Machine Learning Engineer Nanodegree

Capstone Project

Simon Jackson

June 01, 2017

I. Definition

Project Overview

E-commerce sites must implement recommender systems that predict the ratings or preferences of users for products, to help users cope with overwhelming options [1]. Recommender systems ensure that users can be provided with personalised results to save them time and improve the user experience. For example, Netflix will try to present movies to you that you will like, rather than you having to sift through their entire database to find what you are seeking. Similarly, Amazon ought to recommend books you're likely to enjoy, rather than you having to read the blurb of every book to find something appealing.

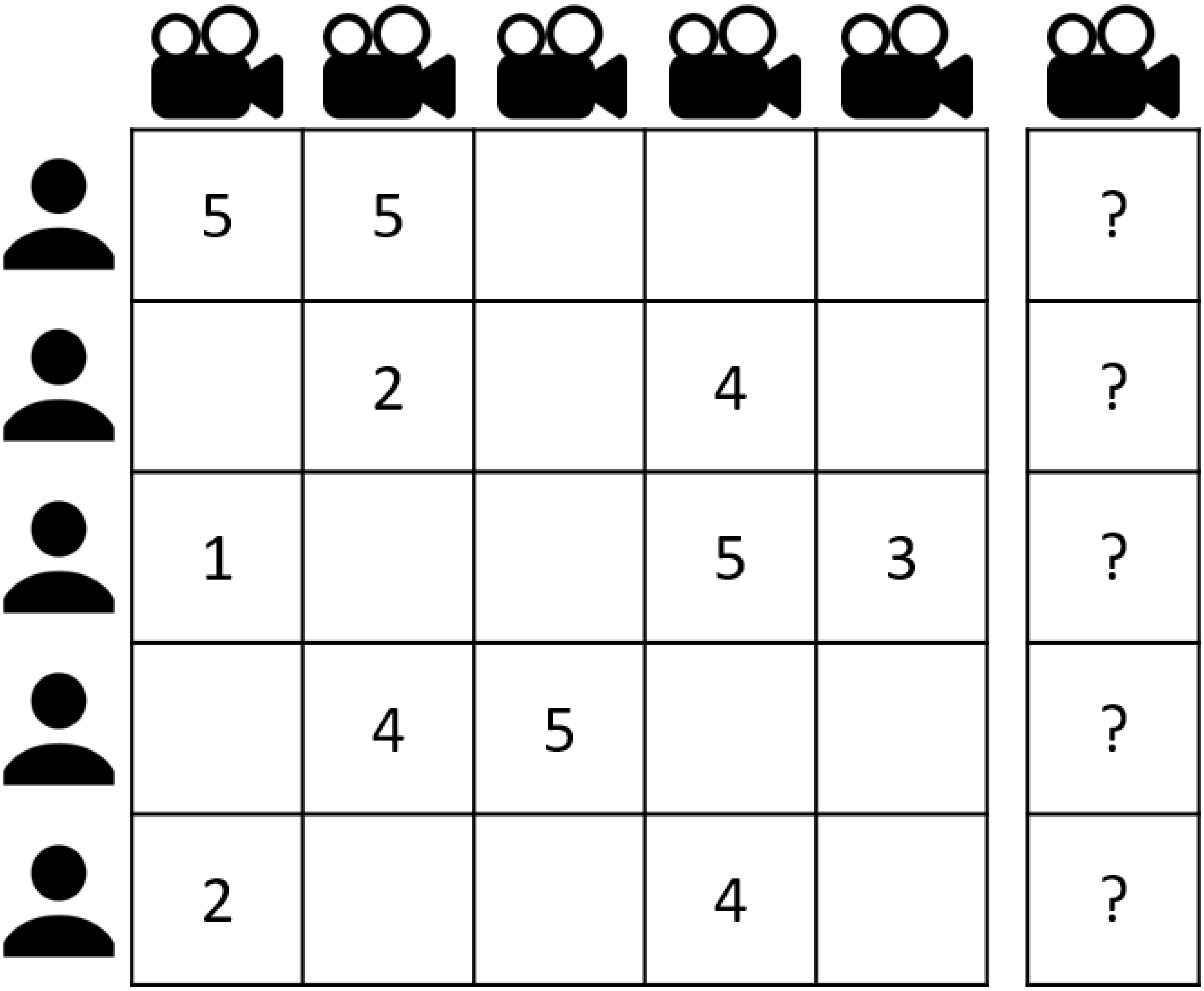
Recommender systems often take the form of collaborative or content-based filtering. The former involves predicting what a user will like based on their similarity to other users[2]. The latter involves matching content to a user based on similarities among the content[3]. Hybrid approaches, which combine content-based and collaborative filtering, can also be constructed.

