Introduction to Recommender Systems: Collaborative Filtering for Movie Recommendations

Abstract:

In the flourishing and thriving world, the information and demand for only required information is increasing rapidly. To make the data more customized to the users, Recommender Systems are developed. The algorithms to build Recommender Systems include Collaborative Filtering, Content Filtering, and Hybrid Filtering, a combination of Collaborative and Content Filtering. Comparatively, Collaborative Filtering gives more compatibility as compared to content-based technology. Collaborative filtering helps in suggesting similar items and ideas based on the user's thinking and history and using this provides the best recommendation. In this chapter, we will focus on the disadvantages of collaborative filtering and try to make the existing algorithm more efficient. Collaborative Filtering can be further classified into Model-Based and Memory-based techniques. This study compares these two different methods used for collaborative filtering for movie recommendations and finds out their advantages, disadvantages, performance, and accuracy. Furthermore, it concludes which method is best suited and most efficient.

Literature Review:

In today's digital world, the constant growth of data and information is a challenge and an opportunity. Users are saturated with massive amounts of data, and the need for tailored and personalized information has never been greater. Recommender systems are a crucial solution to this information overload. They aim to provide users with content and products that meet their preferences and needs. These systems are based on several algorithms, the main ones being collaborative filtering, content filtering, and hybrid filtering. Here, in the field of collaborative filtering, we are focusing primarily on its model-based and memory-based subcategories, examining their advantages, disadvantages, performance, and accuracy in the context of movie recommendations.

Collaborative Filtering is an integral part of recommendation systems. It's all about using what lots of people do and like to suggest things to others. It's really good at telling you what you might like based on what you've done before and what people similar to you have done. It's like listening to what many people say and using that to give you suggestions you'll probably like. This way of making suggestions works well in the real world, and sometimes it even helps you find cool stuff you didn't expect. That's why a lot of people think it's a great way to do things.

Within Collaborative Filtering, two main techniques include Model-Based and Memory-Based approaches. Model-based methods employ statistical models to capture user-item interactions, offering scalability and accuracy advantages. On the other hand, Memory-Based techniques are based on direct similarity measures between users or items, giving simplicity and transparency.

Model-Based Collaborative Filtering is useful in scalability and efficiency, as it can easily handle huge datasets. Its ability to discover latent patterns improves recommendation accuracy. However, it may suffer from limited interpretability and too much focus on one object. On the other hand, Memory-Based Collaborative Filtering excels in simplicity and interpretability but can face an obstacle with thinly distributed data and scalability concerns. It also faces challenges when dealing with cold-start problems for new users or items.

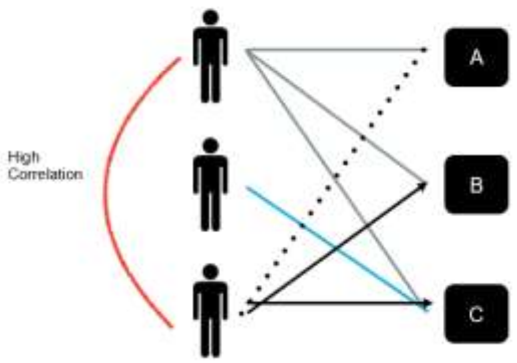
Performance and accuracy parameters play an important role in assessing the effectiveness of Collaborative Filtering techniques. Techniques like Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) capture the predictive accuracy of these methods.

The choice between Model-Based and Memory-Based Collaborative Filtering techniques depends on specific application requirements and dataset characteristics. Model-based methods are preferred in scenarios where scalability and efficiency are required, while Memory-Based in transparency and simplicity requirement. Observations and real-world evaluations will continue to guide the selection of the most suitable approach. Further versions may focus on hybridization, combining the strengths of both approaches to further enhance the efficiency and accuracy of Collaborative Filtering in delivering personalized recommendations, particularly in domains like movie recommendations. In the end, the quest for more efficient Collaborative Filtering algorithms still remains, driven by the relentless demand for personalized information in our information-rich world.

