

VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING



**AN E-COMMERCE WEBSITE WITH AN INTEGRATED
RECOMMENDATION SYSTEM**

By
Pham Minh Quang

A thesis submitted to the School of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Computer Science

Ho Chi Minh City, Vietnam
2020

**AN E-COMMERCE WEBSITE WITH AN INTEGRATED
RECOMMENDATION SYSTEM**

APPROVED BY:

Nguyen Thi Thanh Sang, Ph.D.

(Typed Committee name here)

(Typed Committee name here)

(Typed Committee name here)

(Typed Committee name here)

THESIS COMMITTEE

ACKNOWLEDGEMENTS

I'd like to express my thanks of gratitude and appreciation to my advisor, Dr. Nguyen Thi Thanh Sang, for being my guiding light through each stage of the process. Her sincerity, encouragement, and vision inspired me to achieve my goal. It was such an honor to work and research under her guidance.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	3
TABLE OF CONTENTS	4
LIST OF FIGURES	7
LIST OF TABLES.....	8
LIST OF EQUATIONS.....	9
LIST OF ABBREVIATIONS	10
ABSTRACT	11
CHAPTER 1	12
INTRODUCTION.....	12
1.1. Background	12
1.2. Problem Statement	12
1.3. Scope and Objectives	12
1.4. Assumption and Solution	13
1.5. Structure of Thesis	13
CHAPTER 2.....	14
LITERATURE REVIEW	14
2.1. E-commerce Websites	14
2.2. Recommender Systems	14
2.2.1. Neighborhood Models [7]	15
2.2.1.1. Item-based K-Nearest Neighbors	15
2.2.1.2. User-based K-Nearest Neighbors	15
2.2.2. Matrix-Factorization Model	16
2.2.2.1. Singular Value Decomposition (SVD)	16
2.2.2.2. Incremental Singular Value Decomposition	18
2.2.3. Long-tail Phenomenon	19

2.2.3.1. Explicit Query Aspect Diversification	20
CHAPTER 3	22
IMPLEMENTATION	22
3.1. Libraries and Frameworks	22
3.1.1. ReactJS	22
3.1.2. NodeJS	22
3.1.3. MongoDB.....	22
3.1.4. Flask	23
3.1.5. Surprise-scikit.....	23
3.2. System Design	23
3.2.1. Web Application	24
3.2.1.1. Back-end	24
3.2.1.2. Front-end	25
3.2.1.3. Database.....	26
3.2.1.4. User Interface Design	27
3.2.2. Recommender System.....	34
3.2.2.1. Product Recommendation Engine Construction.....	36
3.2.2.2. Related Products Construction	38
CHAPTER 4	39
EXPERIMENTAL RESULTS AND EVALUATION.....	39
4.1.1. Experimental Platform	39
4.1.2. Evaluation Metrics and Results	39
4.1.2.1. Prediction Accuracy	39
4.1.2.2. Recommendation Ranking Accuracy	42
4.1.2.2.1. Precision@M, Recall@M, F1@M	43
4.1.2.2.2. Average Recommendation Popularity (ARP) [22]	44
4.1.2.2.3. Average Percentage of Long-tail Items [22].....	44

4.1.2.2.4.	Coverage of Long-tail Items [22]	45
4.1.2.2.5.	Normalized Discounted Cumulative Gain (NDCG)	45
4.1.3.	Average Top-M Recommendation Hit Ratio [28]	47
CHAPTER 5	49
CONCLUSION AND FUTURE WORK	49
5.1.	Conclusion	49
5.2.	Future Work	50
PUBLICATIONS	51
REFERENCES	52
APPENDIX	55

LIST OF FIGURES

- Figure 1. Schematic Diagram of the SVD Folding-in Technique [21].
- Figure 2. Pseudo-code of Incremental SGD algorithm.
- Figure 3. The Long-tail Phenomenon in Recommender System [22].
- Figure 4. System Design Overview.
- Figure 5. The sample structure of a web page using ReactJS.
- Figure 6. Database Schema.
- Figure 7. Home page user interface.
- Figure 8. Bundle Purchase block.
- Figure 9. Product Detail page user interface.
- Figure 10. Search page user interface.
- Figure 11. Category page user interface.
- Figure 12. Sample order confirmation email.
- Figure 13. Check-out page user interface.
- Figure 14. Document Management page for admins.
- Figure 15. Document Export/Import page for admins.
- Figure 16. Dataset Management page for admins.
- Figure 17. Model Training page for admins.
- Figure 18. User Recommendation Generator page for admins.
- Figure 19. Related Products Generator page for admins.
- Figure 20. The off-line stage of training and tuning the recommendation model.
- Figure 21. The online stage of producing real-time recommendations.
- Figure 22. Flowchart for recommendation engine.
- Figure 23. The personalized recommendation list displayed in the home page.
- Figure 24. The related products of a product displayed in the product detail page.
- Figure 25. Accuracy loss over different incremental set ratios (MAE).
- Figure 26. Accuracy loss over different incremental set ratio (RMSE).
- Figure 27. Training time over different incremental set ratio.
- Figure 28. The Precision-Recall trade-off of different algorithms.
- Figure 29. Long-tail diversity metrics.
- Figure 30. NDCG.
- Figure 31. The results of top-M Hit Ratio with and without popularity bias control.

LIST OF TABLES

Table 1. A preview of Amazon Electronics dataset.

Table 2. Prediction accuracy comparison between different algorithms.

Table 3. Prediction accuracy comparison between the highest accuracy loss Incremental SVD model and other algorithms.

Table 4. Training time comparison between the Incremental SVD model with the largest incremental set and other algorithms.

Table 5. Precision@M, Recall@M, F1@M comparison between different algorithms.

LIST OF EQUATIONS

- Equation 1. Cosine Similarity between two products.
- Equation 2. Item-based K-Nearest Neighbors rating prediction.
- Equation 3. User-based K-Nearest Neighbors rating prediction.
- Equation 4. Matrix-Factorization Model rating prediction.
- Equation 5. Loss function of the SVD model.
- Equation 6. Prediction Error of SGD.
- Equation 7. Item factor updation of the SVD model.
- Equation 8. User factor updation of the SVD model.
- Equation 9. Biased rating prediction of the SVD model.
- Equation 10. Regularized loss function of the SVD model.
- Equation 11. Explicit Query Aspect Diversification.
- Equation 12. Ranking score using XQuAD.
- Equation 13. Mean Absolute Error.
- Equation 14. Root Mean Squared Error.
- Equation 15. F1@M.
- Equation 16. ARP.
- Equation 17. APLT.
- Equation 18. Coverage of Long-tail Items.
- Equation 19. Discounted Cumulative Gain (DCG).
- Equation 20. Top-M Hit Ratio [28].

LIST OF ABBREVIATIONS

REST: Presentational State Transfer
API: Application Programming Interface
SVD: Singular Value Decomposition
KNN: K-Nearest Neighbors
MERN: MongoDB, ExpressJS, ReactJS, NodeJS
SGD: Stochastic Gradient Descent
ALS: Alternating Least Squares
xQuAD: Explicit Query Aspect Diversification
HTTP: Hypertext Transfer Protocol
JSON: JavaScript Object Notation
JWT: JSON Web Token
ERD: Entity Relationship Diagram
ASIN: Amazon Standard Identification Number
CSV: Comma-separated Values
RMSE: Root Mean Squared Error
MAE: Mean Absolute Error
NDCG: Normalized Discounted Cumulative Gain
ARP: Average Recommendation Popularity
APLT: Average Percentage of Long-tail Items
ACLT: Average Coverage of Long-tail Items

ABSTRACT

Along with the rapid rise of the internet, e-commerce websites have brought enormous benefits for both customers and vendors in recent years. However, many choices are given at the same time makes customers have difficulty in choosing the most suitable products. A rising star solution for this is the recommender system which helps to narrow down the amount of suitable and relevant products for each customer.

In this project, a lightweight but robust E-commerce web application is built and integrated with a collaborative-filtering recommender system for personalizing a user's shopping experience. The web application is built using two trending technologies, ReactJS for the front-end and NodeJS for the back-end. The front-end functions as a bridge between server and user interaction with the application while collecting user interaction data, then sends them to the back-end. The back-end is built in the form of a RESTful API, it receives requests from the front-end, queries data from the database according to the request then sends the response with necessary data to the front-end. Besides, the recommender system is built using the Python web framework, Flask, is responsible for processing the data collected by the web application, building Machine Learning models, and generating recommendations for users. Amazon's Electronic data set is used to evaluate the proposed framework of the recommender system.

CHAPTER 1

INTRODUCTION

1.1. Background

Thanks to the lightning speed growing of web technology, it is getting easier and easier for one to build a lightweight web application, especially E-commerce website, while making sure it still meets their demands. The key to an E-commerce website success is to ensure a good shopping experience which many existing heavy and bulky E-commerce websites have failed to offer. Modern web technologies allow developers to create robust web applications while keeping their complexity at the lowest level. One of them is the MERN stack (stands for MongoDB, ExpressJS, ReactJS, and NodeJS) which minimizes the pain when building web applications while maximizes their power and effectiveness.

In addition to web technology, machine learning is also affordable. Many libraries allow vendors to use machine learning to enhance their business with ease. Having an integrated machine learning system gives business insights into their data, by that, personalized their customers' shopping experience.

1.2. Problem Statement

The bulky and redundant web application can cause negative effects for both customers and businesses. The shopping experience is extremely important for customers. One minor error in the shopping experience, such as the long wait time when adding products to the shopping cart, may result in major problems that avoid customers from coming back.

An effective recommender system will enable businesses to make good use of customers' data, discover crucial relationships and hidden patterns in them. Although the benefit of the recommender system, it is still challenging to build one. Choosing the best-fit algorithm to the existing E-commerce system is crucial when building the recommender system. In the ocean of recommendation algorithms, businesses need to analyze the existing system and data carefully before deciding to go with one.

1.3. Scope and Objectives

In this project, the MERN stack (mentioned before) is used to build an E-commerce web application that has most of the typical functions of an E-commerce website. Furthermore, by

integrating a collaborative filtering recommender system, the application makes use of users' data to personalize their shopping experience. The recommender system is constructed by experimenting with different machine learning algorithms then evaluating them on Amazon's Electronic Dataset. The most accurate algorithm is then integrated into the web application to make predictions and recommendations.

1.4. Assumption and Solution

The final application consists of two sub-systems: the web application and the recommender system server.

The web application is utilized for handling user interaction, ensuring the perfect user experiment, and collecting important data for the recommender system to learn from. The web application front-end is the visual of the final application. It keeps track of and responds to the user interaction with the application, ensures that users always get the data they ask for, and enhances the user shopping experience. The back-end is responsible for storing user data, getting data from the database every time the front-end needs.

The recommender system then uses the collected data to build a machine learning model. After the learning phase, it starts making predictions and recommendations for users. These works are done in separate phases. Because of the heavy, expensive computation of building machine learning models, the training phase is done off-line. The model is re-trained in schedule to make sure the recommendation is up to date. Meanwhile, the recommendations are generated in real-time using the fitted model which helps ensure good user shopping experience.

1.5. Structure of Thesis

This thesis includes six main chapters. The first chapter is a brief description of the project, its background information, scope, objectives, and solutions. The next chapter defines and analyzes key academic theories and concepts on E-commerce Web Application and Recommender System. The third chapter points out the overall approach and general solution of the research, its tools, and frameworks in the development of the application. The fourth chapter gives the in-depth implementation of the application, its experimental result, and evaluation. The last chapter is the conclusive answer to the main research question, and the research future's directions and plans.

CHAPTER 2

LITERATURE REVIEW

2.1. E-commerce Websites

The dramatic rise of online shopping demand around the world has led to the appearance of huge amounts of E-commerce websites. E-commerce represents electronic commerce [2]. Simply put, it is running a business online, services, and goods trading using the Internet. Businesses build their websites for customers to use their products and services with ease. The typical shopping flow of an E-commerce website is not so different from the traditional one. E-commerce websites allow users to purchase products with different methods (e.g, code on delivery, credit card, PayPal), then the goods are shipped to the customers' registered place. Most tasks are performed online. The most common E-commerce websites types are Business-to-Business (B2B), Business-to-Customer (B2C), Consumer-to-Consumer (C2C), and Consumer-to-Business (C2B). For this research, these concepts are not going to be discussed in detail.

E-commerce websites have different approaches to up-sale their business. In Vietnam, most of the E-commerce websites seem to be too greedy, they keep showing the bulk of advertisements and hope that one would fit users' interest. This approach usually results in a dazzling shopping experience, confusing users when visiting the website. To give personalized recommendations to users, modern E-commerce websites are trying to take advantage of the expansion of machine learning. Because good personalized recommendations can give an extra dimension to the user shopping experience, giant e-commerce websites like Amazon.com have made the recommender system an indispensable part of their website.

2.2. Recommender Systems

The recommendation system is a precise type of information filtering method to manifest information items (e.g, movies, music, web sites, books) that match the user's interest [3].

The recommendation can be classified into two ways [1]: content-based filtering predicts items the user might like based on the item description, and collaborative filtering uses the knowledge about preferences of users for some items to make recommendations. In this research, collaborative filtering methods are the primary focus.

Collaborative filtering techniques which are most used [4, 5, 6] look at user previous ratings or transactions to make recommendations. Moreover, it uses the interdependent relationships between users and products for generating recommendations. Two popular methods of collaborative-filtering are neighborhood-based models and latent-factor models. While the neighborhood-based models only focus on users or products at a time, the latent-factor model tries to predict the ratings based on both product and user characteristics, called “latent factors”. These latent factors are not always defined in the dataset but are inferred from the rating patterns. The rating of a user on a product could be computed by taking the dot product of the user factors and product factors.

