ENHANCING THE RECOMMENDATION SYSTEM WITH ATTENTION MECHANISM USING BP NEURAL NETWORK

A PROJECT REPORT

Submitted by

HEMALATHA E (422420104304)

MONISHA S (422420104022)

SNEHA S (422420104035)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING TINDIVANAM

ANNA UNIVERSITY: CHENNAI 600 025

JUNE 2023

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"ENHANCING THE RECOMMENDATION SYSTEM WITH ATTENTION MECHANISM USING BP NEURAL NETWORK"** is the bonafide work of **"HEMALATHA E (422420104304), MONISHA S (422420104022), SNEHA S (422420104035)"** who carried out the project work under my supervision.

SIGNATURE

Dr. L. Jegatha Deborah, M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Assistant Professor,

Department of CSE,

University College of Engineering

Tindivanam,

Melpakkam- 604001.

SIGNATURE

Dr. R. Karthika, M.E.,Ph.D.,

SUPERVISIOR

Assistant Professor,

Department of CSE,

University College of Engineering

Tindivanam,

Melpakkam-604001.

Submitted for the university examination held on ________________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

At the outset we like to express our gratitude to the god almighty. We express our thanks to our Dean **Dr. P.THAMIZHAZHAGAN, M.E., Ph.D.,** for his constant inspiration and encouragement throughout the project.

We pay our grateful acknowledgement and extend our sincere gratitude to our Head of the Department **Dr. L. JEGATHA DEBORATH, M.E., Ph.D.** We also extend our sincere thanks to **Dr. R. KARTHIKA, M.E., Ph.D.,** project internal guide for better guidance and constant encouragement in completing this project work successfully.

We extend our thanks to the project coordinator **Ms. P. SATHYA, M.E.,** for her continual support during the entire project schedule. We thank all the teaching and non- teaching faculty members of the department and also our fellow friends for helping us in providing valuable suggestions and timely ideas for the successful completion of the project.

Last but not least, we extend our immense thanks to our parents and siblings who have been a great source of inspiration and rendered us great support during the course of this project work. We sincerely thank all of them.

ABSTRACT

This work presents a recommender framework based on a back propagation neural network with an attention mechanism. Generally deep neural network is used in recommender system to produce accurate prediction. But it results in high computational, storage costs, and overfitting issues caused by a smaller number of ratings that fed into DNN. To overcome these issues BP neural network with an attention mechanism (BPAM++) is proposed. BPAM++ can not only reduce the high computational, storage costs, and overfitting issues. BP neural network is also responsible for learning the underlying relationship between users, items, and preferences. The user's previous interactions with the things, the features of the items, and contextual information are some examples of the aspects that can be automatically identified by an attention mechanism as having the most impact on the recommendations. An attention mechanism can be used to capture the global impact of the nearest users of the target users on their nearest target user sets. Overall BPAM++ is used to improve the accuracy of the predicted ratings in the recommendation system.

Key words: Recommender system, BP neural network, attention mechanism, overfitting issue, data sparsity.

சுருக்கம்

இந்த வேலை ஒரு கவனம் பொறிமுறையுடன் பாக் புரோபகேஷன் நியூரல் நெட்வொர்கை அடிப்படையாகக் கொண்ட ஒரு பரிந்துரை கட்டமைப்பை முன்வைக்கிறது. பொதுவாக டீப் நியூரல் நெட்வொர்க் துல்லியமான கணிப்பை உருவாக்க பரிந்துரை அமைப்பில் பயன்படுத்தப்படுகிறது. ஆனால் இது அதிக கணக்கீட்டு, சேமிப்பக செலவுகள் மற்றும் டி.என். என்-க்கு அனுப்பப்படும் குறைந்த எண்ணிக்கையிலான மதிப்பீடுகளால் ஏற்படும் அதிகப்படியான சிக்கல்களை விளைவிக்கிறது. இந்த சிக்கல்களை சமாளிக்க பிபி நியூரல் நெட்வொர்க் ஒரு கவனம் பொறிமுறையுடன் (பிபிஏஎம்++) முன்மொழியப்பட்டது. பிபிஏஎம்++ அதிக கணக்கீட்டு, சேமிப்பக செலவுகள் மற்றும் அதிகப்படியான சிக்கல்களைக் குறைக்கின்றது. பயனர்கள், உருப்படிகள் மற்றும் விருப்பங்களுக்கிடையேயான அடிப்படை உறவைக் கற்றுக்கொள்வதற்கும் பிபி நியூரல் நெட்வொர்க் பொறுப்பாகும். இலக்கு பயனர்களின் அருகிலுள்ள பயனர்களின் உலகளாவிய தாக்கத்தை அவர்களின் அருகிலுள்ள இலக்கு பயனர் தொகுப்புகளில் கைப்பற்ற ஒரு கவனம் பொறிமுறையைப் பயன்படுத்தலாம். பரிந்துரை அமைப்பில் கணிக்கப்பட்ட மதிப்பீடுகளின் துல்லியத்தை மேம்படுத்த ஒட்டுமொத்த பிபிஏஎம்++ பயன்படுத்தப்படுகிறது.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S.NO	ABBREVIATIONS	DEFINITIONS
1	BP	Back Propagation
2	DNN	Deep neural network
3	MF	Matrix Factorization
4	CF	Collaborative Filtering
5	NeuCF	Neural Collaborative Filtering
6	DMF	Deep matrix factorization
7	DeepCF	Deep Collaborative Filtering
8	CDL	Collaborative deep learning
9	KNU	*k* Nearest User
10	LNI	*l* Nearest Item
11	BPAM++	BP neural network with attention mechanism

CHAPTER 1

INTRODUCTION

The recommendation system is defined as the computer program that helps the user determine goods and content by predicting the users' rating of each item and presenting them the substance that they would rate highly. The types of recommendation system are content-based filtering, collaborative filtering, hybrid filtering.

1.1 CONTENT-BASED FILTERING

Many of the product's features are required to implement content-based filtering instead of user feedback or interaction. It is a machine learning technique that is used to decide the outcomes based on product similarities. Content-based filtering algorithms are designed to recommend products based on the accumulated knowledge of users. This technique is all about comparing user interest with product features, so providing a significant product feature in the system is essential. It should be the priority before designing a system to select the favorite features of each buyer. These two strategies can be applied in a possible combination. Firstly, a list of features is provided to the user to select the most interesting features.

Secondly, the algorithms keep a record of all the products chosen by the user in the past and make up the customer's behavioral data. The buyer's profile rotates around the buyer's choices, tastes, and preferences and shapes the buyer's rating. It includes how many times a single buyer clicks on interested products or how many times liked those products in wish lists. Content-based filtering consists of a resemblance between the items. The proximity and similarity of the product are measured based on the similar content of the item.

1.2 COLLABORATIVE FILTERING

Collaborative filtering is a technique used in recommender systems to generate recommendations by leveraging the opinions and behaviors of multiple users. It is based on the assumption that users with similar preferences or behaviors in the past will have similar preferences in the future.The main idea behind this approach is to suggest new items based on the closeness in the behavior of similar customers. There are two main types of collaborative filtering: User-based Collaborative Filtering, Item-based Collaborative Filtering.

Collaborative filtering needs a set of items that are based on the user's historical choices. This system does not require a good amount of product features to work. An embedding or feature vector describes each item and user, and it sinks both the items and the users in a similar embedding location. It keeps track of the behavior of all users before recommending which item is mostly liked by users. It also relates similar users by similarity in preference and behavior towards a similar product when proposing a product to the primary customer. The traditional CF recommendation algorithms are generally divided into two categories those are, matrix factorization method, neighborhood-based CF method.

1.2.1 MATRIX FACTORIZATION METHOD

In a recommender system context, the user matrix represents user preferences or behaviors, and the item matrix represents item attributes or features. By decomposing the original matrix into these lower-rank matrices, matrix factorization aims to capture latent factors or hidden features that influence user-item interactions. The MF methods map the users and items into a common representation space. However, the MF methods easily suffer from sparsity issues.

1.2.2 NEIGHBORHOOD-BASED CF METHOD

By introducing the BP neural network into the neighborhood-based CF algorithm, the ratings of the nearest users and items are fed into the BP neural network instead of undertaking a linear combination in the traditional algorithms. A BP neural network-based recommender framework with an attention mechanism is a powerful and flexible tool for a personalized recommendation. By learning from user-item interactions and attending to important features, it can make accurate and relevant recommendations that improve user satisfaction and engagement.

The input layer takes in user and item features, such as ratings of nearest users and nearest items then it will be encoded into a numerical representation. The hidden layers use nonlinear transformations to extract complex features from the input and compute the process while the output layer predicts the rating or probability of a user interacting with an item. The training process of BPAM++ is to alternately iterate the forward propagation and the error back propagation until the stopping condition is reached. The attention mechanism in this framework allows the model to focus on important features of the input while ignoring irrelevant or noisy information. Attention can be thought of as a spotlight that shines on certain parts of the input while dimming or ignoring others. This is accomplished by assigning weights to different features based on their importance to the task at hand.

The attention mechanism is incorporated into the BP neural network to capture the global impact of the nearest users of the target user on their nearest target user sets. Overall, a bp neural network-based recommender framework with an attention mechanism is a powerful and flexible tool for a personalized recommendation. By learning from user-item interactions and attending to important features, it can make accurate and relevant recommendations that improve user satisfaction and engagement.

CHAPTER 2

LITERATURE SURVEY

1. C.-D. Wang, Z.-H. Deng, and P. S. Yu, “Serendipitous recommendation in Ecommerce using innovator-based collaborative filtering,”IEEE Trans. Cybern., vol. 49, no. 7, pp. 2678–2692, Jul.2019 [1].

Collaborative filtering (CF) algorithms have been widely used to build recommender systems since they have the distinguishing capability of sharing collective wisdom and experiences. It aims to provide a personalized and delightful shopping experience by going beyond the user's explicit preferences and introducing them to new and relevant products. However, they may easily fall into the trap of the Matthew effect, which tends to recommend popular items, and hence less popular items become increasingly less popular. Under this circumstance, most of the items in the items, i.e., the recommendation list are already familiar to users and therefore the performance would seriously degenerate in finding cold new items and niche items. To address this issue, in this paper, a user survey is first conducted on online shopping habits in China, based on which a novel recommendation algorithm termed innovator-based CF is proposed that can recommend cold items to users by introducing the concept of innovators. Innovators are a special subset of users who can discover cold items without the help of recommender system. Therefore, cold items can be captured in the recommendation list via innovators, achieving the balance between serendipity and accuracy.

MERITS: It increases the diversity of recommendations. This can help expand users’ horizons and provide a more engaging and personalized shopping experience.

DEMERITS: Risk of irrelevant recommendations. Difficulty In implementation.

It can raise privacy concerns.

2. Q.-Y. Hu, L. Huang, C.-D. Wang, and H.-Y. Chao, "Item orientated, recommendation by multi-view intact space learning with overlapping," Knowl.-Based Syst., vol. 164, pp. 358–370, 2019. [3]

An item-orientated recommendation has recently been proposed to address the issue of finding the most suitable users of a certain item to maximize the revenue of its manufacturer with a limited advertising budget, which cannot be settled very well by the traditional recommendation algorithms. Existing multi-view models are instructive to solve this problem but either need extra information or suffer from scalability issues due to the high complexity when dealing with a large number of records. This paper presents a solution to meet the need of improving both the quality and the scalability of item-orientated recommendations by designing a new multiview model. From the multiple but insufficient view representations, the latent feature representation of each item in the intact space, the view generation function of each user, and the overlapping of preferences among different users can be discovered, which are used to predict the unknown ratings from users to a certain item. Extensive experiments have been conducted on three types of datasets, and the results have confirmed the effectiveness of the proposed.

MERITS: It is used to improve accuracy. Robustness to missing data. Ability to handle diverse data.

DEMERITS: This approach is more complex. Difficult to interpret. It is sensitive to noisy or irrelevant data.

3. H.-J. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in Proc. Int. Joint Conf. Artif. Intell., pp. 3203–3209, 2017 [11].

Recommender systems usually make personalized recommendations with user-item interaction ratings, implicit feedback, and auxiliary information. Matrix factorization is the basic idea to predict a personalized ranking over a set of items for an individual user with the similarities among users and items. In this paper, we propose a novel matrix factorization model with neural network architecture. we construct a user-item matrix with explicit ratings and non-preference implicit feedback. With this matrix as the input, we present a deep structure learning architecture to learn a common low-dimensional space for the representations of users and items. Secondly, the experimental results show the effectiveness of both our proposed model and the loss function. On several benchmark datasets, our model outperformed other state-of-the-art methods. We also conduct extensive experiments to evaluate the performance within different experimental settings.

MERITS: It gives better prediction accuracy. It can handle sparse data. It works efficiently on large-scale datasets. Enables the system to provide highly relevant recommendations.

DEMERITS: This can be time consuming and requires date expertise preprocessing techniques. It struggles with cold start problems. It results in overfittingissues.

4. X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in Proc. Int. Conf. World Wide Web, pp.173-182, 2017 [13].

The key idea behind NCF is to learn user and item embeddings (low-dimensional representations) using neural networks. These embeddings encode the latent factors and capture the underlying relationships between users and items. By leveraging this learned representation, the model can make accurate predictions and generate personalized recommendations. The NeuCF model utilizes a multi-layer perceptron (MLP) to learn the non-linear interactions between user and item embeddings. It also incorporates a matrix factorization component to capture the linear interactions. These two components are combined through element-wise multiplication and concatenated to form the final prediction. Overall, the "Neural Collaborative

Filtering" paper introduces a novel model that integrates neural networks into collaborative filtering to improve recommendation performance. It provides insights into the potential of using deep learning techniques in recommendation systems and contributes to the advancements in the field of personalized recommendation algorithms.

