# Movie Recommendation Systems

***Exploring Methods for Generalized and Personalized Movie Recommendations***

**Xiaoxuan Lu, Xiaoyu Wu, Yiming Li, Violette Pretorius**

## 1 Abstract

Recommendation systems, which are used by many websites and in various e-commerce applications, utilize information-filtering to predict the relationships between users and items and thus assist users in classifying their interests. This paper focuses on movie recommender systems. We built a generalized movie recommender system using sorting and filtering, and a content-based movie recommender system using similarity measures. We evaluated our recommendations with the recommendations from the IMDb website and Amazon Prime, and with the user's watch history from the dataset. Lastly, we deployed our recommender systems online.

## 2 Introduction

Whether it's in streaming videos, online shopping, or social networking, we are always confronted with an overwhelming number of options. In order to assist internet users, recommender systems have been developed to provide more personalized content and services. Commercial websites with recommender systems allow their customers to benefit from lower time and transaction costs while searching for products, often resulting in an improved quality in their choices. Recommender systems also benefit content providers, since platforms that provide recommendations to users earn increased revenue as a result (Isinkaye et al., 2015). Intrigued by this, we decided to investigate how such recommender systems are built, with a specific focus on movie-related applications.

The most straightforward method of recommendation is to recommend the most popular or highly-ranked items. Thus we decided to build a recommender that expands on this simple filter and allows users to select movies based on a wider range of attributes, such as genre, language, and release year. We also experimented with implementing several content-based recommender systems based on different approaches.

## 3 Related Work

Recommender systems are an integral part of many companies and services provided today, providing a key point of revenue for companies such as Netflix and Amazon (Meel et al., 2020). In recent years, various machine learning algorithms for building recommendation systems have been proposed. These are generally split into collaborative filtering, content-based filtering and hybrid filtering (Merve Acilar and Arslan, 2009; Jalali et al., 2010). In this project, we are primarily interested in generalized and content-based recommendation systems.

Content-based recommendation systems strive to suggest products that are comparable to those that a user has previously enjoyed. A content-based recommender's core operation entails comparing the properties of a user profile, which stores their preferences and interests, with the attributes of a content object in order to suggest new items that would be of interest to the user (Lops et al., 2010).

## 4 Data Collection and Description

To develop our recommendation systems, we used The Movies Dataset from Kaggle. This dataset consists of information on features of 45,000 movies that were released before July 2017. These features include the cast, crew, plot overview, keywords, release dates, and languages of the movies. Moreover, it includes 26 million ratings from 270,000 users for all 45,000 movies obtained from TMDb. The rating data is available as the individual entries of users or as vote count and vote average for each movie.

## 5 Methods

We built two kinds of recommender systems in this project. A simple recommender system, and a content-based recommender system. The simple recommendation system offers generalized recommendations based on one of two criteria: popularity or weighted rating. It can also filter the recommendations based on a user's choice of movie genre, language, release year, runtime, and director. The implementation of the system was rather simple. We sorted the movies based on either the vote-count data from the dataset containing movie popularity measures or a weighted rating score obtained from the IMDb weighted rating formula. The formula calculates a weighted rating $WR$ as follows,

$$WR = \frac{v}{c+m} * R + \frac{m}{v+m} * C$$

Where $v$ is the vote count of the movie, $m$ is the minimal vote count required to be listed in the sorting list (in our case it is set to the 5% quantile on the distribution of vote count), $R$ is the vote average of the movie, $C$ is the mean of all vote averages for all movies. After sorting, we filtered through the sorted list according to the user's choices among the available filters and obtained the final recommendation list of movies.

The content-based movie recommender recommends to users the movies which are similar to their preferences. In this part of our research, we defined "similar movies" in two different ways. The first one is to recommend new movies based on similarities between movie taglines and overviews, and the second is based on similarities between the cast, crew, genre, and keywords of different movies. To implement these recommenders, we first tokenized the texts in the dataset and preprocessed them by filtering out the punctuation. We then combined several columns to make new features in the dataset. For example, we combined the taglines and overviews as a new feature called description. We also combined the director, screenplay, five primary actors, genre, and keywords into a new feature. Then we compared several measures to calculate the similarity between texts—cosine similarity with TF-IDF and Countvectorizer, Kullback–Leibler Divergence.

The cosine similarity of two vectors from the same space determines the angle's cosine. The narrower the angle between the two vectors, the higher the cosine similarity. Scikit-Learn provides the TfidfVectorizer transformer to vectorize documents containing TF–IDF numerics. With the help of the built-in transformer, the text information was vectorized, and the cosine similarity calculated. The general formula of cosine similarity is as follows,

$$cos\theta = \frac{A * B}{\mid A \mid * \mid B \mid}$$

The distances between probability distributions are measured by KL Divergence. KL Divergence quantifies the extra bits required to encode a certain data collection P when a model based on a data set $Q$ is used. The general formula for Kullback–Leibler (KL) Divergence is as follows,

$$KL(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

We compared cosine similarity with TF-IDF and KL Divergence on content-based recommender systems. We found that the cosine similarity method works much faster than the KL Divergence method, and they generate similar results. Therefore, we chose cosine similarity for the content-based system. Additionally, we also compared the system based on taglines and overviews with the system based on keywords, genres, cast, and crew. The latter worked better so in the end we determined that 'similarity' is most appropriately based on features such as cast, crew, keywords and genres in our content-based system.