2.2.1. Neighborhood Models [7]

The neighborhood model assumes users or items that are closed to each other would have similar ratings. The neighborhood models making predictions based on “K” nearest neighbors are called Item-based K-Nearest Neighbor (if neighbors are items) and User-based K-Nearest Neighbor (if neighbors are users) [8, 9, 10].

2.2.1.1. Item-based K-Nearest Neighbors

Item-based K-Nearest Neighbors (Item-based KNN) predicts the rating of a user u on item I by finding products similar to i . The similarity score between two products can be computed using Cosine Similarity as:

$$similarity(i, j) = \frac{r_i \cdot r_j}{\|r_i\| \|r_j\|}$$

Equation 1. Cosine Similarity between two products.

where, r_i and r_j are the rating vectors of the two items i and j , respectively, made by users.

Then, to predict rating on an item, the model will select K nearest neighbors for that item. Such prediction can be inferred from neighbors using the weighted mean:

$$\widehat{r_{ui}} = \frac{\sum_{j \in N_u^k(i)} similarity(i, j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} similarity(i, j)}$$

Equation 2. Item-based K-Nearest Neighbors rating prediction.

where $N_u^k(i)$ is the top K neighbors of item i that are rated by user u , r_{uj} is the rating of user u on item j , and $\widehat{r_{ui}}$ is the predicted rating of user u on item i .

2.2.1.2. User-based K-Nearest Neighbors

Similarly, the prediction can be made using a user-based model as:

$$\widehat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} \text{similarity}(u, v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} \text{similarity}(u, v)}$$

Equation 3. User-based K-Nearest Neighbors rating prediction.

where, $\text{similarity}(u, v)$ is the similarity between users u and v , and r_{vi} is the rating of user v on item i .

It is worth noting that in the real E-commerce system, the number of customers and items increases rapidly, there are many problems that need to be addressed, such as, cold-start problem [11], sparsity and scalability [12,13].

2.2.2. Matrix-Factorization Model

Matrix factorization approaches [14] are latent factor models that base on previous user-item interactions to characterize users and items at the same time.

For matrix factorization models, each item i could be represented by a vector $q_i \in \mathbb{R}^f$ and each user u could be represented by a vector $p_u \in \mathbb{R}^f$ where f is the dimension of the joint latent factor space [14]. The vector q_i values measure how “close” the item is to those factors. Similarly, the vector p_u measures the level of interest of the user on items close to those corresponding factors. The approximated rating of user u on item i could then be inferred as:

$$\widehat{r}_{ui} = q_i^T p_u$$

Equation 4. Matrix-Factorization Model rating prediction.

2.2.2.1. Singular Value Decomposition (SVD)

SVD is the most popular matrix factorization method that can be applied in recommender systems to solve sparsity and scalability problems [15,16, 17], used as dimensionality reduction technique and a powerful mechanism [18]. SVD [19] factorizes a matrix R into three matrices M, Σ, U , where q_i makes up the columns of U^T , p_u makes up the rows of M , and the diagonal matrix that acts as a scaler on M or U^T .

For recommendation, because of the sparsity of the rating matrix, such matrix R is impossible to construct. So instead of directly finding M and U , the vectors q_i , p_u such that Equation 4 is satisfied for all users u and items i will be computed. One way to do that is to minimize the squared error on the set of known ratings:

$$\min_{p, q} \sum_{(u, i \in \mathcal{K})} (r_{ui} - q_i^T p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

Equation 5. Loss function of the SVD model.

where λ is the regularization parameter, κ is the set of (u, i) pairs for which r_{ui} is from the training set. This optimization is not convex, in other words, it may be impossible to find the pair of p_u and q_i that minimize the sum. Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS) [20] would be used instead.

The SGD algorithm iterates over the ratings in the training set, for each case, it computes the error as:

$$e_{ui} = (r_{ui} - q_i^T \cdot p_u)$$

Equation 6. Prediction Error of SGD.

Then, vectors p_u and q_i can be updated simultaneously as:

$$q_i = q_i + \alpha(e_{ui} \cdot p_u - \lambda q_i)$$

Equation 7. Item factor updation of the SVD model.

$$p_u = p_u + \alpha(e_{ui} \cdot q_i - \lambda p_u)$$

Equation 8. User factor updation of the SVD model.

where α is the learning rate.

An alternative algorithm for SGD is Alternating Least Squares, which, in some cases, is more effective. ALS techniques tend to fix vectors q_i and treat them as constants, optimize p_u , then fix p_u and optimize q_i , and repeat until convergence [20]. One advantage of this approach is that the computation could be done in parallel because the system computes each p_u , q_i independently of each other. Moreover, for implicit data, ALS is more efficient than SGD.

The matrix factorization model can also be refined by adding biases for both users and items. The predicted rating then is computed as:

$$\widehat{r_{ui}} = \mu + b_u + b_i + q_i^T \cdot p_u$$

Equation 9. Biased rating prediction of the SVD model.

where b_u is the user bias and b_i is the item bias, and μ is the average rating. The observed rating is then computed by adding the biases and average rating, the system now tries to minimize the new loss function:

$$\min_{p^*, q^*, b^*} \sum_{(u, i \in \kappa)} (r_{ui} - \mu - b_u - b_i - q_i^T \cdot p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2)$$

Equation 10. Regularized loss function of the SVD model.

2.2.2.2. Incremental Singular Value Decomposition

The workflow of a recommender system is divided into two different stages: the offline stage or training stage and the online stage. In the online stage, the system gives real-time recommendations to users. A real-time recommendation generated by the SVD model is high-quality and reliable, but the training stage of SVD is computationally expensive and time-consuming. For a $m \times n$ user-item matrix, the time complexity of SVD is almost $\Theta(m)^3$ [21].

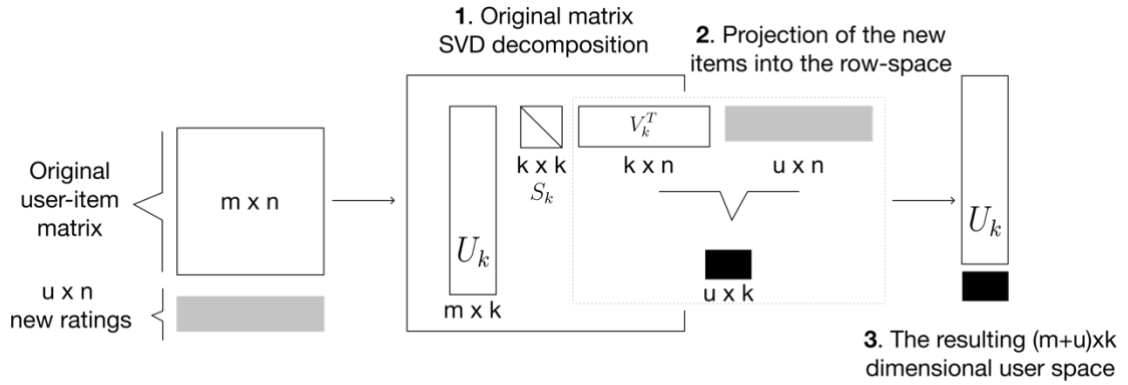


Figure 1. Schematic Diagram of the SVD Folding-in Technique [21].

The fold-in technique is used to project new users to the space of existed user-item matrix. It assumes that the projection of the additional user (or rating) can also provide a reliable and good approximation of the trained model (Figure 1). The Incremental SVD allows the model to incrementally learn the new ratings and makes more reliable and accurate recommendation for new users without the expensive computation.

In this project, inspired by the above technique, a different version of incremental SVD is implemented. Since the SVD approach mentioned in Section 2.2.2.1 is trying to find matrix M and U by minimizing a loss function using SGD or ALS, the incremental SVD approach will also base on that idea. This method can be called incremental SGD rather than incremental SVD since in this technique, instead of running SGD on the train set for n epochs (n is chosen when training the model), the incremental technique's SGD only runs on a mini batch of new ratings for only one epoch. In this approach, the loss function may not converge. However, it offers an acceptable improvement for the existed model while saving a great deal of time and memory compared to a full re-train. A pseudocode of this algorithm is shown in Figure 2 below.

```

Data:  $D = \{ \langle user, item \rangle \}$ , a finite set of new ratings
input:  $n_{factors}, \lambda, \alpha$ 
output:  $M, U$ 
for  $\langle user, item \rangle \in D$  do
  if  $user \notin Rows(M)$  then
     $M_u \leftarrow vector(size : n_{factors})$ 
     $M_u \sim \mathcal{N}(0, 0.1)$ 
  if  $item \notin Rows(U^T)$  then
     $U_i^T \leftarrow vector(size : n_{factors})$ 
     $U_i^T \sim \mathcal{N}(0, 0.1)$ 
   $err_{ui} \leftarrow r_{ui} - M_u \cdot U_i^T$ 
   $M_u \leftarrow M_u + \alpha(err_{ui}U_i - \lambda M_u)$ 
   $U_i \leftarrow U_i + \alpha(err_{ui}M_u - \lambda U_i)$ 

```

Figure 2. Pseudo-code of Incremental SGD algorithm.

2.2.3. Long-tail Phenomenon

There exists a phenomenon in recommender systems called long-tail phenomenon [22] which is the distance between physical and online stores.

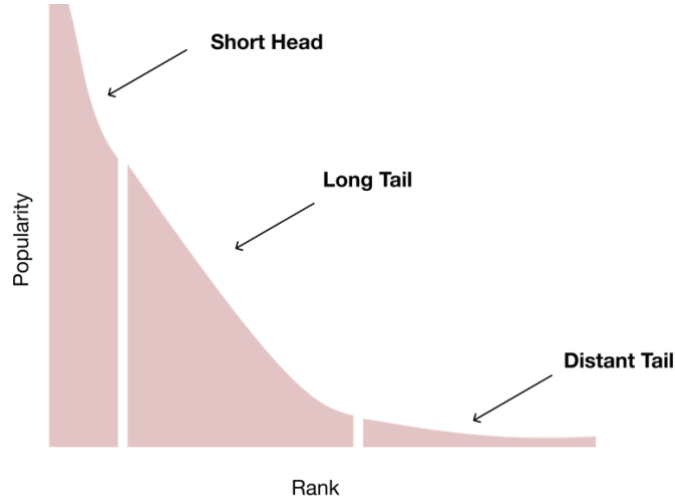


Figure 3. The Long-tail Phenomenon in Recommender System [22].

Figure 3 illustrates that phenomenon in recommender systems. The first white vertical line divides the short head, which is 20% of the items and the tail which accounts for more than 80% of the items. The short head part is extremely popular items which are often offered to customers by physical stores. The tail part consists of two different areas. The long tail is the most distribution of items which are potential that should be included by the recommender system, even though most recommender systems often ignore them. The distant tail includes items with few ratings as known as cold-star items which may have a bad effect on a

collaborative filtering recommender system. As the goal of the recommender system is to build personalized recommendations for each user, the regular problem caused by the biased popularity is that the recommendation lists are very identical amongst users since they include a lot of popular items. A recommender system that only produces very similar recommendations for every user should not be considered as a reliable system.

2.2.3.1. Explicit Query Aspect Diversification

A solution to the above described problem is the Explicit Query Aspect Diversification (xQuAD), which has been used in the context of information retrieval, such as web search engines. The goal of the xQuAD algorithm [22] is to use the existing ranked recommendation list L to produce a new recommend list S ($|S| < |L|$) that manages to control the popularity bias while keeping the acceptable accuracy. The new list is built iteratively using the formula:

$$(1 - \lambda)P(v|u) + \lambda P(v, S'|u)$$

Equation 11. Explicit Query Aspect Diversification.

where $P(v|u)$ is the probability item v is interesting user u and $P(v, S'|u)$ denotes the probability of user u being interested in item v which is not currently in the recommend list S . The parameter λ is used to adjust the popularity bias. There are two different versions of xQuAD, one is the *Binary xQuAD* and another is *Smooth xQuAD* [22]. When selecting an item to be added to list S , the ranking score for each item in the recommendation list is computed as:

$$score = (1 - \lambda)P(v|u) + \lambda \sum_{c \in \{\Gamma, \Gamma'\}} P(c|u)P(v|c) \prod_{i \in S} (1 - P(i|c, S))$$

Equation 12. Ranking score using XQuAD.

Briefly, for an item $v' \in c$, if the recommend list S does not cover (included in) area c (c represents either long-tail (Γ) or short-head (Γ') area), then the estimated user rating or preference will include the additional positive term. Thus, this increases the chance of an item to be selected, balancing the accuracy and the popularity bias.

The Smooth xQuAD adds extra value by adding the term $(1 - P(i|d, S))$ to the overall score for items in an under-represented category. This helps balance the popularity bias, but it also lifts the diversity-accuracy trade-off. On the other hand, in the Binary xQuAD the term $(1 - P(i|d, S))$ only equals to 1 if all of the items in the current recommendation list S haven't covered the category d yet. As soon as one item in a category enters the list S , the

term will be 0 and the positive summation term will not be added to the overall score. This means that the score will be dominated by the base recommender. Therefore, the Binary xQuAD only seeks for the best long-tail item it can instead of diversity.

CHAPTER 3

IMPLEMENTATION

3.1. Libraries and Frameworks

There are some open-sourced libraries and frameworks used to build the application. They can be categorized into two groups: one that is used to build the web application and the other is used to build the machine learning model and the recommender system.

3.1.1. ReactJS

ReactJS is an open-sourced JavaScript library, developed by Facebook, for building the user interface. ReactJS is component-based, meaning that instead of looking at the website page as a large picture, it divides the page into components that can be reused in other pages.

