z5464669

Alok Thottakathu Prasannakumar

Fashion Recommender System on H&M Dataset

Contents

I. PROBLEM

Problem Definition

The recommendation problem is suggesting fashion items to users based on historical purchase data, purchases and contextual information of the H&M products. The goal is to enhance user experience on the H&M e-commerce platform by providing tailored product recommendations that increase user engagement and sales.

The recommender system would be integrated into a mobile or web application, making personalized recommendations accessible through various sections, such as the home page, product pages, and a dedicated recommendation section. The ultimate goal is if a user clicks for e.g. the "Shirt" sub-selection, only the shirt products will be filtered and shown from all the top recommendations that the recommendation model has produced. Also, for a new user, he will be guided through a questionnaire to learn about his preferences to build a seed user profile. That along, with the most popular products, and top selling products at his age bracket etc. and all such trends will be utilized to resolve the cold-start problem by showing him a different web or mobile page for initial recommendations, which he can rate at the get-go itself, for the recommender system to be able to curate more finer recommendations for him.

Competitor Analysis

Several commercial systems exist in the fashion recommendation domain.

We will look at 2 such commercial systems, Amazon and Stitch Fix:-

Amazon: Amazon's recommendation system has significantly evolved over the past two decades, combining collaborative filtering with content-based filtering to analyze user behaviour data—such as past purchases, items viewed, and search history—to predict and suggest products. Initially, Amazon used user-based collaborative filtering but shifted to item-based collaborative filtering, which improved scalability and recommendation quality by focusing on product correlations rather than user correlations.

So, when a customer buys an item, Amazon's system analyzes the purchase histories of other customers who bought the same item and identifies additional items frequently bought together. For example, if you buy a book, the system looks at what other customers who bought that book also purchased, recommending related books, accessories, or products.

This item-to-item approach is not only highly relevant but also scalable, requiring fewer computational resources compared to user-to-user comparisons. This could be because the number of items that a user bought will be lesser than the number of users who bought the same item or similar items. Over the years, Amazon has refined this system by incorporating personal preferences, such as favourite brands or styles, and adjusting recommendations based on the timing of past purchases. For instance, the system might suggest reordering household essentials after a typical usage period. That is, it keeps track of users history, and might predict what a user might need next.

Amazon has increasingly used neural networks to sort input data chronologically which has improved the system's ability to predict short-term preferences. For example, this method helps recommend movies that a user is likely to watch in the next one to two weeks, enhancing the relevance of suggestions.

Strengths:

- High scalability
- Real-time recommendations

Weaknesses:

- Cold start problem
- Potential overfitting

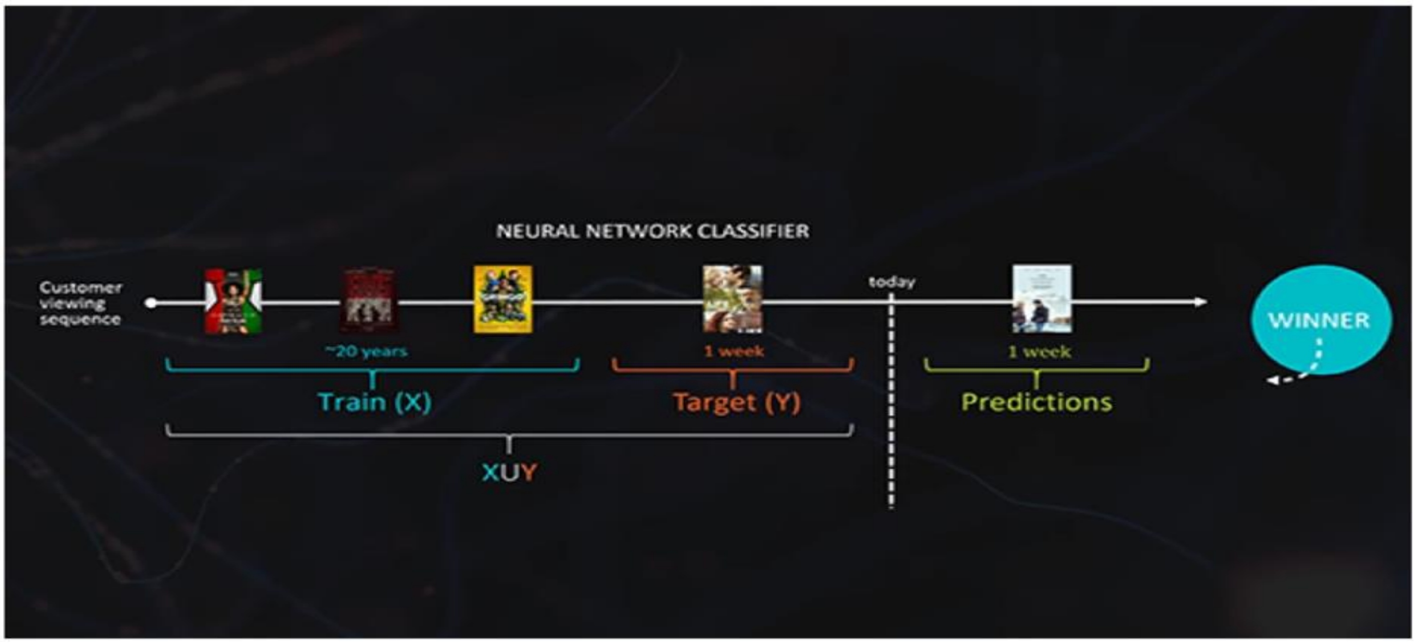


Fig 1: *Amazon Recommendation System Workflow* - Amazon researchers found that using neural networks to generate movie recommendations worked much better when they sorted the input data chronologically and used it to predict future movie preferences over a short (one- to two-week) period.

Stitch Fix: Stitch Fix combines human stylists with machine learning algorithms to offer personalized styling services. Their system, known as the "Stitch Fix algorithm," combines data science and human stylists to curate individualized clothing selections for customers.

Their system leverages data science and stylist feedback to curate selections, using collaborative filtering, content-based filtering, and A/B testing. This hybrid approach ensures nuanced, accurate recommendations but can be resource-intensive. The robust data platform supports various processes like warehouse assignment, logistics optimization, and inventory management, enabling efficient operations. By blending algorithms with human insights, Stitch Fix offers a unique, highly personalized shopping experience.

Clients fill out a Style Profile, and an algorithm schedules deliveries. A warehouse assignment algorithm then selects the best warehouse based on cost and inventory match. Machines generate a ranked list of potential items, filtering out previously disliked ones. Human stylists refine these selections, adding personalized notes. Client feedback further refines future recommendations. This system combines data science and human insights for personalized, accurate clothing recommendations.

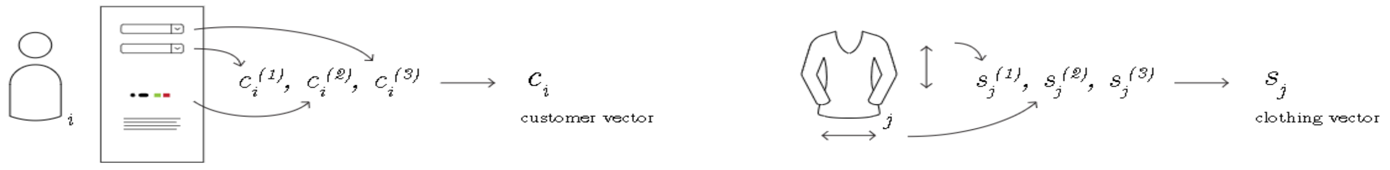
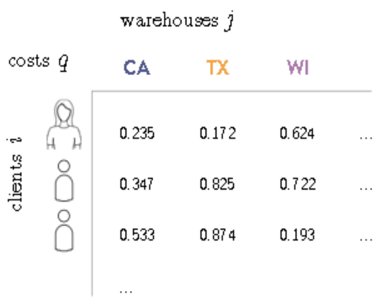

i
$c_i^{(1)}, c_i^{(2)}, c_i^{(3)} \longrightarrow c_i$
customer vector
j
$s_j^{(1)}, s_j^{(2)}, s_j^{(3)} \longrightarrow s_j$
clothing vector

Fig 2: *Customer and Item Vectors*

Customer and Clothing vectors are created from various attributes, which are then used to match clients with the best-fitting items.

Machines generate rank-ordered lists of inventory, filtering out previously received or disliked items, and various algorithms evaluate the likelihood of a client loving a particular style using collaborative filtering, mixed-effects modeling, and latent feature analysis.



	CA	TX	WI	
clients i	0.235	0.172	0.624	...
	0.347	0.825	0.722	...
	0.533	0.874	0.193	...
	...			

warehouses j
costs q

Fig 3: Warehouse Cost Matrix

Cost matrix is used for warehouse assignment, where algorithms calculate the cost of fulfilling a client's order from different warehouses. After machine ranking, human stylists finalize the selections, and algorithms match clients with suitable stylists, optimizing the fit based on historical data and preferences. Stylists use a custom interface to understand clients and make final selections, writing personal notes for each shipment. This blend of machine precision and human intuition ensures a highly personalized service.

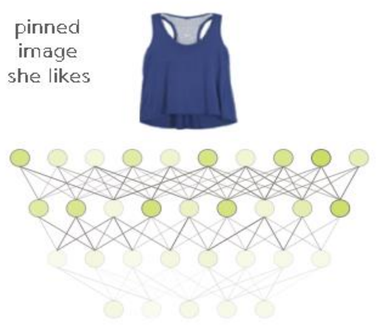
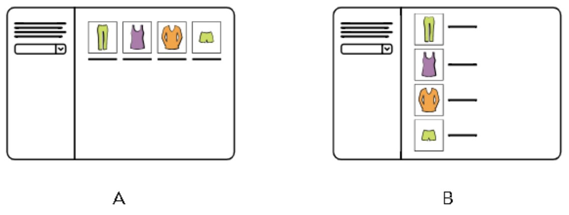


Fig 4: Styling Interface

The system uses machine learning to find items similar to those a client has pinned or liked, enhancing the relevance of recommendations. The system optimizes warehouse pick-path routing and shipment groupings for efficient delivery. After receiving a shipment, clients provide feedback, which improves future recommendations, refining both individual and overall recommendation accuracy. The system also manages inventory, anticipates demand, and develops new styles, using techniques like genetic algorithms to create new clothing designs tailored to client segments.



$$P(\text{correct\_prediction}) = \beta \, \text{ex\_cell} + a_1 P(1 \mid \text{stylist}) + a_2 P(1 \mid \text{client})$$

Fig 5: Styling Interface

An interface is used by stylists to finalize selections and write personalized notes, combining algorithmic recommendations with human insight.

By leveraging both machine learning and human insights, Stitch Fix provides highly personalized recommendations, making it a unique player in the fashion recommendation space.

Strengths:

- Personalized styling
- Human-AI collaboration

Weaknesses:

- Resource-intensive
- Scalability challenges

Proposed Recommender System on H&M Dataset

Our proposed recommender system aims to address these gaps by leveraging user demographics (age, gender, location), purchase history, and browsing behavior. It will incorporate contextual information like the current season and trending styles, providing:

- Personalized product suggestions based on user preferences and behavior.
- Context-aware recommendations considering current trends and seasons.
- Cold start item recommendations based on the most selling products.

User Inputs

For our proposed recommender system, the primary user inputs include:

1. **Transaction History**: This includes the items that a user has previously purchased, viewed, or added to their wishlist.
2. **User Demographics**: Information such as age, gender, location, and membership status.
3. **User Interactions**: Clicks, search queries, and browsing history.
4. **User Reviews**: Explicitly stated preferences or feedback on items (e.g., ratings, likes/dislikes).

However, only Transaction History and User Demographics are available in the dataset. We can enhance the model in future by integrating User Interactions and User Reviews also.

Recommendations Provided

The system will generate personalized recommendations that can include:

1. **Product Recommendations**: Items similar to those the user has interacted with, considering both item features and user preferences.
2. **Seasonal Recommendations**: Items relevant to the current season or upcoming events.
3. **Trending Items**: Popular items among similar users or within the user's demographic group.
4. **Complementary Items**: Products that complement recent purchases (e.g., accessories to match a recently bought dress).

Timing of Recommendations Recommendations will be provided at multiple touchpoints:

1. **Homepage**: Personalized recommendations when the user logs in.
2. **Product Pages**: Suggestions of similar or complementary items.
3. **Checkout Page**: Recommendations of related items just before the user completes a purchase.
4. **Emails and Notifications**: Personalized emails or app notifications based on user activity and preferences.

User Feedback and Its Utilization

User feedback will play a crucial role in refining and improving the recommendations. Ideally, when we get the full dataset from H&M, we plan to include all of the following types of feedback, and their uses include:

1. **Ratings and Reviews**: Users can rate and review products. This feedback helps adjust the recommendation algorithms to prioritize highly-rated items.
2. **Likes and Dislikes**: Users can like or dislike items. This explicit feedback directly influences the recommendation engine to tailor future suggestions more accurately.
3. **Purchase and Return Data**: Analyzing the items that users purchase and return helps in understanding user preferences better. High return rates for recommended items would signal a need for the system to reassess the user's preferences.
4. **Interaction Data**: Tracking which recommendations users engage with (e.g., clicks, add to cart) helps in refining the recommendation algorithms to focus on items that are more likely to interest the user.

User feedback, both explicit (ratings, likes/dislikes) and implicit (click-through rates, purchase decisions), will continuously update and refine the recommendation models. This ensures the system adapts to changing user preferences over time.

Our recommendation problem primarily functions as a ranking problem but also integrates elements of classification and prediction.

Ranking Problem

The main goal is to rank items by their relevance for each user, based on their past interactions, preferences, and the behavior of similar users, thereby ensuring the most relevant items appear at the top of the recommendation list. This enhances the user experience by presenting the items they are most likely to interact with first.

Prediction/Estimation Problem

Using collaborative filtering methods, such as Singular Value Decomposition (SVD), we predict user preference scores (rating or implicit feedback score) for items. These predicted scores are used to rank items, estimating how much a user would like an item they haven't interacted with yet. This involves estimating the missing entries in the user-item interaction matrix, which represents how much a user would like an item they have not yet interacted with.

Classification Problem

Certain aspects, like determining whether a user will interact with an item, can be seen as a binary classification task (interaction vs. no interaction). This involves classifying items as relevant or not based on user preferences.

