Open in app

Following

569K Followers

A Content-Based Recommender for E-Commerce Web Store

A case study of a simple algorithm with Matrix Multiplication and Cosine Similarity

Tan Pengshi Alvin Nov 16, 2020 · 8 min read

Open in app

Recommender systems have become ubiquitous in consumers' daily lives on the online platform, ranging from e-commerce, social media to news outlets. Our preferences and biases are super-charged by machine learning algorithms that learn from our taste, and recommend more of what we desire to see. Because recommender systems have such profound impact on our lives, and consequently on society, I decided to spend my last project of the Metis Data Science Bootcamp (Singapore, Batch 5) on creating a simple recommender algorithm for an e-commerce web store. Before we take off on my project, I would like to give you an overview on two general types of recommender systems:

Content-Based Filtering Recommender: As its name suggest, this type of recommender uses the similarity among the background information of the items or users to propose recommendations to users. For instance, if User A generally gives Sci-Fi movies a good rating, the recommender system would recommend more movies of the Sci-Fi genre to User A. In another instance, if User B is a male of the higher income group, a bank recommender system would likely label User B as a potential customer of the premium investment plan.

- **Pros**: Easy to overcome a cold start problem — when there is zero or few user-item interactions, the recommender is still able to provide good recommendations to the user.
- **Cons:** Requires the background information of items/users. Whenever there are new items/users, these background information has to be catalogued and added in.

Collaborative Filtering Recommender: This type of recommender identifies trends and patterns in previous and other user-item consumers of similar tastes together. For an instance, a news recommender detected that users who consume news favoring Donald Trump are also interested in news related to conspiracy theories. Hence, if there is a user newly interested in Trump news he will also be recommended news related to conspiracy theories.

- **Pros**: Exploits hidden correlations behind user-item interactions, and does not require extensive hand mapping and cataloging of data.
- **Cons**: Prone to cold start problem, and require existing reservoir of user-item interactions before being able to provide meaningful recommendations to existing user.

There are also hybrid recommender systems that incorporate both content-based filtering and collaborative filtering and achieved the best of both worlds. However, such recommender systems are typically advanced and require engineering a neural network.

For this project, due to a sparsity of data over a short timeframe, I decided to employ content-based filtering for recommending products on an e-commerce webstore, by customizing an algorithm that accounts for different categories of product information.

1. Data Transformation

I obtained about 110 million user-product interaction data of an e-commerce webstore over 2 months from Kaggle. Because of the enormity of the data, I decided to scale down the data to about 5 days, and filtering only 'computers' from the product category, which

Open in app

Search this file…

	event_time	event_type	product_id	category_code
0	2019-10-01 00:00:01 UTC	view	1307067	computers.notebook
1	2019-10-01 00:00:05 UTC	view	1480613	computers.desktop
2	2019-10-01 00:00:19 UTC	view	1306631	computers.notebook
3	2019-10-01 00:00:22 UTC	view	1480714	computers.desktop
4	2019-10-01 00:00:37 UTC	view	1701111	computers.peripherals.n
...	...	...	...	...
356720	2019-10-05 23:59:20 UTC	view	1480584	computers.desktop
356721	2019-10-05 23:59:40 UTC	view	1480584	computers.desktop
356722	2019-10-05 23:59:50 UTC	view	1480171	computers.desktop
356723	2019-10-05 23:59:55 UTC	view	1306556	computers.notebook

recommender model and has 3 categories: View, Cart and Purchase. I then proceed to provide a user score based on these user-item interactions — View: 1, Cart: 10, Purchase: 50. Furthermore, I also segregate items into 5 different price categories relative to their item categories. The assumptions in some of the models that are discussed later are that customers tend to shop in the lower-end or higher-end of products. The codes and resulting data frame are shown:

```
df['user_score'] = df['event_type'].map({'view':1,'cart':10,'purchase':
df['user_purchase'] = df['event_type'].apply(lambda x: 1 if x=='purchas
df['price_category'] = 0
for i in df['category_code'].unique():
    df.loc[df['category_code']==i,'price_category'] = pd.qcut(x=df['pri
```

Search this file…

	event_time	event_type	user_score	user_purchase	produ
0	2019-10-01 00:00:01 UTC	view	1	0	13070

