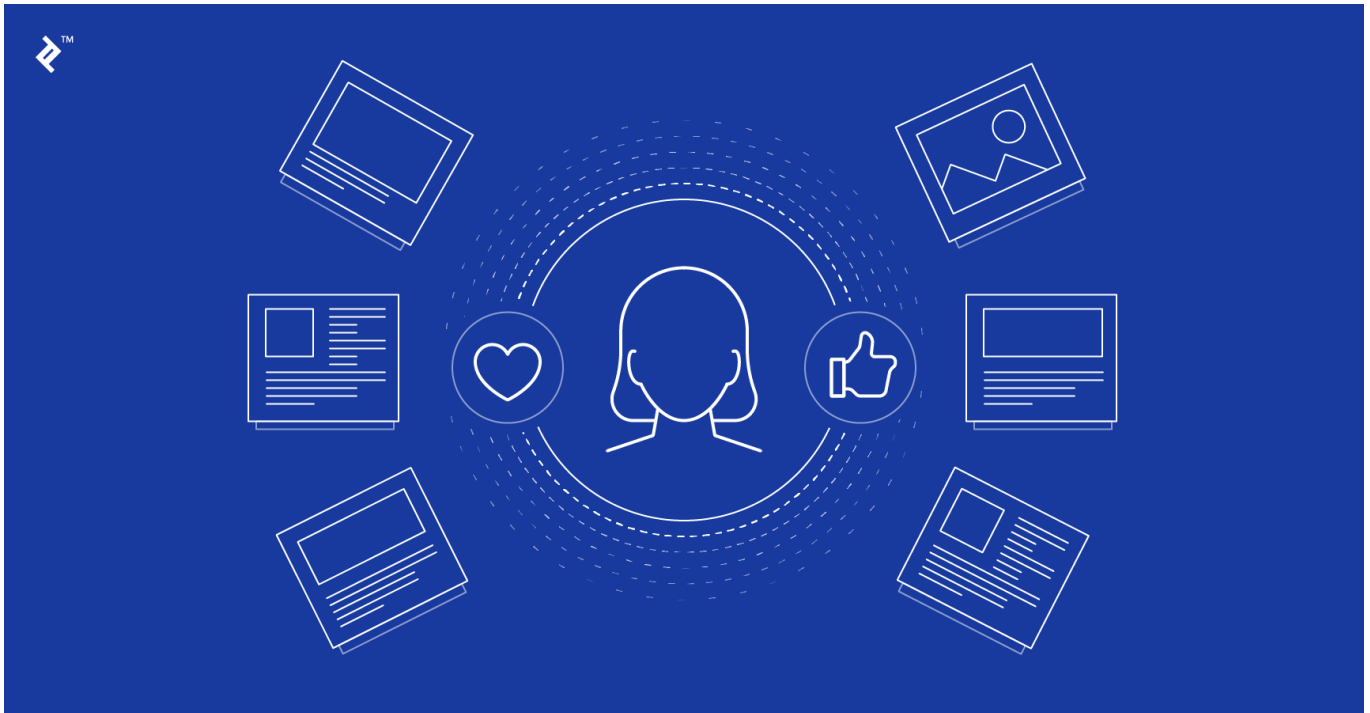
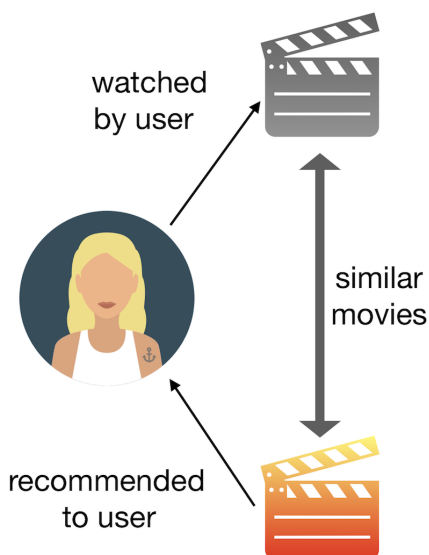