Choosing ReactJS to be the framework for building the user interface, the project is very well-organized and maintainable. Moreover, new pages can be constructed with ease using the existing elements, only new features need to be implemented. The front-end communicates with the server by making requests to it using normal HTTP requests.

3.1.2. NodeJS

NodeJS is an asynchronous event-driven JavaScript runtime, it is designed to build scalable network applications. NodeJS is typically used to build servers for the web application, however, it also allows developers to use its built-in rendering machine to build the front-end.

For this project, NodeJS is responsible for the back-end of the web application. It serves as a middleware between the front-end and the database. ExpressJS is used along with the NodeJS to build the API. The server API follows the RESTful principles. Moreover, the server also communicates with the recommender system to get the recommendations and store them to the database.

3.1.3. MongoDB

MongoDB is an open-sourced database management system (DBMS) which is written in C++. MongoDB uses a document-oriented database which means it stores data in JSON-format documents. It is classified as NoSQL (i.e, it is a non-relational database) database type.

For this project, MongoDB is used to store all the data for the web application (e.g products, users, categories, etc). NodeJS can connect to the MongoDB database, query and aggregate documents from it, then send them to the web user interface.

3.1.4. Flask

Flask is a micro Python web framework. It does not depend on any libraries or tools. Its core is very simple but also extensible. Using Flask, everything is in controlled. It is able to choose any type of database or third-party libraries together with Flask.

For this project, taking advantage of the fact that Flask is implemented using Python, a lightweight server using Flask that acts as the recommender system for the application is implemented. It receives necessary data from the web application, using them to train a machine learning model. Then the model generates recommendations back to the web application.

3.1.5. Surprise-scikit

The surprise-scikit is an easy-to-use Python scikit for building recommender systems. It is implemented with care and maintained regularly.

For this project, the surprise-scikit library is used to experiment when building the machine learning model. Since it supports different algorithms like Item-based KNN, User-based KNN, SVD, and ALS, it is very easy to create a model for a recommender system that performs well.

3.2. System Design

The system consists of two primary sub-systems: the web application and the recommender system. The web application is where the data is displayed to customers. It is the platform for customers to browse and buy goods. Because the goal of this project is to build a lightweight but powerful web application, both the web application system design and interface design must orient toward that. Besides, the separate recommender system is responsible for processing data, make use of it then build a machine learning model that can be integrated into the web application, and make high-quality recommendations for customers. One key feature of the recommender system is that it must be able to produce recommendations in real-time. An overview of the system is illustrated in Figure 4.

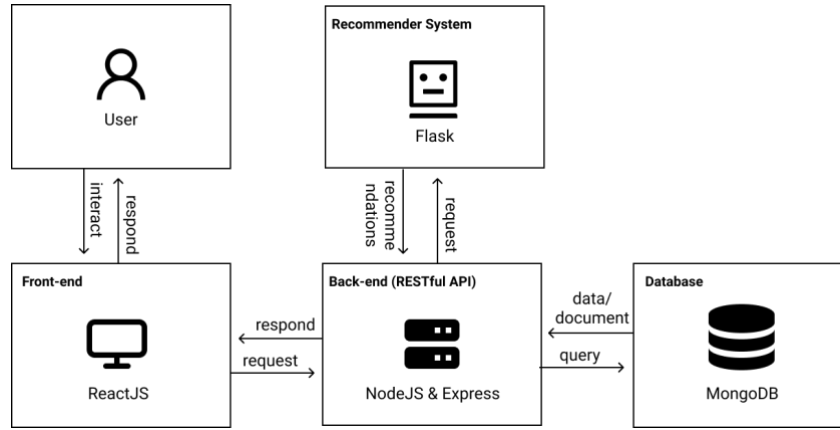


Figure 4. System Design Overview.

3.2.1. Web Application

The web application system can be sliced into three main layers: the interface (front-end), the server (back-end), and the database. Briefly, the front-end handles the interaction between users and the web application, then with these data, it requests the server, getting the right data from the server and displaying it on the websites.

3.2.1.1. Back-end

The back-end, as a RESTful API [24], oversees querying, aggregating data from the database then sending it to the front-end, and storing the new data. As usual, the database is where all the data (e.g, products, users, transactions, categories, etc) securely kept. The front-end and back-end communicate with each other using normal HTTP requests (e.g, GET, POST, PATCH, DELETE, etc).

Besides, some routes are secured, meaning that it requires keys (or tokens) to access. This helps secure users' data from attackers. In this project, the JSON web token (JWT) [25], which is usually used for authorization and information exchange, is used for the authentication. It secures the data transmitting between parties as a JSON object [26]. JWT consists of three parts, separated by dots: the header, the payload, and the signature. The header keeps the basic information about the token, which algorithm it uses, and which type it is. It is then encoded using the *Base64Url* algorithm. The payload typically keeps the data of the user and some additional data. Some common information that can be kept in the payload is the token's expiration time (*exp*), the subject (*sub*), and the audience (*audience*). The payload is also encoded. The last part is the signature, this is created using the header, the payload, and a secret key. Using the algorithm defined in the header, it verifies the validity of

the request, also, it can check the identification of the request's sender. In this project, every time a user logs in successfully, a JWT is returned and saved to the website's local storage. Later, for every request to a secured route, a valid JWT is required in the request's header for authentication. Note that all the information in the token is publicly exposed to anyone, even though they can't change it. This means private information should never be included in the token.

To communicate with the MongoDB database, the server uses the Mongoose library. Mongoose uses Schema to work with MongoDB. Mongoose maps each schema to a MongoDB's collection and defines the shape of the document within that collection. Then it creates models based on defined schemas. Later, these models can be used to easily make queries to the database anywhere in the server.

3.2.1.2. Front-end

The front-end layer is the least abstract layer, it is the visual of the website. Most of the interaction between users and the website occurs in this layer. Using ReactJS, a component-based framework, each page is the combination of many small components. Some components are shared between different web pages, i.e they are reusable. It is very easy for one to create a new web page using ReactJS, all they need to do is grab some components and plug them in the new web page and it works just fine. Figure 5 shows the typical web page structure using components. The front-end is also capable of requesting to the server, extracting user interaction into usable data for further experiment. Moreover, there are pages that only users with admin role can access. These pages allow admins to access the database in an easy way, search, modify, import, and save data directly to the database.

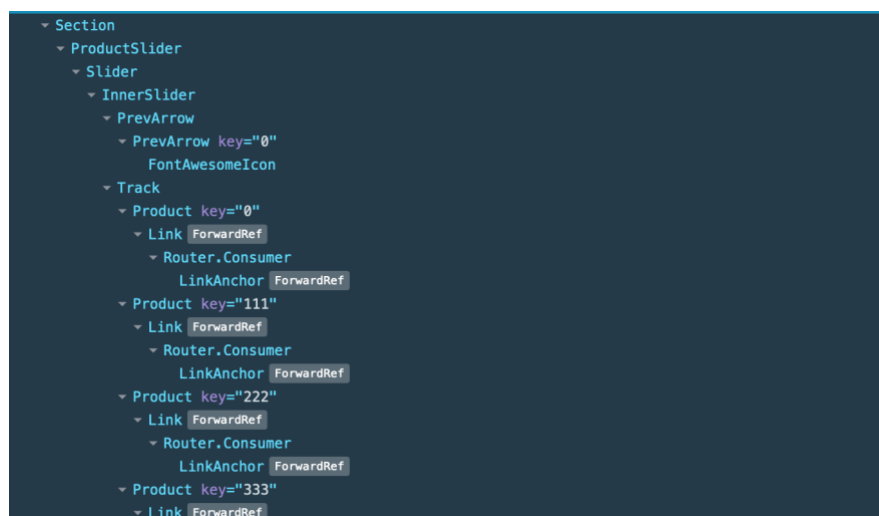


Figure 5. The sample structure of a web page using ReactJS.

3.2.1.3. Database

In this project, the database is designed based on Amazon's Electronic dataset for more convenient experiment and evaluation while keeping the web application to function properly.

It consists of five primary collections (or models) which are the categories, products, ratings, transactions, and users. MongoDB allows each document in a collection to have an embedded document or sub-document, this helps create the relationship (one-to-one, one-to-many) between connected data. Embedding connected data in a document also helps reduce the number of reading operations required to retrieve data. All the sub-documents are stored as an "*ObjectId*", which is created by default at the time each document is initialized, in the main document. To retrieve the sub-documents from a document, Mongoose uses the function called "*populate*" to reference documents in other collections. Figure 6 is the schema of the database.

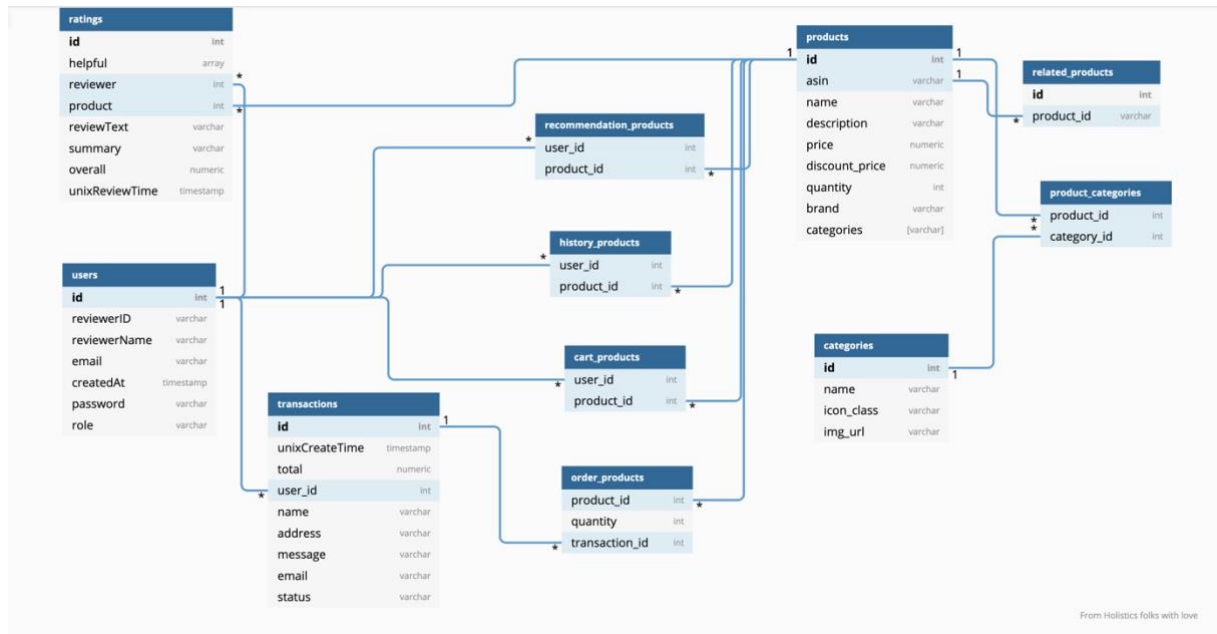


Figure 6. Database Schema.

There are 5 main entities in the database: users, products, ratings, transactions, and categories. Each user has their private information like id, name, email, and password. Besides, they also keep their ratings, transactions as well as recommended products, browsing history, and cart products. Users are categorized into two types based on their role, one is the usual customer and the other is the administrator. Customers use the web page for shopping while administrators use it for managing and organizing the store. The second basic entity is the product. Each product stores the detailed information of itself (e.g id, Amazon Standard Identification Number (ASIN), name, description, price, stock quantity, brand, and

categories). Moreover, each product is linked to their related products, which are generated by the recommender system. The third entity is the category, it is used to keep each category name and image. The most crucial entity for the recommender system is the rating. Each rating keeps its review text, the reviewer, the reviewed product, and the overall rating score. The rating summary, time, and helpful score are also recorded but for the current recommender system, only the first three columns of information are useful. Finally, the transaction keeps track of all user's transactions. It keeps the necessary information for delivery such as the transaction owner's name, id, email, the total price, all the products included in that transaction and the transaction status (whether it is shipped or not).

3.2.1.4. User Interface Design

For the interface of the web application, the design is determined to be modest but functional. There are two versions of the web interface, one for desktop users and the other for mobile users. Because of the mobile limitation, some features in the desktop version may not be accessible in the mobile version.

The home page is where most of the products are displayed. It consists of four sliders. The category slider shows categories based on the user's rating; in case the user is new, random categories are shown. The browsing history is recorded every time the user visits a product, then it is loaded from the database if the user has logged-in, otherwise, from the browser's local storage. The user's recommendation products slider is the result of the recommender system, it shows the products that are predicted to be relevant to the user. Finally, some random products are shown as the bottom of the page.

