Week 5 - Customer Behavior Detection

Zaid Alshanti

zaisahanti@yahoo.com

This project features importing data from sql server database and performing analysis techniques including : IDF, Calculate Pairwise Similarity (Cosine Similarity) ,Build Graph from Similarity ,Community Detection (Louvain Method) for customers based on the items they had purchased. The project outcomes are highly associated with the Marketing, Customer Service and Sales departments.

Data

Shopping_behavior_updated.csv from kaggle, the dataset contains 17 features and 3900 observations, the features are shown and described in Table 1.1 below:

Feature	Type	Description
Customer ID	Nominal	Customers ID numbers
Age	Ratio	Age of customers
Gender	Nominal	Male or Female
Item Purchased	Nominal	What Item The customer purchased
Purchase Amount (USD)	Interval	How much the customer paid
Location	Nominal	Where did the purchase occur
Size	Ordinal	What size the customer purchased
Color	Nominal	What color the customer purchased
Season	Nominal	When did the purchase occur
Review Rating	Interval	The customer rating for the purchase process
Subscription Status	Nominal	Is the customer Subscribed?
Shipping Type	Nominal	What shipping type the customer choose
Discount Applied	Nominal	Was a discount applied?
Promo Code Used	Nominal	Was a promo code used?
Previous Purchases	Ratio	How much the customer purchased before
Payment Method	Nominal	What payment type was used
Frequency of Purchases	Ordinal	How frequent the customer purchase

Table 1.1 - Dataset features

The data was uploaded into a SQL server the server was connected into the Jupyter server as shown in figure 1.1 below:

```
In 3
conn = obdc.connect(
    r"DRIVER={ODBC Driver 17 for SQL Server};"
    r"SERVER=(localdb)\MSSQLLocalDB;"
    r"DATABASE=tempdb;"
    r"Trusted_Connection=yes;"
)
print("Connected to LocalDB!")

Executed at 2025.08.21 08:46:37 in 196ms

Connected to LocalDB!
```

Figure 1.1

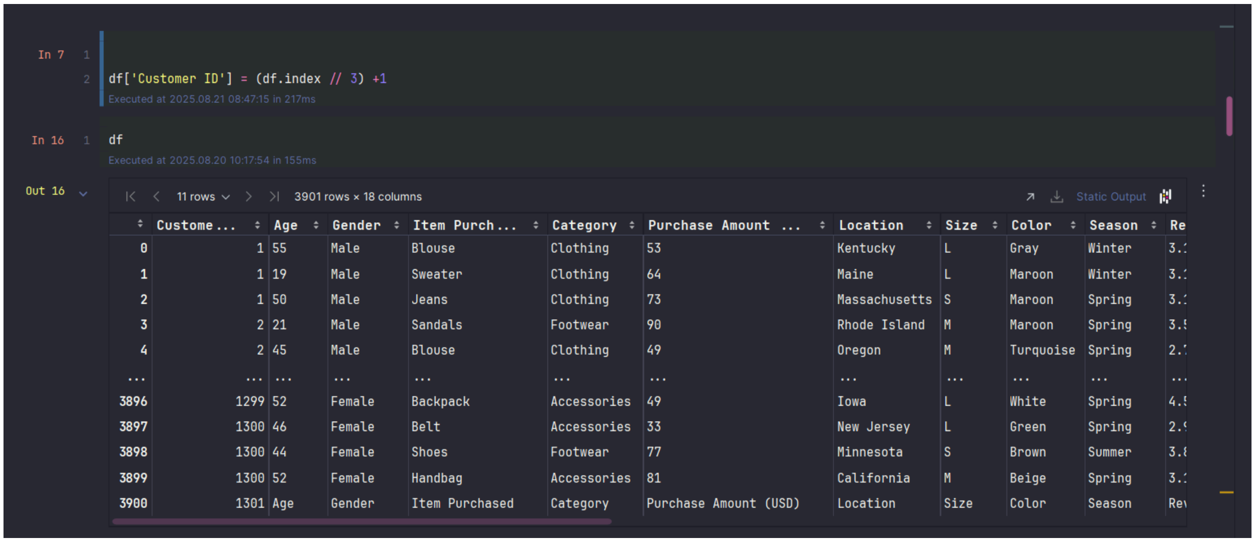
Two features were extracted : Customer ID and Item Purchased as shown in figure 1.2 below :

```
In 88
customer_product = pd.crosstab(df["Customer ID"], df["Item Purchased"])
Executed at 2025.08.21 10:12:46 in 161ms

In 111
customer_product
Executed at 2025.08.21 13:43:42 in 81ms

Out 111
11 rows    1301 rows × 26 columns    Static Output
```

Item Purchased	Backpack	Belt	Blouse	Boots	Coat	Dress	Gloves	Handbag	Hat	Hoodie	...	Scar
1	0	0	1	0	0	0	0	0	0	0	...	
2	0	0	1	0	0	0	0	0	0	0	...	
3	0	0	0	0	1	0	0	0	0	0	...	
4	0	0	0	0	0	0	0	1	0	0	...	
5	0	0	0	0	2	1	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1297	0	0	0	0	0	1	0	0	0	0	...	
1298	0	0	0	0	0	1	0	0	1	0	...	
1299	1	0	0	0	0	0	0	0	0	1	...	
1300	0	1	0	0	0	0	0	1	0	0	...	
1301	0	0	0	0	0	0	0	0	0	0	...	

