RECOMMENDATION ENGINES

Prepared By….

Sandeep Singh [C0727422]

Febin Roy Edakalathur [C0723559]

Amal Das [C0724011]

Alan Salo [C0727079]

Enamol Hassan [C0728399]

Agenda

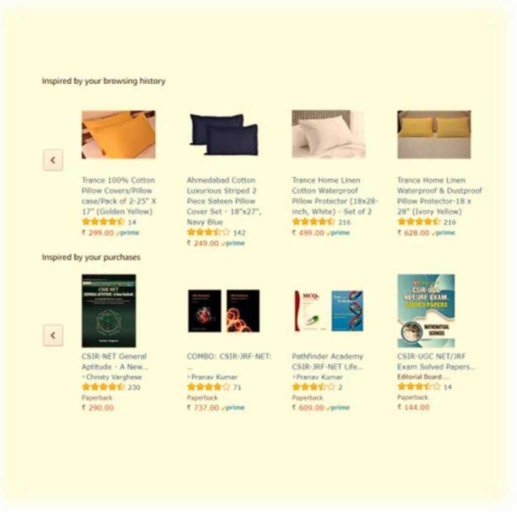
- Introduction
- Applications
- Recommendation Models Types
- Similarity Algorithms
- Implementation using R
- Implementation using Python

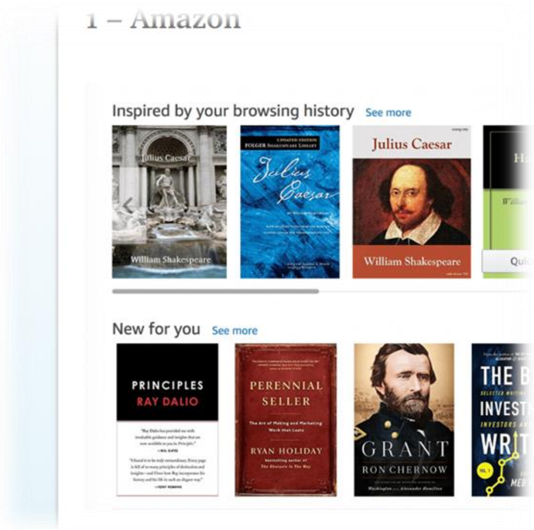
Introduction

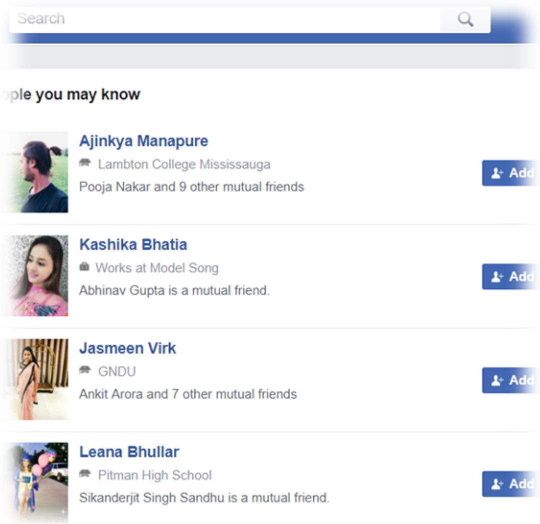
- A **recommendation engine** filters the data using different algorithms and recommends the most relevant items to users.
- It first captures the past behavior of a customer and based on that, recommends products which the users might be likely to buy.

Applications

- Online Shopping
- Entertainment
- Advertisements
- Social Sites
- Travel Advisor
- E - Learning
- Mobile Applications

