Movie Recommendation System
Item-Based Collaborative Filtering

Under the Guidance of
Professor Gary Davis

Bharath Anand
#ID: 02044023
Graduate Student
Department of Data Science
University of Massachusetts
Dartmouth

Balakrishna Vardhineni
#ID: 02069565
Graduate Student
Department of Data Science
University of Massachusetts
Dartmouth

2. Abstract

This project aims to develop a movie recommendation system using the Item-based Collaborative Filtering technique. The system predicts movie recommendations based on users' behavior and past searches. The MovieLens Latest Dataset, which contains 100,000 ratings and 9,000 movies by 600 users, is used to train and test the model. The outcome of this project is a functional movie recommendation system that can provide personalized recommendations to users based on their preferences and history.

3. Introduction and Background

The movie industry has seen exponential growth over the years, and the demand for movies keeps increasing with each passing day. However, with the vast number of movies available, it can be challenging for movie enthusiasts to find films that match their preferences. This is where movie recommendation systems come in. Movie recommendation systems are machine learning-based approaches that use a user's past behavior to filter and predict their possible movie choices. In this report, we will discuss the different filtration strategies for movie recommendation systems, namely content-based filtering and collaborative filtering.

Movie recommendation systems have gained immense popularity in recent times, and they are widely used by streaming platforms such as Netflix, Hulu, and Amazon Prime Video. These platforms use different algorithms and strategies to filter and predict movie preferences based on a user's past behavior. The two most popular filtration strategies used in movie recommendation systems are content-based filtering and collaborative filtering.

3.1 Content-Based Filtering:

Content-based filtering is a movie recommendation system that uses data provided about the items, i.e., movies. This data plays a crucial role and is extracted from only one user. The ML algorithm used for this strategy recommends movies that are similar to the user's preferences in the past. The similarity in content-based filtering is generated by the data about past film selections and likes by only one user.

How it Works:

The recommendation system analyzes the past preferences of the user concerned, and then it uses this information to try to find similar movies. This information is available in the database, such as lead actors, director, genre, etc. After that, the system provides movie recommendations for the user. The core element in content-based filtering is only the data of only one user that is used to make predictions.

3.2 Collaborative Filtering:

Collaborative filtering is a filtering strategy based on the combination of the relevant user's and other users' behaviors. The system compares and contrasts these behaviors for the most optimal results. It's a collaboration of multiple users' film preferences and behaviors.

How it Works

The core element in this movie recommendation system and the ML algorithm it's built on is the history of all users in the database. Basically, collaborative filtering is based on the interaction of all users in the system with the items, i.e., movies. Thus, every user impacts the final outcome of this ML-based recommendation system, while content-based filtering depends strictly on the data from one user for its modeling.

Collaborative Filtering Algorithms:

Collaborative filtering algorithms are divided into two categories:

User-based collaborative filtering: The idea is to look for similar patterns in movie preferences in the target user and other users in the database.

Item-based collaborative filtering: The basic concept here is to look for similar items (movies) that target users rate or interact with.

In conclusion, movie recommendation systems are crucial for movie enthusiasts who want to find movies that match their preferences. The two most popular filtration strategies used in these systems are content-based

filtering and collaborative filtering. While content-based filtering uses only data from one user, collaborative filtering relies on the interaction of all users in the system with the items. As the movie industry keeps growing, movie recommendation systems will play an increasingly vital role in helping movie enthusiasts find the movies they love.

4. Methods:

For this report, we've used the MovieLens Latest Datasets to perform cluster analysis on customer segmentation and the dataset file can be found and downloaded from here.The MovieLens Latest Dataset used in this project is a widely used dataset for developing and testing movie recommendation systems. It contains 100,000 ratings of 9,000 movies given by 600 users. This dataset is relatively small compared to other movie recommendation datasets, but it is still sufficient to build a basic movie recommendation system.