Figure 1.2

Customer IDs were divided on three since there are no duplications in the original feature , this step will ensure the capability of community division for the customers, as shown in figure 1.3 below :



In 7

```
df['Customer ID'] = (df.index // 3) +1
```

Executed at 2025.08.21 08:47:15 in 217ms

In 16

```
df
```

Executed at 2025.08.20 10:17:54 in 155ms

Out 16

11 rows 3901 rows × 18 columns Static Output

	Custome ...	Age	Gender	Item Purch ...	Category	Purchase Amount ...	Location	Size	Color	Season	Re
0	1	55	Male	Blouse	Clothing	53	Kentucky	L	Gray	Winter	3.
1	1	19	Male	Sweater	Clothing	64	Maine	L	Maroon	Winter	3.
2	1	50	Male	Jeans	Clothing	73	Massachusetts	S	Maroon	Spring	3.
3	2	21	Male	Sandals	Footwear	90	Rhode Island	M	Maroon	Spring	3.
4	2	45	Male	Blouse	Clothing	49	Oregon	M	Turquoise	Spring	2.
...	...	...	...	...	...	...	...	...	...	...	...
3896	1299	52	Female	Backpack	Accessories	49	Iowa	L	White	Spring	4.
3897	1300	46	Female	Belt	Accessories	33	New Jersey	L	Green	Spring	2.
3898	1300	44	Female	Shoes	Footwear	77	Minnesota	S	Brown	Summer	3.
3899	1300	52	Female	Handbag	Accessories	81	California	M	Beige	Spring	3.
3900	1301	Age	Gender	Item Purchased	Category	Purchase Amount (USD)	Location	Size	Color	Season	Re

Figure 1.3

Modeling

IDF: Inverse Document Frequency,Words unique to a small percentage of documents (e.g., technical jargon terms) receive higher importance values than words common across all documents

$$IDF \ = \ log(\frac{number\ of\ documents\ in\ the\ corpus}{number\ of\ documents\ containing\ the\ term\ in\ the\ corpus}).$$

IDF was applied as shown in figure 2.1 below:

```python
N = customer_product.shape[0]
df_count = (customer_product > 0).sum(axis=0)
idf = np.log((N + 1) / (df_count + 1)) + 1
idf
```

Executed at 2025.08.21 10:13:10 in 30ms

```
Shirt           3.102753
Shoes           3.188050
Shorts          3.147776
Skirt           3.167711
Sneakers        3.208812
Socks           3.161022
Sunglasses      3.096483
Sweater         3.084060
T-shirt         3.201844
dtype: float64
```

Figure 2.1

After calculating the IDF, the weight for each community was calculated by multiplying IDF with the customer_product dataframe in figure 2.2 :

```python
tfidf = customer_product * idf
```

Executed at 2025.08.21 10:13:19 in 63ms

Figure 2.2

Cosine Similarity

Cosine similarity was calculated using the observations weights as shown in figure 2.3 below :

```
In 112
similarity_matrix = cosine_similarity(tfidf)
similarity_matrix
Executed at 2025.08.21 13:46:19 in 89ms

Out 112
         0.        ],
        [0.        , 0.        , 1.        , ..., 0.        , 0.        ,
         0.        ],
        ...,
        [0.        , 0.        , 0.        , ..., 1.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 1.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         1.        ]])
```

Figure 2.3

Since the objective is community detection , a graph must be constructed as shown in figure 2.4 below:

```
In 115
threshold = 0.7
for i, c1 in enumerate(customer_product.index):
    for j, c2 in enumerate(customer_product.index):
        if i < j and similarity_matrix[i, j] > threshold:
            G.add_edge(c1, c2, weight=similarity_matrix[i, j])
Executed at 2025.08.21 13:48:42 in 1s 432ms
```

Figure 2.4

Lastly, Louvain Community Detection algorithm is applied to the created graphs as shown in figure 2.5 below :

```
In 116
partition = community_louvain.best_partition(G, weight='weight')

print("\nCommunity Assignments:")
for customer, community_id in partition.items():
    print(f"{customer} → Community {community_id}")
Executed at 2025.08.21 13:48:47 in 286ms

1292 → Community 1
1293 → Community 34
1294 → Community 10
1295 → Community 6
1296 → Community 11
1297 → Community 12
1298 → Community 38
1299 → Community 5
1300 → Community 17
1301 → Community 35
```

