

**Movie Recommendation Systems**

BANA 8083 MS Capstone

**Prepared by:**

Hasnat Shad Tahir

**First Reader:**

Dr. Peng Wang

**Second Reader:**

Dr. Dungang Liu

# Abstract

Today, while surfing/purchasing on the internet we are provided with a lot of choices to choose from, this can be time-consuming and frustrating sometimes. There is a need to filter, prioritize, and efficiently deliver relevant information to alleviate the problem of information overload. With the rapid growth of data collection, we can create/modify more efficient systems by effectively using the collected data. Recommendation Systems are information filtering systems that improve the quality of a search result by increasing its relevancy to the user search history or preferences. At present, almost every big company uses these systems: Amazon uses it to suggest products to customers based on their and other similar customers purchasing habits, Youtube uses it to decide what video to play next. Some music application companies like Spotify depend solely on the effectiveness of its recommender system for its success.

Contents

[Abstract 1](#_Toc46067149)

[Introduction 3](#_Toc46067150)

[The Data 4](#_Toc46067151)

[Movie Recommendations Systems 4](#_Toc46067152)

[1. Demographic Filtering: 4](#_Toc46067153)

[2. Content-Based Filtering 6](#_Toc46067154)

[3. Collaborative Filtering 8](#_Toc46067155)

[Singular Value Decomposition 9](#_Toc46067156)

[4. Hybrid Filtering 10](#_Toc46067157)

[Conclusion 11](#_Toc46067158)

[References 12](#_Toc46067159)

# Introduction

Recommender systems are one of the most successful and widespread application of machine learning technologies in ecommerce businesses. In the last few decades, with the rise of streaming platforms, online shopping, and many other such online services, recommender systems have taken an important place in our lives. We need recommender systems that help us in making good decisions by showing us what we might need or by providing us with a better option.

Recommender systems collect information from users’ preference for a set of items/choices and predict the best desired items/choice for them. Building a recommender system depends on a set of considerations, such as type of available data, filtering algorithms, models, techniques, sparsity level of data and desired quality. Demographic filtering, Content-based filtering, collaborative filtering, and hybrid methods are the main four methods of recommender systems.

1. Demographic Filtering: It aims to classify the user based on personal attributes and make recommendations based on demographic classes. It can be used to identify the taste of users that belong to a certain demographic group. If some users in a particular group like or order an item, it is possible that the other users of this group tend to do the same. The users provide the personal data via surveys or can be extracted from the purchasing/browsing history of the users.
2. Content-Based Filtering: These recommender systems suggest similar items/choices based on previous item/choice. The general idea behind these recommender systems is that if a person liked a particular item, he or she will also like an item that is similar to it. Content-based recommender systems are mainly used where either a document or a text is used to describe an item. Thus, Text Analytics play an important role in such recommender systems.
3. Collaborative Filtering: This filtering is probably the most widely implemented and most mature of the recommender systems. In collaborative filtering, the recommender system looks for similarity between users to make predictions. In several cases, the pattern of ratings of users is a useful feature to determine similarity. Singular Value Decomposition will be used to capture the similarity between users and items in the ratings-matrix and further modify collaborative filtering.
4. Hybrid Filtering: A hybrid filtering method uses a combination of collaborative filtering with demographic filtering or collaborative filtering with content-based filtering to obtain better results. Predominantly, collaborative filtering has been combined with content-based filtering to make a hybrid method.

In this project, different Movie Recommendation Systems will be created leveraging Text Analytics and Python. Further, the application and limitations of these systems will be analyzed.

# The Data

TMDb dataset: This dataset is an ensemble of data collected from TMDB and GroupLens. The Movie Details, Credits and Keywords have been collected from the TMDB Open API. Files contain metadata for 45,000 movies released on or before July 2017 listed in the Full MovieLens Dataset. Data points include cast, crew, plot keywords, budget, revenue, release dates, languages, production companies, countries, vote counts, vote averages, and more.

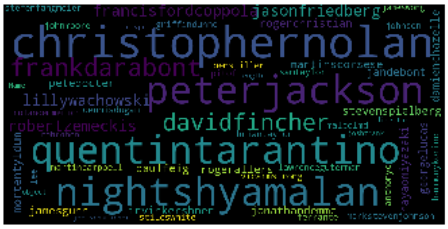
 

Figure 1: Word Clouds of Cast and Director columns

# Movie Recommendations Systems

Building and studying different recommender system:

## Demographic Filtering:

For Demographic filtering, we will score the movies based on the average rating of the particular movie. If we use the existing average rating, it will not be fair since a movie with 9 average rating and only 5 votes cannot be considered better than the movie with 8 average rating and 50 votes. So, we will be using IMDB’s weighted rating parameter which will take care of the aforementioned issue.

