DePaul University – Winter, 2022

CSC 575 Final Project - Final Report

CSC 575 Final Presentation

H&M Personalized Fashion Recommendations

Group Members:

Serena Yang, Zhong hua Xie

Table of Contents

Non-technical Parts

Introduction:

In the era of big data, data analysis and machine learning can help customers make personalized recommendations. It can also help a company to improve the revenue by slight changes. Our project aims to build models to save customers time and prevent users from missing out on the best option based on their' past purchases and the products' features.

H&M Group is a household name clothing brand, and this project will focus on its large amount of product and transaction datasets. With H&M Group's enormous amounts of products, users might not quickly find what interests them or what they are looking for. In the end, they might end up not purchasing anything. Furthermore, personalized recommendations can help customers positively impact sustainability, reduce returns, and minimize transportation emissions based on the same interests and tastes that customer have bought and past orders from users. Therefore, customized customer recommendations are essential for improving customers' shopping experiences and company revenue.

In this project, we mainly used Python to achieve these goals. During the process, we first did some data exploration to get familiar with the datasets and find some insights. Secondly, we created three models: Collaborative Filtering, Content-Based Filtering, and Rule-Based Filtering. After the models were created, we stored the recommender product IDs based on the transaction training dataset to determine how many products that test dataset fails in the recommended lists. Finally, we compared these three models with their accuracy.

We used Teams to communicate, track, and share all documents for this project.

Data Preparation

This project aims to predict the clothes/article ID's each customer will purchase.

There are three CSV files:

- Articles.csv: details for each article_id/clothes available for purchase
    - contains 25 explanatory variables including number, text, binary
    - 106k columns
- Customer.csv: metadata for each customer_id in the dataset
    - 7 columns: text, binary, number
    - 1.37m customers

- Transactions.csv: include each customer's purchase on each date, as well as additional information. Duplicate rows correspond to multiple purchases of the same item.
    - 5 columns contain date, text, number
    - 1.36m customers
    - 31m transactions

The metadata ranges from simple data, such as clothing type and customer age, to text data in product descriptions and clothing images. The datasets can be found in Kaggle.

The data was downsized to a manageable size to demonstrate our recommender systems. The original customer data was converted to a dictionary. We used the random function, and randomly selected 1/32 size of customers from the original customer dataset. Then, we pulled their transactions data into new training data from the new customer dataset.

Technical Parts

Collaborative Recommender

The H&M recommendation competition consists of 3 files: transactions, customers, and articles. Unfortunately, it doesn't include a validation or test file because the competition is still ongoing. Therefore, we split 30% of the transaction dataset into a validation dataset. Our goal is to recommend several articles that will match the validation transaction articles.

The first issue for the recommendation system is how to decide the data structure. For the collaborative recommendation, we opted for a python dictionary. The advantage of choosing a dictionary is the flexibility to expand and edit the dataset to accommodate more features. Also, it is easier to search and index by customer id. Moreover, the dictionary can reduce the iteration redundancy by employing a set() hashing function to eliminate duplicate and other repeated transactions. For example, with the dictionary structure {customer id: set([article id, article id, article id, article id, …])} we were able to reduce the transaction rows from 74691 transactions into 4960 rows.

To find the target's K nearest neighbors, we decided on cosine similarity due to its speed and simplicity to implement. However, our data structure was an obstacle as the target and K neighbor don't share the same list length. To combat this issue, we have to reconstruct two lists into the same length by applying the set's union function to get all items within two lists. Then we will rebuild the list using the if statement and check whether the original list article exists in the union set list.

After the K nearest neighbors were found, we converted the list into a matrix. Similar to finding K the nearest neighbor function, we combined all articles purchased by the K nearest neighbors

into a set() to remove duplications. Then we will construct the matrix using the if statement and iterate over to check whether the customer article exists in the set list.

