HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Capstone project report

Games recommendation system

Submitted by

Doi Sy Thang - 20225528

Ngo Duy Dat - 20225480

Ta Ho Thanh Dat - 20225482

Nguyen Nhat Minh - 20225510

Nguyen Minh Quan - 20225520

Guided by

Assoc.Prof. Than Quang Khoat

Hanoi, May 2024

Abstract

Recommender systems represent a specialized field within machine learning, characterized by its unique attributes and evaluation methods. In this report, we aim to present fundamental approaches and initial perspectives on constructing a recommender system. We will employ methods specifically tailored to address this problem, such as the Content-Based Model (CB), which focuses on comparing the similarity between users or items, inspired by the human tendency for imitation. Additionally, we will explore basic machine learning models, such as the linear model, to experiment with recommender systems. Furthermore, we propose the use of Collaborative Filtering, featuring two main models: the Neighborhood-based model (NB) and the Latent-factor model (LF). While the NB model leverages user-user and item-item similarities based on ratings rather than attributes, as in the CB model, the Latent-factor model will find latent features based on observations, some approaches are implemented such as matrix factorization models, factorization machines, and certain deep learning techniques.

Our project will utilize game data from Steam, instead of widely known datasets like MovieLens or Netflix films. This dataset includes features that can be considered labels, such as "is recommended"(implicit feedback) features and "hours" features. We will propose a strategy to combine these features to generate the most reasonable ratings possible, which we term explicit feedback. Furthermore, for simplicity, our project will not address the cold-start problem but will focus on resolving issues using a warm-start approach.

Contents

Part I
Introduction

1 Motivation

Before learning about machine learning, we used to joke about the "eavesdropping" of technology when saying the phenomenon that searching for a product would immediately result in advertisements for it, seemingly designed to empty our wallets. Therefore, through this course, we wanted to satisfy our curiosity and understand how AI subtly "extracts" money from us. This is achieved through the use of a recommender system, which suggests items based on our past consumption behaviors to appeal to others.

Recommender systems constitute a vast and long-standing field within machine learning. Their emergence and growth are closely tied to the sales and marketing industry, with companies investing heavily to boost their product sales by even fractions of a percent. From search engines using Cookies to promote relevant products, to music and film streaming platform using individually customized data to recommend, it can be seen that this technology plays a crucial part in automatizing processes and maximizing profit. Although this field may not be as exciting or disruptive as current trending technologies like chatbots (Large Language Models) or image generation models (diffusion models), it remains highly important and receives significant investment from businesses seeking to maximize their profits. Annually, these firms spend hundred thousands of dollars for recommendation system R&D, it is estimated that Netflix spend 150 million dollars each year to upgrade this technology. With such amount of investigation, it is understandable that this tool is able to suggest highly related products. Shall this game recommendation system works-it can suggest personally suited games, we can grasp the idea of the technique in general. This attempt laid the foundation for further delves into machine learning in general.

This interest led us to explore this new domain in machine learning. We aim to understand the basics of how products are rapidly recommended to us as we browse platforms like Facebook, TikTok, or various social media and e-commerce sites.

2 Problem formulation

The recommendation problem aims to improve user experiences on platforms through decisions to recommend items suitable for users. Therefore, the Machine learning algorithm focuses on the learning behavior of the user in the recommendation system. Approaching in recommendation problem is divided into two directions including content-based and collaborative filtering. This project's purpose is to develop a system that predicts user ratings and provides personalized game recommendations based on their past interactions. Our approach includes Content-Based Filtering (CB) to analyze game attributes and user profiles, Collaborative Filtering (CF) with Neighborhood-Based Models (NB), and Latent Factor Models (LF) to uncover hidden patterns in user-game interactions.

The project will utilize detailed Steam data, including user interactions and game metadata, to enhance the recommendation quality. We will evaluate the system's performance using metrics such as NDCG@k, F1@k, and Root Mean Square Error (RMSE), LogLoss. By focusing on the warm-start scenario, where initial user interaction data is available, we aim to generate accurate and explicit ratings without addressing the cold-start problem.

Despite the advantages, several challenges must be addressed. The cold-start problem, which involves recommending new users or new games without prior data, is not within the scope of this project. Scalability is another critical challenge, as the system must handle the vast number of users and games on Steam efficiently. Additionally, data sparsity, where users interact with only a small subset of available games, can hinder collaborative filtering methods' performance. By addressing these challenges, we aim to develop a robust and efficient recommender system that enhances user satisfaction on the Steam platform through precise and personalized game recommendations.

3 The Dataset

The dataset used in this project is derived from the Steam Game Dataset available on Kaggle. To address challenges related to data sparsity and scalability, a reduced dataset is employed, consisting of users with 40 or more interactions. This approach aims to enhance the reliability of the recommender system while ensuring computational feasibility and scalability.

3.1 Explanation of our dataset

users.csv: a table of user profiles' public information: the number of purchased products and reviews published. There are a total of 57973 users in the table. Here the list of table attributes:

Feature	**Explanation**
user id	User's auto-generated ID
products	Number of games that the user has purchased
reviews	Number of reviews that the user has published(it equals to the occurence of the user in recommendation data)

Table 1 . Features and Explanations for users

games.csv: a table of games with app_id as the key attributes and information about the games. The table has total of 50751 rows and 13 columns. Here the list of table attributes:

Feature	Explanation
app id	The id of the game
title	The title of the game
date release	Date released of the game
win	Does the game support Windows or not?
mac	Does the game support macOS or not?
linux	Does the game support Linux or not?
rating	This is rating based on assumption made by game developer of Steam on user reviews and positive ratio. It contains 9 extent level: 'Overwhelmingly Positive', 'Very Positive', 'Positive', 'Mostly Positive', 'Mixed',' Mostly Negative', 'Negative',' Very Negative', 'Overwhelmingly Negative'.
user reviews	Amount of user reviews available on the Steam(number of interaction for this game in recommendation data)
positive ratio	the percentage of recommended from user out of all interactions.
price final	Price in US dollars($) calculated after the discount
price original	Price in US dollars($) before the discount
discount	Discount percentage of the game
steam deck	Does the game support Steam Deck or not?

Table 2 . Features and Explanations of the Dataset

recommendations.csv: a table of user reviews: whether the user recommends a product. There is a total of 4 millions ratings here. The table represents a many-many relation between a game entity and a user entity. Here the list of table attributes:

Feature	Explanation
app id	The id of the rated game
helpful	How many users found the recommendation helpful
funny	How many users found the recommendation funny
date	Date of the recommendation published
is recommended	Whether the game is recommended by the user
hours	How many hours the user played the game
user id	User's anonymized ID
recommend id	Autogenerated ID for the recommendation

Table 3 . Features and Explanations of the Dataset

games_metadata.json: the file matches each game app_id to its tags(genres) and its description about the game.

3.2 Insights

At first glance, this dataset appears to have many features and is rich in information. However, after analyzing it and making several our knowledge-based assumptions, which we attempted to prove based on the data, we realized that not all assumptions were correct, and not all features were valuable. If not properly utilized, these features may merely act as noise, ultimately reducing the quality of our model.

users data Regarding the user data, we consider it a significant drawback. Unlike many other datasets that provide information such as age, gender, and occupation, this dataset only includes the number of games a user has purchased and the number of reviews they have evaluate. It could be explained that in the gaming sector, gamers often do not input accurate personal information like name and age, making such data noisy and unnecessary. Instead, the number of products and games purchased, which we can use to group users, proves to be more relevant and useful for our purposes.

games data Regarding the games data, there is a wealth of information to exploit. The most important aspects, in our opinion, are the tags (or game genres) and the game descriptions. Additionally, we have identified several other valuable information to utilize, such as the final price of the game and the ratings given by the developers based on their strategies. We have decided to exclude some information, such as Steam Deck compatibility (as nearly 100% of games support it, according to our survey). For operating system compatibility, games will be recommended to users based on matching operating systems, acting as a filter rather than game features. Other features, such as release dates and discounts, could also be useful; however, to keep the project simple, we have chosen not to analyze user trends over time or discount trends, thus excluding these aspects from our current scope.

recommendations data Regarding the recommendation data, we will only use two features: "is recommended" and "hours" to indicate the players' level of interest in an item. Initially, we attempted to include features like "funny" and "helpful" in our assumptions to generate a new rating. However, the information about "helpful" and "funny" proved to be unclear and insufficient.Our analysis showed that some games with a very high positive ratio and a large number of reviews still had some recommendations against playing them, which received significant "helpful" and "funny" votes. This indicates agreement from other users but doesn't provide a comprehensive view of the game. The dataset lacks depth, as we can't determine who agrees or disagrees with these helpful and funny votes, nor can we understand the text content of recommendations to know if users are criticizing or merely pointing out flaws in a popular game. Therefore, it is challenging to understand user behavior and game trends with limited information. Consequently, we decided to focus on the clearer features: "is recommended" and "hours".

3.3 Common data preprocessing

Rating transformation: In our dataset, there are two labels, True and False, but they are heavily biased towards True, with the number of True labels being eight times that of False

labels. This imbalance results in unbalanced labels for the recommender system. Fortunately, the dataset also includes the number of hours each person has played. Therefore, we proposed an assumption to convert this data into a rating format that reflects the degree to which each user evaluates the products based on the initial is-recommended label and their gameplay time compared to the average of all users.

Specifically, the is-recommended value is converted into -1, 1 instead of 0, 1, and we use quantiles to split gameplay time into 5 bins, with values ranging from 2 to 4 in increments of 0.5. The new rating is calculated as the sum of these two transformed labels. For example, if User 1 recommended Game A and played it for a total of 6 hours (falling into the second bin), the new rating would be $1 + 2.5 = 3.5$. Conversely, if User 2 did not recommend Game B but played it for 50 hours (in the fifth bin), the new rating would be $-1 + 4 = 3$.

This approach ensures that even if a user plays a game extensively but does not recommend it, or if they recommend it but spend very little time on it, the rating will still be around 3, just slightly above average. This method helps balance the data and provide a more nuanced evaluation of user preferences.

At this point, we will have two labels: is recommended (implicit rating) and transformed rating (explicit rating). The models we experiment will utilize one of these two types of data, depending on the suitability of the model to the specific problem type.

Part II
Content-based Filtering

1 Overview

Content-based filtering is a system that focuses on the features of items, rather than user behavior. Instead of looking at what other users liked, it analyzes the of items a user interacts with to suggest similar items. There are 2 main machine-learning approaches for content-based filtering:

Supervised learning: This approach leverages characteristics of the items with the user rating. The benefit of this approach is that it can help us deduce the user preference from their interaction history. Here, we need to build data such that each data point (X,y) has X representing the feature of the item and y being the rating of the user to that one. The feature of each item can be constructed by a technique of vectorization or word embedding. Our purpose is to predict the user rating for an item given that item's characteristics. Thus, the machine learning model can be used is varied, it can be either a classification (SVM, Random Forest, Bayesian model, ...) or a regression model (Random Forest, Linear Regression, ...)