MERITS: Provide personalization recommendations. It can handle large-scale datasets. It overcomes the cold start problem.

DEMERITS: Sparse data can result in less effective learning of latent factors and may lead to suboptimal recommendations.It requires a large amount of data to train effectively.

5. H.Wang , N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in Proc. 21st ACM SIGKDD Int. Conf. Knowl . Discov. Data Mining, pp. 1235–1244, 2015 [21].

Collaborative filtering (CF) is a successful approach commonly used by many recommender systems. However, the ratings are often very sparse in many applications, causing CF-based methods to degrade significantly in their recommendation performance. To address this sparsity problem, auxiliary information such as item content information may be utilized. Nevertheless, the latent representation learned by CTR may not be very effective when the auxiliary information is very sparse. In this paper a hierarchical Bayesian model called collaborative deep learning (CDL), which jointly performs deep representation learning for the content information and collaborative filtering for the ratings (feedback) matrix. The key idea behind CDL is to represent users and items as low-dimensional embeddings using neural networks. These representations capture the non-linear relationships and interactions between users and item. CDL incorporates a collaborative filtering component by using an inner product operation between the user and item embeddings. The model also incorporates an additional component that incorporates content-based features, enabling the integration of item attributes or textual information into the recommendation process.

MERITS: It produces accurate predictions. It can handle large-scale datasets with millions of users and items.

DEMERITS: This can be a challenge for smaller businesses or startups that don’t have larger datasets. it leads to a cold start problem. It is computationally expensive.

6. R. M. Bell and Y. Koren, “Scalable collaborative filtering with jointly derived neighborhood interpolation weights,” in Proc. 7th IEEE Int. Conf. Data Mining, pp. 43–52, 2007 [15].

Recommender systems based on collaborative filtering predict user preferences for products or services by learning past user-item relationships. A predominant approach to collaborative filtering is neighbor Recommender systems based on collaborative filtering predict user hood based ("k-nearest neighbors"), where a user-item preference rating is interpolated from ratings of similar items and/or users. We enhance the neighborhood-based approach leading to substantial improvement of prediction accuracy, without a meaningful increase in running time. First, we remove certain so-called "global effects" from the data to make the ratings more comparable, thereby improving interpolation accuracy. Second, we show how to simultaneously derive interpolation weights for all nearest neighbors, unlike previous approaches where each weight is computed separately. By globally solving a suitable optimization problem, this simultaneous interpolation accounts for the many interactions between neighbors leading to improved accuracy.

MERITS: It improves accuracy. It can be applied to different types of recommendation tasks. It can overcome the cold start problem.

DEMERITS: It is computationally intensive. If the user-item interaction data is sparse, the performance of the approach may be affected.

CHAPTER 3

PROBLEM STATEMENT

Traditional recommendation models may give inaccurate predictions due to data sparsity. Additionally, such models may suffer from high computational, storage costs and overfitting issues.

3.1 Objectives

The main objective of this proposed BPAM++ framework is to improve the accuracy of recommendations while reducing computational and storage costs. To achieve these objectives,

- Neighborhood-based CF algorithm is incorporated into BP neural network, this improves the accuracy of recommendation framework that can capture complex relationships between users and items.
- BP neural network can handle data sparsity and overfitting issues.
- In addition, an attention mechanism is introduced in BP neural network to focus on important features and reduce computational and storage costs.

3.2 Scope of the Project

The scope of a Back propagation neural network-based recommender framework with an attention mechanism can include:

E-commerce platforms: BP Neural Network-based recommendation systems can be used in e-commerce platforms to recommend products to users based on their browsing and purchasing history.

Social Media: BP Neural Network-based recommendation systems can be used in social media platforms to recommend content to users based on their interactions with the platform.

Entertainment: BP Neural Network-based recommendation systems can be used in entertainment platforms to recommend movies, TV shows, and music to users based on their preferences.

Healthcare: BP Neural Network-based recommendation systems can be used in healthcare platforms to recommend treatments and therapies to patients based on their medical history.

Educational platforms: BP Neural Network-based recommendation systems can be used in educational platforms to recommend courses and learning materials to students based on their interests and previous learning experience.

3.3 System Description

The proposed recommendation framework called BPAM++, which is based on Back Propagation (BP) neural network with attention mechanism. By introducing the BP neural network into the neighborhood-based CF algorithm, the ratings of the nearest users and items are fed into the BP neural network. In addition, the attention mechanism is incorporated into the BP neural network to capture the global impact of the nearest users of the target user on their nearest target user sets. To this end, a global attention matrix is introduced, which enables the model to share weights across different target users and obtain more accurate weights of the target user from the nearest users. In summary, by introducing the attention mechanism, BPAM++ combines the local weight matrix and the global attention matrix to predict the missing ratings. The training process of BPAM++ is to alternately iterate the forward propagation and the error back propagation until the stopping condition is reached.

CHAPTER 4

REQUIREMENT ANALYSIS

The requirement analysis is the technical requirement of the software product. It is the first step in the requirement analysis process. It lists the requirement of particular software including functional and non-functional requirements. The requirements also provide useful constraints from a user, an operational and administrative perspective. It is a parameter which describes its user interface, hardware and software requirements.

4.1 Functional Requirements

Functional requirements specify which output file should be produced from the given file. They describe the relationship between the input and output of the system. For each fuctional requirement a detailed description of all data inputs and their source and the range of valid inputs must be specified.

4.1.1 Hardware Requirements

• Processor	- AMD PROA4
• Speed	- 2.50 GHz
• RAM	- 4GB

4.1.2 Software Requirements

• Operating System	- Windows or Linux
• Programming Language	- Python
• Tools	- Netbeans, XAMPP

4.2 Non-Functional Requirements

Describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Non-Functional requirements include quantitative constraints, such as response time(i.e. how fast the system reacts to user commands) or accuracy (i.e. how precise are the systems numerical answers).

- **Scalability**

  Program with larger number of lines can be obfuscated.

- **Performance**

  i. Speed, throughput and execution time depends on the size of the input dataset.

  ii. Improved performance when compared to previous models

- **Flexibility**

  The system shows great flexibility on python compiler.

CHAPTER 5

SYSTEM DESIGN

5.1 System Architecture

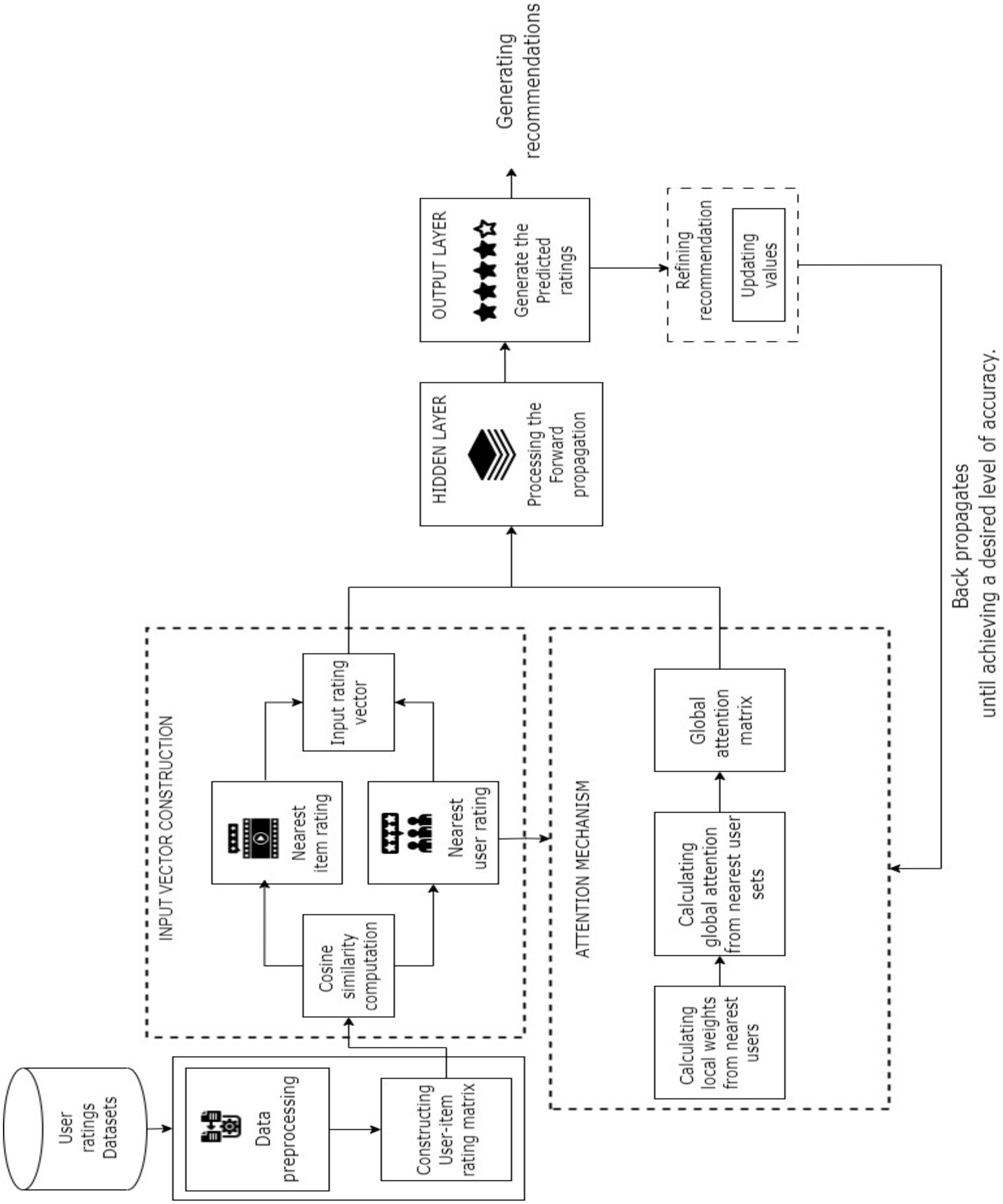

User ratings Datasets
Data preprocessing
Constructing User-item rating matrix
INPUT VECTOR CONSTRUCTION
Cosine similarity computation
Nearest item rating
Nearest user rating
Input rating vector
ATTENTION MECHANISM
Calculating local weights from nearest users
Calculating global attention from nearest user sets
Global attention matrix
HIDDEN LAYER
Processing the Forward propagation
OUTPUT LAYER
Generate the Predicted ratings
Generating recommendations
Refining recommendation
Updating values
Back propagates
until achieving a desired level of accuracy.

Fig 5.1 System Architecture

5.2 Module Description

- Input vector construction
- Attention mechanism
- Forward Propagation
- Back Propagation

5.2.1 Input vector construction

The input vector is constructed based on the rating matrix R, which contains user-item interactions for a set of users and items. For each user u, a set of k nearest users (KNU) and a set of k nearest items (LNI) can be obtained by calculating the cosine similarity between the rating vectors of users and items. The rating vector for each KNU and LNI is then concatenated to form a feature vector. This feature vector is used as input to a BPAM++ to generate personalized recommendations for user. Overall, this input vector construction process allows the model to capture important information from similar items while ignoring irrelevant features that may lead to overfitting issues.

Algorithm

Step 1: Extract rating matrix R from the database.

Step 2: Calculate the cosine similarity between each user's rating vector and all other users' rating vectors in R.

Step 3: Select k nearest users (KNU) for each user based on cosine similarity scores for each user u in R.

Step 4: Calculate the cosine similarity between each item's rating vector and all other items' rating vectors in R.

Step 5: Select k nearest items (LNI) for each item based on cosine similarity scores.

Step 6: For each user-item pair (u, i), create an input vector that includes the rating given by the user to the item, the ratings given by KNU to the item, and the ratings given by the user to LNI.

Pseudo code

```
R = extract_rating_matrix_from_database()
for each user u in R:
for each other_user v in R:
   similarity_score = calculate_cosine_similarity(u, v)
store_similarity_score(similarity_score)
for each user u in R:
   knu = select_k_nearest_users(u, k)
for each item i in R:
for each other_item j in R:
   similarity_score = calculate_cosine_similarity(i, j)
store_similarity_score(similarity_score)
for each item i in R:
   lni = select_k_nearest_items(i, k)
for each user-item pair (u,i) in R:
   knu_ratings=get_ratings_given_by_knu_to_item(knu,i)
   lni_ratings = get_ratings_given_by_user_to_lni(u, lni)
   input_vector = concatenate(rating_given_by_u_to_i, knu_ratings, lni_ratings)
```

5.2.2 Attention mechanism

In the proposed BPAM++ framework, an attention-based architecture is used to learn the complex relationship between target users and their neighbors and the complex relationship between given items and their neighbors. The attention mechanism works by assigning weights to different parts of the input vector based on their importance. It is a global attention mechanism that captures the global impact of the nearest users of the target user on their nearest target user sets. By focusing on important features, the model can generate more accurate predictions while minimizing computational and storage costs.

Algorithm

Step 1: Compute similarity scores between user u and all other users in R.

Step 2: Select k nearest users (KNU) for user u based on similarity scores.

Step 3: Compute local weights for KNU.

Step 4: Normalize local weights.

Step 5: Compute global attention weights of KNU ratings using local weights.