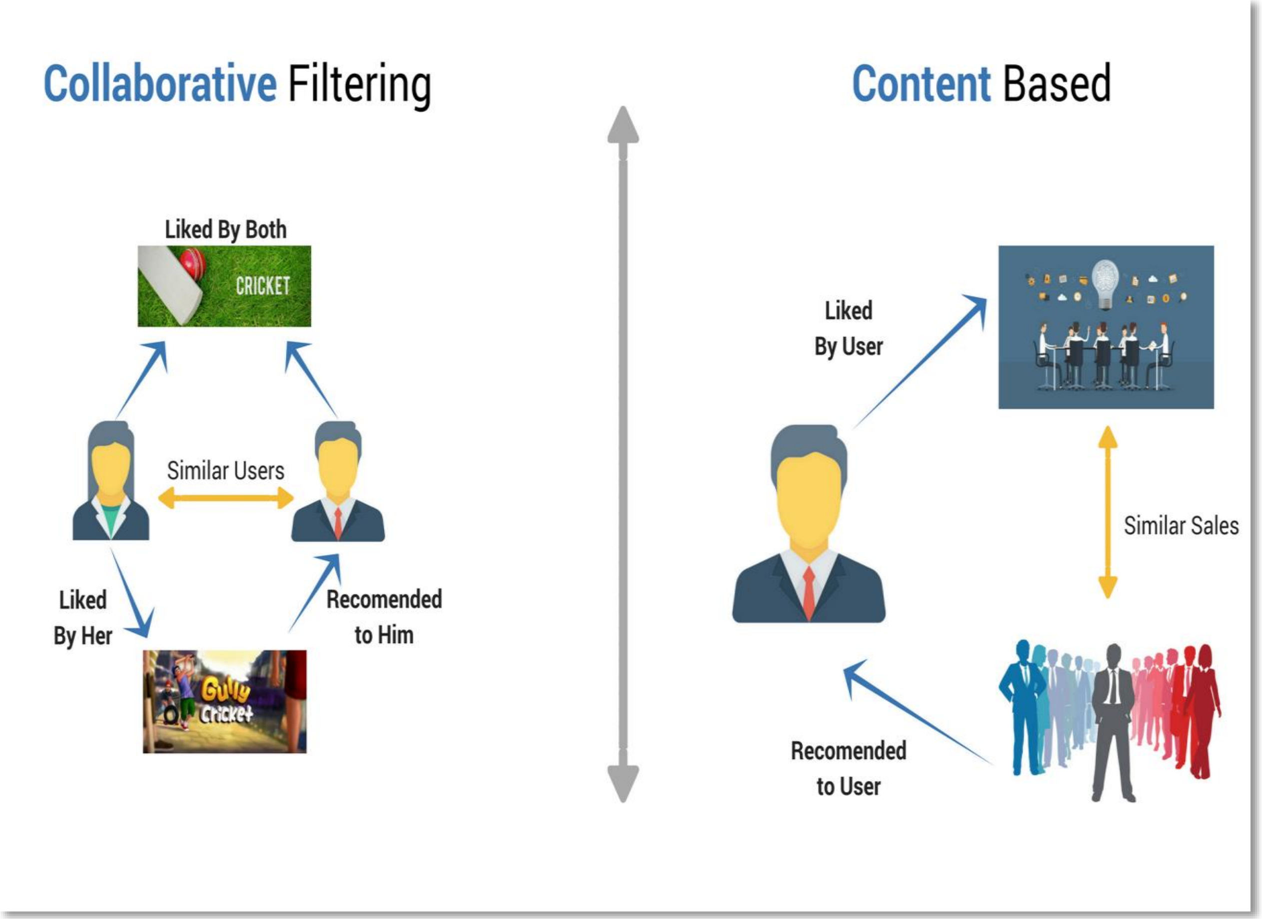






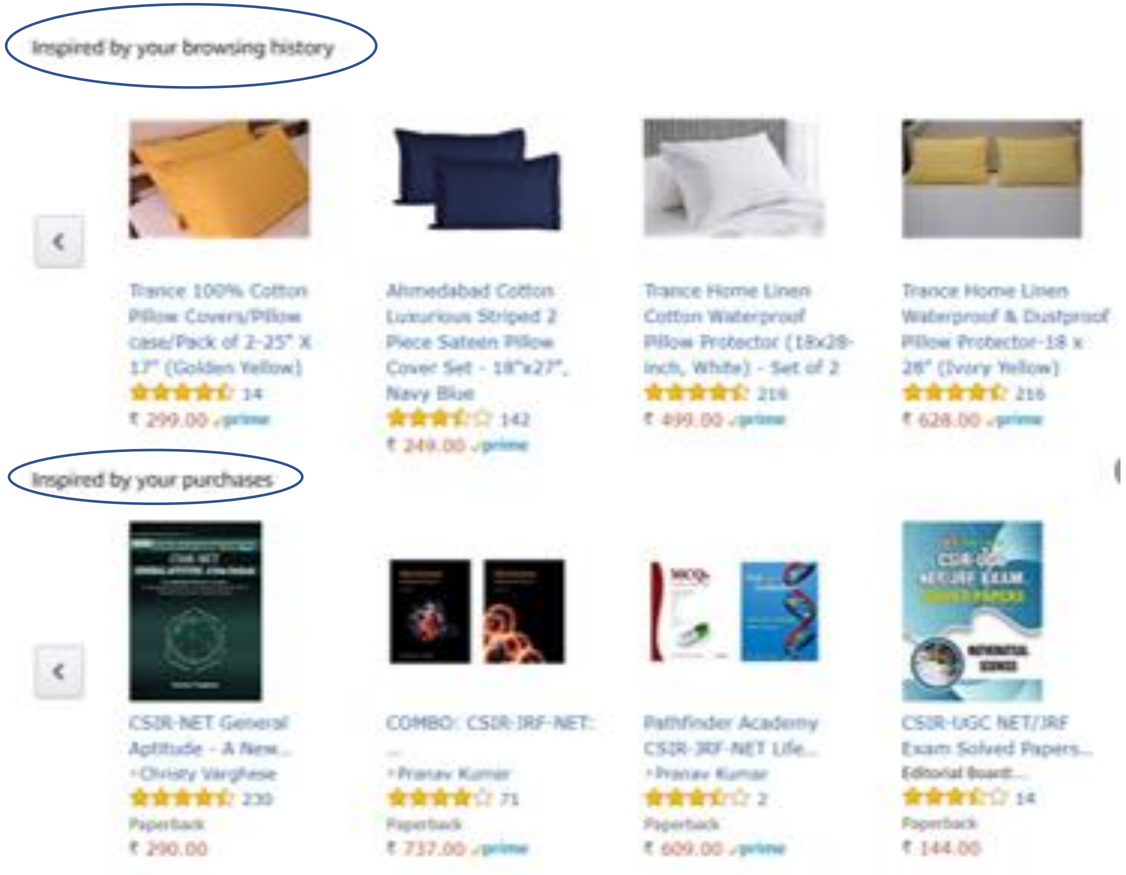
Recommendation Models Types

- Content based filtering
- Collaborative filtering



Content Based Filtering

- Selects items based on the correlation between the content of the items and the user's preferences.
- Works with existing profiles of users and particular item.
- Does not depend on lots of user data, so it is possible to give recommendations to even your first customer.
- For instance – In adjacent pic you can see amazon's recommender system recommends products based on viewed and purchased history.