To extend the idea of the first two recommenders, we created a final content-based recommender that made recommendations based on user watch-history using a Doc2Vec model. Doc2Vec is an extension of Word2Vec that learns semantic associations from 'documents' represented by sentences in the form of word token lists. Using these learned associations, it creates embeddings representative of each document. For our Doc2Vec model we used the default DBOW (Distributed Bag-of-Words) model which is analogous to the Skip-Gram model used by Word2Vec and uses target words to predict the context of surrounding words. However, the paragraph ID or tag is used as input instead of a single target word so words can be sampled from the associated paragraph in order to predict the overall context. DBOW ignores word order and is generally simpler than its contemporary DM (Distributed Memory), often with better performance (Lau and Baldwin, 2016).

Firstly, watch-history per user was compiled using the "ratings" dataset since it provided a list of user IDs and movie IDs that corresponded to each rating. Although there is still much debate on corpus size versus specificity when determining semantic similarity, a study by Altszyler et al. (2017) found that Word2Vec (upon which Doc2Vec is based) performed better when it could take advantage of the whole corpus instead of on one that was filtered according to the domain specificity of the task. Therefore, despite the general findings of the previous content-based recommenders which concluded that recommenders based on metadata cast, crew, keywords and genres were more useful for determining movie similarity based on word embeddings, it was deemed necessary to also include the taglines and plot overview in the metadata 'documents' used to train the Doc2Vec model. This would ensure a larger corpus of words for the model to work with. The metadata documents for each movie were tagged with the original movie titles.

Before the final recommender could be built, it was necessary to preprocess the metadata, and remove null values as well as to ensure that movies in the watch history (taken from the "ratings" dataset) overlapped with the movies dataset containing the metadata. However, this process resulted in many movies being lost from the original dataset. This made the previous decision to work with all the metadata available even more crucial. found that Word2Vec (upon which Doc2Vec is based) performed better when it could take advantage of the whole corpus instead of on one that was filtered according to the domain specificity of the task. Therefore, despite the general findings of the previous content-based recommenders which concluded that recommenders based on metadata cast, crew, keywords and genres were more useful for determining movie similarity based on word embeddings, it was deemed necessary to also include the taglines and plot overview in the metadata 'documents' used to train the Doc2Vec model. This would ensure a larger corpus of words for the model to work with. The metadata documents for each movie were tagged with the original movie titles.

To implement the final recommender, we averaged over the document embeddings for the metadata of each movie in a user's watch history. We were then able to use the model to find the document embeddings semantically closest to the vector representing the user's watch history and output a list of movie recommendations. To evaluate the recommendations we planned to split off a portion of movies from each user's watch history to effectively form a test set to evaluate recommendations. Recommendations would be based on the remaining movies in the user's watch history and then compared with the movies in the test set to see the extent to which the same movies appeared in both sets. However, due to time constraints, we were only able to perform the manual evaluation of recommendations by comparing similarities between the recommendations and the watch history for a user based on movie features such as genre and language.

After creating different kinds of recommender systems, we decided to deploy some of them on the cloud. Initially, we wanted to write a website from scratch. However, due to lack of time, we chose to use the Streamlit package to build a simple website. We deployed our generalized system and also a content-based system based on keywords, genre, cast, and crew.

## 6 Results

The simple recommender generates movies based on a list of featured inputs. For example, Table 1 shows a list of five top-rated action movies in 2012 recommended by our simple recommender, and Table 2 demonstrates a list of top-rated Christopher Nolan's movies.

In general, our weighted rating for each movie is lower than it appears on the IMDb website, which is probably because the votes from IMDb users are much higher than the "vote-count" from our dataset. For example, there are 2.3M votes on "Inception" on IMDb, and our dataset only collected 14K votes. However, our general recommender also produced several matches with the list in IMDb, indicating a certain significance to our approach. For instance, we observed 6/10 matches when we asked our generalized recommender and the IMDb website to suggest ten highly-rated animations.

Futhermore, the results of using content-based recommendations suggest similar movies can be shown in Table 3 which shows similar movies to those recommended by Amazon Prime being suggested by our recommender.