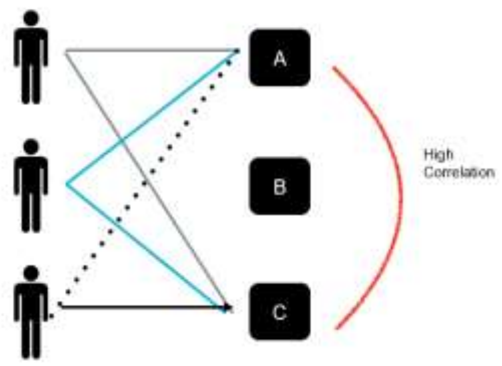
Collaborative Filtering

- Collaborating Filtering is a part of the recommender system. It is a popular technique that makes actual predictions based on user's interests.
- Collaborative Filtering can be done on two bases - User-based and Item-based Filtering.
- In User-based Filtering, the system gives recommendations to the user based on the interests of other users similar to that user.
- In Item-based Filtering, the system gives recommendations based on the similarity between the items rather than the users. Common similarity matrices

User-Based



Item-Based



Difference between User-Based Filtering & Item-Based Filtering:

Parameters	User-Based	Item-Based
Focus	Recommends on the basis of preferences of similar users.	Recommends similar items to the ones the user prefers.
Similarity	Uses similarity measures like cosine similarity or Pearson correlation.	Compares the nature of items with how users have connected with them.
Scalability	High scalability issues as the number of users keeps growing which turns out to be expensive.	More efficient in terms of scalability as the similarity in items is
Cold Start	Insufficient data to provide recommendations for new users.	Less affected by this as item similarity is independent of user duration.
Interpretability	More interpretable as they depict the behaviour of similar users making it easier to understand the reason behind this.	Less interpretable due to their dependency on item-based interactions making it difficult to understand the reasons.
Inefficiency	Limited user interactions can be challenging to find ones similar to preferences	More robust as it focuses on item interactions & similarities can be calculated for less popular ones as well.

Implementation:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
```

```
movies = pd.read_csv('movie.csv')
ratings = pd.read_csv('ratings.csv')
```

```
ratings.head()
```

	userId	movieId	rating
0	1	2	3.5
1	1	29	3.5
2	1	32	3.5
3	1	47	3.5
4	1	50	3.5

