BUDT704: Data Processing and Analysis in Python

PROJECT REPORT

Oilst Product Recommendation System

Submitted By:

Jharna Dharaiya

Neha Sharma

Anirudh Murali

Shrushti Shah

Akshaya Krishnan

Sagar Khandade

Table of Contents

ABSTRACT

With the rise of YouTube, Amazon, Netflix and many other such web services in the past decade, recommender systems have taken more importance in our lives. From e-commerce suggestions to online optimized advertisements, recommender systems are today unavoidable in our daily online journeys. It helps us discover information and settle on choices where we do not have the required learning to judge a specific item. In this project, we are using Olist's store and customer data, the largest department store in Brazilian marketplaces. We use Collaborative Filtering method along with Single Value Decomposition to estimate product ratings for input customers and recommend top 5 most relevant products based on User-based filtering and then Item-based filtering.

INTRODUCTION

The internet has increasingly become the fastest and cheapest way to shop for anything, hence it's no surprise that retail websites took advantage of this new technology. Global e-commerce sales are expected to hit $5.5 trillion in 2022 and around 76% of U.S. adults shop online. The emergence and prevalence of COVID-19 has also had a significant impact on changing consumer purchasing habits, let it be luxury or basic necessities. A business today must have an internet presence to survive. Customers nowadays want individualized, tailored shopping experiences. Stores have always kept a track of sales data, income streams, and customer trends, so it makes sense that companies will now try to optimize the shopping experience by building recommendation engines to enable customers to get personalized product recommendations. It has been demonstrated that implementing personalized experiences online or in marketing campaigns has a significant impact on sales, with one study revealing that retailers leveraging advanced customization capabilities saw a 25% increase in sales. A recommendation system is a subclass of Information filtering Systems that seeks to predict the rating or the preference a user might give to an item. Simply put, it is an algorithm that suggests relevant potential items to users. Eg: In the case of e-commerce which product to buy.

DATA

The dataset has information of over 100,000 orders from the year 2016 to 2018 made at multiple marketplaces in Brazil at Olist's stores. The data obtained is real commercial data, and it has been anonymized in order to preserve privacy. The data allows viewing a single order from multiple dimensions: from the order status, price of products, payment and freight performance to customer location, product attributes and finally review ratings written by customers.

This dataset was generously provided by Olist, the largest department store in Brazilian marketplaces. Olist connects small businesses from all over Brazil to channels without hassle and with a single contract in order to simplify the process. Merchants can sell their products through the Olist Store and ship them directly to the customers using Olist logistics partners.

Hence, the data is extremely relevant for a model like a recommendation system.

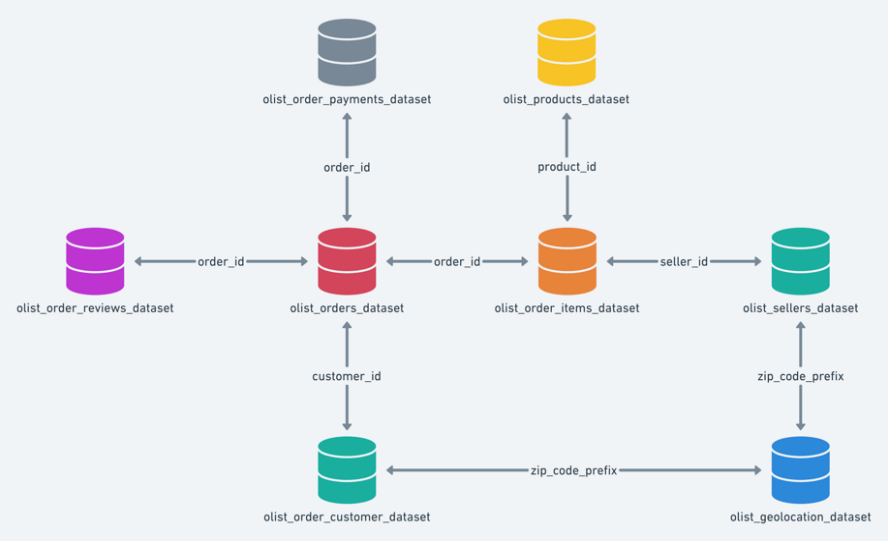


Image Reference: Brazilian E-Commerce Public Dataset by Olist

The link to the dataset:
https://www.kaggle.com/datasets/olistbr/brazilianecommerce?select=olist_order_items_dataset.csv

TYPES OF RECOMMENDATION SYSTEMS

Content Based Filtering

Content-based filtering is used to generate recommendations based on the user's previous actions and explicit feedback. Content-based filtering matches keywords and attributes assigned to objects in a database to a user profile to make recommendations. The level of similarity between items or products is established based on attributes of items liked by the user.