3	2019-10-01 00:00:22 UTC	view	1	0	14807
4	2019-10-01 00:00:37 UTC	view	1	0	170111
...	...	...	...	...	...
356720	2019-10-05 23:59:20 UTC	view	1	0	14805
356721	2019-10-05 23:59:40 UTC	view	1	0	14805
356722	2019-10-05 23:59:50 UTC	view	1	0	14801
356723	2019-10-05 23:59:55 UTC	view	1	0	13065

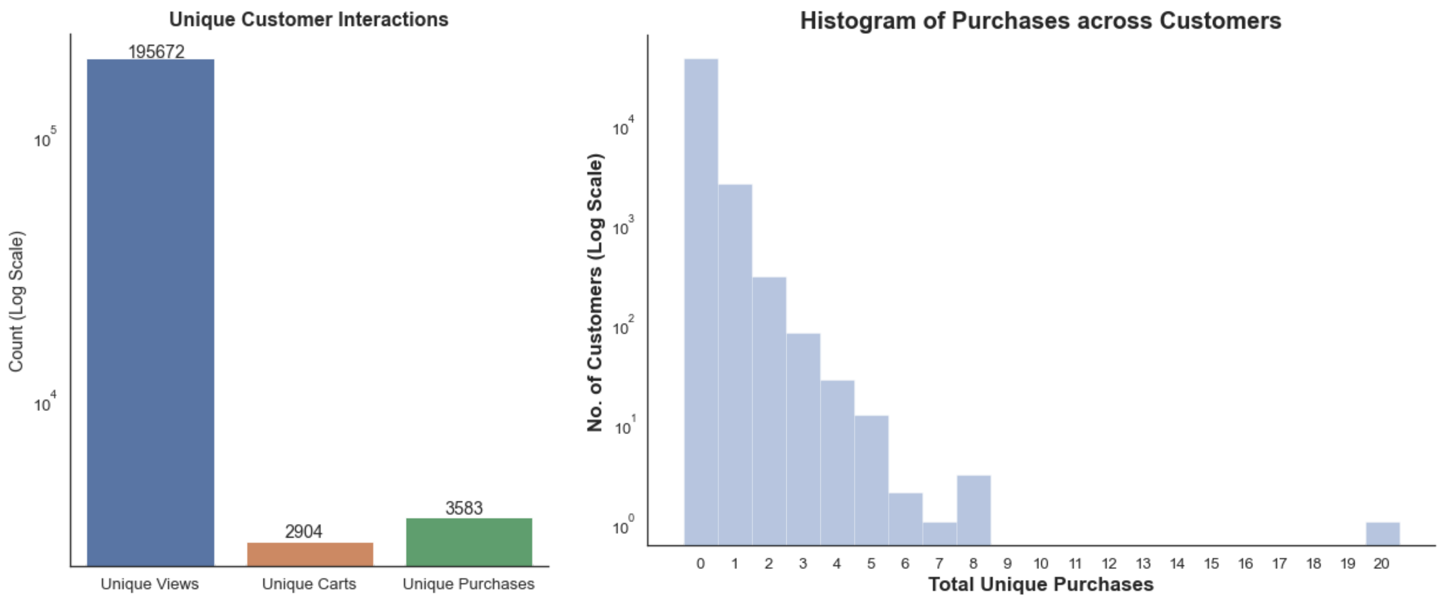
there could be multiple interactions per user per item. Therefore, I further perform a groupby operation to discover the sum of user scores for each unique user-item interaction. This forms the 'ratings' of the User-Item Matrix used for model building later. In addition, I apply MinMaxScaler to the user scores to obtain an interaction score with a value between 0 and 1. An interaction score of above 0.5 indicates a very high probability that a purchase has occurred, while no purchase occurs below the threshold of 0.5. The codes and resulting groupby data frame are shown:

```
group = df.groupby(['user_id','product_id'])['user_score','user_purchas
group['user_purchase'] = group['user_purchase'].apply(lambda x: 1 if x>
group['user_score'] = group['user_score'].apply(lambda x: 100 if x>100

std = MinMaxScaler(feature_range=(0.025, 1))
std.fit(group['user_score'].values.reshape(-1,1))
group['interaction_score'] = std.transform(group['user_score'].values.r

group = group.merge(df[['product_id','category_code','brand','price','p
```