The code implements an Item-based Collaborative Filtering approach to make movie recommendations. The strategy is based on the similarity of items, or in this case, movies. The approach uses the cosine similarity metric and the k-nearest neighbors algorithm to find similar movies to a given movie and recommend them.

4.1 Cosine similarity:

It is a commonly used distance metric in recommendation systems, including in our item-based filtering implementation. The cosine similarity metric measures the similarity between two vectors in a multi-dimensional space, based on the cosine of the angle between them. In the context of recommendation systems, the cosine similarity measures the similarity between items based on the ratings given to them by users.

In our item-based filtering approach, we create a user-item matrix where the rows correspond to movies and the columns correspond to users, with each cell representing the rating given by a user to a movie. We then use this matrix to compute the cosine similarity between each pair of movies. The cosine similarity values range from -1 to 1, with 1 indicating that the two movies are identical in terms of user ratings, and -1 indicating that they are completely dissimilar.

The cosine similarity metric is preferred over other similarity metrics like Euclidean distance or Pearson correlation coefficient for several reasons. First, the cosine similarity metric is not affected by differences in scale or magnitude between two vectors, making it more suitable for sparse data where many ratings are missing. Second, the cosine similarity metric is faster to compute and requires less computational resources than other metrics. Third, the cosine similarity metric has been shown to perform well in many real-world recommendation systems, including the popular Netflix Prize competition.

4.2 KNN Algorithm:

The k-nearest neighbors (KNN) algorithm is a machine learning algorithm used for classification and regression problems. The basic idea behind the KNN algorithm is to find the k-nearest data points in the training set to a new data point and use their labels (in the case of classification) or values (in the case of regression) to make a prediction for the new data point.

In our project, we used the KNN algorithm to build a recommendation system for movies. Specifically, we used the KNN algorithm to find the k-most similar movies to a given movie based on their features such as genre, director, actors, and ratings.

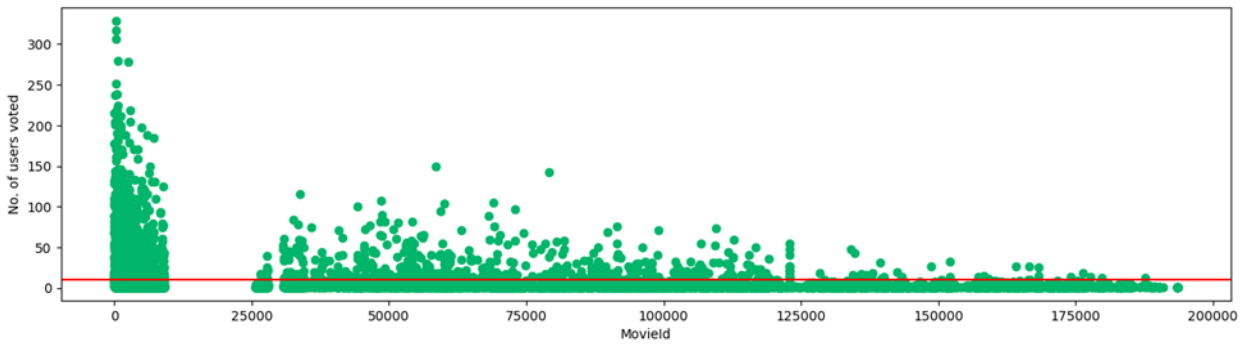
The steps we followed to implement the KNN algorithm in our project were as follows:

- Preprocessing the data: We cleaned and processed the movie data to extract features such as genre, director, actors, and ratings.
- Calculating the similarity: We used the cosine similarity metric to calculate the similarity between movies based on their features. This similarity metric takes into account the direction and magnitude of the vectors representing the movies' features and returns a value between 0 and 1, where 1 represents perfect similarity and 0 represents no similarity.
- Finding the k-nearest neighbors: We used the similarity scores to find the k-most similar movies to a given movie.
- Making recommendations: Once we had found the k-nearest neighbors to a given movie, we used their ratings to make a recommendation for the user.