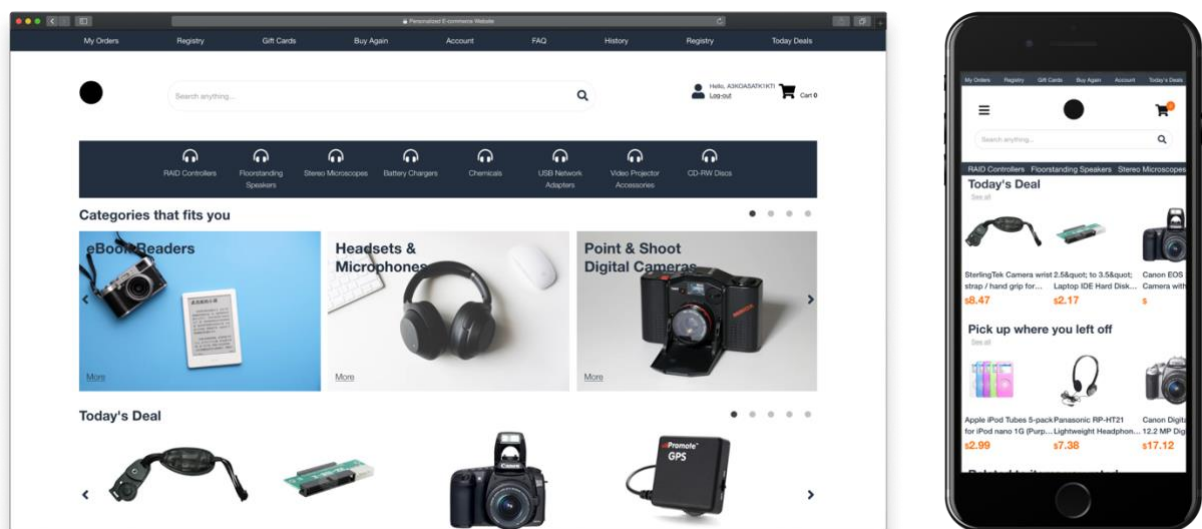
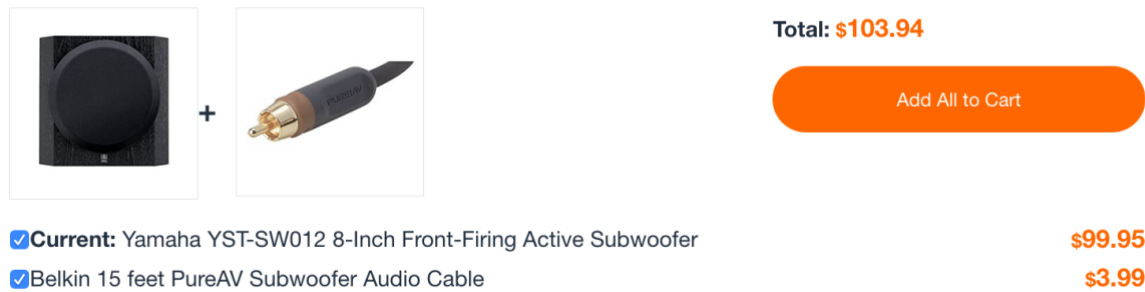


Figure 7. Home page user interface.

The product detail page shows the product's price, description, customers' reviews. It also shows the related products which are generated using the recommender system and products from the same category or brand. On this page, users can write their reviews and submit their rating on the product. The recommender system then catches this event to generate real-time recommendations. Also, the rating is saved in the database for later usage. The "Usually Bought Together" (Figure 8) block allows users to purchase multiple products with one click using the real data from Amazon's dataset. This page is where users decide whether to add the product to their cart, users can also choose the quantity of the product they want to purchase. They can also skip directly to the check-out phase using the "Buy Now" button.

Usually Bought Together



Total: \$103.94

Add All to Cart

- ✓ **Current:** Yamaha YST-SW012 8-Inch Front-Firing Active Subwoofer **\$99.95**
- ✓ Belkin 15 feet PureAV Subwoofer Audio Cable **\$3.99**

Figure 8. Bundle Purchase block.

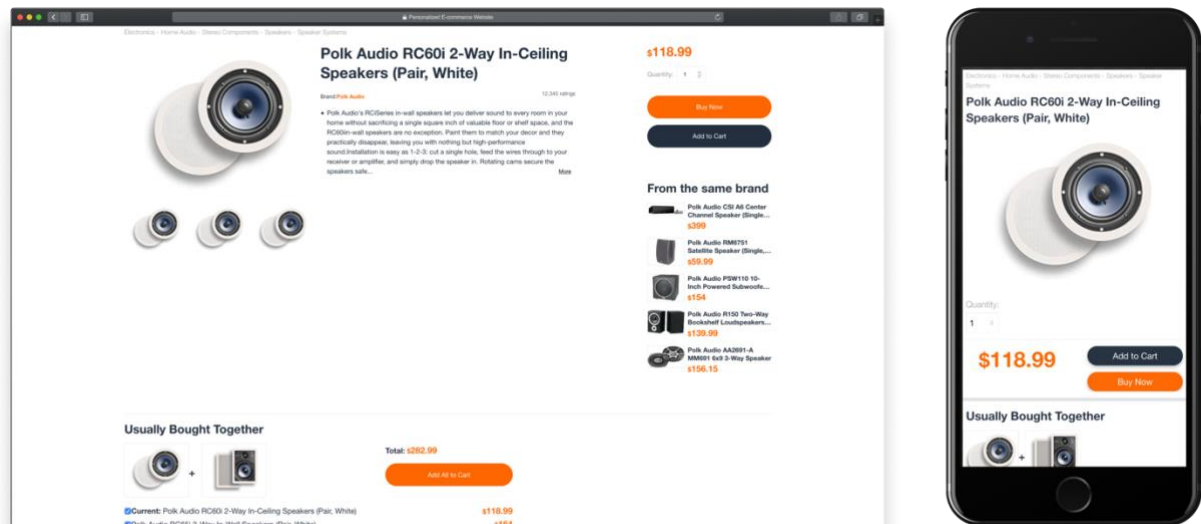


Figure 9. Product Detail page user interface.

When users search for a product using the search bar on the header of the web page, the server will use a regular expression to search for the product whose name, ASIN, or brand

matches the string in the query. The search page shows the result of the search query. In case the search string doesn't match any product, the interface indicates that there are no results for the string query. It also supports sorting and filtering products for a better user experience. Besides, using the categories located in the header, users can browse all products from a specific category. Note that there are some empty categories since the products imported is just a subset of Amazon's dataset. The category search page has almost the same interface as the search page and supports basic sorting functions.

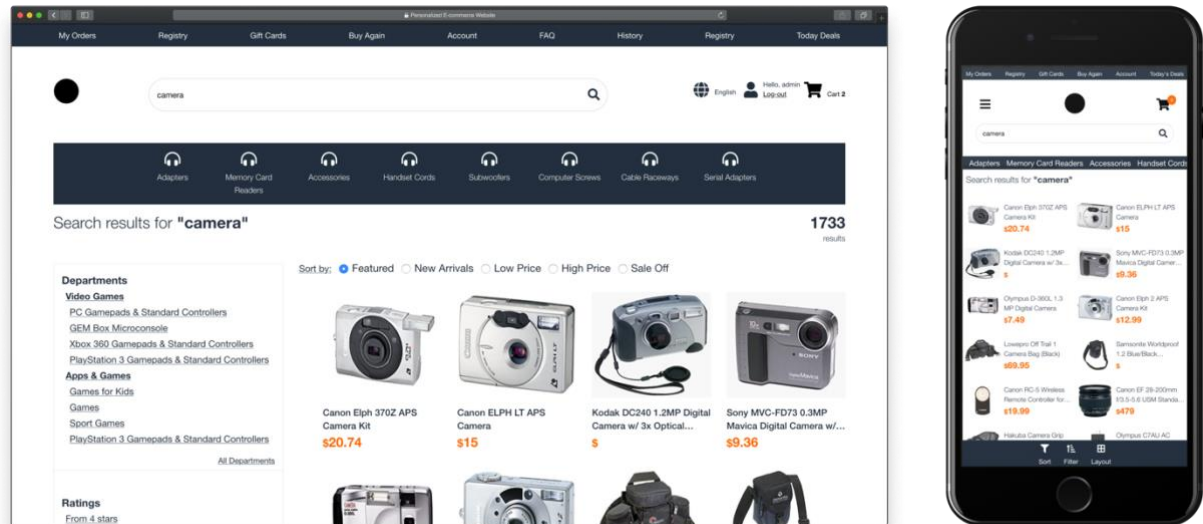


Figure 10. Search page user interface.

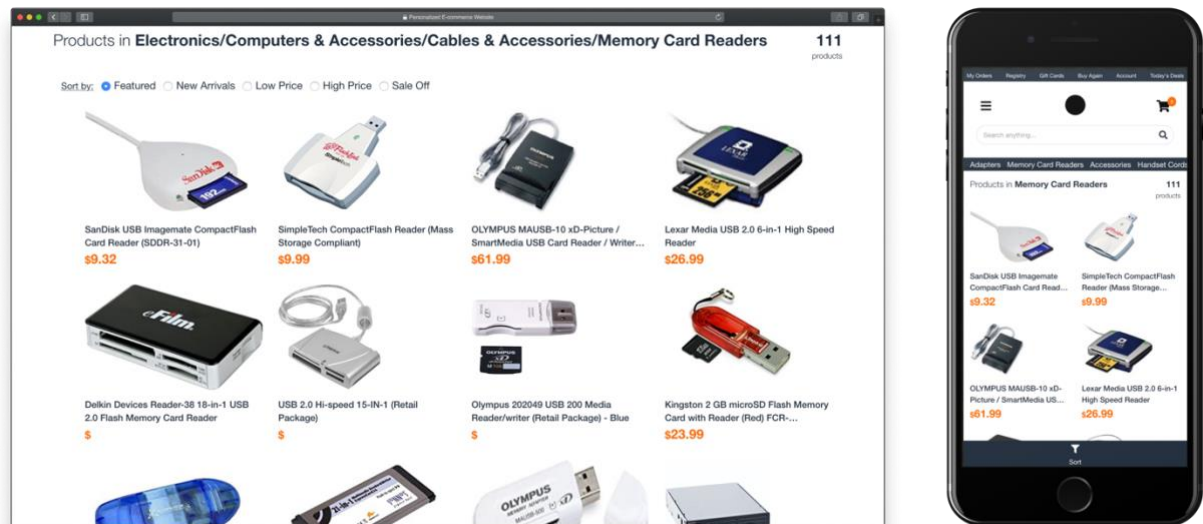


Figure 11. Category page user interface.

The last page for a usual user is the check-out page. On this page, users can see the products that they are going to buy, their price, and quantity. There is a summarization of the purchase that computes the sub-total of the order. Besides, there is a form for users to fill in

with their shipping information. Once all the information is full-filled and validated, a pop-up appears to confirm that the order is successfully placed. After that, a confirmation email is automatically sent to the user's registered email (Figure 12). Each user can track their orders using the built-in order tracking page.

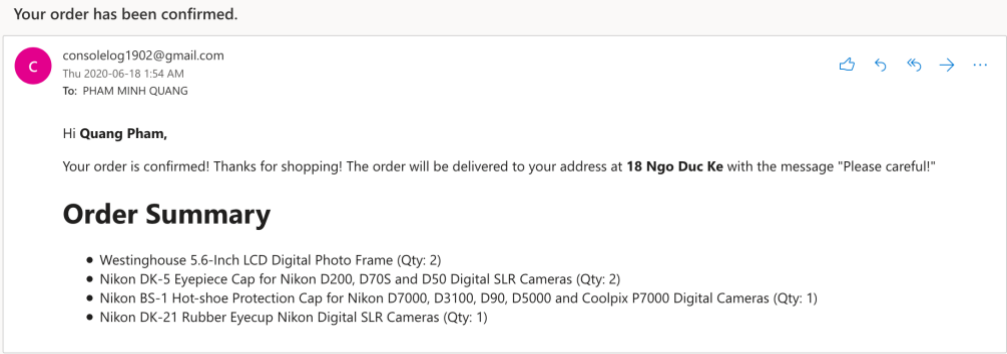


Figure 12. Sample order confirmation email.

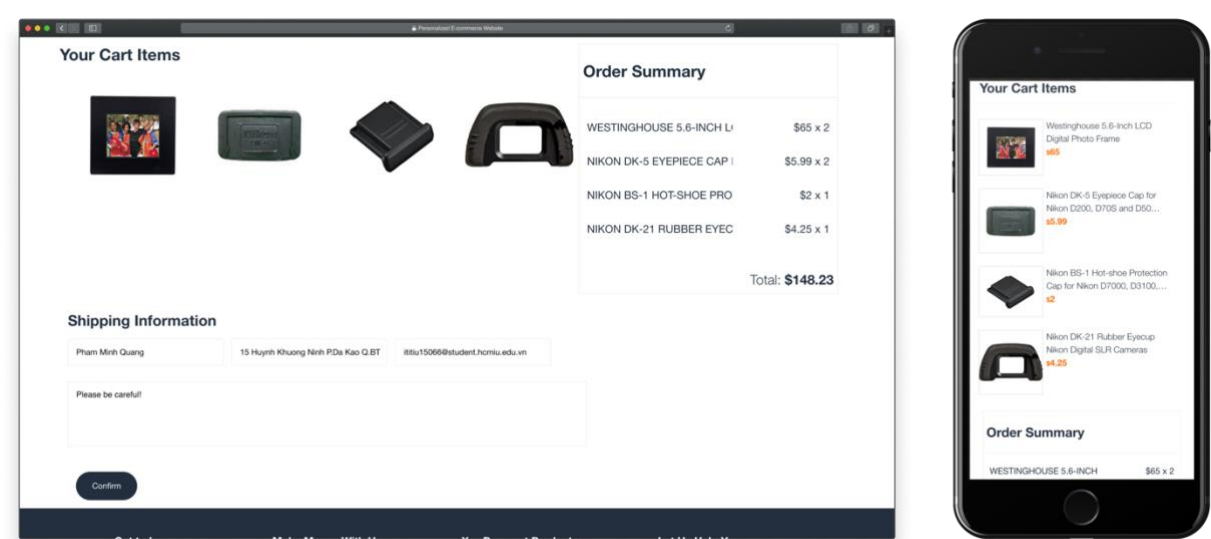


Figure 13. Check-out page user interface.

Besides pages for buyers, some pages are made dedicated to administrators. The first page is useful for admins to manage (create, read, update, and delete) the database's collections (products, users, and categories). For each collection, the admin can create a new document, browse all documents, import documents from a JSON file, and export documents from the database into a JSON or CSV file.