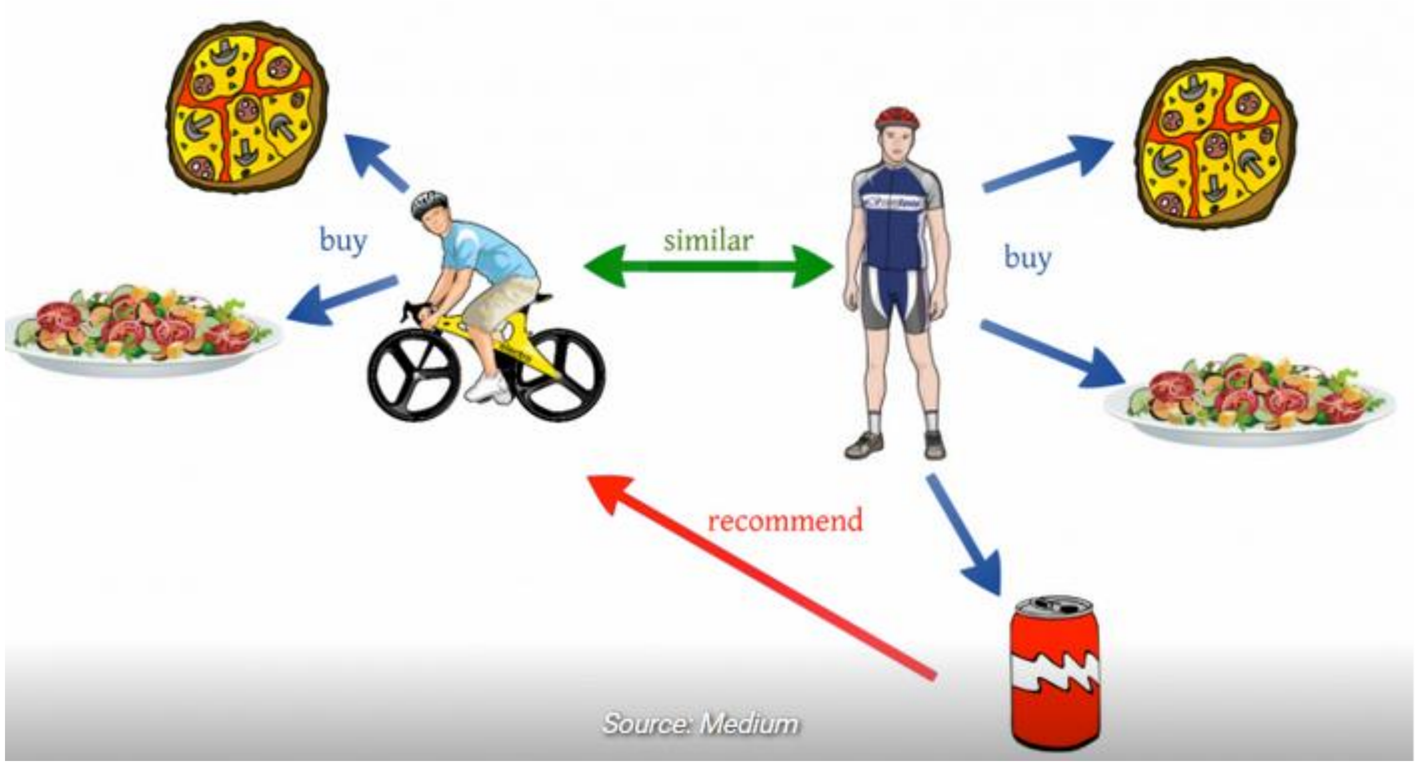


Problems in Content Based Filtering

- Requires manual or automatic indexing – Item feature do not capture everything.
- Needs to learn what content features are important for the users, so takes time.
- It assumes that user’s taste and preference remains more constant over time.
- Provision of discovering something fortunate, especially while looking for something entirely unrelated is absent.

Collaborative Filtering

- Collaborative Filtering is the process of filtering or evaluating items using the opinions of other people.
- For each user, recommender systems recommend items based on how **similar** users liked the item.
- Let's say Alice and Bob have similar interests in video games. Alice recently played and enjoyed the game Legend of Zelda: Breathe of the Wild. Bob has not played this game, but because the system has learned that Alice and Bob have similar tastes, it recommends this game to Bob.



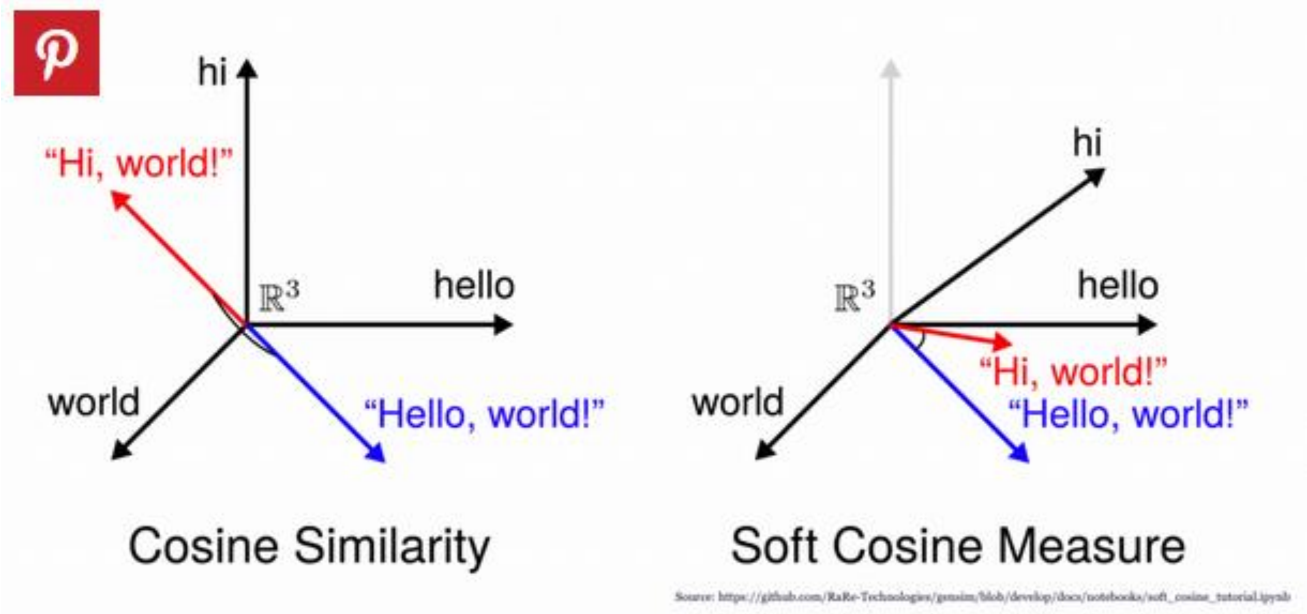
Similarity Algorithms

- Jaccard Similarity
- Cosine Similarity
- Pearson Coefficient

Cosine Similarity

- With this, we are going to evaluate the similarity between two vectors based on the angle between them.
- The smaller the angle, the more similar the two vectors are.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$