***Weighted Rating (WR)*** =

where:

R = average for the movie (mean)

v = number of votes for the movie

m = minimum votes (*in this case, movies having more than votes than 85% movies in the list*)

C = the mean vote across the whole report

We have 721 movies that have votes more that 85% of the movies in the list. After calculating the Weighted Rating for all these movies, following are the top 10 highest rated movies:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Movie Title** | **Average Rating** | **Vote Count** | **Weighted Rating (WR)** |
| 278 | The Shawshank Redemption | 8.5 | 8205 | 8.17 |
| 550 | Fight Club | 8.3 | 9413 | 8.03 |
| 680 | Pulp Fiction | 8.3 | 8428 | 8.00 |
| 155 | The Dark Knight | 8.2 | 12002 | 7.99 |
| 238 | The Godfather | 8.4 | 5893 | 7.98 |
| 27205 | Inception | 8.1 | 13752 | 7.93 |
| 13 | Forrest Gump | 8.2 | 7927 | 7.90 |
| 157336 | Interstellar | 8.1 | 10867 | 7.89 |
| 122 | The Lord of the Rings: The Return of the King | 8.1 | 8064 | 7.82 |
| 1891 | The Empire Strikes Back | 8.2 | 5879 | 7.82 |

Table 1: Top 10 movies based on Weighted Rating (WR)

It can be observed from the above table, even though average rating of ‘The Godfather’ is more than ‘The Dark Knight’ it is ranked below ‘The Dark Knight’ based of the Weighted Rating parameter which takes in vote count into account.

We often observe movie recommendation that are very popular at that time. To get such filtering we can rank movies based on the popularity. Following plot shows the top 10 most popular movies:

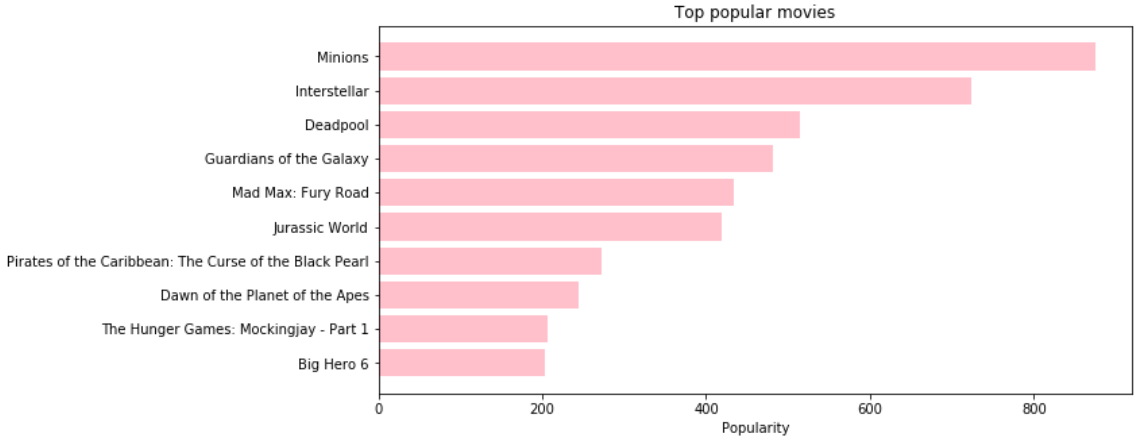


Figure 2: Top 10 movies based on popularity

Something to keep in mind is that demographic filtering provides a general chart of recommended movies to all the users and are not adaptable to the interests and preferences of a particular user.

**Advantages** of Demographic Filtering

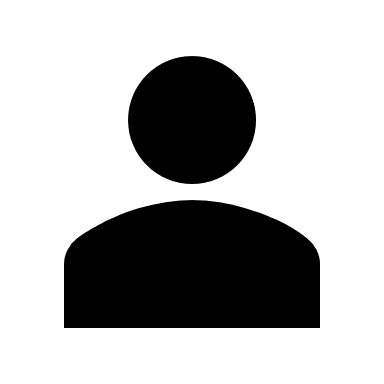
* Can identify cross-genre niches
* Domain knowledge not needed
* Adaptive: quality improves over time

**Limitations** of Demographic Filtering

* Not adaptable to the interests and preferences of a particular user
* Quality dependent on large historical dataset

## Content-Based Filtering

In this kind of recommender systems, the details of the movie such as overview, cast, crew, keyword, tagline, and more are used to find its similarity with other movies. The movies that are most likely to be similar are recommended.