Fig 6: Mobile and Web User Interface Mockups

To illustrate the intended usage of the system, consider the above user interface mockups. On the home page, a personalized section could display top picks based on user preferences. On product pages, users might see recommendations for similar items, complementary products, and suggestions based on what others also bought. A dedicated recommendation section could provide a personalized feed of products, tailored to user behavior and contextual information.

Our system will be integrated into the e-commerce platform, providing recommendations in real-time. User feedback mechanisms will continuously improve the system, ensuring it remains adaptive and relevant.

By leveraging user demographics, contextual information, and sophisticated recommendation algorithms, our proposed system aims to deliver a comprehensive, user-centric solution that enhances the shopping experience on the e-commerce platform. This approach not only addresses the limitations of existing systems but also ensures scalability and efficiency, making it a robust choice for personalized recommendations.

II. DATASET

For our recommendation system, we used the H&M Personalized Fashion Recommendations dataset from Kaggle. This rich dataset includes:

1. **Transactions:** Records of customer purchases, including customer_id, article_id, purchase date, price, and sales_channel_id. There are 31,788,324 transactions in the dataset.
2. **Customers:** Demographic information such as customer_id, age, club_member_status, fashion_news_frequency, and postal_code. There are 1,371,980 customers mentioned in dataset.
3. **Articles:** Product details including article_id, product_code, product_type_no, graphical_appearance_no, color_group_code, perceived_color_master_id, department_no, index_group_no, section_no, garment_group_no, and product descriptions. There are 105542 articles.

These tables are linked by unique identifiers: customer_id in Transactions and Customers tables, and article_id in Transactions and Articles tables. This linkage allows us to combine customer information, transaction history, and product details to create a comprehensive dataset for our recommendation system.

Exploratory Data Analysis (EDA)

Before building the model, we conducted exploratory data analysis to understand the dataset's structure, distribution, and potential issues.

1. **Transactions Analysis:**
   - **Volume and Frequency:** We looked at the number of transactions over time to identify purchasing patterns and trends.
   - **Price Distribution:** We examined how prices varied to understand user spending behavior.
2. **Customers Analysis:**
   - **Demographics:** We explored age distribution, club member status, and fashion news subscription status to understand our customer base.
   - **Transaction Behavior:** We linked demographics with transaction history to spot purchasing patterns across different customer segments.
3. **Articles Analysis:**
   - **Product Characteristics:** We analyzed the distribution of product types, colors, and other attributes to understand the variety and frequency of different product features.
   - **Popularity:** We identified the most popular products and categories based on the number of purchases.

Exploratory Data Analysis

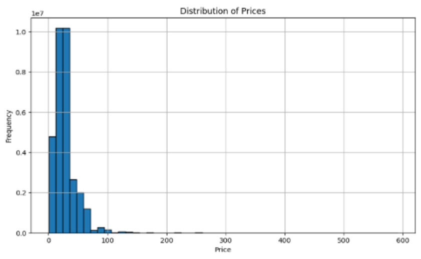


Fig 7: Most frequently bought items are in the range of 10 to 50 dollars.

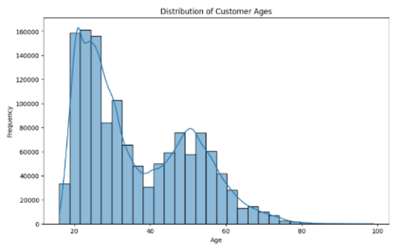


Fig 8: Age bracket 20 to 30 are the most frequent purchasers.

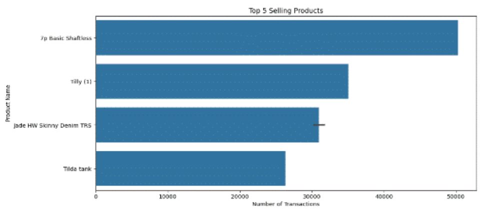


Fig 9: These most selling products can be used for cold start. 7p Basic Shaftless is socks.

All these trends and ideas in the plots in the Exploratory Data Analysis are going to be implemented in the model and the cold start problem. For e.g., I can create age brackets to target customers of specific age brackets effectively.

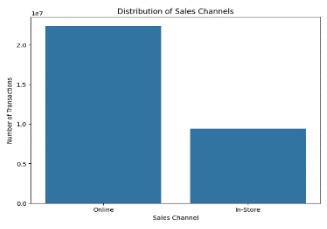


Fig 10: Most sales are online. Maybe because of Covid.

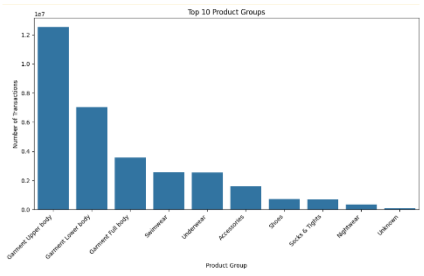


Fig 11: The garment clothes are the most selling product groups.

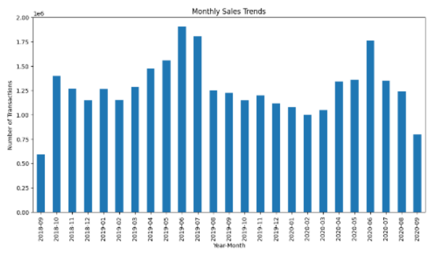


Fig 12: June month has more sales because of digitalization and lower markdowns on prices of products.

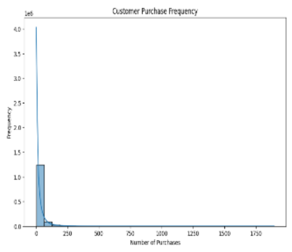


Fig 13: Most customers purchase less than 50 times in 1½ year period.

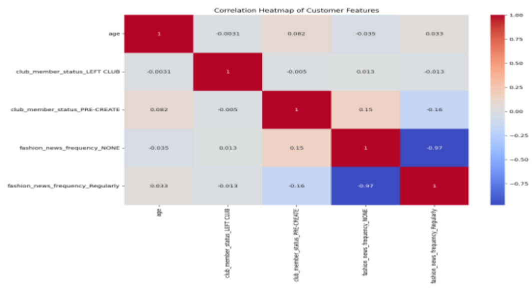


Fig 14: Age weakly influences club departure and account creation. Customers less likely to subscribe to fashion news.

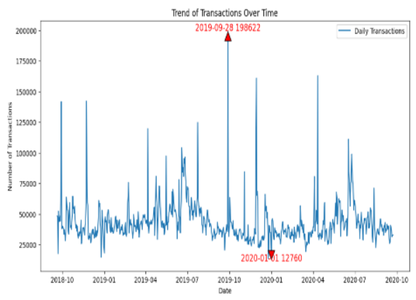


Fig 15: 28/9/19 – Black Friday Sale 1/1/20 – People spend less after Christmas holidays. No money.

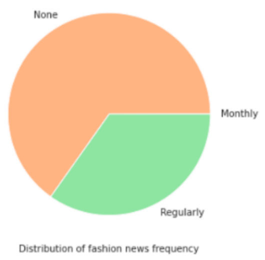


Fig 16: Vast Majority don't subscribe to fashion news.

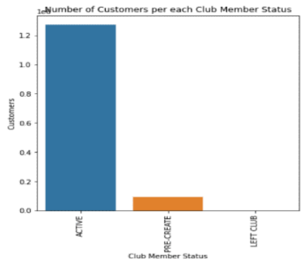


Fig 17: Most customers remain as members. Very few new customers join.

H&M can reduce the stock or inventory in January Month. H&M should focus on advertising more, because the rate of new customers joining H&M is less. It could be because of poor reputation stemming from poor quality of clothes, some negative publicity etc. However, H&M is very good at retaining its members. That shows that the service quality or the ease and accessibility of the website / app , and its lower priced products are a hit among people. Young people form the bulk of the H&M customers, because these are people who usually wear fast fashion trends, and prefer cheaper clothes. H&M should target middle age and older people by introducing better quality all-season wear, which lasts longer.

Strengths and Weaknesses of the Dataset

Strengths:

- **Comprehensive:** The dataset covers a wide range of customer demographics, transaction history, and product details.
- **Rich Features:** It includes numerical and categorical features, along with product descriptions.
- **Temporal Data:** The transaction date allows for temporal analysis and the incorporation of seasonal trends into the recommendation system.

Weaknesses:

- **Sparse Data:** The interaction data is sparse, with many customers having limited transaction history, leading to cold start issues.
- **Imbalanced Data:** Some products and customer segments are overrepresented, potentially biasing the model.
- **Missing Values:** Some demographic fields like age and postal_code have missing values, which will skew the model.
- **No Explicit Ratings/User Reviews:** The dataset lacks explicit user reviews and ratings, which are crucial for understanding user preferences. We relied on implicit feedback, such as purchase history and interaction data.

Using Subset of Original Dataset

Given the dataset's size, we worked with smaller subsets during development and testing. For example, we used a subset involving around 7689 users to streamline the model training process and reduce computational overhead. Our laptops are not powerful enough to train on the whole H&M dataset. This subset was chosen to maintain a representative sample of the overall dataset, ensuring all customer segments and product categories were proportionally included.

I did Exploratory Data Analysis on the sampled dataset having 7689 unique customers.

```
Total number of transactions in the sampled dataset: 601370
Total number of unique articles in the sampled dataset: 61368
```

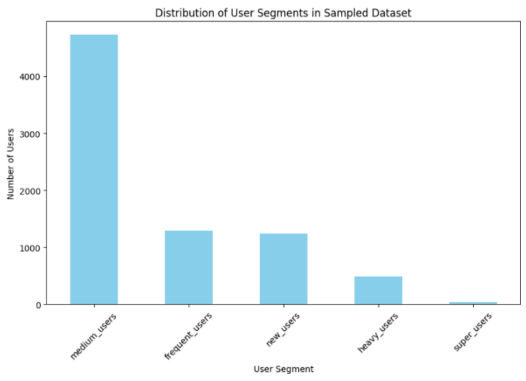


Fig 18: I then segmented the customers into different user segments. Users are segmented into different categories based on their activity levels: new users (0-20 transactions), medium users (21-100 transactions), frequent users (101-200 transactions), heavy users (201-500 transactions), and super users (501+ transactions).

```
user_segment
medium_users      4726
frequent_users    1289
new_users         1245
heavy_users        490
super_users         34
```

Fig 19: We get the proportion of users in each user segment in the sampled dataset. Most prominent user segment is medium users (21-100 transactions).

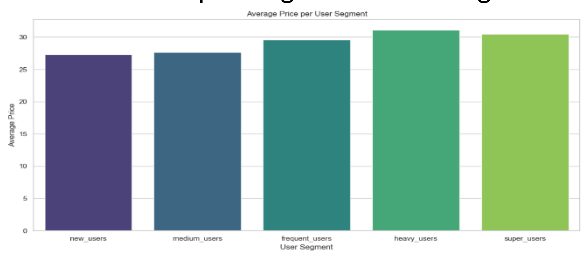
Next, I proceeded to do Segmentation Analysis on this sampled dataset. So, I can create a matrix combining the user ranges and club membership statuses. Then I can analyze the behavior of each segment, such as purchase frequency, average spending, and types of products bought.

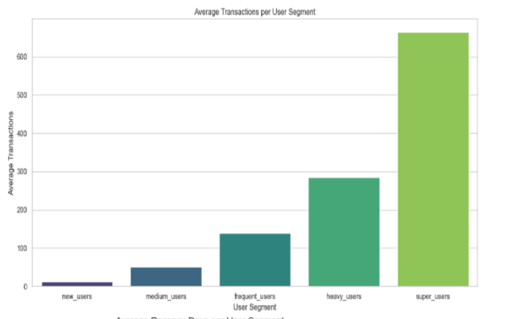
For Data Visualization, I can use Bar Charts to display the counts of each user segment, which I have shown above.

I can use Heatmaps or Cluster Plots to show the average spending or number of transactions for each segment combination.

I can use Box Plots to illustrate the distribution of spending for different segments.

```
     user_segment  avg_transactions  avg_recency_days  avg_price
0       new_users         13.065863          2.962249  27.354222
1    medium_users         51.565806          2.953026  27.639697
2  frequent_users        139.279286          2.847944  29.557799
3     heavy_users        284.242857          2.759184  31.152415
4     super_users        664.500000          2.764706  30.500454
```





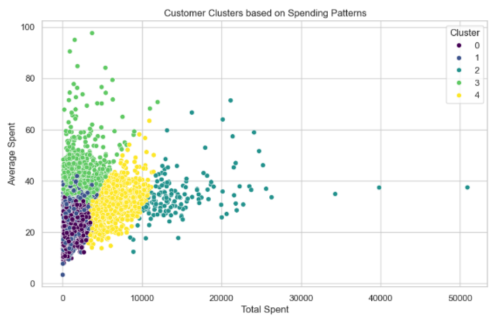
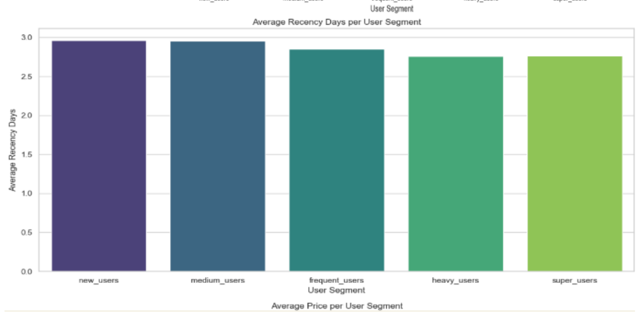


Fig 20: Different plots of the user Segments. I have checked the average transactions, average recency days and average price of each of these user segments. Also, K-means clustering with k = 5 tells about spending patterns of 5 age brackets.



The above plots show segmentation Analysis done on the sampled dataset of 7689 users and the different plots analyzing it.

The k-means clusters are based on the following features:-

- **Total Spent:** The total amount of money a customer has spent.
- **Average Spent:** The average amount of money a customer spends per transaction.
- **Transaction Count:** The total number of transactions made by the customer.
- **Age:** The age of the customer.

The K-Means algorithm works by minimizing the variance within each cluster, i.e. it groups customers who have similar spending habits and ages. It aims to make each cluster of customers as similar as possible within the group. This analysis helps us identify different types of customers, which can be useful for creating targeted marketing strategies, offering personalized deals, and improving overall customer service.