Implementation using R

```
#Load datasets

movies=read.csv(file.choose())
head(movies)

links=read.csv(file.choose())
head(links)

ratings=read.csv(file.choose())
head(ratings)

tags=read.csv(file.choose())
head(tags)

#Import the reshape2 and  stringi library.

library(stringi)
library(reshape2)

#Create ratings matrix with rows as users and columns as movies. We don't need timestamp
rmatrix = dcast(ratings, userId~movieId, value.var = "rating", na.rm=FALSE)
print(rmatrix)

#Removing user ids
rmatrix = as.matrix(rmatrix[,-1])

library(recommenderlab)

#Convert ratings matrix to real rating matrx which makes it dense
real_rmatrix = as(rmatrix, "realRatingMatrix")
print(real_rmatrix)
```

```r
#Create Recommender Model. The parameters are UBCF and Cosine similarity. We take 10 near
rec_mod = Recommender(real_rmatrix, method = "UBCF", param=list(method="Cosine",nn=10))


#Obtain top 5 recommendations for 1st user entry in dataset
Top_5_pred = predict(rec_mod, real_rmatrix[1], n=5)
Top_5_pred


#Convert the recommendations to a list
Top_5_List = as(Top_5_pred, "list")
Top_5_List


library(dplyr)
#We convert the list to a dataframe and change the column name to movieId</em>
Top_5_df=data.frame(Top_5_List)
colnames(Top_5_df)="movieId"

#Since movieId is of type integer in Movies data, we typecast id in our recommendations a
Top_5_df$movieId=as.numeric(levels(Top_5_df$movieId))

#Merge the movie ids with names to get titles and genres</em>
names=left_join(Top_5_df, movies, by="movieId")

#Print the titles and genres</em>
names
```

Output

```
> movies=read.csv(file.choose())
> head(movies)
  movieId                              title                                      genres
1       1                   Toy Story (1995) Adventure|Animation|Children|Comedy|Fantasy
2       2                     Jumanji (1995)                  Adventure|Children|Fantasy
3       3            Grumpier Old Men (1995)                              Comedy|Romance
4       4           Waiting to Exhale (1995)                        Comedy|Drama|Romance
5       5 Father of the Bride Part II (1995)                                      Comedy
6       6                        Heat (1995)                       Action|Crime|Thriller

> #Create Recommender Model. The parameters are UBCF and Cosine similarity. We take 10 nearest
 neighbours
> rec_mod = Recommender(real_rmatrix, method = "UBCF", param=list(method="Cosine",nn=10))
>
>
> #Obtain top 5 recommendations for 1st user entry in dataset
> Top_5_pred = predict(rec_mod, real_rmatrix[1], n=5)
> Top_5_pred
Recommendations as ‘topNList’ with n = 5 for 1 users.
>
>
> #Convert the recommendations to a list
> Top_5_List = as(Top_5_pred, "list")
> Top_5_List
[[1]]
[1] "58559" "1207"  "1721"  "1357"  "8533"

> #Merge the movie ids with names to get titles and genres</em>
> names=left_join(Top_5_df, movies, by="movieId")
>
> #Print the titles and genres</em>
> names
  movieId                        title                    genres
1    1207 To Kill a Mockingbird (1962)                     Drama
2    1357                 Shine (1996)             Drama|Romance
3    1721               Titanic (1997)             Drama|Romance
4   58559      Dark Knight, The (2008) Action|Crime|Drama|IMAX
5    8533         Notebook, The (2004)             Drama|Romance
```

Implementation using Python

```
import turicreate
import pandas as pd
import numpy as np
```

```
#Reading ratings file:
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings = pd.read_csv('u.data', sep='\t', names=r_cols,encoding='latin-1')
```

```
print(ratings.shape)
ratings.head()
```

```
(100000, 4)
```