Pseudo code

```
for each other_user v in R:
    similarity_score = compute_similarity_score(u, v)
    store_similarity_score(similarity_score)
knu = select_k_nearest_users(u, k)
for each user v in knu:
    local_weight = compute_local _weight(u, v)
    store_ local _weight(local _weight)
normalized_ local _weights = normalize_ local _weights()
global_attentionweights_of_knu_ratings=computeglobal_attentionweights_of_
knu_ratings(knu, normalized_local _weights)
```

5.2.3 Forward Propagation

During forward propagation, the input vector is fed into the neural network, which consists of an input layer, a hidden layer, and an output layer. The input layer receives the input vector and passes it through a set of weights to generate a set of activations in the hidden layer. The activations in the hidden layer are then passed through another set of weights to generate a set of activations in the output layer. These activations are then transformed using an activation function (in this case, Sigmoid) to generate a predicted rating for each user-item pair. The output neuron then computes a weighted sum of its inputs, applies another activation function (linear), and produces the final prediction.

Algorithm

Step 1: Compute the input vector for user-item pair (u,i).
Step 2: Compute local weights.
Step 3: Compute global attention weights of KNU and LNI ratings using local weights.
Step 4: Concatenate the input vector with global attention weights of KNU and LNI ratings.
Step 5: Compute the output of the hidden layer using the input vector with attention.
Step 6: Compute the output of the output layer using hidden layer output.

Pseudo code

```
input_vector = compute_input_vector(u, i)
knu_local_weights = compute_local_weights_for_knu(u, i)
lni_local_weights = compute_local_weights_for_lni(u, i)
knu_global_attentionweights=compute_global_attentionweights_of_knu_ratings (knu, knu_local_weights)
lni_global_attentionweights=compute_ global_attentionweights_of_lni_ratings (lni, lni_local_weights)
input_vector_with_attention=concatenate(input_vector, knu_global_attentionweight, lni_ global_attentionweights)
hidden_layer_output=compute_hidden_layer_output(input_vector_with_attention)
predicted_rating = compute_output_layer_output(hidden_layer_output)
```

5.2.4 Back Propagation

The back propagation process in the BPAM++ recommender framework is used to update the weights of the neural network during training. In back propagation, each weight in the neural network is updated by a small amount proportional to its contribution to the overall error between predicted and actual ratings or ranking scores. The training process of BPAM++ is to alternately iterate the forward propagation and the error back propagation until achieving a desired level of accuracy.

Algorithm

Step 1: Compute the error between the predicted rating and the actual rating.

Step 2: Compute the derivative of output layer output with respect to error.

Step 3: Compute the derivative of hidden layer output with respect to output layer output.

Step 4: Compute the derivative of the input vector with attention with respect to the hidden layer output.

Step 5: Split the derivative of the input vector with attention to derivatives of the input vector, KNU ratings, and LNI ratings.

Step 6: Compute the derivative of KNU and LNI local weights with respect to their respective ratings.

Step 7: Update the global attention matrix using derivatives of KNU and LNI attention weights.

Step 8: Update the local weight matrix using derivatives of the input vector and hidden layer output.

Step 9: Update KNU and LNI ratings using derivatives of local weights and ratings.

Step 10: Repeat steps 1-9 for all training examples until convergence.

Pseudo code

```
error = compute_error(predicted_rating, actual_rating)
d_output_layer_output = compute_derivative_of_output_layer_output(error)
d_hidden_layer_output = compute_derivative_of_hidden_layer_output
 (output_layer_weights, d_output_layer_output )
d_input_vector_with_attention = compute_derivative_of_input_vector_with_
attention(hidden_layer_weights, d_hidden_layer_output)
d_input_vector = split_derivative_of_input_vector_with
_attention(d_input_vector_with_attention)
 d_knu_ratings = split_derivative_of_knu_ratings(d_input_vector_with_attention)
d_lni_ratings = split_derivative_of_lni_ratings(d_input_vector_with_attention)
d_knu_attention_weights = compute_derivative_of_knu_local_weights(knu_ratings,
d_knu_ratings)
 d_lni_attention_weights =
compute_derivative_of_lni_local _weights(lni_ratings, d_lni_ratings)
global_attentionweights+=learning_rate * (outer_product(d_knu_attention_weights)
+ outer_product(d_lni_attention_weights))
  local_weight_matrix  +=  learning_rate  *  outer_product(d_hidden_layer_output,
  d_input_vector)
for each user v in knu:     knu_ratings[v] += learning_rate *
dot_product(d_knu_attention_weights[v], lni_ratings)
 for each item j in lni:     lni_ratings[j] += learning_rate *
dot_product(d_lni_attention_weights[j], knu_ratings)
```

5.3 THE OVERALL ALGORITHM FRAMEWORK OF BPAM++

```
Input: Rating matrix: $R$; number of nearest users: $k$; number of nearest items: $l$; trade-off parameter: $\alpha$
Output: Model parameters: $\boldsymbol{\Theta}$
Randomly initialize model parameters:
    $\boldsymbol{\Theta}$={A,{$\mathrm{W}^{u}$, $\mathrm{b}_{input}^{u}$, $w_{hidden}^{u}$, $b_{hidden}^{u}$, $\forall$ $u$ ϵ U
Obtain the $k$ nearest users (KNU) of each user $u$, i.e., the KNU index vector $nu$=[ $nu_1$,$nu_2$,$nu_3$,……$nu_k$ ].
Obtain the $l$ nearest items (LNI) of each item $i$, i.e., the LNI index vector $ni$=[ $ni_1$,$ni_2$,$ni_3$,…...$ni_l$ ].
repeat
    for all $u$ ϵ U do
        Form the KNU rating vector $\mathrm{p}_{i}^{u}$ of user u on each item $i$.
        Form the LNI rating vector $\mathrm{q}_{i}^{u}$ of each item i from user $u$.
        Perform rating vector processing on $\mathrm{p}_{i}^{u}$ and $\mathrm{q}_{i}^{u}$.
        Construct the training sample $\mathrm{d}_{i}^{u}$ for each item $i$.
        for all $\mathbf{d}_{i}^{u}$ ϵ $D^{u}$ do
            Predict the rating $\hat{r}_{u,i}$
            Update $A_{n^{u*}}$, $\mathrm{W}^{u}$, $\mathrm{b}_{input}^{u}$, $w_{hidden}^{u}$, $b_{hidden}^{u}$
        end for
    end for
until converges.
```

5.4 UML DIAGRAMS

5.4.1 USE CASE DIAGRAM

A use case diagram is a graphical representation of the interactions between a system and its users or actors. It depicts the functionality of a system from the user's perspective by showing the different use cases or tasks that the system can perform and the actors that interact with the system to achieve those tasks.

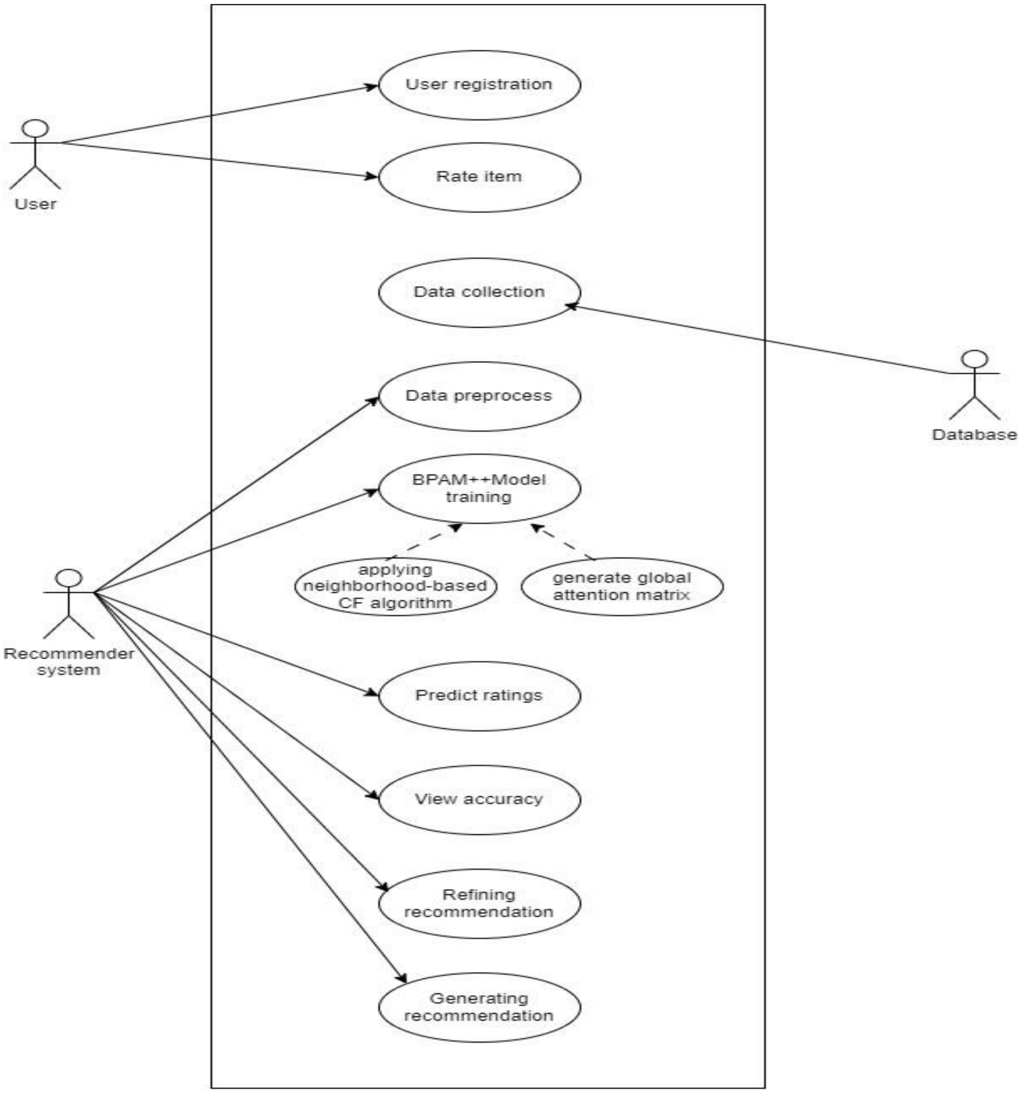


Fig 5.4.1 Use case diagram

5.4.2 CLASS DIAGRAM

Class diagrams depict the structure of a system by showing the classes, their attributes, and the relationships between them. They can help developers to understand the relationships between different objects in the system and how data flows between them.

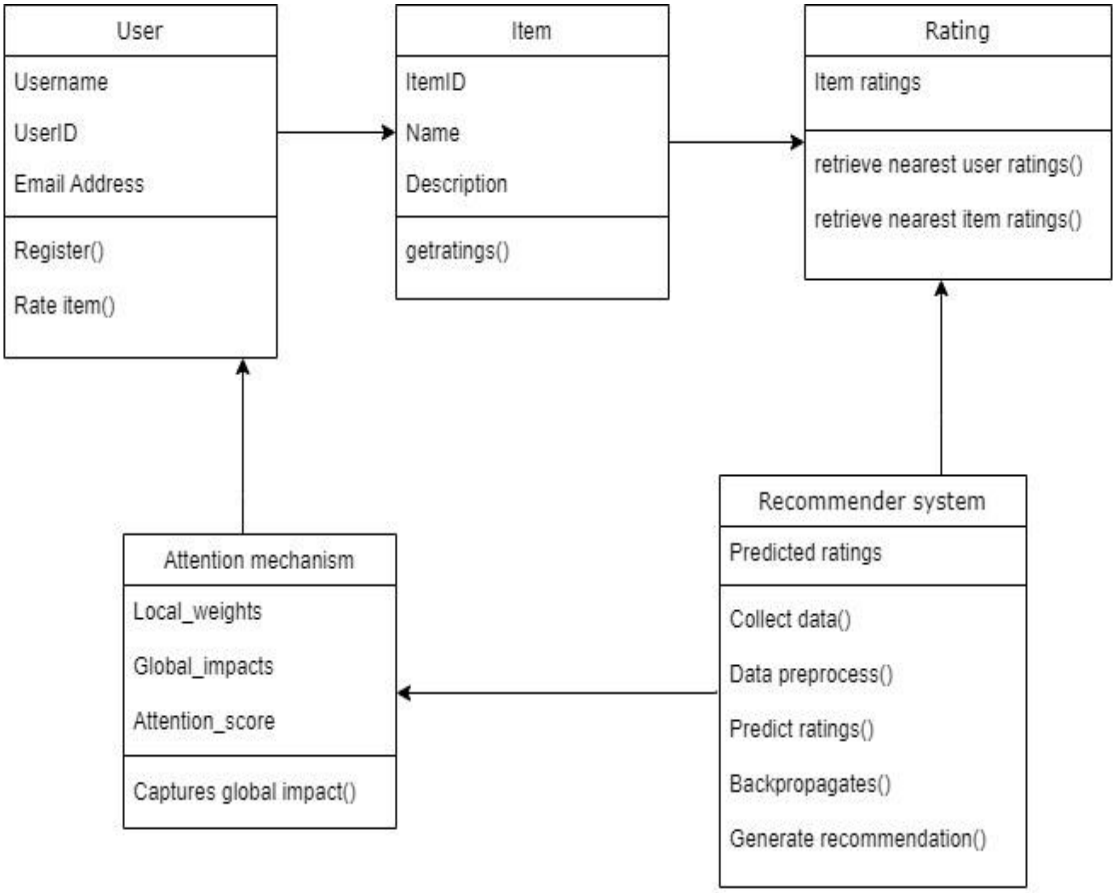


Fig 5.4.2 Class Diagram

5.4.3 SEQUENCE DIAGRAM

Sequence diagrams depict the interactions between objects in a system over time, showing the order in which messages are sent and received. They can help developers to understand the behavior of the system during different scenarios and identify potential issues.

