Recommender Systems – An Introduction

Every app that you use on the internet collects data about your activity, about how you interact with things, what you search for, who do you interact with, etc.

These apps know you better than you know yourself. One of the most common applications of this collected data is Recommender Systems. A recommender system is an application of machine learning that predicts the future preference of a set of products for a user and provides personalized suggestions.

For example, Facebook uses your social media activity to improve your newsfeed and to decide which ads to show, YouTube uses their recommendation systems to decide which video to play next and to display relevant content on your home page,

Amazon uses it to suggest products you should buy. In the age of cut-throat competition between streaming services- Netflix comes out as the clear winner with its 25 billion USD annual revenue, majorly due to the quality of its recommender system.

In 2009 Netflix offered a million dollars to anyone who could improve the quality of recommendations by just 10%. We can't stress enough the importance of recommender systems and their applicability in our day-to-day lives.

They also act as information filtering systems as they filter out irrelevant content and provide users with a personalized experience of whatever service they are using. They are used to predict a user's liking or preference for a particular product or service.

Types of Recommender Systems

There are primarily two techniques for building recommendation engines, the others are either extensions or hybrid recommender systems (a combination of these) :

1) Content-Based Filtering

The main idea here is to suggest items based on a particular item. For example, when you are building a movie recommendation system, it would take into account a user's preference for a movie using metrics such as ratings and then use item metadata, such as genre, director, description of the movie, cast, and crew, etc to find movies similar to the ones that a user has liked.

2) Collaborative Filtering

The collaborative filtering recommendation technique depends on finding similar users to a target user to make personalized recommendations. Collaborative filtering recommender systems do not require item metadata like content-based recommendation systems. It relies solely on past user-item interactions to render new recommendations.

Content-Based Recommender Systems

Content-based recommendation systems work more closely with item features or attributes rather than user data. They predict the behaviour of a user based on the items they react to.

A classic problem that millennials have today is finding a good movie to binge-watch over the weekend without having to do too much research.

Collaborative Filtering Recommender Systems

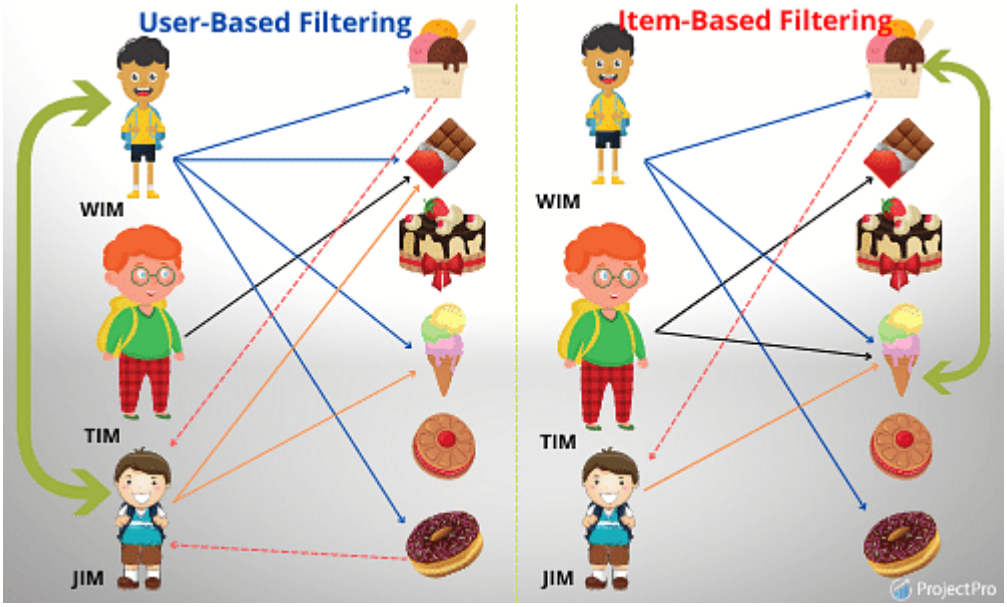
The movies recommended using the above approach are not really personalized in the sense that content-based recommendation engines do not capture the personal tastes and biases of a user.