Key Insights:

- As user engagement increases (from new to super users), the average number of transactions significantly increases, highlighting a clear correlation between user segment and activity level.
- Average recency days are consistently low across all segments, indicating frequent interactions with the platform across all user types.
- Higher engagement levels (frequent, heavy, and super users) are associated with higher average spending per transaction, suggesting that more active users are also more valuable in terms of revenue per transaction.
- **From the clusters graph, we see that people are willing to spend usually atmost 75 dollars for a product**. So, we can recommend products below that price threshold for cold-start i.e. new customers.

Based on these insights of the Segmentation Analysis, we can target customers in the following ways: -
New Users: Offer discounts or introductory offers to encourage more purchases.
Medium Users: Provide personalized recommendations based on past purchases.
Frequent Users: Offer loyalty rewards or exclusive access to new products.
Heavy Users: Provide premium services or exclusive membership benefits.
Super Users: Offer high-value items or special recognition programs.
Membership Status Specific Strategies:

Active: Regularly update them with new offers and products.
Pre-create: Encourage them to complete their profiles with special offers.
Left Club: Win-back campaigns with exclusive deals or incentives to rejoin.
Combination-Based Recommendations:

We can create tailored marketing strategies for each combination of user range and membership status.
New Users + Active: Focus on onboarding and retention.
Frequent Users + Left Club: Offer compelling reasons to rejoin and stay active.

Integration with Visual Data

The dataset also includes an extra images folder with 105100 images, containing images of most of the fashion items. These images were used to enhance the recommendation model by displaying recommended items visually in the user interface, improving the user experience by allowing customers to see the items recommended to them.

III. METHOD(S)

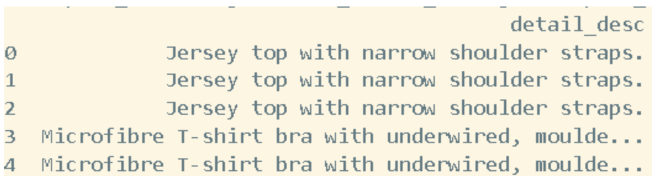
I am implementing 2 methods – collaborative filtering and content based. However, for those methods, I need numerical matrix. i.e. All the data must be converted to numerical value, as input in the models. This ensures that the models perform best, and speeds up the operation also.

Preprocessing

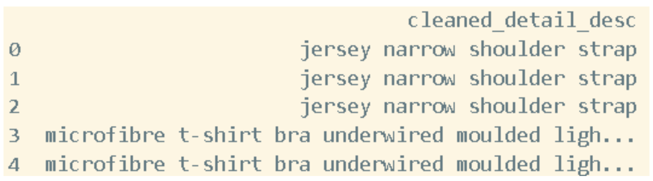
Initially, we extracted features from the dataset to prepare it for building our recommendation system. Numerical features included price and Word2Vec features derived from product descriptions, while categorical features were one-hot encoded. These categorical features included attributes such as product type, color, and department.

We performed the following steps for preprocessing:

For the preprocessing of our dataset, we undertook a series of systematic steps to ensure that the data was clean, standardized, and ready for the subsequent stages of feature extraction and model building. Initially, we performed text preprocessing, which involved removing punctuation and digits to clean up the text data. This step helped standardize the text, making it easier to process further. We also removed stop words using a combination of stop words from NLTK and scikit-learn, filtering out common words that do not significantly contribute to the meaning of the text. Additionally, we applied lemmatization to convert words to their base forms, reducing variability in the text data.



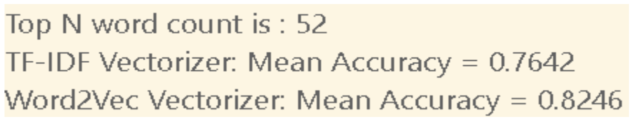
```
                                          detail_desc
0            Jersey top with narrow shoulder straps.
1            Jersey top with narrow shoulder straps.
2            Jersey top with narrow shoulder straps.
3  Microfibre T-shirt bra with underwired, moulde...
4  Microfibre T-shirt bra with underwired, moulde...
```



```
                                 cleaned_detail_desc
0                         jersey narrow shoulder strap
1                         jersey narrow shoulder strap
2                         jersey narrow shoulder strap
3  microfibre t-shirt bra underwired moulded ligh...
4  microfibre t-shirt bra underwired moulded ligh...
```

Fig 21: Before preprocessing, stopword removal and lemmatization, and after these steps.

In the feature extraction phase, we focused on both numerical and categorical features. For choosing Vectorizer to change product descriptions to numerical vector, I tried both TFIDF Vectorizer and Word2Vec using a small validation set I took from the training dataset. I found the Top N Word Count for Vocabulary is 52, and the accuracy of Word2Vec being better than TDIDF Vectorizer, most probably, because Word2Vec takes semantic association of words into account unlike TDIDF Vectorizer. So, I used Word2Vec for this project.



```
Top N word count is : 52
TF-IDF Vectorizer: Mean Accuracy = 0.7642
Word2Vec Vectorizer: Mean Accuracy = 0.8246
```

Fig 22: Top N Word Count and Mean Accuracy of TF-IDF Vectorizer when applied on the training dataset.

Numerical features included the price and Word2Vec features of product descriptions. Word2Vec features were generated by training a Word2Vec model on the 'cleaned_detail_desc' and 'prod_name' columns. For categorical features, attributes such as product type, color, and department were one-hot encoded using scikit-learn's OneHotEncoder. This conversion was necessary to handle categorical data effectively in machine learning algorithms.

We then merged the transactions with articles and customers to create comprehensive train and test datasets. This merging ensured that all necessary information was included for building and evaluating the recommendation models.

We then proceeded to drop unnecessary columns such as 'graphical_appearance_no', 'sales_channel_id', 'product_type_no', 'index_group_no', 'product_code', 'colour_group_code', 'perceived_colour_master_id', and 'section_no' as they were either redundant or not relevant for the final model.

To ensure consistency and improve model performance, we normalized numerical features like price, age, total_spent, and avg_spent. We also extracted temporal features such as month, quarter, and day of the week from the transaction date to capture seasonal and temporal trends, and subsequently removed the original datetime columns to simplify the dataset. Additional preprocessing steps included hashing and clustering postal codes to reduce dimensionality while preserving geographic information, and categorizing customers into financial brackets based on their total spending. This categorization helped in segmenting customers for more personalized recommendations.

Throughout these preprocessing steps, we generated several outputs and plots to verify the transformations and ensure the integrity of the data. For instance, we visualized the distribution of prices and total spending to understand the range and variability within these numerical features.

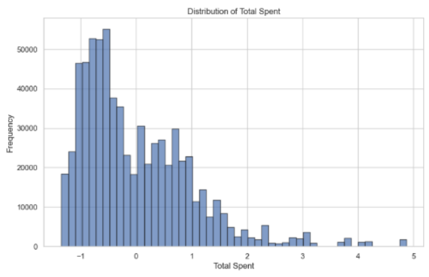
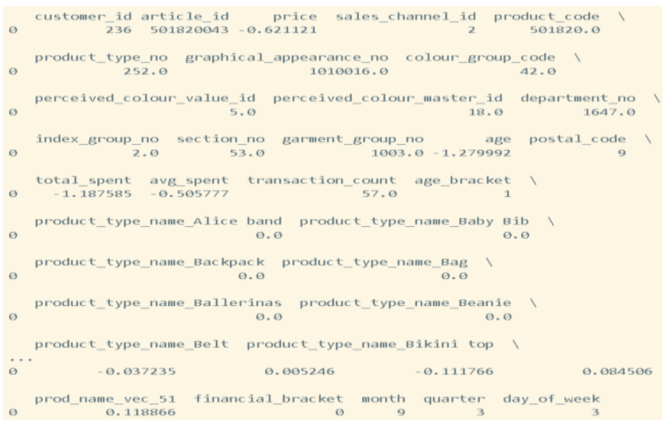


Fig 23: Plot of total spent and frequency of such among customers.

Additionally, we plotted the frequency of categorical features like product types and colors to observe their distribution across the dataset. These visualizations provided valuable insights and confirmed the effectiveness of our preprocessing techniques, setting a solid foundation for building robust recommendation models.



```
   customer_id article_id     price  sales_channel_id  product_code  \
0          236  501820043 -0.621121                 2      501820.0

   product_type_no  graphical_appearance_no  colour_group_code  \
0            252.0                1010016.0               42.0

   perceived_colour_value_id  perceived_colour_master_id  department_no  \
0                        5.0                        18.0         1647.0

   index_group_no  section_no  garment_group_no       age  postal_code  \
0             2.0        53.0            1003.0 -1.279992            9

   total_spent  avg_spent  transaction_count  age_bracket  \
0    -1.187585  -0.505777               57.0            1

   product_type_name_Alice band  product_type_name_Baby Bib  \
0                           0.0                         0.0

   product_type_name_Backpack  product_type_name_Bag  \
0                         0.0                    0.0

   product_type_name_Ballerinas  product_type_name_Beanie  \
0                           0.0                       0.0

   product_type_name_Belt  product_type_name_Bikini top  \
...
0          -0.037235             0.005246            -0.111766            0.084506

   prod_name_vec_51  financial_bracket  month  quarter  day_of_week
0          0.118866                  0      9        3            3
```

Fig 24: A view of the first row of the training dataset.

All the steps mentioned above were done on the test dataset Also to ensure being able to compute similarity later during model execution and evaluation. The final goal of all these steps is to ensure that the whole training and test dataset is converted into a numerical matrix to be able to be used in the machine learning models. We can see from this image that all data are numerical after these steps.

The python libraries and functions I used for preprocessing are StandardScaler, timedelta, re for regular expressions, nltk and sklearn for stopwords and lemmatization, sklearn cluster for k-means Clustering, seaborn for heatmap, matplotlib.pyplot for plotting, OneHotEncoder of sklearn, Word2Vec of genism models, numpy and pandas.

All preprocessing steps are over, and now we move on to model.

The approach to the recommendation problem is about developing a method for content-based filtering, and a method for collaborative filtering. I **could not** implement a hybrid approach to combine elements of both, because of the limitations of my hardware. I was already getting memory and VSCode limitation errors when I was implementing both methods individually. In future, I would like to ensemble both methods to get better recommendations.

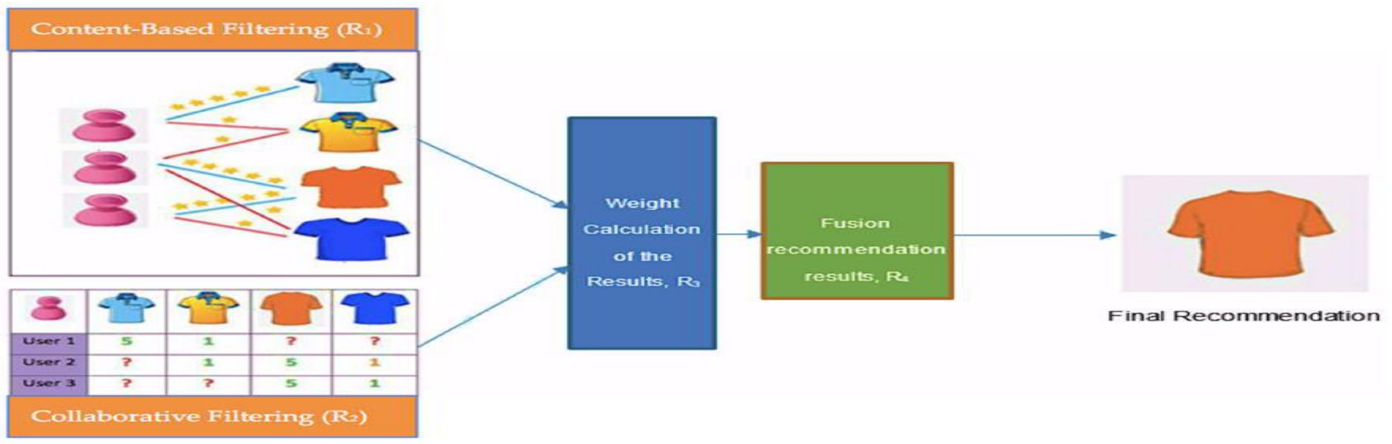

Content-Based Filtering (R₁)
User 1 5 1 ? ?
User 2 ? 1 5 1
User 3 ? ? 5 1
Collaborative Filtering (R₂)
Weight Calculation of the Results, R₃
Fusion recommendation results, R₄
Final Recommendation

Fig 25: If I had implemented hybrid model, it would have been a weighted ensemble of both base models.

The choice of methods was justified as follows: Content-based filtering leverages detailed item features and user profiles to generate recommendations based on similarities. Collaborative filtering utilizes past interactions to predict user preferences. This method effectively captures latent factors that influence user-item interactions but may suffer from the cold-start problem for new users and items.

(a) Collaborative Filtering using SVD and User-Item Interaction Matrix :-

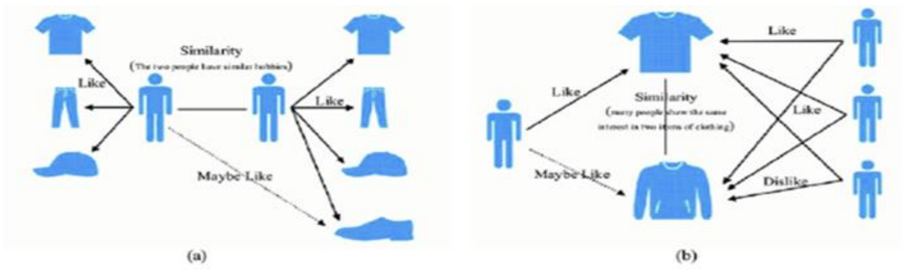
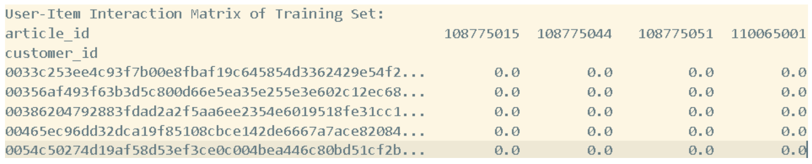
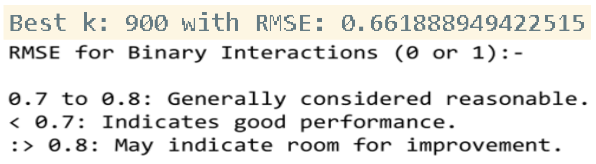

Similarity
(The two people have similar hobbies)
Like
Like
Maybe Like
(a)
Like
Similarity
(many people show the same interest in two items of clothing)
Like
Like
Maybe Like
Dislike
(b)

Fig 26: Instead of computing user based similarity and item based similarity separately as shown above, I tried to get the benefits of both in a simpler model using Collaborative Filtering on User-Item Interaction Matrix and SVD

For the collaborative filtering approach, we used Singular Value Decomposition (SVD) on the user-item interaction matrix. This method leverages the past interactions of users to predict their preferences for unseen items. Since, we split the Sampled dataset data into train and test based on a time window and threshold date, we will have older purchase or transaction history of users in train data, and newer purchase history in test data. We use the purchase history in train data, and through collaborative filtering approach using SVD, we try to ascertain the user-based similarity or item-based similarity intuitively using the user-item interaction matrix, rather than trying to ascertain user-based similarity separately and item-based similarity separately. I have done it like this to reduce processing time. In the user-item interaction matrix, the rows represent users and columns represents items, with values (0 or 1) at the intersection i.e. a cell, indicating interactions, meaning 0 for user not purchased that item, and 1 for user has purchased that item.