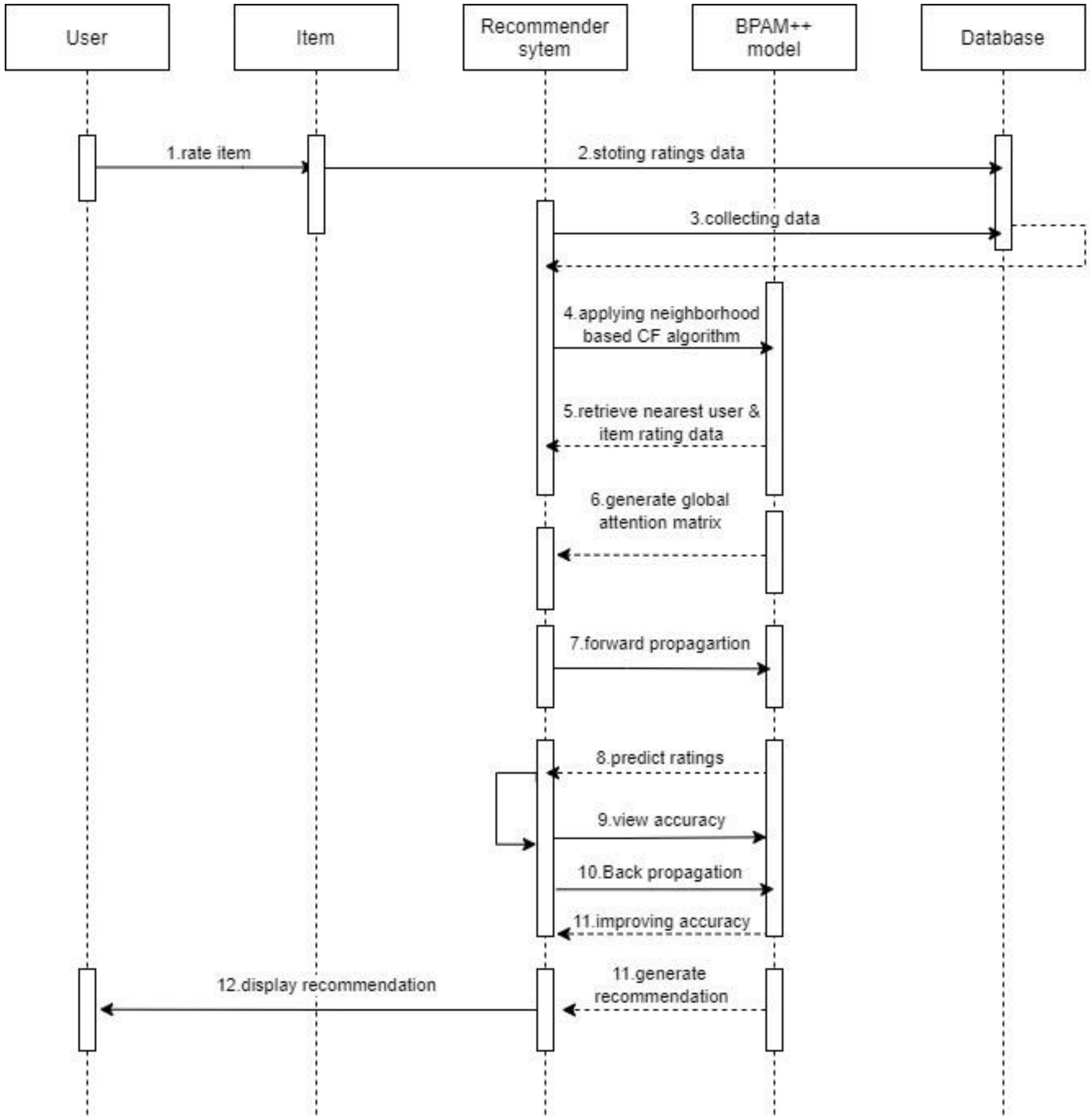


Fig 5.4.3 Sequence Diagram

5.4.4 COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is a type of UML diagram that is used to model the dynamic behavior of a system by illustrating the messages exchanged among objects or parts during the execution of a use case scenario.

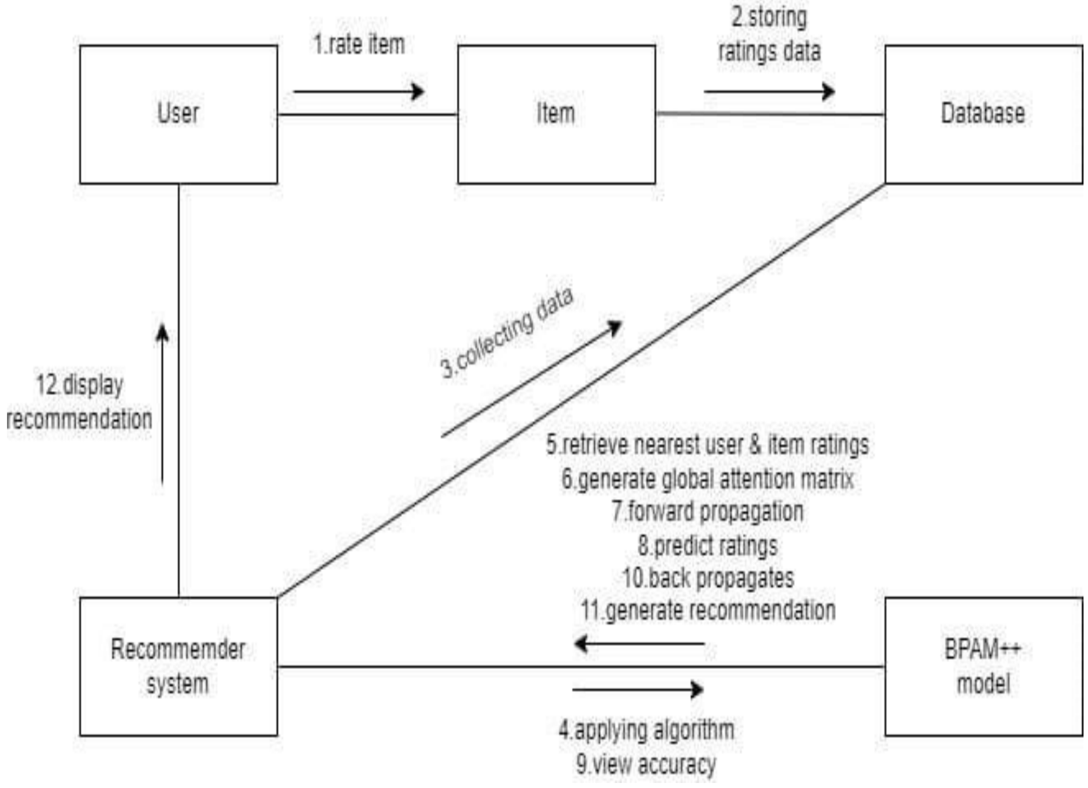


Fig 5.4.4 Collaboration Diagram

5.4.5 STATECHART DIAGRAM

State diagrams depict the states and transitions of objects in a system, showing how objects behave in response to events. They can help developers to understand the behavior of the system during different states and identify potential issues.

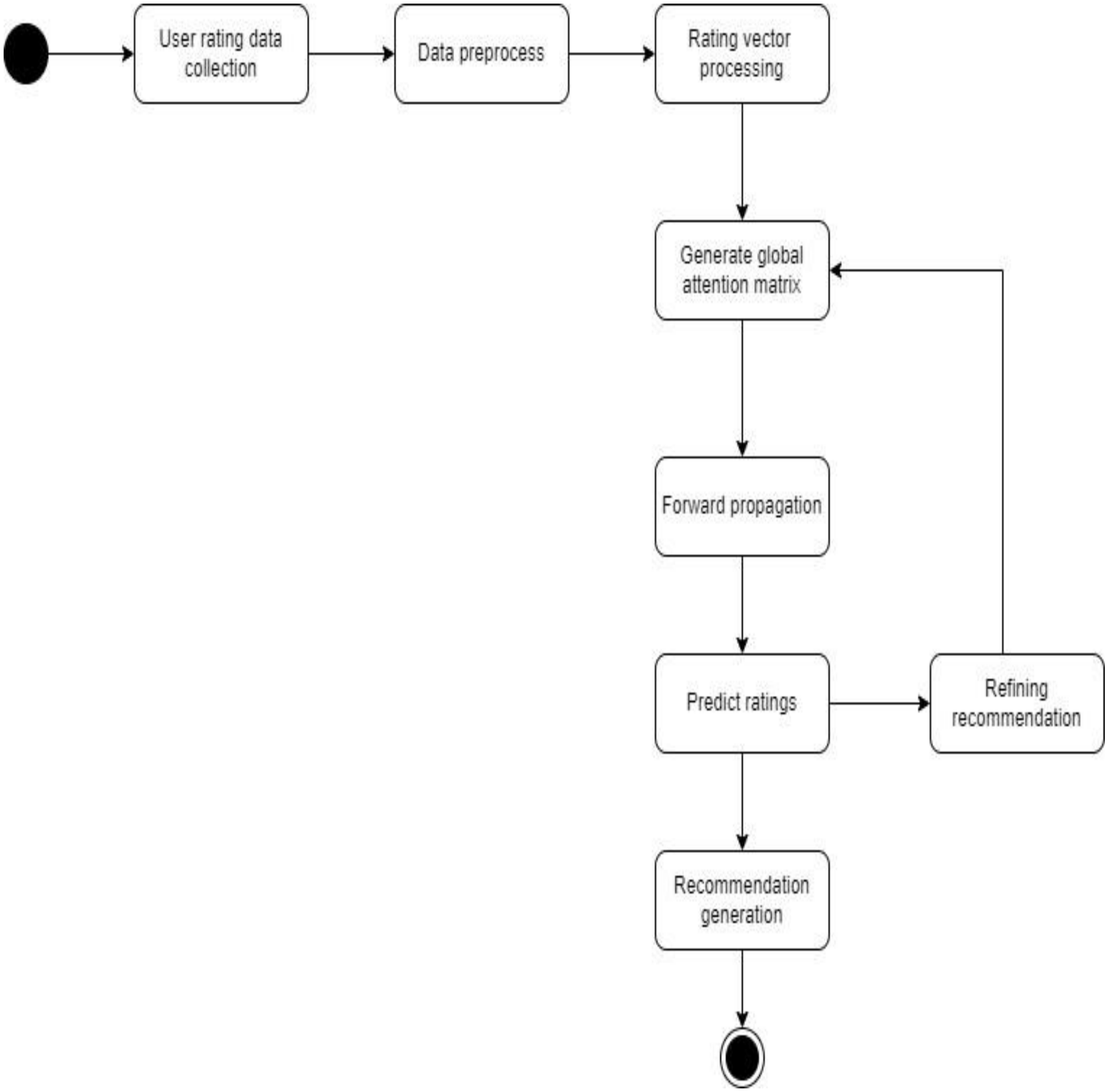


Fig 5.4.5 Statechart Diagram

5.4.6 ACTIVITY DIAGRAM

Activity diagrams depict the flow of control in a system by showing the activities and decisions involved in a process. They can help developers to understand the various steps involved in a process and identify potential bottlenecks or inefficiencies.

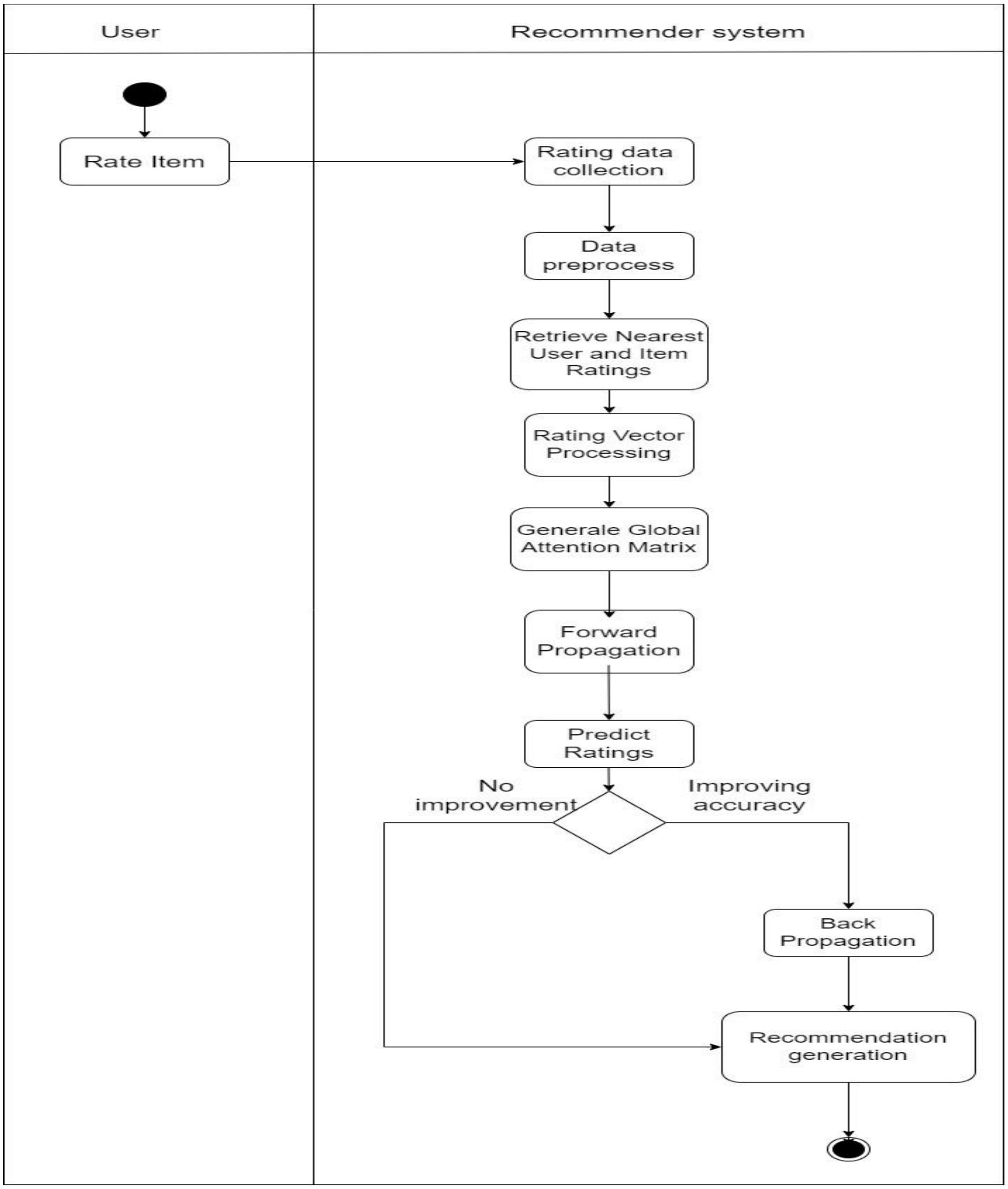


Fig 5.4.6 Activity Diagram

5.4.7 COMPONENT DIAGRAM

Component diagrams depict the physical and logical components of a system and the relationships between them. They can help developers to understand how the various components of the system work together and how changes to one component may impact others.