The collaborative filtering technique depends solely on the historical preferences of a user and the interaction between a user and an item. User preference is studied in two ways: **explicitly-** this is the rating or feedback given to an item by the user explicitly, implicitly wherein a user preference is noted in terms of clicks, views, hovering time, etc.

There are basically two types of collaborative filtering recommendation methods based on whether they assume there is an underlying model governing the data.

1) Memory-Based Collaborative Filtering

Also known as neighborhood-based filtering in which past interactions between a user and item are stored in user-items interaction matrix. The recommendations are made either in a user-based or item-based fashion.



- **User-Based Collaborative Filtering:**

This method aims at finding similar users and recommendations are made to a user based on what a similar user has liked. The similarity between users is calculated using either Cosine Similarity or Pearson Correlation.

Example :-

	Product A	Product B	Product C	Product D	Similarity(i, B)
User A	4.		2.	5.	0.75
User B	5.		4.	4.	1.
User C		1.	4.		1.
User D		3.			NA
User E	4.	5.	4.	4.	1.

Each row represents a user and each column a product. The last column is to record the similarity between a target user and a given user. Let User B be the target user. The similarity between any user and User B can be computed using Pearson Correlation only if they have given ratings to at least one common product. User D does not share any product rating with B therefore we can't compute similarity. Also note that while User B and C have similar rating for Product C, the similarity is not

that high, since it's the only product they both have rated in common. Now, we are able to predict the rating that user B would give based on the similarity with other users. User B is most similar to User E, so the rating that user B would give to Product B would be close to 5.

That was easy, right? But one of the main disadvantages of user-based filtering is user preferences can change over time, therefore these calculations have to be updated regularly.

- ## Item Based Collaborative Filtering

Here, new items are recommended to users based on their similarity with the items that the user has rated- highly in the past. In the below table we'll fill in the blanks vertically.

	Product A	Product B	Product C	Product D	Product E
User A	4.		2.	5.	
User B	5.	3.	4.	4.	
User C			4.		
User D	4.	3.			
User E	4.	5.	4.	4.	
•					

Now, calculating the similarity between items isn't as straightforward as calculating the similarity between users. The similarity between users can be calculated as -

$$similarity(i,j) = \frac{\sum_u^U r_{(u,i)} r_{(u,j)}}{\sqrt{\sum_u^U r_{(u,i)}^2} \sqrt{\sum_u^U r_{(u,j)}^2}}$$

1. If you want to compute the similarity between Product A and B, first multiply all the ratings the users have given to these products and sum the results.

	Product A	Product B	Product C	Product D	A X B
User A	4.		2.	5.	0
User B	5.	3.	4.	4.	15.
User C			4.		1.
User D	4.	3.			12.
User E	4.	5.	4.	4.	20.
•					
					Sum = 47

2. Now, we'll square the ratings of Product A and B separately and take the sum of these squares. Since we are only dealing with Product A and B at the moment, I'll remove C and D from the table for simplicity.

	Product A	Product B	A X B	A squared	B squared
User A	4.		0	16	0
User B	5.	3.	15.	25.	9.
User C			1.	1.	1.
User D	4.	3.	12.	16.	9.
User E	4.	5.	20.	16.	25.
•					
			Sum = 47	X= 73	Y = 43

3. Now, we'll take the square roots of the sum of these squares for A and B and multiply them.

   Squareroot(73)*squareroot(43) =56

	Product A	Product B	A X B	A squared	B squared
User A	4.		0	16	0
User B	5.	3.	15.	25.	9.
User C			1.	1.	1.
User D	4.	3.	12.	16.	9.
User E	4.	5.	20.	16.	25.
•					
			Sum = 47	X= 73	Y = 43
				1.	1.

On multiplying the square roots, we get 56. Now, after dividing 47 with 56 we get 0.83 which is in fact the similarity between product A and B.

47/56=0.83

Similarly, you can compute the similarity between any 2 products and recommend a product similar to the product the user has rated before.

