RECOMMENDER SYSTEM

Online Retail

Table of Contents

Abstract

Internet has redefined the manner in which daily tasks such as online shopping, paying bills, watching movies, communicating, etc., are accomplished. In the older shopping methods were the products were mass produced for a single market and audience is no longer viable. Markets based on long product and development cycles can no longer survive. To stay competitive, markets need to provide different products and services to a diverse set of customers with different needs. The shift to online shopping has made it incumbent on producers and retailers to customize for customer needs while providing more options than were possible before. This, however, poses a problem for customers who must now analyze every offering in order to determine what they actually need and will benefit from. To aid customers in this scenario, we discuss some of the common recommender systems techniques that have been employed and their associated trade-offs.

Problem Statement and Solution

Recommendation algorithms provide an effective form of targeted marketing by creating a personalized shopping experience for each customer.

A recommendation engine filters the data using different algorithms and recommends the most relevant items to users. It first captures the past behaviour of a customer and based on that, recommends products which the users are likely to buy.

Recommender systems can be personalized, non-personalized, attribute-based, item-to-item correlation, and people-to-people correlation.

The model in the problem is built for **‘Germany’** region.

Here we outline the approaches that have been used.

Popularity - The popularity based recommendation system works with the trend, it does not offer any personalization. It uses the items which are in trend and recommends such products to the customers. It is typically used when there is a cold start problem when we do not have information of the product or the customer. In our model it is used as a fall back when the information is sparse.

Content Based – Unlike collaborative methods that only rely on the user-item interactions, content based approaches use additional information about users and/or items. The dataset used here has limited information to extend personalization beyond certain item and user characteristics. We have considered the product description to build the content based model, other characteristics like the product category could be included for the items. With little more details on the user like the age and sex, the model can be expanded to provide more personalised content.
Content based methods suffer far less from the cold start problem than collaborative approaches: new users or items can be described by their characteristics (content) and so relevant suggestions can be done for these new entities. Only new users or items with previously unseen features will logically suffer from this drawback, but once the system old enough, this has few to no chance to happen.

Collaborative - Collaborative methods for recommender systems are methods that are based solely on the past interactions recorded between users and products in order to produce new recommendations. The strength of this approach is that it analyzes existing active customers with similar preferences and characteristics of the current customer to build the recommendations. In our model we have built the interaction matrix between the CustomerID and StockCode based the quantity of the item purchased.

- *Item-Item* - Here we find items similar to the ones the user already "positively" interacted with (purchased x units of the

product). Two items are considered to be similar if most of the users that have interacted with both of them did it in a similar way. This method is said to be “item-centred” as it represent items based on interactions users had with them and evaluate distances between those items to make recommendations.

- *User – User* - The approach is based on the search of similar users in terms of interactions with items. As, in general, every user have only interacted with a few items, it makes the method pretty sensitive to any recorded interactions (high variance). On the other hand, as the final recommendation is only based on interactions recorded for users similar to our user of interest, we obtain more personalized results (low bias). The our model we consider two similar users who are 75% closest match to our user to whom we need to provide recommendations, and recommend products which are not purchased by our user but are frequently purchased by the similar users. The recommendation collaborates the similarity of the items between the neighbours and prepares the must recommend bucket to include products which are most preferred by both the neighbours and rounds up the top 10 recommendations with the uncommon items from the closest neighbour.

Hybrid - Hybrid recommender systems combine two or more recommendation strategies in different ways to benefit from their complementary advantages. In our model we have combined item to item collaborative filtering along with content based and popularity based.

Association rule mining – Association rules are used to recommend products based on their presence along with other products. When two products are purchased together, the presence of one item in a transaction can be used to determine the second product also being in the same transaction. This is very useful when making recommendations to new users who wish to make purchases.
This can assist retailers to determine product placement and promotion optimization (for instance, combining product incentives).

We use the Apriori algorithm and the measure with support, confidence and lift metrics, the minimum threshold for the metrics are set to min_support=0.03, min_confidence=0.65, min_lift=2

Inside Story

Details on the Dataset

The Recommender System is built on the online retail dataset available on the UCI weblink.

https://archive.ics.uci.edu/ml/datasets/online+retail

Features

Below are the features and the type of data –

1. **InvoiceNo**: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'C', it indicates a cancellation.
2. **StockCode**: Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.
3. **Description**: Product (item) name. Nominal.
4. **Quantity**: The quantities of each product (item) per transaction. Numeric.
5. **InvoiceDate**: Invoice Date and time. Numeric, the day and time when each transaction was generated.
6. **UnitPrice**: Unit price. Numeric, Product price per unit in sterling.
7. **CustomerID**: Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.
8. **Country**: Country name. Nominal, the name of the country where each customer resides.