```
final_dataset = ratings.pivot_table(index='movieId', columns='userId',
values='rating')
final_dataset.head()
```

userId	1	2	3	4	5	6	7	8	9	10	...	6410	6411	6412	6413	6414	6415	6416	6417	6418	6419
movieId																					
1	NaN	NaN	4.0	NaN	NaN	5.0	NaN	4.0	NaN	4.0	...	3.0	NaN	NaN	3.0	3.0	3.0	NaN	4.0	4.5	NaN
2	3.5	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	2.0	3.0
3	NaN	4.0	NaN	NaN	NaN	3.0	3.0	5.0	NaN	NaN	...	NaN	NaN	NaN	NaN	3.0	3.0	NaN	1.0	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.5	NaN

5 rows × 6419 columns

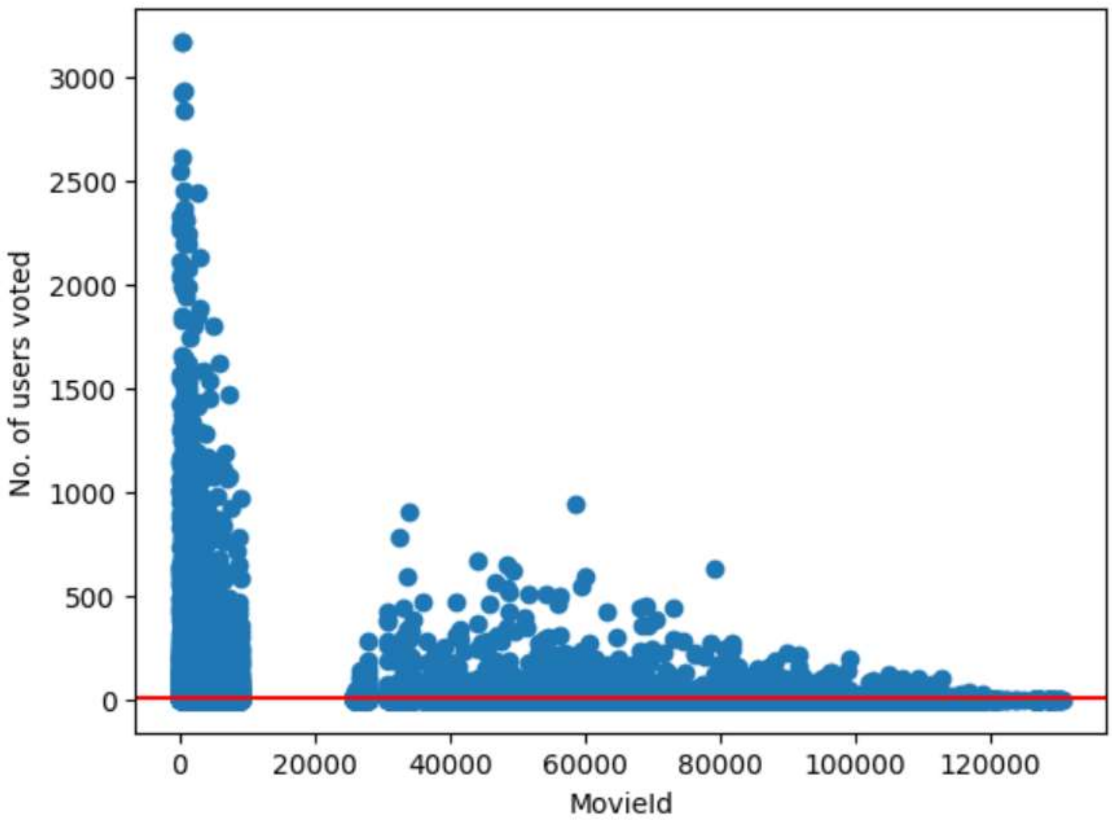
```
final_dataset.fillna(0,inplace=True)
final_dataset.head()
```

userId	1	2	3	4	5	6	7	8	9	10	...	6410	6411	6412	6413	6414	6415	6416	6417	6418	6419
movieId																					
1	0.0	0.0	4.0	0.0	0.0	5.0	0.0	4.0	0.0	4.0	...	3.0	0.0	0.0	3.0	3.0	3.0	0.0	4.0	4.5	0.0
2	3.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	2.0	3.0
3	0.0	4.0	0.0	0.0	0.0	3.0	3.0	5.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	3.0	0.0	1.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.5	0.0

5 rows × 6419 columns

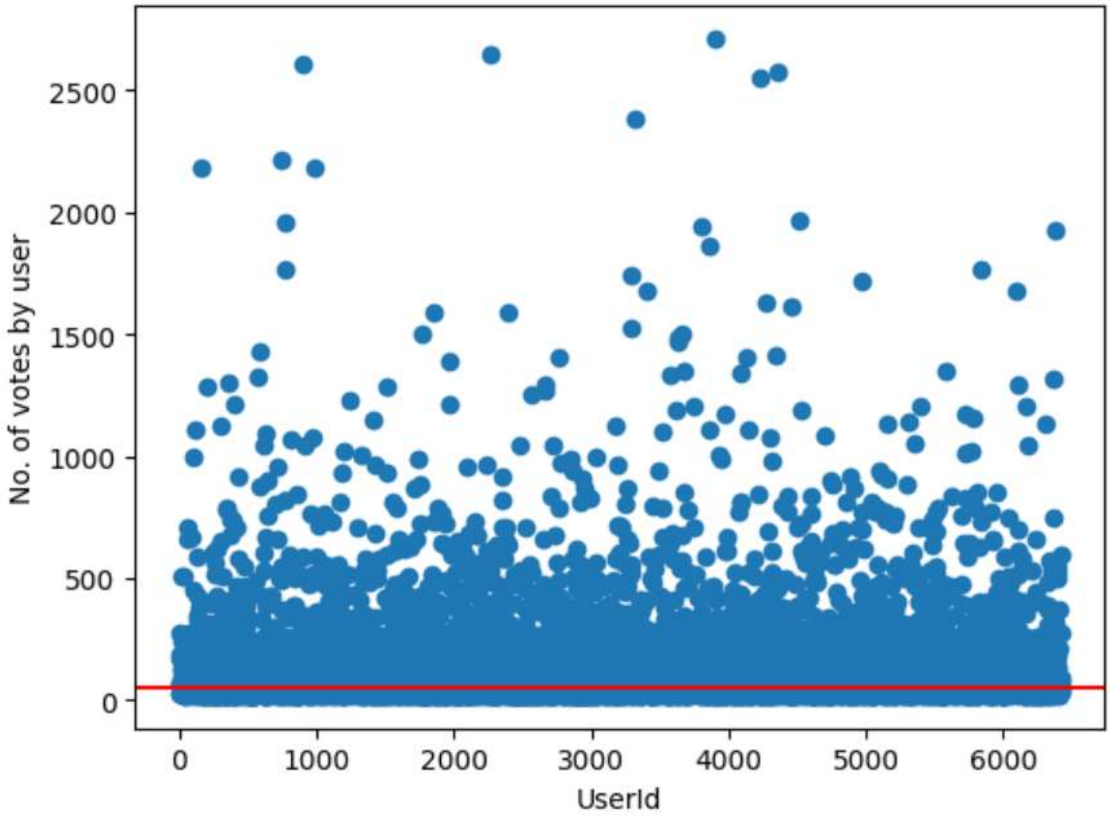
```
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

```
# ratings['rating'].plot(kind='hist')
plt.scatter(no_user_voted.index,no_user_voted)
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()
```




```
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