In this project, I will compare the ability of various recommender systems to predict how different users will rate movies (out of 5-stars) that have **not been rated by any users**. This problem prohibits the use of purely collaborative filtering approaches, instead of depending on content-based or hybrid recommender systems. These systems will be trained by combining information from the MovieLens 20M and IMDB 5000 Movie datasets, which are both available via the online machine learning challenge platform, Kaggle.

Problem Statement

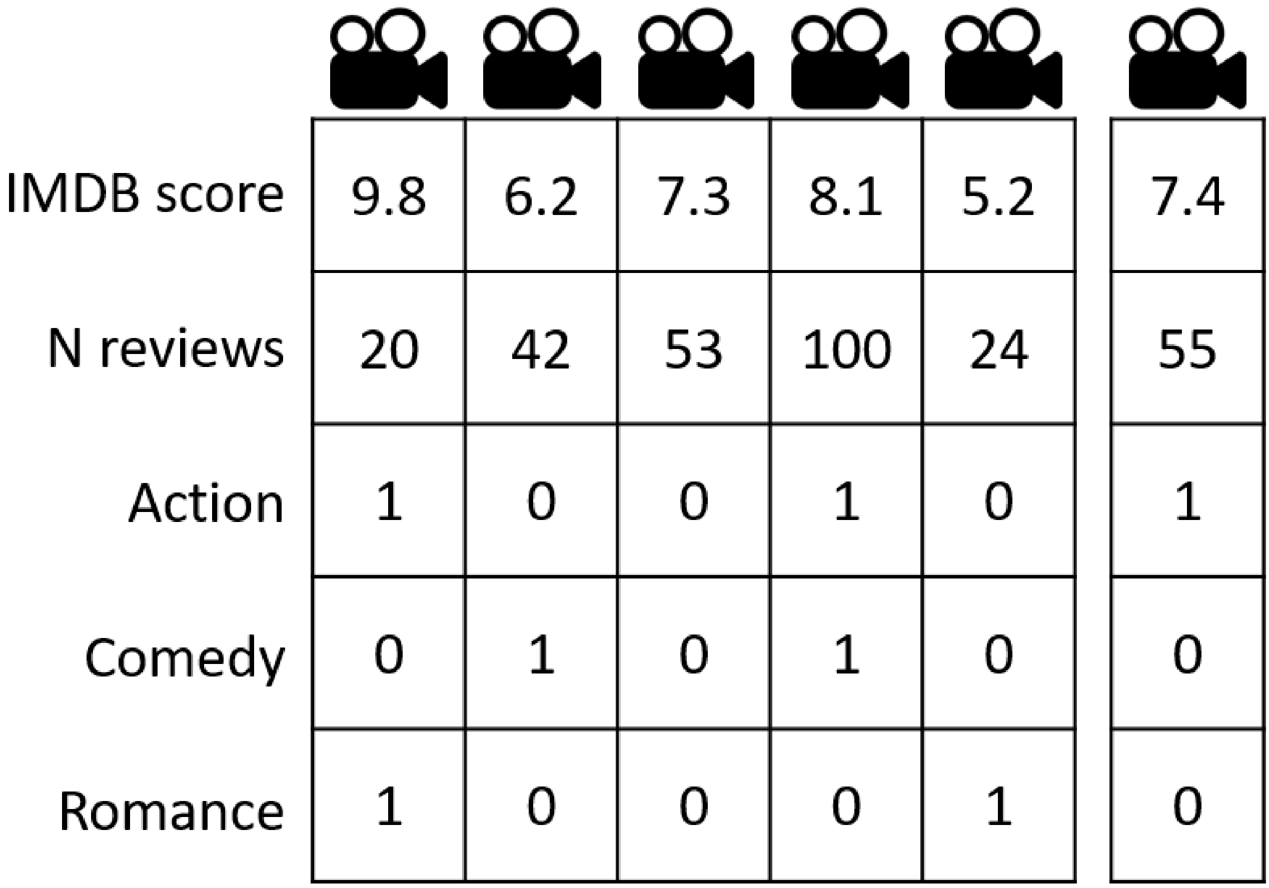
Imagine you work at Netflix and have added new movies to the service. You'd like to recommend these to people who will like them. It's impossible to determine whether a particular user might like one of these movies based on other, similar users because other users have not rated the film. It's also challenging to see how the user rated other similar movies because user ratings are relatively sparse.

