Recommender Systems Report

Abstract

In this abstract, we explore the methodologies and techniques behind collaborative and content-based recommender systems. We delve into collaborative filtering's reliance on user behavior patterns and similarities, as well as content-based filtering's focus on item attributes and user preferences. And demonstrate an example for each oneof them.

Introduction:

Collaborative filtering and content-based methods are the two most commonly used recommender systems. Collaborative filtering relies on user-item interactions and similarities between users or items to make recommendations. In this study, we created a model-based collaborative filtering which first creates a utility matrix and then decomposes it using SVD. On the other hand, content-based filtering leverages the features of items and user preferences to generate recommendations. In this study, we used the TF-IDF method to obtain meaningful features from the description of each product, combined it with other features of each product, and then built the cosine similarity matrix for it.

content-based recommender system

First step is to load the corresponding dataset and investigate it.
After loading the dataset into a dataframe we can see the first 5 elements and all our features.

	index	product	category	sub_category	brand	sale_price	market_price	type	rating	description
0	1	Garlic Oil - Vegetarian Capsule 500 mg	Beauty & Hygiene	Hair Care	Sri Sri Ayurveda	220.0	220.0	Hair Oil & Serum	4.1	This Product contains Garlic Oil that is known...
1	2	Water Bottle - Orange	Kitchen, Garden & Pets	Storage & Accessories	Mastercook	180.0	180.0	Water & Fridge Bottles	2.3	Each product is microwave safe (without lid), ...
2	3	Brass Angle Deep - Plain, No.2	Cleaning & Household	Pooja Needs	Trm	119.0	250.0	Lamp & Lamp Oil	3.4	A perfect gift for all occasions, be it your m...
3	4	Cereal Flip Lid Container/Storage Jar - Assort...	Cleaning & Household	Bins & Bathroom Ware	Nakoda	149.0	176.0	Laundry, Storage Baskets	3.7	Multipurpose container with an attractive desi...
4	5	Creme Soft Soap - For Hands & Body	Beauty & Hygiene	Bath & Hand Wash	Nivea	162.0	162.0	Bathing Bars & Soaps	4.4	Nivea Creme Soft Soap gives your skin the best...

How many data samples do we have?

(27555, 10)
We have 27555 data samples and 10 features.

Do we have any missing values?

```
index             0
product           1
category          0
sub_category      0
brand             1
sale_price        0
market_price      0
type              0
rating         8626
description     115
dtype: int64
```

As it shows we have a noticeable number of missing values, especially the significant amount of missing ratings. And since we plan to use ratings as a content feature this is concerning because it indicated approximately ⅓ of the data have missing ratings.
But for product and brand, we only have one missing value so it can easily be dropped.
And since description is not something to be easily imputed we have no choice but to drop its missing values as well.
For the rating problem, we can always use imputation methods but we can also go extreme and train a regression predictor for rating values. In this scenario, we chose the second.
For the sale_price and market price we can introduce a new feature:
Discount which is market_price - sale_price and then delete these two features to avoid complication.
Now let's encode the categorical data before model training.
For the brand feature, we chose frequency encoding and dropped other categorical data because they had too many unique values.
Then two data frames are created: train (including rating values) and tes (all rows with missing rating)
Now train is split into train and test.
MinMaxScaler is chosen as the scaler metric and performed. Then a XGBRegressor is trained on the training sample. With these parameters:

```
param_lst = {
    "learning_rate" : [00.1,0.1,0.15,0.3,0.5],
    "n_estimators" : [100,500,1000,2000,3000],
    "max_depth" : [3,6,9],
    "min_child_weight" : [1,5,10,20],
    "reg_alpha" : [0.001,0.01,0.1],
    "reg_lambda" : [0.001,0.01,0.1]
}
```