```
plt.scatter(no_movies_voted.index,no_movies_voted)
plt.axhline(y=50,color='r')
plt.xlabel('UserId')
plt.ylabel('No. of votes by user')
plt.show()
```




```
final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted >
50].index]
final_dataset
```

userId	1	2	3	5	7	8	11	13	14	16	...	6403	6405	6406	6409	6410	6412	6414	6417	6418	6419
movieId																					
1	0.0	0.0	4.0	0.0	0.0	4.0	4.5	4.0	4.5	3.0	...	0.0	4.0	0.0	0.0	3.0	0.0	3.0	4.0	4.5	0.0
2	3.5	0.0	0.0	3.0	0.0	0.0	0.0	3.0	0.0	0.0	...	0.0	3.5	0.0	0.0	0.0	0.0	0.0	4.0	2.0	3.0
3	0.0	4.0	0.0	0.0	3.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	3.0	1.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.5	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
115569	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
115617	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
116797	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
116823	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
118696	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

6420 rows × 3966 columns

```
sample = np.array([[0,0,3,0,0],[4,0,0,0,2],[0,0,0,0,1]])
sparsity = 1.0 - ( np.count_nonzero(sample) / float(sample.size) )
print(sparsity)
```

```
0.7333333333333334
```

```
csr_sample = csr_matrix(sample)
print(csr_sample)
```

```
  (0, 2)	3
  (1, 0)	4
  (1, 4)	2
  (2, 4)	1
```

```
csr_data = csr_matrix(final_dataset.values)
final_dataset.reset_index(inplace=True)
```

