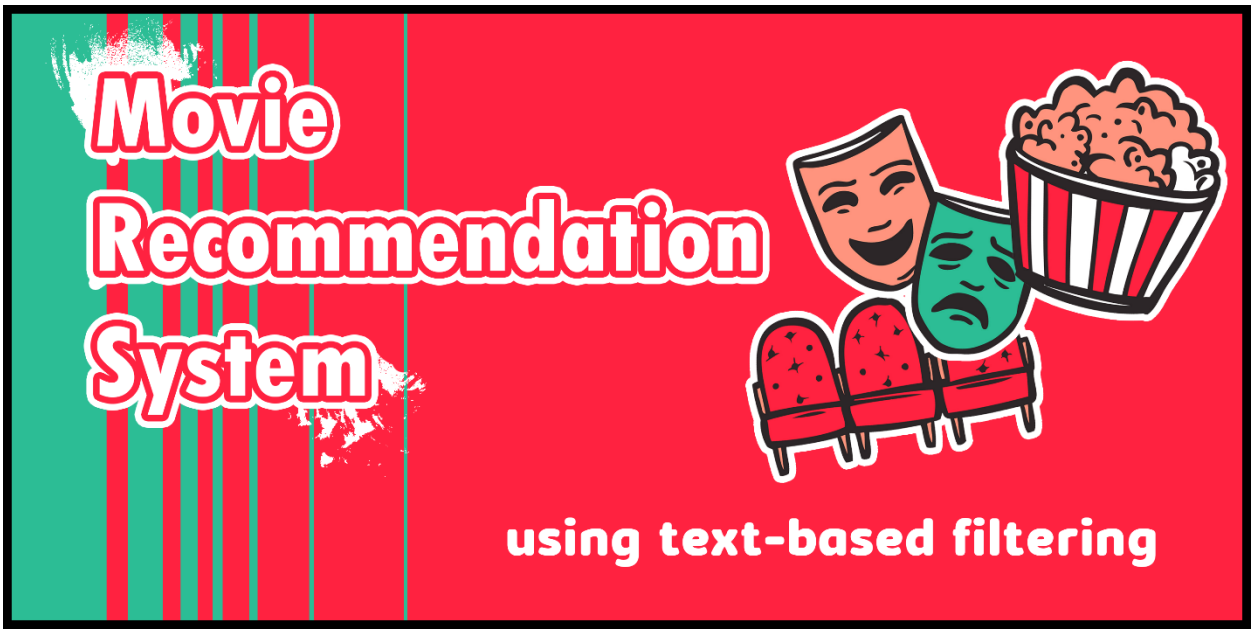


Preface

The goal of this project was to develop a means of recommending movies to people based on their interest in a single movie.

This is often the most information that an advertising company has to go on when creating targeted ads for potential customers, the standard use case being targeting someone with the ad when they are looking up a specific movie on a search engine.

I will be using unsupervised methods in order to cluster the movie genres appropriately for more suitable recommendations, and will be using text data scraped from imdb.com to determine similarity among movies within their genre clusters.

1. Data

I have used data from several sources for this project.

First, I determined the movies that would be used in this study from the MovieLens 20m dataset. This dataset consists of approximately 20 million user/movie/rating datapoints. I selected only movies which have been watched by at least 50 users. I also used this dataset for testing, limiting the tests to users who have rated 1000 or more movies.

https://grouplens.org/datasets/movielens/20m/

Second, I scraped data from IMDb, both for information on each movie, and for movie reviews from which I would mine relevant keywords.

https://www.imdb.com/

2. Method Overview

In order to initially separate movies into distinct groups, I used **Agglomerative Hierarchical Clustering,** an unsupervised method which separates the dataset into nested groups via a defined sense of 'closeness'.

Once the movies were in their genre clusters, I processed the text from the collected review data, and used cosine similarity to determine the most similar movies for each target movie. I combined similarity and overall IMDb rating to make the final selection of five recommended movies.

3. Data Collection and Processing

Part 1 – Movie Scraping

I wrote code using the Python package BeautifulSoup to extract potentially relevant data from IMDb for each movie I would be potentially recommending. I acquired a considerable amount of information, such as the director, principal actors, and movie duration, none of which was used in this project, but could be useful for posterity. I also recorded the movie genres, IMDb rating, age rating, and release year, all of which would be taken into account in my model.