This problem of estimating user ratings for new movies is visualised in the Figure below. In this Figure, rows represent users and columns represent movies in the database. User ratings from 1 to 5 populate the matrix cells but are sparse. The right-most column represents the introduction of a new film for which no user ratings exist. The problem is to estimate the scores for these cells denoted "?".



	🎥	🎥	🎥	🎥	🎥	🎥
👤	5	5				?
👤		2		4		?
👤	1			5	3	?
👤		4	5			?
👤	2			4		?

A separate source of information is required to predict these ratings: features about the movies. In this project, this information will be features scraped from the movie review site, IMDB. The Figure below represents an example of how this data looks.



	🎥	🎥	🎥	🎥	🎥	🎥
IMDB score	9.8	6.2	7.3	8.1	5.2	7.4
N reviews	20	42	53	100	24	55
Action	1	0	0	1	0	1
Comedy	0	1	0	1	0	0
Romance	1	0	0	0	1	0

Unlike the rating data for which user’s score are unknown, information from the IMDB database is known for all movies, including those being newly added for which the ratings wish to be estimated. This information makes it possible to determine how similar the new movie is to movies that already exist in the user-rated database. These similarity scores can be used to derive rating estimates.

In summary, the goal is to create a recommender system that will predict users’ ratings of new movies, given that an external source of information about the movie (e.g., from IMDB) is available.

Metrics

Given a data set of known movie ratings, models investigating this problem can be evaluated by the accuracy of their predictions. For this project, these ratings are treated as a continuous variable. Therefore, an appropriate metric for evaluating model performance will be the root mean square error (RMSE). The RMSE has been the metric used in similar problems such as the famous Netflix Prize).

The RMSE is calculated by squaring the error terms (residuals) for predictions on a given set of data points, calculating the means of these, and taking the square root. For a vector of true values (*y*'s) and corresponding predicted values (*y*-hat), the formula for calculating RMSE is shown below:

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n}}$$

A value of zero indicates that all predictions perfectly match the true values. The more positive the value, the worse the performance.

II. Analysis

Data Exploration

The data used in this project comes from two open-source projects:

- MovieLens 20M Dataset: Over 20 Million Movie Ratings and Tagging Activities Since 1995
- IMDB 5000 Movie Dataset: 5000+ movie data scraped from IMDB website

The MovieLens dataset is the principal data source for the collaborative filtering component of the model. It contains individual user ratings of movies on a 5-star scale (with five being the best and one being the lowest). In total, there are 20,000,263 movie ratings given by 138,493 unique users.

The IMDB dataset is to be used as the key source for the content-based filtering component of the model. It contains public information about 5,000 movies and includes the following variables:

> *“movie_title” “color” “num_critic_for_reviews” “movie_facebook_likes” “duration” “director_name” “director_facebook_likes” “actor_3_name” “actor_3_facebook_likes” “actor_2_name” “actor_2_facebook_likes” “actor_1_name” “actor_1_facebook_likes” “gross” “genres” “num_voted_users” “cast_total_facebook_likes” “facenumber_in_poster” “plot_keywords” “movie_imdb_link” “num_user_for_reviews” “language” “country” “content_rating” “budget” “title_year” “imdb_score” “aspect_ratio”*

These two data sets can be used in combination to train and test a hybrid recommender model for predicting the ratings that users will give “new” movies.