When the build_matrix function and K nearest neighbors functions were completed, we tested the compatibility with the data structure. We then combined the two functions into the get_knn function, which returns the K nearest neighbors and their mean and similarity scores.

```
demo_customer_id = '990e58645b49ec6b0c0069900d86b6d243ce21744e33502de86a74521c682c68'

#demo for function get_knn
get_knn(train_data, demo_customer_id, 5)
```

	similarity	mean
2707700c614496d8af2ce10f7c9a295367f5c95ccbfff14b9076865ebbc8be62	0.333333	0.146341
2ddbfc293dc1ca2776ef595ab2cb90d3ddf88734641121d17387f6b58f6c7d2f	0.308607	0.170732
ba29755b3de1ab95a235dc0c9dfade64920156a836fec89d1126523574091cf3	0.235702	0.073171
1ca787f98bd3da4bab0f694209e69dc35cfef9769954300e9bf61c445ebe7439	0.210819	0.365854
afcbc713086d2d6b2fdf0fd2d680158cf22677f42a7a011793cb68d702043c06	0.198030	0.414634

The original matrix factorization requires filtering out non-rated items to get the mean value of the neighbor. However, since we are working with a binary matrix, 0 was also crucial because it represents the customer did not purchase the item before. Therefore, we sum the K nearest neighbor binary matrix including zero for the mean score to get the mean purchase frequency with the combination of all items purchased by other K nearest neighbors. The cosine similarity score remained unchanged and was included with the mean score data frame.

Next, the prediction function was designed to determine the recommendations given the data frame from the get_knn function. First, we create the binary matrix given the build_matrix function. Then we use get_knn, which calls two additional functions to find the mean and the cosine similarity score. The prediction function will then subtract by the article value to create a weighted matrix that reflects the purchase volume of each k nearest neighbor. The weighted matrix will be less likely to favor a high volume purchase neighbor and heavily penalized when they do not purchase items that appear on the matrix. After K nearest neighbors have been weighed, the weighed value will multiply by its similarity score. The multiplication will reflect the true value of each article by its k nearest neighbor cosine similarity. This operation was done to reward neighbors with a similar purchasing pattern and behavior. This method also deters and eliminates high authority caused by customers with high purchase volume, preventing the potential of highly skewed recommendations caused by high purchase volume customers.

```
#demo for predict function
predict(train_data, demo_customer_id, 50, 12)
```

```
[160442007,
 372860001,
 372860002,
 670752001,
 508184020,
 539197012,
 658841001,
 678861001,
 728162002,
 728162001,
 690980004,
 751592001]
```

The chart below demonstrates the step-by-step process of transforming the binary matrix into recommendations and how it concluded the best recommendations by the sum of article after multiplying the similarity score.

Target	0	0	1	0	1	0	1		
	Article 1	Article 2	Article 3	Article 4	Article 5	Article 6	Article 7	mean	similarity
Customer 1	1	1	1	0	0	0	0	0.428571	0.523
Customer 2	0	1	0	0	1	0	1	0.428571	0.742
Customer 3	0	1	1	0	1	1	1	0.714286	0.333
Customer 4	0	1	1	1	1	0	0	0.571429	0.542
Customer 5	0	1	0	1	0	1	0	0.428571	0.491

↓ Weighted by customer's mean

	Article 1	Article 2	Article 3	Article 4	Article 5	Article 6	Article 7	mean	similarity
Customer 1	0.571429	0.571429	0.571429	-0.42857	-0.42857	-0.42857	-0.42857	0.428571	0.523
Customer 2	-0.42857	0.571429	-0.42857	-0.42857	0.571429	-0.42857	0.571429	0.428571	0.742
Customer 3	-0.71429	0.285714	0.285714	-0.71429	0.285714	0.285714	0.285714	0.714286	0.333
Customer 4	-0.57143	0.428571	0.428571	0.428571	0.428571	-0.57143	-0.57143	0.571429	0.542
Customer 5	-0.42857	0.571429	-0.42857	0.571429	-0.42857	0.571429	-0.42857	0.428571	0.491

