be less engaged with the platform, and it may be necessary to implement strategies to encourage more user activity and feedback. Additionally, the boxplot of item ratings indicates that more than 75% of the items have only been rated once, making them helpless to the dataset. This information is crucial for building an effective recommendation system as it highlights the need to focus on items with multiple ratings to provide more accurate and relevant recommendations to users.

* *The number of users by the number of ratings*

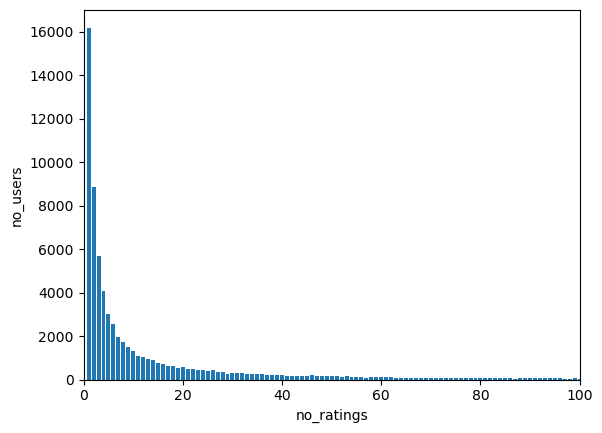


Figure 4

This bar chart reveals that there are 16,191 users who have only rated items once in the system. These users can be removed since they do not bring much value to our CF-based recommendation system. Interestingly, there is still one user who provided a significantly high number of ratings as 7,630 ratings is the highest number of ratings by a user.

* *The number of products by the number of ratings*

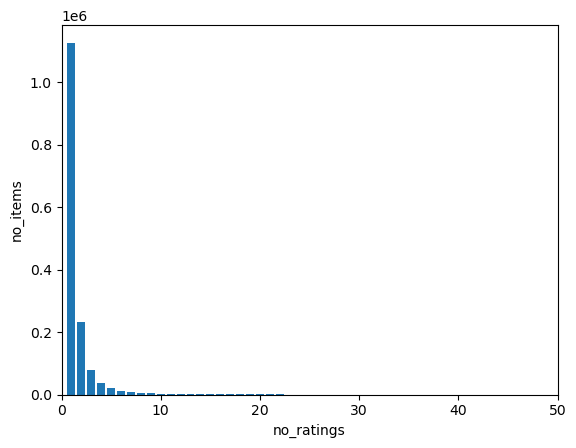


Figure 5

This bar chart reveals that there is a total of 1,127,396 unique products that have only received a single vote. However, there is one particular item that has stood out with a staggering 888 votes. It is important to consider the vast number of products that have received only a single vote, which also does not bring too much value to our CF-based recommendation system.

* *Visualize by scatter plot*

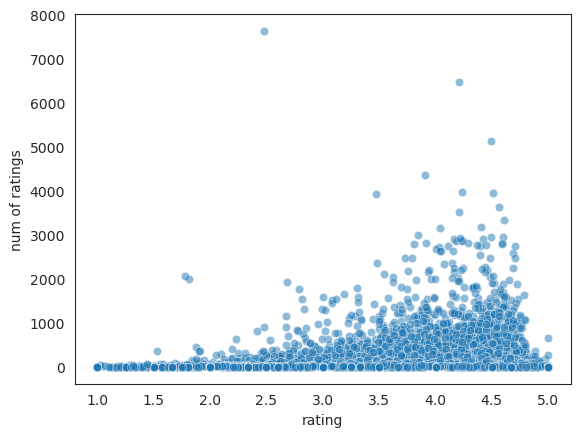


Figure 6

Figure 6 reveals that users who have provided multiple ratings tend to have a higher average rating. This finding suggests that users who rate frequently are more likely to be loyal and enthusiastic users who trust and appreciate the items available on the platform.

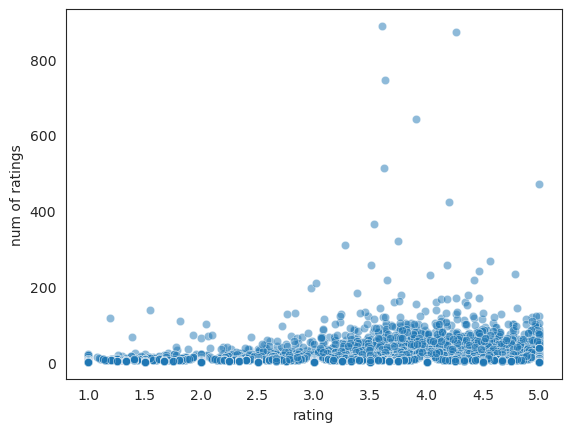


Figure 7

The scatter plot above shows the correlation between an item's average ratings and the number of ratings by users. Most popular items (those with high ratings) tend to be rated highly by users, which reflects the practical trend that highly rated items are recommended more often. This is because items with high ratings are perceived to be of good quality, and hence are preferred over those with lower ratings.