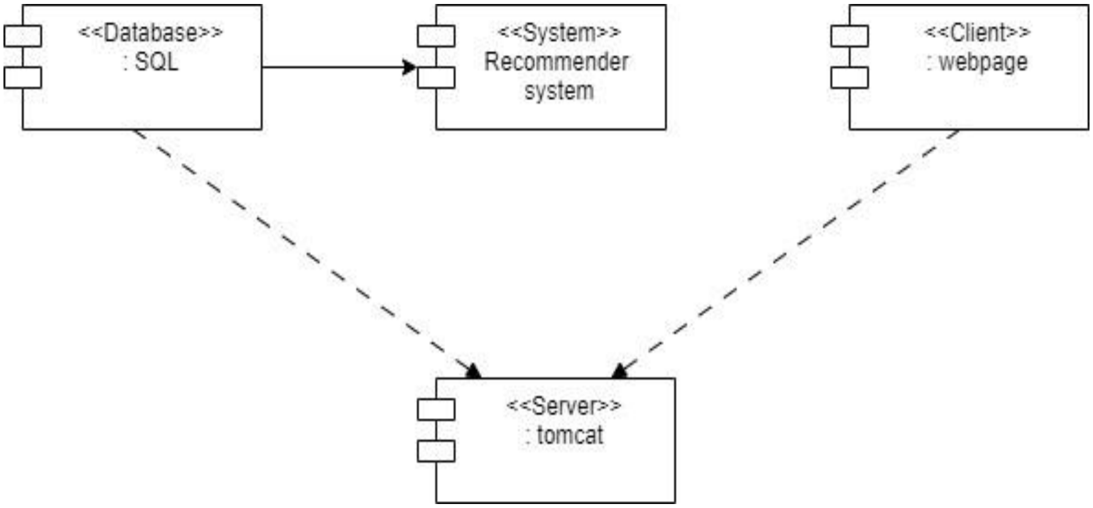


Fig 5.4.7 Component Diagram

5.4.8 DEPLOYMENT DIAGRAM

Deployment diagrams depict the physical deployment of a system's components on hardware or software platforms. They can help developers to understand how the system is physically deployed and identify potential issues with hardware or software configurations.

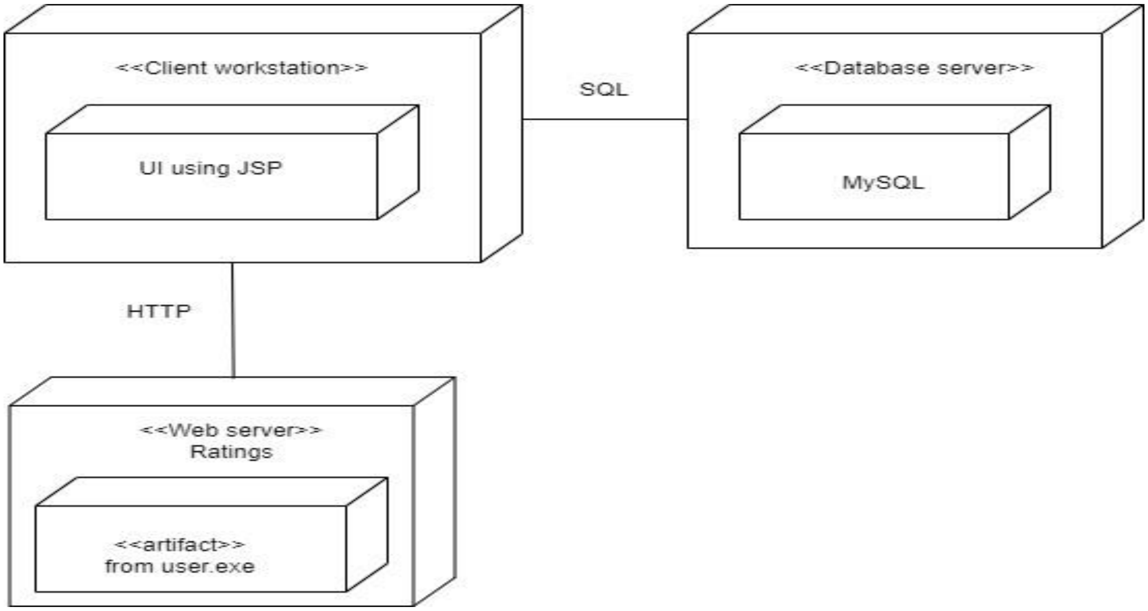


Fig 5.4.8 Deployment Diagram

5.4.9 PACKAGE DIAGRAM

A package diagram is a type of UML (Unified Modeling Language) diagram that shows the organization and dependencies among packages in a system. A package is a container for related classes, interfaces, and other packages, and is used to organize and structure the software system.

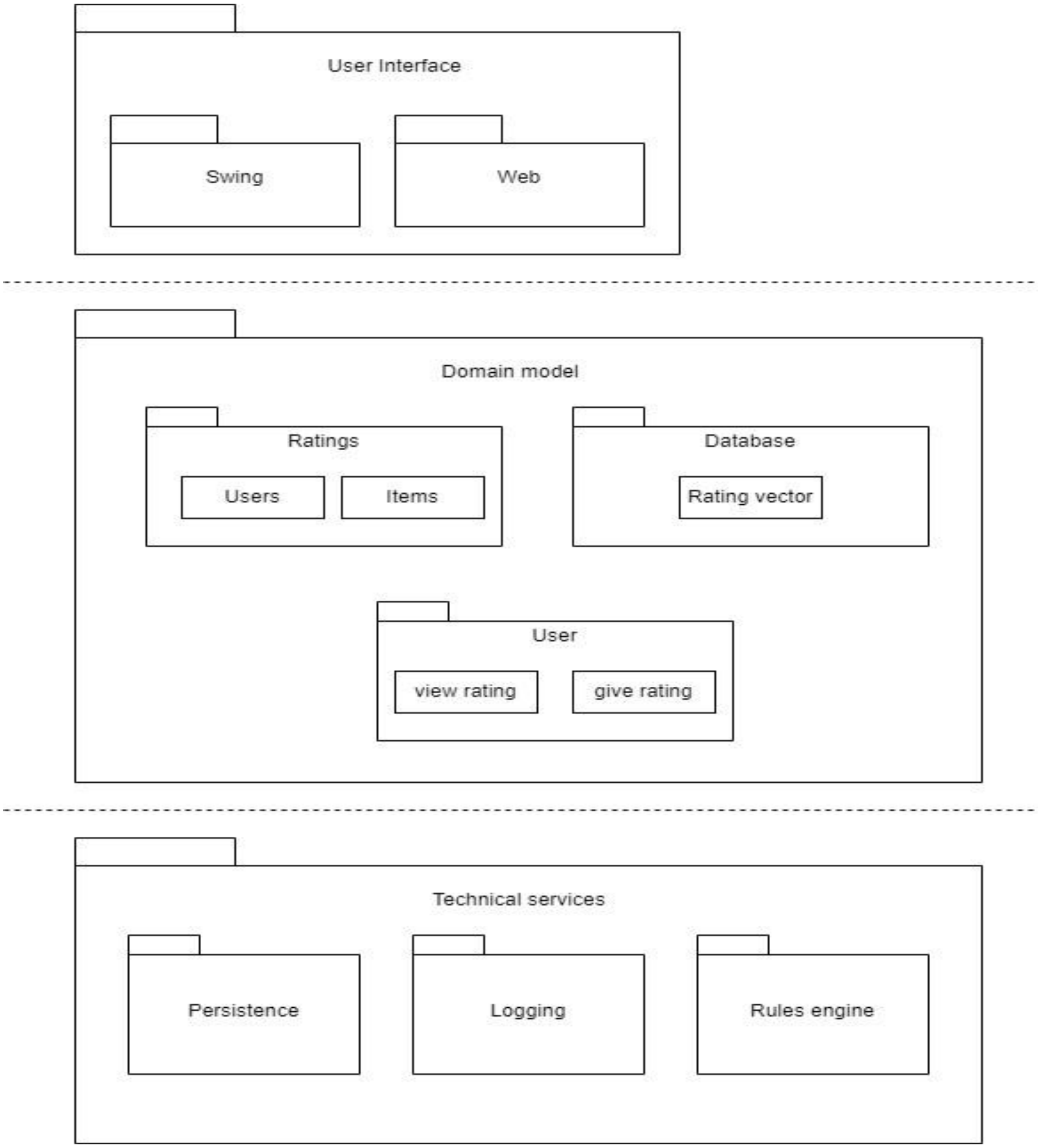


Fig 5.4.9 Package Diagram

5.5 DATA FLOW DIAGRAM

A data-flow diagram is a way of representing a flow of data through a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops, Specific operations based on the data can be represented by a flowchart.