Products
Create New Product
Product List
Import/Export

Users
Create New User
Users List
Import/Export

Categories
Create New Category
Category List

Product List

Showing 20 of 97 products

Sort by: Newest

Category: Electronics, Accessories & Supplies, Audio & Video Accessories, I

ASIN	Name	Price	Disc. Price	Image		
B0000TNZDK	Logitech Harmony SST-659 Universal Remote Control (Grey) (Discontinued by Manufacturer)	\$67.99			✕	Edit
B000623HE8	Logitech Harmony 680 Universal Remote Control (Discontinued by Manufacturer)	\$99.99			✕	Edit
B0007PCB6O	Universal Remote Control MX-350 Complete Control Osiris IRRF Remote	\$127.5			✕	Edit
B00093IIRA	Logitech Harmony 880 Advanced Universal Remote Control	\$379.99			✕	Edit
B000F7JCRA	Sony RM-VL600 8-Device Universal Learning Remote (Discontinued by Manufacturer)	\$89.75			✕	Edit

Figure 14. Document Management page for admins.

Hello, admin
Logout
Cart 0

Home Audio Crossovers &...
Accessories
Camcorder
Laptop Network Adapters
Radar Systems
Antitrust
Adapters & Converters
Lighting

Products
Create New Product
Product List
Import/Export

Import Products

Upload your JSON file here: No file selected.

Export Products

Export all products into a CSV file.

File Type: CSV

You have chosen to open:

products.csv
which is: Comma Separated Spreadsheet (.csv) (23.9 from: blob)

What should Firefox do with this file?

☐ Open with: Microsoft Excel (default)

☒ Save File

☐ Do this automatically for files like this from now on.

Money With Us

Blog

Sell Your Services on Xyz

Xyz Payment Products

Xyz Business Card

Shop With Points

Let Us Help You

Your Account

Your Order

Figure 15. Document Export/Import page for admins.

Additionally, since there is a recommender system integrated into the web application, there is a page for the admin to use the recommender system without having to code, called RecLab (Recommender System Lab). This page allows admins to upload a new dataset, backup the existing dataset, train the model, and save it using their configuration. When the admin uploads the CSV file using the file uploader, a preview of the data appears. The preview shows some rows of the data for the admin to make sure that they uploaded the correct file. When the admin presses the upload button, the server will check if the file is not empty and is in the correct format. If the validation fails, a message will appear to indicate the

error. Admins also can download a back-up of the dataset on the server, in case some errors happen.

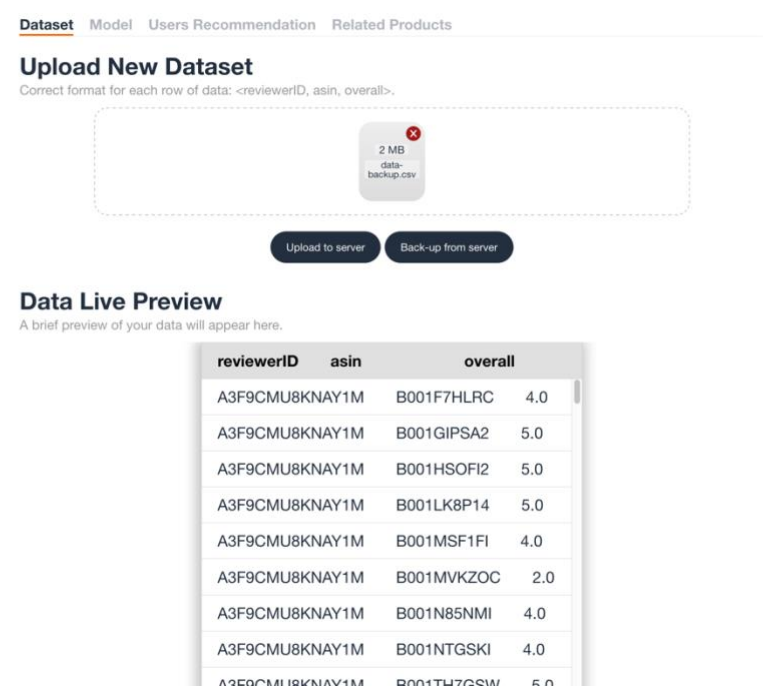


Figure 16. Dataset Management page for admins.

The RecLab page also allows admins to experiment with the model. The admins can choose the model parameters and algorithms for training, then decide whether to save it on the server or their computer. Once the model is trained, the interface displays the experimental results, which are the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). This page helps admins, who are non-technical people, to do the machine learning stuff in the most intuitive way.

RecLab

Manage recommender system dataset, train and test model.

Dataset **Model** Users Recommendation Related Products

Train Your Model

Select the model, the dataset, and the parameters for training.

Select algorithm:

Incremental SVD

Training Type:

80:20

Dataset:

On server

Number of factors

20

Number of epochs

100

Learning rate

0.005

Reg. Parameter

0.1

Random State

42

☐ Save model to server(*CAUTION: This will overrides the last trained model)

☐ Save model to your computer.

Train Model

Training Results

Information about your trained model will appear here.

mae: 0.7703473842537791

rmse:

1.0634111930290355

Figure 17. Model Training page for admins.

Since the initial recommendations for users (or related products for products) must be produced manually at first, there is a section to fasten the process, in which the admin can do such thing with only a click. The admin can create recommendations for many users at once. The recommender system uses the pre-trained Incremental SVD model on the server to do so. For generating the related products, the recommender system uses the Item-based KNN model instead.

Dataset Model **Users Recommendation** Related Products

Generate Recommendations for Users

Handy user recommendation generator using Incremental SVD model.

83

Showing 17 of 17 users

<input checked="" type="checkbox"/>	username
<input checked="" type="checkbox"/>	AFXTKAO0CB354
<input type="checkbox"/>	A1435P5AMCPB3X
<input checked="" type="checkbox"/>	A2PB325LKXYDOL
<input checked="" type="checkbox"/>	APKBGB3JBWL5X
<input checked="" type="checkbox"/>	A150QS4IZB3XJ
<input checked="" type="checkbox"/>	A38T55CCT4B3E6
<input checked="" type="checkbox"/>	A1P3DDFL423B3C
<input checked="" type="checkbox"/>	A2FRTX7JBU8B3Y
<input checked="" type="checkbox"/>	A8CKH8XB33XGN

Number of recommendation products 50

Generate

Figure 18. User Recommendation Generator page for admins.

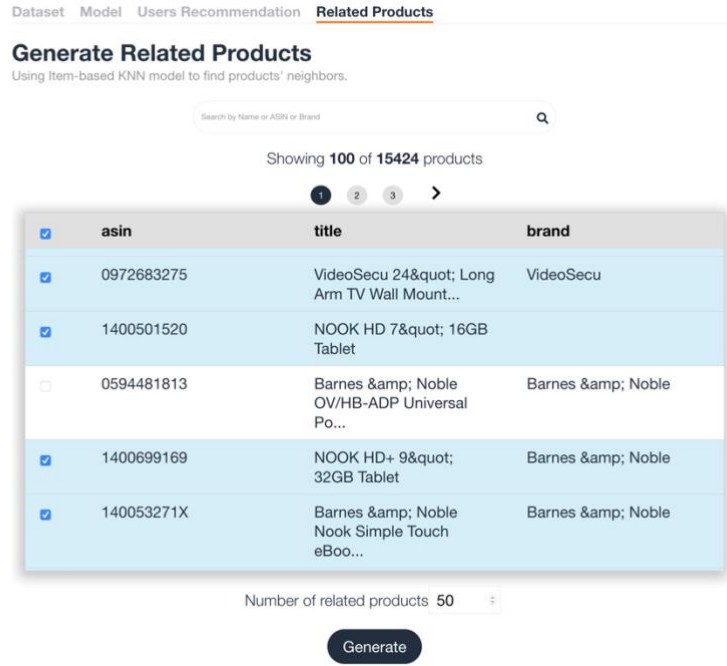


Figure 19. Related Products Generator page for admins.

3.2.2. Recommender System

In this project, the process of building the recommender system consists of two primary parts. Firstly, experiments and optimizations are done with the recommendation model for predicting the user ratings (Figure 20). Then, the xQuAD method is used to improve the relevance and reliability of the top-K recommendation generated by the recommender system in the first phase and use it in the web application to recommend products for users (Figure 21).

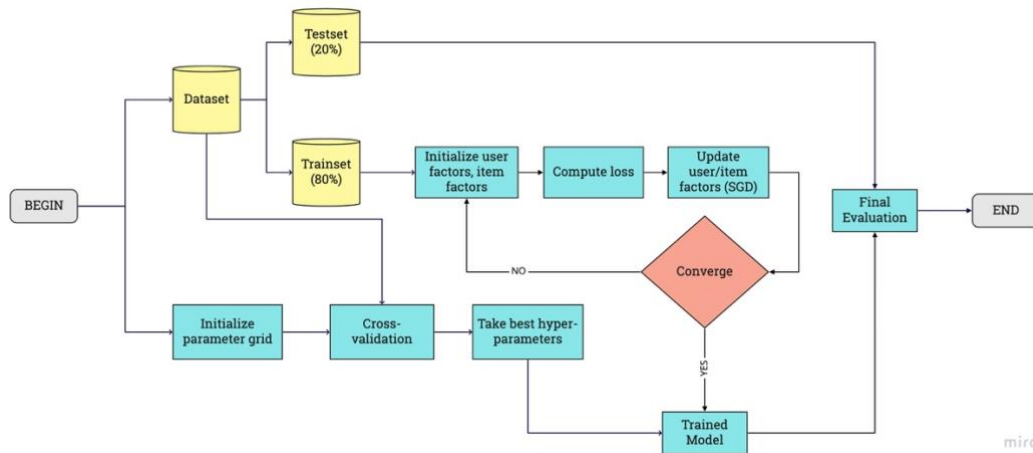


Figure 20. The off-line stage of training and tuning the recommendation model.

To optimize the model for prediction accuracy, experiments are conducted with same training dataset and testing dataset on four different models which are Item-based, User-based K-

Nearest Neighbors, SVD, and ALS. Random Search [23] is used to find the optimal hyperparameters. Two evaluation metrics Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are used to evaluate prediction accuracy. The mathematical definition of RMSE and MAE are in Equation 13 and Equation 14.

$$MAE = \frac{1}{n} \sum_{u,i} |\hat{r}_{ui} - r_{ui}|$$

Equation 13. Mean Absolute Error.

$$RMSE = \sqrt{\frac{1}{n} \sum_{u,i} (\hat{r}_{ui} - r_{ui})^2}$$

Equation 14. Root Mean Squared Error.

where, \hat{r}_{ui} and r_{ui} are the predicted and actual rating of user u on item i and n is the number of ratings.

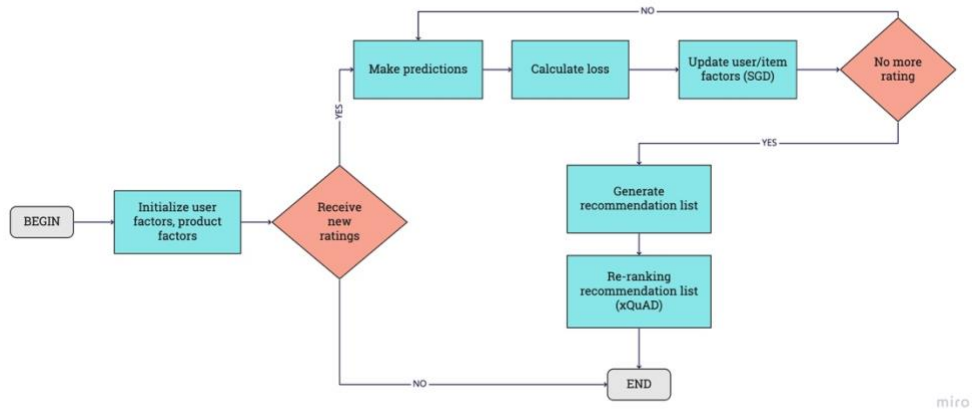


Figure 21. The online stage of producing real-time recommendations.

By experiments, the SVD model, which has the best performance is used for producing personalized recommendations. As mentioned before, an incremental version of SVD will also be implemented. The approach to implementing Incremental SVD is slightly different from the method in [21]. The Incremental SVD model functions the same way as SVD model but it allows partial fitting in which the training starts with the previous model. In other words, only computation of new ratings needed to be done.

Then, to produce less biased recommendations, the flexible approach is followed to control the balance of the recommended items distribution in different areas, introduced in [22]. This approach enables the system to be tuned to have better coverage of long-tail items

while having acceptable accuracy. The framework is built using the xQuAD algorithm introduced in **2.2.3.1**. The re-ranked recommendation list, whose size is smaller than the original list, is the final recommendation list.

A simple Flask server is then established to serve as a bridge for the web application to communicate with the recommender system. The Flask server receives requests from the web application server whenever there is a new rating and feeds the data from the request to the recommender system. The recommender system then follows the process introduced above to generate recommendations and sends it back, as a response, to the web application server where it is stored in the database.

Besides, the trained Item-based KNN model is used to get the related products (neighbor products) for each product. These related products are navigation for users to find products that are close to products of their interest.

Since the web application also supports building recommendations in large batches for admins, the recommender system also has some specific routes for that. Using admin pages, admins can select what products or users to build recommendations for. The recommender system is also where the dataset for training recommendation models and the model itself are stored. The dataset can only be updated or downloaded by admins (Figure 16). For security, each request to the recommender system is authenticated by the back-end. If the validation failed, the request is also aborted.