## Model

### Hybrid User-Item based Collaborative Filtering [6]

* *Introduction*

There are two common types of Collaborative Filtering algorithms: user-based CF and item-based CF. User-based algorithms seek for users that have similar preference patterns (in terms of rating products) to the person in question and propose items that have received high ratings from these similar users. Item-based algorithms, on the other hand, analyze the similarity between objects rather than users. They suggest things to the user that are similar to other items that the user has rated highly. These two strategies are quite popular since they are simple to implement and produce outcomes that are easily explained. They do, however, confront significant challenges: the user-based technique for comparing and detecting similarities across users is computationally difficult. Scalability and sparsity reduce the quality of recommendation by making it difficult to determine how similar users are.

To improve the accuracy of our recommender system, we apply a hybrid user-item based CF to produce a more customized product suggestion for a user while solving the typical challenges of data sparsity and scalability. CBR and average filling are employed to lessen the sparsity of the user-item matrix. Pre-grouping users with similar qualities into clusters by Kmeans can help with scalability by reducing the number of users that must be utilized when making recommendations. Following sparsity reduction and K Mean clustering, the closest cluster for a target user is identified, and then calculated item similarity in each cluster is applied to predict ratings.

* *Model explanation*

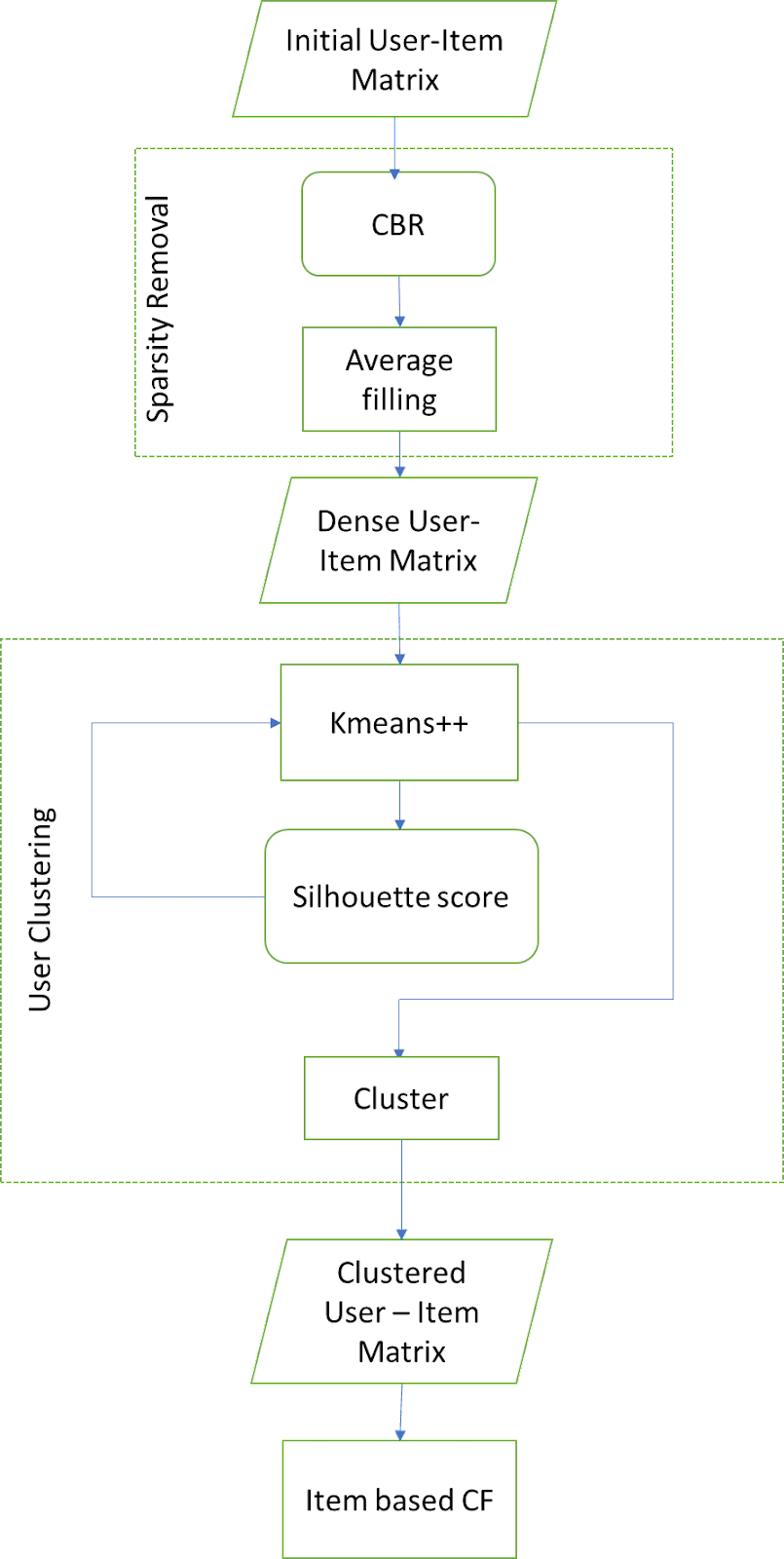
****

Figure 8. Hybrid user-item based Collaborative Filtering

*User – Item Matrix*

Firstly, in order to apply our model, we must convert the dataset into a user-item matrix, with the columns representing our unique productID and the rows representing the userID. Then we normalize the rating by obtaining the average rating of each user because some people tend to give greater ratings than others. After normalization, items with ratings lower than the user's average receive a negative value, while items with ratings higher than the user's average receive a positive value.

*Sparsity Reduction*

Next step is Sparsity Reduction. We will use CBR followed by average filling to fill the Nan value. Because we presume that similar users have similar ratings for the same things, CBR is an easy strategy to utilize here because it assumes that similar circumstances require similar answers. Individual user ratings create the cases, and the similarity between two users is determined by how similar their rating patterns are. Pearson correlation is the similarity metric employed in this case. Next, for each target user, we will generate a list of the top K similar users based on the similarity scores between this person and the other users.

We then utilize the obtained cases to fill in the target user's nan value. The rating for each unrated item is approximated based on the rating of this item by these similar users. This is calculated as follows:

Where is the rating of the item t by user i; is the similarity between user i and user u; k is the number of similar users under consideration.

It is conceivable that all users have not rated the target item, in which case the unrated item will still remain nan after employing CBR. These empty cells are subsequently filled using average filling:

Where is the rating for item i by user u, and k is the number of other items currently rated by user u.

*User Clustering*

Third step is user clustering. We have a dense user-item matrix after sparsity reduction. We then cluster users using Kmeans++.

K-means method is sensitive to the initialization of the centroids or mean points, while Kmeans++ use sampling to identify initial cluster centroids based on an empirical probability distribution of the points' contribution to overall inertia. This method also accelerates convergence.

In order to find an optimal number of clusters, we can use silhouette analysis. The silhouette coefficient or silhouette score for clustering technique is a measure of how similar a data point is within-cluster (cohesion) compared to other clusters (separation).

Now, based on the Pearson correlation, we next compute the item-item similarity matrix for each cluster.

*Item-based CF*

For the recommendation part, for an individual user, we first determine the cluster that is closest to the user based on the Euclidean distance between the user and the cluster centers. The top most comparable items for an unrated item t for this user are then found using the item-item similarity matrix for this cluster.

The prediction, using the weighted average formula, of user u’s rating for t is:

Where is the rating for item t by user u; is the similarity between item i and item u; k is the number of most similar items under consideration.

Since the user-item matrix have been normalized, in order to predict the user’s ratings, we must add the user's average item rating score back to the item score. Then, the top things predicted with high ratings will then be recommended to the user based on the predictions.

* *Pseudocode*

*Sparsity removal:*

* Input: user-item matrix
* Output: dense user-item matrix
* Steps:

+ Calculate the similarity between users using Pearson

+ Sort list of top k similar users

+ For each unrated item, calculate estimated rating

+ Use average filling for remain nan value

*Kmeans:*

* Input: dense user - item matrix filled by CBR and average filling
* Output: sets of k clusters and similarity matrix of each cluster
* Steps:

+ Divide dense item-base matrix into K clusters

+ Optimal K by silhouette score

+ Calculate item similarity inside each cluster

*Item-based filtering:*

* Input: clustered user-item matrix, target user
* Output: predicted rate for unrated items
* Steps: for each user in test set

+ Find the closet cluster for target user

+ Predict the removed rating using the weighted sum

+ The user will be recommended the top items predicted with high ratings.