Advantages:

Content Based algorithms are the most useful for making recommendations when the data ratings in the system are insufficient. This is due to the possibility that the user has rated other things with comparable attributes. Therefore, despite the lack of data, the model should be able to produce suggestions by leveraging both, the ratings and the item properties.

Disadvantages:

There is no diversity in the recommendations provided by this model. This is a disadvantage because, if a user has never interacted with a particular category of products, that item will never be recommended. They are also ineffective for giving recommendations for new users. To produce reliable predictions without overfitting, it is important to have a large dataset of ratings available.

Collaborative Filtering

Collaborative filtering is the process of predicting the interests of a user by identifying preferences and information from multiple users. This is accomplished by applying techniques involving collaboration among numerous agents, data sources, etc. to filter data for information or patterns. This is based on the assumption that customers A and B are likely to have similar tastes in other products if they have similar tastes in one product.

Types of Collaborative Filtering:

Memory-based method – It is also often known as neighborhood collaborative filtering. In essence, the rating of user-item pairs is predicted based on the neighborhood. It indicates that users who share your views will produce solid and consistent suggestions.

Model Based Approach – Machine learning is used in this approach. Features linked with the dataset are parameterized as model inputs and decision trees, rule-based approaches, latent factor models etc are used for predicting values.

Advantages:

The main advantage of collaborative filtering models are their ease of use and high level of coverage. It is also useful because it captures even the nuanced characteristics and does not require the understanding of the item.

Disadvantages:

Collaborative filtering models are not friendly for recommending new items to users because there has been no interaction between the item and the user.

MODEL & APPROACH

Step 1: Data Exploration

Most of the data from the dataset was already cleaned and processed, hence we directly proceed to exploring the data and drawing insights.

```
# Checking counts of different order statuses
orders.order_status.value_counts()
```

```
delivered      96478
shipped         1107
canceled         625
unavailable      609
invoiced         314
processing       301
created            5
approved           2
Name: order_status, dtype: int64
```

Image Reference: Jupyter Notebook

As we can see from the value counts above, there are order statuses which include 'canceled', 'unavailable', 'processing', 'created' and 'approved'. Therefore, we only choose ratings for products which have been 'delivered', 'shipped' or 'invoiced' as we have a good level of certainty that the products in those orders have reached the customer for them to give a fair review.

We also check what are the average number of products being sold in each order, which is approximately 1 product per order. This number is obviously very low, and it is hypothesized that it is due to the fact that customers are not being recommended the relevant products according to their tastes and preferences. Hence, a recommendation system will be even more impactful in order to improve the cross-selling rate.

The number of customers who made a repeat purchase is also very low, around 3.19% came back and made a purchase on the website, out of the 94,000+ customers who have shopped with Olist.

A recommendation system can improve both the re-purchase and cross-selling rate for Olist by introducing new customers to products they are more likely to buy, based on their rating history and purchase history.

```
average_products_per_order = product_ratings_valid.g
    'product_id').product_id.mean()
average_products_per_order
```

```
1.1472425467979788
```

```
product_ratings_valid.groupby('order_id')['product_i
    kind='bar', use_index=True)
```

```
<AxesSubplot:>
```

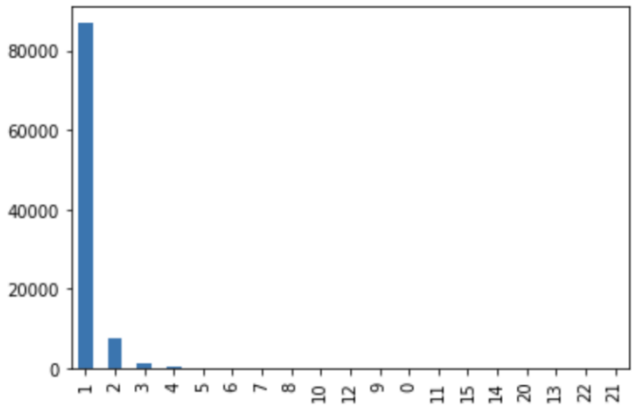


Image Reference: Jupyter Notebook

Step 2: Build Scoring Matrix of products vs customers for Item-based and User-based Collaborative Filtering Model

```
# Build the scoring matrix of products vs customers for Item-based and User-based Collaborative Filtering Model

rmat = product_ratings_valid.groupby(['product_id', 'customer_unique_id'])['review_score'].max().unstack().fillna(0)
rmat_index = rmat.index
rmat_T_index = rmat.T.index
rmat_T_columns = rmat.T.columns
rmat_T = scipy.sparse.csr_matrix(rmat.T.values)
rmat = scipy.sparse.csr_matrix(rmat.values)

print('rmat done!')
```