Below is a list of the major steps taken to clean the data and prepare it for analysis:

- **Remove duplicated movie information**
    - The IMDB dataset originally contained 5043 rows. However, these included duplicate cases. After removing duplicates, a total of 4919 movies were retained.
- **One-hot encode movie information**
    - Much of the movie information in the IMDB dataset was stored as string variables. In such cases, data was converted into one-hot encoded variables. To demonstrate, the genre(s) of each movie, which is a vital piece of information, was recorded in the IMDB dataset as a string variable in a format such as “Action|Adventure|Fantasy|Sci-Fi”. This variable was processed so that splits were conducted at the presence of each “|”, and then the values were turned into one-hot encodings. That is, instead of a single “genres” variable, a column was created for each genre label appearing in the data set (e.g., “Action”) which was given the value 1 for movies that had received this label, and 0 otherwise.
- **Imputing missing values**
    - There were a number of missing values in the IMDB dataset. Missing values were present for categorical (one-hot encoded) and continuous variables. Missing values were imputed as the mode for categorical variables and as the mean for continuous variables. One exception to this was “gross”, which recorded the

movie’s gross profit. Missing data here was more likely to indicate that no gross profit had been recorded or made. Thus, missing values were set to zero on this variable.

- **Transform continuous variables**
    - Continuous variables relating to the movies (e.g., “gross”, “actor_1_facebook_likes”) were transformed in two ways. First, variables that did not have a normal/Gaussian distribution were operated on to make their distribution more normal. In most cases, the best option was to calculate the log value of the variable. In some, the square root of the variable was taken. Finally, each continuous variable was normalised to have a value between 0 and 1. This was because these variables were going to be used for computing similarity among movies. If they were on very different scales, then this may bias the similarity algorithm.
- **Retain data for movies only appearing in both data sets**
    - User ratings were given to many more movies that those included in the IMDB dataset. For this project, however, only movies that existed in both data sets were of interest. Therefore, user ratings given to movies that did not exist in the IMDB dataset were removed, leaving 13,426,294 ratings. Doing this cleaning involved finding a common identifier for movies across the datasets. This common identifier was the identifier assigned to each movie by IMDB. This

information was stored in a hashtable in the MovieLens dataset, which could be bound to user ratings. The same ID could be extracted from the movie URL from the IMDB dataset in the “movie_imdb_link” column. For example, the URL for James Cameron’s Avatar was "http://www.imdb.com/title/tt0499549/?ref_=fn_tt_tt_1". The movie ID is “0499549”, which is prefixed by “tt”. The regular expression (regex) ‘(tt[0-9]+)’ was used to extract this value from each URL, and then the “tt” was removed. This process resulted in both datasets containing a common movie ID, which was used to drop user ratings for movies that did not exist in the IMDB dataset.