Exploratory Data Analysis

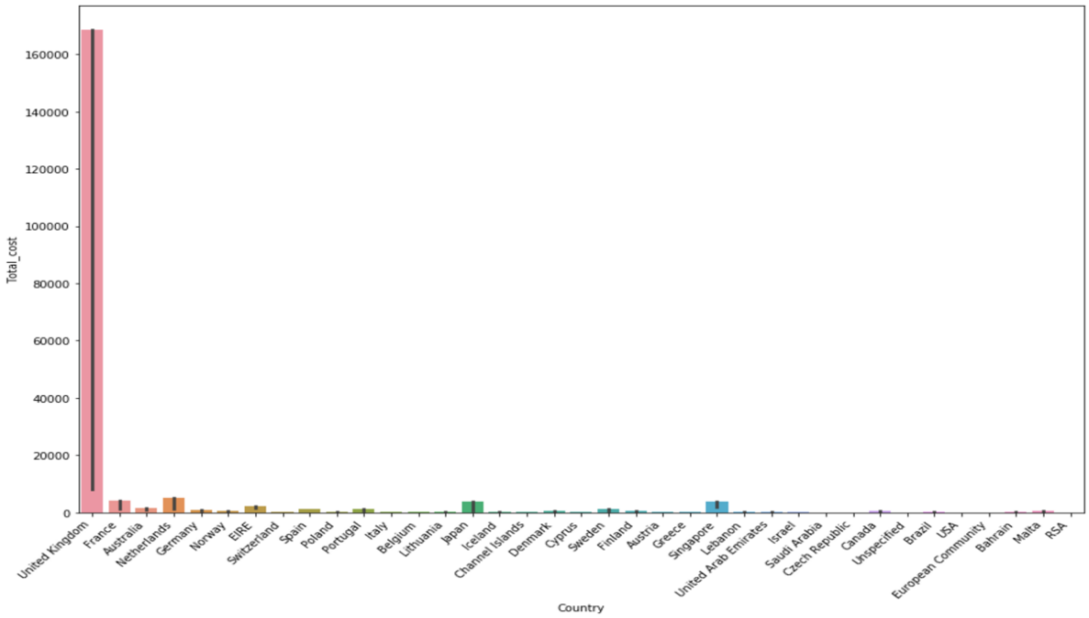
Below are the insights from the dataset

1. Total number of records is 541909
2. Analysis on the data and its types The dataset contains 8 features.
3. There are 38 countries data in the retail dataset and includes one country tagged Unspecified, we need more information on this country.
4. 'Unspecified' country has contributed ONLY for purchases (2667 units of currency) and no return or cancelled transactions.
5. Description and Customer ID have null and empty values. There are 1454 empty records for Description and 135080 for Customer ID
6. Percentage of missing customer IDs is pretty significant (25%) and that could impact the results
7. There are negative quantity records, the negative quantity related to returned or cancelled items, the invoice No for return transactions start with C.
8. Dataset contains Sale and Return/Cancelled orders. Sale transactions are 22064 transactions and return/cancelled transactions are 3836
9. The data contains a year of transactional information from 2010-12-01 to 2011-12-09
10. Drop all null records (Description and CustomerID) and list the popular items
11. Use the StockCode to identify the popular item
12. There are about 15% of return or cancel transactions, we can do some study on the return transactions to identify items that have patterns in these transactions and help prevent future cancellations.
13. Germany region has 9495 records

Visualization

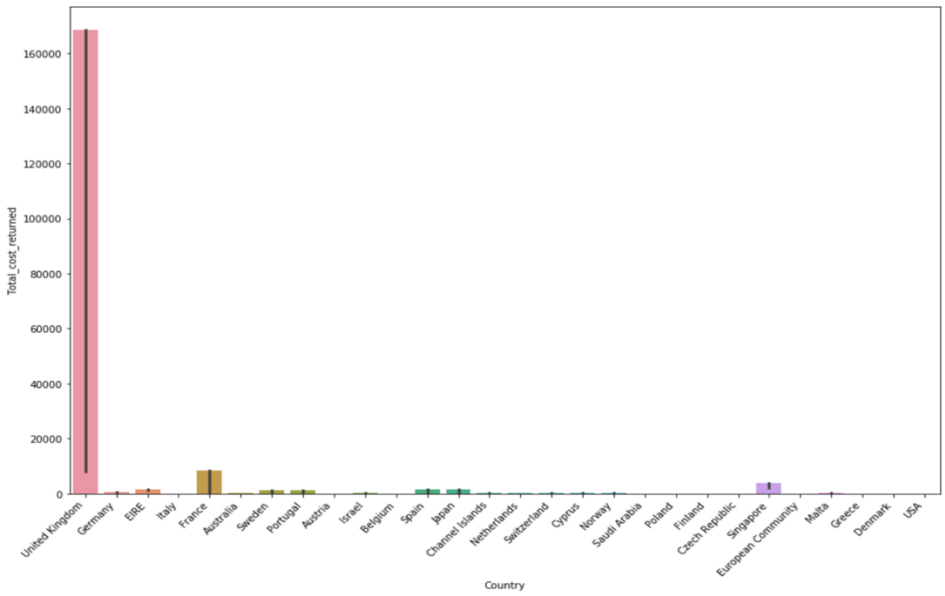
Purchase transactions:

1. United Kingdom and Netherlands have contributed as the top two revenue generators, while Saudi Arabia and Bahrain have contributed the least



Return/Cancel transactions:

1. United Kingdom, EIRE and France have the highest rate of returns or cancelled transaction amounts, while European Community and Saudi Arabia have the least cancelled transaction amounts.



Country	Quantity	Total_cost_returned_abs
United Kingdom	-260939	540518.16
EIRE	-4196	15260.68
France	-1624	12311.21
Singapore	-7	12158.90
Germany	-1815	7168.93

2. An interesting observation is Singapore had only 7 items each of quantity 1 returned and has the 4th highest return amount. These products have StockCode 'M' with description 'Manual', there seems to be something wrong here as the product codes aren't available. Will need to examine these invoices more in detail.

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total_cost_returned_abs
144830	C548830	M	Manual	-1	2011-04-04 13:08:00	162.60	12744.0	Singapore	162.60
144831	C548830	M	Manual	-1	2011-04-04 13:08:00	2382.92	12744.0	Singapore	2382.92
144832	C548830	M	Manual	-1	2011-04-04 13:08:00	239.30	12744.0	Singapore	239.30
144833	C548830	M	Manual	-1	2011-04-04 13:08:00	1252.95	12744.0	Singapore	1252.95
144834	C548834	M	Manual	-1	2011-04-04 13:09:00	2053.07	12744.0	Singapore	2053.07
406404	C571750	M	Manual	-1	2011-10-19 11:16:00	3949.32	12744.0	Singapore	3949.32
406405	C571750	M	Manual	-1	2011-10-19 11:16:00	2118.74	12744.0	Singapore	2118.74

Summary of Initial Findings

Revenue contributors

Increase in revenue (Purchasing power)

Country	Quantity	Total_cost
United Kingdom	4269472	7.308392e+06
Netherlands	200937	2.854463e+05
EIRE	140525	2.655459e+05
Germany	119263	2.288671e+05
France	111472	2.090240e+05