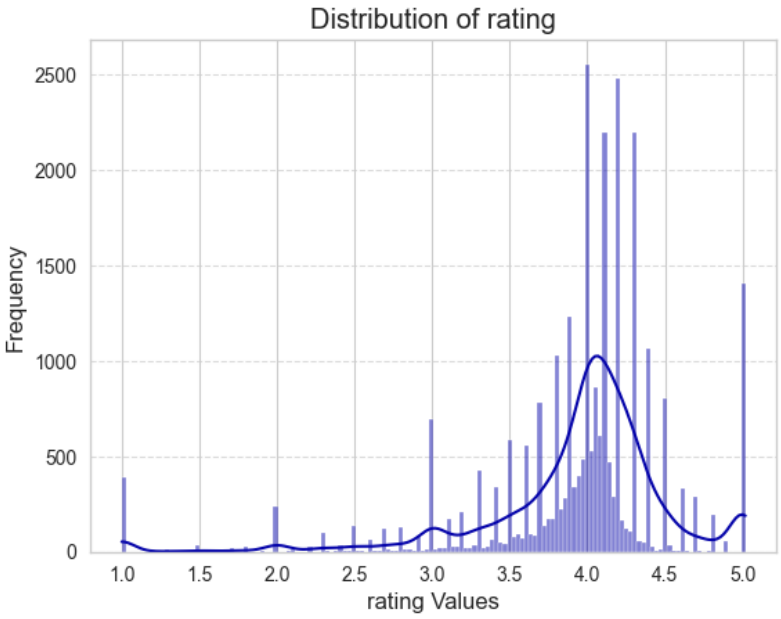
	Model	MAE	RMSE	Score
0	XGBoost	0.4752568941460547	0.6893887830143849	0.08188747972698851

Above is the result obtained from this training process but it doesn't seem very pleasant however because still, we believe it would be better than simple imputation we go on and fill the missing values of the initial test df with this regressor. And then save the merged_df as a new csv which now has no missing values.

The new df is loaded so now we can actually start building the recommender system.

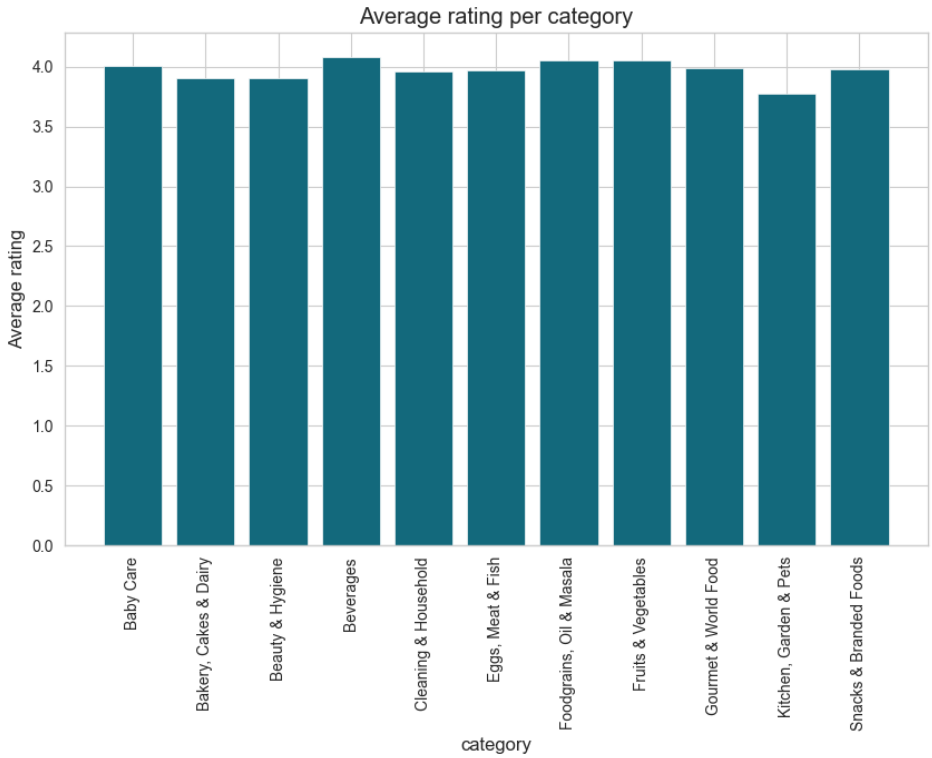
Let’s visually investigate the data a little bit.

What is the distribution for numerical values like?