Image Reference: Jupyter Notebook

Step 3: Compute the cosine similarity matrices for both scoring matrices.

```
#Computing the cosine similarity matrix
cosine_sim = cosine_similarity(rmat)
cosine_sim_T = scipy.sparse.csr_matrix(cosine_similarity(rmat_T))

cosine_sim = pd.DataFrame(cosine_sim, index=rmat_index, columns=rmat_index)
cosine_sim_T = pd.DataFrame(cosine_sim_T, index=rmat_T_index, columns=rmat_T_index)

print('cosine done!')
```

Image Reference: Jupyter Notebook

Step 4: Train Test Split. Keeping aside 30% of the data for test set. Cross validating SVD model.

```
# Collaborative filtering model
reader = Reader()
data = Dataset.load_from_df(product_ratings_valid[['customer_unique_id', 'product_id', 'review_score']], reader)

print('data loaded!')

# split data into train test
trainset, testset = train_test_split(data, test_size=0.3)

print('splitting done!')

# train
svd = SVD()
svd.fit(trainset)

print('training done!')

# run the trained model against the testset
test_predictions = svd.test(testset)

print('prediction done!')

# cross-validate
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

Image Reference: Jupyter Notebook

Step 5: Generate Product Recommendations and Estimated Ratings

```
def collab_filter_model(customer_id, product_id, n_recs, df, cosine_sim, consine_sim_T, svd_model, rmat_T, rmat_T_index):
    '''
    This function represents a hybrid recommendation system, it will have the following flow:
        1. Use an item-based cosine similarity to compute the 50 most similar products
        2. Use a user-based cosine similarity to compute the 10 most similar customer Single Value Decomposition (SVD)
        4. Compute the predicted rating for products found through similar customers
        4. Return the top n products with the highest predicted rating

    params:
        customer_id (Integer) : The customer_id
        product_id (Integer) : The product_id
        n_recs (Integer) : The number of recommendations to be given
        df (DataFrame) : Original dataframe with all product information
        cosine_sim (DataFrame) : The cosine similarity dataframe
        cosine_sim_T (DataFrame) : The transpose of cosine similarity dataframe
        svd_model (Model) : SVD model
        rmat (Sparse Array) : Users vs Products rating matrix (User-based)
        rmat_columns (DataFrame Index) : Columns of rating matrix
    '''

    # find 20 most similar customers
    top_20_idx = []
    for left, right in zip(consine_sim_T.indptr[:-1], consine_sim_T.indptr[1:]):
        n_row_pick = min(20, right - left)
        top_20_idx.append(consine_sim_T.indices[left + np.argpartition(
            consine_sim_T.data[left:right], -n_row_pick)[-n_row_pick:]])

    # transforms result from sparse matrix into a dict as {matrix_index: [user_1, user_2,..., user_5]}
    similar_customers_dict = {}
    for idx, val in enumerate(top_20_idx):
            similar_customers_dict.update({idx: val.tolist()})

    # gets actual user ids from data based on sparse matrix position index
    similar_customers_final = {}
    for customer, similar_customers in similar_customers_final.items():
        idx = rmat_T.index[customer]
        values = []
        for value in similar_customers:
            values.append(rmat_T.index[value])

        similar_customers_final.update({idx: values})

    # transforms list of users: [similar_user1, similar_user2] into list of user: [job1, job2]
    customer_products = {}
    for customer, similar_users in similar_customers_final.items():
        # remove the user itself from similar_users (since cos_sim(user_1, user_1) is 1)
        try:
            del similar_users[similar_users.index(customer)]
        except:
            pass
        # get movie ids from list of movies rated by similar users.
        # also apply extra logic to get the most high rated movies from the similar users
        product_recs = ratings[(ratings['user_id'].isin(similar_users)) & ratings['rating']>=3]
        if movies_rec.empty:
            product_recs = ratings[(ratings['user_id'].isin(similar_users)) & ratings['rating']>=2]
        if movies_rec.empty:
            product_recs = ratings[(ratings['user_id'].isin(similar_users)) & ratings['rating']>=1]
        product_samples = product_recs.sample(n=10, random_state=33)['product_id'].values
        customer_products.update({customer: list(set(product_samples))})

    # transform dictionary into list of tuples and save on DataFrame
    customer_product_tuple = [(customer, products) for customer, products in customer_products.items(
    ) for product in customer_products]
    customer_product_df = pd.DataFrame(customer_product_tuple, columns=['customer_id', 'product_id'])