Search this file…

	user_id	product_id	user_score	user_purchase	interaction_score	ca
0	269253210	1401933	1	0	0.025	co

4	512617890	1307285	1	0	0.025	co
...	...	...	...	...	...	...
195668	557140924	1306861	1	0	0.025	co
195669	557155858	9700243	1	0	0.025	co
195670	557162041	6600870	1	0	0.025	co
195671	557162041	30900907	1	0	0.025	co



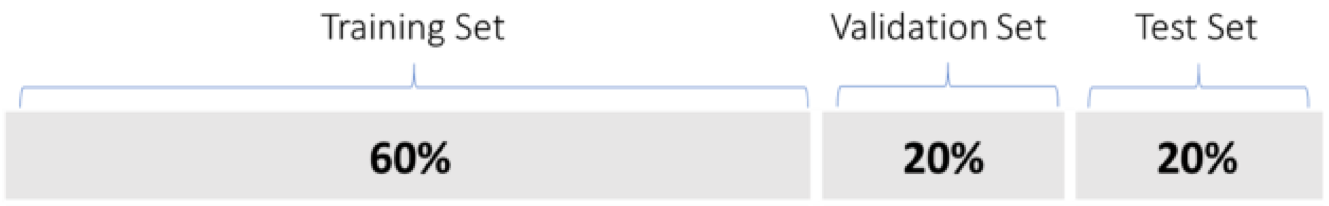
Dissecting the target (View/Cart/Purchase) of the data set with a bar plot and histogram (Image by Author)

Before model selection and development, I dive into data visualization of the target of the recommender system (View/Cart/Purchase). Unsurprisingly, the target is highly imbalanced, with about 1.8% view-to-purchase conversion. Moreover, a large majority of customers did not purchase anything at all, according to the histogram. This signals that it would be challenging for the recommender system to accurately predict purchase, as we shall see.

3. Model Selection

Open in app

paradigm:



Splitting of data set into 60% Training, 20% Validation and 20% Test. (Image by Author)

Subsequently, I transformed the training set into a sparse User-Item Matrix, filling the empty entries as zero for user score:

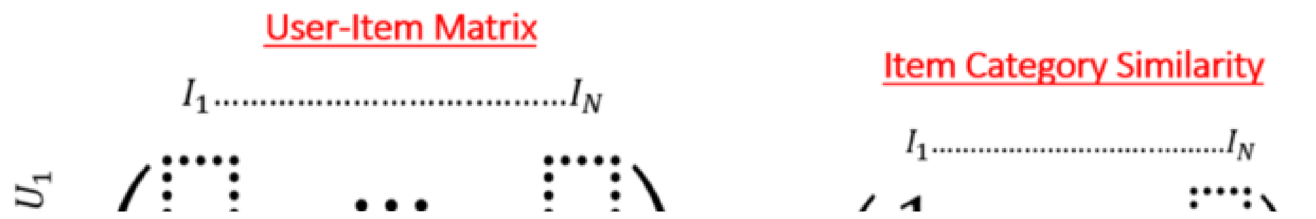
```
X_train_matrix = pd.pivot_table(X_train,values='user_score',index='user
X_train_matrix = X_train_matrix.fillna(0)
```

X_train.py hosted with ❤ by GitHub view raw

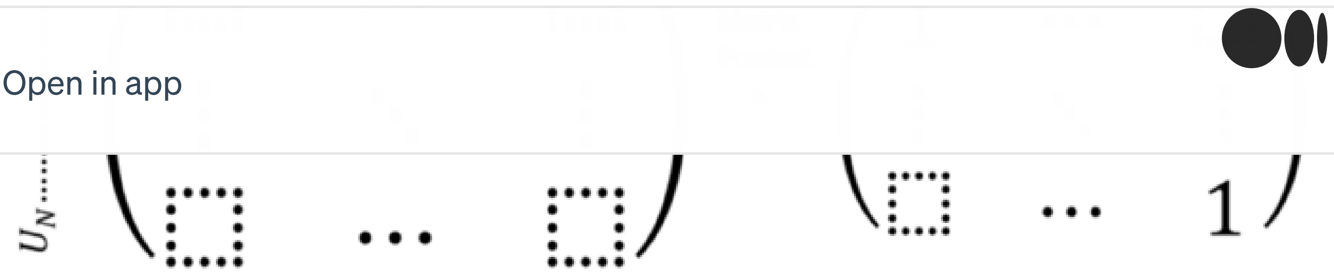
Now, there are 3 different content-based filtering models that I will consider:

- Filtering by item category
- Filtering by item category and price category
- Filtering by item category, price category and brand

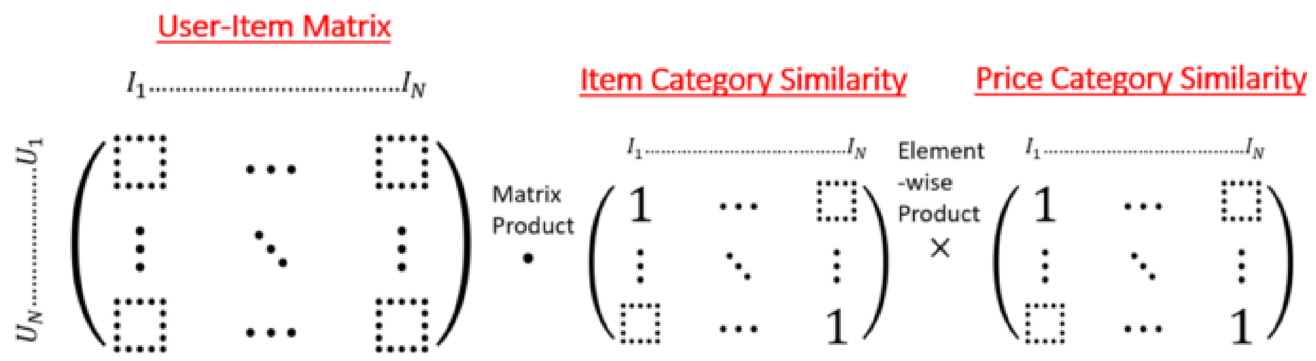
For each of the model, I will apply a matrix product between the User-Item Matrix and the Item-Item Similarity Matrix/Matrices according to the algorithm illustrated below:



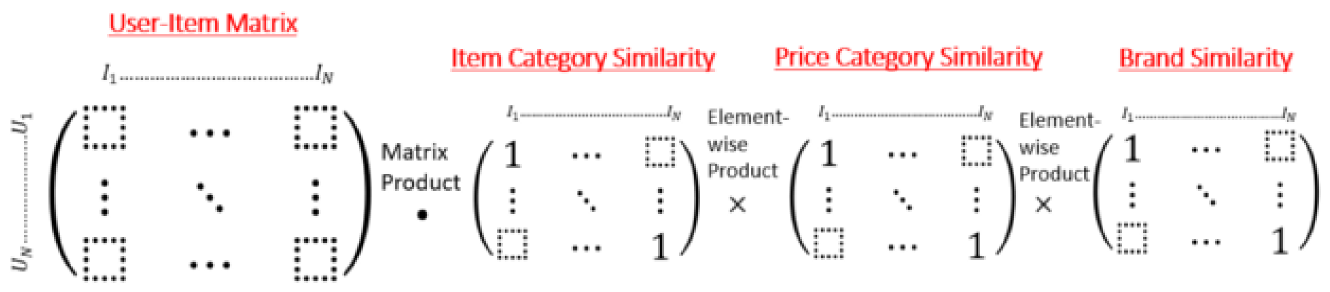
Open in app



Model 1: Filtering by item category (Image by Author)



Model 2: Filtering by item category and price category (Image by Author)



Model 3: Filtering by item category, price category and brand (Image by Author)

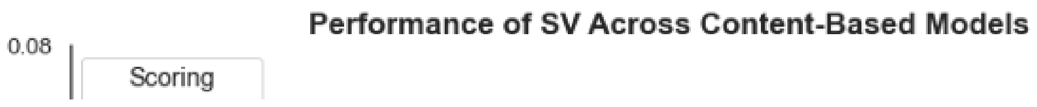
Once the matrix multiplication is completed, the MinMaxScaler is once again applied to obtain the trained User-Item Matrix of predicted interaction scores. For simplicity of illustration, I will demonstrate the codes for Model 3, as shown below:

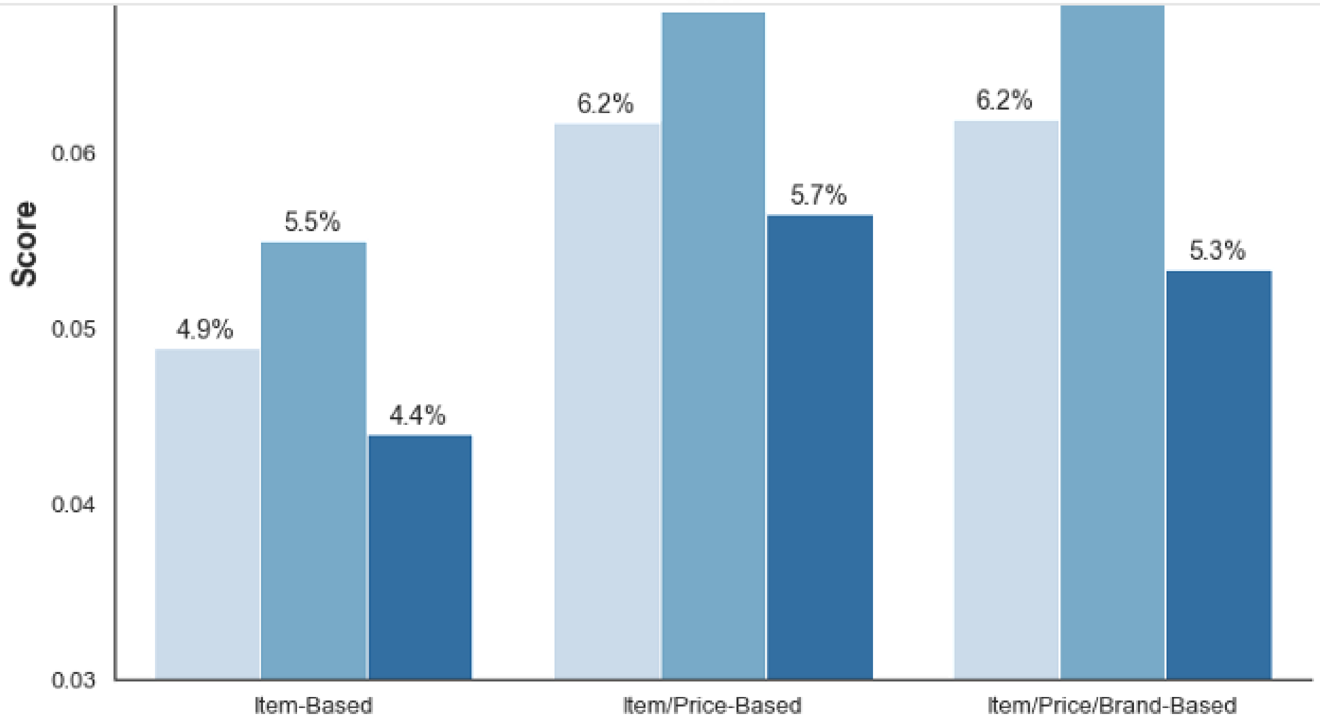
```
product_cat = X_train[['product_id','price_category','category_code','
product_cat = product_cat.sort_values(by='product_id')

price_cat_matrix = np.reciprocal(euclidean_distances(np.array(product_
euclidean_matrix = pd.DataFrame(price_cat_matrix,columns=product_cat['

cos_similar_matrix = pd.DataFrame(cosine_similarity(dt_matrix.values),

tfidf_vectorizer = TfidfVectorizer()
doc_term = tfidf_vectorizer.fit_transform(list(product_cat['brand']))
dt_matrix1 = pd.DataFrame(doc_term.toarray().round(3), index=[i for i
dt_matrix1 = dt_matrix1 + 0.01
cos_similar_matrix1 = pd.DataFrame(cosine_similarity(dt_matrix1.values

similarity_matrix = cos_similar_matrix.multiply(euclidean_matrix).mult
content_matrix = X_train_matrix.dot(similar_matrix)
std = MinMaxScaler(feature_range=(0, 1))
```