United Kingdom and Netherlands have contributed as the top two revenue generators

Country	Quantity	Total_cost
Brazil	356	1143.60
RSA	352	1002.31
Czech Republic	671	826.74
Bahrain	260	548.40
Saudi Arabia	80	145.92

Saudi Arabia and Bahrain have contributed the least

Decrease in revenue (Cancelled transactions)

Country	Quantity	Total_cost_returned_abs
United Kingdom	-260939	540518.16
EIRE	-4196	15260.68
France	-1624	12311.21
Singapore	-7	12158.90
Germany	-1815	7168.93

United Kingdom, EIRE and France have the highest rate of returns or cancelled transaction amounts

Country	Quantity	Total_cost_returned_abs
Czech Republic	-79	119.02
Greece	-1	50.00
Austria	-54	44.36
Saudi Arabia	-5	14.75
European Community	-2	8.50

European Community and Saudi Arabia have the least cancelled transaction amounts.

Purchase trends

The data is skewed to the European market and it can be seen that France and United Kingdom have a good purchase trend of items purchased during the 1st 3 quarters of 2011 and the last quarter of 2011. They have 7 and 5 items common in the two segments of purchase history.

There are lot of countries (Japan, Portugal, Greece, Israel, USA, Brazil, RSA, Austria, Hong Kong, Canada, Saudi Arabia) where there are no common items purchased during two segments of purchase history.

Data Cleaning and Data Pre-processing

1. Fill missing Description with default description title
2. Fill missing CustomerID with default customer ID (say 99999)
3. Convert CustomerID column to integer
4. Remove 'POST' StockCode from the dataset, it represents a POSTAGE and doesn't relate to a product
5. Filter cancelled transactions (InvoiceNo start with 'C')
6. Replace the outliers for Quantity, replace the 99 percentile value with the mean of quantity
7. Convert StockCode to string and remove spaces
8. Drop labels 'InvoiceNo', 'Country', 'InvoiceDate', 'Description' and 'UnitPrice'
9. Adjust the columns, move the CustomerID to the start of the dataset
10. Create a placeholder with the Customer, StockCode and Quantity information.
11. Create a mapper of StockCode and Description

Recommendation Techniques (with results)

Popularity based filtering

As the name suggests Popularity based recommendation system works with the trend. It basically uses the items which are in trend. For example, if any product which is usually bought by every new user then there are chances that it may suggest that item to the user who just signed up.

Shows positive results under the conditions of data sparsity and cold-starting. Based on the popularity characteristics of products.

The problems with popularity based recommendation system is that the personalization is not available with this method i.e. even though you know the behaviour of the user you cannot recommend items accordingly.

```
def popularGermanItems():
    purchase_g = german_sale_trans_df_orig.groupby(['StockCode'])['Quantity'].sum().reset_index()
    purchase_g_sorted = purchase_g.sort_values("Quantity", ascending=False).reset_index().head(10)
    top_selling_stockcodes = purchase_g_sorted['StockCode'].head(10).tolist()
    return top_selling_stockcodes
```

```
printProducts(popularGermanItems())
```

```
22326 - ROUND SNACK BOXES SET OF4 WOODLAND
15036 - ASSORTED COLOURS SILK FAN
20719 - WOODLAND CHARLOTTE BAG
21212 - PACK OF 72 RETROSPOT CAKE CASES
22585 - PACK OF 6 BIRDY GIFT TAGS
22629 - SPACEBOY LUNCH BOX
22554 - PLASTERS IN TIN WOODLAND ANIMALS
22961 - JAM MAKING SET PRINTED
22423 - REGENCY CAKESTAND 3 TIER
16045 - POPART WOODEN PENCILS ASST
```

Content-based filtering

Unlike collaborative methods that only rely on the user-item interactions, content based approaches use additional information about users and/or items. The dataset used here has limited information to extend personalization beyond certain item and user characteristics. We have considered the product description to build the content based model, other characteristics like the product

category could be included for the items. With little more details on the user like the age and sex, the model can be expanded to provide more personalised content.
Content based methods suffer far less from the cold start problem than collaborative approaches: new users or items can be described by their characteristics (content) and so relevant suggestions can be done for these new entities. Only new users or items with previously unseen features will logically suffer from this drawback, but once the system old enough, this has few to no chance to happen.

The contents can be described using labels and the labels are given a weight of how well they describe the article. Using these labels and user preferences, nearest neighbour or clustering algorithms can be used to recommend other articles to the active user. However, new users with limited information and limited number of labels pose a challenge to this method. Common algorithms that are applicable include k-nearest neighbour, clustering, Bayesian, and artificial neural networks.

A set of attributes describing a user or am item is computed and they are then used to make recommendations to the user. The attributes are compared with keywords describing the recommendations as mentioned. Keywords used to make recommendations are weighted using term frequency/inverse document frequency (TF-IDF) method to measure importance. Term frequency TF is calculated from N items that could potentially be recommend to user as.

TF is represented as ($f_{t,d} = \sum_{t \in d} ft, d$): Basically says how frequent the term 't' occurs in document'd'

Keywords that appear in many different documents are not useful when distinguishing between relevant and irrelevant documents. To do that, inverse document frequency is used. Inverse document frequency IDF is calculated as

IDF is defined as:

$$IDF(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Where N is the total number of documents in the collection also known as cardinality of document space;

$|\{d \in D : t \in d\}|$ is the number of documents where term t is present

In order for recommender systems to be able to generate recommendations, content must be structured and easy to parse. If this is not, then the item must be described manually. Another problem is being able to differentiate between a bad item and a good item based on retrieved information. A bad item using same keywords as good item will also get recommended. Two other major drawbacks are lack of information about a user, and overspecialization. When a new user is introduced into the system, their preferences and profiles are not aggregated. The user would not have given enough ratings, and reviews to products. This leads to insufficient information to generate recommendations. When the system is only able to recommend certain items based on user's profile, it leads to overspecialization.