    # sort similarity values in decreasing order and take top 50 results
    sim = cosine_sim[cosine_sim.index.isin(
        customer_product_df.product_id)][customer_product_df.product_id.to_list()]
    sim = list(enumerate(cosine_sim[product_id]))
    sim = sorted(sim, key=lambda x: x[1], reverse=True)
    sim = sim[1:50]

    # get product metadata
    product_idx = [i[0] for i in sim]
    products = df.iloc[product_idx][['product_id', 'product_category_name', 'review_score',
                                     'product_category_name_english']]

    # predict using the svd_model
    products['est'] = products.apply(lambda x: svd_model.predict(
        customer_id, x['product_id'], x['review_score']).est, axis = 1)

    # sort predictions in decreasing order and return top n_recs
    products = products.sort_values('est', ascending=False)

    return products.head(n_recs)
```

Image Reference: Jupyter Notebook

We use the top 20 most similar customers to the user and make a list of the products that they have purchased. We then find the most similar products in that list and predict the estimated rating the user would give it. Based on that, we recommend the top 5 products with the highest estimated ratings.

RESULTS

We use Root-mean-square deviation (RSME) and Mean absolute error (MAE) to analyze the results obtained for our model run across the 5 folds.

Overall, the model shows surprisingly low RSME and MAE values. While they are not the most desirable scores, as more and more customers start rating products, we can expect the RSME and MAE numbers to improve even further to better predict estimated ratings and provide the most relevant recommendations.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
test_rmse	1.240159	1.250068	1.260422	1.250892	1.237536
test_mae	0.971592	0.981695	0.988680	0.982309	0.972559
fit_time	1.264764	1.287470	1.282062	1.292372	1.257900
test_time	0.106308	0.103470	0.226109	0.219415	0.221575

Image Reference: Jupyter Notebook

	Product ID	Product Category	Estimated Score
0	d1c427060a0f73f6b889a5c7c61f2ac4	computers_accessories	4.522296
1	52c80cedd4e90108bf4fa6a206ef6b03	garden_tools	4.461156
2	25e2023ed83352bde98dc1490d14c3d8	toys	4.453456
3	e2bab04d6a471ed7a3629fbe2a64b55a	perfumery	4.381897
4	61a4100ccd6d9c4c808a1fd954ddb8ad	small_appliances	4.306061

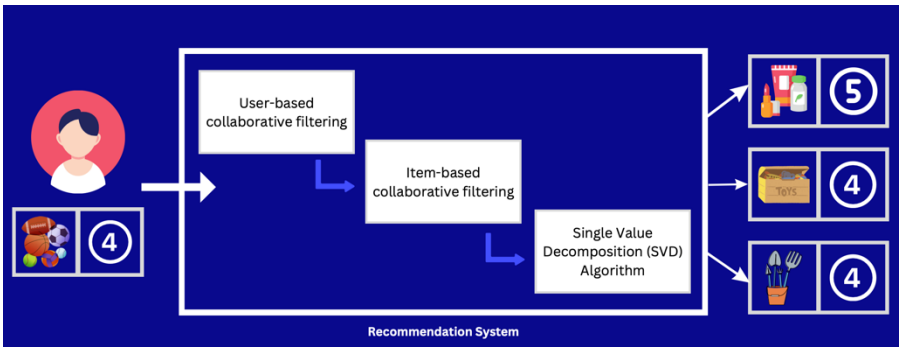
Image Reference: Jupyter Notebook

The products being recommended are from various categories, which will improve the rate of cross-selling between different categories, while still providing relevant items as liked by customers similar to that of the input user.

CONCLUSION

In conclusion, the model is a simplified yet effective version of how a real-life recommendation system would work for big datasets. While there are limitations like scarcity of ratings data and the issue of items without ratings not being recommended, there is a lot of scope for future

improvements like use of Hybrid models that incorporate Content based filtering and optimization of prices based on user similarities. Hence, the recommendation system is a powerful tool that can help Olist provide a better and enjoyable consumer experience while boosting sales and customer retention.



REFERENCES

1. Brazilian E-Commerce Public Dataset by Olist - https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce?datasetId=55151&searchQuery=recommendation&select=olist_products_dataset.csv
2. Recommendation Systems Explained - https://towardsdatascience.com/recommendation-systems-explained-a42fc60591ed
3. How to Build Simple Recommender Systems in Python - https://medium.com/swlh/how-to-build-simple-recommender-systems-in-python-647e5bcd78bd
4. Building a Product Recommendation System with Collaborative Filtering - https://www.datasource.ai/uploads/6b86b1630562b323a26143f90d97fe08.html