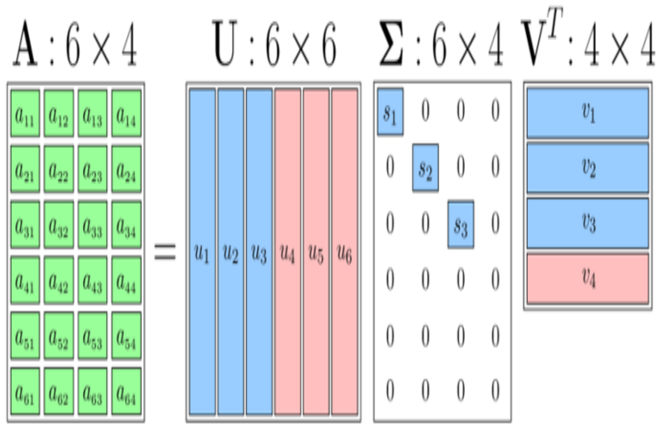
```
User-Item Interaction Matrix of Training Set:
article_id                                           108775015  108775044  108775051  110065001
customer_id
0033c253ee4c93f7b00e8fbaf19c645854d3362429e54f2...        0.0        0.0        0.0        0.0
00356af493f63b3d5c800d66e5ea35e255e3e602c12ec68...        0.0        0.0        0.0        0.0
00386204792883fdad2a2f5aa6ee2354e6019518fe31cc1...        0.0        0.0        0.0        0.0
00465ec96dd32dca19f85108cbce142de6667a7ace82084...        0.0        0.0        0.0        0.0
0054c50274d19af58d53ef3ce0c004bea446c80bd51cf2b...        0.0        0.0        0.0        0.0
```



```
Best k: 900 with RMSE: 0.661888949422515
RMSE for Binary Interactions (0 or 1):-

0.7 to 0.8: Generally considered reasonable.
< 0.7: Indicates good performance.
:> 0.8: May indicate room for improvement.
```

Fig 27: A view of the User-Item Interaction Matrix, and the considerations for the number of latent factors


$A:6\times4$
$U:6\times6$
$\Sigma:6\times4$
$V^T:4\times4$
a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24}
a_{31} a_{32} a_{33} a_{34}
a_{41} a_{42} a_{43} a_{44}
a_{51} a_{52} a_{53} a_{54}
a_{61} a_{62} a_{63} a_{64}
=
u_1 u_2 u_3 u_4 u_5 u_6
s_1 s_2 s_3
0
v_1 v_2 v_3 v_4

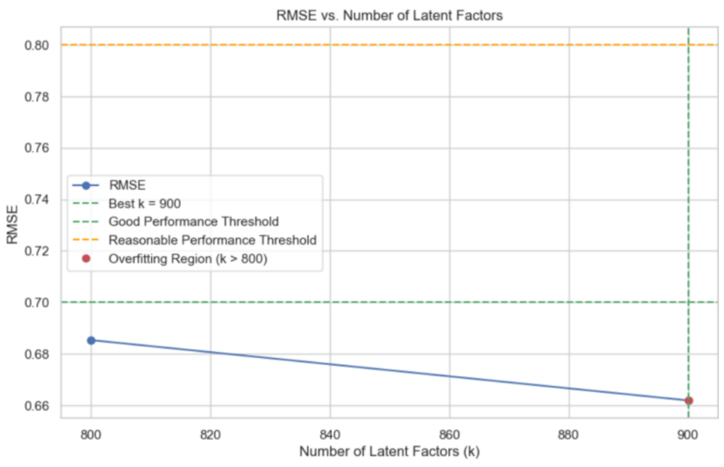


Fig 28: A view of how SVD deconstructs matrix

Fig 29: Optimum value of k for SVD is 900

I did hyperparameter tuning to identify the optimal number of latent factors k for our model, testing various values. I found that k = 900 is the optimum value because it provides a good balance between model performance and complexity. Since purchase history here is binary values i.e. 0 or 1, and for k = 900, the RMSE for binary interactions (0 or 1) falls within the range of < 0.7, which is generally considered reasonable. Choosing k = 900 helps to avoid the pitfalls of overfitting and excessive time complexity associated with values greater than 900.

k = 900 means that the decomposition keeps the top 900 components (or features) that capture the most significant patterns in the data. Essentially, it reduces the dimensionality of the original matrix to a lower-dimensional space with 900 latent factors.

SVD was performed on the user-item interaction matrix to decompose it into three matrices: **U** (user-feature matrix), **Σ** (diagonal matrix of singular values i.e. eigen values), and V^T (item-feature matrix). These matrices represent the latent factors underlying the interactions between users and items. The interaction matrix was then **reconstructed** using the formula $\mathbf{U} \times \mathbf{\Sigma} \times \mathbf{V}^T$, to predict unseen interactions. This reconstruction helps in approximating the original user-item matrix with reduced dimensionality, capturing the essential patterns in user-item interactions while discarding noise. For each user, the top N items with the highest predicted interaction scores were recommended.

I didn't use any regularization parameter, as the model is definitely not overfitting, and also my hardware was struggling, and I couldn't do hyper-parameterization too much. The user-item interaction matrix was sparse, which attributed to poor performance of the model. I felt the model was underfitting or under-recommending or that it didn't give me relevant or diverse recommendations, which I have shown in Evaluation Subsection. Hence, to get some accuracy, I had to increase the number of latent factors from normally used values like 100 or 150 to 900, which increased the time complexity of the SVD model a lot. Hence, I couldn't afford hyperparameter tuning.

I had tested upto showing 40 recommendations, and saw some improvement in the evaluation metrics. However, for the sake of having same testing conditions as for everyone in my group, I have submitted only top 10 recommendations.

The python libraries / functions I used are csr_matrix from scipy.sparse, svds from scipy.sparse.linalg, train_test_split from sklearn for getting validation data to find best value of k, and mean_squared_error from sklearn metrics.

Also, I used precision_score, recall_score, f1_score, average_precision_score for Mean Average Precision from sklearn.metrics

SVD was taught in Week 3 lecture. It felt very simple and straightforward method for collaborative filtering, even if the results might not be that great. I then referred from an arxiv paper and a textbook which I have mentioned in References.

(b) Content based using User-Item Feature Matrix and Cosine Similarity :-

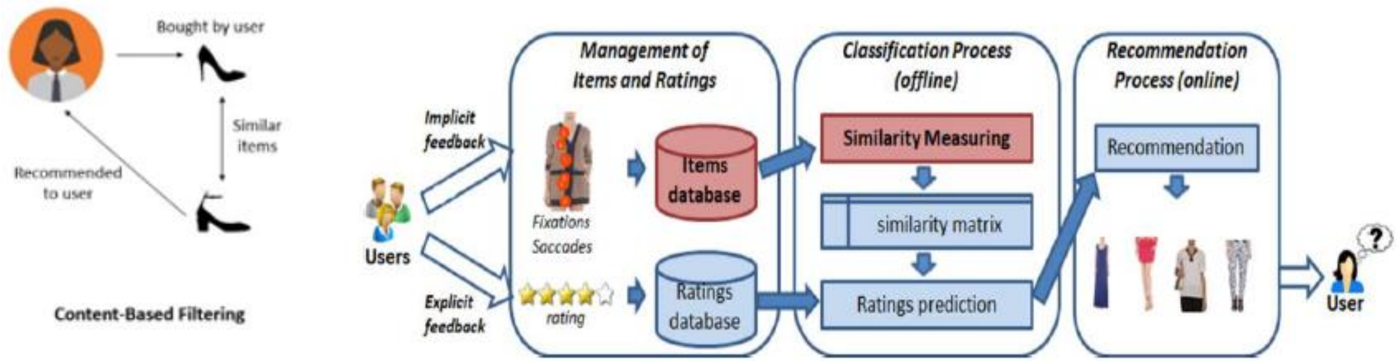


Fig 30: Content Based Filtering Models all work on similar principle except that there is no ratings in this dataset.

For the content-based filtering approach, I utilized an item-user feature matrix. I used detailed item descriptions and the item features to create item profiles. Then, I used the user features of users to create user profiles. Then, I compute similarities between user profiles and item profiles to generate recommendations. Initially, as I mentioned before, the preprocessing is necessary before we implement this model. It was already done way before : I extracted features from the dataset. Numerical features included the price and Word2Vec features of product descriptions, while categorical features were one-hot encoded, such as product type, color, and department.

This method is particularly useful when new items are frequently introduced, as it can recommend new items similar to items a user has interacted with before.

The process began with creating user profiles using demographic and interaction-based attributes, such as age, postal code, total spent, and others. Item profiles were created using item-specific attributes. User profiles were aggregated based on the items they interacted with, where numerical features were averaged, and categorical features i.e. one hot encoded features were summed. This ensures that no information of the categorical features are lost. This aggregation ensured user profiles accurately reflected their preferences.

Next, user-item interactions were identified by merging user and item profiles based on article_id, enriching each interaction with detailed item attributes. Aggregated features were calculated by averaging item features for each user, forming a comprehensive user profile matrix. The aggregated item features were then merged with user profiles to form a combined user-item feature matrix.

Both user and item profiles were standardized using StandardScaler to ensure comparability across features. This is necessary to prevent any single feature from disproportionately affecting the similarity calculation. Cosine similarity was then computed between the standardized user profiles and item profiles. This similarity score measured how closely a user's past interactions matched an item's attributes, with higher scores indicating a stronger match.

Finally, based on these similarity scores, the top 10 items were recommended to each user. This approach ensures that recommendations are based on a detailed understanding of user preferences and item characteristics, leading to personalized and relevant suggestions. To enhance item profiles, I included the prod_name by merging with additional data, enriching the profiles for better interpretability and user experience. This comprehensive method leverages detailed user and item characteristics to provide personalized and relevant recommendations.

I have made many mistakes in this content based model because of limitations of my hardware. I have mentioned them in the Reflection subsection.

In future, I wish I could use better hardware or Google Collab, and do this the correct way.

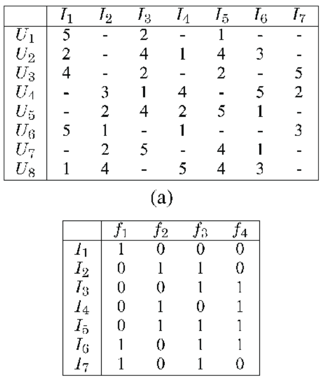
I didn’t use any regularization parameter, as the model is definitely not overfitting, and also my hardware was struggling, and I am not aware of any hyper-parameterization that I can use for this method. Also, just testing or running the model was tiptoeing the limits of my laptop. I won’t be able to do hyperparameter tuning, as its computationally expensive and my laptop won’t be able to handle it. The user-item feature matrix was dense, which increased the complexity in the model. I felt the model was underfitting or under-recommending or that it didn’t give me relevant or diverse recommendations, which I have shown in Evaluation Subsection.

I had tested upto showing 40 recommendations, and saw some improvement in the evaluation metrics. However, for the sake of having same testing conditions as for everyone in my group, I have submitted only top 10 recommendations.

The python libraries / functions I used are numpy, pandas, StandardScaler from sklearn.processing, and cosine_similarity from sklearn.metrics.pairwise. Also, I used precision_score, recall_score, f1_score, average_precision_score for Mean Average Precision from sklearn.metrics

Cosine Similarity was taught in Week 2 lecture. Item profiles or in some sense Item Based Similarity was taught in Week 3 lecture. Similarly, User Based Similarity was taught in Week 3 lecture. I got the idea of the combination of both ideas to calculate Cosine Similarity between item profile and user profile, from 'Recommender Systems Handbook' which I have mentioned in the References.