Content based filterring uses the 'Description' of the product using the term frequency vectorizer to make recommendations

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(german_stock_desc_df['Description'])
doc_term_matrix = tfidf_matrix.todense()
df = pd.DataFrame(doc_term_matrix,
                  columns=tfidf.get_feature_names(),
                  index=german_stock_desc_df['Description'])

from sklearn.metrics.pairwise import linear_kernel
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
from sklearn.metrics.pairwise import cosine_similarity
indices = pd.Series(german_stock_desc_df.index, index=german_stock_desc_df['Description'])
```

```
# Function that takes in product description as input and outputs most similar products
def get_recommendations(stockCode, cosine_sim=cosine_sim):
    #Get the product desc from the stockcode
    desc = german_stock_desc_df[german_stock_desc_df['StockCode'] == stockCode]['Description'].tolist()[0]

    # Get the index of the movie that matches the description
    idx = indices[desc]

    # Get the pairwsie similarity scores of all products with that product
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the products based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar products
    sim_scores = sim_scores[1:11]

    # Get the product indices
    product_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar products
    return german_stock_desc_df[german_stock_desc_df['StockCode'].isin(german_stock_desc_df['StockCode'].iloc[product_i
```

```
try:
    print(get_recommendations(stockCode).to_string(index=False))
except:
    print('StockCode Not Recognised !!', '\n')
    print('Take a look at popular products.....', '\n')
    printProducts(popularItems('Germany'))
```