Overall, the KNN algorithm helped us build a recommendation system that was able to provide personalized movie recommendations to users based on their preferences and the similarity between movies.

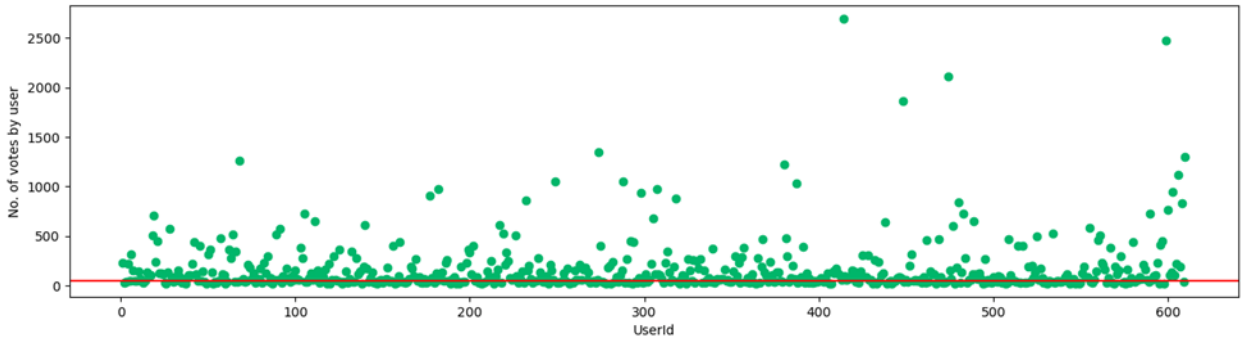
5. Results:

This graph represents, how many users have voted for each and every movie.



This helps us understand how all the movies here are voted and the red line represents the minimum number of votes for each movie.

The graph below helps us understand how many votes each user has voted.



This helps us understand that almost all the users have voted and it can help us recommend the movie much better.

After performing the item-based filtering through a function made for the recommendation. We can use the function to get recommendations as follows:

Here, all the above movies listed are recommendations for the movie 'Iron Man'.

```
In [17]: get_movie_recommendation('Iron Man')
```

Out[17]:

	Title	Distance
1	Up (2009)	0.368857
2	Guardians of the Galaxy (2014)	0.368758
3	Watchmen (2009)	0.368558
4	Star Trek (2009)	0.366029
5	Batman Begins (2005)	0.362759
6	Avatar (2009)	0.310893
7	Iron Man 2 (2010)	0.307492
8	WALL·E (2008)	0.298138
9	Dark Knight, The (2008)	0.285835
10	Avengers, The (2012)	0.285319

Let’s try another movie called ‘Memento’.

```
In [18]: get_movie_recommendation('Memento')
```

Out[18]:

	Title	Distance
1	American Beauty (1999)	0.389346
2	American History X (1998)	0.388615
3	Pulp Fiction (1994)	0.386235
4	Lord of the Rings: The Return of the King, The...	0.371622
5	Kill Bill: Vol. 1 (2003)	0.350167
6	Lord of the Rings: The Two Towers, The (2002)	0.348358
7	Eternal Sunshine of the Spotless Mind (2004)	0.346196
8	Matrix, The (1999)	0.326215
9	Lord of the Rings: The Fellowship of the Ring,...	0.316777
10	Fight Club (1999)	0.272380

The above-listed movies are recommended for the users who watch ‘Memento’.

6. Discussions & Conclusion:

The item-based filtering approach is a popular technique used for movie recommendation systems. This approach utilizes the movie's rating data and user information to recommend similar movies to the user's preferred movies. The implementation of this approach involves several steps, including filtering movies and users based on minimum votes, reducing sparsity using the csr_matrix function, and applying the cosine distance metric to identify similar movies. The NearestNeighbors algorithm is then used to fit and predict the recommended movies.