Movie A (Horror)

Movie B (Thriller)

Movie C (Horror)

Similar

Recommended

Likes

Figure 3: Content-Based Filtering

Movie Overview based recommendation system:

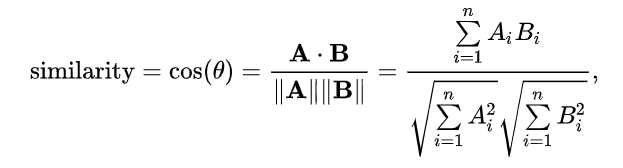
First, we will only use the plot overview of a movie to build a content-based recommendation system.

|  |  |
| --- | --- |
| **Movie** | **Overview** |
| The Shawshank Redemption | Framed in the 1940s for the double murder of his wife and her lover, upstanding banker Andy Dufresne begins a new life at the Shawshank prison, where he puts his accounting skills to work for an amoral warden. During his long stretch in prison, Dufresne comes to be admired by the other inmates -- including an older prisoner named Red -- for his integrity and unquenchable sense of hope. |
| Fight Club | A ticking-time-bomb insomniac and a slippery soap salesman channel primal male aggression into a shocking new form of therapy. Their concept catches on, with underground "fight clubs" forming in every town, until an eccentric gets in the way and ignites an out-of-control spiral toward oblivion. |

Table 2: Sample plot overview

We will compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview. Term Frequency is the relative frequency of a word in a document and is given as (term instances/total instances). Inverse Document Frequency is the relative count of documents containing the term is given as log(number of documents/documents with term) The overall importance of each word to the documents in which they appear is equal to TF \* IDF. This will generate a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one cell) and each row represents a movie. This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score. We will use sklearn’s TfIdfVectorizer to build the TF-IDF matrix.

The generated TF-IDF matrix contains over 20,000 words to describe all the movies in the dataset. Now, we will be using the cosine similarity to calculate a numeric quantity that gives the similarity between two movies. The cosine similarity score is independent of magnitude and is relatively easy and fast to calculate. Mathematically:



We have used sklearn's linear\_kernel() to calculate the cosine similarity score. Also, we have defined a function that takes in movie name as input and gives out a list of top 5 most similar movies based on cosine similarity score.

Function outputs:

|  |  |
| --- | --- |
| **Movie** | **Top 5 Similar Movies** |
| The Dark Knight Rises | The Dark Knight Rises |
| Batman Returns |
| Batman: The Dark Knight Returns, Part 2 |
| Batman Forever |
| Batman |
| Superman | Superman II |
| Superman Returns |
| Superman IV: The Quest for Peace |
| Central Intelligence |
| Horse Camp |

Table 3: Movie recommendations based on Cosine similarity score

The function is doing a great job recommending movies based on plot overview. We observe that recommendation for Batman movies contain all the Batman movies and same in the case of Superman. If a person liked “The Dark Knight Rises” or “Superman” just because of the director and not the overview, in such a case this recommender is not generating relevant recommendations. So, the quality of recommendations can be improved by increasing the scope of data, we can do it by adding other details such as actors, directors, genre, etc.

After extracting top 3 actors, director, keywords, and genres from the data, we will again find the cosine similarity scores. But this time we will use sklearn’s CountVectorizer() instead of generating a TF-IDF matrix, as it will not make sense if we down-weight the presence of an actor/director if he or she has acted or directed in relatively more movies. We defined the same function with a new cosine similarity score.

|  |  |
| --- | --- |
| **Movie** | **Top 5 Similar Movies** |
| The Dark Knight Rises | The Dark Knight |
| Batman Begins |
| Amidst the Devil's Wings |
| The Prestige |
| Romeo Is Bleeding |

Table 4: Movie recommendations based on modified cosine similarity score

We can say that our recommender has been successful in capturing more information due to more metadata and has given us better recommendations.