* *Complexity:* with m: the number of items, n: the number of users

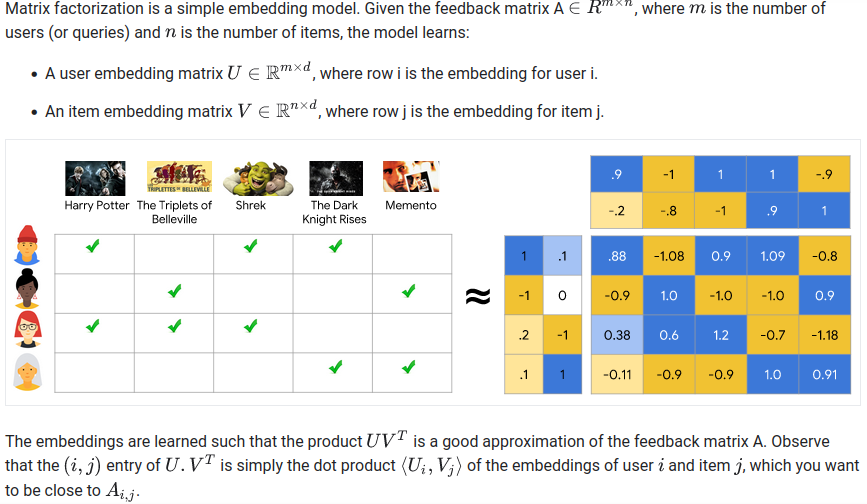
+ Time complexity:

+ Space complexity:

### Matrix Factorization and Deep Learning [7]

* *Introduction*

A common method in collaborative filtering for creating recommendation systems is matrix factorization (MF):

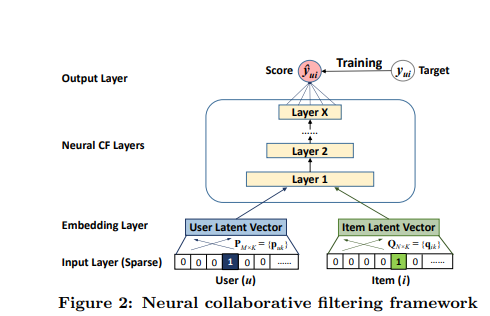
 [8]

Original matrix is decomposed into much smaller 2 matrices each representing all users and items such that the rating of original matrix Rij approximately equal to the dot product U(i) and I(j).

When applied to neural collaborative filtering (NCF) models, the fact that MF can only represent linear correlations between user and object attributes is one of its key drawbacks. This could limit its ability to accurately anticipate ratings or interactions because it may not be able to capture more intricate, non-linear links between user-item interactions. Another drawback of MF is the fact that MF in NCF models makes the assumption that all users and items are equally important, which may not always be the case in real-world circumstances where certain users or products may be more influential than others.

This model makes use of deep learning techniques to simulate non-linear interactions and capture the significance of certain people and items in order to get over these restrictions. Deep learning has been shown to be a really good model in many fields such as fraud detection, computer vision, image classification, OCR, ... because its ability to capture complex relationships between data features. Deep learning and MF, two widely used techniques, are combined in this model.

* *Model explanation*



Users and Items still go through embedding layers to be projected to smaller latent space with the same idea as MF: each user and item will be represented as k-dimension vector with k is the embedding size

Then instead of just getting the dot product as the output like in normal MF, an MLP neural network with hidden layers receives these vectors as input to train on the combined characteristics of the user and the product. By minimizing the loss function between the predicted output and the actual output of the pairs (user, item) in the training set, the NCF model can be trained.

* *Pseudocode*
* Users and items go through embedding layers for each to project to latent space. Then, concat 2 latent spaces together to be the first layer of neural network.
* Create 3 more hidden layers:

+ Each layer reduces half the electrons

+ 2 first hidden layer goes with a drop-out rate

+ The number of perceptron in the last layer determine number of related factors between users and items

* Activation function for the hidden layers

+ activation = 'relu'

+ weights initialization = ‘he\_uniform’

* Output layers:

+ 1 neuron

+ Activation = ‘sigmoid’

+ Weights initialization = ‘glorot\_uniform’

* Loss function and optimizer for the model:

+ Optimizer = keras.optimizers.Adam

+ loss = binary-cross-entropy

* *Complexity:* e is embedding size and i, u are respectively the number of item and user

+ Time complexity:

+ Space complexity:

### Self-attention model [9]

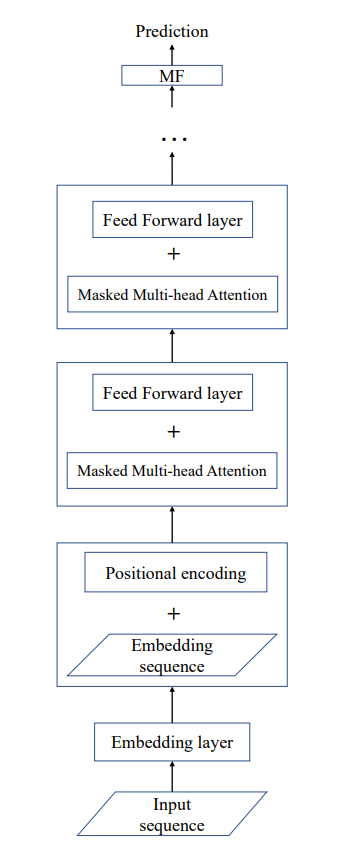


Figure 9. Flow chart of Self-attention Model

* *Introduction*

Most traditional Recommendation models try to take advantage of existing ratings to predict missing ratings. However, these models often do not use data about the time a user rate a product, thus, this can give recommendations contrary to the user's current preferences. And to address this problem, many sequence-based models are born. Many of these models use similar ideas to what we have in Natural Language Processing because of the correlation in the analyzing sequence of items and the sequence between words. And one of the simplest models to follow this trend is the use of RNN (with the support from GRU or LSTM). However, the drawbacks of these models are that they cannot handle a sequence in parallel, but have to handle it item by item. This increases the time complexity of the whole model, and the final result might not be as good as expected. Therefore, in this project, we propose a model that can solve this problem by using self-attention mechanisms.

* *Model explanation*

Embedding Layer: Similar to NLP models, we can use an embedding layer as the first layer. If this is not applied, the only way to represent items as vectors is to use one-hot encoding vectors. This will make storing and calculating vectors very difficult when the dimension of the vector is equal to the number of items in the dataset. At the same time, representing items as one-hot vectors will not be able to represent the relationship between items and the model would not work effectively. Using embedding layers will not only help reduce the dimension of the vector representing items, but also be able to represent more relationships between items.

Multi-head attention: After obtaining embedding vectors, we can proceed to use self-attention layers. The self-attention layer will assign each item in a sequence the vector query(q), key(k), and value(v). And then, we can calculate the attention of the item at “j” position to the item at the “i” position by using the following operations:

* Compute key-query affinities:
* Compute attention weights from affinities (softmax):
* Compute outputs weighted sum of values:

Overall, a formula can generalize on a sequence that:

+ dk is the dimension of the vector k, or in our case, it’s the size of an embedding vector. We use it just to scale the dot product and make sure that it does not explode.

+ softmax here denotes a row-wise softmax normalization function. Thus, every item depends on other items in the same sequence.

And to create those three matrices Q, K, V. We simply put our input sequence through three fully connected layer(with no bias) and the formula is as follow:

But self-attention layer will only give us only one single perspective of the sequence. And so if we want to extract the information from the sequence from many views, it’s where the multi-head attention pops up. Multi-head attention use the same procedure as the self-attention mechanism. The only difference is that we will decompose the Q, K, V matrices into multiple ‘heads’ at first, and then after carrying out self-attention on all those decomposed matrices, we can finally concatenate them at the final step. Notice that using multi-head attention won’t increase the complexity of the model.

* No sense of position: Unlike traditional RNN where we just keep unrolling the sequence item to item, in a multi-head attention layer, we input all the items at once. And thus, there is no sense of sequence in the input. And to solve this problem we can simply add a positional encoding right after the embedding layer. The formula is as follow:
* Lack of nonlinearity: As we can see from all the formulas above, there is no nonlinearity in the formula of self-attention model, it’s all just weight average. Or in other words, stacking more and more self-attention layers will only re-averages the value vector. To address this problem, we can simply put a simple feed forward layer after the self-attention layer with ReLU activation function.
* Masking the future: If we only use the model above, you can see that an item might get information from the latter part of the sequence. This is not what we want since we want the model to predict the sequence. So in order to prevent this ‘peaking’ behavior and to enable parallelization, we mask our attention to future words by setting attention score to negative infinity.

And after many blocks of multi-head attention and feed forward layers, we can add a final prediction layer using the following formula:

Where is the result after b block, and is the embedding matrix.

By minimizing the loss function between the predicted output and the actual targe in the training set, the self-attention can be trained.

* *Pseudocode*
* Let the sequence of a user go through embedding layer to project to latent space
* Add positional encoding to the embedding result
* For each block:

+ Create normalization layer

+ Create a masked multi-head attention layer

+ Create normalization layer

+ Create a linear layer with ReLU activation

+ Create a linear layer

* Create normalization layer
* Create a matrix factorization layer for prediction
* Loss function and optimizer for the model:

+ Optimizer: Adam optimizer

+ Loss: Binary cross entropy

* *Complexity*
* Time complexity: The complexity of the whole model is *where n is the sequence length and d is the embedding size.*
* Space complexity: The space complexity is *where i is the number of items*.