Part 2 – Review Scraping

I needed something that would establish similarity between each movie beyond just their genres. After noticing that IMDb reviews were conveniently ordered in terms of how useful people considered them, I concluded that scraping the top 3 reviews for each movie would serve this purpose, as the most useful reviews would presumably contain a lot of relevant keywords.

Part 3 – Data Cleaning

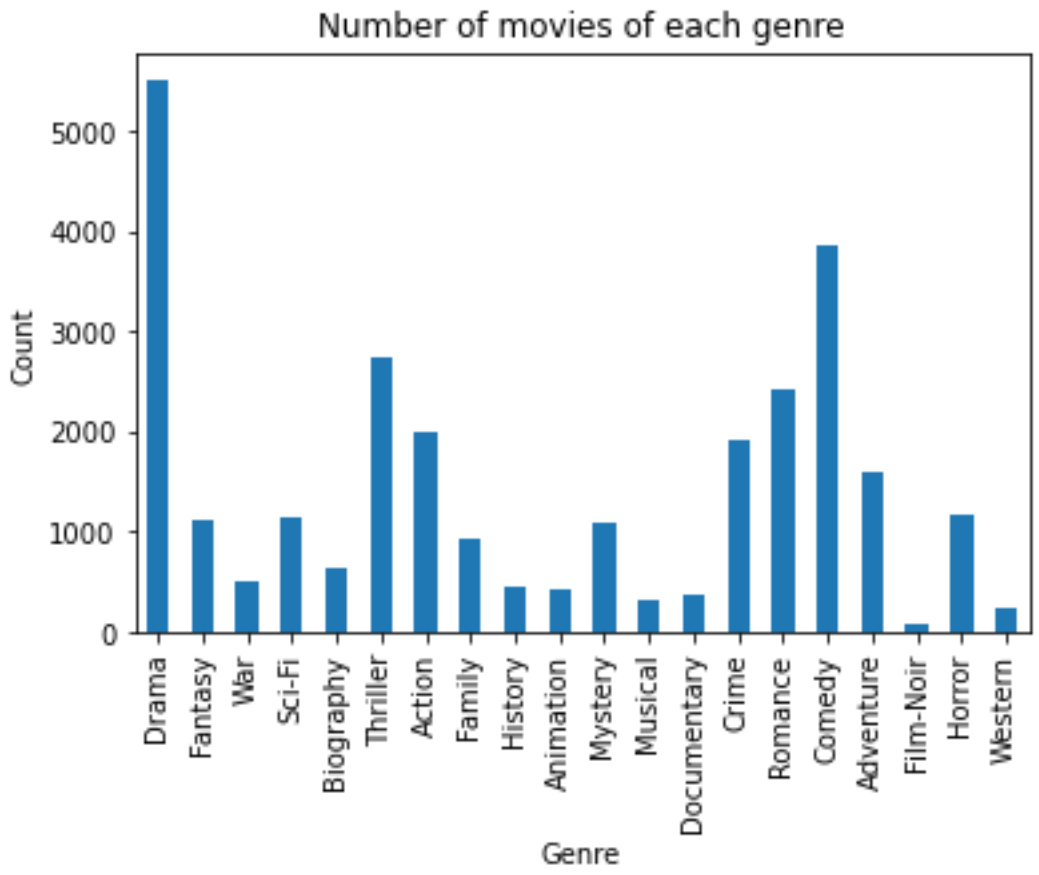
Fortunately, the IMDb data (and hopefully the scraping code I wrote) were of high enough quality that not much data cleaning was needed. Besides using the pandas library to adjust how the data was structured, I only found a minor case of duplicate entries, which I was able to identify and deal with via removing the duplicate entries. The data was ready for the next step.