Unsupervised learning: This method cannot extract users' preferences from their history to give personalized recommendations like supervised technique but its purpose is to filter out the most similar items to a specific item. This approach do not require user rating in the dataset as it only focus on the similarity between items. First, we also need to construct the feature of each item by vectorization or word embedding like supervised one. Once having the feature, we can simply calculate the cosine similariy between each pair of items or apply a clustering algorithm here to group similar objects together.

In this project, we will use some basic machine learning algorithms including KNN, Kmeans, Ridge Regression, Lasso Regression and Regression Random Forest for both approaches.

2 Content-based filtering schema

2.1 Data preprocessing

Our original dataset has a total of more than 50000 games .However, due to the limitation in computer resources and time, we only works on a smaller dataset which has 10000 games. We also works with users that have larger or equal than 30 ratings for those 10000 games, which there are 1214 users and 70951 reviews. The game dataset has many attributes but in the content-based system, we only focus on the attributes that affect user choices for games the most which are description and tags. The second step is to combine two 2 column tags and description to form a new attributes called "content". Then we will tokenize, remove stop words and do stemming. Tokenize means splitting the string to tokens. Stop words like "and", "but", "or" are redundant features so it is necessary to remove them. Finally, tokens need to be stemmed - turned into its base words. For example, such word "going", "goes" are transformed into "go". This is done by the method EnglishStemmer from the library nltk.

2.2 Word Embeddings

Word embedding is a representation of text in the form of real-valued vector. This is a crucial step before feeding the data into machine learning models. There are many ways to embed words: Count Vectorization, TF-IDF vectorization, Word2Vec, ...In this project, we choose the simple way: TF-IDF vectorization.

Term frequency-Inverse document frequency is a measure that quantify the importance of words, phrases, etc in a document amongst a collection of documents known as a corpus.

2.2.1 Term frequency

This measures the frequency of a word appears in a specific document. The formula is the raw count of a word divided by the total number of words in the document. High TF indicates a word is more popular in that document.

2.2.2 Inverse document frequency

This metric acts as a penalty for words that are too common in the corpus. IDF helps emphasize words that are more likely to convey the specific content or topic of a document. The formula for IDF is:

$$IDF(w) = \log_2(N/DF(w))$$

where N is the number of documents in the corpus and DF(w) is the number of documents that contain word w.

2.2.3 TF-IDF Vectorization

TF-IDF score of a word is simply the product of TF score and IDF score of that word. In the corpus, each document is represented by a high-dimensional vector where each dimension is corresponding to a word and the value of that dimension is the TF-IDF score for that word. The number of dimensions is either the number of unique word in the corpus or user-defined.

2.3 PCA for dimensionality reduction

The data we are dealing with is textual data which has a very big number of dimensions. Before going to the step of applying machine learning algorithms for this data, it is useful to conduct a dimensionality reduction process.

Dimensionality reduction is a group of techniques used in machine learning in order to reduce the number of features in the dataset but preserve most important information. In machine learning task, textual data often has very high number of dimensions and high dimensional data can leads to various problem. The first one is expensive cost of computer resources and computation time. Secondly, there can be many irrelevant features that lead to overfitting. Notably, some distance metrics are mush less useful in high dimensional space, especially the an distance. This is because in high dimensional space, there is a strange phenomenon where the distances between points tend to become very similar. This leads to the ratio between the nearest and farthest points approaching 1, essentially making all points uniformly distant. This can be refered as Curse of Dimensionality. It is proved that in high

dimensions, most datapoints concentrate on the surface of a hypersphere and the hypersphere is vastly empty. Since most points relatively lies on the surface of this "sphere", the additional dimensions do not contribute significantly in their distance. Therefore, the pairwise distance gradually becomes blur and less informative as the number of dimensions increase. This is a complex challenge for machine learning model that relies heavily on distance between data points such as KNN or KMean.

Hence, in this project, we apply a popular dimensionality reduction called PCA. PCA which stands for Principal Component Analysis is a method to create a set of vectors which we called Components that can represent the data. Here is the steps of PCA.

Normalization This step is to normalized data to ensure each feature contribute equally to the model. Each element value in a data point can be normalized as:

$$Z_{(normalized)} = \frac{Z - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation

Building Covariance Matrix A covariance matrix gives information about the variance of each feature (how it varies about its mean) which lies on the main diagonal of the matrix and the covariance between each pair of feature. The covariance here tells how a feature change when the other feature increase or decrease.

$$\begin{bmatrix} var(x) & cov(x,y) & cov(x,z) \\ cov(x,y) & var(y) & cov(y,z) \\ cov(x,z) & cov(y,z) & var(z) \end{bmatrix}$$

Figure 1 . A covariance matrix

Finding eigenvectors and eigenvalues Once we have a covariance matrix A with size NxN (N is the number of features in original data), we will need to find n eigenvectors X and n eigenvalues λ of A.

$$AX = \lambda X$$

Each eigenvectors here is corresponding to a component and the component with higher eigenvalue is considered to be more important as it captures more information about the data variance - how data spread.

Building feature vectors After obtain the components, we will decide which components to keep, which is where the number of dimensions is reduced. Normally, an explained variance ratio chart is draw to find out the most important components. It is a rule of thumb to keep the components that explained for a total of 0.8 variance. From that, we will obtain a new dataset with p features - each feature vector is corresponding to a component that we keep.

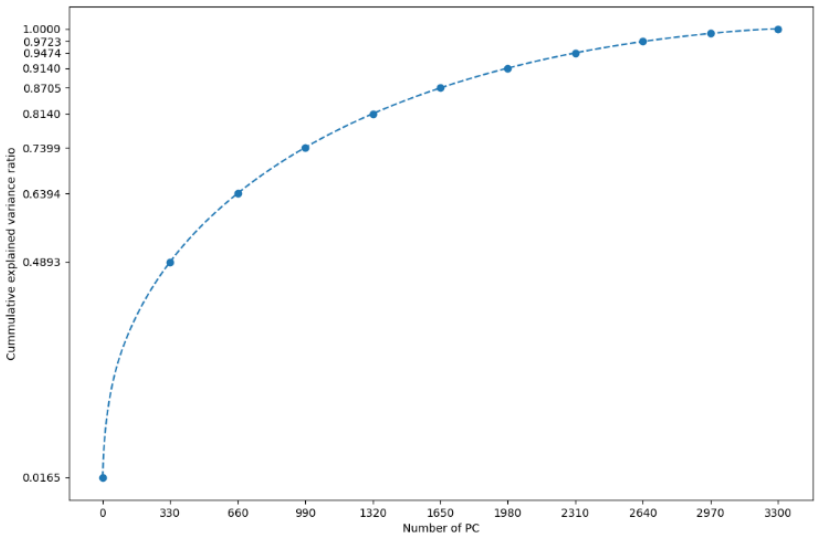


Figure 2 . Cumulative explained variance ratio graph

For example, in the project, we keep the first 1300 components as they account for a total of 80% of variance. Thus, the number of dimension has been reduced from 3300 to 1300.

2.4 Cosine similarity KNN

The idea here is to grab top most similar games to the interested game by approximating the similarity between each pair of games. One way to do that is calculating the cosine similarity between the TF-IDF vectors. The formula for cosine similarity is:

$$similariy(A, B) = cos(\theta) = \frac{AB}{|A||B|} \tag{1}$$

Cosine similarity is a common metric to measure the "similarity" of 2 vectors. The higher it is, the more similar between the directions of 2 vectors. Thus, a games B is considered to be more content-related to game A than game C if the cosine similarity between 2 tf-idf vectors of A and B is higher that that of A and C. Hence, we build a cosine similarity matrix C of size NxN for N being the number of games in the dataset(10000) where each element C_{ij} is the cosine similarity between game i and game j. Then, it is easy to retrieve top k games that is most content-similar to a target game.

2.5 Kmeans clustering

Kmeans clustering is a handy algorithm for the purpose of grouping similar content-based games into a same group. The only hyperparamter we try to tune here is the number of clusters k. As high-dimensional data is impossible to be visualized, here we need a metric for unsupervised learning.

2.5.1 Silouhette score

The silhouette score is a common metric used to evaluate the quality of clustering performed by K-means. It measures how well data points are grouped within their assigned clusters compared to data points in other clusters. The silouhette score is calculated for each data point and then taken average of all data points. The silouhette score of a point is calculated as:

$$\frac{b-a}{max(a,b)}$$

Where:

a is the average distance between the data point and all other data points within the same cluster (intra-cluster distance).

b is the distance between the considered data point to the nearest point but in a different cluster that the data point does not belong to (inter-cluster distance)

The range of this score is from -1 to 1. If the score is close to 1, that means b is much larger than a so the data point might be well clustered. The score approaches 0 as b is slightly larger than a which signifies that there could be overlapped clusters or the difference between clusters are not significant. A negative score tells that the data point is likely to be in a wrong cluster as b $<$ a.

Thus, ideally, we want the kmeans has a average silouhette score close to 1 and no point has a negative score. This metric is helpful to determine the number of clusters k. We do that by calculating the silhouette score for each value of k and take the k that maximize the score.

2.5.2 Evaluation

Here the graph representing silhouette scores corresponding to different values of number of clusters k.