	user_id	movie_id	rating	unix_timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

```python
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings_train = pd.read_csv('ua.base', sep='\t', names=r_cols, encoding='latin-1')
ratings_test = pd.read_csv('ua.test', sep='\t', names=r_cols, encoding='latin-1')
ratings_train.shape, ratings_test.shape
```

```
((90570, 4), (9430, 4))
```

```python
n_users = ratings.user_id.unique().shape[0]
n_items = ratings.movie_id.unique().shape[0]
```

```python
data_matrix = np.zeros((n_users, n_items))
for line in ratings.itertuples():
    data_matrix[line[1]-1, line[2]-1] = line[3]
```

```python
from sklearn.metrics.pairwise import pairwise_distances
user_similarity = pairwise_distances(data_matrix, metric='cosine')
item_similarity = pairwise_distances(data_matrix.T, metric='cosine')
```

```python
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        #We use np.newaxis so that mean_user_rating has same format as ratings
        ratings_diff = (ratings - mean_user_rating[:, np.newaxis])
        pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff) / np.array([np.abs(similarity).sum(axis=1)]).T
    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])
    return pred
```

```python
user_prediction = predict(data_matrix, user_similarity, type='user')
item_prediction = predict(data_matrix, item_similarity, type='item')
```

```python
train_data = turicreate.SFrame(ratings_train)
test_data = turicreate.SFrame(ratings_test)
```

```python
popularity_model = turicreate.popularity_recommender.create(train_data, user_id='user_id', item_id='movie_id', target='rating
```

```
popularity_recomm = popularity_model.recommend(users=[1,2,3,4,5],k=5)
popularity_recomm.print_rows(num_rows=25)
```

```
+---------+----------+-------+------+
| user_id | movie_id | score | rank |
+---------+----------+-------+------+
|    1    |   1599   |  5.0  |  1   |
|    1    |   1201   |  5.0  |  2   |
|    1    |   1189   |  5.0  |  3   |
|    1    |   1122   |  5.0  |  4   |
|    1    |   814    |  5.0  |  5   |
|    2    |   1599   |  5.0  |  1   |
|    2    |   1201   |  5.0  |  2   |
|    2    |   1189   |  5.0  |  3   |
|    2    |   1122   |  5.0  |  4   |
|    2    |   814    |  5.0  |  5   |
|    3    |   1599   |  5.0  |  1   |
|    3    |   1201   |  5.0  |  2   |
|    3    |   1189   |  5.0  |  3   |
|    3    |   1122   |  5.0  |  4   |
|    3    |   814    |  5.0  |  5   |
|    4    |   1599   |  5.0  |  1   |
|    4    |   1201   |  5.0  |  2   |
|    4    |   1189   |  5.0  |  3   |
|    4    |   1122   |  5.0  |  4   |
|    4    |   814    |  5.0  |  5   |
|    5    |   1599   |  5.0  |  1   |
|    5    |   1201   |  5.0  |  2   |
|    5    |   1189   |  5.0  |  3   |
```

```
#Training the model
item_sim_model = turicreate.item_similarity_recommender.create(train_data, user_id='user_id', item_id='movie_id', target='rat

#Making recommendations
item_sim_recomm = item_sim_model.recommend(users=[1,2,3,4,5],k=5)
item_sim_recomm.print_rows(num_rows=25)
```

Output

user_id	movie_id	score	rank
1	423	0.988204108622238	1
1	202	0.949776457466242	2
1	655	0.8052522974614879	3
1	403	0.7722151641172307	4
1	568	0.7653118053465399	5
2	50	1.1256258487701416	1
2	181	1.0651773168490484	2
2	7	0.9998190838557023	3
2	121	0.94162796323116	4
2	9	0.831989913032605	5
3	313	0.6353766620159149	1
3	328	0.6032880300825293	2
3	315	0.5422587123784152	3
3	331	0.5355071858926252	4
3	332	0.5316696112806146	5
4	50	1.1311477082116264	1
4	288	1.0487151145935059	2
4	181	0.9505999386310577	3
4	7	0.9417778807027	4
4	302	0.9139021464756557	5
5	195	1.0183543920516969	1
5	202	0.9353599468866984	2
5	56	0.8479394096316714	3

- You receive recommended movie_id for user_id 1,2,3,4 and 5.

References

- https://towardsdatascience.com/how-to-build-a-recommendation-engine-quick-and-simple-aec8c71a823e
- https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/
- https://medium.com/@mark.rethana/building-a-song-recommendation-system-using-cosine-similarity-and-euclidian-distance-748fdfc832fd
- https://towardsdatascience.com/collaborative-filtering-based-recommendation-systems-exemplified-ecbffe1c20b1
- https://en.wikipedia.org/wiki/Cosine_similarity
- https://www.data-mania.com/blog/how-to-build-a-recommendation-engine-in-r/
- https://www.analyticsvidhya.com/blog/2016/03/exploring-building-banks-recommendation-system/