3.2.2.1. Product Recommendation Engine Construction

When a user submits the review using the form provided in the product detail page, the front-end sends the reviewer name, the product ASIN, the overall rating as an object attached to the request to the backend server. The back-end uses the data received to make another request to the recommender system server to build the personalized recommendation list. If the user or the product is new (not in the training set), the recommender system feeds the rating to the pre-trained SVD model using the incremental approach in Section 3.2.2. Otherwise, the recommendation system uses the existing data to make a recommendation list. The xQuAD algorithm (introduced in Section 2.2.3.1) is used to increase the coverage of long-tail items of the recommendation list. The recommendation list is saved to the database and used by the front-end to display the recommendation list on the home page. Figure 22 summarizes the process of generating user recommendations.

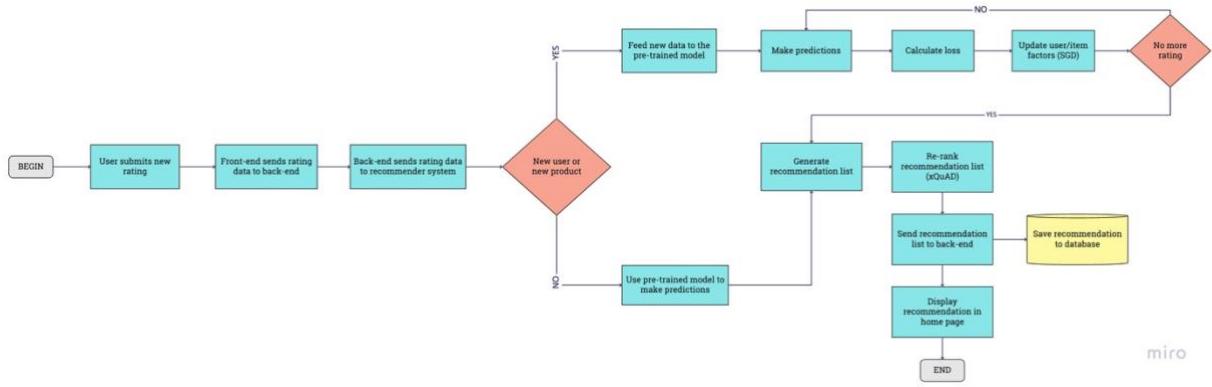


Figure 22. Flowchart for recommendation engine.

In detail, the recommender system uses the pre-trained SVD model to predict the user's rating on all products and recommend the top highest rated products to the user. To predict the rating for a pair of user and product, the recommender system first gets the user's factor p_u and bias b_u and the product's factor q_i and bias b_i . For existing users and products in the train data, the prediction is computed using Equation 9. Otherwise, if the product or the user is unknown, its bias is set to 0 and its factor is randomly initialized from a normal distribution using the model's mean and standard deviation (the default mean and the standard deviation is set to 0 and 0.1, respectively). The model then iteratively runs the Stochastic Gradient Descent for one epoch to update the new user or product biases and factors. After the partial fitting part, for each user, the model predicts the rating of that user and all products. The top-N highest rated products are used to build the recommendation list. The xQuAD algorithm is then used to produce a re-ranked recommendation list, whose size is smaller than the original list, in order to minimize the effect of the popularity bias problem. The output recommendation list consists of 50 to 100 products is the final recommendation for the user. It is then displayed in the user interface (Figure 23).

Related to items you rated



Figure 23. The personalized recommendation list displayed in the home page.

3.2.2.2. Related Products Construction

For each product, the administrator can make a POST request to the recommender system to find its related products. The recommender system then uses the pre-trained Item-based KNN model to find K-nearest neighbors of a product. To get the K-nearest neighbors of a product, the recommender system uses Item-based KNN model to get K number of products that have the highest similarity score.

Once the related products are generated, they are saved to the database and displayed on the product detail page (Figure 24).



Figure 24. The related products of a product displayed in the product detail page.

For ease-of-use, administrators can use the utility page to build large amounts of related products, same as generating recommendation for users. In this page, the admins can find the product they want to process using the built-in search page, then choose the number of related products they want to create, press the “Generate” button and just wait for the process to complete.

CHAPTER 4

EXPERIMENTAL RESULTS AND EVALUATION

4.1.1. Experimental Platform

In this section, four models are evaluated using Amazon’s Electronic dataset [29]. The dataset has 7,824,482 ratings of 4,201,696 distinct users and 476,002 distinct products. After taking a deep look into the dataset, it is observed that most users have rated less than 10 times (the 99th percentile of number of ratings per user is 12 ratings). For accurate evaluation on the top-N recommendation, an appropriate number of ratings for each user is necessary, therefore, users with less than 50 ratings are removed. The size of the dataset is reduced to 125,871 ratings of 1,540 users and 48,190 items. Without the pre-processing, the dataset can adversely affect the quality of the model. The training set size is 80% of the dataset and the rest are for the testing set.

user	item	overall
A3BY5KCNQZXV5U	B000AM6QH4	5.0
AT09WGFUM934H	B00005105L	4.0
A1K4G5YJDJQI6Q	B00B57PXL I	3.0
A328S9RN3U5M68	B00JGL37FO	5.0

Table 1. A preview of Amazon Electronics dataset.

4.1.2. Evaluation Metrics and Results

For this experiment, the framework is evaluated on two different types of metrics. The prediction accuracy metric will measure how good the model is in making prediction. Besides, an additional experiment is done to demonstrate the quality of the incremental approach of the SVD-based model. Later, using the ranking accuracy metric, the effectiveness of the xQuAD algorithm on enhancing the top-N recommendation quality as well as its affection on the ranking accuracy of the system is also evaluated.

4.1.2.1. Prediction Accuracy

As mentioned in Section 3.2.2, the two metrics used for evaluating prediction accuracy are Root Mean Squared Error and Mean Absolute Error. Using the cross validation, the results in Table 2 shows the average accuracy evaluated on 5-fold cross validation. Bolded values

indicate the lowest error. The score of Incremental SVD stands out with the corresponding RMSE and MAE of approximate 0.985 and 0.727, respectively. The results of the two neighborhood models seem identical since their predictions are produced similarly. The ALS model performs worst amongst all.

	User-based KNN	Item-based KNN	Incremental SVD	ALS
RMSE	1.04473	1.056267	0.985255	1.205341
MAE	0.774074	0.761715	0.727166	0.928139

Table 2. Prediction accuracy comparison between different algorithms.

Besides, to observe the performance of the incremental SVD approach, the training set is divided into two subsets with the ratio of x and $1 - x$. First, the model is trained on the larger subset. Later, follow the incremental approach, the other sub-set (called the incremental set) is incrementally trained. Finally, the model is tested on the testing set and observe how the prediction accuracy varies compared to the original model (trained on the full training set).

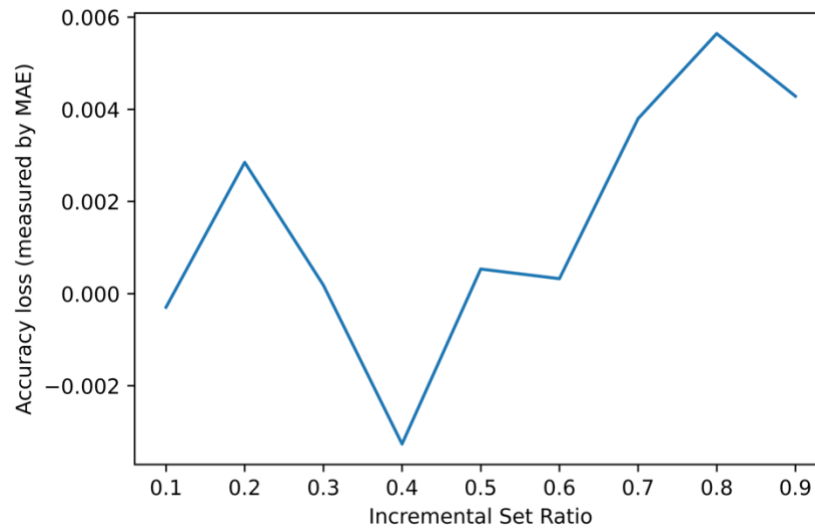


Figure 25. Accuracy loss over different incremental set ratios (MAE).

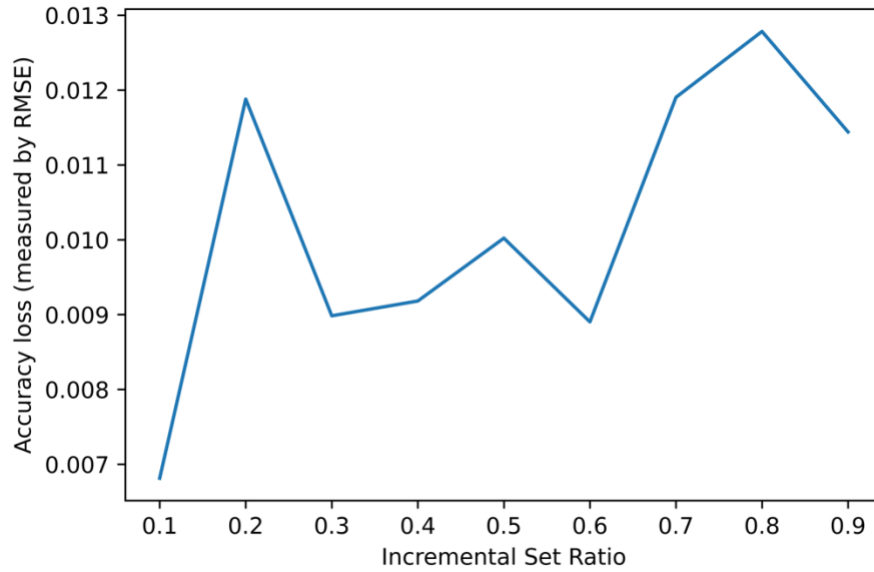


Figure 26. Accuracy loss over different incremental set ratio (RMSE).

Figure 25 and Figure 26 plot the prediction accuracy loss when training the model with different incremental data set ratios. The highest accuracy loss measured by MAE and RMSE are approximately 0.0056 (0.76% drop) and 0.0127 (1.28% drop), respectively. Both occur when the incremental set size is very large, about 80% of the full train set. This indicates that the model trained using the incremental technique is almost as accurate as the model trained on the full train set. It is very surprising that for the MAE metric when the incremental set ratio is 0.4, the model is more accurate than the original model (implies by the negative accuracy loss).

	User-based KNN	Item-based KNN	Incremental SVD (with highest accuracy loss)	ALS
RMSE	1.04473	1.05626	0.99993	1.205341
MAE	0.77407	0.76171	0.73303	0.928139

Table 3. Prediction accuracy comparison between the highest accuracy loss Incremental SVD model and other algorithms.

As implied in Table 3, even when the effect of incremental learning on prediction accuracy is biggest, i.e the model has the highest accuracy loss, its RMSE and MAE is still higher than other algorithms. This once again proves the efficiency of the Incremental SVD model when it isn't only able to incrementally learn the new data which saves a huge amount of time but also secures the quality of predictions of the original model.

Furthermore, the time needed to incrementally train a model is much smaller compared to the full train set model. Figure 27 shows that the larger the incremental set, the shorter the time it takes to train the model. This happens because, in incremental learning, we only train new ratings on one epoch which saves a lot of time. The dotted red line indicates the amount of time the model needs when training on the full train set.

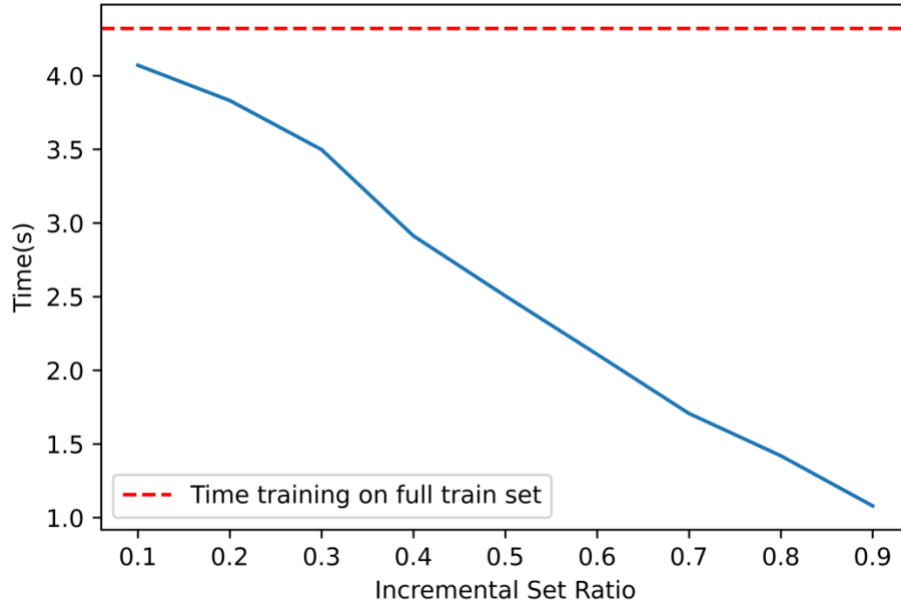


Figure 27. Training time over different incremental set ratio.

	User-based KNN	Item-based KNN	Incremental SVD (with the largest incremental set)	ALS
Training Time(s)	1.08905	167.6470	1.07949	187.5913

Table 4. Training time comparison between the Incremental SVD model with the largest incremental set and other algorithms.

4.1.2.2. Recommendation Ranking Accuracy

To evaluate the recommendation ranking accuracy, two different types of ranking metrics are used. One is the decision support metrics, in which Precision@M, Recall@M, and F1@M are computed. Another one measures how much the trade-off is between balancing popularity bias and ranking accuracy, in which the recommendation popularity, its coverage of long-tail items, and Normalized Discounted Cumulative Gain [27] are computed.