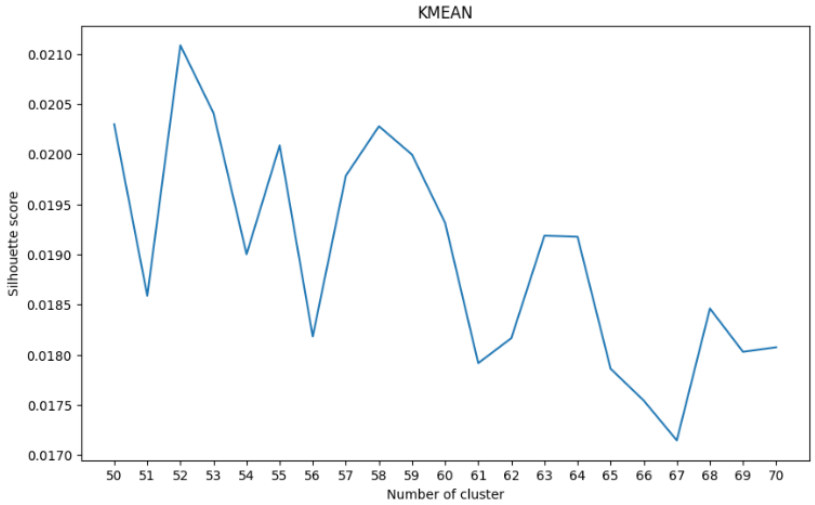


Figure 3 . Silhouette score evaluation before applying PCA

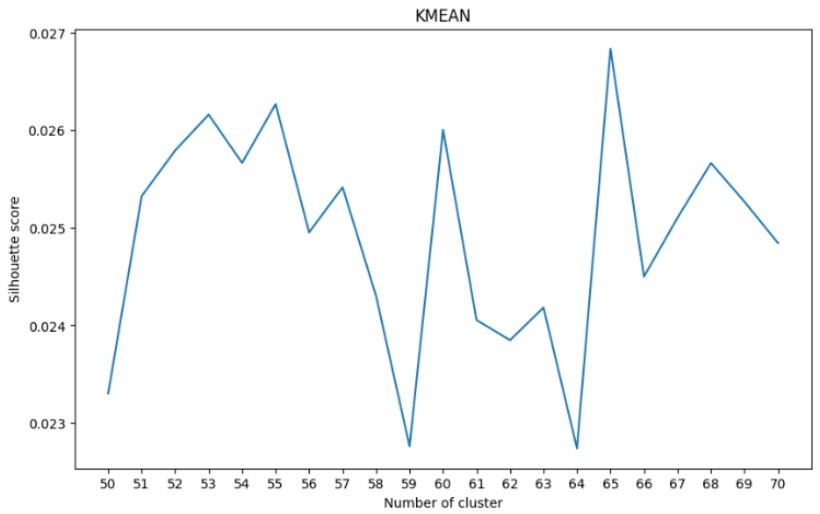


Figure 4 . Silhouette score evaluation after applying PCA

As can be seen from the graph, the silhouette score is very low, positive but highly close to 0 for all k. PCA does improve the score but does not much. This is likely due to the fact that Kmeans is not good at handling high dimensional data, the Euclidean metric cannot tell much about the distance between data points as the dimension number gets too high. Although we have conducted PCA, the number of dimension remains still very high. We will still choose k=65 as it maximize the score and then use some domain knowledge to test the algorithm performance. Main topics of each cluster can be extracted by getting the top terms (the words corresponding to the element having highest value in tf-idf vector) in cluster center.

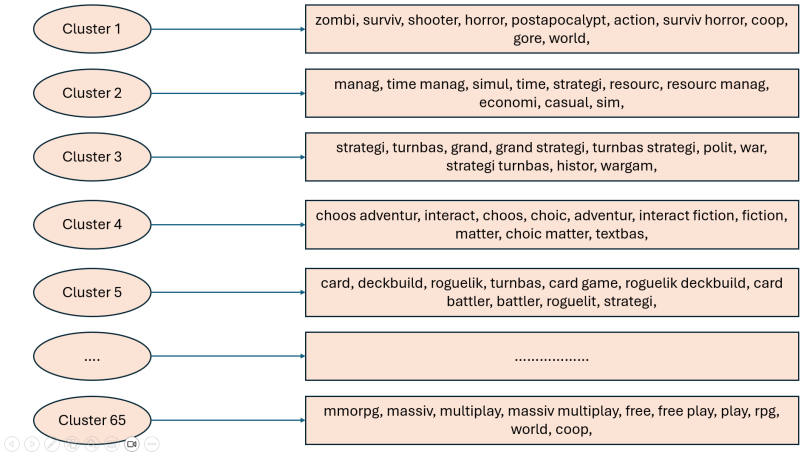


Figure 5 . Top 10 terms of each cluster

The received result is not too bad, top 10 terms of each cluster does represent a different game category. This can be utilized for recommendation purpose.

2.6 Ridge and Lasso Regression

2.6.1 Overview

Another approach in content-based filtering is to predict user rating by the content of the games. As our rating is numerical, this is a regression task. Two algorithms utilized here are Ridge Regression and Lasso Regression. Both algorithms are the improved version of Linear Regression which they add a regularization term to the mean squared error to prevent the model from overfitting. The only difference between 2 algoritms is that while Ridge use L2 regularization, Lasso use L1 regularization. Given y be the true label, $\hat{y}$ be the predicted value, w is the vector of coefficients found by the algoritm.
Then Ridge aims to minimize:

$$\sum(y - \hat{y})^2 + \lambda\|w\|^2$$

And Lasso aims to minimize:

$$\sum(y - \hat{y})^2 + \lambda\|w\|^1$$

2.6.2 Evaluation

We use kfold cross validation with k=5 to evaluate the regression models. Here the graph resprenting the root mean squared error of each model corresponding to different λ value.

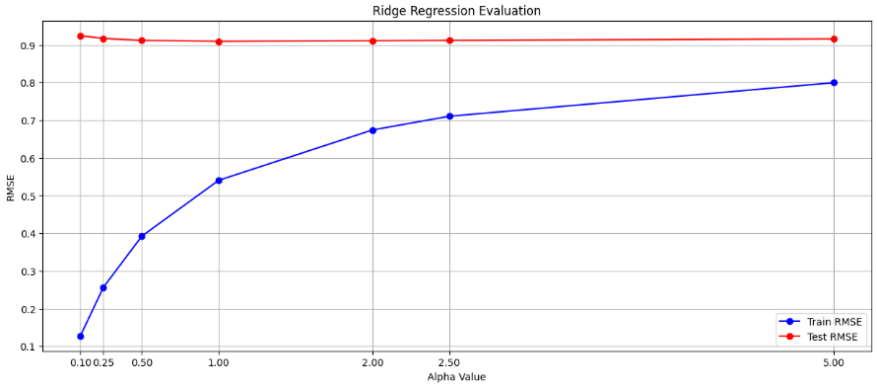


Figure 6 . Ridge Regression evaluation

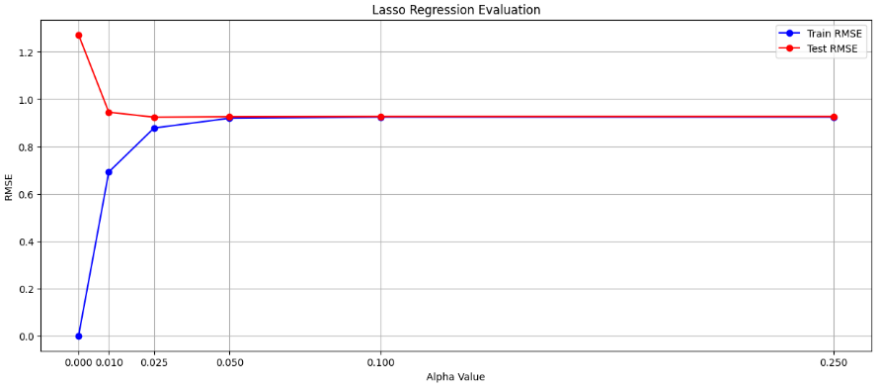


Figure 7 . Lasso Regression evaluation

For Ridge regression we choose $\lambda = 1$ as at this point, the test mean squared error stop reducing and the train error is acceptable. For Lasso Regression, any value of lambda from 0.025 does not make any difference so lambda value can be chosen as 0.25. The test error in both algorithms are fairly good, lying around 0.8.

2.7 Regression Random Forest

Regression Random Forest is an ensemble model that combines multiple regression tree to tackle the drawback of a single tree that is too prone to overfiting. Each tree in a random forest is trained on a random sample of data and the final prediction is the average of results produced by all trees.

2.7.1 Regression tree

Regression tree works by binary splitting the training data recursively into smaller subsets based on specific criteria. A regression tree is a binary tree where each node (except the leaf node) is split into 2 child nodes. A regression is built from top to bottom. Each node divide the data into 2 subsets by compare the value of a specific feature named A of data to a threshold : subset 1 (the left child node) representing all the data point that has A<threshold and subset 2(the right child node) representing all the data point that has A>=threshold. For example, we are working with a data with 2 features age and height and our goal is to predict the weight of people. A root node can divide our data into 2 groups: people that has height<170cm and people that has height>=170cm. The divide strategy is chosen by evaluating the mean squared error obtained after splitting the data based on that strategy. For example, to determine the divide strategy for the root node, we first consider the feature height. If the data has value of height being 3,4,5, it would consider two thresholds 3 and 4. The rmse is then calculated and we choose the threshold that minimize it for example here we take 3. Then we consider the feature age and do the same, for example we obtain the best threshold to divide the users by their age is 20. 2 best dividing strategies corresponding to 2 features are then compared to choose the best strategy for the root node. This process is repeated over and over until we get a leaf node. A leaf node can be a single data point or defined manually by setting the min number of samples to be able to keep split. The prediction for a data point is received by taking the average of the labels of all data points in the leaf that point belongs to.

2.7.2 Regression random forest hyperparameters

A regression random forest model has some key hyperparameters:

n_estimators : number of trees in the forest, higher number often lead to better generalization performance of the model.

min_samples_leaf: the minimum number of samples in a node to be considered as a leaf, higher value makes the tree less complex but might miss some information of the original data.

max_features: the max number of feature used in each tree, lower value reduce the computational cost and make the model less prone to overfitting but can result in underfitting.

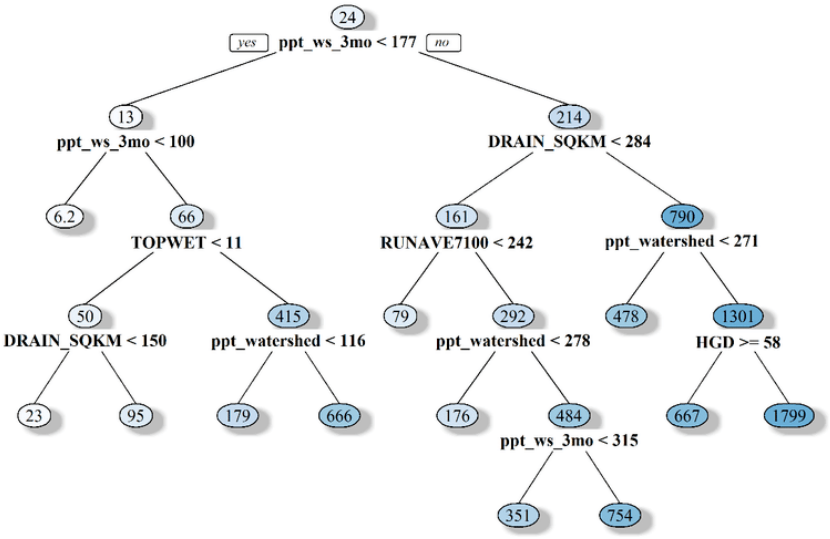


Figure 8 . A regression tree(source: ResearchGate)

2.7.3 Evaluation

We use k-fold validation with k=5 to evaluate the model. Here the result table:

	20	50	100	200
max_feature=n_features min_sample_leafs=1	0.959	0.945	0.942	0.927
max_feature="sqrt" min_sample_leafs=5	0.914	0.903	0.899	0.900
max_feature="sqrt" min_sample_leafs=10	0.925	0.923	0.921	0.921
max_feature=200 min_sample_leafs=5	0.879	0.871	0.869	0.868
max_feature=200 min_sample_leafs=10	0.927	0.922	0.922	0.920
max_feature=400 min_sample_leafs=5	0.933	0.926	0.924	0.922
max_feature=400 min_sample_leafs=10	0.928	0.922	0.922	0.921

Figure 9 . Random forest evaluation result

Part III
Neighbor-based CF

Collaborative filtering (shortened to CF) is a method of making automatic predictions or filtering, about the interests of a user by deriving from the available data about the preferences of many other users, hence the collaborating nature. CF utilizes a rather natural phenomenon: similar people tend to like similar things, and based on such similarity the system can give predictions about a user's preferences. It assumes that if person A has the same opinion as person B on some specific issues, A is more likely to share B's opinion on a different issue than that of a random person. By this nature, there will always be people who do not follow common sense, this algorithm fails to suit these 'gray sheep'.