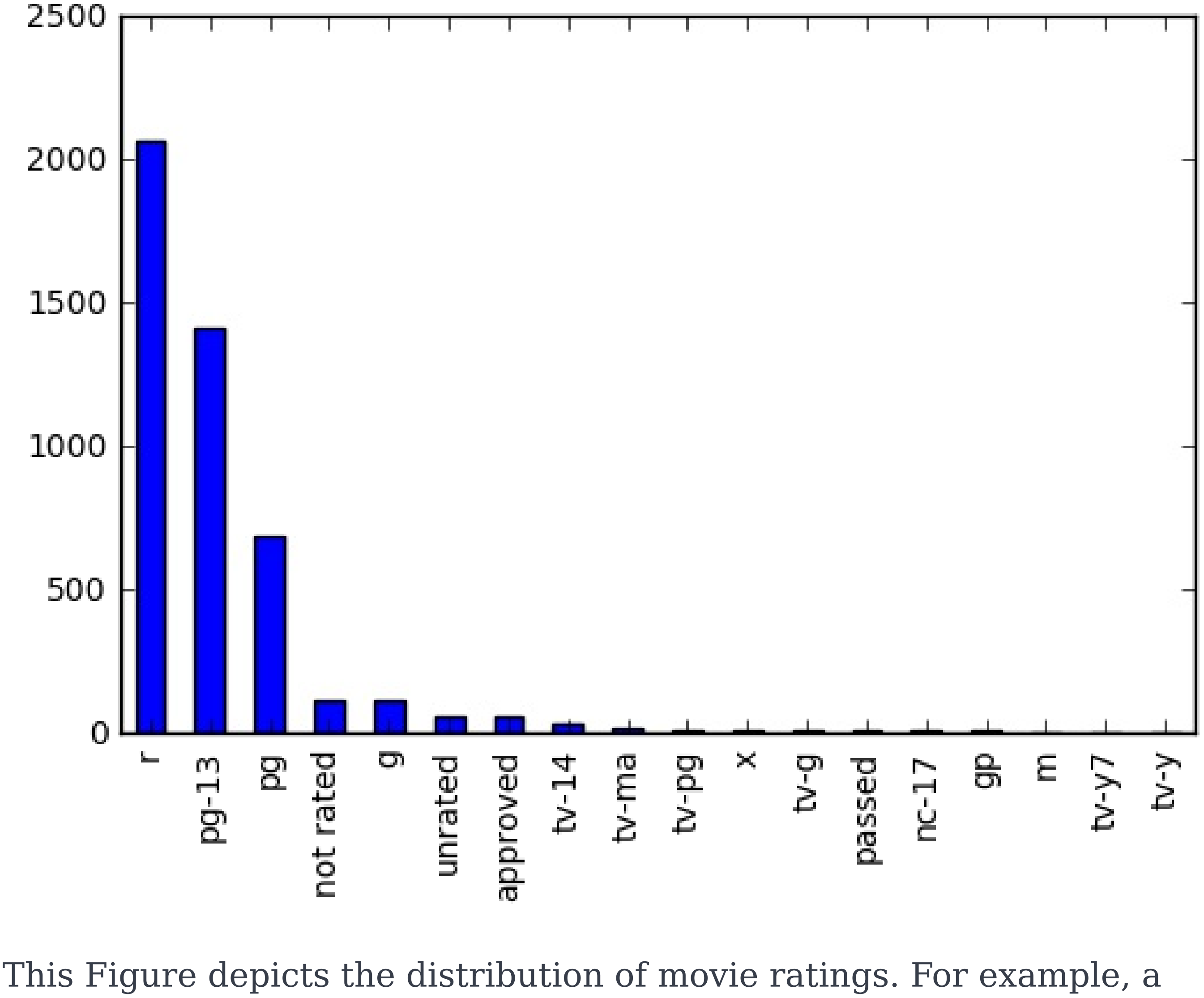
- **Retaining data only for users with many ratings**
  - Mentioned above, a significant number of ratings were still kept. When transformed to a user-by-movie-dataset, this matrix was huge. This project had to be executed on a basic laptop that did not possess the hardware needed to operate on such a large data set. Therefore, to reduce the data set size ratings from a sample of users were retained. Given the sparsity of the data, ratings were retained for users who had given at least 1000 movie ratings. This left 487,691 ratings given by 385 unique users.
- **Converting user ratings into a sparse matrix**
  - The final 487,691 ratings were converted into a sparse 385 (users) x 4919 (movies) matrix.

Exploratory Visualisation

Here we visually explore some descriptive information about the variables used to create the recommender system.