▼ Recommendation Engine For Amazon

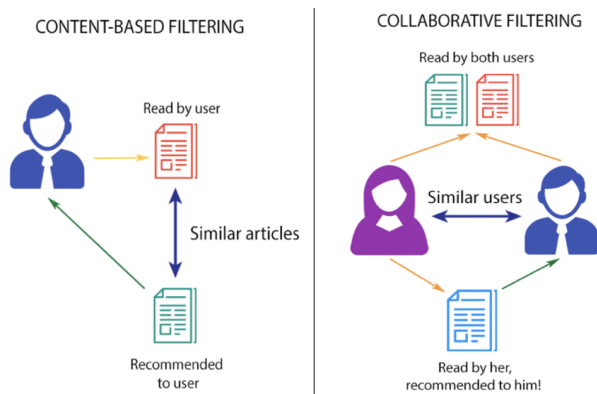


Recommender systems can be classified into Two types:

Content-based recommenders: suggest similar items based on a particular item. This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations. The general idea behind these recommender systems is that if a person likes a particular item, he or she will also like an item that is similar to it. And to recommend that, it will make use of the user's past item metadata. A good example could be YouTube, where based on your history, it suggests you new videos that you could potentially watch.



Collaborative filtering engines: these systems are widely used, and they try to predict the rating or preference that a user would give an item-based on past ratings and preferences of other users. Collaborative filters do not require item metadata like its content-based counterparts.



▼ Analysis Task

Exploratory Data Analysis:

- Which movies have maximum views/ratings?
- What is the average rating for each movie? Define the top 5 movies with the maximum ratings.
- Define the top 5 movies with the least audience.¶

```
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
df = pd.read_csv('/content/1655387347_amazonmoviesandtvratings_2.zip')
```

```
df.head()
```

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie
0	A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	NaN


df.shape

(4848, 207)

3	AVIY68KFPQ57D	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN
---	---------------	-----	-----	-----	-----	-----	-----	-----	-----

df_org = df.copy()


df.describe().T

	count	mean	std	min	25%	50%	75%	max	
Movie1	1.0	5.000000	NaN	5.0	5.00	5.0	5.0	5.0	
Movie2	1.0	5.000000	NaN	5.0	5.00	5.0	5.0	5.0	
Movie3	1.0	2.000000	NaN	2.0	2.00	2.0	2.0	2.0	
Movie4	2.0	5.000000	0.000000	5.0	5.00	5.0	5.0	5.0	
Movie5	29.0	4.103448	1.496301	1.0	4.00	5.0	5.0	5.0	
...	
Movie202	6.0	4.333333	1.632993	1.0	5.00	5.0	5.0	5.0	
Movie203	1.0	3.000000	NaN	3.0	3.00	3.0	3.0	3.0	
Movie204	8.0	4.375000	1.407886	1.0	4.75	5.0	5.0	5.0	
Movie205	35.0	4.628571	0.910259	1.0	5.00	5.0	5.0	5.0	
Movie206	13.0	4.923077	0.277350	4.0	5.00	5.0	5.0	5.0	

206 rows × 8 columns

Task 1 - Which movies have maximum views/ratings?

#Movie with highest views
df.describe().T['count'].sort_values(ascending=False)[:1].to_frame()

	count	
Movie127	2313.0	

#Movie with highest Ratings
df.drop('user_id',axis=1).sum().sort_values(ascending=False)[:1].to_frame()

	0	
Movie127	9511.0	

Task 2 - What is the average rating for each movie? Define the top 5 movies with the maximum ratings

```
df.drop('user_id',axis=1).mean().sort_values(ascending=False)[:5].to_frame()
```

	0
Movie1	5.0
Movie66	5.0
Movie76	5.0
Movie75	5.0
Movie74	5.0

Task 3 - Define the top 5 movies with the least audience

```
df.describe().T['count'].sort_values(ascending=True)[:5].to_frame()
```

	count
Movie1	1.0
Movie71	1.0
Movie145	1.0
Movie69	1.0
Movie68	1.0

Task 4 - Recommendation Model

```
!pip install scikit-surprise
from surprise import Reader
from surprise import accuracy
from surprise import Dataset
from surprise.model_selection import train_test_split
from surprise import SVD
from surprise.model_selection import cross_validate
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/>
 Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.7/dist-pack
 Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.7/dist-packag
 Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
 Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-package
 Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-packages

```
df_melt = df.melt(id_vars = df.columns[0],value_vars=df.columns[1:],var_name="Movies",valu
```