## Experiment design

### Evaluate metrics

In this report, we use three ways to evalute the performance of methods:

* Mean Absolate Error (MAE) measures errors between actual values and values predicted by the model. The lower the MAE is, the higher the prediction is, and the better the performance of the algorithm is.
* The HR@10 intuitively measures whether the test item is present on the top-10 list.
* The NDCG accounts for the position of the hit by assigning higher scores to hits at top ranks.

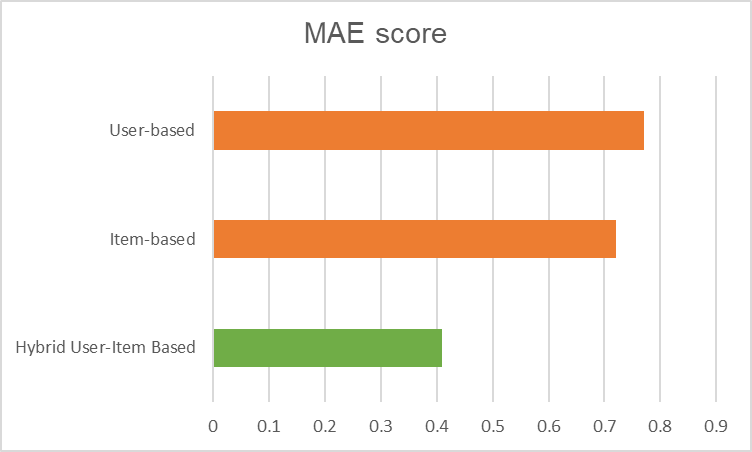
### Hybrid User-Item based Collaborative Filtering

* *Preprocess dataset*
* Because our model only uses ratings to predict, we may remove the ‘timestamp’ column.
* Drop out users have rated less than 20 items and products have less than 20 rates in order to confirm that the quantity of users also meets our requirements.
* Normalize the dataset
* The dataset is partitioned into k equal-sized subgroups in k-fold validation.
* With k = 5, 20% of data is used for validation and 80% for training data.
* *Train the model*
* In each iteration, fill the 80% training data using CBR and average filling.
* Apply Kmeans to divide the dataset into clusters and calculate item-item similarity for each cluster.
* For each user in the validation set, find the closest cluster and predict removed rating by item-based CF.
* *Evaluate*
* In the test set, randomly drop 5 values of each user.
* Use model to predict the dropped items
* Calculate MAE score:

+ Kfold:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
| 0.41171 | 0.4215 | 0.47094 | 0.35557 | 0.40031 |
| 0.41201 | | | | |

+ Compare with traditional Memory Collaborative Filtering:



The graph above indicates that MAE score of hybrid user-item based CF is

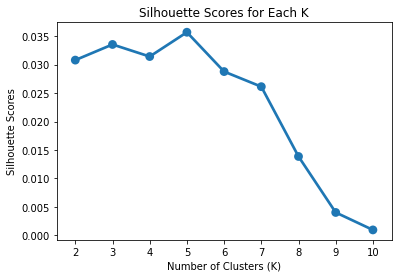
considerably lower than one of two traditional CF model. That means the hybrid model have predicted ratings more correctly.

* Parameters for tuning:

+ Tuning top similar users and top similar items by MAE score:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Similar items  Similar users | 5 | 10 | 15 | 20 |
| 5 | 0.44656 | 0.45986 | 0.78323 | 2.052546 |
| 10 | 0.45795 | 0.48017 | 0.62998 | 0.83588 |
| 15 | 0.41826 | 0.50596 | 0.6750 | 1.842 |
| 20 | 0.41201 | 0.46975 | 1.61628 | 0.934 |

+ The number of clusters (k in k means):



From the two figures above, it is clear that the best parameter for our model in this case will be: the top 20 similar users; the top 5 similar items, and the  
number of cluster for Kmeans equal 5.

### Matrix Factorization and Deep Learning

* *Preprocess dataset*
* Filter out the irrelevant users and items (item < 7, user < 10 interactions), we also remove the ‘timestamp’ column.
* Keep ratings ≥ 3 only and convert to 1 indicate \*has interaction\*
* Select latest interaction of each user as the test set
* Randomly select 1 interaction from each user as validation set to tune parameter
* Convert users and items ids to number
* *Train the model*
* For each positive instance, generate some negative instances to balance the train dataset
* Minimize the log-loss, use 0.2 size of train set as validation for early-stop
* *Evaluate the ncf\_model*
* For each user in the validation set, randomly select 99 non-interact items + 1 item of that user
* Use the model to predict chances of interactions of those 100 items
* Calculate HR@10 and NDCG@10 based on those 100 items
* Parameter for tuning:

+ Negative ratio (how many negative instances per positive instance) (we chose [4, 6, 8])