4. Exploratory Data Analysis and Genre Clustering

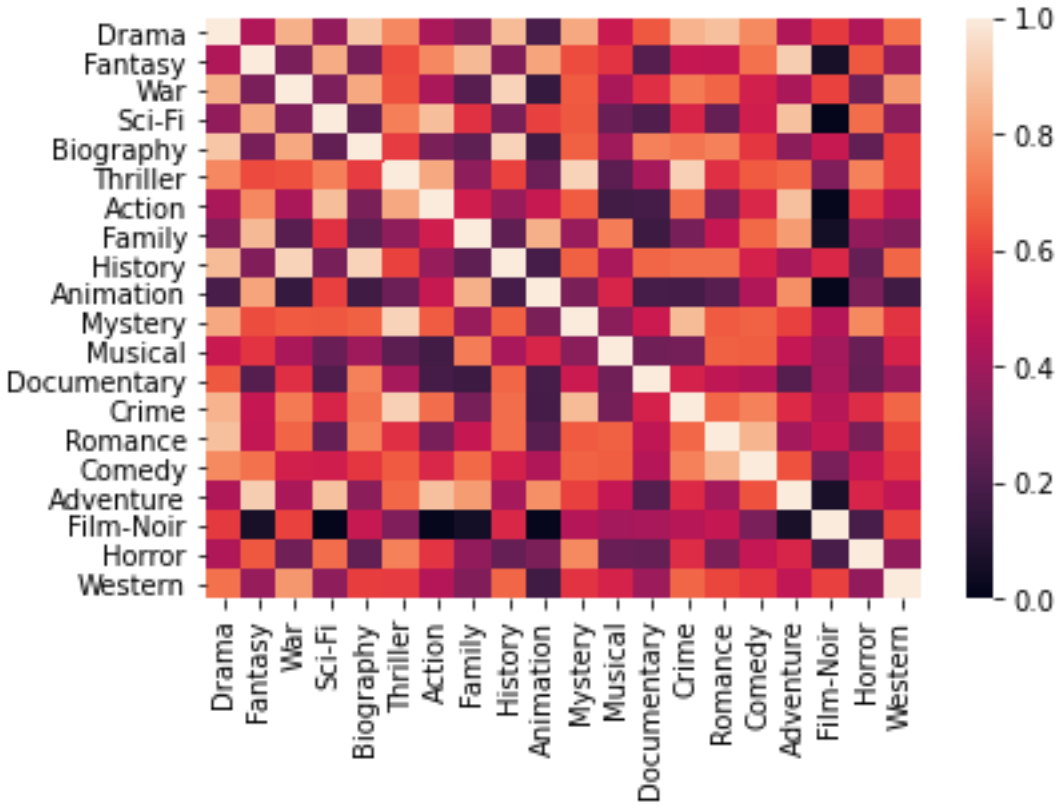
The purpose of this part of the project was largely to gain insight into the movie data, specifically the variation among genres, and to cluster the data accordingly.

First, I created dummy variables for each genre, so that each genre corresponded to a column of the data frame. Each (movie,genre) entry contained a 1 if the movie belonged to that genre, otherwise a 0. Many movies belonged to multiple genres, an aspect which is to be relevant in this project.

First, I graphed the number of movies belonging to each genre in the dataset, as shown below:

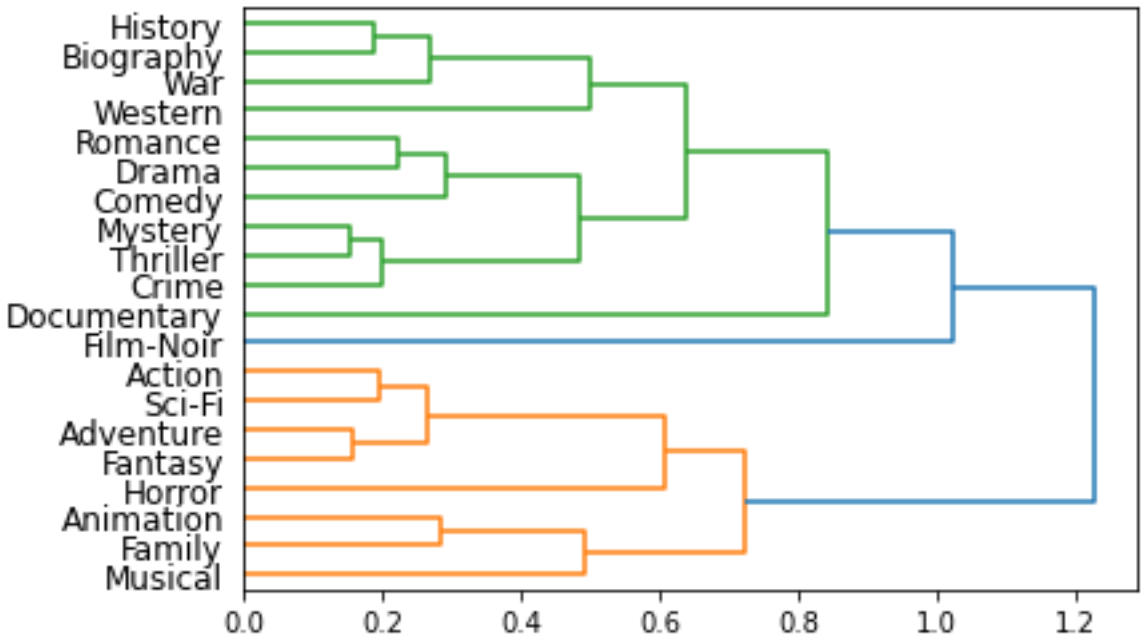


Then, I created a heatmap using the Seaborn package, to determine similarity between genres. I evaluated this similarity via comparing how many movies of each genre each user had watched, standardizing these results, and then using cosine similarity to build the heatmap, shown below.