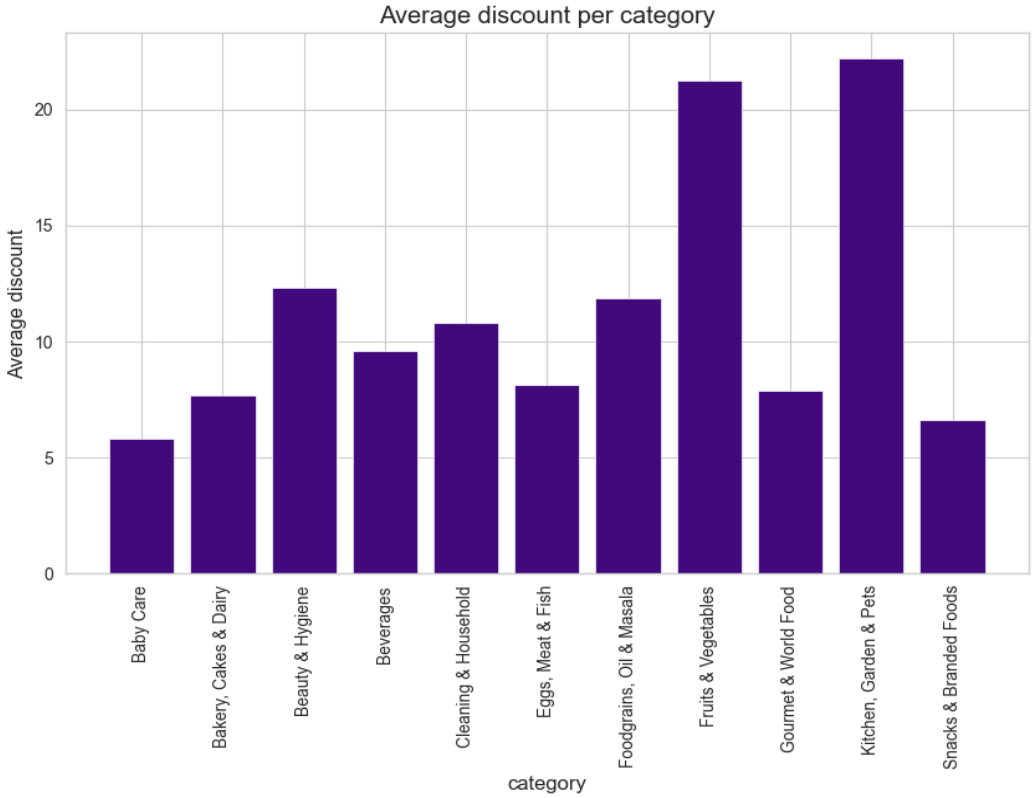
As it shows non of them are normally distributed. (sale_price and market_price are dropped)

What is the average rating per category?



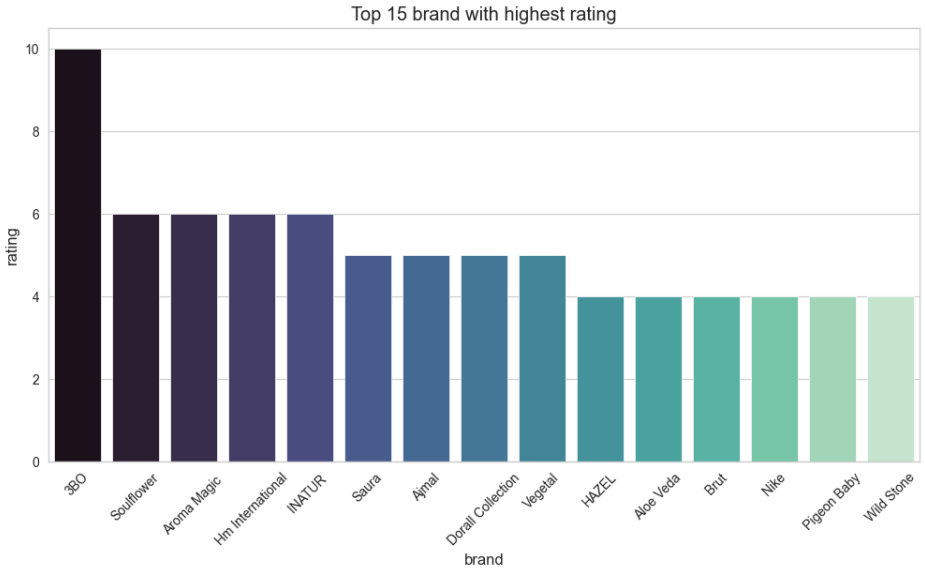
It shows that the average rating per category doesn't differentiate much.