There are four main types of collaborative filtering: Memory-based, Model-based, Hybrid, and Deep-learning. The following sections will focus on the implementation of the first method, which uses user rating data to compute the similarity between users or items.

There are two approaches for measuring similarity: User-based, which measures the similarity between target users and other users, and Item-based, which focuses on the similarity between the items that target users rate and other items. This report will focus mainly on the user-based method for demonstration. The item-based method can be easily implemented by swapping the user and item when performing CF on the rating data.

1 Data preprocessing

The model in this section uses users' ratings as input. Skimmed from the Recommendations dataset, only users who have more than 60 reviews are selected, along with the corresponding games(app_id), opinions(is_recommended) and hours spent playing(hours). Each pair of opinion and play hour is combined and fitted into 1-5 interval with step of 0.5 called explicit rating to seperate from implicit rating, which is basically the player's intention of recommendation. The more time spent playing, the higher its contribution in total rating. The evaluation is based on explicit ratings.

2 Utility matrix

In CF, we represent the user-item relationship by constructing a utility matrix from the rating data. We can consider each row to be an item and each column to be a user, the element at column u and row i thus corresponds to the rating that user u gives to item i.

	u_1	u_2	u_3	u_4
i_1	0	0	?	3
i_2	1	1	3	?
i_3	3	?	?	?
i_4	2	2	1	?

Figure 10 . Utility matrix

Since not all items are rated by a user, there will be missing values scattered across the matrix (represented as ?). The main goal of CF is to fill in these missing values using the existing data and then give out recommendations based on the predicted rating data.

3 Similarity functions

In order to find the similarity between each user, we need to derive a suitable function *sim* for the task. Once calculated, we can rank all other users based on the similarity to one user and select the top k most similar users to predict the rating for an item that they have not rated before. The similarity function will take two vectors as an input and return a value that represents the similarity. The value should be in the range [-1, 1] with the more similar the inputs the closer the value is to 1. There are many ways to calculate the similarity between two users, these are some popular ones, two of which are implemented in this project.

3.1 Cosine similarity

Cosine sim measures the similarity of two attributes by calculating the angle between two corresponding vectors. Mathematically, it is the division between the dot product of vectors and the product of the magnitude of vectors, as presented in this formula

$$sim(x, y) = \frac{\sum_{i \in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum_{i \in I_x} r_{x,i}^2} \sqrt{\sum_{i \in I_y} r_{y,i}^2}} \tag{2}$$

Given all the scores are vectorized, we can use this sim to evaluate the chance, the smaller the angle between two vectors, the greater the similarity between those two corresponding games.

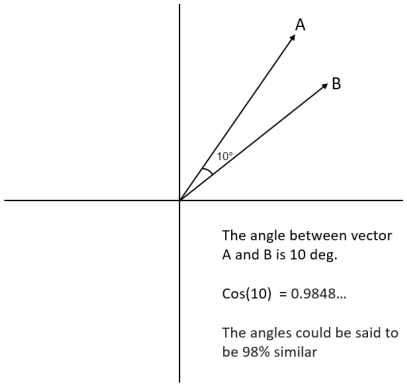


Figure 11 . Visualization similarity of two vectors in angle

3.2 Pearson similarity

Another way is to use the Pearson similarity:

$$sim(x,y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r_x})(r_{y,i} - \bar{r_y})}{\sqrt{\sum_{i \in I_x} (r_{x,i} - \bar{r_x})^2} \sqrt{\sum_{i \in I_y} (r_{y,i} - \bar{r_y})^2}} \tag{3}$$

with I_x being the items rated by user x, I_xy being the items rated by both x and y and $\bar{r_x}$ being the average rating given by x.

3.3 Distance based similarity

In this project, we also experiment with some "distance-based" similarity functions. Given a distance function D, we can convert it to a similarity function in many ways, ideally a decreasing positive function that is 1 at the point 0 (since the distance is non-negative). An example is to use the following conversion:

$$sim(x,y) = \frac{1}{1 + D(x,y)} \tag{4}$$

These methods may be less expensive to compute but the main drawbacks is that they are not invariant to scaling (which is not an issue for cosine and Pearson). Since we use 1-5 interval ratings in stead of boolean ones, the scaling would have some effect on the system in this case.

4 Get prediction

The predicted rating can be calculated as an aggregation of similar users' ratings. Suppose we have U the set of k most similar users, there are some different ways to get the predicted ratings.

- Taking the average rating

$$r_{u,i} = \frac{1}{k} \sum_{u' \in U} r_{u',i} \tag{5}$$

- Use similarity as weight

$$r_{u,i} = \mu \sum_{u' \in U} sim(u, u') r_{u',i} \tag{6}$$

- Normalize the ratings before calculation

$$r_{u,i} = \bar{r_u} + \mu \sum_{u' \in U} sim(u, u')(r_{u',i} - \bar{r_u}) \tag{7}$$

with

$$\mu = \left(\sum_{u' \in U} |sim(u, u')| \right)^{-1}$$

Applying this method to all the missing values, we will get the completed utility matrix from which we can give recommendations for an user by getting the top rated items that the user has not rated yet according to the system.

	u_1	u_2	u_3	u_4
i_1	0	0	?	3
i_2	1	1	3	?
i_3	3	?	?	?
i_4	2	2	1	?

(a) Not filled

	u_1	u_2	u_3	u_4
i_1	0	0	**0.0**	3
i_2	1	1	3	**0.9**
i_3	3	**1.9**	**1.9**	**1.9**
i_4	2	2	1	**2.9**

(b) Filled

Figure 12 . Completed utility matrix

5 Evaluation

For CF systems, the predicted rating will be a floating value instead of a whole number. In this case, we can consider recommendation as a regression problem and we use Root Mean Square Error (RMSE) as an evaluation for accuracy.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2} \tag{8}$$

Another metric we could use for accuracy is the Coefficient of determination or R2-score which measure the goodness of fit of a model (how well the model fits the set of observed data). In this case, the closer R2 to 1, the better the model's predicted values.

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \widehat{y_i})^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2} \tag{9}$$

Using the above evaluations, we have the following tables:

N-neighbors	**1**	**3**	**5**	**7**	**9**
UU-IESquared	1.085951	1.034979	1.020903	1.013421	1.008104
UU-IManhattan	1.100333	0.040083	1.021925	1.013053	0.007300
UU-Cosine	1.245777	1.054529	1.012215	0.993344	0.983135
UU-Pearson	1.245777	1.054529	1.012215	0.993344	0.983135

Table 4 . RMSE of different CF models

N-neighbors	**1**	**3**	**5**	**7**	**9**
UU-IESquared	0.141340	0.220056	0.241126	0.252208	0.260036
UU-IManhattan	0.118447	0.212344	0.239606	0.252753	0.261215
UU-Cosine	-0.130006	0.190313	0.253988	0.281544	0.296236
UU-Pearson	-0.130006	0.190313	0.253988	0.281544	0.296236

Table 5 . R2-scores of different CF models

However, in most modern recommendation systems, we do not care much about the accuracy of the ratings but rather how the user would interact with the recommended items, and do they agree with the recommendation.

Part IV
Latent Factor CF

1 Data Preprocessing

This preprocessing step works with the selection features data of the user and item. Each method has a little difference in constructing data to be suitable for the model. However, feature selection is the same in all of models.

1.1 Mutual Information

Mutual information(MI) rooted in information theory and data analysis, embodies a profound mathematical concept to quantify the amount of information obtained about one random variable through the observation of another random variable. This concept is raised from the notion of entropy, which measures the uncertainty associated with random variable. The mutual information $MI(X, Y)$ between X and Y captures how much knowing one variable reduces the uncertainty about the other.

$$MI(X;Y) = \sum_{y} \sum_{x} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

It is evident that Mutual information can also be expressed as the Kullback-Leibler (KL) divergence between the joint distribution p(x,y) and the product of the marginal distributions p(x)p(y).When X and Y are independent, their joint distribution factorizes into the product of their marginals, resulting in zero MI.

In our dataset for metadata games, there are only game genres(tags) without any labels, thus it is not possible to apply Information Gain, Mutual Information serves as an alternative solution. For each genre (tag), we calculate its MI with the target variable (game). Higher MI values indicate that the feature is more informative or relevant for predicting the target variable. We extract top 10 tags with highest MI value, which is:'Atmospheric', 'RPG', 'Strategy', '2D', 'Simulation', 'Casual', 'Adventure', 'Action', 'Singleplayer', 'Indie'.

1.2 Discretization

For game features, in addition to 10 genres, we will also include information related to price and the developer's rating. For the rating, we will use one-hot encoding, marking a 1 for the corresponding rating and 0 for the others. As for the price, we will combine it with user information, which will be detailed below in the user features section.

For user features, we will use the number of products and an additional feature we created, which is the average price of the games the user has recommended. We will discretize both of these values based on quantiles: products will be divided into 5 bins corresponding to 5 ranges, while price will be divided into more than 10 bins, used for both games and users. The reason for using more bins for price is that the price distributions for games and users are quite different, so we need finer granularity.

The reason we discretize rather than standardize using z-norm is that with many features for genre and rating, if we only z-norm and create one feature for price and products, it will result in unbalanced features, causing the model to learn poorly and bias the important features towards genre or developer ratings. Moreover, we use quantiles instead of ranges based on mean and variance because the variance is very large, so such a split would only focus around the mean. Thus, splitting by quantiles is, in our opinion, the most stable and suitable approach in this case.