+ Last-layer number of perceptron → determine also the embedding size (we chose [8, 16, 32])

+ Drop-out rate (we chose [0, 0.25, 0.5])

+ Learning-rate for Adam optimizer (we chose [0.001, 0.005, 0.01])

+ batch-size for train set (we chose [1024, 2048])

* Here are some results on the validation set (only the best result on different negative ratios and last-layer sizes are shown):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Negative ratio** | **Final size** | **Batch size** | **Learn rate** | **Drop rate** | **HR@10** | **NDCG@10** |
| 4 | 8 | 2048 | 0.01 | 0.25 | 0.568 | 0.33 |
| 4 | 16 | 1024 | 0.005 | 0.5 | 0.604 | 0.3516 |
| 4 | 32 | 2048 | 0.005 | 0.5 | 0.603 | 0.352 |
| 6 | 8 | 2048 | 0.01 | 0.25 | 0.61952 | 0.36777 |
| 6 | 16 | 2048 | 0.001 | 0.5 | 0.62721 | 0.37346 |
| 6 | 32 | 2048 | 0.005 | 0.5 | 0.62639 | 0.37556 |
| 8 | 8 | 1024 | 0.005 | 0.25 | 0.611 | 0.365 |
| 8 | 16 | 1024 | 0.001 | 0.5 | 0.6334 | 0.378 |
| **8** | **32** | **2048** | **0.005** | **0.5** | **0.6464** | **0.3933** |

Test the best hyper-parameters set on the test set we get HR@10 = 0.4732 and NDCG@10 = 0.26. We also try the traditional MF, tuning hyper-parameters and get the HR@10 = 0.3953 and NDCG@10 = 0.2202.

### Self-attention model

* *Preprocess dataset:*
* Filtering out the irrelevant users and items (item < 7, user < 10 interactions)
* Keep ratings ≥ 3 only
* Select the latest interaction of each user as the test set, and then for the remaining dataset, we also choose the latest interaction of each user as the validation set.
* For the training set, we will transform the buying history of each user into a sequence.
* Convert users and items ids to number
* *Train the model:*
* For each positive instance, generate a negative instance for a better performance of the model.
* Minimize the log-loss.
* *Evaluate*
* For each user in the validation\_set, randomly select 99 non-interact items + 1 item of that user
* Use the model to predict chances of interactions of those 100 items
* Calculate HR@10 and NDCG@10 based on those 100 items
* Hyperparameter for tuning:

+ Learning rate

+ Embedding size (have to modify to be divisible by number of heads)

+ Drop-out rate

+ Sequence length

+ Batch-size for training set

+ Number of blocks

+ Number of heads (for multi-head attention layer)

* Below are some of the results we got on validation set (using best result for each hyperparameter):

|  |  |  |
| --- | --- | --- |
|  | NDCG@10 | Hit rate@10 |
| Learning rate = 0.01 | 0.4267 | 0.65 |
| Learning rate = 0.001 | 0.4228 | 0.6549 |
| Learning rate = 0.0001 | 0.2092 | 0.3611 |
| Embedding size ~ 50 | 0.3335 | 0.5614 |
| Embedding size ~ 100 | 0.4228 | 0.6549 |
| Embedding size ~ 200 | 0.4158 | 0.6508 |
| Number of blocks = 1 | 0.3816 | 0.608 |
| Number of blocks = 2 | 0.4228 | 0.6549 |
| Number of blocks = 3 | 0.2679 | 0.4648 |
| Number of heads = 1 | 0.3335 | 0.5614 |
| Number of heads = 2 | 0.357 | 0.5855 |
| Number of heads = 3 | 0.4228 | 0.6549 |
| Number of heads = 4 | 0.3541 | 0.5855 |
| Sequence length = 30 | 0.3659 | 0.5988 |
| Sequence length = 40 | 0.357 | 0.5855 |
| Sequence length = 50 | 0.3732 | 0.6029 |
| Sequence length = 60 | 0.4228 | 0.6549 |
| Batch size = 128 | 0.42 | 0.6415 |
| Batch size = 256 | 0.4228 | 0.6549 |

Therefore, the best combination is with batch size = 256, learning rate = 0.001, sequence length = 60, embedding size = 99, the number of blocks = 2, number of heads = 3, dropout rate = 0.4. And using this on the test set, we got NDCG@10 = 0.345, HR@10 = 0.5586.

