**Links :**

[**https://github.com/grahamjenson/list\_of\_recommender\_systems**](https://github.com/grahamjenson/list_of_recommender_systems)

[**https://www.pinterest.de/dataliftoff/recommender-systems/python-libraries/**](https://www.pinterest.de/dataliftoff/recommender-systems/python-libraries/)

**-**[**https://www.kaggle.com/gunnvant/building-content-recommender-tutorial**](https://www.kaggle.com/gunnvant/building-content-recommender-tutorial)

1. Recommendation engines used by the official ted page, will be a degrees of magnitude more sophisticated than what I can demonstrate here and would also involve use of some sort of historical user-item interaction data.
2. generate recommendations just using content when you don't have any user-item interaction data
3. when you are starting out new and still want to provide the consumers of your content relevant contextual recommendations
4. recommend talks based on the similarity of their content, the first thing I will have to do is to, create a representation of the transcripts

that are amenable to comparison. One way of doing this is to create a tfidf vector for each transcript.

TfidfVectorizer

cosine\_similarity(matrix)

get\_similar\_articles

 couple of more things:

1. tf-idf with unigrams, you can try using bigrams and see if you get better results.
2. Try using pre-trained word vectors such as word2vec to create vector representation of just the Titles and try to find similarity using cosine distance

[**https://www.kaggle.com/saurav9786/recommender-system-using-amazon-reviews**](https://www.kaggle.com/saurav9786/recommender-system-using-amazon-reviews)

* Amazon uses currently item-item collaberrative filtering, which scales to massive datasets and produces high quality recommendation system in the real time.
* This system is a kind of a information filtering system which seeks to predict the "rating" or preferences which user is interested in.

Types of recommendations

There are mainly 6 types of the recommendations systems :-

1. Popularity based systems :- It works by recommeding items viewed and purchased by most people and are rated high.It is not a personalized recommendation.
2. Classification model based:- It works by understanding the features of the user and applying the classification algorithm to decide whether the user is interested or not in the prodcut.
3. Content based recommedations:- It is based on the information on the contents of the item rather than on the user opinions.The main idea is if the user likes an item then he or she will like the "other" similar item.
4. Collaberative Filtering:- It is based on assumption that people like things similar to other things they like, and things that are liked by other people with similar taste. it is mainly of two types: a) User-User b) Item -Item
5. Hybrid Approaches:- This system approach is to combine collaborative filtering, content-based filtering, and other approaches .
6. Association rule mining :- Association rules capture the relationships between items based on their patterns of co-occurrence across transactions.

Attribute Information:[¶](https://www.kaggle.com/saurav9786/recommender-system-using-amazon-reviews#Attribute-Information:)

● userId : Every user identified with a unique id

● productId : Every product identified with a unique id

● Rating : Rating of the corresponding product by the corresponding user

● timestamp : Time of the rating ( ignore this column for this exercise)

1. Popularity based

1. *Check the distribution of the rating*
2. On the basis of Rating.count()

CF is based on the idea that the best recommendations come from people who have similar tastes. In other words, it uses historical item ratings of like-minded people to predict how someone would rate an item.Collaborative filtering has two sub-categories that are generally called memory based and model-based approaches

Memory-based collaborative filtering system

from surprise import KNNWithMeans

Parameter : *user\_based true/false to switch between user-based or item-based collaborative filtering*

Model-based collaborative filtering system

 advantage of these methods is that they are able to recommend a larger number of items to a larger number of users, compared to other methods like memory based approach. They have large coverage, even when working with large sparse matrices.

new\_df1.pivot\_table

from sklearn.decomposition import TruncatedSVD

np.corrcoef(decomposed\_matrix)

[**https://www.kaggle.com/kanncaa1/recommendation-systems-tutorial**](https://www.kaggle.com/kanncaa1/recommendation-systems-tutorial)

User Based Collaborative Filtering

* Collaborative filtering is making recommend according to combination of your experience and experiences of other people.
* First we need to make user vs item matrix.
* Each row is users and each columns are items like movie, product or websites
* Secondly, computes similarity scores between users.
* Each row is users and each row is vector.
* Compute similarity of these rows (users).
* Thirdly, find users who are similar to you based on past behaviours
* Finally, it suggests that you are not experienced before.
* Lets make an example of user based collaborative filtering
* Think that there are two people
* First one watched 2 movies that are lord of the rings and hobbit
* Second one watched only lord of the rings movie
* User based collaborative filtering computes similarity of these two people and sees both are watched a lord of the rings.
* Then it recommends hobbit movie to second one as it can be seen picture \*
* User based collaborative filtering has some problems
* In this system, each row of matrix is user. Therefore, comparing and finding similarity between of them is computationaly hard and spend too much computational power.
* Also, habits of people can be changed. Therefore making correct and useful recommendation can be hard in time.
* In order to solve these problems, lets look at another recommender system that is item based collaborative filtering

1. Item Based Collaborative Filtering

* In this system, instead of finding relationship between users, used items like movies or stuffs are compared with each others.
* In user based recommendation systems, habits of users can be changed. This situation makes hard to recommendation. However, in item based recommendation systems, movies or stuffs does not change. Therefore recommendation is easier.
* On the other hand, there are almost 7 billion people all over the world. Comparing people increases the computational power. However, if items are compared, computational power is less.
* In item based recommendation systems, we need to make user vs item matrix that we use also in user based recommender systems.
* Each row is user and each column is items like movie, product or websites.
* However, at this time instead of calculating similarity between rows, we need to calculate similarity between columns that are items like movies or stuffs.
* Lets look at how it is works.
* Firstly, there are similarities between lord of the rings and hobbit movies because both are liked by three different people. There is a similarity point between these two movies.
* If the similarity is high enough, we can recommend hobbit to other people who only watched lord of the rings movie as it can be seen in figure below. \*
* **Item Based Collaborative Filtering**

["userId","movieId","rating"]

data.pivot\_table(index = ["userId"],columns = ["title"],values = "rating")

movie\_watched = pivot\_table["Bad Boys (1995)"]   
similarity\_with\_other\_movies = pivot\_table.corrwith(movie\_watched)  *# find correlation between "Bad Boys (1995)" and other movies*   
similarity\_with\_other\_movies = similarity\_with\_other\_movies.sort\_values(ascending=False)

**[https://www.kaggle.com/sriharshavogeti/collaborative-recommender-system-on-goodreads](https://www.kaggle.com/sriharshavogeti/collaborative-recommender-system-on-goodreads" \t "_blank)**

Naive item-similarity based recommendar system

Dataset =

|  |  |  |
| --- | --- | --- |
| **book\_id** | **user\_id** | **rating** |

Create dictionary corresponding to each book id with its mapping and its key value as user\_ids:rating

dictVectorizer = DictVectorizer(sparse=True)

vector = dictVectorizer.fit\_transform(listOfDictonaries)

cosine\_similarity(vector)

np.argsort(pairwiseSimilarity[row])[-7:-2][::-1]

[**https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender**](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender)

* [11. Recommendation Engines](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender#11)
  + [11.1. Mean Rating](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender#11.1)
  + [11.2. Weighted Mean Rating](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender#11.2)
  + [11.3. Cosine User-User Similarity](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender#11.3)
  + [11.4. Cosine TFIDF Movie Description Similarity](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender#11.4)
  + [11.5. Matrix Factorisation With Keras And Gradient Descent](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender#11.5)
  + [11.6. Deep Learning With Keras](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender#11.6)
  + [11.7. Deep Hybrid System With Metadata And Keras](https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender#11.7)

df\_train.pivot\_table(index='User', columns='Movie', values='Rating')