```
df_melt
```

	user_id	Movies	Rating	
0	A3R5OBKS7OM2IR	Movie1	5.0	
1	AH3QC2PC1VTGP	Movie1	NaN	
2	A3LKP6WPMP9UKX	Movie1	NaN	
3	AVIY68KEPQ5ZD	Movie1	NaN	
4	A1CV1WROP5KTTW	Movie1	NaN	
...	
998683	A1IMQ9WMFYKWH5	Movie206	5.0	
998684	A1KLIKPUF5E88I	Movie206	5.0	
998685	A5HG6WFZLO10D	Movie206	5.0	
998686	A3UU690TWXCG1X	Movie206	5.0	
998687	AI4J762YI6S06	Movie206	5.0	

998688 rows × 3 columns

```
rd = Reader()
data = Dataset.load_from_df(df_melt.fillna(0),reader=rd)
data

<surprise.dataset.DatasetAutoFolds at 0x7f28307b4a90>
```

```
trainset, testset = train_test_split(data,test_size=0.25)
```

```
#Using SVD (Singular Value Descomposition)
svd = SVD()
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f281877f750>
```

```
pred = svd.test(testset)
```

```
accuracy.rmse(pred)
```

```
RMSE: 1.0257
1.0256918506819663
```

```
accuracy.mae(pred)
```

```
MAE: 1.0120
1.0119898562536265
```

```
cross_validate(svd, data, measures = ['RMSE', 'MAE'], cv = 3, verbose = True)
```

Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

	Fold 1	Fold 2	Fold 3	Mean	Std
RMSE (testset)	1.0258	1.0263	1.0264	1.0261	0.0003
MAE (testset)	1.0119	1.0122	1.0121	1.0121	0.0001
Fit time	42.22	41.97	43.26	42.48	0.56
Test time	3.99	3.61	3.52	3.71	0.20

```
{'test_rmse': array([1.025757, 1.02626198, 1.02635116]),
 'test_mae': array([1.01188035, 1.01216395, 1.01211925]),
 'fit_time': (42.22485136985779, 41.965415239334106, 43.25736141204834),
 'test_time': (3.9892451763153076, 3.610214948654175, 3.5221920013427734)}
```

```
def repeat(ml_type,dframe):
    rd = Reader()
    data = Dataset.load_from_df(dframe,reader=rd)
    print(cross_validate(ml_type, data, measures = ['RMSE', 'MAE'], cv = 3, verbose = True)
    print("--"*15)
    usr_id = 'A3R50BKS70M2IR'
    mv = 'Movie1'
    r_u = 5.0
    print(ml_type.predict(usr_id,mv,r_ui = r_u,verbose=True))
    print("--"*15)
```

```
repeat(SVD(),df_melt.fillna(df_melt['Rating'].mean()))
```

Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

	Fold 1	Fold 2	Fold 3	Mean	Std
RMSE (testset)	0.0829	0.0874	0.0867	0.0857	0.0020
MAE (testset)	0.0097	0.0098	0.0097	0.0098	0.0001
Fit time	41.34	42.81	41.45	41.87	0.67
Test time	3.81	3.80	3.67	3.76	0.07

```
{'test_rmse': array([0.08286557, 0.08739083, 0.08669842]), 'test_mae': array([0.0097
-----
user: A3R50BKS70M2IR item: Movie1      r_ui = 5.00    est = 4.40    {'was_impossible':
user: A3R50BKS70M2IR item: Movie1      r_ui = 5.00    est = 4.40    {'was_impossible':
-----
```

```
#trying grid search and find optimum hyperparameter value for n_factors
from surprise.model_selection import GridSearchCV
```

```
param_grid = {'n_epochs':[20,30],
              'lr_all':[0.005,0.001],
              'n_factors':[50,100]}
```

```
gs = GridSearchCV(SVD,param_grid,measures=['rmse','mae'],cv=3)
data1 = Dataset.load_from_df(df_melt.fillna(df_melt['Rating'].mean()),reader=rd)
```

```
gs.fit(data1)
```

```
gs.best_score
```

```
{'rmse': 0.08477932654295557, 'mae': 0.008972411775508957}
```

```
print(gs.best_score["rmse"])
```

```
print(gs.best_params["rmse"])
```

```
0.08477932654295557
```

```
{'n_epochs': 30, 'lr_all': 0.001, 'n_factors': 50}
```

[Colab paid products](#) - [Cancel contracts here](#)

0s completed at 1:03 AM