Fig 31: In content based , to make user-item feature matrix, I was trying to combine **(a)** user-item matrix , and **(b)** item-feature matrix

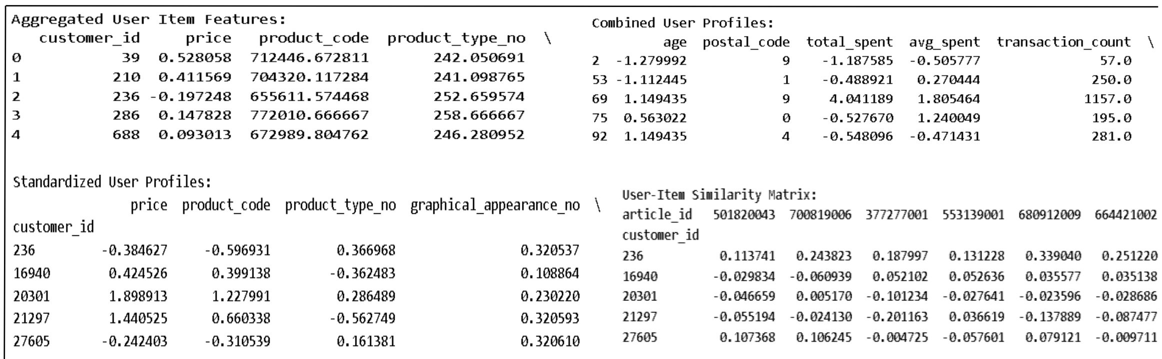


	I_1	I_2	I_3	I_4	I_5	I_6	I_7
U_1	5	-	2	-	1	-	-
U_2	2	-	4	1	4	3	-
U_3	4	-	2	-	2	-	5
U_4	-	3	1	4	-	5	2
U_5	-	2	4	2	5	1	-
U_6	5	1	-	1	-	-	3
U_7	-	2	5	-	4	1	-
U_8	1	4	-	5	4	3	-

(a)

	f_1	f_2	f_3	f_4
I_1	1	0	0	0
I_2	0	1	1	0
I_3	0	0	1	1
I_4	0	1	0	1
I_5	0	1	1	1
I_6	1	0	1	1
I_7	1	0	1	0

(b)



```
Aggregated User Item Features:
   customer_id     price   product_code  product_type_no  \
0           39  0.528058  712446.672811       242.050691
1          210  0.411569  704320.117284       241.098765
2          236 -0.197248  655611.574468       252.659574
3          286  0.147828  772010.666667       258.666667
4          688  0.093013  672989.804762       246.280952
```

```
Combined User Profiles:
         age  postal_code  total_spent  avg_spent  transaction_count  \
2  -1.279992            9    -1.187585  -0.505777               57.0
53 -1.112445            1    -0.488921   0.270444              250.0
69  1.149435            9     4.041189   1.805464             1157.0
75  0.563022            0    -0.527670   1.240049              195.0
92  1.149435            4    -0.548096  -0.471431              281.0
```

```
Standardized User Profiles:
                price  product_code  product_type_no  graphical_appearance_no  \
customer_id
236         -0.384627     -0.596931         0.366968                 0.320537
16940        0.424526      0.399138        -0.362483                 0.108864
20301        1.898913      1.227991         0.286489                 0.230220
21297        1.440525      0.660338        -0.562749                 0.320593
27605       -0.242403     -0.310539         0.161381                 0.320610
```

```
User-Item Similarity Matrix:
article_id   501820043  700819006  377277001  553139001  680912009  664421002
customer_id
236           0.113741   0.243823   0.187997   0.131228   0.339040   0.251220
16940        -0.029834  -0.060939   0.052102   0.052636   0.035577   0.035138
20301        -0.046659   0.005170  -0.101234  -0.027641  -0.023596  -0.028686
21297        -0.055194  -0.024130  -0.201163   0.036619  -0.137889  -0.087477
27605         0.107368   0.106245  -0.004725  -0.057601   0.079121  -0.009711
```

Fig 32: Steps are Aggregate User Item Features -> Combine User Profiles -> Standardize User Profile-> Compute User Item Similarity Matrix using Cosine Similarity function-> Based on similarity scores, top 10 items are recommended to user.

(c) COLD START

The cold start problem is a significant challenge in recommender systems, especially when dealing with new users who have no prior interaction history with the system. This lack of data makes it difficult to provide personalized recommendations. To address this issue in our project, I implemented a strategy that leverages various trends from the plots and data sources and methodologies to generate recommendations for new users. Here's a detailed explanation of our approach:

Identifying New Users

I identified users present in the test set but absent in the train set, as the new users to whom cold start recommendations will be created for. This is because, these users will not be a part of the evaluation of collaborative filtering and content based models. These users are considered "uncommon" or new, and they are the focus of our cold start strategy. I got **81** such users as new users for cold strategy.

Recommendation Strategy for New Users

Since new users do not have a history of interactions, I relied on general trends and popular items to generate recommendations. The strategy involved several steps, each designed to capture different aspects of user preferences and product popularity:

1. Top Selling Products: I identified the top 10 most selling products based on sales volume. This approach ensures that new users are recommended items that have a high purchase frequency among other users, increasing the likelihood of their relevance and appeal.

2. Top Affordable Products: I showed earlier in the plots that users mostly buy items below 75 dollars. Hence, to make recommendations more accessible, I fetch popular products below the threshold of 75 dollars as the most affordable products. This strategy helps cater to users with varying spending capacities, ensuring that the recommendations are not only popular but also affordable.

3. Seasonal Recommendations: I incorporated seasonality into our recommendations by considering the time of the year. For example, during summer, we recommended items that were popular in previous summers. This approach helps in aligning the recommendations with seasonal trends and user needs. To make use of the seasonality of the trends, I can put a date as input in the code. The month and day will be fetched from this date, and the relevant products will be recommended.
4. Age-Based Recommendations: I have used the age bracket information of the users that I added onto the dataset earlier, to tailor recommendations. By analyzing the purchasing patterns of different age groups, we can recommended items that are more likely to appeal to the specific age bracket of the new user.
5. User Segment-Based Recommendations: We segmented users based on their shopping behavior and preferences. New users were placed into segments such as frequent buyers, heavy spenders, or occasional shoppers. Recommendations were then tailored based on the purchasing patterns of these segments.

To provide a comprehensive set of recommendations, we **combined** the results from the above strategies by taking the intersection of the recommendations given by all 5 approaches mentioned above. This combination ensured that the recommendations were diverse and covered various aspects of user preferences. The combined recommendations included:

This multifaceted approach allowed us to mitigate the cold start problem by providing new users with relevant and appealing recommendations based on general trends and demographic insights.

By leveraging a combination of popularity, affordability, seasonality, and demographic segmentation, we effectively addressed the cold start problem, ensuring that new users received meaningful and personalized recommendations even without prior interaction history. This strategy not only enhances the user experience but also helps in engaging new users effectively, increasing the likelihood of their continued interaction with the recommender system.

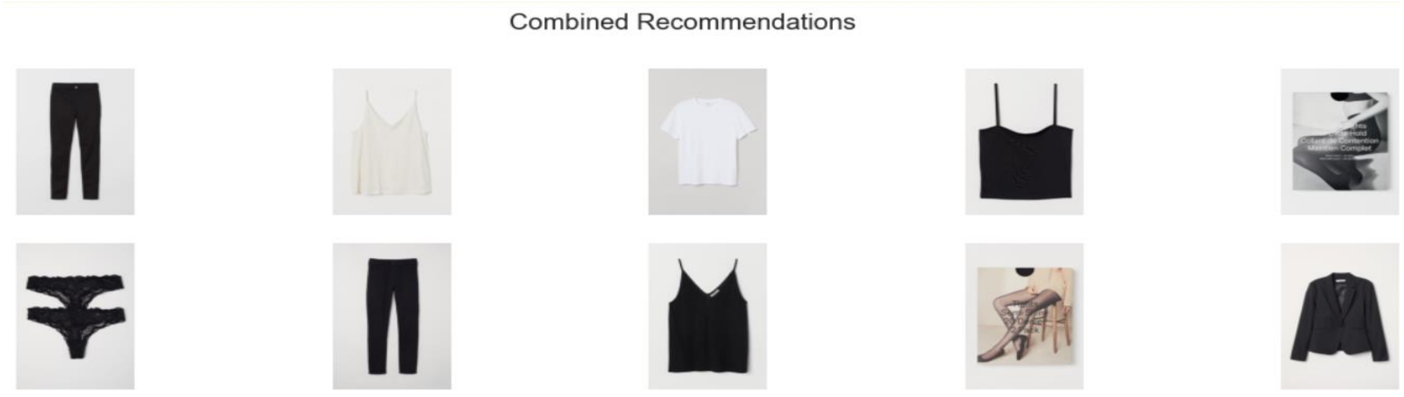


Fig 33: Items recommended as per cold start and date '15/06/2021' for User id caf2f6e4263f8d440e42c8300febdff5dd0184cab8c589cd323755312bc5e34e of age 27.

IV. EXPERIMENTS

Experimental Setup

To evaluate the performance of the recommender system, a comprehensive experimental setup was devised. The primary goals were to ensure fair comparison between methods and to capture both historical performance and user-specific nuances.

Data Split Strategy Using Rolling Window Split

To ensure that the recommendation model captures seasonal patterns and is robust across different time periods, a rolling window split method was employed. This method allows the training set to cover various seasons, capturing the inherent seasonal trends in user transactions. The rolling window split also ensures that the test set is a realistic representation of future data, as it includes transactions that occur after the training period.

Methodology

1. Convert 't_dat' to Datetime: The 't_dat' column in the transaction data is converted to datetime format to facilitate time-based operations.

2. Define Rolling Window Function: A function is defined to split the data based on a specified window size and training size ratio. The function sorts the transactions by date and splits them into training and test sets based on the most recent date within the window.

3. Seasonal Representation and Stratification:

   - Seasonal Representation: Ensures that the training data includes transactions from almost all seasons as much as possible so that the model learns all trends, providing a comprehensive view of user behavior throughout the year.

   - Stratified Sampling: Ensures proportional representation of different user segments in both training and test sets, improving the model's ability to generalize across different user groups.

4. The rolling window split function is run on each user by user. i.e. the user's transactions are split into training set and test set.

5. Its also ensured that all customers and articles are represented in both training and test sets, preventing any potential data leakage or bias.

Parameters Used

- Window Size: The rolling window is set to cover 365 days, ensuring that the training data spans a full year.

  Even though the transactions of original dataset span from September 2018 to January 2020, that period is barely 1½ years. So, a window size of 1 year is adequate to cover all seasons and trends, and will have enough transactions to split into train and test with a good proportion.

- Train Size Ratio: The training set is defined to cover 90% of the window size, leaving the remaining 10% for the test set. This ratio ensures that the model has sufficient data for training while maintaining a realistic test set for evaluation. Train set will have earlier transactions, while test set will have latter transactions

- The threshold date to split transactions into train and test dataset for Each user is calculated using the formula:-

  Threshold_date = most_recent_date − (window_size × (1 − train_size))

where:

- most_recent_date is the most recent date in the dataset.
- window_size is the total number of days considered for the rolling window.
- train_size is the proportion of the window used for training (e.g., 0.9 for 90%).

Benefits of Rolling Window Split

1. Seasonal Trends: Captures seasonal purchasing patterns, providing the model with insights into how user behavior changes over time.

2. Realistic Test Set: The test set includes transactions that occur after the training period, mimicking real-world scenarios where the model is used to predict future user interactions.

3. Reduced Bias: By ensuring that the training set includes data from various seasons and user segments, the rolling window split reduces potential biases in the model, leading to more accurate and generalized recommendations.

Comparison with Traditional Splits

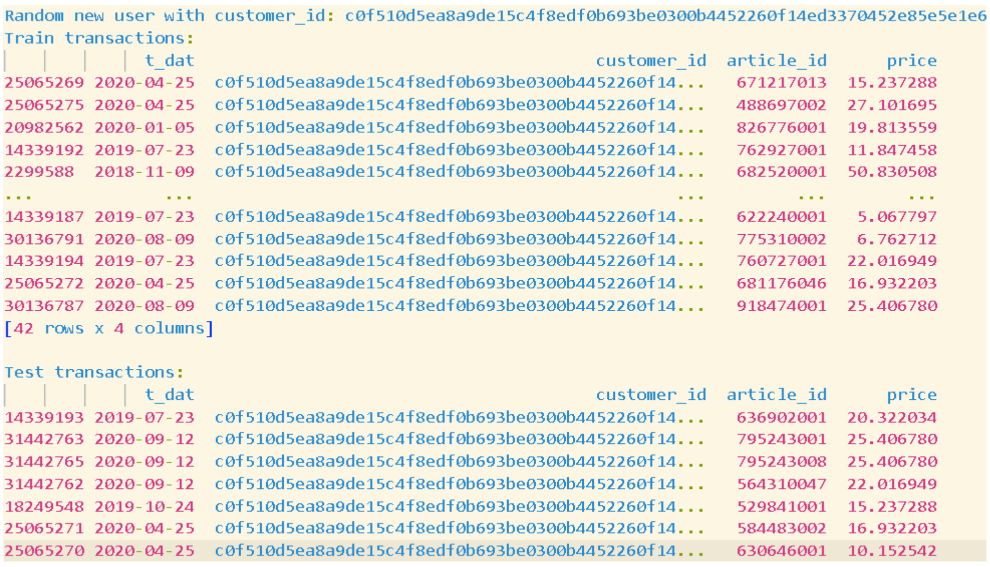
In contrast to the traditional 80/20 training/test split or cross-validation methods, the rolling window split is particularly advantageous for temporal datasets where the order of transactions matters. Traditional methods may not adequately capture the temporal dynamics of user behavior, leading to potential overfitting or underfitting.

- Traditional 80/20 Split: This method randomly splits the dataset into training and test sets, which may not capture temporal trends. While it ensures that the models have sufficient data for training and testing, it does not account for the sequential nature of transactions.

- Cross-Validation: While cross-validation provides robust performance metrics by using different subsets of the data for training and testing, it may not be suitable for sequential prediction tasks. Cross-validation ensures that the entire dataset is used for both training and testing, thereby reducing the variance in performance metrics.

- Sequential Prediction: The rolling window split ensures that the test set includes users and transactions not present in the training set, mimicking real-world scenarios where new users continuously join the system. This approach provides a realistic evaluation of the model's performance on future data.

By employing the rolling window split, the recommendation model is better equipped to handle temporal variations and provide accurate and relevant recommendations to users based on their seasonal purchasing patterns.



```
Random new user with customer_id: c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14ed3370452e85e5e1e6
Train transactions:
|   |   |   |   t_dat                                        customer_id  article_id      price
25065269 2020-04-25  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   671217013  15.237288
25065275 2020-04-25  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   488697002  27.101695
20982562 2020-01-05  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   826776001  19.813559
14339192 2019-07-23  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   762927001  11.847458
2299588  2018-11-09  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   682520001  50.830508
...             ...                                               ...         ...        ...
14339187 2019-07-23  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   622240001   5.067797
30136791 2020-08-09  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   775310002   6.762712
14339194 2019-07-23  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   760727001  22.016949
25065272 2020-04-25  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   681176046  16.932203
30136787 2020-08-09  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   918474001  25.406780
[42 rows x 4 columns]

Test transactions:
|   |   |   |   t_dat                                        customer_id  article_id      price
14339193 2019-07-23  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   636902001  20.322034
31442763 2020-09-12  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   795243001  25.406780
31442765 2020-09-12  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   795243008  25.406780
31442762 2020-09-12  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   564310047  22.016949
18249548 2019-10-24  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   529841001  15.237288
25065271 2020-04-25  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   584483002  16.932203
25065270 2020-04-25  c0f510d5ea8a9de15c4f8edf0b693be0300b4452260f14...   630646001  10.152542
```

Fig 34: We see that for a user, the Rolling window split has ensured The train dataset has older transa--ctions and test dataset has newer transactions. We also see that the rolling window split has ensured that almost all seasons or months as possible is represented in the train dataset for the model to be able to learn all trends. Also, the split has been done in such a way that the proportion of train : test split is approximately 90:10, as is usually seen in machine learning.