```
StockCode                          Description
    20676                RED RETROSPOT BOWL
    20685             DOORMAT RED RETROSPOT
    20749         ASSORTED COLOUR MINI CASES
    21042               RED RETROSPOT APRON
    21212   PACK OF 72 RETROSPOT CAKE CASES
    21238                  RED RETROSPOT CUP
    21242               RED RETROSPOT PLATE
    21498                RED RETROSPOT WRAP
    21844                  RED RETROSPOT MUG
    23295  SET OF 12 MINI LOAF BAKING CASES
```

Collaborative filtering

Collaborative methods for recommender systems are methods that are based solely on the past interactions recorded between users and items in order to produce new recommendations. These interactions are stored in the so-called “user-item interactions matrix”.

Users	User-item interactions matrix	Items
Shoppers/Buyers	Quantity of product purchased	Product

The strength of this approach is that it analyzes existing active customers with similar preferences and characteristics of the current customer to build the recommendations.

To implement an item based collaborative filtering, KNN is a perfect go-to model and also a very good baseline for recommender system development. But what is the KNN? KNN is a non-parametric, lazy learning method. It uses a database in which the data points are separated into several clusters to make inference for new samples.

KNN does not make any assumptions on the underlying data distribution but it relies on item feature similarity. When KNN makes inference about an item, KNN will calculate the “distance” between the target item and every other item in its database, then it ranks its distances and returns the top K nearest neighbour items as the most similar item recommendations.

Wait, but how do we feed the dataframe of ratings into a KNN model? First, we need to transform the dataframe of ratings into a proper format that can be consumed by a KNN model. We want the data to be in an m x n array, where m is the number of items and n is the number of users. To reshape dataframe of ratings, we’ll pivot the dataframe to the wide format with items as rows and users as columns. Then we’ll fill the missing observations with 0s since we’re going to be performing linear algebra operations (calculating

distances between vectors). Let’s call this new dataframe a “dataframe of item features”.

Our dataframe of item features is a sparse matrix with a shape of (1664, 94). We definitely don't want to feed the entire data with mostl y 0s in float32 datatype to KNN. For more efficient calculation and le ss memory footprint, we need to transform the values of the dataframe into a scipy sparse matrix.

KNN’s performance will suffer from curse of dimensionality if it uses “euclidean distance” in its objective function. Euclidean distance is unhelpful in high dimensions because all vectors are almost equidistant to the search query vector (target item features). Instead, we will use cosine similarity for nearest neighbour search. There is also another popular approach to handle nearest neighbour search in high dimensional data

The final part of the recommendation system is to make the actual recommendations which is to calculate top m recommendations from the computed neighbourhood of customers. Two prominent techniques that are used are most- frequent item recommendation, and association rule-based recommendations. In Most-Frequent Item Recommendation, neighbourhood N is scanned frequency count of purchases is calculated for each neighbour. All the products are then sorted according to the frequency and m most frequently bought products that is not purchased by the current customer are recommended.

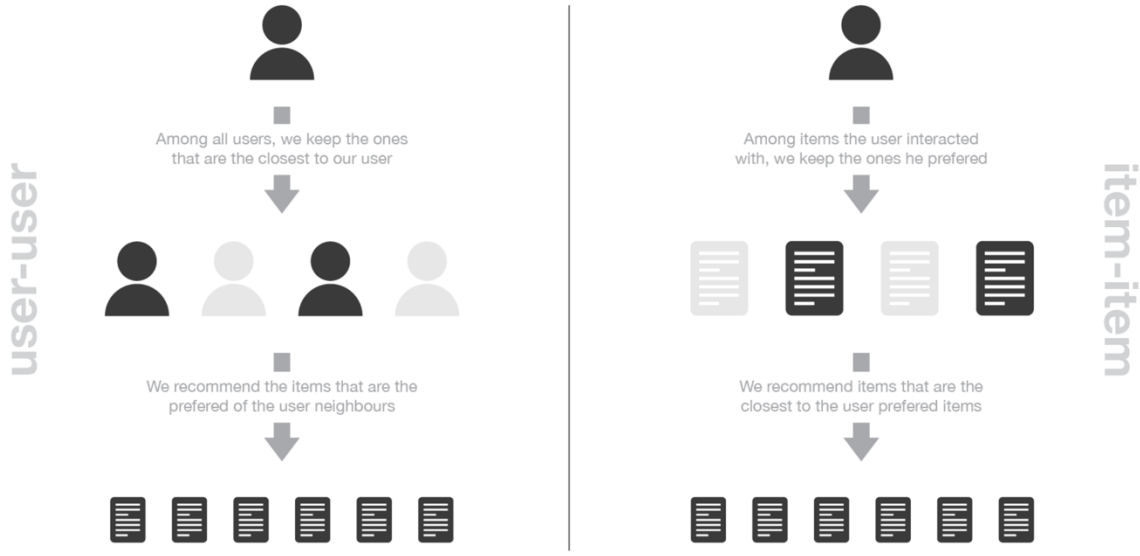


Illustration of the difference between item-item and user-user methods.

Item based

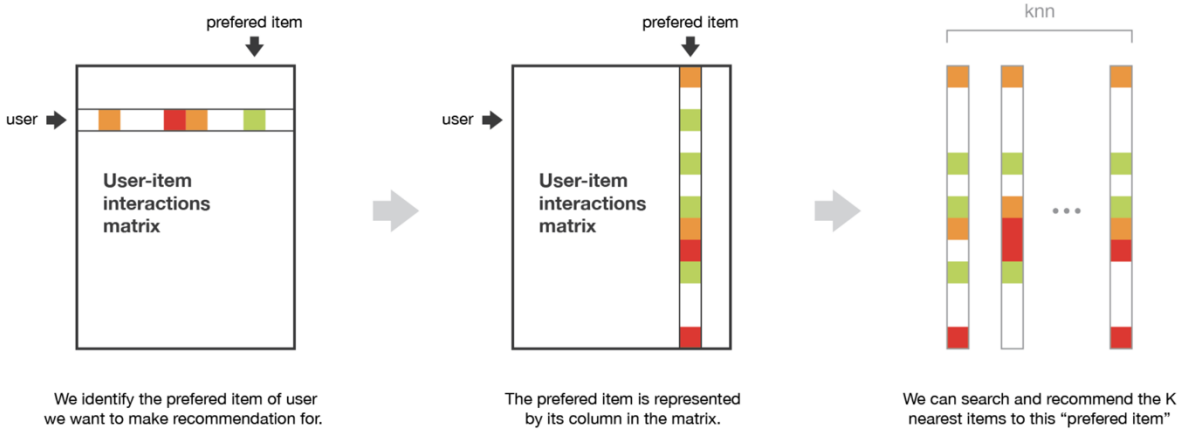
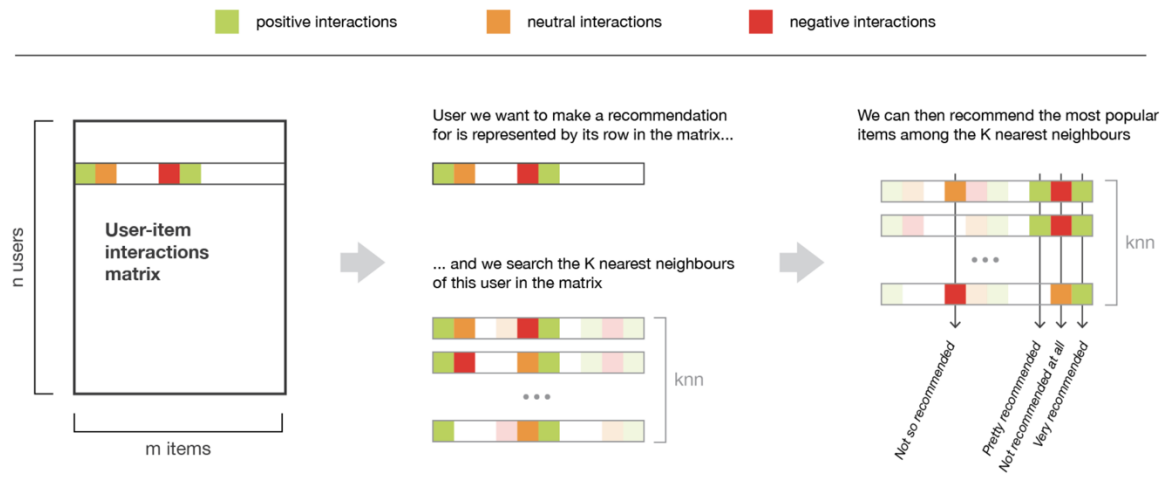


Illustration of the item-item method.

Here we find items similar to the ones the user already "positively" interacted with (purchased x units of the product). Two items are considered to be similar if most of the users that have interacted with both of them did it in a similar way. This method is said to be "item-centred" as it represent items based on interactions users had with them and evaluate distances between those items to make recommendations.

User based



The model will recommend items from the 1st nearest neighbour plus evaluates items from the 2nd nearest neighbour if the cosine distance is > 0.75.
The model will then filter the top 10 items from the two neighbours and picks the common items plus the uncommon items from the 1st nearest neighbour as the recommendation list.

```
def userBasedCollabFilter(customerID):
    customerIDIndex = getCustomerIDIndex(customerID)
    if(customerIDIndex == -1):
        print('Customer ID not Recognised!!!','\n')
        print('Take a look at popular products.....','\n')
        printProducts(popularGermanItems())
    else:
        recommends = make_recommendation_collab(
        model_knn=model_knn,
        data=cust_user_features,
        fav_product=customerID,
        index_value=customerIDIndex,
        n_recommendations=10)

        similarCustomersDict = similarCustomers(recommends)
        print(similarCustomersDict)
        print('\n')
        dictEmpty = bool(similarCustomersDict)
        if(dictEmpty):
            most_similar_customer= list(similarCustomersDict.keys())[0]
            #Get top 10 items for this customer and we will recommend these to our target customer

            #Get the list of items (top 10) not purchased by target customer but most frequently purchased by similar c
            target_customer_item_list = german_sale_trans_group_df[german_sale_trans_group_df['CustomerID'] == customer
            most_similar_cust_item_list = german_sale_trans_group_df[german_sale_trans_group_df['CustomerID'] == most_s

            items_not_purchased = [item for item in most_similar_cust_item_list if item not in target_customer_item_lis

            cust_stock_item_df = german_sale_trans_group_df[(german_sale_trans_group_df['CustomerID'] == most_similar_c
            qty_mean = cust_stock_item_df['Quantity'].mean()