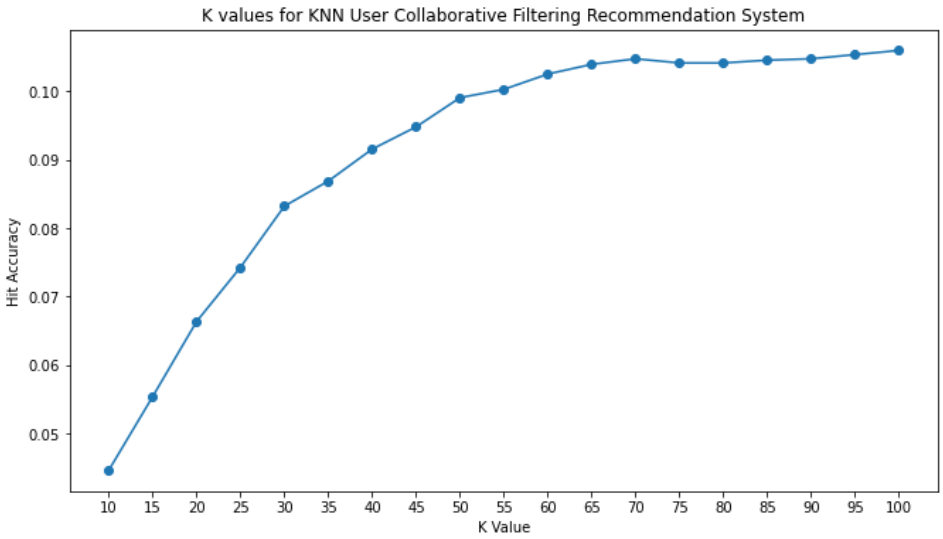
↓ Multiply by similarity

	Article 1	Article 2	Article 3	Article 4	Article 5	Article 6	Article 7	mean	similarity
Customer 1	0.298857	0.298857	0.298857	-0.22414	-0.22414	-0.22414	-0.22414	0.428571	0.523
Customer 2	-0.318	0.424	-0.318	-0.318	0.424	-0.318	0.424	0.428571	0.742
Customer 3	-0.23786	0.095143	0.095143	-0.23786	0.095143	0.095143	0.095143	0.714286	0.333
Customer 4	-0.30971	0.232286	0.232286	0.232286	0.232286	-0.30971	-0.30971	0.571429	0.542
Customer 5	-0.21043	0.280571	-0.21043	0.280571	-0.21043	0.280571	-0.21043	0.428571	0.491

Sum of Articles

Article 2	1.330857
Article 5	0.316857
Article 3	0.097857
Article 7	-0.22514
Article 4	-0.26714
Article 6	-0.47614
Article 1	-0.77714

With the recommendation system, we tested the algorithm with the validation dataset.



After testing 20 different K values, the recommendation system achieved an accuracy of 9.99%, with K value = 50. While the accuracy continued to improve as the K value increased. However, the calculation complexity and runtime also increase drastically. Therefore, we decided K value = 50 is ideal for this dataset.

The result was surprising given the size of articles with 106k of possible options to choose from. However, we also noticed that the validation dataset was inadequate for the recommender system. The validation dataset should be tested by click rate and track how many customers clicked the recommendations given by the system. That way, we can determine if the system provides meaningful information for customers.

Content-Based Recommender

Content-Based Recommender is to give recommendations to a user based on items with "similar" content in the user's profile. In other words, after a user purchases an item, we calculate the similarities between the purchased item and all the other items, find the top similar articles, and recommend them to the user.

We used a matrix to calculate all the similarities between the target item and other available items to achieve this goal. Initially, we used the most common way to calculate the Transaction training dataset to get all the similarities with cosine similarity in the Pandas DataFrame line by line, sort the DataFrame by Cosine similarity, and finally return the top K items for each user. Then ran the Transaction test dataset to see whether the user purchased item is in the recommended list. However, the running time is vast due to our enormous data size. Roughly calculated, we need 11 days to finish running the entire Transaction training dataset by getting the average time of generating one list of recommendations for one article.

Apparently, the straightforward way is not suitable for this dataset; therefore, we made slight changes to improve the running time in a few places. First, we used SET{} to store the data locally for further use to save time. One set stores all the articles IDs and each article's features that we want to analyze. Another set stores each user's ID, and each user purchases articles' IDs.

```
featuresPerItem
```

```
defaultdict(set,
            {888727009: {1, 4, 5, 9, 18, 265, 1005, 1641, 888727, 1010016},
             568601006: {1, 4, 5, 9, 11, 264, 1008, 1212, 568601, 1010016},
             720125004: {4, 5, 7, 12, 26, 273, 1005, 8310, 720125, 1010005},
             611415005: {1, 7, 18, 43, 61, 286, 1017, 1338, 611415, 1010016},
             796210002: {1, 7, 15, 18, 42, 272, 1009, 1722, 796210, 1010004},
             868042002: {1, 4, 5, 9, 18, 259, 1010, 1941, 868042, 1010016},
             583558001: {2, 53, 72, 265, 1013, 1344, 583558, 1010023},
             768440002: {1, 3, 9, 10, 15, 255, 1005, 1636, 768440, 1010016},
             621846001: {1, 4, 5, 9, 61, 306, 1017, 1338, 621846, 1010021},
             703973004: {4, 6, 73, 76, 265, 1005, 7616, 703973, 1010001},
             756633002: {1, 2, 15, 72, 274, 1025, 1723, 756633, 1010023},
```