As can be seen, the lighter squares correspond to higher similarity. For example, adventure and fantasy have a lot of overlap, whereas action and musical do not.

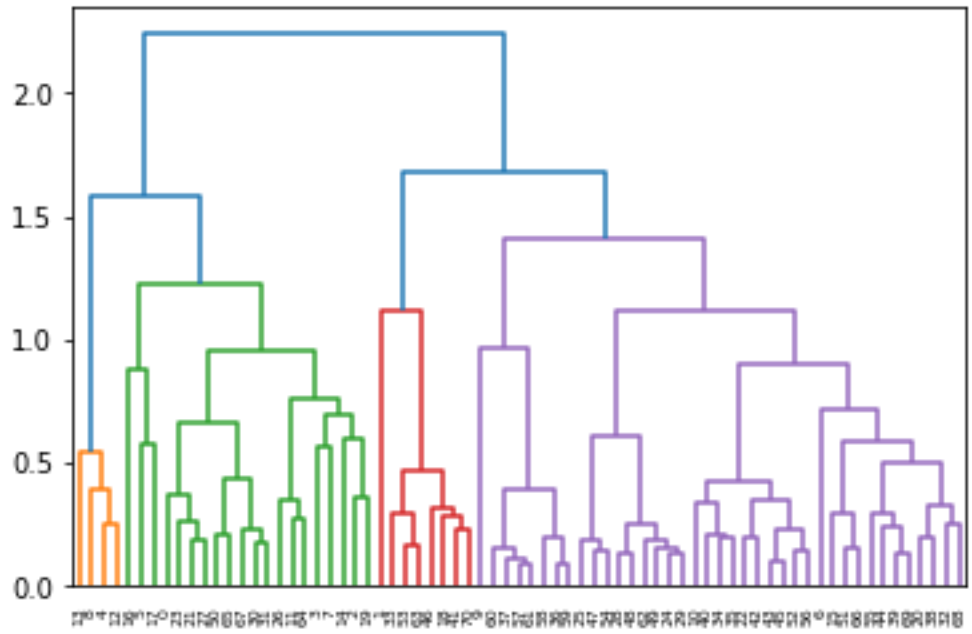
Since genre information is important when making recommendations to people (someone who has expressed interest in horror is unlikely to want to be recommended children's animation), I took this idea further, and used this information to cluster the data, the clusters being shown in the dendrogram below:



The way in which this is to be interpreted is that genres connected at a lower numerical value are ‘closer’ than ones connected higher up. As such, adventure and fantasy are very close, as are animation and family (since many animated movies are directed at children).

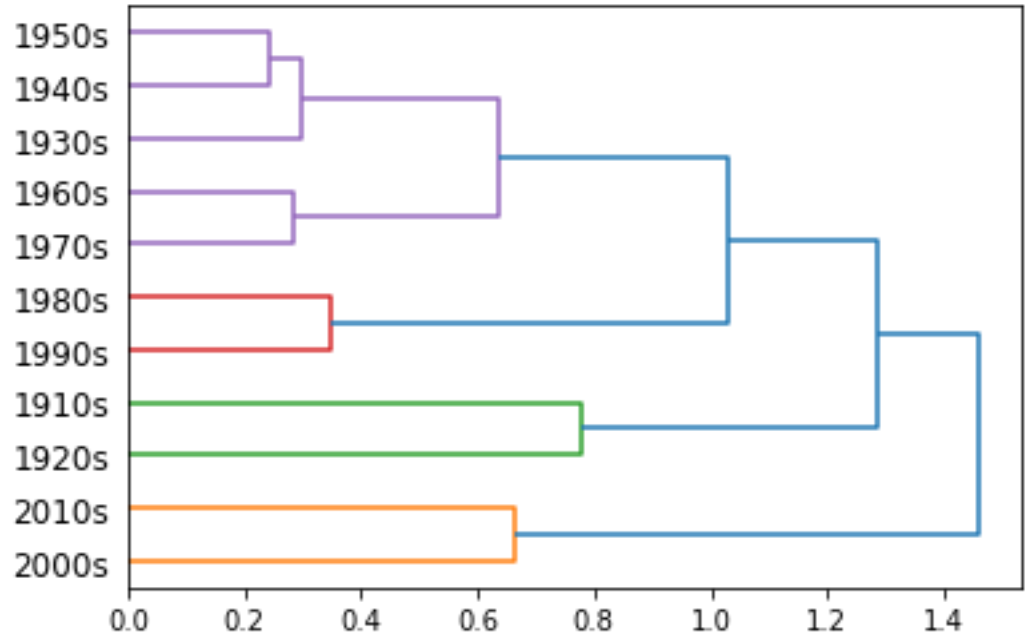
However, these results still didn’t have the specificity I wanted, so I decided to cluster the data again using pairs of genres instead. I would then be able to make the distinction between say, family comedies and horror comedies, which would not be possible with single genres.