Using this approach, we can recommend movies to users in a fast and efficient manner. By filtering out movies and users with minimum votes, we

can eliminate the noise and focus on the most popular and relevant movies. The csr_matrix function can reduce the sparsity of the dataset, which is a common problem in recommendation systems. By using the cosine distance metric, we can identify similar movies and recommend them to the user.

Overall, the item-based filtering approach is an effective way of recommending movies to users. It allows us to utilize the rating data and user information to recommend movies that are similar to the user's preferred movies. By using this approach, we can provide a personalized movie recommendation system that meets the user's preferences and needs.

7. References:

1. Comprehensive Guide on Item-Based Collaborative Filtering

Url: https://towardsdatascience.com/comprehensive-guide-on-item-based-recommendation-systems-d67e40e2b75d

2. Item-to-Item Based Collaborative Filtering

Url: https://www.geeksforgeeks.org/item-to-item-based-collaborative-filtering/

3. Movielwns Dataset

Url: https://grouplens.org/datasets/movielens/latest/

4. A Brief Guide to Movie Recommendation Systems Using Machine Learning

Url: https://labelyourdata.com/articles/movie-recommendation-with-machine-learning#_content-based_filtering

Other References:

- Python 3: https://www.python.org/.
- Pandas: https://pandas.pydata.org/.
- NumPy: https://numpy.org/.
- Seaborn: https://seaborn.pydata.org/.
- Matplotlib: https://matplotlib.org/.

- K-Means Clustering: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

Appendix

The code can be found in the below attached appendix:

Item based filtering

This strategy is generally used because the movie doesn't change all that much. In contrast to User-based, where we must run the model often, we can restart this model once a week.

In this kernel, We look at the implementation of Item based filtering

In [1]:
```
import pandas as pd
import numpy as np

from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors

import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
ratings = pd.read_csv('/ASm/ratings.csv')
movies = pd.read_csv('/ASm/movies.csv')
```

In [2]:
```
ratings.head()
```

Out[2]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Ratings dataset has

- userId - unique for each user
- movieId - using this feature ,we take the title of the movie from movies dataset
- rating - Ratings given by each user to all the movies using this we are going to predict the top 10 similar movies
- timestamp - At what time the rating was recorded.

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

Out[3]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Movie dataset has

- movieId - once the recommendation is done, we get list of all similar movieId and get the title for each movie from this dataset.
- title - The movie title
- genres - which is not required for this filtering approach

In [4]:
```
final_dataset = ratings.pivot(index='movieId',columns='userId',values='rating')
final_dataset.head()
```

Out[4]:

userId	1	2	3	4	5	6	7	8	9	10	…	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	NaN	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	…	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	…	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	…	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 610 columns

In [5]:
```
final_dataset.fillna(0,inplace=True)
final_dataset.head()
```

Out[5]:

userId	1	2	3	4	5	6	7	8	9	10	…	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	…	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	…	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	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	2.0	0.0	0.0
4	0.0	0.0	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	0.0	0.0	0.0
5	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	3.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 610 columns

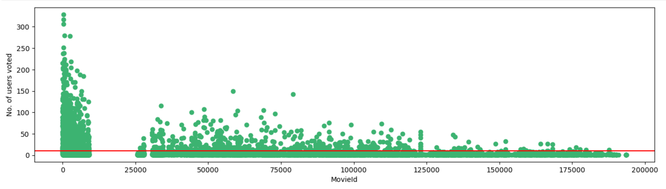
In the actual world, ratings are scarce, and the majority of data points come from highly regarded films and devoted viewers. In order to qualify the movies for the final dataset, we will add certain filters to reduce the noise.

- A movie needs at least 10 users to vote for it to qualify.
- A user must have voted on a minimum of 50 movies to be eligible.