2 Matrix Factorization

Matrix Factorization (MF) is technique that emerged from 2006 during Netflix prize competition, it focus on decomposing the rating matrix into factorized matrices which conceal many latent features and information about the rating. In Netflix dataset, it do no exist any information about films or users except from their interaction(rating), thus mostly variants evolved from this approach solely focus on approximate this rating matrix. In this report, we experiment some basic algorithms,i.e., Funk-SVD, SVD++, Non-negative matrix factorization(NMF), Alternative least squares(ALS), Bayesian personal ranking matrix factorization(BPR-MF) for implicit rating. Moreover, some techniques incorporating side info are also proposed to have a bigger insight into the field of recommender system problem from the perspective of Matrix Factorization.

2.1 Matrix Factorization without Side Info

2.1.1 Traditional SVD

Singular Value Decomposition(SVD) is a powerful and beautiful mathematical technique in linear algebra to decompose the a matrix into three component matrices, revealing the intrinsic of the original matrix. It factorizes the matrix $A_{m\times n}$ into the 3 matrices: $U_{m\times m}$, $(\Sigma)_{m\times n}$, $V_{n\times n}$ with k $\ll$ min(m,n), then SVD is also known as a dimensionality reduction technique. Particularly, U is an $m \times m$ orthogonal matrix representing the left singular vectors, Σ is an $m \times n$ diagonal matrix with non-negative real numbers on the diagonal known as singular values, and V^T is the transpose of an $n \times n$ orthogonal matrix containing the right singular vectors.

SVD [1] was first used by Billsus and Pazzani in 1998 in recommendation service. They explored a few machine learning algorithms for collaborative filtering tasks and identified SVD as the best-performing one. However, this method needs large storage for completed matrix and its computation complexity is intensive, which is not suitable. Especially, with the rapid growth of information and data today, data for recommender systems is extremely large, so using this method would incur prohibitively high computational costs.

Thus,below method inspired by SVD implemented during the ongoing of Netflix Prize competition by Funk applies explicit learning process. Funk-SVD can also be called by Matrix Factorization(MF).

2.1.2 Funk-SVD

Neighbor-based model: As for k-neighbor collaborative filtering for item-item and user-user, the computation complexity is usually large with many users in real life as well the space complexity for the utility matrix is high, and the utility matrix is sparse because of at least interaction from the user to item. In addition, k-neighbor collaborative filtering does not discover the latent space between the user and the item.

Therefore, based on the important fact of traditional SVD that the matrices that result in the decomposition can have significantly lower dimension than the original matrix, Funk proposed MF method decomposing interaction matrix into only two matrices instead of three as in the traditional approach, and employs an iterative approach that focuses on optimizing the low-rank. To understand it more clearly, let be an example of idea matrix factorization with 4 people and 4 items. The rating matrix will be estimated by matrix factorization to show that

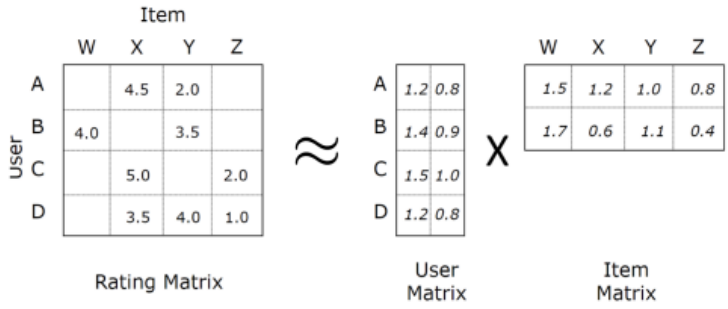


Figure 13 . Idea of matrix factorization (from material [10])

In this example, we use the multiplication result of the user matrix $P_T \in R_{4\times2}$ and item matrix $Q \in R_{2\times4}$ to represent it. The rating value of the user for each item is defined as $P_u ser_T xQ$. After obtaining those two decomposition matrices P and, Q we can predict the missing values in the rating matrix from P, and Q's product.
The final rating is created by also adding in the aforementioned baseline predictors that depend only on the user or item. Thus, rating is predicted by the rule

$$\hat{r_{ui}} = p_u^T q_v$$

Up to now, we have built a loss function for the process training of matrix factorization, with the idea of learning the latent factor between user and item, the model focuses on estimating vector user and vector item for prediction train. Let p_u, q_v be vector users, items, and $r_u v$ real rating value.

$$L(p,q) = \sum_{(u,v)\in train}(r_{uv} - p_u^T q_v)^2 + \lambda(\|p_u\|^2 + \|q_v\|)$$

Where λ is the regularization parameter.Stochastic gradient algorithms are used more effectively for minimum loss function.

According to SGD, we update the parameter by two equations following here.

$$p_u \leftarrow p_u + \eta(q_v e_{ui} - \lambda p_u)$$
$$q_v \leftarrow q_v + \eta(p_u e_{ui} - \lambda q_v)$$

Adding biases

One benefit of the matrix factorization approach to collaborative filtering is its flexibility in dealing with various data aspects and other application-specific requirements. However, much of the observed variation in rating values is due to effects associated with either users or items, known as biases or intercepts, independent of any interactions. For instance, in a recommendation system, some users might consistently rate items higher or lower than others, regardless of the specific items they are rating. Similarly, some items might generally receive higher or lower ratings across all users. Thus, incorporating bias terms serves to improve the accuracy and interpretability of the model. The system tries to identify the portion of these values that individual user or item biases can explain, subjecting only the true interaction portion of the data to factor modeling. A first-order approximation of the bias involved in rating r_{ui} is as follows:

$$b_{ui} = b_u + b_i + \mu$$

with b_u, b_i indicate the observed average of user u and item i, respectively; μ is the overall average rating and the predicted score is :

$$\hat{r_{ui}} = \mu + b_i + b_u + p_u^T q_v$$

hence, the loss function is :

$$L(p,q) = \sum_{(u,v)\in train} (r_{uv} - p_u^T q_v - b_i - b_u - \mu)^2 + \lambda(\|p_u\|^2 + \|q_v\| + b_i^2 + b_u^2)$$

For SGD optimizer, in each iteration, updating p and q is similar to above, but we also update b_i, b_u according to the following formula.

$$b_u \leftarrow b_u + \eta(e_{ui} - \lambda b_u)$$
$$b_i \leftarrow b_i + \eta(e_{ui} - \lambda b_i)$$

2.1.3 SVD++

SVD++ [2] is also raised in Netflix Prize Competition , as version of explicit learning improvement of Item Collaborative Filtering and implicit learning improvement of SVD. Particularly, we see how Item Collaborative works:

$$p_{uj} = \sum_{i\in N(u)\cap S(j.K)} w_{ji} r_{ui}$$

where N(u) is the user u's rated item set, S(j,K) is set of top-K most relevant to item j, r_{ui} is user u's scrore on item i. In ItemCF, w_{ij} is defined as $w_{ij} = \frac{N(i)\cap Nj}{N(i)}$ where N(i),

N(j) are numbers of users who are interested in item i, j, but in SVD++ the parameter is learnable parameter. Since the dimensionalisty of w is very large, then authors proposed to dimensionality reduction by decomposing w into $x^T y$. In SVD++, we can extend the Funk-SVD by adding the implicit rating :

$$p_{uj} = \mu + b_t + b_u + p_u^T q_v + \frac{1}{\sqrt{|N(u)|}} x^T y$$

Also, in order to not have too many parameters to cause over-fitting, we can arbitrarily set x = q to reduce the number of parameters. So the final SVD++ can be written as the following equation

$$p_{uj} = \mu + b_t + b_u + q_v^T (p_u + \sum_{j \in N(u)} \frac{1}{\sqrt{|N(u)|}} y_j)$$

The learning process of b is the same as the SVD with b_u and b_i, for p,q,y we can use SGD to minimize the loss function by formula:

$$\begin{aligned} y_j &\leftarrow y_j + \eta(e_{ui}|R(u)|^{-\frac{1}{2}} q_i - \lambda y_j) \\ p_u &\leftarrow p_u + \eta(e_{ui} q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \eta(e_{ui}(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j) - \lambda q_i) \end{aligned}$$

2.1.4 ALS

The training process using gradient descent can be quite slow because the convergence rate depends on the learning rate. Meanwhile, we observe that the loss function for each user or item can be optimized by solving the derivative equation. Note however, that if we fix the set of variables X and treat them as constants, then the objective is a convex function of Y and vice versa. Our approach will therefore be to fix Y and optimize X, then fix X and optimize Y , and repeat until convergence.

This approach is known as ALS(Alternating Least Squares). For our objective function, thealternating least squares algorithm is as follows:

2.1.5 Non-negative Matrix Factorization (NMF)

Non-negative matrix factorization(NMF) is similar to conventional matrix factorization methods but imposes the condition that the matrix factorization entries must be non-negative.It draws inspiration from deal-world dataset which often has non-negative entries,eg,the rating in recommendation scenario or the value of pixels in the image(0-255). Thus, NMF enhances the capability of recommender systems by introducing non-negativity constraints to the factorization process, which can lead to more interpretable and meaningful latent features.Unlike traditional Matrix Factorization (MF) that allows both positive and negative entries in the factorized matrices, NMF restricts the matrices P and Q to contain only non-negative values.

Specifically, factorized matrices can be computed using the multiplicative update rules, which iteratively update the matrices P and Q to minimize the difference of original matrix

Algorithm 1 Alternating Least Squares (ALS)

Procedure ALS (p_u, q_v)
Initialization $p_u \leftarrow 0$
Initialization matrix q_v
repeat
Fix q_v, solve p_u by minimizing the objective function (the sum of squared errors)
for $u = 1$ **to** m **do**

$$p_u = (\sum_{r_{ui} \in r_{u*}} q_v q_v^T + \lambda I_k)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} q_v$$

end for
Fix p_u, solve q_v by minimizing the objective function similarly
for $i = 1$ **to** n **do**

$$q_v = (\sum_{r_{ui} \in r_{u*}} p_u p_u^T + \lambda I_k)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} p_u$$

end for
until convergence

and the product of P and Q and have to ensure remaining the non-negativity of P and Q through iterations.[3] has proposed and proved its convergence through the auxiliary function the update rule as follows for measure of Euclidean distance:

$$P_{ij}^{(t+1)} \leftarrow P_{ij}^{(t)} \frac{((Q^{(t)})^T V)_{ij}}{((Q^{(t)})^T Q^{(t)} P^{(t)})_{ij}}$$

$$Q_{ij}^{(t+1)} \leftarrow Q_{ij}^{(t)} \frac{(V(P^{(t+1)})^T)_{ij}}{(Q^{(t)} P^{(t+1)} (P^{(t+1)})^T)_{ij}}$$

2.1.6 Bayesian Personal Ranking-Matrix Factorization (BPR-MF)

In many recommendation scenarios, explicit feedback such as ratings or reviews is sparse or unavailable. Therefore, leveraging implicit feedback, such as clicks, purchases, or views, becomes essential. Bayesian Personal Ranking (BPR) is a state-of-the-art algorithm designed to effectively utilize implicit feedback for personalized recommendations.