```
knn = NearestNeighbors(metric='cosine', algorithm='brute',
n_neighbors=20, n_jobs=-1)
knn.fit(csr_data)
```

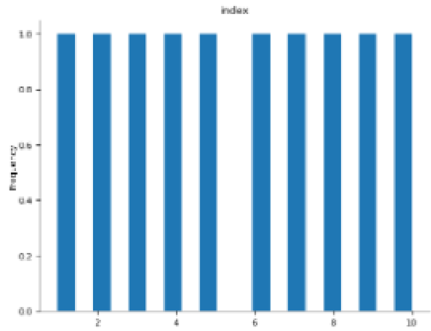
```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

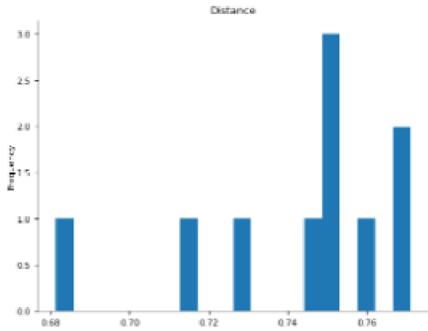
```python
def get_movie_recommendation(movie_name):
    n_movies_to_reccomend = 10
    movie_list = movies[movies['title'].str.contains(movie_name)]
    if len(movie_list):
        movie_idx= movie_list.iloc[0]['movieId']
        movie_idx = final_dataset[final_dataset['movieId'] ==
movie_idx].index[0]
        distances , indices =
knn.kneighbors(csr_data[movie_idx],n_neighbors=n_movies_to_reccomend+1)

        rec_movie_indices =
sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist()
)),key=lambda x: x[1])[:0:-1]
        recommend_frame = []
        for val in rec_movie_indices:
            movie_idx = final_dataset.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index
            recommend_frame.append({'Title':movies.iloc[idx]['title'].v
alues[0],'Distance':val[1]})
        df =
pd.DataFrame(recommend_frame,index=range(1,n_movies_to_reccomend+1))
        return df
    else:
        return "No movies found. Please check your input"
```

```python
get_movie_recommendation('Georgia')
```

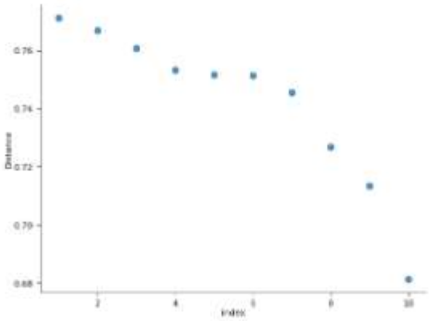
Index	Title	Distance
1	Safe Passage (1994)	0.7711027969128865
2	Angels and Insects (1995)	0.7669789298563784
3	Mighty Aphrodite (1995)	0.7606115389114131
4	City Hall (1996)	0.7531080792825611
5	What Happened Was... (1994)	0.7515918405438605
6	Antonia's Line (Antonia) (1995)	0.7514843387479659
7	Beautiful Girls (1996)	0.7453589518972888
8	Family Thing, A (1996)	0.7267833524428638
9	Crossing Guard, The (1995)	0.7135375730959093
10	Restoration (1995)	0.6813208553940657

Distributions

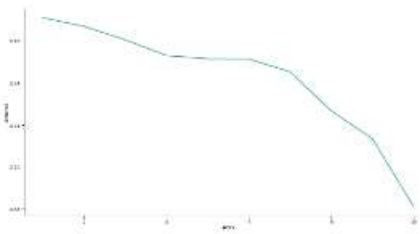


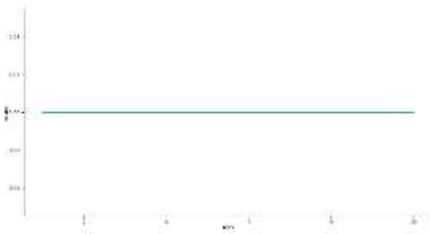


2-d distributions



Time series





Values