### Compare three hybrid Collaborative Filtering model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MF | Hybrid user-item based CF | MF & Deep Learning | Self-attention |
| NDCG@10 | 0.2202 | 0.2113 | 0.248 | 0.345 |
| Hit rate@10 | 0.3953 | 0.4115 | 0.4601 | 0.5586 |

The above table, as well as the MAE score chart given in the evaluation section of the Hybrid user-item based model, clearly illustrate that hybrid methods provide better recommendations compared to traditional ones. Among them, sequence-based methods stand out with the best Hit rate and NDCG. This is because sequence-based methods take into account the current preference of users on items, making them more effective in generating accurate recommendations.

# Conclusion

Nowadays, recommendation systems have become a valuable tool for information filtering, providing users with relevant information based on their needs and preferences. Among the various techniques used in recommendation systems, collaborative filtering (CF) has gained significant popularity. CF works by identifying patterns of similarity in past interactions between users and items to make personalized recommendations. However, traditional CF techniques often suffer from certain limitations, such as the cold start problem, data sparsity, and scalability issues. To overcome these limitations, hybrid CF approaches have emerged as a promising solution. Hybrid CF combines different recommendation methods to mitigate the drawbacks of individual methods, resulting in more accurate and personalized recommendations. These systems enable businesses to better understand their users' needs and preferences, and help users to efficiently navigate large volumes of information to find the most relevant and useful content.

In this project, we have implemented and evaluated three different hybrid collaborative filtering-based recommendation techniques. The first technique is designed to solve the scalability and sparsity issues commonly encountered in traditional memory-based systems by using imputation and clustering. This approach allows us to fill in missing data and group similar users or items together, resulting in more accurate and personalized recommendations. The second technique we employed uses deep learning to represent more complex relationships between users and items than the traditional linear relationships found in matrix factorization (MF) models. By using deep learning, we can capture nonlinear interactions and dependencies between users and items, resulting in more accurate and diverse recommendations. Finally, the third technique takes into account the current preferences of users by using a self-attention model commonly used in natural language processing (NLP). This approach enables us to generate recommendations that are tailored to each user's current preferences, resulting in more accurate and relevant recommendations.

However, while hybrid CF techniques have shown promise in overcoming some of the limitations of traditional collaborative filtering, there are still some issues to be addressed. For instance, the first method we implemented to solve the sparsity and scalability problems requires a large amount of memory resources to store the filled user-item matrix. The imputation and clustering approach used in this method fills in missing data and groups similar users or items together. While this can result in more accurate and personalized recommendations, it also requires a significant amount of memory to store the complete user-item matrix. This can be problematic for datasets with a large number of users and items, and may not be feasible for systems with limited memory resources.

When it comes to deep learning approaches in recommendation systems, the number of parameters and training time increases proportionally with the number of users and items. This means that as the dataset gets larger, the amount of computational resources required for training also increases. Moreover, deep learning models often have a large number of hyper-parameters that need to be tuned in order to achieve optimal performance. Tuning these hyper-parameters can be a time-consuming process that requires further knowledge and expertise.

In conclusion, recommendation system is an evolving field and there are still lots of novel techniques as well as novel approaches to improve current techniques that we would love to test in further project.

# References

|  |  |
| --- | --- |
| [1] | [Online].https://qymatix.de/en/recommendation-systemsnowadays/. |
| [2] | [Online].https://www.themarketingtechnologist.co/building-arecommendation-engine-for-geeksetting-up-the-prerequisites-13/. |
| [3] | K. &. A.-J. H. Almohsen, "Recommender Systems in Light of Big Data," *International Journal of Electrical and Computer Engineering (IJECE),* vol. 5, no. 10.11591/ijece.v5i6.pp1553-1563, pp. 1553-1563, 2015. |
| [4] | X. a. T. M. K. Su, "A Survey of Collaborative Filtering Techniques," *Adv. Artif. Intell. ,* pp. 421425:1-421425:19., 2009. |
| [5] | J. L. J. M. Jianmo Ni, "Justifying recommendations using distantly-labeled reviews and fined-grained aspects," *Empirical Methods in Natural Language Processing (EMNLP),* 2019. |
| [6] | Z. F. Nitin Pradeep Kumar, "Hybrid User-Item Based Collaborative Filtering," *Procedia Computer Science,* vol. 60, no. ISSN 1877-0509, pp. 1453-1461, 2015. |
| [7] | X. e. a. He, "Neural Collaborative Filtering.," *Proceedings of the 26th International Conference on World Wide Web ,* p. n. pag, 2017. |
| [8] | [Online].https://paperswithcode.com/paper/meta-prod2vecproduct-embeddings-usingside?fbclid=IwAR319NXRs6OvGoSzXyPZxCrBXgPs-. |
| [9] | W.-C. a. J. M. Kang, "Self-Attentive Sequential Recommendation," *IEEE International Conference on Data Mining (ICDM) ,* pp. 197-206, 2018. |