BPR[4] is a ranking-based approach that aims to optimize the ordering of items for each user, rather than predicting absolute ratings. It operates under the assumption that users prefer certain items over others, and this preference can be inferred from observed implicit feedback. BPR is considered a generic optimization criterion (BPR-Opt) for personalized ranking, derived from a Bayesian analysis of the ranking problem. This method also provides a generic learning algorithm for optimizing models with respect to BPR-Opt.

BPR-Opt

The optimization criterion BPR-Opt is formulated within a Bayesian framework. We aim to maximize the posterior probability of the model parameters Θ given the observed user preferences $>_u$:

$$P(\Theta \mid >_u) \propto P(>_u \mid \Theta) P(\Theta)$$

Here, $P(\Theta)$ represents the prior distribution over the model parameters, and $P(>_u \mid \Theta)$ is the likelihood of the observed preferences given the model parameters. The key assumptions in this method are that all users act independently of each other and each pair (i, j) for each user is also independent. Thus, the above formulation can be rewritten as:

$$\prod_{u \in U} P(>_u \mid \Theta) = \prod_{(u,i,j) \in D_s} P(i >_u j)^{\delta(u,i,j) \in D_s} \prod_{(u,i,j) \notin D_s} (1 - P(i >_u j \mid \Theta))^{1 - \delta(u,i,j) \notin D_s}$$

where δ is the indicator function with values 0 or 1. This function can be simplified as:

$$\prod_{u \in U} P(>_u \mid \Theta) = \prod_{(u,i,j) \in D_s} P(i >_u j)$$

Here, the probability that user u prefers item i over item j is modeled using the logistic sigmoid function:

$$P(i >_u j \mid \Theta) = \sigma(\hat{x}_{uij}(\Theta))$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic sigmoid function, $\hat{x}_{uij}$ express relationship of user u prefer item i to item j. The optimization criterion can thus be expressed in logarithmic form, incorporating the Bayesian idea of maximizing the posterior probability:

$$\begin{aligned}
\text{BPR-Opt} &= \ln P(\Theta \mid >_u) \\
&\propto \ln(P(>_u \mid \Theta) P(\Theta)) \\
&= \ln \left(\prod_{(u,i,j) \in D_s} P(i >_u j) P(\Theta) \right) \\
&= \ln \left(\prod_{(u,i,j) \in D_s} \sigma(\hat{x}_{uij}) P(\Theta) \right) \\
&= \sum_{(u,i,j) \in D_s} \ln(\sigma(\hat{x}_{uij})) + \ln P(\Theta) \\
&= \sum_{(u,i,j) \in D_s} \ln(\sigma(\hat{x}_{uij})) - \lambda_\Theta \|\Theta\|^2
\end{aligned}$$

BPR-Learning Algorithm

The BPR optimization algorithm can be summarized in the following steps:

Bayesian Personalized Ranking-Matrix Factorization (BPR-MF)

In BPR-optimization, we deal with triplets $(u, i, j) \in D_s$.In matrix factorization, the matrix $\hat{X}$ is approximated by WH^T, where the model parameters are $\Theta = (W, H)$.The gradient of $\hat{x}_{ui}$ with respect to Θ is given by:

Algorithm 2 BPR Optimization Algorithm

1: Initialize model parameters Θ
2: **repeat**
3: Sample a minibatch of triplets (u, i, j) from training data
4: Update Θ to minimize the loss function based on $\hat{x}_{uij}$

$$\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \Theta \right)$$

5: **until** convergence

$$\frac{\partial}{\partial \theta} \hat{x}_{ui} = \begin{cases} h_{if} - h_{jf} & \text{if } \Theta = w_{uf} \\ w_{uf} & \text{if } \Theta = h_{if} \\ -w_{uf} & \text{if } \Theta = h_{jf} \\ 0 & \text{else} \end{cases}$$

This formal framework of BPR-MF effectively leverages implicit feedback for personalized ranking, providing a robust method for recommendation systems where explicit feedback is sparse or unavailable.

2.1.7 Empirical results

Below are the results of four simple models derived from matrix factorization:

Table 6 . Performance results of models for explicit feedbacks

Evaluation Metric	**Funk-SVD**	**SVD++**	**ALS**	**NMF**
RMSE	textbf0.9436	0.9442	0.9448	1.0238
F1 Score@10	0.6935	0.6939	0.7152	0.7384
NDCG@5	0.8280	0.8283	0.8361	0.8206
Time	3.53	80.53	**0.49**	4.85

The results indicate that, although SVD is a simple method, it achieves the best RMSE on the test set. However, ALS, which uses the same loss function as SVD, while not performing as well in terms of accuracy, demonstrates a superior advantage in algorithm runtime, being significantly faster.

The second table presents the results of BPR compared to an ALS algorithm when dealing with implicit feedback:

Table 7 . Performance results of models for implicit feedbacks

Evaluation Metric	ALS	BPR-MF
F1 Score@10	0.7980	0.9018
NDCG@5	0.9234	0.9825

Since BPR is designed for learning based on ranking, its evaluation metrics significantly outperform those of ALS.

2.2 Matrix Factorization with Side Info

2.2.1 Collective Matrix Factorization (CMF)

Collective Matrix Factorization (CMF) is an advanced matrix factorization technique designed to handle multiple interrelated datasets simultaneously. Unlike traditional matrix factorization, which deals with a single matrix at a time, CMF integrates information from multiple matrices, capturing the dependencies and interactions among different types of entities. This approach has proven particularly effective in relational learning and addressing the cold-start problem in recommendation systems. The fundamental concepts was first proposed in [5] by extending conventional matrix factorization method by incorporating multiple matrices and aimed to approximate it.

Based on this idea, [6] also put forward formulas by jointly factorizing the rating matrix X_{ui} along with the user attributes matrix U_{up} and the item attributes matrix I_{ip}, simultaneously introducing new matrices C_{pk} and D_{qk} for user and item attributes, but sharing the A_{uk} and B_{ik} matrices between factorizations:

$$\min_{A,B,C,D} ||X - AB^T||^2 + |U - AC^T||^2 + ||I - BD^T||^2$$

or put it in more intuitively understandable form:

$$X \approx AB^T + \mu + b_A + b_B, \quad U \approx AC^T + \mu_u, \quad I \approx BD^T + \mu_i$$

The new matrices C and D are excluded from the prediction formula, yet their inclusion in the minimization objective enhances the estimates for A and B. This is because they must account for both the interactions and the additional information, reducing the likelihood of overfitting to the observed interactions. Consequently, these latent factors are compelled to align with the non-latent attributes, leading to better generalization to new data.

ALS and L-BFGS optimizer algorithm were also proposed to solve the loss function. However, for simplicity, we merely utilize ALS for our porject and the concepts, mathematical foundations are also mentioned above.

2.2.2 Other formula for CMF

In addition to the traditional formulation of Collective Matrix Factorization (CMF), an alternative approach, known as the "offsets" model, is proposed in the literature [6]. This model is designed to address cold-start scenarios more effectively by directly integrating

side information into the optimization process. The formulation of the "offsets" model is as follows:

$$\min_{A,B,C,D,m,n} ||I_x(X - \mu - m - n - (A + UC)(B + ID)^T)||^2 + R$$

Here, matrices A, B, C, and D represent the latent factors of users and items, respectively, while matrices U and I represent user and item attributes, respectively. Vectors m and n represent the free offsets introduced based on observed interactions data.

The core idea of the "offsets" model is to calculate a base matrix of latent factors by combining user/item attributes linearly. Additionally, free offsets are introduced based on observed interactions data to refine the latent factors further. This approach enhances the model's adaptability to new data and improves its performance, especially in cold-start scenarios where interaction data is limited.

Similar to the previous, for simplicity, we solely employ ALS algorithm in our study. It's important to note that while the "offsets" model is particularly relevant for addressing cold-start scenarios in recommendation systems, our study focuses on a traditional recommendation setting and does not explicitly tackle cold-start problems. Nevertheless, the "offsets" model provides a flexible framework for incorporating side information into the optimization process, which can potentially enhance the model's performance across various recommendation scenarios.

2.2.3 Factorization Machine

Factorization Machines [7] are powerful predictive modelling techniques especially designed to handle sparse and high-dimensional data, which is typical feature in recommender system. Traditional linear model methods can handle with high-dimensional data but often fail to capture interactions between features, leading the explosion in the number of parameters. On the other hand, model like SVM can handle-dimensional data but struggle with high sparsity. Therefore, FM are proposed as a new model that combines the advantages of Support Vector Machines (SVM) with factorization models. Specifically, the model equation for a factorization machine of degree d = 2 is defined as:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

where:

- $\hat{y}$ is the predicted output.
- w_0 is the global bias.
- w_i are the weights for the linear terms associated with feature x_i
- x_i is the i-th feature of the input vector $\mathbf{x}$.
- $\mathbf{v}_i$ is the latent vector associated with feature x_i

- $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ is the dot product of the latent vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, representing interaction between x_i and x_j

It is evident that the first term $w_0 + \sum_{i=1}^{n} w_i x_i$ is linear equation representing the linear terms), The interesting of equation lies on the second term: $\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$, which uses the $\langle v_i v_j \rangle$ approximating interaction weight w_{ij} as latent factors to capture the interactions between different features. Contrary to the traditional approach, such as the SVM with usage of polynomial kernel function, modelling weight directly $\sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbf{w_{ij}} x_i x_j$ requires $O(n^2)$ parameters, FM utilizes factorization with latent factor with O(k×n) parameters, k $\ll$ n, leading to fewer parameters and better scalability.

It can be inferred that FM serves as an improvement over matrix factorization as it can incorporate out-of-band information into the model, with the number of parameters being only O(n×k), significantly fewer than in CMF.By factorizing the interaction weights into latent vectors, FMs can generalize better, learning useful patterns even when data is sparse.

One particularly remarkable and advantageous aspect of this model also lies in computation complexity, while [7] demonstrates the model equation of factorization machine can also be computed in linear time O(n×k) instead of easy-to-see O(k $\times n^2$) by reformulating pairwise interaction as

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j = \frac{1}{2} \sum_{f=1}^{k} \left[\left(\sum_{i=1}^{n} v_{i,f} x_i \right)^2 - \sum_{i=1}^{n} v_{i,f}^2 x_i^2 \right]$$

And model parameters(w_0, w, y) can be learned by gradient descent methods, [7] also proposes formulation for stochastic gradient descent

2.2.4 Empirical results

The following are the results of two matrix factorization algorithms with side information compared to factorization machines.It can be observed that FM performs slightly better than the other two algorithms.

