The problem statement involves recommendation based on relationship or interaction between CustomerIds and matching Products.

* Users are the txn.CustomerID
* Items are products.Product

**Data cleaning**

Df.na().size() shows that the orderdate is np.nan – so can be dropped

Also emailed is no use – we already have the CUSTOMERID

*dfTxn.drop(columns=['CustomerEmailAddress','OrderDate'], inplace=True)*

**Exploratory data analysis**

1. The data is evenly spread across men and women gender

Female 876675

Male 877840

1. The customer age is spread between 20 to 48 with mean at 34 years and a standard deviation of 8.6 years

count 1.754515e+06

mean 3.399160e+01

std 8.649210e+00

min 2.000000e+01

25% 2.600000e+01

50% 3.400000e+01

75% 4.200000e+01

max 4.800000e+01

Based on the groups of [Age category (considering we divide the age groups into bins] there is a data is not evenly spread which means Age will play a role in recommendation.

If we add the gender-based spread to each age categories – the spread doesn’t change. Which means Gender is not a decisive factor in this dataset.

Chart, bar chart

Description automatically generated

Chart, bar chart

Description automatically generated

1. Customer device types are almost uniformly spread out

Android 440655

Linux 440730

Windows 436535

iOS 436595

1. Payment mode is also evenly distributed

credit\_card 350428

debit\_card 351549

net\_banking 351181

upi 350435

wallet 350922

1. Most widely transaction is for “USB Stick” 43.7k,

Mean does not change much if Usb Stick is excluded

Mean without USB Stick = 21933

Mean with USB Stick = 22209

So, we USB Stick cannot be an outlier

1. Most products are priced around 5K than other price tags

Chart, bar chart

Description automatically generated

**Collaborative Filtering method**

The Collaborative filtering method for recommender systems is a method that is solely based on the past interactions that have been recorded between users and items, in order to produce new recommendations. Collaborative Filtering tends to find what similar users would like and the recommendations to be provided and in order to classify the users into clusters of similar types and recommend each user according to the preference of its cluster.

How do we give a Score or Interaction Score to each Product with respect to a CustomerId. Customer might have transacted with the product multiple times so that has to taken into account as well.

In this dataset - a product has 4 different actions or relation with a customer–

['Item Returned', 'Payment Incomplete', 'Order Cancelled', 'Delivered']

We will try to give weightages to each type of relation

It makes sense that we include only the DELIVERED products as basis for recommending but if we go by our personal buying pattern we might like a Product – add to cart and then due to some reason – maybe higher price, not immediately needed, poor recent feedbacks we decide to not go ahead and cancel the order.

We might Cancel an order after buying - based on other reviews that came later or maybe because we don’t need it anymore or there was an error in delivery.

Payment incomplete maybe be based on some technical error in the payment gateway or some backend error of the ecommerce website. But the customer had liked the item. He/She might retry.

Items that are returned are basically because customer did not like it and they should not be kept in the recommendation and given the lowest weightage or maybe negative weightage.

Based on this understanding our weightages can be like this –

* Delivered = 50
* Payment Incomplete = 10
* Order Cancelled = 1
* Item returned = 0

So, if a customer C1 has the following set of txns –

P1 – delivered

P2 – payment incomplete

P2 – payment incomplete

P2 – delivered

P3 – order cancelled

P4 – delivered

P5 – np.nan (no interaction)

P4 – item returned

The net score of C1 with P1 is 50

The net score of C1 with P2 is (10 + 10 + 50) = 70

The net score of C1 with P3 is 1

The net score of C1 with P4 is 50 + 0

The net score of C1 with P5 is 0 (replacing. np.nan with 0)

So the customer C1 can be represented as a vector [P1\_score, P2\_score, P3\_score, P4\_score, P5\_score], considering there are only 5 different products

C1 = [50, 70, 1, 50, 0]

Similarly, we would like to vectorize all other customers – where each product becomes a feature or an attribute of the customer. The weight of each attribute/feature will be the price interaction score mentioned above.

I have tried another run with an Age based score as well

Once we have vectors ready with the net score or interaction score or order status score – we can use NearestNeighbour algorithm to group or cluster together all the customers.

We will use sklearn.neighbors.NearestNeighbors which implements an unsupervised nearest neighbors learning

Hyper-parameters used are –

* Algorithm = “brute”
* n\_neighbours = 5,
* metric = “cosine”

These are the hyperparameters of the model. I have tried to change the hyper parameters to use n\_neighbours = 3. But the similarity percentage did not show any significant improvement.

I tried using the “pearson” distance as metrics but it would involve measuring Euclidean distance between the vectors. In our problem statement – it is the orientation of the customer that is important rather the degree or intensity of orientation.

The metrics for measuring similarity is “cosine”. Cosine similarity is is measured by the cosine of the angle between two vectors and determines whether two vectors are going in the similar direction. If you see cosine on a right angle, ie 90 degree is 0 and cosine of 0 degree is 1 , which means if two vectors are pointed in the same direction – the angle between between them will be nearly 0 and hence similarity will nearly 1 (which is nothing like cosine of the angle between the two vectors)

Once a customer comes on board – we should be able to give him top 5-10 recommendations.

We find the vectors of the all the nearest customers and then average it out. This will give a vector whose scalar components are averaged out based on all the nearest neghbours. We can order the vector’s dimension based on the interaction score (descending=true) and take the top 10 indices.

Confidence on the model can be measured by cosine similarity which will range from 0 to 1. Similarity close to 1 will mean we are getting a higher confidence score. Similarity close to 0 will mean we are getting a lower confidence score.

The only with collaborative filtering is that the prediction of the model for a given {**customerId🡪Product**} pair is the dot product of the corresponding embeddings. So, if an **customerId** is not seen during training, the system cannot generally create an embedding for it and hence cannot query the model with this **customerId**. This issue is known as the **cold-start problem.**

To address the “cold start” problem I will use the **Content based filtering mechanism** – it will give the top N recommendations from the existing txns that have already happened. Any new customer can be provided this recommendation or any customer who is searching for a product – we will be able to provide a set of 10 recommendations of similar products that people have transacted in the past. Here also the analysis is done once with the Interaction Score and then based on Age.

Some examples are provided in the report.txt