4.1.2.2.1. Precision@M, Recall@M, F1@M

In this project, a product is considered relevant to a user if it is rated higher than 3.0 and is recommended when the system predicts its rating to be high than 3.0. By that, Precision@M is computed as the proportion of the first M recommended items that are relevant and Recall@M is the proportion of relevant items that are recommended in the first M recommended items. The F1@M is then computed as:

$$F1@M = 2 * \left(\frac{Precision@M * Recall@M}{Precision@M + Recall@M} \right)$$

Equation 15. F1@M.

Table 5 shows the average results of precision and recall for different M (M either equals to 5 or 10). Bolded values indicate the highest score. The accuracy of all models also seems to be very good. The SVD model gives the best performance amongst all when its score is the highest in most metrics. However, the difference is very small. Beside SVD, the ALS model also does a pretty good job of giving relevant recommendations. Its precision-recall trade-off (illustrates in Figure 28) of the ALS model is smaller compared to the SVD model. The SVD model's precision starts at 0.94 (for M equals to 5) but steeply declines to 0.93 (0.01 drop) when M increases to 10. On the other hand, the ALS model's precision only drops for approximately 0.002 when M increases to 10. It can be concluded that the ALS model is affected less by the precision-recall trade-off and have a better score for Recall@5 and F1@5 compared to the Incremental SVD. However, when more items are included in the computation (Precision@10, Recall@10, and F1@10) the Incremental SVD starts to shine.

	User-based KNN	Item-based KNN	Incremental SVD	ALS
Precision@5	0.927435	0.9350	0.94149	0.931838
Precision@10	0.920895	0.926721	0.92983	0.929331
Recall@5	0.386646	0.390225	0.394250	0.449436
Recall@10	0.708193	0.709384	0.718627	0.705142
F1@5	0.545764	0.5506	0.555771	0.606399
F1@10	0.800658	0.803617	0.810702	0.801861

Table 5. Precision@M, Recall@M, F1@M comparison between different algorithms.

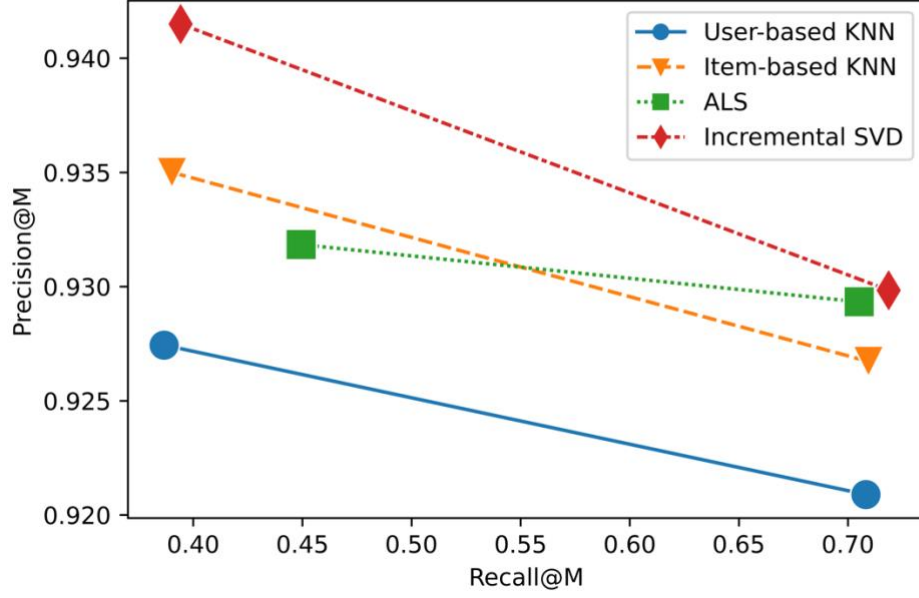


Figure 28. The Precision-Recall trade-off of different algorithms.

4.1.2.2.2. Average Recommendation Popularity (ARP) [22]

In this metric, the average popularity of the recommended list is computed as:

$$ARP = \frac{1}{|U_t|} \sum_{u \in U_t} \frac{\sum_{i \in L_u} \phi(i)}{|L_u|}$$

Equation 16. ARP.

where $\phi(i)$ is the number of times product i has been rated in the training data, L_u is the recommend list for user u and U_t is a set of users in the testing data.

4.1.2.2.3. Average Percentage of Long-tail Items [22]

This metric is useful to compute the average proportion of long-tail items in the recommended list for users:

$$APLT = \frac{1}{|U_t|} \sum_{u \in U_t} \frac{|\{i, i \in (L_u \cap \Gamma)\}|}{|L_u|}$$

Equation 17. APLT.

where Γ represents the long-tail items. This gives the average ratio of items in each user's recommended list that is covered by the long-tail set.

4.1.2.2.4. Coverage of Long-tail Items [22]

Since the APLT could be high even when the system recommends different users with the same set of items, this metric measures the fragment of long-tail items that the recommendation list has covered amongst the testing users:

$$Coverage = \frac{\sum_{u \in U_t} \sum_{i \in L_u} 1(i \in \Gamma)}{|\Gamma|}$$

Equation 18. Coverage of Long-tail Items.

where $1(i \in \Gamma)$ equals to 1 when item i in Γ , L_u the recommendation list for each user u in the test set U_t , and Γ is the long-tail items set.

4.1.2.2.5. Normalized Discounted Cumulative Gain (NDCG)

Discounted Cumulative Gain (DCG), a popular ranking metric, can be computed as the weighted sum of the relevance of ranked items. Besides, Ideal DCG (IDCG) can be computed by sorting the recommendation list by the true relevance, in descending order, then use the same formula as DCG. DCG assumes that highly relevant items are better than marginally relevant items so that it buffs items at the top of the ranking while adding a penalty for relevant items at lower rank. DCG can be computed as:

$$DCG = relevance_1 + \sum_{i=2}^k \frac{relevance_i}{\log_2(i + 1)}$$

Equation 19. Discounted Cumulative Gain (DCG).

where $relevance_i$ is the graded relevance of the result at position i in the recommendation list.

In this project, to compute the NDCG for a user's recommendation list, the descending sorted rating list of that user (in the test set) is turned into a boolean list, where it's 1 if the rating is higher than 3.0, otherwise 0. This list is then used to compute the IDCG. The recommendation list also converted into a boolean list as well, where it's 1 if the product is

rated higher than 3.0 in the test set, this boolean list is then used to compute the DCG of the recommendation list. The NDCG score for each top-N recommendation list is the result of the IDCG divided by the DCG.

The dataset is divided into two sub-sets: the short-head set which includes approximately 21.5% most rated products and the rest for the long-tail set. Then, to build the top-N recommendation, the recommender system uses the trained model to predict each rating in the testing set. In this experiment, the top-50 highest-rated products for each user are considered as their recommendation. From the top-50 recommendation list, the xQuAD algorithm is applied to get the final ranked top-10 recommendation list.

The result in Figure 29 is measured using two different versions of the xQuAD algorithm on re-ranking the recommendation list. The λ in the x-axis label is the regularization parameter of the ranking scoring function, it adjusts how strongly the biased popularity is controlled. As the λ gets bigger, more long-tail items are included in the recommendation list (measured by APLT and ACLT), and the average popularity of the recommendation decreases (measured by ARP). As expected, the Binary xQuAD has a modest effect on the diversity of the recommendation because it just tries to include the best long-tail item it can and stop once it achieves it. On the other hand, the Smooth xQuAD does a very good job of increasing the number of long-tail items and decreasing the popularity of the recommendation list. Surprisingly, Figure 30 shows that the Smooth xQuAD also increases the ranking accuracy (measured by NDCG). This happens since the users in the test set tend to prefer long-tail items over short-head items. So when applying to the xQuAD method, both the ranking and the number of long-tail items increase, which results in increasing the ranking accuracy. This goes the opposite direction with the result in [22] where the ranking accuracy drops when the regularization parameter is bigger.

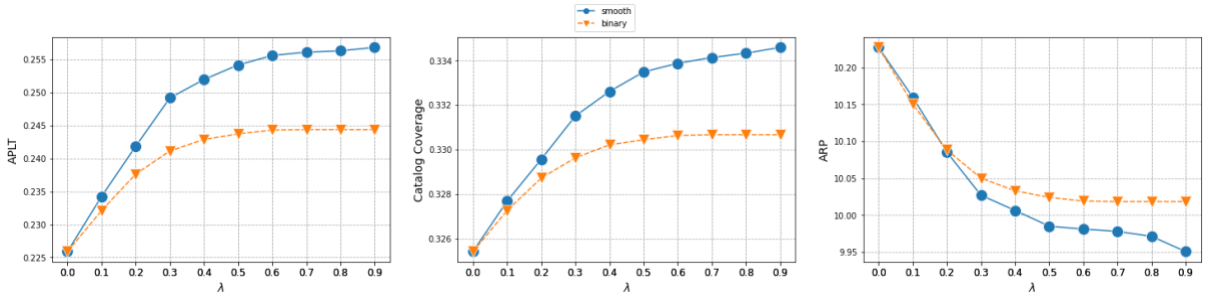


Figure 29. Long-tail diversity metrics.

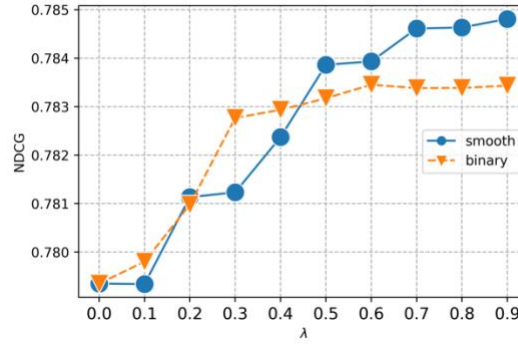


Figure 30. NDCG.

4.1.3. Average Top-M Recommendation Hit Ratio [28]

The top-M hit ratio can be computed as the fraction of products that are highly rated by the user in the test set that also appear in the top-M of the recommendation list. The top-M hit ratio for a user u is calculated as:

$$H(m, u) = \frac{N(m, u)}{N(u)}$$

Equation 20. Top-M Hit Ratio [28].

where $N(m, u)$ is m number of user's highly rated products, and $N(u)$ is the number of products in the recommendation list. The top-M hit ratio is the same as the recall. However, in this experiment, they are computed in different ways. They are both measured using the same training and testing set, however, the recall is computed based on the predictions on products that users have rated in the testing set. On the other hand, the top-M hit ratio uses the predictions of the model on all products that the user hasn't rated. To compute the top-M hit ratio, the recommender system will run through all of the products in the data set, it predicts the user rating on each product then take the top-M highest rated products for recommendation. Then using Equation 20, the top-M hit ratio is then computed. This explains why the top-M hit ratio is low compares to the recall computed in Section 4.1.2.2.1 even they are almost the same.

In this project, the top-M hit rate is measured for two different approaches to building the recommendation list. One approach applies the xQuAD method (controlled) to the recommendation list to reduce the popularity bias and the other doesn't (un-controlled). The result in of the top-M hit rate over M in the range of 50 to 200 (Figure 31).

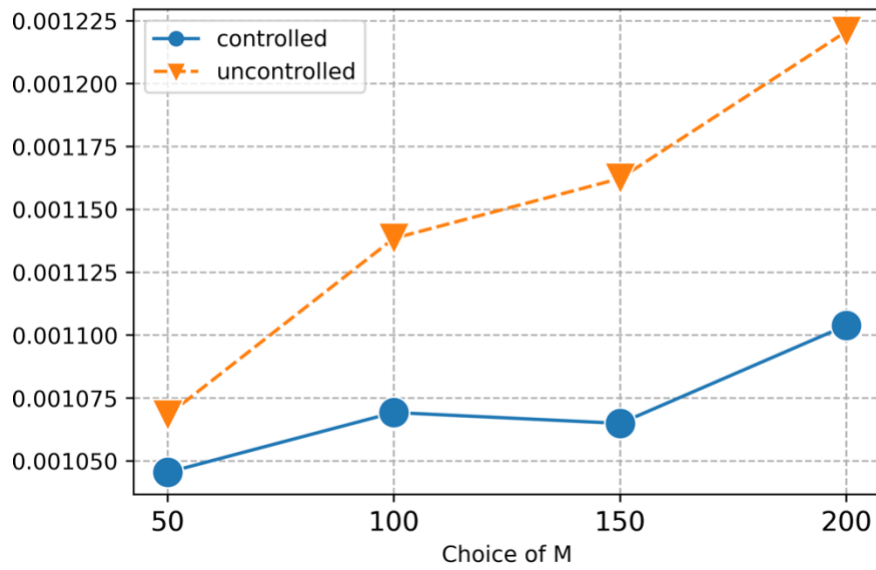


Figure 31. The results of top-M Hit Ratio with and without popularity bias control.

It is implied that the top-M hit ratio increases as M increases. This is because when increasing M, the nominator of the hit ratio also goes up while its denominator is fixed. In other words, the number of relevant products remains the same, but the number of relevant products appears in the recommendation list increases. As the xQuAD method decreases the accuracy of the recommendation list, the hit rate of the controlled version is smaller than the uncontrolled one. This reflects the popularity bias and ranking trade-off mentioned in Section 4.1.2.2.5.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. Conclusion

By integrating the recommender system into the web application, all the data from the users and their interaction are well used. A goal of building a light-weight E-commerce web application with an integrated recommender system is achieved.