The dendrogram for this is below, however there were too many clusters for me to be able to include the genre names in the graphic:



What can be immediately interpreted from this is that there are four distinct large clusters, each containing many smaller subclusters. The practical use of the dendrogram besides visualization is as a good visual aid for where the cutoff for the clusters would be. I decided to use two different cutoff values (to create clusters and 'superclusters') in case the clusters alone contained too few options.

I then applied a similar process to clustering movies based on release decade, another relevant feature when it comes to recommendations. Here is the resulting dendrogram:



The results are pretty intuitive.

I then combined the clusters for genres and dates to get the overall clusters each movie would belong to.

5. Preprocessing

This part of the process was primarily to extract meaningful information from the movie review data I had collected earlier.

For each movie, I combined the reviews into a single block of text, and separated it into 1-grams (single words) and bigrams (pairs of words). I used a Porter stemmer from the nltk package to stem the words. Stemming means reducing similar words, such as take and taken, to a single base 'word', to make document comparison more effective.

I then used Tf-Idf (term frequency/inverse document frequency) to rank the relevance of each 1-gram and bigram in the entire corpus (all the reviews for all the movies). I saved the top ranked words and word pairs to a file, and then went over the file manually, removing any words and word pairs that weren't valuable comparators. For example, 'good movie' offers little information, while 'car chase' gives insight into the content of the movie itself.

At the end, I was left with approximately 2000 1-gram and bigram features to determine similarity between the movies.

I finally iterated through the reviews for each movie again, recording how many of each relevant word/bigram were in the reviews in a dataframe. For example, the data entry for 'Toy Story' and 'cowboy' would be '3' if the word 'cowboy' appears in the reviews for Toy Story 3 times.

6. Modeling

After preparing the data and developing the model, it was finally time to see it in action.

I used four variations of the model to evaluate performance.

First, I used a purely random model, that separated the movies based on their age rating, but otherwise made the selection at random.

Second, I used a model that again separated the movies based on age rating, but chose the top 5 movies by IMDb rating instead of choosing them randomly.