Sample of the first SET

```
itemsPerUser
```

```
defaultdict(list,
            {'ff14443609338d705070b99c7616442393cb4c281151cd827fd142a2c7b8e49f': [888727009,
              253448056,
              639976002,
              869900001,
              888507001,
              633977002,
              853097001,
              253448001,
              615047001],
             '1954d32d7f103e23ee78776c7adc1cf31b4d6545d068b11d9dfe7c43cfb9b657': [568601006,
              608834002,
              655784006,
              553423001,
              568601018,
```

Sample of the second SET

After creating these two SETs, we replaced the Cosine Similarity with Jaccard Coefficient to calculate the distances. The difference between Jaccard Similarity and Cosine Similarity is that Jaccard similarity takes only a unique set while cosine similarity takes total weights of the list.

To get the most similar items, we also used SET{} to store each user's name and a list of recommender articles' IDs. We can get most K numbers similarity items by passing the two SETs which mentioned above into the Jaccard method.

```
query = transactions_train_dict[0]['article_id']
mostSimilar(query, 50)
```

```
[(1.0, 888727008),
 (0.6, 932365005),
 (0.6, 928040002),
 (0.6, 927530006),
 (0.6, 927530004),
 (0.6, 927529001),
 (0.6, 916273002),
 (0.6, 909087001),
 (0.6, 906645009),
 (0.6, 906645007),
 (0.6, 899225001),
 (0.6, 899129001),
 (0.6, 893045003),
```

Sample of the list of similarity article' IDs

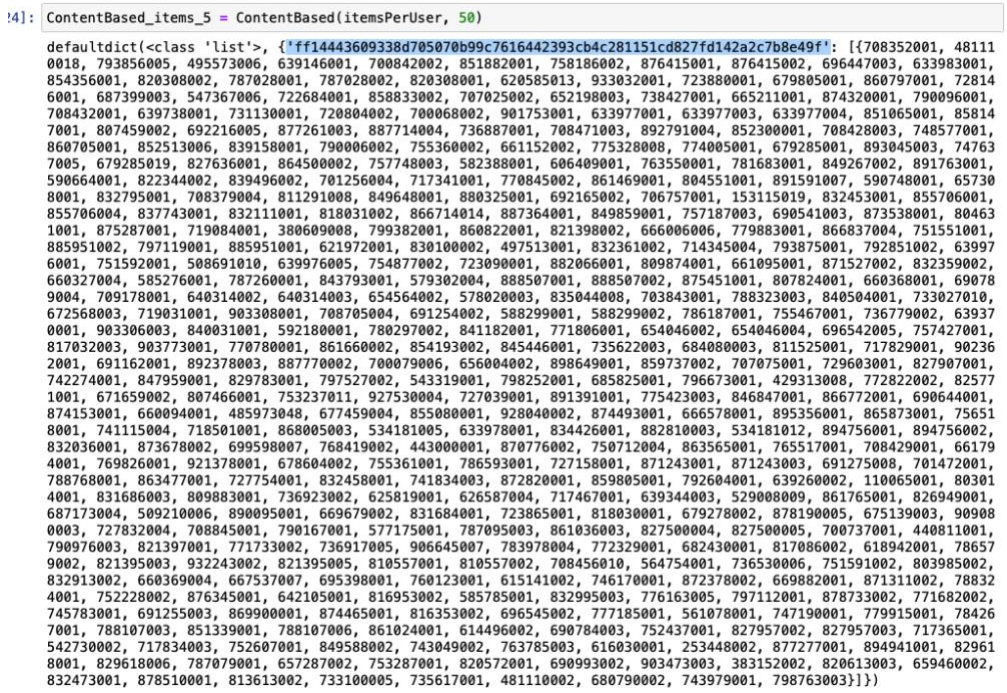
By using this method, the model only takes 1.5 hrs to finish running on the training dataset. Compared to the original running time, it is a considerable improvement.