The source code of the web application is carefully written, it is considered clean and maintainable. Using the key feature of ReactJS, components, it is now extremely easy for any developers to create a new web page using components created in this project. Since the RESTful API follows the typical Model-Routes-Controller coding structure of NodeJS and ExpressJS, making new routes (endpoints) is very straightforward. Overall, the web application supports most of the features of an E-commerce website and has a user-friendly, minimal interface that supplies a smooth shopping experience for every user. Besides, administrator pages are very useful for managing the recommender system, its dataset, and models.

For sampling and experimenting with the recommender system, the proposed method can overcome some limitations of usual recommender methods. First, experiments are done on many algorithms and the one that fits best is chosen, which was the SVD-based recommendation model. After that, to deal with the expensive computation of the SVD algorithm in the online stage, the SVD is upgraded by applying the Incremental SVD method. Then, to produce high-quality recommendations, the popularity bias of personalized recommendations is balanced while keeping an acceptable ranking accuracy (measured by NDCG) using the xQuAD algorithm. Finally, different metrics are applied to evaluate the effectiveness of the recommendation model on different aspects. As shown in the experiments, the proposed recommendation model can gain accuracy above 90% and include more popular items in the recommendation lists. This makes the recommender system achieve more effective performance.

The micro recommender system is carefully integrated into the web application and it is secured using the authentication route of the back-end server. The size of the recommender system is kept reasonable to prevent the project from becoming bulky and heavy. The

recommendation model and its dataset can always be renewed by administrators using the web interface.

Even though all the components of the application are kept as lightweight as possible, the major drawback of this application architecture is that it's required two different servers, one for the back-end and one for the recommender system which may be costly.

5.2. Future Work

A future optimization would be to combine the recommender system with the back-end server which reduces the bulkiness of the application. Besides, an automatic pipeline is going to be built for combining all of the methods and algorithms proposed in this project. Such pipeline can be used with flexibility to fasten the process of building a recommender system. Implicit data is also planned to be used along with the existing explicit data to build a more effective model.

PUBLICATIONS

1. Minh Quang Pham, Thi Thanh Sang Nguyen, Pham Minh Thu Do, Adrianna Kozierkiewicz (2020), “Incremental SVD-based Collaborative Filtering Enhanced with Diversity for Personalized Recommendation”, the Proceedings of the 12th International Conference on Computational Collective Intelligence, Da Nang, Vietnam. (Accepted)

REFERENCES

1. Lops, P., De Gemmis, M., Semeraro: Recommender System Handbook, Springer, Boston, (2011).
2. Dehkordi, L.F., Shahnazari, A. and Noroozi, A.: A study of the factors that influence the acceptance of e-commerce in developing countries: A comparative survey between Iran and United Arab Emirates. *Interdisciplinary Journal of Research in Business*, 1(6), (2011).
3. Zheng, W.: Recommendation System Based on Collaborative Filtering (2008).
4. Naumov, Maxim, et al.: Deep learning recommendation model for personalization and recommendation systems (2019). arXiv preprint arXiv:1906.00091
5. Xiong, Ruibin, et al.: Deep hybrid collaborative filtering for web service recommendation, *Expert systems with Applications*, 110, 191-205 (2018).
6. Jiang, Liaoliang, et al.: A trust-based collaborative filtering algorithm for E-commerce recommendation system, *Journal of Ambient Intelligence and Humanized Computing*, 10(8), 3023-3034 (2019).
7. Koren, Y.: Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In: 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, USA (2008).
8. J.L. Herlocker, J.A. Konstan, A. Borchers, J. Riedl: An algorithmic framework for performing collaborative filtering, *SIGIR Forum*, 51, 227–234 (2017).
9. B. Sarwar, G. Karypis, J. Konstan, J. Riedl: Item-based collaborative filtering recommendation algorithms, in: *Proc. 10th Int. Conf. World Wide Web*, ACM, New York, NY, USA, pp. 285–295 (2001).
10. Jannach, D., Lerche, L., Kamehkhosh, I., & Jugovac, M.: What recommenders recommend: An analysis of recommendation biases and possible countermeasures. *User Modeling and User-Adapted Interaction* (2015).
11. S. Cleger-Tamayo, J.M. Fernández-Luna, J.F. Huete: Top-n news recommendations in digital newspapers, *Knowl. Based Syst.* 27, 180–189 (2012).
12. Claypool, Mark, et al.: Combing content-based and collaborative filters in an online newspaper (1999).
13. Sarwar, Badrul, et al.: Analysis of recommendation algorithms for e-commerce, *Proceedings of the 2nd ACM conference on Electronic commerce* (2000).

14. Bell, R.M., Koren, Y., and Volinsky, C.: Matrix Factorization Techniques for Recommender System (2009).
15. Barathy, R., and P. Chitra: Applying Matrix Factorization in Collaborative Filtering Recommender Systems, 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), IEEE (2020).
16. Nilashi, Mehrbakhsh, Othman Ibrahim, and Karamollah Bagherifard: A recommender system based on collaborative filtering using ontology and dimensionality reduction techniques, *Expert Systems with Applications*, 92, 507-520 (2018).
17. Teodorescu, Oana Maria, Paul Stefan Popescu, and Marian Cristian Mihaescu: Taking e-assessment quizzes-a case study with an SVD based recommender system., *International Conference on Intelligent Data Engineering and Automated Learning*, Springer, Cham (2018).
18. Zarzour, Hafed, et al: A new collaborative filtering recommendation algorithm based on dimensionality reduction and clustering techniques, 2018 9th International Conference on Information and Communication Systems (ICICS), IEEE (2018).
19. Alter O, Brown PO, Botstein D.: Singular value decomposition for genome-wide expression data processing and modeling. *Proc Natl Acad Sci U S A*, 97, 10101-6 (2000).
20. Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan: Large-scale Parallel Collaborative Filtering for the Netflix Prize, *Proc. 4th Int'l Conf. Algorithmic Aspects in Information and Management*, LNCS 5034, Springer, 337–348 (2008).
21. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender System. In: *5th International Conference on Computer and Information Science* (2002).
22. Abdollahpouri, H., Burke, R., Mobasher, B.: Managing Popularity Bias in Recommender System with Personalized Re-ranking, (2019).
23. Marius, K., Derek, B.: Diversity, Serendipity, Novelty, and Coverage: A Survey and Empirical Analysis of Beyond-Accuracy Objectives in Recommender Systems, (December 2016).
24. Bergstra, J., Bengio, Y.: Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13, 281–305 (2012).

25. Malakhov, K. & Kurgaev, A.P. & Velychko, V.: Modern restful api dls and frameworks for restful web services api schema modeling, documenting, visualizing. PROBLEMS IN PROGRAMMING. 059-068. 10.15407/pp2018.04.059 (2018).
26. Jones, M., Bradley, J., Sakimura, N.: JSON Web Token. doi:10.17487/RFC7519. ISSN 2070-1721. RFC 7519 (May 2015).
27. JWT.IO: JSON Web Tokens Introduction, jwt.io. Retrieved June 2020.
28. Wang, Y., Wang, L., Li, Y., He, D., Liu, T., Chen, W.: A Theoretical Analysis of NDCG Type Ranking Measures. Journal of Machine Learning Research, 30 (2013).
29. Cypr de, L., Renke, P., Zheng, Wu., CS224W Final Report, Stanford (2015).

APPENDIX

Source code for re-ranking recommendation using xQuAD method.

```
53 def xquad(recommendation, user_profile, short_items, long_items, n_epochs=10, reg=.35, binary=False):
54     """ """
68     # In case there are not enough items in the recommendation list
69     n_epochs = min(len(recommendation), n_epochs)
70
71     final_result = []
72     S = []
73
74     # In order to select the next item to add to S,
75     # we compute a score for each item in R\S (in R not in S)
76     for n in range(n_epochs):
77         # For selecting the best score
78         best_score = 0
79         best_item = None
80
81         for r in recommendation:
82             v = r[0]
83             p_vu = r[1]
84
85             # v already in S, so skip it.
86             if v in S:
87                 continue
88
89             p_vu = (p_vu - 1.0) / (5.0 - 1.0)
90             score = (1 - reg) * p_vu
91
92             summation = 0
93
94             for d in (short_items, long_items):
95                 # Because P(v|c) is 0 if 'v' not in category 'd'
96                 if v not in d:
97                     summation += 0
98                     continue
99
100                 # The ratio of items in the user profile which belong to category 'd'
101                 p_du = sum([item in d for item in user_profile]) / len(user_profile)
102
103                 # Binary - equals to 1 when item 'i' in list 'S' already covers category 'd', 0 otherwise
104                 if binary:
105                     p_id_s = 1 if any([i in d for i in S]) else 0
106                 else:
107                     # Smooth - the ratio of items in list 'S' that covers category 'd'
108                     p_id_s = sum([i in d for i in S]) / len(S) if len(S) > 0 else 0
109
110                 # pi = (1 - P(i|d, S))
111                 pi = 1 - p_id_s
112
113                 summation += p_du * pi
114
115             score += reg * summation
116
117             if score > best_score:
118                 best_score = score
119                 best_item = v
120
121         S.append(best_item)
122         final_result.append((best_item, best_score))
123
124     return tuple(final_result)
```

Source code for partial fitting of Incremental SVD.

```
7 class ISVD(SVD):
8     def partial_fit(self, X):
9         X = X.copy()
10
11         u_ids = X['u_id'].unique().tolist()
12         i_ids = X['i_id'].unique().tolist()
13
14         n_user = len(self.user_dict)
15         n_item = len(self.item_dict)
16         n_new_user = 0
17         n_new_item = 0
18
19         for u_id in u_ids:
20             if u_id not in self.user_dict:
21                 self.user_dict[u_id] = n_user
22                 n_user += 1
23                 n_new_user += 1
24
25         for i_id in i_ids:
26             if i_id not in self.item_dict:
27                 self.item_dict[i_id] = n_item
28                 n_item += 1
29                 n_new_item += 1
30
31         X['u_id'] = X['u_id'].map(self.user_dict).astype(np.int32)
32         X['i_id'] = X['i_id'].map(self.item_dict).astype(np.int32)
33
34         # Start with the previous model
35         pu = self.pu
36         qi = self.qi
37         bu = self.bu
38         bi = self.bi
39
40         X = X.values
41
42         # Add rows for existed pu, qi, bu, bi
43         pu_ = np.random.normal(0, .1, (n_new_user, self.n_factors))
44         qi_ = np.random.normal(0, .1, (n_new_item, self.n_factors))
45         bu_ = np.zeros(n_new_user)
46         bi_ = np.zeros(n_new_item)
47
48         pu = np.concatenate((pu, pu_))
49         qi = np.concatenate((qi, qi_))
50         bu = np.concatenate((bu, bu_))
51         bi = np.concatenate((bi, bi_))
52
53         # Run SGD for 1 epoch
54         start = self._on_epoch_begin(0)
55         pu, qi, bu, bi = _run_epoch(X, pu, qi, bu, bi, self.global_mean, self.n_factors, self.lr, self.reg)
56         self._on_epoch_end(start)
57
58         self.pu = pu
59         self.qi = qi
60         self.bu = bu
61         self.bi = bi
```


Source code for Recommender System class.

```
1 import pandas as pd
2 from helper.modeling import load_model
3 from collections import defaultdict
4
5
6 class RecSys:
7     def __init__(self, model_path):
8         self.model, self.predictions, self.trainset = load_model(model_path)
9
10    def recommend(self, u_id, n=50):
11        # Get all products
12        all_items = self.get_full_dataset()['i_id'].unique()
13        user_profile = set(self.get_user_profile(u_id))
14
15        # Predict the user rating on each product
16        recommendations = []
17        for i_id in all_items:
18            if i_id in user_profile:
19                continue
20
21            prediction = self.model.predict_pair(u_id, i_id)
22            recommendations.append((i_id, prediction))
23
24        recommendations.sort(key=lambda x: x[1], reverse=True)
25        return recommendations[:n]
26
27    def get_full_dataset(self):
28        return pd.concat([self.trainset, self.predictions.drop(columns=['prediction'])])
29
30    def get_testset(self):
31        return self.predictions.drop(columns=['prediction'])
```

```

33 def get_top_n(self, n=10, from_test_set=False):
34     """Returns a dictionary of top-N recommendations if `from_test_set` is False,
35     otherwise top-N highest rated products, for each user in test set.
36     Args:
37         n (int): Number of maximum products for each user recommendation.
38         from_test_set (bool): Whether to get the top-N highest rated products.
39     """
40     # Get top n recommendation from test set
41     top_n = defaultdict(list)
42
43     for uid, iid, true_r, est in self.predictions.values:
44         score = true_r if from_test_set else est
45         top_n[uid].append((iid, score))
46
47     # Then sort the predictions for each user and retrieve the k highest ones.
48     for uid, user_ratings in top_n.items():
49         user_ratings.sort(key=lambda x: x[1], reverse=True)
50         top_n[uid] = user_ratings[:n]
51
52     return top_n
53
54 def get_short_head_and_long_tail_items(self, threshold):
55     """Returns the short head and long tail items as a tuple.
56     Args:
57         threshold (int): If user's number of ratings larger than the threshold, it is in short-head.
58         Else, it is in long-tail.
59     """
60     # using set for faster look-up time
61     dataset = self.get_full_dataset()
62     data_items = dataset.groupby(by='i_id').count().reset_index()
63
64     short_head_items = set(data_items[data_items['rating'] >= threshold]['i_id'].values)
65     long_tail_items = set(data_items[data_items['rating'] < threshold]['i_id'].values)
66
67     self.short_head = short_head_items
68     self.long_tail = long_tail_items
69
70     return short_head_items, long_tail_items
71
72 def get_user_profile(self, u_id):
73     return self.trainset[self.trainset['u_id'] == u_id]['i_id'].unique()
74

```