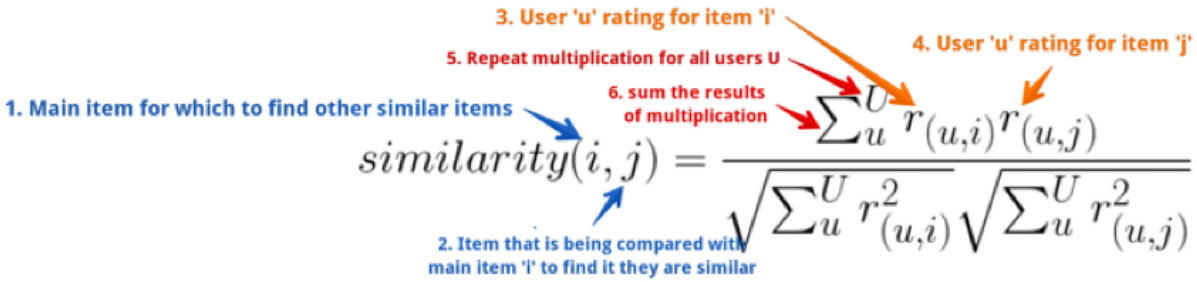


Image Credit: towardsdatascience.com

The formula makes a lot more sense now. Item-based filtering avoids the dynamic preference problem posed by user-based filtering.

1) Memory-Based Collaborative Filtering Python

We will use the Amazon Ratings (Beauty Products) to implement memory-based collaborative filtering in python.

This Kaggle dataset is very similar to the example we have used above. We have UserId, ProductId , Rating, and Timestamp as columns. Using UserId, ProductId and Ratings we'll try to find users like our target user (selected randomly) and recommend products to our target user- these recommended products would be the products we previously found similar users have liked.

```
In [2]: amzn_ratings= pd.read_csv('ratings_Beauty.csv')
```

```
In [3]: amzn_ratings.head()
```

Out[3]:

	UserId	ProductId	Rating	Timestamp
0	A39HTATAQ9V7YF	0205616461	5.0	1369699200
1	A3JM6GV9MNOF9X	0558925278	3.0	1355443200
2	A1Z513UWSAAO0F	0558925278	5.0	1404691200
3	A1WMRR494NWEWV	0733001998	4.0	1382572800
4	A3IAAVS479H7M7	0737104473	1.0	1274227200

The dataset has over 2 million data points but we will randomly sample 25K rows.

```
In [4]: amzn_ratings= amzn_ratings.sample(25000)
```

```
In [5]: amzn_ratings.shape
Out[5]: (25000, 4)
```

```
In [6]: unique_users=amzn_ratings['UserId'].unique()
        print(len(unique_users))
        24401
```

```
In [7]: unique_products=amzn_ratings['ProductId'].unique()
        print(len(unique_products))
        17378
```

Out of the 25000 rows, around 24000 users are unique, and 17000 products are unique. With these unique User Id's and Product Id's, we'll create a n X m matrix where n is the number of unique users and m is the number of unique products.

In [8]:
```
new_df=pd.DataFrame(unique_users)
new_df.columns=['UserId']
```

In [9]:
```
for product in unique_products:
    new_df[product] = None
```

In [12]:
```
for j,user in enumerate(new_df['UserId'].values):
    for i in range(amzn_ratings[amzn_ratings['UserId']==user].shape[0]):
#         print(user)
        product_id=amzn_ratings[amzn_ratings['UserId']==user]['ProductId'].iloc[i]
        rating = amzn_ratings[amzn_ratings['UserId']==user]['Rating'].iloc[i]
        new_df[product_id][j]= rating
```

In [13]:
```
new_df.sample(5)
```

Out[13]:

	UserId	B000FZXGCE	B000QAVNA0	B004WPHRZA	B0031AA5NS	B00016QZCK	B0007IQMVG	B005RRBR9O	B0009DVDTU
22173	A35SVWOQAMC0TN	None	None	None	None	None	None	None	None
6592	A3U2H4N7O2X5B8	None	None	None	None	None	None	None	None
2142	A313WR14HH8LYM	None	None	None	None	None	None	None	None
7831	A2NLMYSCIMV41M	None	None	None	None	None	None	None	None
11418	AMX565M99SD62	None	None	None	None	None	None	None	None

5 rows × 17379 columns

As most of the users have rated a single product, you would find that the ratings for most products are initially set to None and the matrix is sparse. Now, we'll randomly pick a user and find users similar to our target user.