Since different features might cause different results, we created two models with two different features to find the most accurate model. In the first model, we chose all the category types like product type, color, pattern, perceived color, etc. However, we only chose two significant types for the second model: product code and product type code. With the features number decreased, the running time also got minimized.

```
['article_id', 'product_code', 'product_type_no', 'graphical_appearance_no', 'colour_group_code',
'perceived_colour_value_id', 'perceived_colour_master_id', 'department_no','index_group_no', 'section_no',
'garment_group_no']
```

List of the second model's features

We first stored all the predictions for all the transaction article IDs from the training dataset to get accuracy. By running the test transaction dataset to determine how many users purchased items in the recommended list group by users' ID.



```
[4]: ContentBased_items_5 = ContentBased(itemsPerUser, 50)

defaultdict(<class 'list'>, {'ff14443609338d705070b99c7616442393cb4c281151cd827fd142a2c7b8e49f': [{708352001, 48111
0018, 793856005, 495573006, 639146001, 700842002, 851882001, 758186002, 876415001, 876415002, 696447003, 633983001,
854356001, 820308002, 787028001, 787028002, 820308001, 620585013, 933032001, 723880001, 679805001, 860797001, 72814
6001, 687399003, 547367006, 722684001, 858833002, 707025002, 652198003, 738427001, 665211001, 874320001, 790096001,
708432001, 639738001, 731130001, 720804002, 700068002, 901753001, 633977001, 633977003, 633977004, 851065001, 85814
7001, 807459002, 692216005, 877261003, 887714004, 736887001, 708471003, 892791004, 852300001, 708428003, 748577001,
860705001, 852513006, 839158001, 790006002, 755360002, 661152002, 775328008, 774005001, 679285001, 893045003, 74763
7005, 679285019, 827636001, 864500002, 757748003, 582388001, 606409001, 763550001, 781683001, 849267002, 891763001,
590664001, 822344002, 839496002, 701256004, 717341001, 770845002, 861469001, 804551001, 891591007, 590748001, 65730
8001, 832795001, 708379004, 811291008, 849648001, 880325001, 692165002, 706757001, 153115019, 832453001, 855706001,
855706004, 837743001, 832111001, 818031002, 866714014, 887364001, 849859001, 757187003, 690541003, 873538001, 80463
1001, 875287001, 719084001, 380609008, 799382001, 860822001, 821398002, 666006006, 779883001, 866837004, 751551001,
885951002, 797119001, 885951001, 621972001, 830100002, 497513001, 832361002, 714345004, 793875001, 792851002, 63997
6001, 751592001, 508691010, 639976005, 754877002, 723090001, 882066001, 809874001, 661095001, 871527002, 832359002,
660327004, 585276001, 787260001, 843793001, 579302004, 888507001, 888507002, 875451001, 807824001, 660368001, 69078
9004, 709178001, 640314002, 640314003, 654564002, 578020003, 835044008, 703843001, 788323003, 840504001, 733027010,
672568003, 719031001, 903308001, 708705004, 691254002, 588299001, 588299002, 786187001, 755467001, 736779002, 63937
0001, 903306003, 840031001, 592180001, 780297002, 841182001, 771806001, 654046002, 654046004, 696542005, 757427001,
817032003, 903773001, 770780001, 861660002, 854193002, 845446001, 735622003, 684080003, 811525001, 717829001, 90236
2001, 691162001, 892378003, 887770002, 700079006, 656004002, 898649001, 859737002, 707075001, 729603001, 827907001,
742274001, 847959001, 829783001, 797527002, 543319001, 798252001, 685825001, 796673001, 429313008, 772822002, 82577
1001, 671659002, 807466001, 753237011, 927530004, 727039001, 891391001, 775423003, 846847001, 866772001, 690644001,
874153001, 660094001, 485973048, 677459004, 855080001, 928040002, 874493001, 666578001, 895356001, 865873001, 75651
8001, 741115004, 718501001, 868005003, 534181005, 633978001, 834426001, 882810003, 534181012, 894756001, 894756002,
832036001, 873678002, 699598007, 768419002, 443000001, 870776002, 750712004, 863565001, 765517001, 708429001, 66179
4001, 769826001, 921378001, 678604002, 755361001, 786593001, 727158001, 871243001, 871243003, 691275008, 701472001,
788768001, 863477001, 727754001, 832458001, 741834003, 872820001, 859805001, 792604001, 639260002, 110065001, 80301
4001, 831686003, 809883001, 736923002, 625819001, 626587004, 717467001, 639344003, 529008009, 861765001, 826949001,
687173004, 509210006, 890095001, 669679002, 831684001, 723865001, 818030001, 679278002, 878190005, 675139003, 90908
0003, 727832004, 708845001, 790167001, 577175001, 787095003, 861036003, 827500004, 827500005, 700737001, 440811001,
790976003, 821397001, 771733002, 736917005, 906645007, 783978004, 772329001, 682430001, 817086002, 618942001, 78657
9002, 821395003, 932243002, 821395005, 810557001, 810557002, 708456010, 564754001, 736530006, 751591002, 803985002,
832913002, 660369004, 667537007, 695398001, 760123001, 615141002, 746170001, 872378002, 669882001, 871311002, 78832
4001, 752228002, 876345001, 642105001, 816953002, 585785001, 832995003, 776163005, 797112001, 878733002, 771682002,
745783001, 691255003, 869900001, 874465001, 816353002, 696545002, 777185001, 561078001, 747190001, 779915001, 78426
7001, 788107003, 851339001, 788107006, 861024001, 614496002, 690784003, 752437001, 827957002, 827957003, 717365001,
542730002, 717834003, 752607001, 849588002, 743049002, 763785003, 616030001, 253448002, 877277001, 894941001, 82961
8001, 829618006, 787079001, 657287002, 753287001, 820572001, 690993002, 903473003, 383152002, 820613003, 659460002,
832473001, 878510001, 813613002, 733100005, 735617001, 481110002, 680790002, 743979001, 798763003}]})
```