Figure 2.5

Evaluation

After using Modularity score evaluation , the model scored approximately 0.79, which indicates clearly divided communities, the communities graphs are visualized in figure 3.1 below :

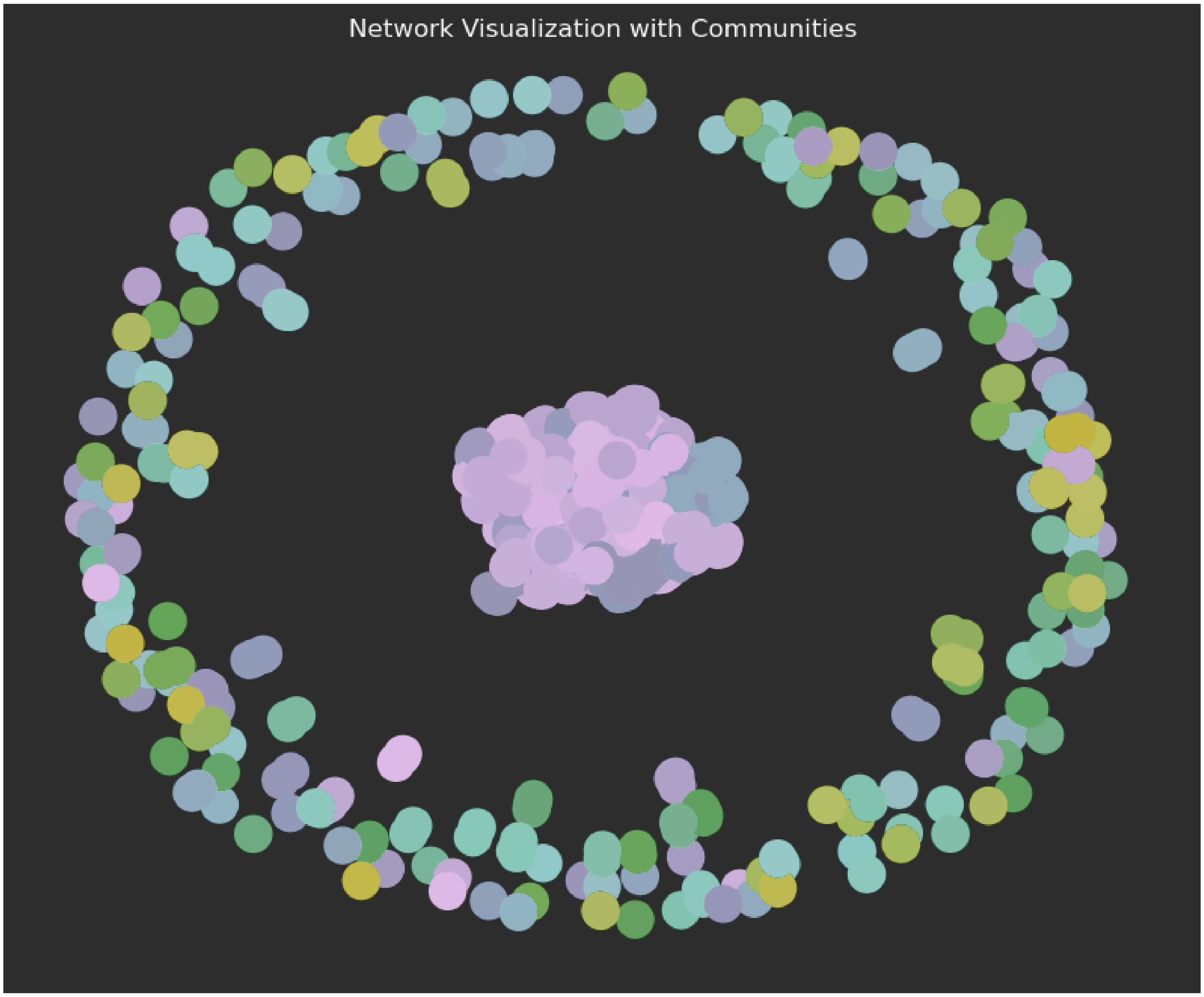


Figure 3.1

References

https://www.kaggle.com/datasets/zeesolver/consumer-behavior-and-shopping-habits-dataset

https://northernprotector.medium.com/modularity-score-6019955f0580

https://www.mygreatlearning.com/blog/types-of-data/

https://medium.com/analytics-vidhya/implement-louvain-community-detection-algorithm-using-python-and-gephi-with-visualization-871250fb2f25

https://youtu.be/0zuiLBOIcsw?si=kM19_VR16jBTWAhr