of predicted interaction scores above 0.5 are labelled as predicted purchase by customers.

```
content_matrix = pd.DataFrame(content_matrix,columns=sorted(X_train['pr
content_df = content_matrix.stack().reset_index()
content_df = content_df.rename(columns={'level_0':'user_id','level_1':'
X_valid = X_valid.merge(content_df,on=['user_id','product_id'])

X_valid['predicted_purchase'] = X_valid['predicted_interaction'].apply(
```

Now, comparing the actual purchase and predicted purchase by customer for the three models, Model 3 appear to be the most promising with highest precision, without sacrificing too much on Recall as compared with Model 2. Nonetheless, the precision and recall for all models are very low, as expected from my analysis in the EDA process.





Performance of 3 Content-Based Recommender Models during Simple Validation. (Image by Author)

4. Model Evaluation

Choosing Model 3 as the final model for my recommender system, I proceed to re-train both the Training and Validation data set together, and then evaluate on the Test data set, achieving a Recall of 5.0% and Precision of 7.8%.

Confusion Matrix of Test Set, showing both Recall and Precision. (Image by Author)

However, when I evaluate Precision and Recall based on number of unique purchases made, an interesting trend is noticeable. The Precision of the model is actually much higher for customers who have at least made one purchase! Hence, this means that when the model precision of the model is thus brought down by the false positive of predicted purchases of customers who eventually did not purchase.

Also, it is observed that Recall is much higher for customers who purchased more products. This could be because how model is measured is more biased towards purchasing customers. Hence, further tuning and adjusting could be done to raise the model's Recall.

5. Model Demonstration

In this section, I will demonstrate the user-item interactions of a particular customer (User ID: 518044530) in the Training/Validation data set and Test data set and also the potential recommendations that the model will provide for the customer. The following user-item interactions are captured in the Training/Validation data set:

Customer (User ID: 518044530) in the Training/Validation data set.

In the Test data set, we see that the model accurately predicted the user purchase. This is because in the Training/Validation data set, the customer had a high interaction score of a similar item.

Taking a random sample of the top recommendations of this customer (User ID: 518044530), we see that the model churned out products of similar category, price range and brand that he/she previously interacted with:

Random top 10 recommendations for Customer (User ID: 518044530)

6. Conclusion

project of Metis Data Science Bootcamp. Although the model is far from perfect, it showcases how a Content-Based Recommender can filter across more than one category of product information. In the future, ideally, a more effective hybrid recommender system could be attempted by building a deep neural network.

At the end of this 12-weeks bootcamp, I would really like to extend my heartfelt gratitude to my talented instructor Neo Han Wei, from whom I have learnt a great deal and also my fellow batch-mates especially Daniel Chang and Li Xinni who have lent moral and emotional support to make this learning journey possible.

Also, if you are interested, below are the links to the other 4 projects that I have painstakingly crafted during the bootcamp:

Project 4

Strategic Analysis of Trump Rallies with NLP and Time Series

A case study with Topic Modeling, Clustering and Time Series Prediction

towardsdatascience.com

Project 3

Predicting Satisfaction of Airline Passengers with Classification

A case study with KNN, Logistic Regression, Gaussian NB, Decision Trees and Random Forest.

towardsdatascience.com

Open in app

Predicting the Market Value of FIFA Soccer Players with Regression

A case study with Linear, LASSO, Ridge, Elastic Net and Polynomial Regression.

towardsdatascience.com

Project 1

MTA Turnstile Traffic Analysis to Optimize Street Engagements

A Simple Exploratory Data Analysis Project for Metis Data Science Bootcamp

towardsdatascience.com

> *Lastly, thank you very much for reading! Here is the link to my GitHub, which contains all the codes and presentation slides for this project. Also reach me on my LinkedIn or comment here below to discuss!*

Data Science Machine Learning Recommender Systems Ecommerce

Content Based Filtering

Open in app rite Help Legal

Get the Medium app

Download on the App Store GET IT ON Google Play