In [6]:
```
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

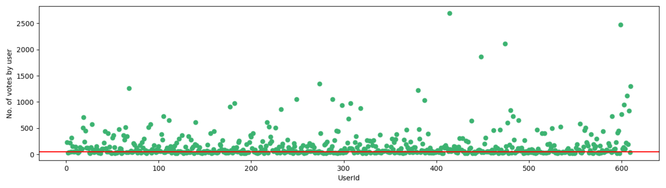
In [7]:
```
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')

f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_user_voted.index,no_user_voted,color='mediumseagreen')
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()
```



In [8]:
```
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

In [9]:
```
f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_movies_voted.index,no_movies_voted,color='mediumseagreen')
plt.axhline(y=50,color='r')
plt.xlabel('UserId')
plt.ylabel('No. of votes by user')
plt.show()
```



In [10]:
```
final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted > 50].index]
final_dataset
```

Out[10]:

userId	1	4	6	7	10	11	15	16	17	18	…	600	601	602	603	604	605	606	607	608	610
movieId																					
1	4.0	0.0	0.0	4.5	0.0	0.0	2.5	0.0	4.5	3.5	…	2.5	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	5.0
2	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	…	4.0	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0
3	4.0	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	2.0	0.0
5	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	2.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0	0.0	4.0	…	0.0	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	5.0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
174055	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
176371	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
177765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…	0.0	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
179819	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
187593	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

2121 rows × 378 columns

Our final_dataset has dimensions of 2121 * 378 where most of the values are sparse. I took only small dataset but for original large dataset of movie lens which has more than 100000 features, this will sure hang our system when this has feed to model. To reduce the sparsity we use csr_matrix scipy lib. I'll give an example how it works

In [11]:
```
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
```

In [12]:
```
csr_sample = csr_matrix(sample)
print(csr_sample)
```

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

- As you can see, the csr sample has values assigned as rows and column indexes rather than sparse values. Value is 3 for the 0th row and 2nd column. Examine the original dataset to see where the values should be. This is how it operates when returning to the original dataset using the todense technique.
- The majority of skleam uses sparse matrices. This will undoubtedly help us perform better.

In [13]:
```
csr_data = csr_matrix(final_dataset.values)
final_dataset.reset_index(inplace=True)
```

We utilize cosine distance metric which is highly fast and preferred than pearson coefficient. Please refrain from using euclidean distance, which is useless when the numbers are equally spaced.

In [14]:
```
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
```

In [15]:
```
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
knn.fit(csr_data)
```

Out[15]:

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

In [16]:
```
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'].values[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"
```

In [17]:
```
get_movie_recommendation('Iron Man')
```

Out[17]:

	Title	Distance
1	Up (2009)	0.368857
2	Guardians of the Galaxy (2014)	0.368758
3	Watchmen (2009)	0.368558
4	Star Trek (2009)	0.366029
5	Batman Begins (2005)	0.362759
6	Avatar (2009)	0.310893
7	Iron Man 2 (2010)	0.307492
8	WALL-E (2008)	0.298138
9	Dark Knight, The (2008)	0.285835
10	Avengers, The (2012)	0.285319

In [18]:
```
get_movie_recommendation('Memento')
```

Out[18]:

	Title	Distance
1	American Beauty (1999)	0.389346
2	American History X (1998)	0.388615
3	Pulp Fiction (1994)	0.386235
4	Lord of the Rings: The Return of the King, The...	0.371622
5	Kill Bill: Vol. 1 (2003)	0.350167
6	Lord of the Rings: The Two Towers, The (2002)	0.348358
7	Eternal Sunshine of the Spotless Mind (2004)	0.346196
8	Matrix, The (1999)	0.326215
9	Lord of the Rings: The Fellowship of the Ring,...	0.316777
10	Fight Club (1999)	0.272380

Our model works perfectly predicting the recommendation based on user behaviour and past search. So we conclude our collaborative filtering here.