Sample of a list of recommended articles based on all the products that one user purchased. K = 50, for each article.

After running these models, the accuracy for the content-based filtering with all categories is 0.201% and for the content-based filtering with only two features is 0.182%, where the K = 50, since we want to control the recommendation number for all models.

This model got such an insignificant accuracy because the Content-Based Recommender method recommends a similar item that a user already purchased. For example, user A purchased a white tee; the content-based recommender system will recommend user A a bunch of the most similar white Tees based on the similarity matrix. However, this user will only purchase one white tee. Therefore, this recommender system could perform better on the exchange page or in the reviewing/shopping stage before user purchase.

Rule-Based Recommender

Rule-based filtering is also called knowledge-based filtering, which provides recommendations to users based on predefined rules. After the data exploration, other than the first two types of models, we decided to do a model to get the top K items in the past three weeks. First, to get the newest date from the transaction dataset, filter all the transactions before the (newest date-3 weeks), which is '2020-09-01', and then group 'customer_id' by the 'article_id'. Finally, return the most top K items.

```
[16]: top_50(transactions, 50)

[16]: [751471038,
       896169002,
       714790028,
       909370001,
       912095007,
       913033002,
       673677023,
       809238001,
       715624010,
       928839001,
       896152002,
       860797001,
       678942001,
       804992017,
       717490070,
       770315020,
       878190005,
       921378001,
       902419001,
```

Sample of top 50 items in the past three weeks model output

After using the same logic from the content-based recommender part to get the accuracy for Top K items in the past three weeks model, the result is 0.05% with K=50. The accuracy is so low because three weeks transaction is a tiny portion of the transaction test dataset. There a large number of user could not purchase those items.

Therefore, we created another model with popular K items among the entire dataset. The final accuracy is 0.29% with K=50, which is not bad. However, this accuracy is so low because the length of the article dataset is about 106k, and 50 items in a really small portion among the entire article dataset. Therefore, the accuracy for the popular K items among the entire dataset model will increase with the K number increase.

Model Comparing

Model Name	*Accuracy*
Collaborative Recommender	9.99%
Content-Based Recommender with 10 features	0.2%
Content-Based Recommender with 2 features	0.18%
Rule-Based Recommender (Top K items in the past 3 weeks)	0.05%
Rule-Based Recommender (Popular K items among the entire dataset)	0.29%

Comparing these few algorithms, Collaborative Recommender performs the best after the user purchases one item. The Popular K items among the entire dataset model could also perform well with an increased K number. The Content-Based Recommender System could perform better on the exchange page or in the reviewing/shopping stage before user purchase.

Conclusion

We were able to see a glimpse of the potential of recommender systems. However, we were unable to demonstrate it thoroughly with the validation dataset. The current validation dataset stemmed from the training dataset, which only contains transactions information. It reflected the poor performance of content-based and rule-based recommenders. If we could use a different validation method, we can see the potential of the recommenders simply by tracking the click rate. Or we could employ a/b testing method by splitting customers into two groups and if the recommenders improve customer browsing time and increase sales.

Another concern with the KNN based recommender system is the computational complexity. We want to be responsive and not cause constraints on existing servers and systems. We should be able to further improve the runtime by utilizing more customer information and generating labels which further separate customers into smaller groups that share similar features. For example, we can implement an age group by using customers' age and geographical locations.

In conclusion, we successfully created three different types of recommender systems. However, the current validation dataset could not demonstrate its true capability. The recommender systems required a different evaluation method.