In [57]:
```
user=pd.DataFrame(new_df.iloc[2142]) #randomly select user
user=user.drop(['UserId'])
user[user.notnull().values]
```

Out[57]:

	2142
B00538TSMU	2

In [58]:
```
subset=new_df[new_df[user[user.notnull().values].index[0]].notnull()]
subset.head()
```

Out[58]:

	UserId	B000FZXGCE	B000QAVNA0	B004WPHRZA	B0031AA5NS	B00016QZCK	B0007IQMVG	B005RRBR9O	B0009DVDTU
521	A2VUO0N2FOUGUS	None	None	None	None	None	None	None	None
621	A77QW0Z5T228E	None	None	None	None	None	None	None	None
2142	A313WR14HH8LYM	None	None	None	None	None	None	None	None
2659	A2B47983H0Z7F8	None	None	None	None	None	None	None	None
3074	A1AC2DES3OIXGX	None	None	None	None	None	None	None	None

5 rows × 17379 columns

The user we have selected (User ID A313WR14HH8LYM) has index 2142 and has rated only one product with ID **B00538TSMU** and has given it 2 stars. We have tried to find other users in our dataset that have at least rated this product.

```
In [59]: subset = subset.replace([None],0)
         subset=subset.set_index('UserId')

In [60]: subset.head()
```

Out[60]:

UserId	B000FZXGCE	B000QAVNA0	B004WPHRZA	B0031AA5NS	B00016QZCK	B0007IQMVG	B005RRBR9O	B0009DVDTU	B00HJD8NLY
A2VUO0N2FOUGUS	0	0	0	0	0	0	0	0	0
A77QW0Z5T228E	0	0	0	0	0	0	0	0	0
A313WR14HH8LYM	0	0	0	0	0	0	0	0	0
A2B47983H0Z7F8	0	0	0	0	0	0	0	0	0
A1AC2DES3OIXGX	0	0	0	0	0	0	0	0	0

5 rows × 17378 columns

```
In [61]: subset['cos_similarity']= 0.0
         for user in subset.index:
             cos=cosine_similarity(subset.loc['A313WR14HH8LYM'].values.reshape(1,-1),subset.loc[user].values.reshape(1,-1))
             subset['cos_similarity'][user]=cos
```

Now, we'll replace all NONE values with zeros because while computing similarity NONE can't be used. We have also set the User Id as the index of our subset data frame. Next, we'll compute the cosine similarity of every other user rating with the user ratings of A313WR14HH8LYM.

```
In [62]: subset['cos_similarity']
```

Out[62]:

```
UserId
A2VUO0N2FOUGUS    1.000000
A77QW0Z5T228E     1.000000
A313WR14HH8LYM    1.000000
A2B47983H0Z7F8    0.894427
A1AC2DES3OIXGX    0.894427
A2M5J16NSHDHGN    0.894427
AYA2OJR7N0OD2     0.894427
A303EFAS9Y6LGX    0.894427
ANBOE4NP1WJGP     0.894427
AP02PVZ4X4JRQ     0.894427
A1EXK150NNNTC     0.894427
A2C9Y9PV0KYP5N    0.894427
A3KMLV92H5GA4D    0.894427
A2UFMTJ2ENPI3B    0.894427
A232JZRL721KTA    0.894427
A2IYE12DM37RMG    0.894427
A1YQCNJ4S4CE7H    0.894427
AI2LO8JQ0HH5P     0.632456
A288SXIAN7SJ31    0.894427
A17JQEJXSMEVBP    0.894427
A2ESFPXBQ4KFAP    0.894427
Name: cos_similarity, dtype: float64
```

We see that the top three users are most similar since the last A313WR14HH8LYM is our target user itself we'll only take the top 2- A2VUO0N2FOUGUS and A77QW0Z5T228E are the users most similar to our target user.

Limitations of Memory-Based Collaborative Filtering Recommendation Method

- Memory-based filtering systems do not scale easily, even with 24K distinct users it took a while to find similar users to a single user.
- When using huge datasets, the memory-based collaborative filtering technique is close to impractical.

- This technique also has the “cold-start” problem, it’s hard to get recommendations for new users or recommend new products.