**Results:**

To evaluate the performance of the Content-based recommender system, correct/incorrect recommendation were identified based on the deviation in the ratings provided by a user(deviation of more than 1 makes the recommendation incorrect for the user).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User ID** | **Movie Rated by the user (M1)** | **Ratings given to M1** | **Recommended movies (M2)** | **Ratings given to M2** | **Deviation** | **Correct?**  **Yes = 1**  **No = 0** |
| 1 | Cape Fear | 2 | Sleepers | 3 | -1 | 1 |
| 1 | Sleepers | 3 | Cape Fear | 2 | 1 | 1 |
| 2 | Apollo 13 | 5 | Philadelphia | 4 | 1 | 1 |
| 2 | Batman Forever | 4 | Batman | 5 | -1 | 1 |
| 2 | Wolf | 3 | Batman | 5 | -2 | 0 |
| 3 | Speed | 2.5 | Young Guns | 4 | -1.5 | 0 |
| 4 | Batman Forever | 4 | Superman | 5 | -1 | 1 |
| 4 | Golden Eye | 4 | First Blood | 4 | 0 | 1 |
| 4 | Pulp Fiction | 5 | Reservoir Dogs | 5 | 0 | 1 |
| 4 | Star Trek: The Motion Picture | 4 | Star Trek IV: The Voyage Home | 3 | 1 | 1 |

Table 5: Correct/incorrect recommendations by Content-based recommendation system

The above table is a chunk of the main table and it shows some incorrect recommendations provided by the system.

**Accuracy of the Content-based recommendation system = ~*76 %***

**Advantages** of Content-based Filtering

* Able to recommend new as well as unpopular movies
* Quality of recommendations tend to improve over time
* Able to cater to users with unique tastes
* Provides explanation by mentioning the content features

**Limitations** of Content-based Filtering

* It relies only on item features only, and not the user preferences
* Determining what characteristics of the item the user dislikes or likes is not obvious
* Anyone querying for recommendations based on a movie will receive the same recommendations, regardless of who she/he is.

## Collaborative Filtering

Collaborative Filtering relies on how other users responded to these same items. It does not rely of features of the item, but the preferences from other users. There are two categories of Collaborative Filtering:

1. User-based: measure the similarity between target users and other users
2. Item-based: measure the similarity between the items that target users rate/ interact with and other items

**Advantages** of Collaborative Filtering

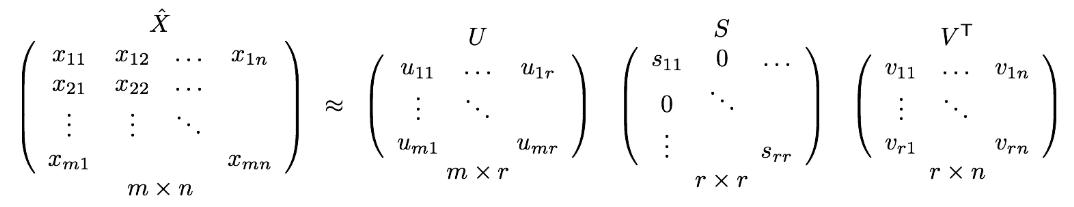
* No domain knowledge or feature selection is required since interactions yield recommendations
* Implicit user feedback is sufficient
* Can recommend different movies and develop new interests among users

**Limitations** of Collaborative Filtering

* Scalability: computation grows with both the customer and the product
* Sparsity: similarity between two very different movies could be very high simply because they have similar rank for the only user who ranked them both

### Singular Value Decomposition

One way to handle the scalability and sparsity issue created by Collaborative Filtering is to leverage a latent factor model to capture the similarity between users and movies. We will use Root Mean Square Error (RSME) to evaluate the rating predictions for movies given a user. Lower the RMSE, better the performance. To achieve minimal RMSE, Singular Value Decomposition (SVD) is adopted as shown in the below formula.



X denotes the utility matrix, and U is a left singular matrix, representing the relationship between users and latent factors. S is a diagonal matrix describing the strength of each latent factor, while V transpose is a right singular matrix, indicating the similarity between items and latent factors. Latent factor is a property that a user or an item have. For instance, for movies, latent factor can refer to the genre that the music belongs to. SVD decreases the dimension of the utility matrix by extracting its latent factors. Essentially, we map each user and each item into a latent space with dimension r. Therefore, it helps us better understand the relationship between users and items as they become directly comparable. We will use the Surprise library to implement SVD. We get a mean RSME of 0.8966.

To evaluate our trained model for prediction, we check the ratings user with the userID 30 has given to different movies and the predictions made by model.

|  |  |  |  |
| --- | --- | --- | --- |
| **User ID** | **Movie ID** | **Rating** | **Predicted Rating** |
| 30 | 1 | 4 | 3.85 |
| 30 | 2 | 2 | 3.51 |
| 30 | 6 | 4 | 4.00 |
| 30 | 8 | 4 | 3.68 |
| 30 | 11 | 4 | 3.88 |

Table 6: Actual vs Predicted Ratings for UserID 30

It can be observed from the above table that the ratings predicted by the model are very close to the actual ratings.