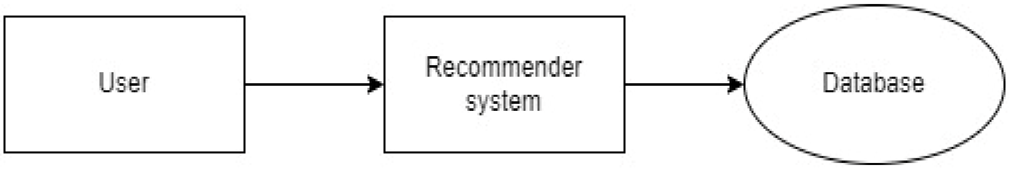


Fig 5.5.1 DFD Level 0

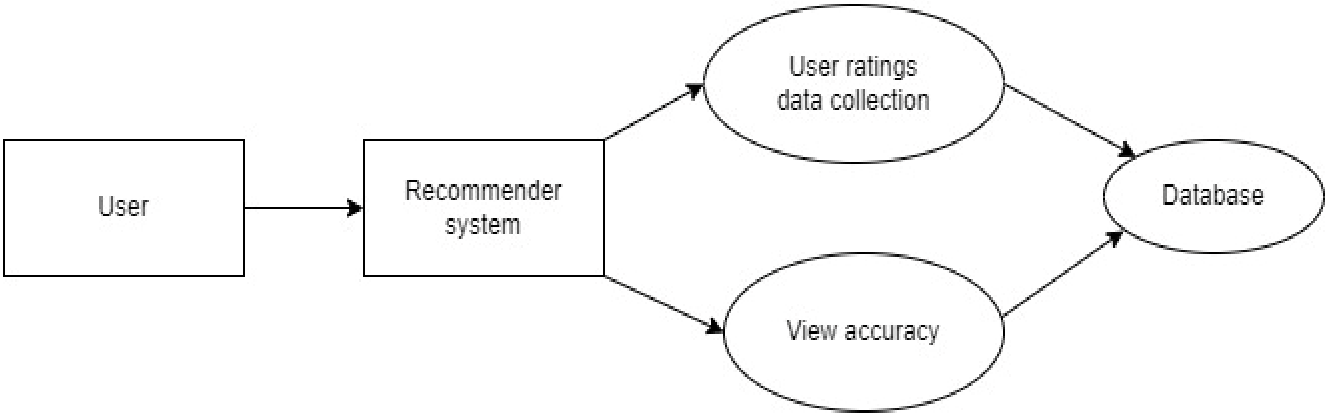


Fig 5.5.2 DFD Level 1

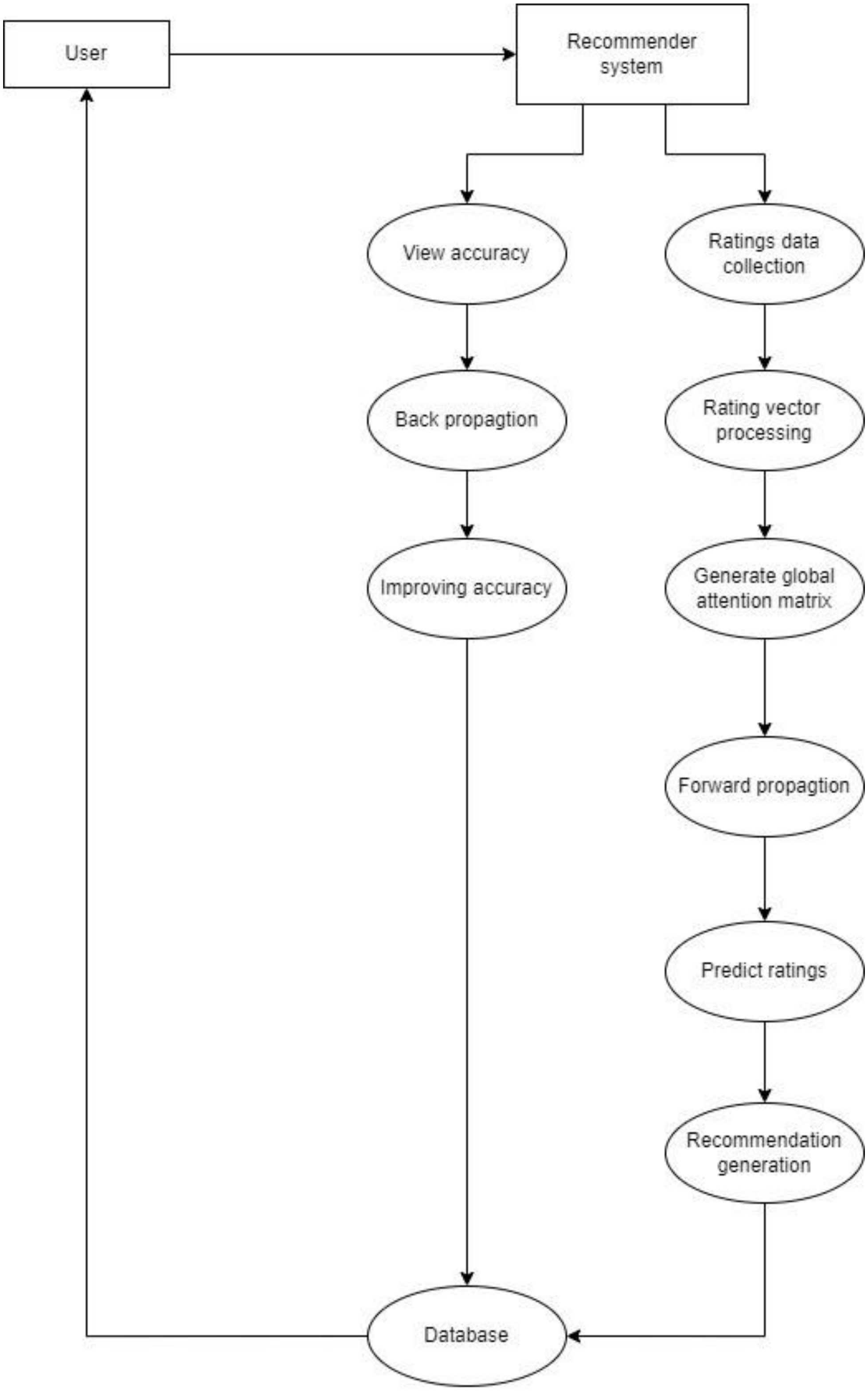
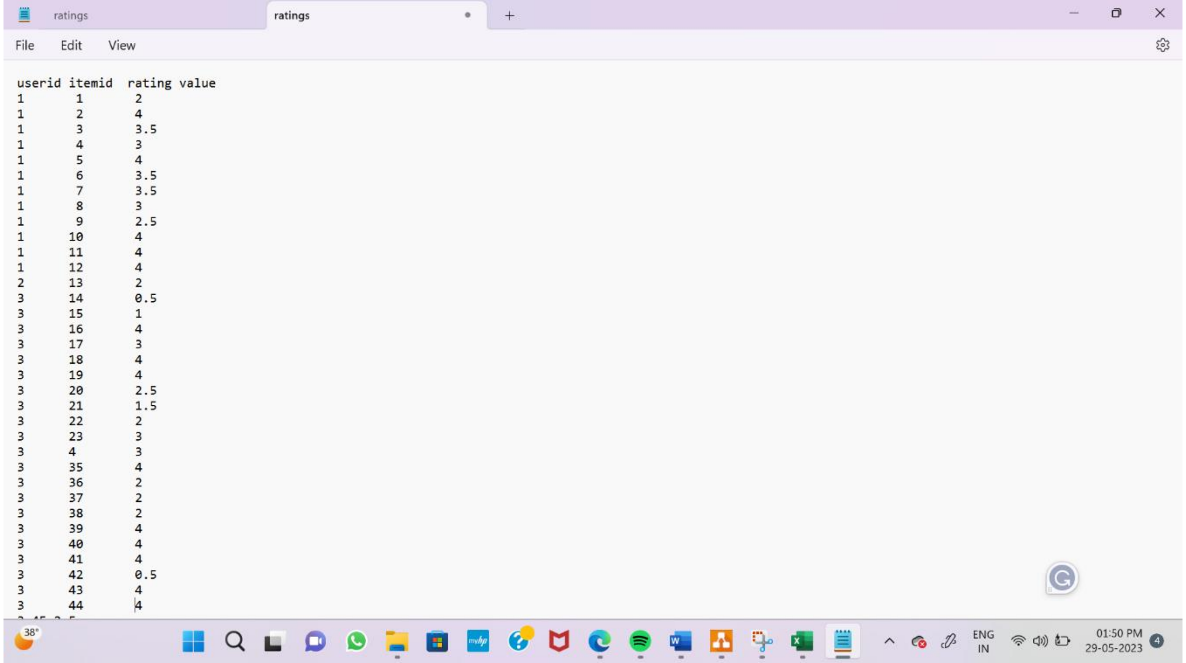


Fig 5.5.3 DFD Level 2

CHAPTER 6

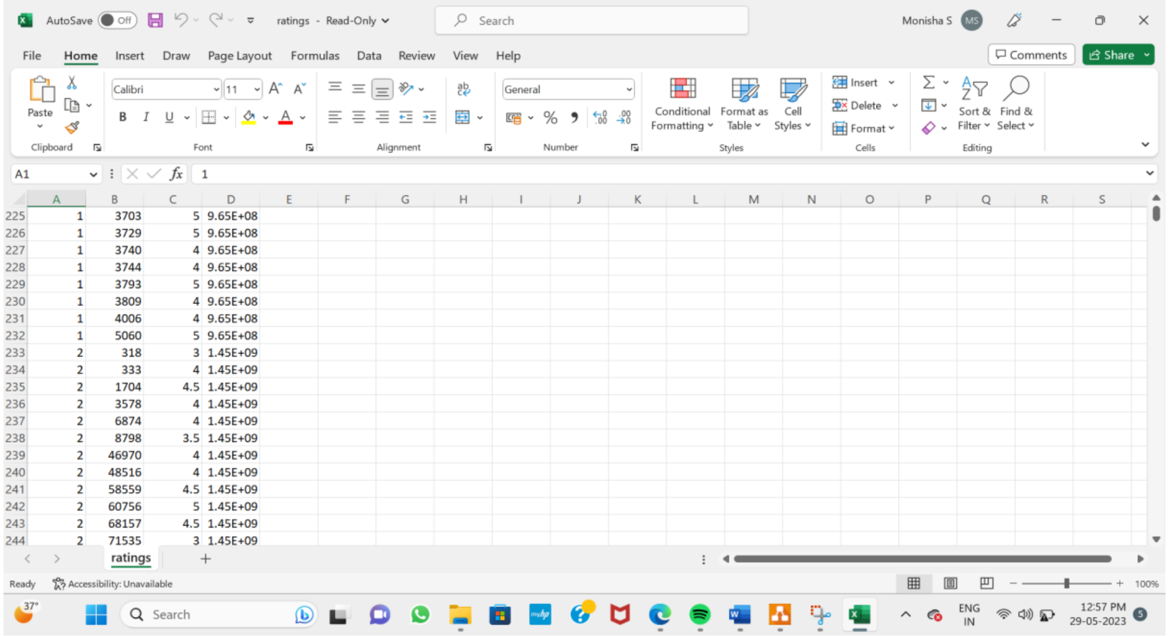
IMPLEMENTATION

6.1 BPAM++ - master Data set



```
userid itemid  rating value
1       1       2
1       2       4
1       3       3.5
1       4       3
1       5       4
1       6       3.5
1       7       3.5
1       8       3
1       9       2.5
1       10      4
1       11      4
1       12      4
2       13      2
3       14      0.5
3       15      1
3       16      4
3       17      3
3       18      4
3       19      4
3       20      2.5
3       21      1.5
3       22      2
3       23      3
3       4       3
3       35      4
3       36      2
3       37      2
3       38      2
3       39      4
3       40      4
3       41      4
3       42      0.5
3       43      4
3       44      4
```

Fig 6.1 BPAM++ - Filmtrust ratings



	A	B	C	D
225	1	3703	5	9.65E+08
226	1	3729	5	9.65E+08
227	1	3740	4	9.65E+08
228	1	3744	4	9.65E+08
229	1	3793	5	9.65E+08
230	1	3809	4	9.65E+08
231	1	4006	4	9.65E+08
232	1	5060	5	9.65E+08
233	2	318	3	1.45E+09
234	2	333	4	1.45E+09
235	2	1704	4.5	1.45E+09
236	2	3578	4	1.45E+09
237	2	6874	4	1.45E+09
238	2	8798	3.5	1.45E+09
239	2	46970	4	1.45E+09
240	2	48516	4	1.45E+09
241	2	58559	4.5	1.45E+09
242	2	60756	5	1.45E+09
243	2	68157	4.5	1.45E+09
244	2	71535	3	1.45E+09

Fig 6.2 BPAM++ - ml-la ratings

6.2 Language Used

Python

Python is a popular programming language among malware analysts due to its versatility and ease of use. Python's extensive library of modules and tools can streamline the process of analysing malware samples and identifying their behaviour. Python's automation capabilities also come in handy for automating tasks and processes within the malware analysis workflow, making the process more efficient and streamlined. One of the significant advantages of using Python for malware analysis is the availability of libraries and tools for tasks like disassembly or reverse engineering. These tools enable analysts to extract and analyse the underlying code of malware samples, helping them to understand how the malware works and what it does.

Additionally, Python's high-level syntax and dynamic typing allow analysts to write concise code that is easy to understand and maintain. Its cross-platform compatibility ensures that analysts can use Python on a wide range of operating systems and devices. Overall, Python's flexibility, ease of use, and extensive library of modules make it an ideal choice for analysing malware samples and understanding their behaviour.

CHAPTER 7

TESTING

7.1 TESTING OBJECTIVES

Testing is performed to identify errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test Each test type addresses a specific testing requirement. Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases: assemblies and/or a finished product. It is the process of exercising software.

- Static analysis is used to investigate the structural properties of the Source.
- Dynamic testing is used to investigate the behavior of the source code.

TYPES OF TESTING

7.1.1 Unit Testing

Unit testing involves the design of test cases that validate that the Internal program logic is functioning properly, and that program input produces valid output. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the 38 completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined input and expected results.

7.1.2 Integration Testing

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the setoff modules which make up the project. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then tested. This approach is evolved from unstructured testing of small programs. The new module and its interring communications. The product development can be staged, and modules integrated in as they complete unit testing. Testing is completed when the last module is integrated.

7.2 TESTING TECHNIQUES

Software testing is essential for correcting errors. Any engineering product can be tested in one of the following two ways:

7.2.1 White Box Testing

White box testing examines the internal structure, logic, and code of the software. This testing is also called as Glass box testing. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

- Flow graph notation
- Cyclometric complexity

7.2.2 Black Box Testing

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised.

It fundamentally focuses on the functional requirements of the software.

- Graph based testing methods
- Equivalence partitioning
- Boundary value analysis
- Comparison testing

7.2.2.1 Graph based testing methods

Graph-based testing methods utilize graphs or graph-like structures to model and analyze software systems for testing purposes. These techniques leverage the relationships and dependencies between components, functionalities, or scenarios to design and execute tests.

7.2.2.2 Boundary value Analysis

Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values in a range. The idea comes from the boundary. Given that we have a set of test vectors to test the system, a topology can be defined on that set.

7.2.2.3 Equivalence partitioning

Equivalence partitioning or equivalence class partitioning (ECP) is software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once.

7.2.2.4 Comparison Testing

Comparison testing involves comparing the contents of databases files, folders, etc. A comparison testing can be carried out in two ways Direct comparison testing: It is a testing of particular parts of each site against teach other. Usually, multiple sites are compared side by side.

CHAPTER 8

RESULTS AND DISCUSSION

8.1 PERFOMANCE ANALYSIS

Performance analysis for recommender systems involves evaluating the effectiveness and efficiency of the system in providing accurate and relevant recommendations to users. To perform a comprehensive performance analysis, you can use a combination of offline evaluation, where historical data is used to evaluate the system, and online evaluation, where real-time user interactions are monitored and analyzed. It is also recommended to use a variety of evaluation metrics to capture different aspects of system performance.

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}\left(r_{u,i} - \hat{r}_{u,i}\right)^2} \longrightarrow (1)$$

$$MAE = \frac{1}{N}\sum_{u,i}\left|r_{u,i} - \hat{r}_{u,i}\right| \longrightarrow (2)$$

where $\hat{r}_{u,i}$ denotes the predicted rating value, $r_{u,i}$ is the actual rating, and N denotes the number of tested ratings. Smaller values of RMSE and MAE indicate the better performance.

Comparison with different methods

While comparing to other recommendation models (i.e NeuMF, DMF, DeepCF) BPAM++ reduces the computational , storage costs and overfitting issues.

8.1.1 Comparison on Computational costs

We can observe that the computational costs of BPAM++ model is significantly lower than the computational costs of these DNNs-based models namely NeuMF, DMF and DeepCF.