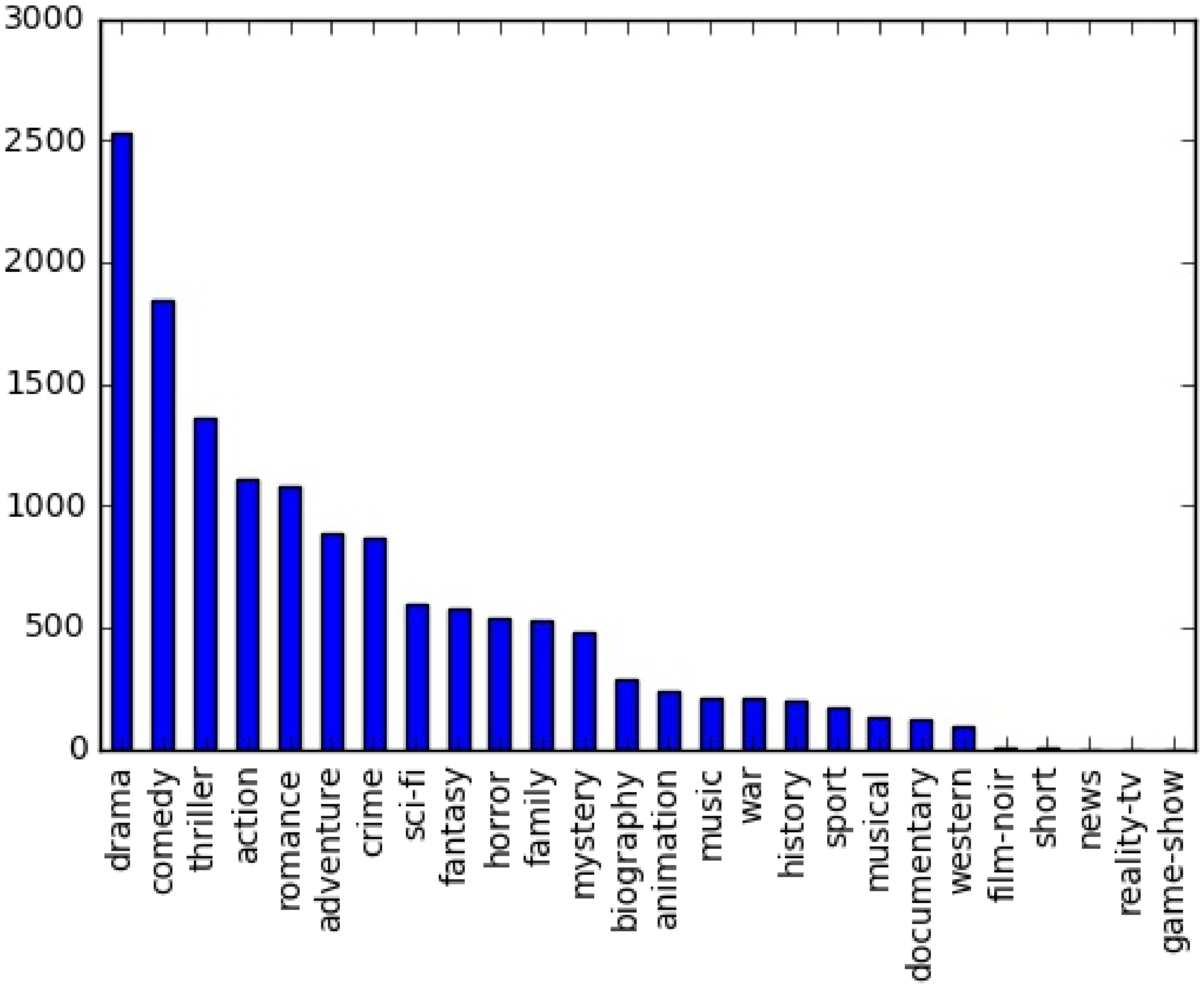
We will first explore discrete variables that were ultimately one-hot encoded:

Frequency plot of movie ratings



This Figure depicts the distribution of movie ratings. For example, a little over 2000 movies included in the analysis had a rating of “r”, a little under 1550 had a rating of “pg-13”, and so on. Most movies had ratings of “r”, “pg-13”, and “pg”.