By comparing our recommendations with those

| Movie Title | Year | Weighted Rating |
|---|---|---|
| The Dark Knight Rises | 2012 | 7.3 |
| The Avengers | 2012 | 7.2 |
| Life of Pi | 2012 | 6.8 |
| The Hobbit | 2012 | 6.7 |
| Skyfall | 2012 | 6.6 |

Table 1: Five top-rated action movies recommended by general recommender.

| Movie Title | Weighted Rating | IMDb Rating |
|---|---|---|
| The Dark Knight | 8.3 | 9.0 |
| Memento | 8.1 | 8.4 |
| Inception | 8.1 | 8.8 |
| Interstellar | 8.1 | 8.6 |
| The Prestige | 8.0 | 8.5 |

Table 2: Five top-rated movies directed by Christopher Nolan generated by simple recommender. The weighted rating for each film is derived using information from our dataset, and the IMDb rating is based on the number displayed on the IMDb website.

on Prime Video, we observed that the recommendations based on the director, cast, genre, keywords, etc., yielded more matches than the other one. On the other hand, the higher degree of relevance in Prime Video's recommendations may mean that their recommendations are based on other attributes that we did not consider or that it combines multiple algorithms to provide better recommendations to its users.

For the final watch-history-based recommender which used Doc2Vec embeddings of the movie metadata, we were able to obtain lists of recommendations using the user ID and the number of desired recommendations as input.Although time constraints meant that no rigorous evaluation could be implemented, we were often able to find consistencies between movies recommended for the user and features like language or genre. For example, when comparing recommendations and watch history for user 30007 both the watch history and recommendations featured Japanese language movies as well as movies with similar genres including: drama, action, and sci-fi.

| Movies based on OT | Movies based on DCGK | Movies from Amazon Prime |
|---|---|---|
| United States of Secrets (Part One) | The Lost World: Jurassic Park | The Lost World: Jurassic Park |
| The Evil Within | E.T. | Jurassic World |
| Forgotten | Catch Me If You Can | Jurassic Park III |
| Highway | Schindler's List | E.T. |
| Secrets and Lies | Always | Transformers: Age of Extinction |

Table 3: Some movie recommendations if we input a specific movie title, e.g., "Jurassic Park". (Left) A list of movies generated by utilizing OT (overviews and taglines). (Middle) A list of movies based on DCGK (director, casts, genre, keywords etc). (Right) A list of movies generated by Prime Video.

## 7 Conclusion

Our generalized recommender is novel in many aspects as it involves many additional movie features to better accommodate the users. It also resulted in satisfying recommendations that were comparable with the IMDb website. However, as further research, we could try to do web scraping and compare large lists of recommendations. For our content-based recommender, time constraints and coding issues made it hard to implement the intended evaluation method on the watch-history-based recommender. We could also try to improve the performance of Doc2Vec by using pre-trained embeddings, as well as complete the Word2Vec model and compare their performances. In the future, we also would like to store all of the models and preprocessed datasets in pickles so we can import them into our app to make it run faster. Moreover, we could also compare different similarity methods based not only on speed but also on accuracy and robustness, provided we use better methods of evaluation.

## 8 Author Contribution

Violette and Xiaoxuan worked on building content-based recommender with three different methods. Xiaoyu and Yiming worked on building generalized recommendation system. Xiaoyu evaluated them briefly by comparing our results with the results on IMDb and Amazon Prime. Violette evaluated the watch-history-based recommender by comparing recommendations to watch history. Xiaoxuan and Yiming made two websites for the generalized recommender and the content-based recommender. Finally, Xiaoxuan deployed the generalized recommender on the Heroku cloud.

## References


Altszyler, E., Sigman, M., & Slezak, D. F. (2017). Corpus specificity in LSA and word2vec: The role of out-of-domain documents. *CoRR*, *abs/1712.10054*. http://arxiv.org/abs/1712.10054

Isinkaye, F., Folajimi, Y., & Ojokoh, B. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, *16*, 261–273. https://doi.org/10.1016/j.eij.2015.06.005

Jalali, M., Mustapha, N., Sulaiman, M. N., & Mamat, A. (2010). Webpum: A web-based recommendation system to predict user future movements. *Expert Systems with Applications*, *37*, 6201–6212. https://doi.org/10.1016/j.eswa.2010.02.105

Lau, J. H., & Baldwin, T. (2016). An empirical evaluation of doc2vec with practical insights into document embedding generation. *CoRR*, *abs/1607.05368*. http://arxiv.org/abs/1607.05368

Lops, P., de Gemmis, M., & Semeraro, G. (2010). Content-based recommender systems: State of the art and trends. *Recommender Systems Handbook*, 73–105. https://doi.org/10.1007/978-0-387-85820-3_3

Meel, P., Bano, F., Goswami, A., & Gupta, S. (2020). Movie recommendation using content-based and collaborative filtering. *Advances in Intelligent Systems and Computing*, 301–316. https://doi.org/10.1007/978-981-15-5113-0_22

Merve Acilar, A., & Arslan, A. (2009). A collaborative filtering method based on artificial immune network. *Expert Systems with Applications*, *36*, 8324–8332. https://doi.org/10.1016/j.eswa.2008.10.029