            #Get top 10 stockCodes sold more than the mean qty
            first_cust_top10_stockCodes = cust_stock_item_df[cust_stock_item_df['Quantity'] >= qty_mean]['StockCode'].h
            print('1st Customer - Top 10 StockCodes')
            print(first_cust_top10_stockCodes.values)
            print('\n')
            #Check the next closest customer to the target customer and evaulated the item basket if the distance > .75
            try:
                if(list(similarCustomersDict)[1]):
                    second_closest_customer = list(similarCustomersDict)[1]
                    distance = similarCustomersDict[second_closest_customer]
                    if(distance > 0.75):
                        print('2nd Customer - Top 10 StockCodes')
```

```
                        second_most_similar_cust_item_list = german_sale_trans_group_df[german_sale_trans_group_df['Cus
                        second_cust_items_not_purchased = [item for item in second_most_similar_cust_item_list if item
                        second_cust_stock_item_df = german_sale_trans_group_df[(german_sale_trans_group_df['CustomerID'
                        second_cust_qty_mean = second_cust_stock_item_df['Quantity'].mean()

                        second_cust_top10_stockCodes = second_cust_stock_item_df[second_cust_stock_item_df['Quantity']
                        print(second_cust_top10_stockCodes.values)
                        print('\n')
                        prod_to_recommend = filter_top10(first_cust_top10_stockCodes, second_cust_top10_stockCodes)
                        if(len(prod_to_recommend) >= 1):
                            print('Product Recomendations for Customer - ',customerID)
                            print('\n')
                            printProducts(prod_to_recommend)
                        else:
                            print('No products to recommend','\n')
                            print('Take a look at popular products.....', '\n')
                            printProducts(popularGermanItems())
            except IndexError:
                print('Invalid')

        else:
            print('No products to recommend','\n')
            print('Take a look at popular products.....', '\n')
            printProducts(popularGermanItems())
```

```
In [103]: customerID = 12468 #Has an item from Customer 2
          userBasedCollabFilter(customerID)
```

```
{13817: 0.8017804874263846, 12518: 0.7915542143349228, 12481: 0.7899492861238996, 12472: 0.7477686722481475, 12712:
0.7427780263723879, 12613: 0.698914685504644, 12474: 0.6798952512896485, 12705: 0.6572407385172263, 13816: 0.64613635
21971923, 12645: 0.5578316115615872}


1st Customer - Top 10 StockCodes
['20712' '22333' '22555' '23433' '22551' '23475' '23474' '20981']


2nd Customer - Top 10 StockCodes
['22705' '22708' '22534' '22037' '22551' '22029' '23290' '23084' '16169E'
 '22544']


Common stockCode:
 ['22551']
Product Recomendations for Customer -  12468


22551 - PLASTERS IN TIN SPACEBOY
20712 - JUMBO BAG WOODLAND ANIMALS
22333 - RETROSPOT PARTY BAG + STICKER SET
22555 - PLASTERS IN TIN STRONGMAN
23433 - HANGING QUILTED PATCHWORK APPLES
23475 - WOODLAND SMALL PINK FELT HEART
23474 - WOODLAND SMALL BLUE FELT HEART
20981 - 12 PENCILS TALL TUBE WOODLAND
```

```
Invalid or Empty Customer ID will recommend Popular StockCodes
```

```
customerID = '' #Has an item from Customer 2
userBasedCollabFilter(customerID)
```

```
Customer ID not Recognised!!!

Take a look at popular products.....

22326 - ROUND SNACK BOXES SET OF4 WOODLAND
15036 - ASSORTED COLOURS SILK FAN
20719 - WOODLAND CHARLOTTE BAG
21212 - PACK OF 72 RETROSPOT CAKE CASES
22585 - PACK OF 6 BIRDY GIFT TAGS
22629 - SPACEBOY LUNCH BOX
22554 - PLASTERS IN TIN WOODLAND ANIMALS
22961 - JAM MAKING SET PRINTED
22423 - REGENCY CAKESTAND 3 TIER
16045 - POPART WOODEN PENCILS ASST
```

Hybrid filtering

To avoid problems that exist in both content-based and collaborative filtering systems, hybrid solutions have been proposed. Solutions include: implement both filtering separately and combine the results, incorporating characteristics of content-based filtering to collaborative adding characteristics of collaborative filtering to content- based filtering systems and new algorithms that incorporate both systems' techniques. Combining different recommender systems approach involves building two different recommender systems based on collaborative-based and content-based approaches. Many recommender systems are implemented using collaborative- based approach with content-

based user profiles generated through content-based approach. The profiles are then used to find similarity between users rather than items which help the system overcome some of the sparsity-related limitations. Recommendations can be generated through collaborative filtering first. They are then compared against current user profile to determine if it's interesting to the user or not and to present it. Curse of dimensionality occurs when a lot of features exist per item that makes it difficult to cluster or compare them. The most common approach is to use dimensionality reduction algorithm on a group of content based profiles. This allows performance improvements since it reduces the amount of preferences/features that must be compared to generate the recommendations.

In the model we built, as we do not have user features, we have relied on the item features to build the hybrid model. We have used the 'Description' and 'Quantity' features of the item in hybrid model.

Hybrid Model = Item based Collaborative filtering + Content based filtering + Popularity based filtering

The model will recommend the top 10 similar items applying item based collaborative filtering and content based (based on the item description).

The model will 1st select top 10 similar items based on the cosine similarity of the items (collaborative filtering) and follows it up with the top 10 similar items based on the content based similarity. The model then compares the two sets of top 10 recommendations and prepares the final recommendation list which contains common items from the collaborative and content based approaches and uncommon items from collaborative filtering.