**Advantages** of Singular Value Decomposition

* Captures essence of all the input features
* Optimal low-rank approximation
* No cold start problems
* Able to recommend new items to users

**Limitations** of Singular Value Decomposition

* It does not care about the movie type. It works purely based on an assigned movie ID and predicts ratings based on how the other users have rated the movie.
* Computationally expensive

## Hybrid Filtering

To address the limitation of the SVD based recommender system, we will build a hybrid recommender that brings together techniques we have implemented in the content-based and collaborative filtering. We will define a function that takes in ‘UserId’ and ‘Movie’ as input to generate a list top 10 recommendations. Output of the Hybrid recommendation system:

|  |  |  |
| --- | --- | --- |
| **User ID** | **Movie** | **Recommendations** |
| 1 | Kung Fu Hustle | Delicatessen |
| Sophie's Choice |
| Z |
| Gallipoli |
| Betty Fisher and Other Stories |
| Longtime Companion |
| Standing in the Shadows of Motown |
| Ghost Dog: The Way of the Samurai |
| American Heart |
| Trick |

Table 7: Hybrid movie recommendations for UserID 1 and Kung Fu Hustle

|  |  |  |
| --- | --- | --- |
| **User ID** | **Movie** | **Recommendations** |
| 100 | Kung Fu Hustle | Delicatessen |
| Trick |
| Gallipoli |
| Sophie's Choice |
| Ghost Dog: The Way of the Samurai |
| Z |
| American Heart |
| Stranger Than Paradise |
| Born Free |
| Standing in the Shadows of Motown |

Table 8: Hybrid movie recommendations for UserID 100 and Kung Fu Hustle

It can be observed from the above output tables that for the same movie we are getting different movie recommendations for different users. We have successfully built a Hybrid Recommendation System that provides recommendations based on both content and user preferences.

# Conclusion

Recommender systems can create great strategic advantages for companies that implement them, and customer satisfaction can be achieved along with customer loyalty. In this project, we have successfully built and studied the advantages & limitations of four different recommendation systems:

1. Demographic Filtering based
2. Content-based
3. Collaborative Filtering based
4. Hybrid Filtering based

Studying the results of all recommender systems, we observed that Demographic filtering is very elementary and cannot be used practically, Content-based filter is good at identifying very similar styles movies. Collaborative filtering is good at spanning gaps across genres and consistently recommending movies with high ratings. The Hybrid filtering takes advantage of Content-based and Collaborative filtering and gives the best quality recommendations. It beautifully combines the concepts of Content-based and Collaborative Filtering to build an engine that gives movie suggestions to a particular user based on the estimated ratings that it had internally calculated for that user.

**Code for the project: (**<https://github.com/hst22/Movie-Recommender-Systems/blob/master/Recommender%20System.ipynb>)

# References

* Seyednezhad, S., Cozart, K., Bowllan, J. and Smith, A. (2018). A Review on Recommendation Systems: Context-aware to Social-based. [online] Available at: https://arxiv.org/pdf/1811.11866.pdf#:~:text=On%20the%20other%20hand%2C%20in [Accessed 19 Jul. 2020].
* Luo, S. (2018). Intro to Recommender System: Collaborative Filtering. [online] Towards Data Science. Available at: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26.
* Scikit-learn.org. (2018). *sklearn.feature\_extraction.text.TfidfVectorizer — scikit-learn 0.20.3 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.TfidfVectorizer.html.
* scikit-learn.org. (n.d.). sklearn.metrics.pairwise.linear\_kernel — scikit-learn 0.23.1 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.linear\_kernel.html [Accessed 19 Jul. 2020].
* ‌ Scikit-learn.org. (2018). sklearn.feature\_extraction.text.CountVectorizer — scikit-learn 0.20.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html.
* ‌ Kite.com. (2019). Code Faster with Line-of-Code Completions, Cloudless Processing. [online] Available at: https://kite.com/python/docs/ast.literal\_eval.
* ‌Hug, N. (2017). Home. [online] Surprise. Available at: <http://surpriselib.com/>.
* Wikipedia.org. (2014). Wikipedia. [online] Available at: <http://wikipedia.org>.
* Google. (2019). Google Homepage. [online] Available at: http://google.com.