Table 8 . Performance Results

Evaluation Metric	**CMF**	**CMF-Other Formula**	**FMs**
RMSE	0.9444	0.9393	**0.9170**
F1 Score@10	0.7143	0.7114	0.7196
NDCG@5	0.8353	0.8337	0.8281

CMF is designed to be used in cold-start scenarios with external information. However, we did not use external information in our implementation, which prevented CMF from utilizing its strengths. This is a significant limitation that needs to be addressed in future work on the cold-start problem.

3 Deep Learning Approach

Deep learning has succeeded in many tasks such as text, speech, and image. Recently, several methods of recommendation have been applied to deep learning with successful results such as Neural Collaborative Filtering [8]. In this part, we discuss some simple The deep learning model for learning interaction between user and item with implicit data and explicit data.

3.1 Neural Collaborative Filtering (NCF)

This section shows the first model approach to deep learning to learn the latent factor between the user and item by non-linear layer under framework matrix factorization. By learning in Black Box, the latent factor user-item space can be more efficient than the linear model latent factor.

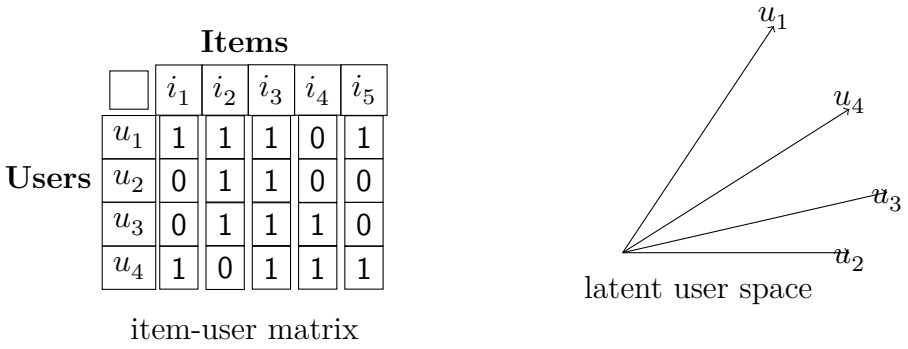


Figure 14 . Example of limitation of MF in latent space

Matrix Factorization linear map user and item in latent space with K dimension.Figure 2 illustrates the limit of MF for latent space with 4-dimensional latent space (detail in [2]). We use the cosine coefficient to measure the similarity of two users, considering that $s(u_1, u_4) = 0.75 > s(u_3, u_4) = 0.57 > s(u_2, u_4) = 0.35$ mean u_4 closest to u_1. Beside $s(u_2, u_1) = 0.7 > s(u_3, u_1) = 0.57$, there is a conflict here where u_1 is nearest u_4 and u_2 with coefficient 0.75 and 0.7 respectively, however u_4 and u_2 are more different.

Up to now, we have discussed using a multi-layer perceptron (MLP) to learn the user-item interaction function. learning with implicit feedback, and based on the method exploring another method. NCF architecture is shown in Figure 12, where the input is vector one hot identity of users and items, and map on the embedding layer to take the vector embedding of users and items. Afterward, the concatenated vector of the user vector and item vector throughout the non-linear layer, results in the output layer with a score of interaction between the user and the item.

Let be $P \in R^{MXK}, Q \in R^{NXK}$ are the embedding matrix of users and items with K latent factor, M user and N items,θ_f denotes the NFC's parameter. u_i, v_j are one hot vector of users and items. We now consider that NCF's prediction model

$$score(u_i, v_j) = f(u_i^T P, v_j^T Q | Q, P, \theta_f)$$

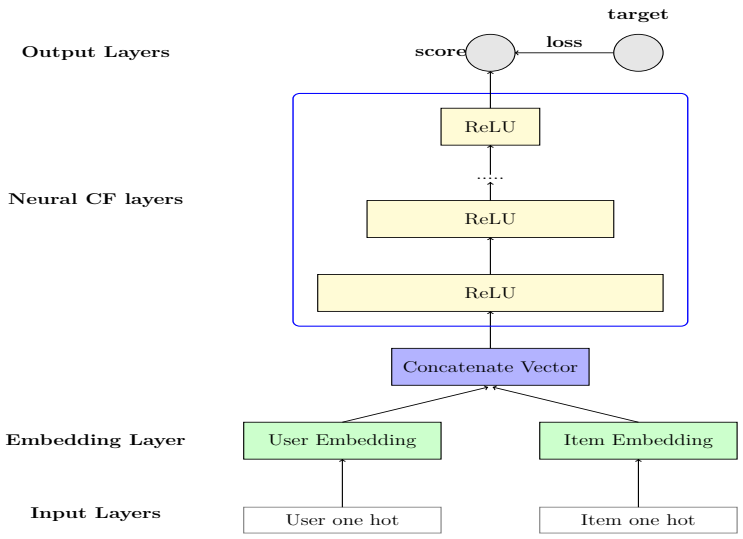


Figure 15 . Neural Collaborative Architecture

show in detail the neural collaborative interaction function of users and items.

$$f(u_i^T P, v_j^T Q|Q, P, \theta_f) = \phi_{out}(\phi_X(...\phi_2(\phi_1([u_i^T P, v_j^T Q])))$$

Where [.,.] concatenate operation, and $\phi_1, \phi_2, ..., \phi_X$ are perceptron layer with non-linear *Relu* activation. Learning in implicit feedback of NCF using the Backpropagation algorithm and Adam optimization for the objective function.

$$\hat{y}_{u_i v_j} = \frac{1}{1+e^{-score(u_i, v_j)}}$$

$$L(w) = -\frac{1}{|train|} \sum_{(u_i, v_j) \in train} y_{u_i v_j} Log(\hat{y}_{u_i v_j})$$

where $y_{u_i v_j}$ is a indicator variable, with $y_{u_i v_j} = 1$ when the user are recommended the item and otherwise. Therefore, the score interaction of users and items belongs to the range [0,1], and the model output is true or false recommended. However, data interaction of users and items is biased with large units with $y_{u_i v_j} = 1$. At the end, it will show the model using features of the user and item to learn with explicit data, that transform based on the hours of user interact with item.

3.2 Neural Matrix Factorization (NeuMF)

Following the section, we discuss other models using the element-wise layer which is a linear layer to decision similarity between user and item, and MLP (in framework NCF) a non-linear layer to learn interaction between user and item. The element-wise and MLP layers share embedding of the user and item shown in Figure 17.

Let K be the latent factor in MLP embedding, and T be the latent factor element-wise factor.P_u^{mlp}, P_u^{elw} are matrix embedding of the user, and Q_v^{mlp},Q_u^{elw} be matrix embedding of the item as for MLP and element-. The learning formula of NewuMF is shown below

$$\phi_{MLP}(u_i, v_j) = \phi_x..(\phi_2(\Phi_1([u_i P_{u_i}^{mlp}, v_j P_{v_j}^{mlp}])))$$

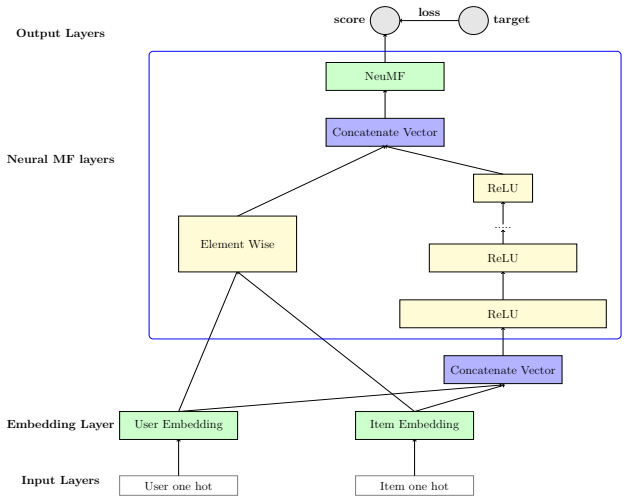


Figure 16 . Neural Matrix Factorization Architecture

$$\phi_{ELW}(u_i, v_j) = (u_i P_{u_i}^{elw}) \odot (v_j Q_{v_j}^{elw})$$

$$\hat{y}_{u_i v_j} = \phi_{NeuMF}(u_i, v_j) = W \begin{bmatrix} \phi_{MLP}(u_i, v_j) \\ \phi_{ELW}(u_i, v_j) \end{bmatrix} + b$$

Learning of NeuMF using implicit data with the input of user and item just one hot vector of user and item. Then throughout the MLP layer and ELW layer learn the interaction and similarity of users and items. the loss function of NeuMF also uses the Logloss function between implicit feedback and prediction. and the backpropagation algorithm and Adam algorithm for the training model. Instead of, mapping the user and item in latent space by MLP, embedding element-wise learning correlated the user and item in the latent factor. That can give efficient learning implicit compared with MLP, the result of the two models is shown in figure 18.

3.3 NCF Using Feature (NCFF)

By the framework of Neural Collaborative Filtering (NCF) in the previous section, we can apply feature vectors of users and items instead of identical values of users and items. This ideal can solve cold start problems (problems with new users who have no information about items) by gaining information from users during interaction with the recommendation system. Following the section, show the formula learning with features of NCF that are the same with wide deep architecture in [9] .

Let x_u, x_v be the features vector of users and items, and t_v be the tag of the item (category of item) represented by one hot vector, as well as t_u be the vector distribute of tags of the user. W_{MXA}, W_{NXB} are Linear weight matrices of embedding continuous features of users and items (note that continuous feature normalizing and not using discretization process in section 1.2). And $L \in R^{CXZ}$ are matrix embedding of feature tags of users and items with C being the number of tags and Z being the dimension of embedding features of tags.