What is the average discount per category?



Discount has the highest rate for kitchen, garden, and pets. Which seems logical duo to the fact that this category had the least amount of average rating.

What are the top 15 brands with the highest ratings?



What categories are these brands mostly related to?

['Beauty & Hygiene' 'Kitchen, Garden & Pets' 'Baby Care'
 'Cleaning & Household']

Now that the EDA process is finished let's again encode the categorical data we are going to keep which in this case is the only category that is going to be hot encoded. Now to use the description column of the dataframe it is needed to extract features from it using tf-idf method.

After obtaining the tfidf_matrix and merging it with the rest of the dataframe which contained encoded categories and numerical values (rating, discount), the cosine similarity matrix is calculated. (cos of the angel between two points in features space). Df['product'] is used as the key for mapping.
Then we created a function called recommend_product_based_on_click to recommend the top 9 similar items based on the similarity score calculated for a given product name.

For example here we try 'Onion Herbal Hair Growth Oil' product. And the result is:

20924	Bringha Oil
25142	Non-Sticky Hair Oil - Walnut & Almond
20598	Naturals Bhringraj Hair Oil
18860	Hair Oil - For Strong, Long & Thick Hair
17828	Hair Oil - Almond Drops
3779	Cold Pressed Avocado Carrier Oil
9803	Regrowth Hair Oil
26721	Hair Fruits Shining Black Conditioning Shampoo
20645	Vatika - Enriched Coconut with Hibiscus Hair Oil

Name: product, dtype: object

Collaborative-filtering recommender system

After loading the dataset. We observe that our dataframe looks like this:

	UserId	ProductId	Rating	Timestamp
0	A39HTATAQ9V7YF	0205616461	5.0	1369699200
1	A3JM6GV9MNOF9X	0558925278	3.0	1355443200
2	A1Z513UWSAAO0F	0558925278	5.0	1404691200
3	A1WMRR494NWEWV	0733001998	4.0	1382572800
4	A3IAAVS479H7M7	0737104473	1.0	1274227200

After checking the number of missing values and observing there are none, we begin to implement a model-based collaborative filtering. First step is to build the user-item utility matrix.

A part of it looks like this:

ProductId	0205616461	0558925278	0733001998	0737104473	0762451459	1304139212	1304139220	130414089X	130414643X	1304146537
UserId										
A00205921JHJK5X9LNP42	0	0	0	0	0	0	0	0	0	0
A00473363TJ8YSZ3YAGG9	0	0	0	0	0	0	0	0	0	0
A00700212KB3K0MVESPIY	0	0	0	0	0	0	0	0	0	0
A0081289HG0BXFQJQUWW	0	0	0	0	0	0	0	0	0	0
A01247753D6GFZD87MUV8	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
AZZTJQ7CQZUD8	0	0	0	0	0	0	0	0	0	0
AZZVCBG5G4EV8	0	0	0	0	0	0	0	0	0	0
AZZWJ3LICUEKJ	0	0	0	0	0	0	0	0	0	0
AZZWPNME0GQZ2	0	0	0	0	0	0	0	0	0	0
AZZZLM1E5JJ8C	0	0	0	0	0	0	0	0	0	0

91656 rows × 6384 columns

Each user's rating of a specific product is stored. And the empty ones left are filled with 0. Since there is a lot of extra data in here (empty zeros) it makes the data highly complex. A dimensionality reduction technique is needed. the most common collaborative-filtering method is SVD to decompose this matrix into (in here) 10 components. So only the most relevant data is kept.
Now after obtaining the correlation matrix from the decomposed matrix. For a given product id we can get its correlation score with other products. And filter the top 9 products which have a higher correlation than 0.90.
Example:
i = "0205616461"
Output:
['130414089X',
'4057362843',
'5357955832',
'5357956111',
'6162071103',
'6175005570',
'9745343412',
'9788071074',
'9788071554']