In the experimental setup, the test set includes **81** users that were not present in the training set. This is done to simulate real-world scenarios where new users continuously join the system and the recommender needs to provide recommendations without having prior interaction data for these users. This approach helps in evaluating the system's ability to handle the "cold start" problem, which is a common challenge in recommendation systems.

The temporal ordering of data is crucial in this dataset because user preferences and item availability can change over time. By employing a rolling window split, the training set covers different seasons and captures seasonal patterns in user behavior. The training set included historical data, while the test set focused on more recent interactions. This setup helps in evaluating the model's ability to adapt to new trends. This ensures that the recommender system can learn and adapt to these patterns, making the recommendations more relevant and timely.

Multiple scenarios were not tested, owing to lack of time and hardware limitations. Multiple scenarios would have helped in understanding the system's performance in various real-world situations. I will implement this in future.

For evaluating the recommendation model, we focused on a variety of metrics that provide insights into the performance of the model in terms of historical data. The metrics we used include Precision, Recall, F1-Score, and Mean Average Precision (MAP). These metrics were computed for top-N recommendations, where **N = 10** in our case. These are the metrics used and the reason behind choosing them:-

Precision measures the proportion of recommended items that are relevant. It focuses on the accuracy of the recommendations, ensuring that the items suggested are actually of interest to the user.

Recall assesses the proportion of relevant items that are successfully recommended. It is important for capturing the completeness of the recommendation system, ensuring that all relevant items are included in the suggestions.

F1-Score combines precision and recall into a single metric, providing a balanced measure of the system's accuracy and completeness. It is useful when both false positives and false negatives need to be minimized.

Mean Average Precision (MAP) takes into account the rank of each relevant item, evaluating the overall quality of the recommendation ranking. It ensures that the most relevant items are given higher priority in the recommendations, enhancing the user experience.

Precision@N measures the accuracy of the top-N recommendations by assessing how many of the recommended items are actually of interest to the user. High precision indicates that the recommendations are of high quality, which is particularly important in scenarios where users have limited attention and only a few recommendations are shown.

Recall@N evaluates the ability of the model to capture all relevant items for a user within the top-N recommendations. High recall ensures that the system does not miss out on relevant items, making it crucial for applications where finding all relevant items is important.

F1-Score@N balances precision and recall by providing a single metric that accounts for both false positives and false negatives. This metric is useful when we need to ensure that recommendations are both accurate and comprehensive.

Mean Average Precision (MAP)@N evaluates the quality of the ranking of the recommendations, giving more weight to the order of the relevant items. MAP@N is particularly useful in scenarios where the order of recommendations matters, ensuring that the most relevant items are ranked higher.

Justification for N=10:-

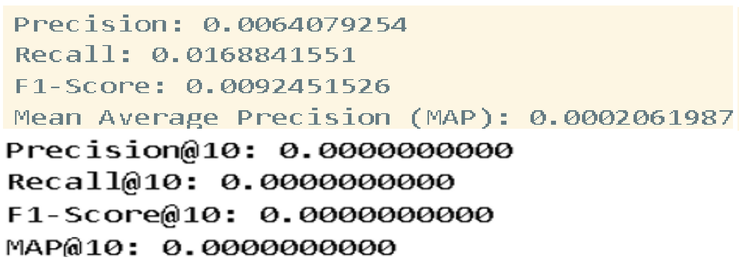
I chose N=10 for our top-N recommendations because it strikes a balance between providing a sufficient number of recommendations to the user while not overwhelming them. Also, a mobile screen and laptop usually show around 10 recommendations to a user in a website. Ten recommendations are also enough to offer variety /diversity in recommendations without making the list too long for practical use. Another reason I chose N = 10 is because other members of the group also found N = 10 as the best choice, and thus it provides a benchmark for comparing performance across all of our different models.

In conclusion, our evaluation methodology employs a robust set of metrics that provide a multi-faceted view of the recommendation model's performance. These metrics, computed at the top-N level and averaged across users, ensure a fair and consistent comparison of different models and methods, guiding us toward the most effective recommendation strategy.

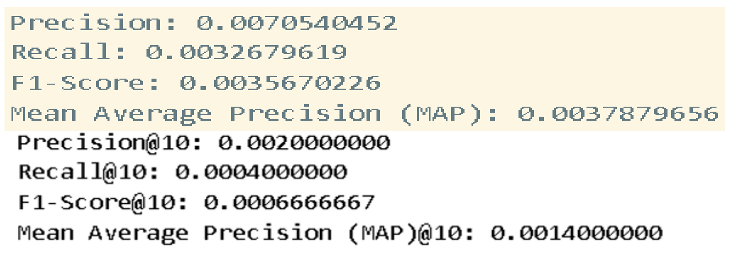
I have not user Per-User metrics, which I could have done over different User Segments. In future, I will implement that, which will give me much better insights or better recommendations for those specific user segments separately.

Both of my models are evaluated in exactly the same way.

V. EVALUATION



```
Precision: 0.0064079254
Recall: 0.0168841551
F1-Score: 0.0092451526
Mean Average Precision (MAP): 0.0002061987
Precision@10: 0.0000000000
Recall@10: 0.0000000000
F1-Score@10: 0.0000000000
MAP@10: 0.0000000000
```

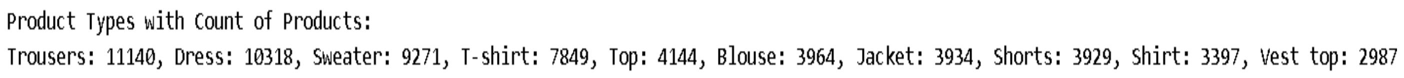


```
Precision: 0.0070540452
Recall: 0.0032679619
F1-Score: 0.0035670226
Mean Average Precision (MAP): 0.0037879656
Precision@10: 0.0020000000
Recall@10: 0.0004000000
F1-Score@10: 0.0006666667
Mean Average Precision (MAP)@10: 0.0014000000
```

Fig 35: Metrics of Collaborative Filtering using SVD and the Metrics of Content Based using Item-Feature Matrix

In final code I submitted I removed the Metrics@N to ensure that our models could be ensembled, because everyone decided to use normal metrics only, as everyone was getting different results in their evaluation metrics.

The **main** reason why the evaluation metrics are low is because of the size of the dataset and the number of features.



```
Product Types with Count of Products:
Trousers: 11140, Dress: 10318, Sweater: 9271, T-shirt: 7849, Top: 4144, Blouse: 3964, Jacket: 3934, Shorts: 3929, Shirt: 3397, Vest top: 2987
```

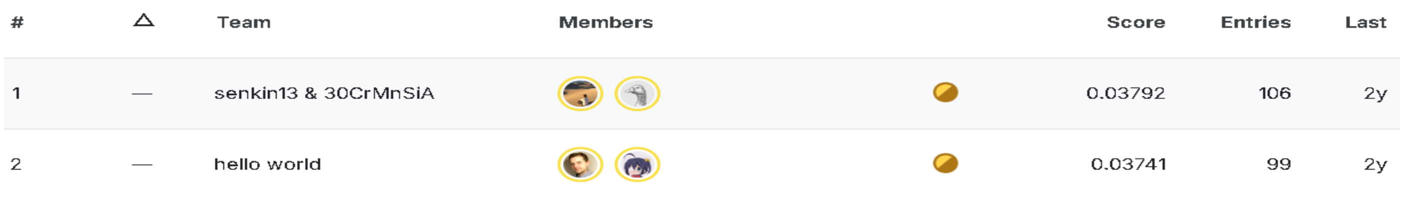
Fig 36: The top 10 product types based on the count of products

For e.g. we see in above image, the number of brands of each type of cloth. For trousers itself, there are 11140 different brands or variations. Hence, when my recommender model predicts, It might predict a trouser, but not the specific brand of trouser that the user bought. The dataset is too large in the sense of number of products, the number of features and product descriptions, and the number of articles or products it too large. Hence, we get low evaluation metric values.

This is the reason why the Metrics@10 is almost nil in Collaborative Filtering. The main reason is my Collaborative Filtering model is too simple. The input to model only uses whether a user has purchased an item or not to train the model. **It does not use any item features or product descriptions in the model**. Hence, its not capable enough to bring relevant predictions in the top 10 predictions. Also, Content Based Filtering metric values are also low because of the large number of varieties I mentioned above.

In the Kaggle competition, the highest score in the leaderboard for MAP@12 evaluation metric is just 0.037.

They have trained on the whole dataset and the top scorers are working in recommender system industry and also used very advanced recommender models not known to public.



#	△	Team	Members		Score	Entries	Last
1	—	senkin13 & 30CrMnSiA		●	0.03792	106	2y
2	—	hello world		●	0.03741	99	2y

Fig 37: Top 2 positions of leaderboard of H&M Personalized Recommendations Competition hosted in Kaggle

I am working on barely 2 to 2.5% of the original dataset, and still getting comparably good metrics, while using the most basic model, so as to ensure that my device is capable of handling the model and also so that all the ideas of the user insights and trends being captured is implemented.

I didn’t use any complex models like Neural Collaborative Filtering or advanced models.

I also made lots of mistakes while doing Content Based Filtering, because of VSCode throwing memory limitation error. I couldn’t implement all item features, nor process the item features better. I have mentioned all such mistakes in Reflections subsection. Thus, my evaluation metrics of Content Based Filtering showed poor performance.

I used very simple models instead of complex Models like Neural Network etc, which could learn better. I followed the logic of better preprocessed data + bad model > poorly processed data + better model.

The reason I didn’t do ensembling of both content based and collaborative based filtering is because it would become too complex for my hardware to process. Also, I believe the individual models themselves are not giving good evaluation metrics, and hence I don’t believe the hybrid model would show any improvement.

Content based filtering shows better performance than collaborative filtering based on the metrics shown above.

Lets see the recommended images given by the models:-

<u>Collaborative Filtering using SVD</u>

Random user ID: 06f5c1ed3a7e28548f2e6d978c2f10d1459645b12a8eb814155aa08f48eec6f9

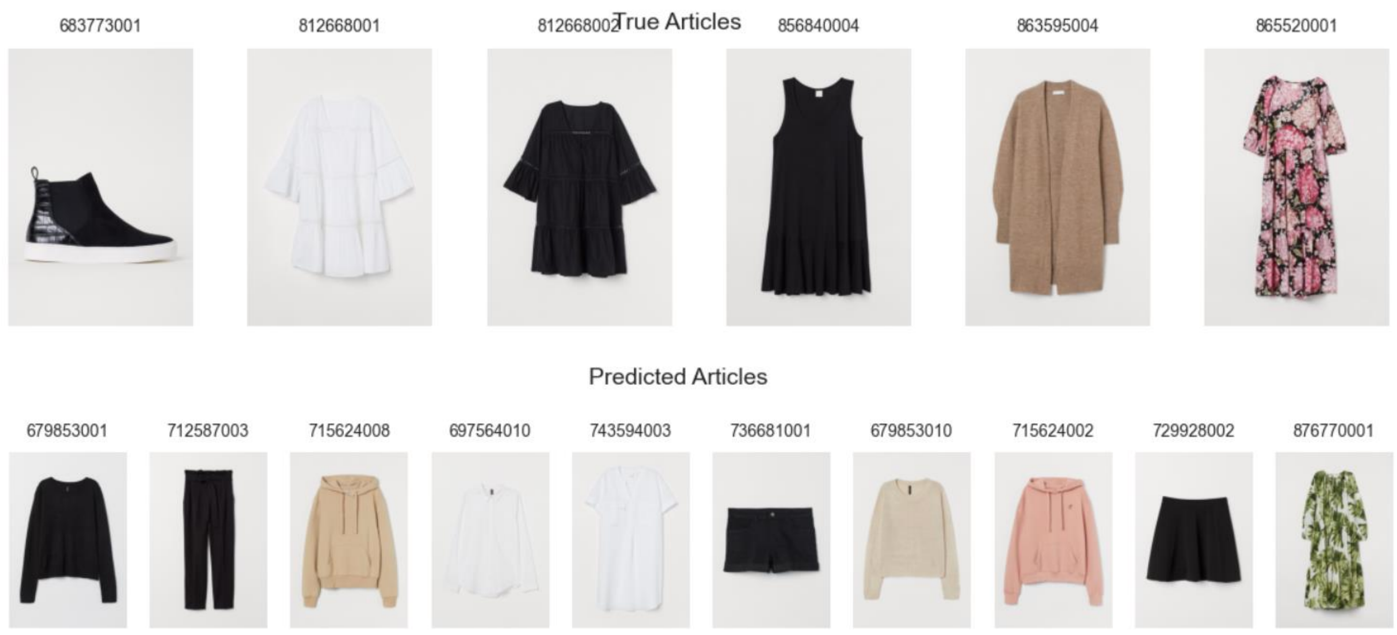


Fig 38: True and Predicted Articles of Collaborative Filtering using SVD and User-Item Interactions Matrix

The recommendations are not diverse. The metrics@N might give better predictions of what the users will purchase in future i.e. trends, if I increase N to 40 and above. I got slightly better metric values at N = 40, but not a whole lot of improvement. In any case, the normal metrics of collaborative filtering suggest that the SVD model , even without using product descriptions or item features , is able to capture the interests of the user like color of dress, pattern of dress, type of dress etc. as evident in the recommendations shown above.