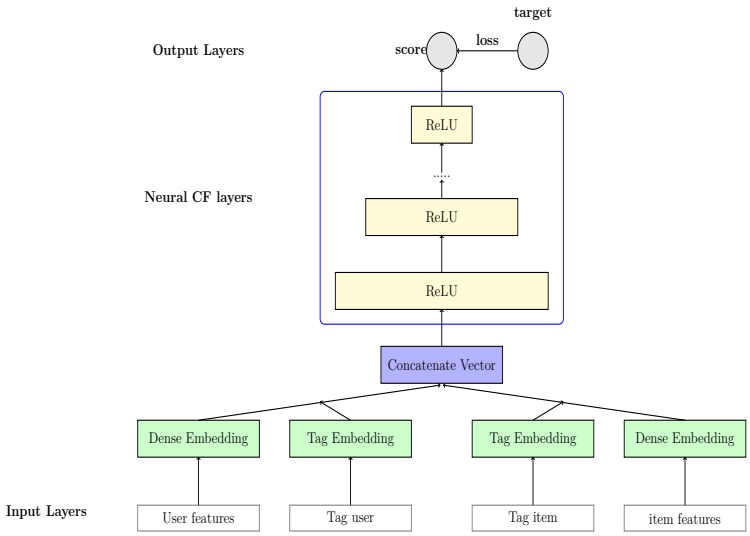


Figure 17 . Neural Collaborative Feature Architecture

$$y_{u_i v_j} = \phi_X(...\phi_2(\phi_1(\begin{bmatrix} [BN(x_{u_i} W_{MXA} + b), t_{u_i} L] \\ [BN(x_{v_j} W_{NXB} + b), t_{v_j} L] \end{bmatrix} + b)))$$

The score prediction of the user and item interaction is the same as NCF in hidden. However, the Input layer is a linear weight layer to learn features from the user and item . After that, feature embedding will be batch-normalized stability in the process of learning. Training NCFFV is the same as NCF, which uses the Backpropagation and Adam algorithm.

$$L(W, b) = \sqrt{\frac{1}{|train|} \sum_{(u_i, v_j) \in train} (y_{u_i v_j} - \hat{y}_{u_i v_j})^2}$$

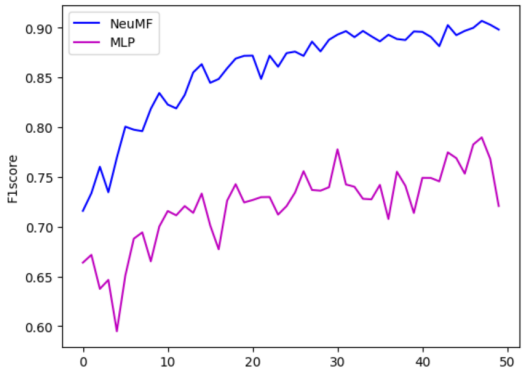
with $W \in \theta_{model}$ (parameter in the model).

This proposed method, it not use the implicit feedback label of the user, because of bias in implicit feedback with a large of positive feedback. Instead based on the hourly interaction of a user with a game transforms to explicit feedback with a ranking score label in [1, 1.5, 2, 2.5, 3,3.5, 4, 4.5, 5] that showed in preprocessing of data in the previous section. The result of the features model is shown in table 9. The limitations of the deep learning approach for recommendation models include not fully understanding the model's mechanisms and learning the latent factors between users and items, making it challenging to adjust parameters such as the number of nodes per layer, depth, and width of the model, as well as tuning model parameters. Therefore, difficulties arise in adjusting these parameters in models

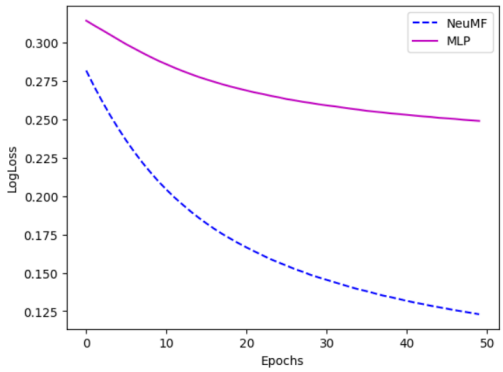
3.4 Empirical results

Table 9 . Performance Results MLP, NeuMF (8 latent factors for) in implicit data

Evaluation Metric	**NeuMF**	**MLP**
LogLoss	0.123	0.249
F1 Score@10	0.90	0.51
NDCG@10	0.92	0.59
	NCFF (explicit)	
RMSE	1.03	
F1 Score@10	0.76	
NDCG@10	0.9	



(a) F1score with top 10 recommendation in train



(b) LogLoss

Figure 18 . Comparision performance of NeuMF and MLP in F1score and Logloss

4 Evaluation metrics

RMSE

Root Mean Square Error (RMSE) is a commonly used metric to measure the accuracy of a recommender system by quantifying the difference between predicted and actual ratings. It is calculated as the square root of the average of the squared differences between each predicted rating ($\hat{y}_i$) and the corresponding actual rating (y_i). The formula for RMSE is:

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

where n is the number of ratings. In the context of recommender systems, RMSE helps in understanding how closely the predicted ratings match the actual user preferences, making it a valuable tool for evaluating the performance of rating predictions.

LogLoss

Log loss is a simple loss metric in binary classification problems, that apply in training with implicit feedback with labels including 1 and 0.

$$Logloss = -\frac{1}{N}\sum_{i}^{N} y_i Log(\hat{y}_i)$$

F1@k

The F1 Score@k is a metric that combines precision and recall at a specific cutoff rank k to provide a single measure of ranked recommendation tasks. For implicit rating with binary value, Precision@k ($P@k$) measures the proportion of true positive recommendations among the top k recommendations, while recall@k ($R@k$) measures the proportion of true positive recommendations among all relevant items. While for explicit rating,threshold is used as a benchmark between true and false labels, as in implicit rating. The F1 Score@k is the harmonic mean of precision@k and recall@k, and its formula is:

$$\text{F1@k} = 2 \cdot \frac{P@k \cdot R@k}{P@k + R@k}$$

where precision@k ($P@k$) is $\frac{\text{True Positives in top } k}{k}$ and recall@k ($R@k$) is $\frac{\text{True Positives in top } k}{\text{Total Relevant Items}}$. The F1 Score@k is particularly important in scenarios where the balance between precision and recall at the top k recommendations is crucial, such as recommending the most relevant items to users while minimizing irrelevant suggestions.In this report, we choose k =5, threshold = 3.5 for F1-score.

NDCG@k

If $F1@k$ only cares about which items are in the top K, then $NDCG@k$ cares about the order of each item in the top list of the recommender system. This is extremely important because in practice it demonstrates that users often pay close attention to the first k recommended products and consistently prioritize the first item suggested.

NDCG@k (Normalized Discounted Cumulative Gain) measures the quality of ranked recommendations in a recommender system. It considers the relevance and rank of recommended items, emphasizing the importance of placing relevant items higher in the list. The Discounted Cumulative Gain (DCG) at position p is calculated as:

$$\text{DCG@5} = \sum_{i=1}^{5} \frac{rel_i}{\log_2(i+1)}$$

where rel_i is the relevance score of the item at position i. NDCG@5 normalizes DCG by the ideal DCG (IDCG) at 5, which is the DCG for the ideal ranking:

$$\text{NDCG@5} = \frac{\text{DCG@5}}{\text{IDCG@5}}$$

NDCG@5 ranges from 0 to 1, with 1 indicating a perfect ranking. This metric is particularly useful for evaluating the top-k recommendations, ensuring that the most relevant items are recommended to users. We also choose threshold = 3.5 and k = 5 for this evaluation metric.

Part V
Conclusion

We are implementing algorithmic approaches to solve recommendation problems in gaming, as well as evaluating data processing. The challenges in recommendation problems have been manifested in this project, from sparsity in user interactions to the fact that these basic algorithms do not address the cold start problem. This project provides the most comprehensive overview of the recommendation problem and its approaches. Currently, researchers in the field are still exploring algorithms to optimize user experience.

Further work

This project provides a foundational approach to developing a recommender system on Steam and various preprocessing techniques. Our attempts focus on the warm-start problem, assuming sufficient users and items available in train data. Further work could explore strategies to handle the cold-start problem by our models or other techniques such as a hybrid model that combines many ideas and models.

Besides, the current model does not consider how user preferences and game popularity change over time, this is our big weak point. Incorporating temporal aspects into the model could help us to improve recommendations by accounting for evolving trends and user behavior. This could be achieved through a time-aware recommendation model or some sequence-based approaches like recurrent neural networks(RNNs).

Furthermore, we could enrich our dataset with additional information gathered from Steam to have a deeper and clearer insight into Steam as well as game behaviors among the young. This enriched data set can be leveraged to employ a more advanced and complex model, enabling a more comprehensive solution to this problem. There are numerous modern and intriguing techniques, which we can have gleaned from [11], or some of the cutting-edge deep learning approaches relevant today, such as the use of diffusion models [13] or variational autoencoders(VAEs)[12] in recommender systems by research groups in Netflix in 2018. In general, with the trend of complex and large-scale models, many approaches are proposed to address the problem in many criteria, aligning more closely with real-world systems rather than solely relying on conventional evaluation metrics. This is most evident in the increased profitability of companies using such systems.

Acknowledgment

We would like to express our sincere gratitude to Professor Than Quang Khoat, our instructor for this final semester project, for giving us invaluable knowledge throughthe Machine Learning course. We also extend our heartfelt thanks to Teacher Assistant Hoang Van An for his unwavering supports in practical lessons. Besides, giving respect to community research in field of recommendation problems, whose public research material in the conference and article about artificial intelligence and machine learning. Finally, this report is heard by all members of our team.

References

[1] Billsus, D. and Pazzani, M. J. (1998) "Learning Collaborative Information Filters." Proceedings of the 15th International Conference on Machine Learning, Madison, 26-30 July 1998, pp. 46-54.

[2] Y. Koren, "Factor in the neighbors: Scalable and accurate collaborative filtering," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 1, pp. 1–24, 2010.

[3] Lee, D. D., Seung, H. S. (2000). Algorithms for Non-negative Matrix Factorization. *Advances in Neural Information Processing Systems 13 (NIPS 2000)*, pp. 556–562.

[4] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian Personalized Ranking from Implicit Feedback," *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI2009)*, pp. 452-461, 2009.

[5] A. P. Singh and G. J. Gordon, "Relational Learning via Collective Matrix Factorization" *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 650-658, 2008.

[6] D. Cortes, "Cold-start recommendations in Collective Matrix Factorization", *arXiv preprint arXiv:1809.00366*, 2018.

[7] Rendle,S.(2010). "Factorization machines", *2010 IEEE International Conference on Data Mining (pp. 995–1000).*

[8] Xiangnan He,Lizi Liao,Hanwang Zhang (2017) "Neural Collaborative Filtering" *International World Wide Web Conference Committee*

[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen (2016) " Wide Deep Learning for Recommender Systems"

[10] YuefengZhang,yuefeng.zhang@pku.edu.cn (2022) "An Introduction to Matrix Factorization and Factorization Machines in Recommendation System, and Beyond"

[11] Ricci, F., Rokach, L., and Shapira, B. (Eds.). (2015). *Recommender Systems Handbook* (2nd ed.). Springer.

[12] Liang, D., Altosaar, J., Charlin, L., & Blei, D. (2018). Variational Autoencoders for Collaborative Filtering. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, Stockholm, Sweden, Volume(Issue), Page range.

[13] Wang, W., Xu, Y., Feng, F., Lin, X., He, X., Chua, T.-S. (2023). *Diffusion Recommender Model.* arXiv preprint arXiv:2304.04971.