2) Model-Based Collaborative Filtering Recommender System

As we have seen above in the memory-based recommender system, the user-item interaction matrix is very sparse, to use it more efficiently we can reduce or compress the user-items interaction matrix into two matrices using a model. The huge sparse item matrix is decomposed into two smaller dense matrices- a user-factor matrix that has user representations and an item-factor matrix that has item representation using **Matrix Factorization Techniques for Recommender Systems.**

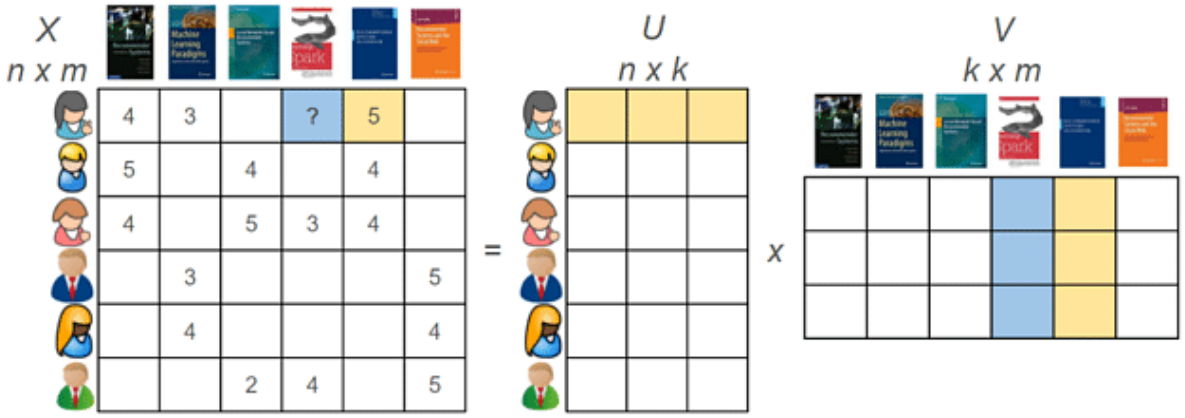


Image Credit: heartbeat. fritz

The factor that I have mentioned above is nothing but a **latent factor** that captures the similarity between users and items. A latent factor can represent a property or concept that a user or an item might have. For a movie’s dataset, latent factor can represent the genre the movie belongs to. There are many algorithms for Matrix Factorization like Singular Value Decomposition(SVD), Principal Component Analysis(PCA), and Non-Negative Matrix Factorization(NMF).

Here we will use the SVD algorithm for matrix factorization of the recommender system. The recommendation model is turned into an optimization problem and we measure how good we are in predicting the ratings of an item for a user by metrics like Root Mean Squared Error(RMSE). SVD decreases the dimensions of our user-items interaction matrix by using latent factors.

There can be useful features in the data like a User X who is a girl and a teenager who likes funny and female-oriented movies. The Princess Diaries is funny and female-oriented and would be a good recommendation. However, we do not hand engineer these features, we let the model discover these hidden representations of user interests and item attributes.

Implementing a Model-Based Recommendation System

We will use the same Amazon Beauty Products data for implementing the model recommender system. For implementing SVD for matrix factorization we would use the Surprise package which can be downloaded from here.

```
In [73]: from surprise import Reader, Dataset, SVD
         from surprise.model_selection import cross_validate
         reader = Reader()
         amzn_ratings.head()
```

Out[73]:

	UserId	ProductId	Rating	Timestamp
944445	AWGZCMOXDI6MX	B002WTC37U	5.0	1380672000
1471345	A2TPWGI8IKIZMD	B0062A5VQ0	5.0	1381536000
487130	AECBX2LY4RH0V	B000YOFQII	4.0	1392076800
1812712	A3G007LQX6KGOD	B00AWLB9EI	5.0	1377216000
645026	ACFXE3E4683II	B001D3K6U2	1.0	1364774400

Next, we'll transform the dataset to be read in an appropriate format by SVD. We'll use RMSE as the measure to evaluate SVD's performance.

```
In [78]: data = Dataset.load_from_df(amzn_ratings[['UserId', 'ProductId', 'Rating']], reader)
         svd = SVD()
         cross_validate(svd, data, measures=['RMSE', 'MAE'],cv=5)
```

```
Out[78]: {'fit_time': (1.4939820766448975,
           1.5491185188293457,
           1.5653443336486816,
           1.5732753276824951,
           1.5585706233978271),
          'test_mae': array([1.04025586, 1.04436361, 1.02914625, 1.05638693, 1.04534467]),
          'test_rmse': array([1.29339797, 1.30993288, 1.28994208, 1.318918  , 1.31245999]),
          'test_time': (0.0370633602142334,
           0.04062962532043457,
           0.0424191951751709,
           0.04679751396179199,
           0.03824901580810547)}
```

```
In [79]: trainset = data.build_full_trainset()
         svd.fit(trainset)
```

```
Out[79]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f0390bfaa20>
```