```
def itemBasedCollabFilter(productCode):
    stockIndex = getStockCodeIndex(productCode)
    if(stockIndex == -1):
        print('StockCode not Recognised!!!','\n')
        print('Take a look at popular products.....','\n')
        printProducts(popularGermanItems())
    else:
        recommends = make_recommendation_collab(
        model_knn=model_knn,
        data=cust_item_features,
        fav_product=productCode,
        index_value=stockIndex,
        n_recommendations=10)

        top10product_collab_list = productsToRecommend(recommends)
        top10product_content = get_recommendations_content(productCode)
        top10product_content_list = top10product_content['StockCode'].tolist()
        print('Top 10 StockCodes - Item Collab \n',top10product_collab_list)
        print('Top 10 StockCodes - Content based \n',top10product_content_list)
        prod_to_recommend = filter_top10(top10product_collab_list, top10product_content_list)
        if(len(prod_to_recommend) >= 1):
            print('\n Product Recomendations for ',productCode, '-', german_sale_trans_df_orig[german_sale_trans_df_ori
            print('\n')
            printProducts(prod_to_recommend)
        else:
            print('No products to recommend','\n')
            print('Take a look at popular products.....', '\n')
            printProducts(popularGermanItems())
```

```
itemProductCode = '10125'
itemBasedCollabFilter(itemProductCode)
```

```
Top 10 StockCodes - Item Collab
 ['22433', '21884', '21883', '16011', '22708', '22610', '22502', '22608', '22609', '21882']
Top 10 StockCodes - Content based
 ['16168M', '20750', '20751', '22544', '22547', '22608', '22741', '23388', '23389', '47310M']
Common stockCode:
 ['22608']

 Product Recomendations for  10125 - MINI FUNKY DESIGN TAPES


22608 - PENS ASSORTED FUNKY JEWELED
22433 - WATERING CAN GREEN DINOSAUR
21884 - CAKES AND BOWS GIFT  TAPE
21883 - STARS GIFT TAPE
16011 - ANIMAL STICKERS
22708 - WRAP DOLLY GIRL
22610 - PENS ASSORTED FUNNY FACE
22502 - PICNIC BASKET WICKER SMALL
22609 - PENS ASSORTED SPACEBALL
21882 - SKULLS TAPE
```

Invalid or empty product code will make the model recommend popular products

```
itemProductCode=''
itemBasedCollabFilter(itemProductCode)
```

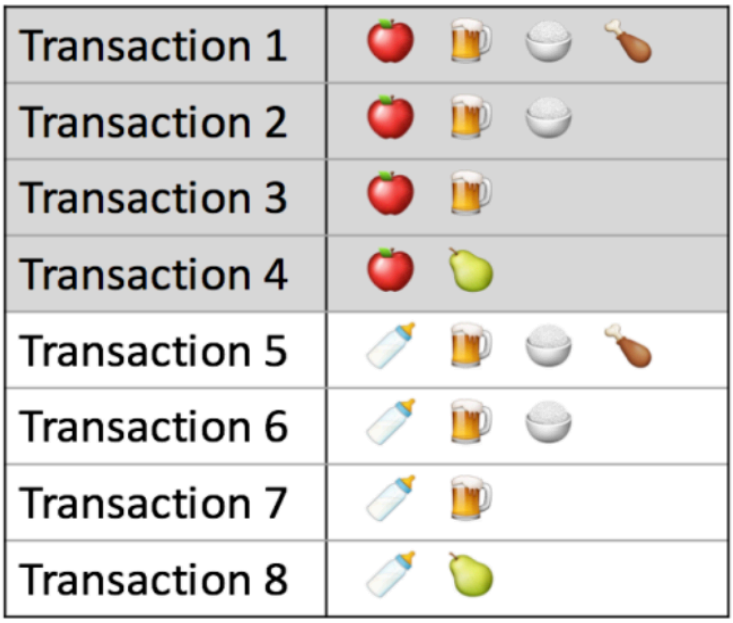
```
StockCode not Recognised!!!

Take a look at popular products.....

22326 - ROUND SNACK BOXES SET OF4 WOODLAND
15036 - ASSORTED COLOURS SILK FAN
20719 - WOODLAND CHARLOTTE BAG
21212 - PACK OF 72 RETROSPOT CAKE CASES
22585 - PACK OF 6 BIRDY GIFT TAGS
22629 - SPACEBOY LUNCH BOX
22554 - PLASTERS IN TIN WOODLAND ANIMALS
22961 - JAM MAKING SET PRINTED
22423 - REGENCY CAKESTAND 3 TIER
16045 - POPART WOODEN PENCILS ASST
```

Association rule mining (Market Basket Analysis)

Association rules are used to recommend products based on their presence along with other products. When two products are purchased together, the presence of one item in a transaction can be used to determine the second product also being in the same transaction. This is very useful when making recommendations to new users who wish to make purchases. To define association more formally, a collection of products m products {P1, P2, P3,,...Pm} belongs to set P. We say a transaction T from set of transactions D is a subset of P, such that the transaction contains products from P. Each transaction can be uniquely identified as TID. A transaction T contains set X, a subset of products from P and it is a subset of T. Association rules implies that there exists Y, subset of P and there is no mutual product between X. This means that whenever products from X exist in a transaction T, there is high likelihood that products from Y will also exist in the same transaction. Two variables, confidence c, and support s are used to measure the quality of the associations made. Support measures how frequent the association happens in the entire set of

transactions as shown in and confidence measures the frequency of both products occurring whenever one product exists in the transaction.

Consider an example with 8 transactions, each transaction having the item combinations listed below.



Measure 1: Support. This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears.

Support {Apple} = 4/8 = 50%
Support {Apple + Beer + Rice} = 2/8 = 25%

Measure 2: Confidence. This says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears.

Confidence of {apple -> beer} = Support {apple, beer} / Support {apple}

= (3/8) / (4/8)
= 3/4 = 75%

Measure 3: Lift. This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is.

Lift {apple -> beer} = Support {apple, beer) / (Support{apple} * Support{beer})
= 1

A lift value greater than 1 means that item Y is *likely* to be bought if item X is bought, while a value less than 1 means that item Y is *unlikely* to be bought if item X is bought.

A major drawback is the support for a lot rules. Association rules are slower and not very effective when a lot of mining rules are used to make recommendations.

Below is the result of running the Apriori algorithm on the Germany dataset with min_support=0.03, min_confidence=0.65, min_lift=2.