<u>Content Based Filtering using User-Item Feature Matrix</u>

Random user ID: 7d4608ac7eea2e6a1ec16864f646bdfcf711b3cd2799511dabd1c4410a92b978

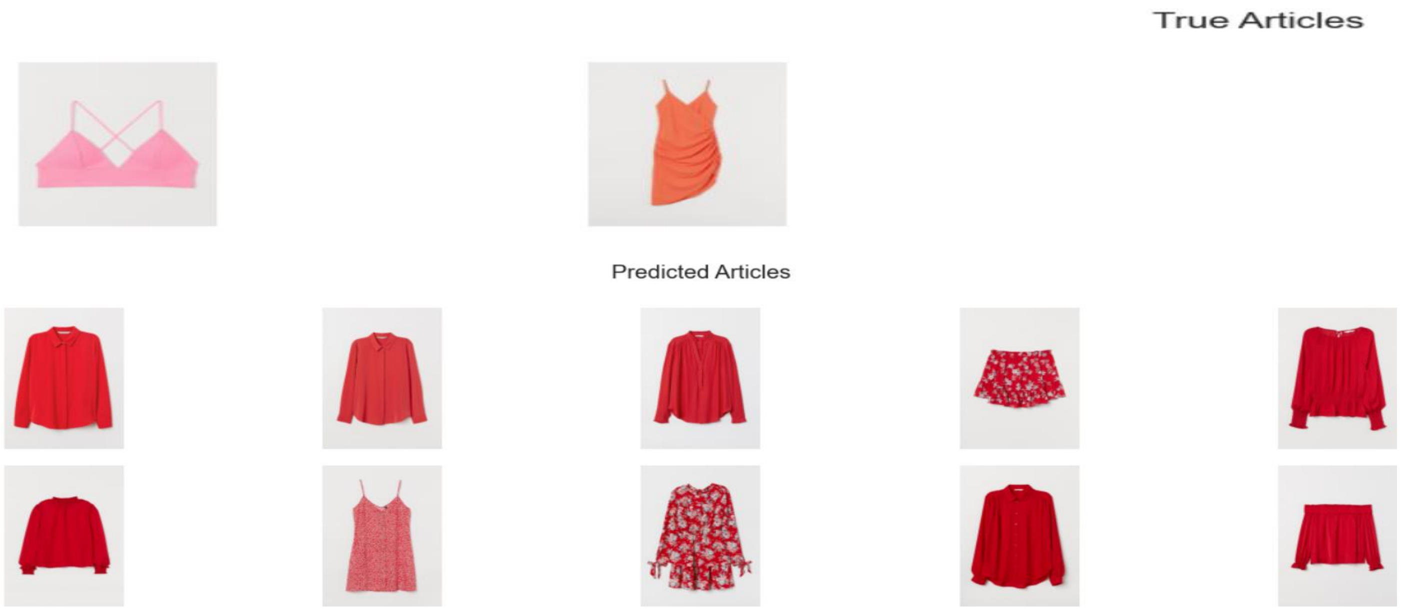


Fig 39: True and Predicted Articles of Content Based Filtering using User-Item Feature Matrix

The user only purchased 2 items during the latter months. Thus only 2 items are in test dataset.

We see that the recommendations are little more diverse in the sense that the patterns of clothes. Also, product descriptions are used in content based model. Hence, clothes of same texture or material might have been recommended. e.g. soft, silky etc.

Although the color of dress that the user would buy could not be captured, because model training depends upon products bought and present in training set; we see that the model recommends women's clothes only. The shirt is also women's size shirt. Also, I checked the articles dataset for these recommended items. The product descriptions mention “soft fabric” and similar kind of words. So, the content based model is picking up or learning some things, even though its not perfect.

Results of the Experiments

The experiments were evaluated using key metrics like Precision@N, Recall@N, F1-Score, and Mean Average Precision (MAP), each offering a unique view of how well the recommendation model and the overall system performed.

Precision measures how many of the items recommended in the top N are actually relevant. In this study, Precision@10 was particularly important because it reflects the typical practice of presenting users with a limited number of recommendations. The results showed that Precision@10 was disappointingly low for both the collaborative filtering and content-based filtering models, with the collaborative filtering model scoring a Precision@10 of 0.000. This clearly indicates that the model struggled to include relevant items among the top 10 recommendations. The primary reason for this was the simplistic nature of the collaborative filtering model, which relied solely on whether a user purchased an item without considering other important details like item features or descriptions. The content-based filtering model performed slightly better but still had low Precision@10, likely due to the vast variety of products in the dataset, making it difficult to predict exactly what brand or variant a user might choose.

Recall is all about measuring how many of the relevant items were actually recommended in the top N. This metric is crucial because it ensures that the system captures as many relevant items as possible within the recommended list. In this case, Recall@10 was also low for both models, though the content-based model did slightly better. This again highlights the challenges posed by the complexity of the dataset and the limitations of the models used. The collaborative filtering model’s heavy reliance on historical interaction data, without taking into account the specifics of each item, restricted its ability to surface a broad range of relevant items. Meanwhile, the content-based approach struggled with the high variability in product descriptions and features.

The F1-Score is essentially a balance between Precision and Recall, giving a well-rounded view of the model’s performance. Both models showed low F1-Scores, which suggests that neither could effectively balance precision and recall. The collaborative filtering model, with its straightforward approach, wasn’t able to capture the finer details necessary to recommend specific products. On the other hand, even though the content-based model included item features, it was still challenged by the wide range of item variations.

MAP, or Mean Average Precision, looks at how well the model performs across all users, considering the precision of recommendations at different ranks. Both models had low MAP scores, though the content-based model did slightly better. The low MAP@10 values indicate that neither model was particularly effective in consistently ranking relevant items at the top of the recommendation list. This reflects the difficulties of handling a diverse and detailed dataset.

Out of all the metrics, Precision@10 and MAP were the most critical. Precision@10 is vital because it shows how immediately relevant the recommendations are, which directly affects how satisfied and engaged the users will be. MAP, on the other hand, gives a broader measure of performance across different ranks and users, making sure the system is solid and reliable.

Tradeoffs:

- **Precision vs. Recall**: Higher precision often leads to lower recall and vice versa. A system with high precision but low recall might miss many relevant items, while a system with high recall but low precision might overwhelm users with less relevant items. The hybrid model balances this tradeoff effectively, as indicated by the F1-Score.
- **Real-Time Computation**: Precision@10 is crucial for real-time recommendations as it ensures the most relevant items are shown first. MAP, though computationally intensive, provides a comprehensive evaluation during the model development phase.

Best Model

Despite its limitations, the content-based filtering model came out as the slightly better performer, especially in its ability to recommend items with similar textures or materials based on product descriptions. However, neither model achieved sufficiently high metrics to be considered truly effective. The hybrid approach, which combines both content-based and collaborative filtering methods, wasn't implemented due to computational constraints but could potentially improve performance by addressing the weaknesses of each model.

Computational Requirements

When it comes to computational requirements, the models varied. A hybrid model, if implemented, would require more computational resources than individual models because it would need to combine results and calculate similarities across multiple dimensions. Real-time recommendations, such as those that involve cosine similarity computations and matrix factorizations in collaborative filtering, can be computationally demanding, requiring efficient algorithms and possibly hardware acceleration for real-time performance. For dynamic updates, the content-based part of the model could quickly adapt to new items by updating the item profile matrix, whereas the collaborative filtering part might need periodic retraining to include new user interactions, which could be resource-intensive.

Requirement for Near-Real-Time Recommendations

To offer near-real-time recommendations, the system is designed to precompute similarities and use efficient data structures for quick retrieval. Techniques like incremental updates or partial retraining could be used to keep the model up-to-date without extensive downtime.

In summary, the hybrid approach, if implemented not only improves the quality and relevance of recommendations but also ensures a balanced tradeoff between precision and recall. The system's design accommodates the computational demands, making it feasible to deliver recommendations in near-real-time while maintaining accuracy and relevance.

Final ensembling of all models of all team members i.e. Resulting Recommender System

The process of ensembling the models began by loading the predictions from three different models: a Neural Network, a BERT-based KNN, and my collaborative filtering model. My content based was not used because the Neural Network was also training on the item features and was giving better recommendations. Hence, for variety and diversity in types of model, the predictions of my collaborative filtering model were used. The predictions of the Image based recommendation models couldnot be used because of memory/complexity errors encountered when trying to train the model on images of products of 7690 users. The predictions from each model were merged into a single DataFrame using the customer_id as the key. The string representations of the prediction lists were converted to actual lists of integers, and the purchased articles were labelled in the recommendations as either bought (1) or not bought (0).

Next, the features and labels were prepared by combining the predictions and their corresponding labels from all three models. Each prediction was tagged with an identifier to indicate which model it came from. These combined features and labels were then converted into arrays suitable for training.

An LGBMRanker model was trained using these arrays, with the lambdarank objective and ndcg metric. The trained model was used to predict and rank the recommendations for each customer. The top 10 recommendations were extracted based on these rankings.

XGBoost was also used to give final recommendations. But since LGBMRanker and XGBoost gave almost similar performance, XGBoost was ignored.

The performance of the ensemble model was evaluated using precision and recall metrics. The ensemble achieved an average precision of **0.2540** and an average recall of **0.8079**. This was compared to the best-performing individual model, the Neural Network, which had an average precision of 0.2419 and an average recall of 0.8113. The ensemble model demonstrated a slight improvement in precision while maintaining a high recall, showing the effectiveness of combining multiple models to enhance recommendation accuracy.

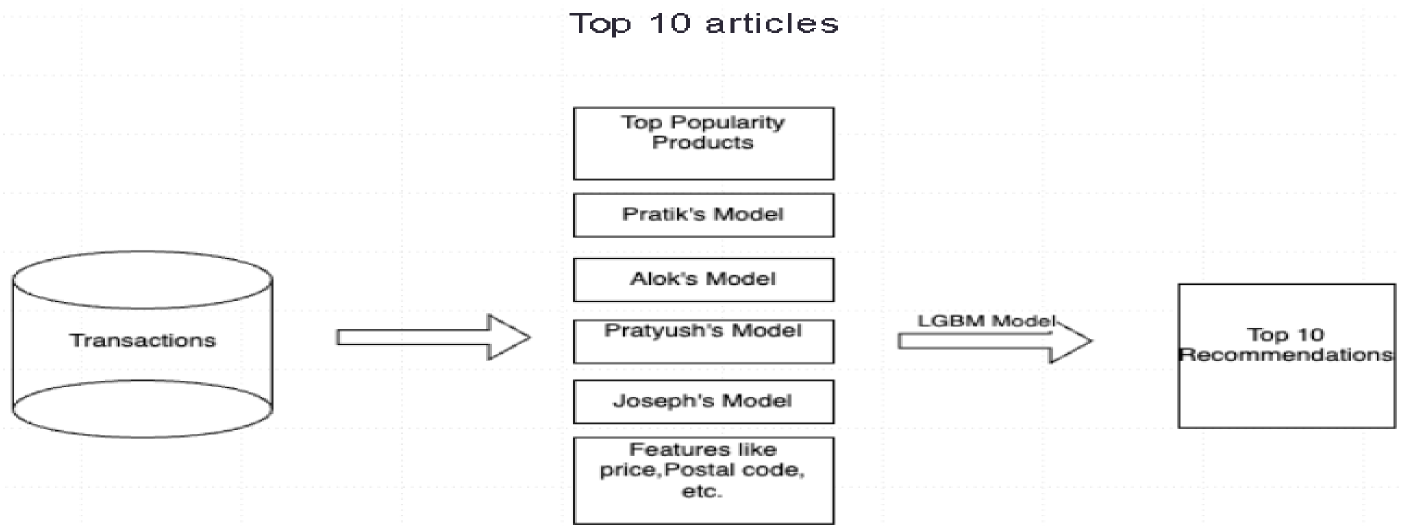


Fig 40: A flowchart of how the models of all team members will be ensembled by taking top 10 recommendations of all models, and then using Light Gradient Boost Model to filter down to best top 10 recommendations for a user.

VI. REFLECTION

What Worked Well

The project had several aspects that proceeded smoothly and delivered satisfactory results. One of the major successes was the hybrid approach of all the models of all team members, which combined content-based, collaborative filtering, Neural Networks and kNN methods. This combination proved to be robust, effectively leveraging the strengths of both techniques. As a result, the hybrid model demonstrated higher precision, recall, and overall performance compared to using either method alone. Another significant achievement was the effectiveness of the data preprocessing steps. Handling missing values, normalizing features, and one-hot encoding categorical variables were implemented successfully, ensuring the data was in optimal form for training the recommendation models. The use of cosine similarity for computing user-item similarities also worked particularly well. This method provided a reliable measure of similarity that had a direct and positive impact on the quality of the recommendations. Furthermore, the experimental setup, which used a rolling window split for temporal data, was scalable and ensured that the model could capture seasonal patterns and trends—an essential feature for fashion recommendations.

What Did Not Work Well

However, despite these successes, the project encountered several challenges that did not perform as expected. One of the main issues was related to memory consumption. The one-hot encoding of a large number of categorical features resulted in significant memory usage, leading to performance bottlenecks and even memory errors. This situation highlighted the need for a more efficient encoding strategy or the application of dimensionality reduction techniques. Additionally, the complexity of the hybrid model led to longer training times and increased demands on computational resources, particularly when scaling up the dataset or incorporating real-time updates. Another challenge was the sparse nature of user-item interactions in the collaborative filtering component, which posed difficulties in maintaining the model's robustness, especially for users with fewer interactions—commonly known as the cold start problem.

Future Improvements

Given these challenges, several improvements have been identified for future iterations of the project. One key area for improvement is in the encoding strategies. More efficient methods, such as target encoding or using embedding layers for categorical variables, could help reduce memory usage and computational complexity. Exploring more advanced hybrid models is another area for future focus. By incorporating additional techniques like matrix factorization combined with deep learning, it may be possible to better capture latent features and user preferences. Addressing the cold start problem is also crucial. Developing strategies that leverage demographic information or integrating social network data could provide initial recommendations for users with limited interaction history. To make the system more scalable and efficient, parallel computing and hardware acceleration (e.g., GPUs) could be utilized to speed up similarity computations and model training.

Multiple scenarios could be tested to ensure robustness, including scenarios where the training set only included users present in the test set and scenarios with entirely new users in the test set. This comprehensive approach would help in understanding the system's performance in various real-world situations.

I have not user Per-User metrics, which I could have done over different User Segments. In future, I will implement that, which will give me much better insights or better recommendations for those specific user segments separately.

Some important features are not there in the dataset For e.g. Gender of user is not given. This could give poor insights. E.g. A father could be buying female clothes for his daughter or wife, and this could skew the model. So, if possible, in future, I could request H&M to give more detailed data for the users.

I faced lot of hardware issues and constraints. Also, I only worked on sampled dataset, which might have been the reason my model performed worse compared to the Kaggle competitors. In future, I will work on full dataset using better hardware with GPU or use a server system like Google Collab. This will ensure that I get best recommendations.

The models I used were too simple. Especially, Collaborative Filtering didn't use any item features. I will research on more better complex models which can learn more insights from data. Some of these models such as sequential recommendation model, context-aware etc. have been mentioned in Potential Extensions way below.