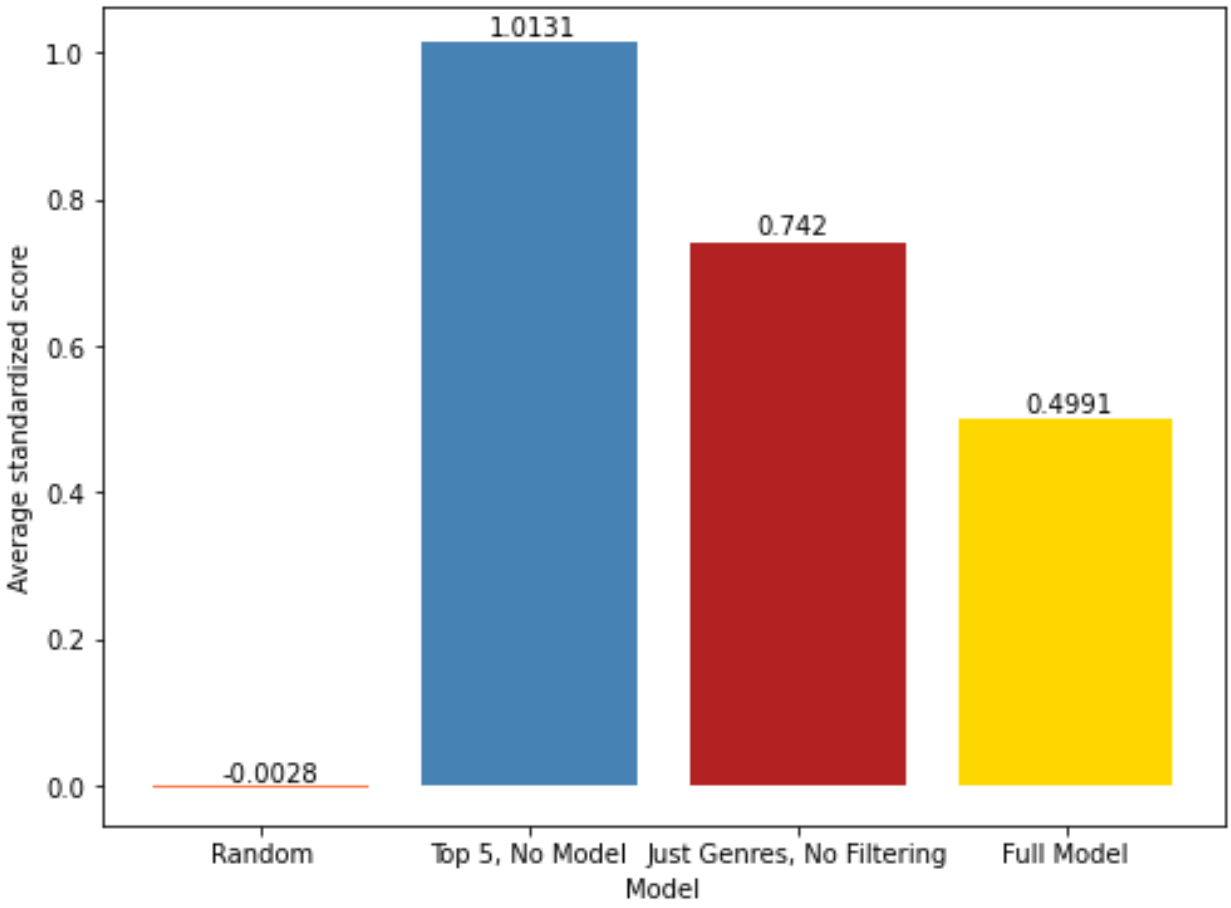
Third, I used a model that incorporated the different genre clusters into the mix, but didn't use any filtering based on the reviews.

Finally, I used the full model, that filtered based on age rating, genre clusters, and the keywords from the movie reviews.

I also decided to use three different metrics for evaluation.

The first was the average standardized score of the recommended movies. I would establish this score via testing the model on the users from the MovieLens dataset. First, I would standardize the scores they gave each movie (so that they have a mean of zero and a standard deviation of 1). Then, I would iterate through all the users, picking 10 randomly sampled movies for each user, and average the standardized score of the movies returned by the model. For the random model, the standardized score would feasibly be around zero, while a performant model would want as high a score as possible.

Here are the results for each model:

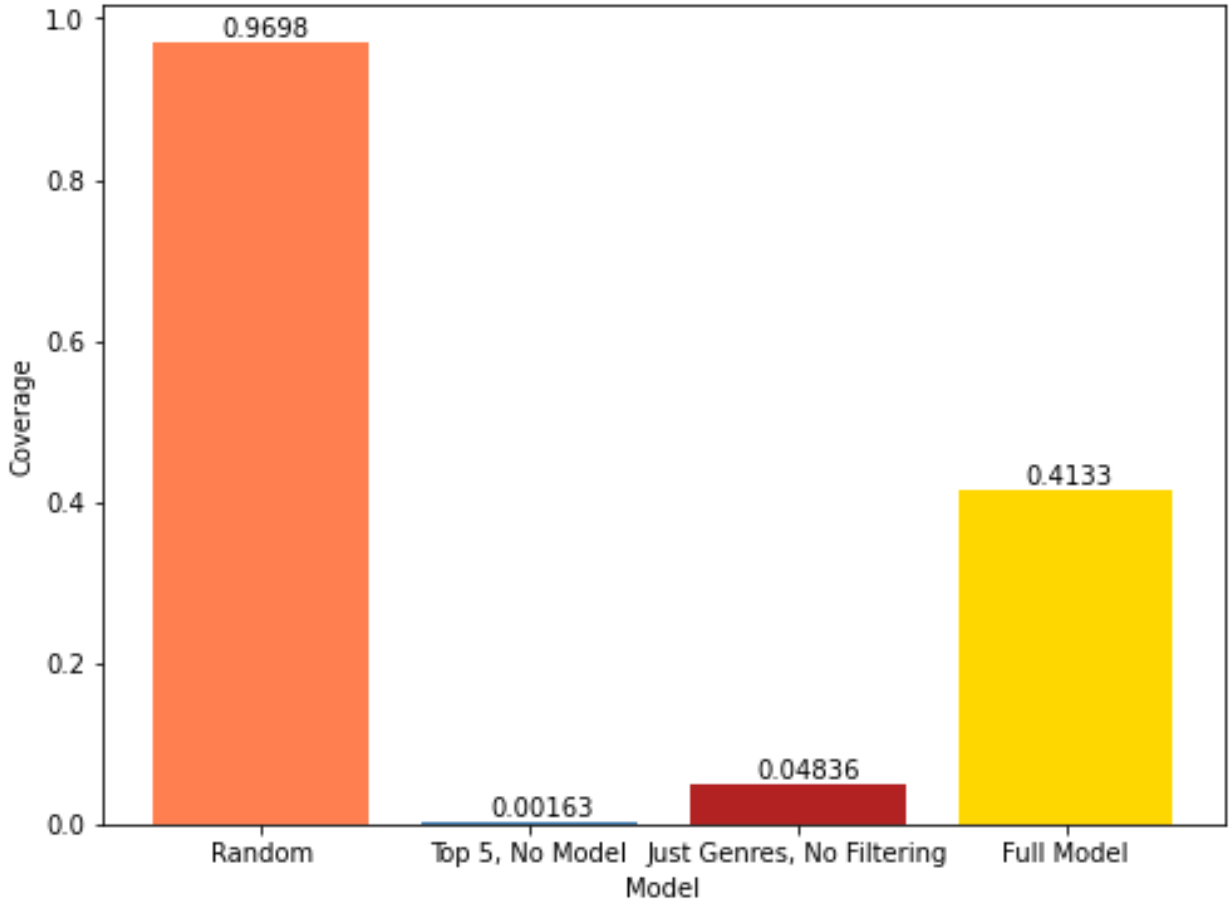


As expected, the random model essentially resulted in an average score of zero (it provided no valuable information).

The full model had a much better score of approximately 0.5, meaning that the average recommendation was 0.5 standard deviations above the mean. Both top 5 models outperformed the complete model via this metric, which was to be expected, because the most popular movies are naturally going to have very high ratings.

However, this metric is not the only important one, as will be seen below.