```
asso= association_rules(freq_items, metric='confidence', min_threshold=.65).sort_values(by='lift', ascending=False)
asso.sort_values(by="lift", ascending=False)
```

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
(SPACEBOY CHILDRENS CUP)	(SPACEBOY CHILDRENS BOWL)	0.043764	0.041575	0.037199	0.850000	20.444737	0.035380	6.389497
(SPACEBOY CHILDRENS BOWL)	(SPACEBOY CHILDRENS CUP)	0.041575	0.043764	0.037199	0.894737	20.444737	0.035380	9.084245
(CHILDRENS CUTLERY SPACEBOY)	(CHILDRENS CUTLERY DOLLY GIRL)	0.048140	0.050328	0.039387	0.818182	16.256917	0.036965	5.223195
(CHILDRENS CUTLERY DOLLY GIRL)	(CHILDRENS CUTLERY SPACEBOY)	0.050328	0.048140	0.039387	0.782609	16.256917	0.036965	4.378556
(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS)	0.056893	0.052516	0.045952	0.807692	15.379808	0.042964	4.926915
(SET/6 RED SPOTTY PAPER CUPS)	(SET/6 RED SPOTTY PAPER PLATES)	0.052516	0.056893	0.045952	0.875000	15.379808	0.042964	7.544858
(JAM JAR WITH GREEN LID)	(JAM JAR WITH PINK LID)	0.035011	0.063457	0.032823	0.937500	14.773707	0.030601	14.984683

There is a 20 times more likihood to buy SPACEBOY CHILDRENS BOWL than an avg customer if they but a SPACEBOY CHILDRENS CUP. This rule is "true" in 90% of the cases (confidence). This can be used as insight to recommend SPACEBOY CHILDRENS BOWL for those customers who bought SPACEBOY CHILDRENS CUP.

Also, a customer is 15 times more likely to buy 'SET/6 RED SPOTTY PAPER PLATES' than an average customer (lift) if he/she buys 'SET/6 RED SPOTTY PAPER CUPS'. This rule is "true" in 87% of the cases (confidence). This can be used as insight to recommend SET/6 RED SPOTTY PAPER PLATES for those customers who bought SET/6 RED SPOTTY PAPER CUPS.

Limitations –

Listed below are the limitations with the model

1. Customer behavioural patterns have not been accounted for as the data is a static dataset based on the customer's historical data and the customer engagement metrics are not available. Example - duration of browsing, clicks on a particular product and number of clicks, the links of webpage; scroll, close, refresh of page; selection, search, copy, paste, bookmark.
2. Product level discounts or promotions are not available. Recommendations are not factored on discounts which could enhance the personalization experience.
3. Quantity of the product alone is the feature that is used in building the collaborative filtering. The results might not be personalised as quantity feature is not quite impactful. Ideally we will need to factor on the category of the product too to make the recommendations personalized.
4. Personalization on a product brand is not available as the dataset does not have the brand level details.
5. The models are evaluated based on the intuitional sense the recommendation system is making, evaluation of the model can be made after productionalizing against live data.
6. Data is not clean, distinct stockcodes do not have a unique product description. Data processing to achieve this unique mapping has resulted in retaining only the 1st occurrence of the description data for a stockcode and dropping the rest.

   Pls note the number of records in the dataset is not impacted

Conclusion and Future work –

Conclusion

Recommender systems allow e-commerce sites to be highly customizable for the user and buyer. They allow companies to better understand their users, provide personalized stores, and in turn increase customer satisfaction and loyalty. They are implemented by utilizing various existing data mining tools and

adapting them to current needs. Popular approaches include using association rules, collaborative filtering and content-based filtering and hybrid filtering. Recommendations using association rules are generated based on previous transactions the user has already displayed interest in. Collaborative filtering allows the active user to get recommendation based on products that users with similar interest have purchased and rated positively (in this use case, purchased more of that item), and by using the active user's previous purchase pattern and transaction history to build a model that provides a new set of similar products. Content- based filtering compares the user's personal profile and preferences with the database to find products that are of interest and align with the active user and present them. Recommendations can range from being personalized to community driven and allow for a wide range of possibilities. The recommendations are also being refreshed due to the nature of changing search history, ratings, and arrival of new products. This also poses many challenges which include cold start, handling anonymous users, creating a social recommender system that can accommodate more than one active user, handling various different data sources and scalability with increased data.

Future Work

1. Content Based Recommendation can be enhanced from recommending products based on the product description to recommend products based on the category of the product. This approach will provide recommendations of related items and will help drive better customer engagement and satisfaction.

2. Investigate on NLP (Natural Language Processing) models which assign weights to words and may be alphabets and recommend closer matches.

3. Collaborative Filtering algorithm can be improvised to provide recommendations on the products at the category level.

4. Recommend items of the user depending on the frequency of purchases based on the purchase pattern and historical data that is available. Example – If there is a pattern in the customer purchase patterns at the start of every month or start of the week recommend same + similar items to replenish the stock.

5. Evaluate other machine learning models like the unsupervised learning to determine the customer segmentation and Random Forest classifier to validate the likelihood of customer purchasing the recommended product.

6. Productionalize the model to read data in real time from a Kafka Topic and store the recommendations along with the metrics in the database.

7. Interface the model with a user interface to have a visual (may be images of the product recommendations) of the recommendations.

8. User based recommendation to include popular items too besides the qty of the items purchased by the other users.

9. Model return/cancelled transactions to derive insights into the return pattern of the items and the customers.

10. Devise strategy to convert the quantity feature to a rating_scale, the current available algorithms matrix factorization and Surprise package work well with rating scale up to 10.