We have used cross-validation to train our recommender model using 5 folds- which basically means the whole dataset is iterated over 5 times- each time 4 parts of the dataset are used for training the model and one part for evaluating the recommender system. The mean absolute error and RMSE is given for each iteration or fold. Let's now choose a user of our choice, USER_ID A281NPSIMI1C2R, and view all the products this user has rated.

```
In [80]: amzn_ratings[amzn_ratings['UserId']=='A281NPSIMI1C2R']
```

Out[80]:

	UserId	ProductId	Rating	Timestamp
1061948	A281NPSIMI1C2R	B003JTAWOS	5.0	1361145600
1238864	A281NPSIMI1C2R	B004HKGPW6	4.0	1337990400
86088	A281NPSIMI1C2R	B00028EZME	5.0	1085788800
1061890	A281NPSIMI1C2R	B003JTA1IU	5.0	1278806400
658250	A281NPSIMI1C2R	B001E6TDYS	5.0	1133308800
1784962	A281NPSIMI1C2R	B00AE078YM	3.0	1359936000
1516138	A281NPSIMI1C2R	B006NTA9T4	5.0	1326412800
1511349	A281NPSIMI1C2R	B006L6A10G	5.0	1391731200
1969963	A281NPSIMI1C2R	B00GMIXWTU	5.0	1400371200

The next step is to predict the rating this user would give to a product that he/she has not rated before.

```
In [82]: svd.predict('A281NPSIMI1C2R','B004WPHRZA')
```

```
Out[82]: Prediction(uid='A281NPSIMI1C2R', iid='B004WPHRZA', r_ui=None, est=4.421921536637576, details={'was_impossible': False})
```

The system predicts that the user will rate the beauty product B004WPHRZA with 4.4, which is in a sense within the limits in which this user rates products, 3-5. So, model-based collaborative methods try to accurately predict a user's preference(or rating) for an item and then recommend items that a user would like based on the predicted ratings.

Advantages of Memory-Based Collaborative Filtering Recommender Systems

- They are easy to scale and can be used to work on super large datasets.
- Overfitting can be avoided if the data on which we have trained is representative of the general population.
- The prediction speed is much faster than memory-based models- since you only query the model, not the whole dataset.

Disadvantages of Memory-Based Collaborative Filtering Recommender Systems

- The quality of predictions is solely dependent on the quality of the model built.
- It is not very intuitive especially if you use Deep Learning models.

Evaluating a Recommender System

While training any Machine Learning Algorithm it is absolutely necessary to measure its performance to be able to improve upon that in the future. While evaluating most recommender systems isn't very straightforward like other machine learning systems, there are still ways to know how your recommender system is doing.

- **Mean Absolute Error**: This gives the average of the difference between actual values and predicted values in a model-based recommender system. The values would be ratings given to products by a user. Absolute is taken to remove the negative sign as we are only interested in the magnitude and not the sign of our rating. The smaller the MAE the better is your model at prediction.
- **Root Mean Squared Error**: Like MAE, Mean Squared error also takes care of the negative sign but by squaring the predictions. MSE doesn't work well when we have ratings on different scales, for this reason, we take the root of the Mean squared error to get RMSE. It normalizes the mean result to the same scale. RMSE is also better at dealing with outliers.
- **Personalization**: How personalized the recommendations are is also very important. Your recommender should not recommend the same movies to different kinds of users. This can be checked by checking the similarity in a recommendation for users.
- **Human Evaluation**: Human interpretation is absolutely necessary to make sure the recommender does not only recommend popular items to the user. It

should also be ensured that the recommendations are diverse and explainable. Amazon makes explainable results by showing, “Because you liked X, you might also like ……. Customers who bought this also bought ….”.