Table.no.8.1.1 Comparison on computational cost

Models	**Costs**
BPAM++	204
NeuMF	2,736
DMF	98,432
DeepCF	270,464

8.1.2 Comparison on Storage cost

We can observe that the storage costs of BPAM++ model is significantly lower than the storage costs of these DNNs-based models namely NeuMF, DMF and DeepCF.

Table.no.8.1.2 Comparison on storage cost

Datasets	**BPAM++**	**NeuMF**	**DMF**	**DeepCF**
ml-la	60,390	829,312	5,701,632	8,519,168
filmtrust	122,148	288,912	2,702,848	3,791,104
jd-1	1,573,929	2,009,232	25,732,096	32,325,376
jd-2	1,480,500	1,890,592	24,213,504	30,427,136
jd-3	1,571,094	2,005,632	25,686,016	32,267,776
Automotive	163,968	383,632	4,036,096	5,427,456
Beauty	939,204	2,759,632	29,192,704	38,187,264
Kindle Store	2,387,805	10,415,152	101,668,864	135,161,088

8.1.3 Comparison on addressing overfitting issues

We can observe that the overfitting issues of BPAM++ model is significantly lower than the overfitting issues of these DNNs-based models namely NeuMF, DMF and DeepCF.

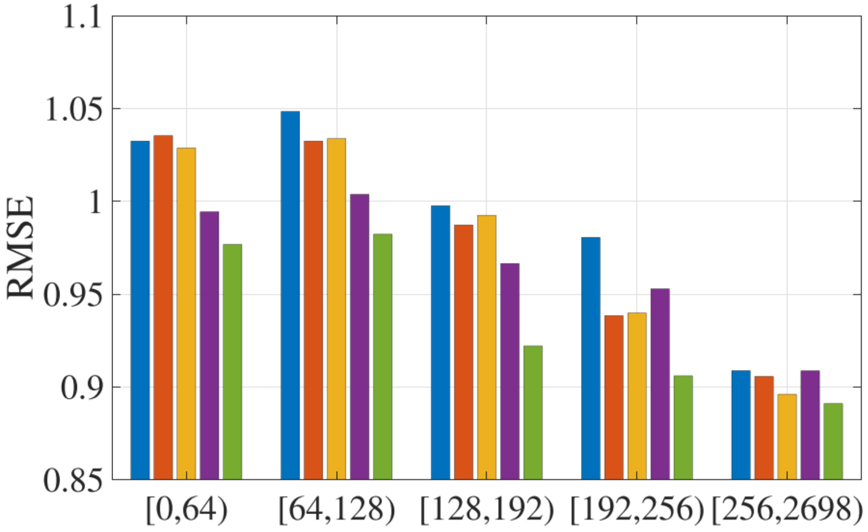


Fig 8.1.3.1 RMSE Comparison on overfitting issues

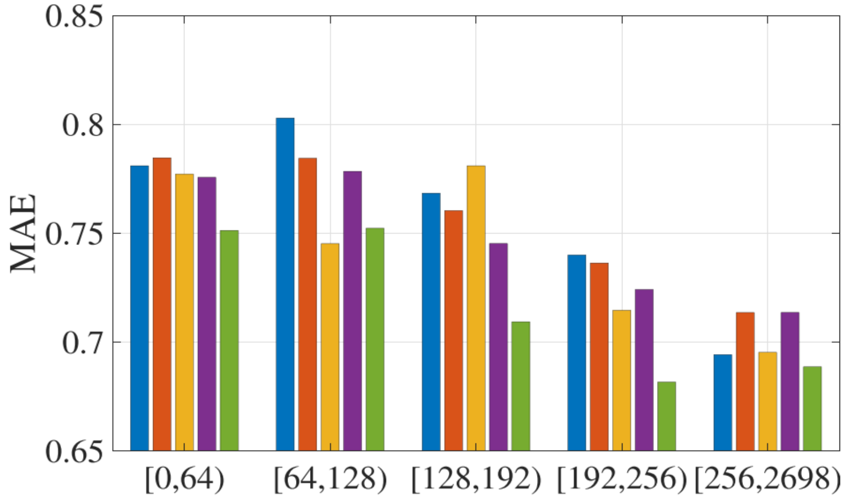


Fig 8.1.3.2 MAE Comparison on overfitting issues

🟦 NeuMF 🟥 DMF 🟨 DeepCF 🟪 BPAM 🟩 BPAM++

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENT

9.1 CONCLUSION

In this work, we have proposed a novel recommender framework based on BP neural network with attention mechanism, called BPAM++. In the proposed BPAM++ framework, the BP neural network is utilized to learn the complex relationship between the target users and their neighbors and the complex relationship between the given items and their neighbors. In this way, BPAM++ outperforms SVD++ which only utilizes the traditional matrix factorization and neighbor models.

Compared with DNNs, shallow BP neural network can not only reduce the computational and storage costs, but also prevent the model from overfitting caused by the small number of ratings. Besides, an attention mechanism is introduced in BPAM++ to capture the global attention weights about users' impact on their nearest target user set. We also compare BPAM++ with our previously proposed BPAM model, in which the rating information of the target user is not considered. In particular, BPAM can be regarded as a special case of BPAM++. Extensive experiments on eight benchmark datasets demonstrate that our proposed model distinctly outperforms state-of-the-art methods.

9.2 FUTURE WORK

In this work, we mainly focus on the rating matrix in the recommender systems. In the future, we aim to introduce content information of users and items or social relationship of users to calculate similarities between users and items instead of only utilizing the rating vectors of users and items, so that we can obtain their more accurate nearest neighbors. Richer information usually leads to better performance. Moreover, except for the simple concatenation, it is also encouraged to explore other aggregation methods to construct the input vector.

REFERENCES

1. C.-D. Wang, Z.-H. Deng, J.-H. Lai, and P. S. Yu, “Serendipitous recommendation in E-commerce using innovator-based collaborative filtering,” IEEE Trans. Cybern., vol. 49, no. 7, pp. 2678–2692, Jul. 2019.
2. J. Ma, J. Wen, M. Zhong, W. Chen, and X. Li, “MMM: Multi-source multi-net micro-video recommendation with clustered hidden item representation learning,” Data Sci. Eng., vol. 4, no. 3, pp. 240–253, 2019.
3. Q.-Y. Hu, L. Huang, C.-D. Wang, and H.-Y. Chao, “Item orientated recommendation by multi-view intact space learning with overlapping,” Knowl.Based Syst., vol. 164, pp. 358–370, 2019.
4. P. Hu, R. Du, Y. Hu, and N. Li, “Hybrid item-item recommendation via semiparametric embedding,” in Proc. Int. Joint Conf. Artif. Intell., pp. 2521–2527, 2019.
5. C.-C. Hsu, M.-Y. Yeh, and S.-D. Lin, “A general framework for implicit and explicit social recommendation,” IEEE Trans. Knowl. Data Eng., vol. 30, no. 12, pp.2228–2241, Dec. 2018.
6. C. Xu et al., “Recurrent convolutional neural network for sequential recommendation,” in Proc. Int. Conf. World Wide Web, pp. 3398–3404, 2019.
7. O. Barkan, N. Koenigstein, E. Yogev, and O. Katz, “CB2CF: A neural multiview content-to-collaborative filtering model for completely cold item recommendations,” in Proc. 13th ACM Conf. Recommender Syst., pp. 228–236, 2019.

8. L. Huang, C.-D. Wang, H.-Y. Chao, J.-H. Lai, and P. S. Yu, “A score prediction approach for optional course recommendation via cross-user-domain collaborative filtering,” IEEE Access, vol. 7, pp. 19 550–19 563, 2019.

9. W. Fu, Z. Peng, S. Wang, Y. Xu, and J. Li, “Deeply fusing reviews and contents for cold start users in cross-domain recommendation systems,” in Proc. AAAI Conf. Artif. Intell., pp. 94–101, 2019.

10. A. Ferraro, “Music cold-start and long-tail recommendation: Bias in deep representations,” in Proc. 13th ACM Conf. Recommender Syst., pp. 586–590, 2019.

11. H.-J. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, “Deep matrix factorization models for recommender systems,” in Proc. Int. Joint Conf. Artif. Intell.,pp. 3203–320 , 2017.

12. W. Yan, D. Wang, M. Cao, and J. Liu, “Deep auto encoder model with convolutional text networks for video recommendation,” IEEE Access, vol. 7, pp. 40 333–40 346, 2019

13. X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” in Proc. Int. Conf. World Wide Web, pp. 173–182, 2017.

14. W.-D. Xi, L. Huang, C.-D. Wang, Y.-Y. Zheng, and J. Lai, “BPAM:Recommendation based on BP neural network with attention mechanism,” in Proc. Int. Joint Conf. Artif. Intell., pp. 3905–3911, 2019.

15. R. M. Bell and Y. Koren, “Scalable collaborative filtering with jointly derived neighborhood interpolation weights,” in Proc. 7th IEEE Int. Conf. Data Mining, pp. 43–52, 2007.

16. J. Rongfei, J. Maozhong, and L. Chao, “Using temporal information to improve predictive accuracy of collaborative filtering algorithms,” in in Proc. 12th Int.

AsiaPacific Web Conf., pp. 301–306, 2010.

17. B. K. Patra, R. Launonen, V. Ollikainen, and S. Nandi, “A new similarity measure using Bhattacharyya coefficient for collaborative filtering in sparse data,” Knowl.-Based Syst., vol. 82, pp. 163–177, 2015.

18. C. C. Krueger, “The impact of the Internet on business model evolution within the news and music sectors,” PhD Thesis, pp. 1–397, 2006.

19. Y. Koren, “Factorization meets the neighborhood: A multifaceted collaborative filtering model,” in Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, pp. 426–434, 2008.

20. Y. Gong and Q. Zhang, “Hashtag recommendation using attention-based convolutional neural network,” in Proc. Int. Joint Conf. Artif. Intell., pp. 2782–2788 , 2016.

21. H. Wang, N. Wang, and D.-Y. Yeung, “Collaborative deep learning for recommender system,” in Proc.21st ACM SIGKDD Int. Conf. knowl. Discov. Data Mining, pp.1235-1244, 2015.

22. A. T. Goh, “Back-propagation neural networks for modeling complex systems,” Artif. Intell. Eng., vol. 9, no. 3, pp. 143–151, 1995.

23. A. van den Oord, S. Dieleman, and B. Schrauwen, “Deep content based music recommendation,” in Proc. Int. Conf. Neural Inf. Process. Syst., pp. 2643–2651, 2013.

24. A. Mnih and R. R. Salakhutdinov, “Probabilistic matrix factorization,” in Proc. 20th Int. Conf. Neural Inf. Process. Syst., pp. 1257–1264, 2008.

25. P. Li, Z. Wang, Z. Ren, L. Bing, and W. Lam, “Neural rating regression with abstractive tips generation for recommendation,” in Proc. 40th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval, pp. 345–354, 2017.

APPENDIX

read_Data.py

```
import numpy as np
import pandas as pd
import scipy.sparse as sparse
from datetime import datetime
from pandas import Series
from sklearn.decomposition import PCA
from sklearn.neighbors import NearestNeighbors


def unique(old_list):
    # Minimize the use_id and item_id in the dataset
    count = 0   ## Update from count = 1
    dic = {}
    for i in range(len(old_list)):
        if old_list[i] in dic:
            old_list[i] = dic[old_list[i]]
        else:
            dic[old_list[i]] = count
            old_list[i] = count
            count += 1
    return old_list

def readTrainData(filename ='data/filmtrust/ratings.txt'):


    data = pd.read_table(filename, header=None, sep=' ')
```

```
#
# # read ml-1m
print(data)
row = list(data[0])
column = list(data[1])
value = list(data[2])

row = unique(row)
column = unique(column)

numberOfUser = max(row) + 1
numberOfItem = max(column) + 1

#
print(len(row))
print(len(column))
print(len(value))
# # print(value)
mtx = sparse.coo_matrix((value, (row, column)),
shape=(numberOfUser, numberOfItem))
mtx = mtx.todense()
print(mtx)
mtx = np.array(mtx)
mdx = np.zeros([mtx.shape[0], mtx.shape[1]])

mean_user = []
```

```
    for i in range(mtx.shape[0]):
        temp = mtx[i]
        if len(temp[temp != 0]) == 0:
            mean_user.append(0)
        else:
            mean_user.append(np.mean(temp[temp != 0]))
    for i in range(mtx.shape[0]):
       for j in range(mtx.shape[1]):
           if mtx[i][j] != 0:
               mdx[i][j] = mtx[i][j]
           else:
               mdx[i][j] = mean_user[i]

    return mdx, numberOfUser, numberOfItem, mtx
```