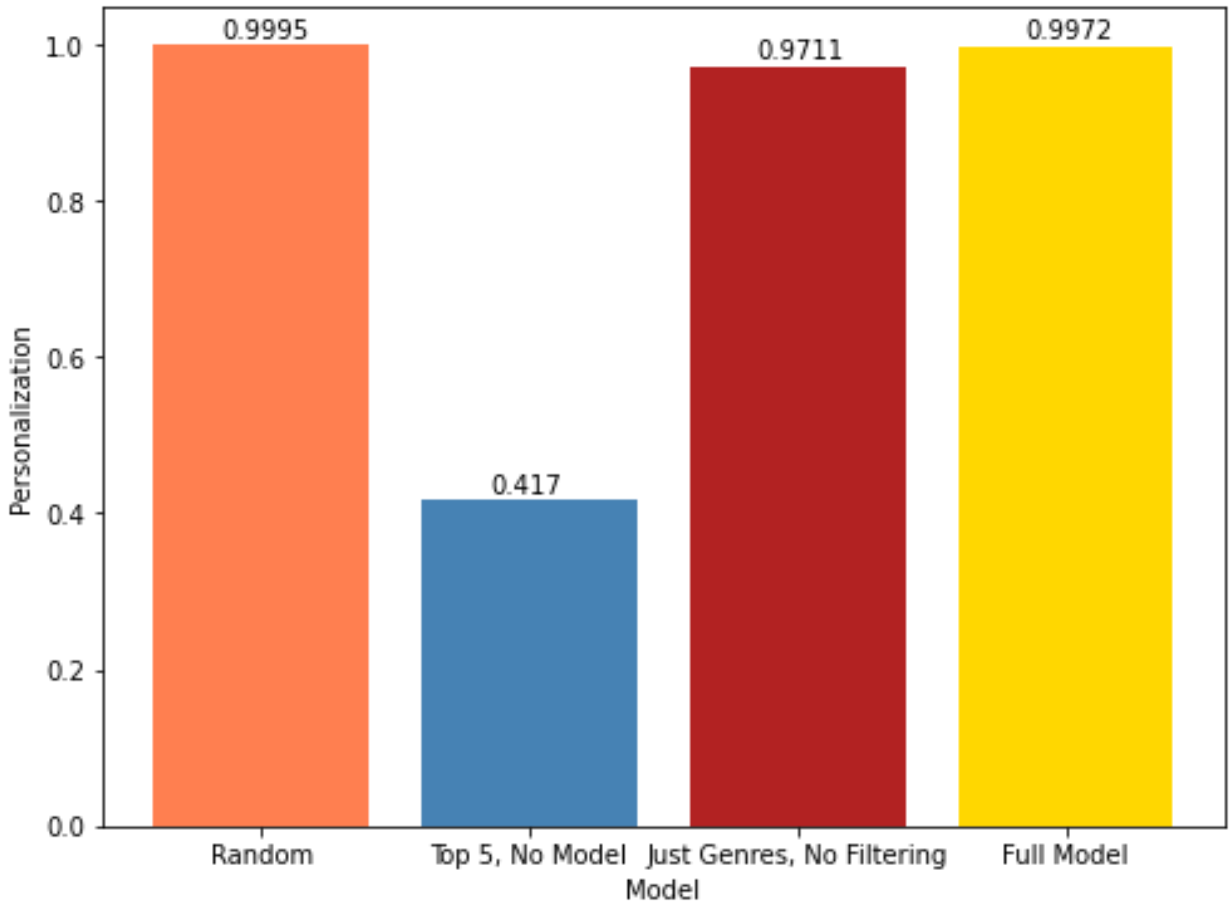
The second metric to be used was coverage, which is simply what proportion of the movies in the database are being recommended at all. This is important because companies such as streaming services are going to want to draw customers' attention to as much of their catalog as possible so that they remain interested.



Of course, the random model provided almost full coverage, because there are no restrictions on which movies are selected. However, the full model *significantly* outperformed the other two, recommending over 40% of the total catalog, while the other two only recommended 0.2% and 5% respectively. Again, this is important because while people are likely to enjoy the top 5 movies in a given category, they're also likely to have already seen them, so a much more diverse set of recommendations is going to keep them interested in the service.

Thirdly, I incorporated personalization, which measured how different the recommendations were based on each movie. As an extreme example, if all movies had zero overlap in the movies recommended to them, the model would have a personalization score of 1. If every movie resulted in the same recommendations, the personalization score would be zero. A high personalization score is valuable because it means the model is more tailored to specific users,

and in the case of movies, isn't just going to be going for the common choices, which people are more likely to have already seen.



Again, the random model has close to a score of 1, as expected. The full model is almost as performant though, with a score of 0.9972, indicating that the recommendations for each movies are largely different. The model taking only genres into account also has a high personalization score, though notably less than the full model, and the top 5 model has a far lower score than the rest.

When taking all these metrics into account, I believe it can be concluded that the full model performs very well, providing recommendations with a high average score from the users, while being very personalized and covering a large part of the overall catalog.

7. Conclusion

After analyzing the model's performance, I can conclude that it fulfils all the performance requirements for an effective recommender. Its recommendations are, on average, scored significantly higher than the mean by the test users, the recommendations have a guaranteed baseline relevance to the original movie due to the genre clustering and additional similarity

measures from the keyword filtering, the recommendations are highly personalized to the original movie, and they draw from a significant percentage of the overall catalog.

Additionally, the matrix of recommended movies only needs to be constructed once, and then accessed to provide the recommendations for each movie extremely quickly. It can also be periodically updated for new releases, as IMDb themselves update regularly, and reviews come in quickly.

Therefore, the model can remain current via periodic updates, which is a practical necessity.

Some additional filtering processes could be added in the future. Incorporating the information on the director and principal actors could potentially be valuable, since people are often drawn to specific figures in the industry. This is accomplished to a degree via the text filtering, since some of the keywords are celebrity names.

8. Acknowledgements

https://www.springboard.com/ for a wealth of information on data science.

https://towardsdatascience.com/ for some additional insight into certain aspects of data science, specifically recommendation system performance metrics and text filtering.