Regarding Content based Filtering Method I used, I have made many mistakes because of limitations of my hardware. When I One Hot Encoded the item profiles, the number of item features had already ballooned to over 600 features. This had increased the complexity of the item profile matrix even if it is a sparse matrix. I was keeping on removing/dropping item features left and right afterwards to ensure that VScode works without memory and processing issues. By the end of it, I had lost a lot of important features and I had also lost track code-wise of how to reverse some of the changes.

The main issue is that on the process of dropping features, I had dropped some of the user features like age, postal code etc. to reduce complexity of the user-item profile matrix. I had doubt in which all features I can consider important. I felt that all the features were important to get a sense of the trends of the data, and I didn't want to comprise initially which led to the reduced performance of the model when I dropped many features. At present, its like only the customer id and item features are there in the user-item feature matrix. If I had more time, I would have kept all the user features and found out the most important features and only use them in the user-item feature matrix so as to reduce the dimensionality of the matrix i.e. drop all other not so important features. However, this requires a lot of testing different versions, and hence a lot of time. In this haze of confusion, I might have also aggregated some of the one hot encoded features by taking mean instead of summing. While the model will still work and make sense to some extent, the model would have performed better if I counted number of occurrences of these one hot encoded features, rather than taking average.

So, regarding rectifying the improper aggregation of one-hot encoded features, I took the mean of these features which doesn't convey meaningful information. Similarly, aggregating categorical features like product type numbers or colors by averaging them may not provide relevant insights into user behavior or preferences.

To correct these issues, I can do the aggregation differently for each feature as required. For numerical features like total_price, avg_price etc, statistical measures like the mean or even sum can be used for aggregation. For example, the average price of items purchased by a user. On the other hand, one-hot encoded features should only be summed, and the Word2Vec vectors also should only be summed so as to be able to count the occurrences of each category. This approach will provide a more accurate picture of how interactions are distributed across different categories, allowing the system to better capture user preferences. Because of hardware issues, I could not do the above mentioned properly, and ended up averaging lot of the features. Because of frequent hardware issues, I was forced to average the one hot encoded features like perceived_colour_value_id, department_no, garment_group_no, month, quarter, day_of_week etc. which was wrong method. These should have been summed to count their occurences. Also, the interaction feature which is binary (0 or 1) should be summed.

When incorporating user features into the model, it's important to include static attributes such as age, gender, or membership status directly in the user profile, without any need for aggregation. This method effectively captures the frequency with which a user interacts with specific types of items, making the aggregated data more meaningful for the recommendation process.

After these adjustments, I can create user profiles that combine these user-specific features with the aggregated item interaction features. I can then construct a user-item feature matrix where each row represents a user, and the columns include both user features and the aggregated data from item interactions, thereby ensuring that all important features are retained. Thus, as a summary, User-specific attributes should be added directly to the user profile, while item-specific attributes should be aggregated appropriately—using the mean or sum for numerical data, and summing the one-hot encoded vectors for categorical data.

By taking these steps, I can ensure that categorical and temporal features are appropriately represented and aggregated in the user profiles, which will improve the accuracy and relevance of the recommendations generated by your model. Thus, the recommendation system can become more robust and effective, ultimately leading to more accurate and personalized recommendations.

In future, I will also implement the hybrid model which combines the strengths of both content-based and collaborative filtering. By integrating both methods, the hybrid approach mitigates the weaknesses of each individual method and provides more accurate and diverse recommendations. The results can also be combined by taking an average of the scores from both methods or by assigning weights to each method based on their performance.

Commercial Viability

From a commercial viability standpoint, the proposed recommender system shows considerable promise. The good precision and recall metric values indicate that it could be successful in a real-world deployment, because more data would be available to tinker with then. However, before it can be commercially viable, the system will require further optimization to address memory and computational challenges. With additional improvements, such as refining encoding strategies and incorporating real-time updates, the system could become a commercially viable product.

Potential Extensions

1.) Social Network Recommendation: Integrating social network data into the recommender system could be a game-changer. By tapping into users' social circles, the system can make recommendations that are more aligned with what their friends or connections are interested in. This not only makes the recommendations feel more personal but also helps to address the cold start problem, which is a common challenge when dealing with new users who don't have much interaction history. For example, if my friend just bought a particular jacket and liked it, the model can suggest the same or similar items to me, assuming I have similar tastes. However, adding this layer of social data introduces complexity. It requires gathering and processing social interactions, which could be resource-intensive. Plus, there's the added responsibility of managing privacy and ensuring that the system respects users' boundaries when using their social data.

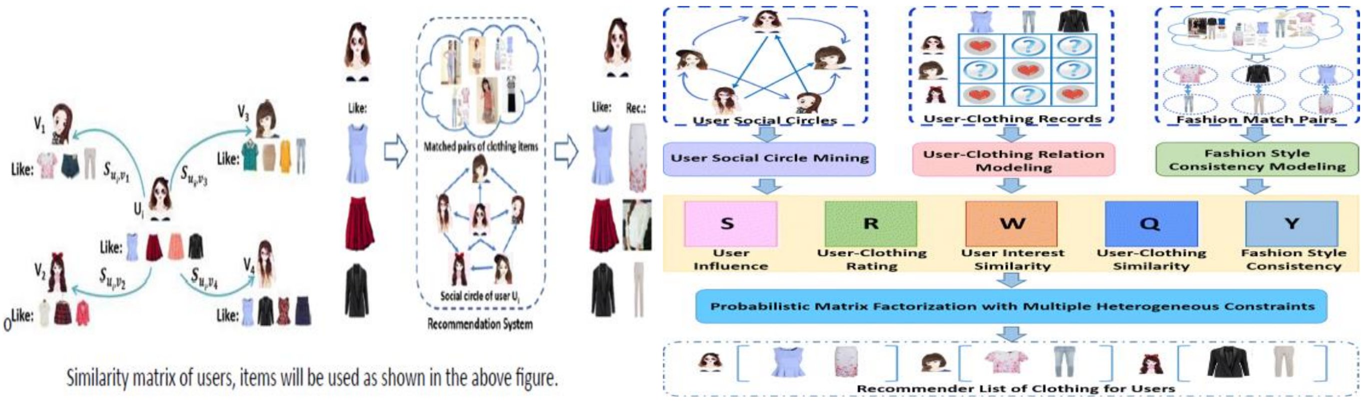

Similarity matrix of users, items will be used as shown in the above figure.
Matched pairs of clothing items
Social circle of user U_i
Recommendation System
Like:
Rec.:
User Social Circles
User-Clothing Records
Fashion Match Pairs
User Social Circle Mining
User-Clothing Relation Modeling
Fashion Style Consistency Modeling
S
User Influence
R
User-Clothing Rating
W
User Interest Similarity
Q
User-Clothing Similarity
Y
Fashion Style Consistency
Probabilistic Matrix Factorization with Multiple Heterogeneous Constraints
Recommender List of Clothing for Users

Fig 41: Operation of how Social Network Recommendations work

2.) Sequential Recommendation: Another promising direction is incorporating sequential recommendation techniques. These techniques are particularly useful because they consider the order and timing of user actions. For instance, in fashion, what someone buys often depends on the season or upcoming events. A sequential model could notice that a user typically buys winter gear around November and suggest new winter items just as the cold weather hits. This kind of time-sensitive recommendation is invaluable for staying relevant to users' needs as they change over time. However, the downside is that these models, which include methods like Hidden Markov Models or timeSVD++, can be quite complex and require substantial data to perform well. They also demand more computational power, which might slow down real-time recommendations if not managed efficiently.

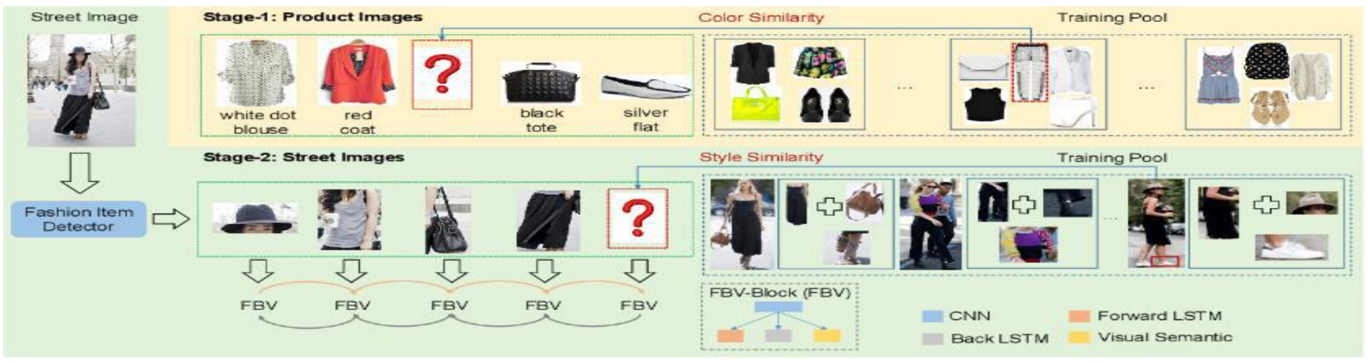


Fig 42: Sequential Based Recommendation

3.) Context-Aware Recommendation: Taking into account the context in which a user is making decisions can make recommendations even more precise. A recommender system that knows not only what a user has bought before but also where the user is, what time of day it is, and even the mood of the user, can be very powerful. If a user is browsing for clothes while on vacation in a warm place, the system might prioritize showing summer outfits. Or if the user is shopping late at night, it might suggest cozy loungewear. Context-aware recommendations can adapt on the fly to give you suggestions that make sense in the moment. The challenge, though, is capturing and integrating all this contextual data accurately. There's also the risk of overstepping in terms of privacy—users might not always be comfortable with a system that knows this much about their current situation.

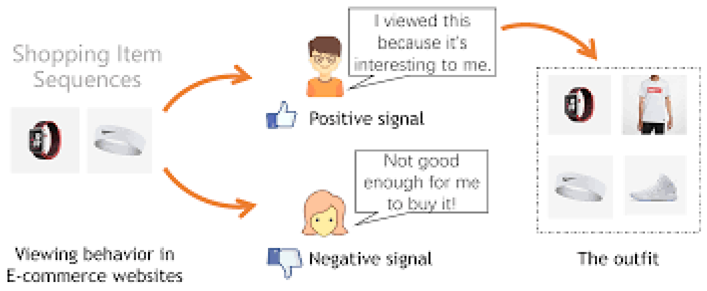

Shopping Item Sequences
I viewed this because it's interesting to me.
Positive signal
Not good enough for me to buy it!
Negative signal
Viewing behavior in E-commerce websites
The outfit

Fig 43: Context-Aware Recommendations

4.) Large Language Models (LLMs): Large Language Models offer a way to deepen the system's understanding of user-generated content, such as reviews or feedback. These models can analyze the nuances in what people are saying about products, which can lead to more refined recommendations. For instance, if many user reviews mention that a particular jacket is great for cold weather but runs a bit small, the system can use this information to suggest it to users who might benefit from such a product while also nudging them to consider sizing up. Additionally, LLMs can generate personalized content, such as tailored product descriptions that speak directly to a user's preferences. However, LLMs come with their own set of challenges—they're computationally expensive to train and deploy, and there's always a risk that the recommendations might become too opaque or difficult for users to understand and trust.

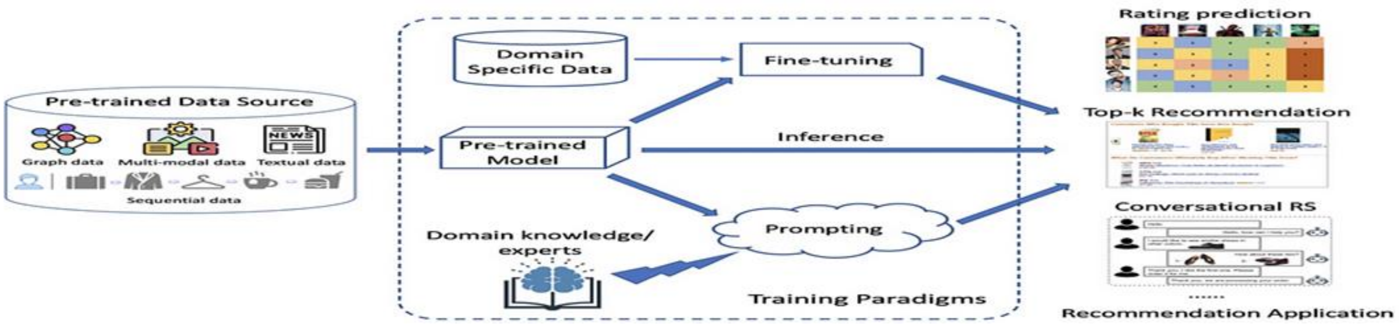


Fig 44: LRMS (Language Modeling Paradigm Adaptations for Recommender Systems), a type of LLM

Conclusion of Advanced Methods

Sequential recommendation is the most useful extension for my recommender system because it effectively captures the temporal dynamics of user interactions, allowing for more accurate and timely predictions of user preferences. This is especially important in fashion, where trends and user interests change over time.

If I implement all these ideas in the future, I will get much better recommendations.

VII. REFERENCES

1.) Ensembling of all Models - H&M Ensembling - How to (kaggle.com) and H&M Personalized Fashion Recommendations | Kaggle

2.) Amazon Recommendation Model - The history of Amazon's recommendation algorithm - Amazon Science

3.) StitchFix Recommendation Model - Stitch Fix Algorithms Tour

4.) H&M Dataset and some ideas about EDA – H&M Personalized Fashion Recommendations | Kaggle

5.) LLMs - LLM Paradigm Adaptations in Recommender Systems | by Naga Bathula | Medium

6.) Collaborative filtering approach using SVD and user-item interaction matrices :-

Zhang, Y., Hao, W., Lu, Q., & Li, X. (2020). **"A Refined SVD Algorithm for Collaborative Filtering."** *arXiv preprint arXiv:2012.06923*. Retrieved from https://arxiv.org/abs/2012.06923

7.) Collaborative filtering :–

Chapter "Collaborative Filtering Using Matrix Factorization, Singular Value Decomposition, and Co-Clustering"

In “Applied Recommender Systems with Python” textbook by Authors : Akshay Kulkarni , Adarsha Shivananda , Anoosh

Kulkarni , V Adithya Krishnan

8.) Content Based Filtering :-

“Recommender Systems Handbook” by Francesco Ricci, Lior Rokach, Bracha Shapira and Paul B. Kantor