GetKNearestNeighbor.py

```
import numpy as np
import pandas as pd
import scipy.sparse as sparse
from datetime import datetime
from pandas import DataFrame
from pandas import Series
from sklearn.decomposition import PCA
from sklearn.neighbors import NearestNeighbors
import  warnings
```

```
def get_neighbors(mdx, k, numberOfUser):

    warnings.filterwarnings("ignore")

    # Using PCA for dimensionality reduction
    start = datetime.now()
    pca = PCA(n_components=200, copy=True)
    pca_mtx = pca.fit_transform(mdx)
    end = datetime.now()
    print(start, end)

    # Build the KNN model
    start = datetime.now()
    neigh = NearestNeighbors(n_neighbors=k + 1,
    algorithm='auto', metric="cosine", n_jobs=1)
    neigh.fit(pca_mtx)
    end = datetime.now()
    print(start, end)

    # Obtain the k nearst neighbors for each user

    start = datetime.now()
    distance, neighbor = neigh.kneighbors(pca_mtx)
    print(start, end)

    ulist = list()
```

```
dlist = list()
nlist = list()

for user in range(numberOfUser):
    for i in range(1, k + 1):
        ulist.append(user)
        n_uesrs = neighbor[user] # 提取 user 对应的 neighbor user
        d_users = distance[user] # 提取 user 对应的 neighbor user 的 distance
        user_index = n_uesrs.index(user) # 获得 user 自己在列表对应的 index
        del n_uesrs[user_index]
        del d_users[user_index]
        nlist += n_uesrs
        dlist += d_users

df_neighbor = DataFrame(columns=['user', 'neighbor', 'distance'])
df_neighbor['user'] = ulist
df_neighbor['neighbor'] = nlist
df_neighbor['distance'] = dlist

df_neighbor['similarity'] = 0.5 + 0.5 * df_neighbor['distance']
df_neighbor.to_csv('user_neighbors.csv', index=False)
new_neighbor = []
for i in range(len(nlist)):
    new_neighbor.append(nlist[i])
return new_neighbor
```

```
    def k_neighbors(mdx, k_user,numberOfUser):
        neighbor_user = get_neighbors(mdx, k_user, numberOfUser)

        return neighbor_user
```

BPnetworkAttention.py

```
# #! /usr/bin/env python
# # coding=utf-8
import math
import random
import numpy as np
random.seed(0)

def rand(a, b):
    # Generate random values between a and b
    return (b - a) * random.random() + a

def make_matrix(m, n, fill=0.0):
    # Generate a matrix with m*n dimensions
    mat = []
    for i in range(m):
        mat.append([fill] * n)
    return mat

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))
    # Sigmoid activation function
```

```
def sigmoid_derivative(x):
    return x * (1 - x)
    # Derivative of sigmoid

class BPNeuralNetwork:
    def _init_(self):
        self.input_n = 0
        self.hidden_n = 0
        self.output_n = 0
        self.input_cells = []
        self.hidden_cells = []
        self.output_cells = []
        self.input_weights = []
        self.output_weights = []
        self.input_correction = []
        self.output_correction = []

    def setup(self, ni, nh, no):
        self.input_n = ni + 1
        self.hidden_n = nh
        self.output_n = no

        # init cells
        self.input_cells = [1.0] * self.input_n
        self.hidden_cells = [1.0] * self.hidden_n
        self.output_cells = [1.0] * self.output_n
        # init weights
        self.input_weights = make_matrix(self.input_n, self.hidden_n)
        self.output_weights = make_matrix(self.hidden_n, self.output_n)

        # # random activate
        for i in range(self.input_n):
            for h in range(self.hidden_n):
                self.input_weights[i][h] = rand(-0.2, 0.2)

        for h in range(self.hidden_n):
            for o in range(self.output_n):
                self.output_weights[h][o] = rand(-2.0, 2.0)

        # init correction matrix
        self.input_correction = np.zeros([self.input_n, self.hidden_n])
        self.output_correction = np.zeros([self.hidden_n, self.output_n])

    def predict(self, inputs,attentionWeightKNN, attentionRate):
        # activate input layer
        for i in range(self.input_n - 1):
            self.input_cells[i] = inputs[i]

        # activate hidden layer
        for j in range(self.hidden_n):
            total = 0.0
            for i in range(self.input_n - 1):
                # print(attentionWeightKNN[i][j])
                total += self.input_cells[i] * (attentionRate * attentionWeightKNN[i][j] +
self.input_weights[i][j])
            total += self.input_cells[self.input_n - 1]

            self.hidden_cells[j] = sigmoid(total)

        # activate output layer
        for k in range(self.output_n):
            total = 0.0
            for j in range(self.hidden_n):
                total += self.hidden_cells[j] * self.output_weights[j][k]
            self.output_cells[k] = sigmoid(total)
        return self.output_cells[:]

    def back_propagate(self, case, label, learn, correct,attentionWeightKNN,
attentionRate,att_correct_KNN):
        # feed forward
        self.predict(case, attentionWeightKNN,attentionRate)
        # get output layer error
        output_deltas = [0.0] * self.output_n
        for o in range(self.output_n):
            error = label[o] - self.output_cells[o]
            output_deltas[o] = sigmoid_derivative(self.output_cells[o]) * error
        # get hidden layer error
        hidden_deltas = [0.0] * self.hidden_n
        for h in range(self.hidden_n):
            error = 0.0
    for o in range(self.output_n):
        error += output_deltas[o] * self.output_weights[h][o]
    hidden_deltas[h] = sigmoid_derivative(self.hidden_cells[h] * error)
# update output weights
for h in range(self.hidden_n):
    for o in range(self.output_n):
        change = output_deltas[o] * self.hidden_cells[h]
        self.output_weights[h][o]+=learn*change+correct * self.output_correction[h][o]
        self.output_correction[h][o] = change

# update input weights
for i in range(self.input_n):
    for h in range(self.hidden_n):
        change = hidden_deltas[h] * self.input_cells[i]
        self.input_weights[i][h] += learn * change + correct * self.input_correction[i][h]
        self.input_correction[i][h] = change

# update attention mechanism weights
for i in range(self.input_n - 1):
    for h in range(self.hidden_n):
        change = hidden_deltas[h] * self.input_cells[i] * attentionRate
        attentionWeightKNN[i][h] += learn * change + correct * att_correct_KNN[i][h]
        att_correct_KNN[i][h] = change

# get global error
error = 0.0
for o in range(len(label)):
```

```
            # print(label[o], self.output_cells[o])
            error += 0.5 * (label[o] - self.output_cells[o]) ** 2
            error /= len(label)
        return error, attentionWeightKNN, att_correct_KNN

    def RMSE(self, preds, truth):
        return np.sqrt(np.mean(np.square(preds - truth)))

    def train(self, cases, labels,att_weights_knn, att_correct_knn,
    attention_rate = 0.5, limit=10000, learn=0.2, correct=0.1):
        for j in range(limit):
            for i in range(len(cases)):
                label = labels[i]
                case = cases[i]
                error, att_weights_knn, att_correct_knn= self.back_propagate
    (case, label, learn, correct, att_weights_knn, attention_rate, att_correct_knn)

        return att_weights_knn, att_correct_knn
```

evaluationIndicator.py

```
import numpy as np
# Calculate the value of RMSE
def RMSE(predict_list, y_test,user):
    diffMat = np.array(predict_list) - np.array(y_test)
    sqdiffMat = diffMat ** 2
    length = len(sqdiffMat)
    sqdifference = sqdiffMat.sum()
    singe_user_RMSE = (sqdifference / len(sqdiffMat) ) ** 0.5

    return sqdifference, length, singe_user_RMSE * 5

# Calculate the value of MAE
def MAE(predict_list, y_test, user):
    diffMat = np.array(predict_list) - np.array(y_test)
    sqdiffMat = abs(diffMat)
    length = len(sqdiffMat)
    sqdifference = sqdiffMat.sum()
    singe_user_MAE = sqdifference / len(sqdiffMat)

    return sqdifference, length,singe_user_MAE * 5
```

RatingPredict.py

```
import numpy as np
import math
import  evaluatingIndicator
from sklearn.preprocessing import MinMaxScaler
import  BPnetworkAttention
import  random
from pandas import DataFrame
from datetime import datetime
import pandas as pd

def rand(a, b):
    # Generate random values between a and b
    return (b - a) * random.random() + a

def make_matrix(m, n, fill=0.0):
    # Generate a matrix with m*n dimensions
    mat = []
    for i in range(m):
        mat.append([fill] * n)
    return mat

def general_attention(number, hidden_n):
    # Generate the weight of atttntion
    # number：the number of users' neighbors
    # hidden_n: the number of neurons in the hidden layers
    att_weights = make_matrix(number, hidden_n)
    att_correct = make_matrix(number, hidden_n)

    return att_weights, att_correct

def data_spilit(x, y, pct):
    # x：input data
    # y：the label
    # pct：the percent of training data
    length = len(y)
    index = round(length * pct)
    indexes = np.array(range(0, length))
    random.shuffle(indexes)
    trn_idxes = indexes[0:index]
    tst_idxes = indexes[index:length]
    # print(trn_idxes)
    # print(x[trn_idxes,:])
    x_train = x[trn_idxes, :]
    x_test = x[tst_idxes, :]
    y_train = y[trn_idxes, :]
    y_test = y[tst_idxes, :]
    return x_train, x_test, y_train, y_test, indexes

def number_of_zero(data):
    count = 0
    for e in data:
        if e == 0:
            count += 1
    return count

def predict_item_and_user(number_of_x, number_of_y, neighbor, mtx_np, k, att_weights,
att_correct, input_n, hidden_n, mtx_ds,ar):
    rmse_sum_difference = 0
    mae_sum_difference = 0
    count_sum = 0
    RMSE_list = list()
    MAE_list = list()

    start = datetime.now()
    for id in range(1, number_of_x):
```

```
id_of_neighbors = list(neighbor[id][:])

# Import rating information of all neighbors of users into X_np
X = []
att_weights_knn = []
att_correct_knn = []
#print ("id %d:" % id, id_of_neighbors)
for neighbor_id in id_of_neighbors:
    X.append(mtx_ds[neighbor_id])
    att_weights_knn.append(att_weights[neighbor_id])
    att_correct_knn.append(att_correct[neighbor_id])
X_np = np.array(X, dtype=float)
X_np = np.reshape(X_np, (k, number_of_y))

# Store user rating information
y = mtx_np[id]
y = np.reshape(y, (1, number_of_y))

# Transpose, each line denotes the rating information of all neighbors
X_np = X_np.T
y = y.T

# Pick out items that have been evaluated by user u
y_new = []
x_new = []
origine_index = []  # Record the original index
count = 0
```

```
        # print(y)
        for keys in range(number_of_y):
            if y[keys] >  0:
                x_new.append(X_np[keys])
                y_new.extend(y[keys])
                origine_index.append(keys)
                count += 1
        # Convert list to array form for easy training
        y_new = np.reshape(y_new, (count, 1))
        x_new = np.reshape(x_new, (count, k))

        # Split data into training and test sets
        X_train, X_test, y_train, y_test, indexs = data_spilit(x_new, y_new, 0.75)

        # Creating a neural network
        nm = BPnetworkAttention.BPNeuralNetwork()
        nm.setup(input_n, hidden_n, 1)
        attention_rate = ar  # attention ratio
        att_weights_knn, att_correct_knn = nm.train(X_train, y_train, att_weights_knn,
att_correct_knn,
       attention_rate, 100, 0.0001, 0.1)  # Importing data into BP neural network for training

        #Update att_weight and att_correct
        i = 0
        for neighbor_id in id_of_neighbors:
            att_weights[neighbor_id] = att_weights_knn[i]
            att_correct[neighbor_id] = att_correct_knn[i]
            i += 1

        # Predict the test set and get the test results
        #print(att_weights_knn)
        predict_list = []  # for storing the predicted ratings
        for i in range(len(X_test)):
            predict = nm.predict(X_test[i], att_weights_knn, attention_rate)
            predict_list.append(predict)

        predict_list = np.array(predict_list)

        RMSEsqdifference, length ,RMSE = evaluatingIndicator.RMSE(predict_list, y_test, id)  # return RMSE
        rmse_sum_difference += RMSEsqdifference
        count_sum += length

        MAEsqdifference, length, MAE = evaluatingIndicator.MAE(predict_list, y_test, id)  # return MAE
        mae_sum_difference += MAEsqdifference
        RMSE_list.append(RMSE)
        MAE_list.append(MAE)

    end = datetime.now()

    RMSE = (rmse_sum_difference / count_sum) ** 0.5
    print('the RMSE about the prediction of all users: %f' % (RMSE * 5))
    MAE = mae_sum_difference / count_sum
    print('the MAE about the prediction of all  users: %f' % (MAE * 5))
    RMSE_list.append(RMSE)
    MAE_list.append(MAE)

    return  RMSE, att_weights, att_correct, RMSE_list, MAE_list

def rating_predict(numberOfUser, numberOfItem, neighbor_user,     mtx_np, k_user, mtx_ds,ar):
    '''
    :param numberOfUser: the number of users
    :param numberOfItem: the number of items
    :param neighbor_user: the KNN neighbor matrix of users
    :param mtx_np: rating matrix
    :param k: the number of users' neighbors
    :param mtx_ds: Rating matrix after data preprocessing
    :ar: Attention rate
    :return:
    '''
    mm_mtx_np = np.array(mtx_np) / 5
    mm_mtx_np_T = mm_mtx_np.T
    mtx_ds = mtx_ds / 5
    input_n = k_user  # the number of neurons in input layer
    hidden_n = int(k_user / 2)  # the number of neurons in hidden layer
    att_weights_user, att_correct_user = general_attention(numberOfUser, hidden_n)

    print(mm_mtx_np)
    print(mtx_ds)
    RMSE_user, att_weights_user, att_correct_user, RMSE_list, MAE_list =
predict_item_and_user(
        numberOfUser, numberOfItem, neighbor_user, mm_mtx_np, k_user,
        att_weights_user, att_correct_user, input_n, hidden_n, mtx_ds,ar)
```

__init__.py

```
import  GetKNearestNeighbor
import read_Data
import RatingPredict

if _name_ == "_main_":
    k_user = 10
    ar = 2
    filename = 'data/filmtrust/ratings.txt'
    mtx_ds, numberOfUser, numberOfItem, mtx_np = read_Data.readTrainData(filename)
    neighbor_user= GetKNearestNeighbor.k_neighbors(mtx_np, k_user, numberOfUser)
    RatingPredict.rating_predict(numberOfUser, numberOfItem, neighbor_user, mtx_np,
k_user, mtx_ds,ar)
```