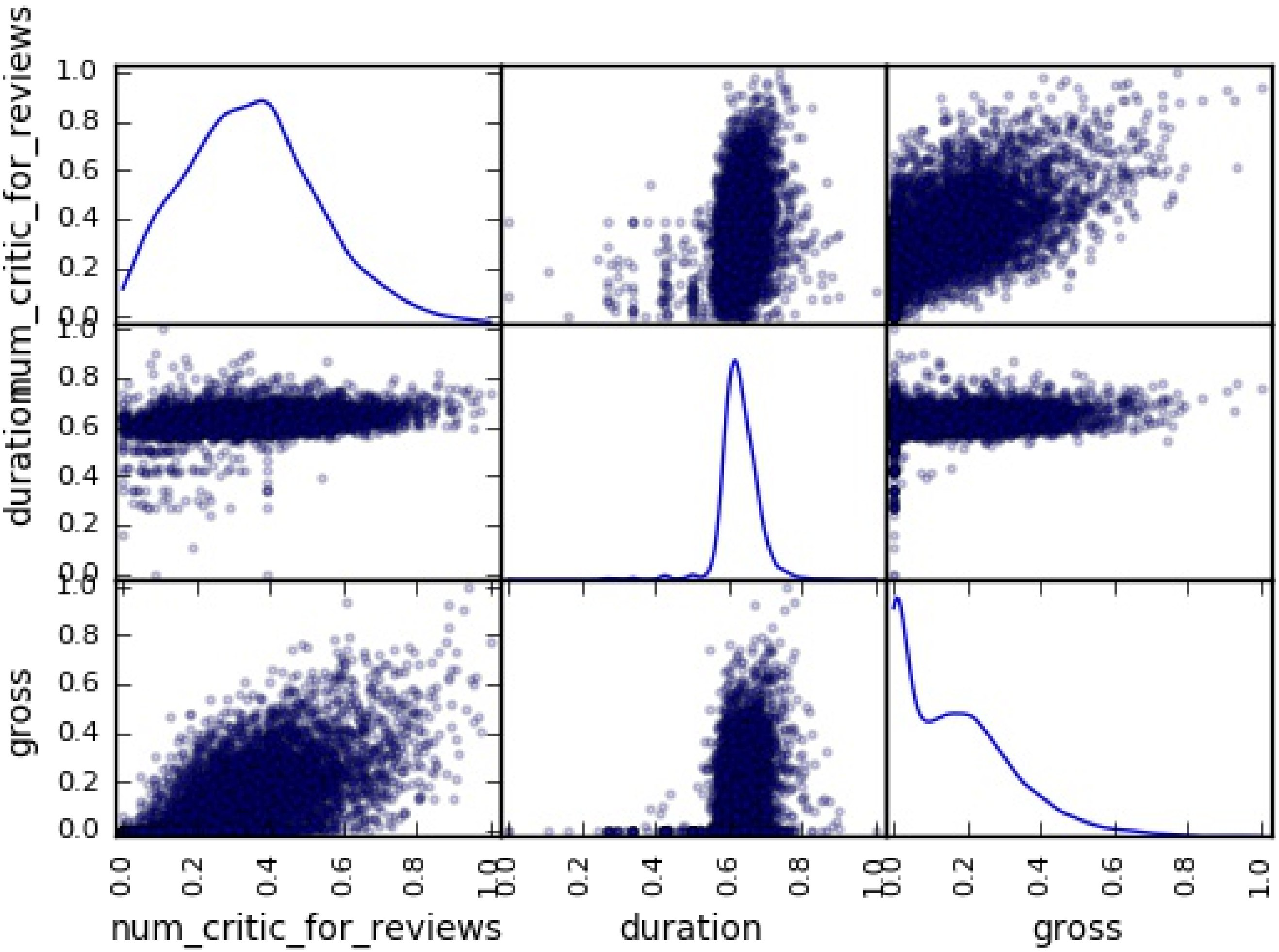
Frequency plot of movie genre tags



This Figure shows the distribution of genre tags assigned to the movies included in this analysis. Note that, unlike movie ratings above, multiple genre tags could be assigned to each movie. “Drama” was the most common genre tag, followed by “comedy”, “thriller”, “action”, and so on.

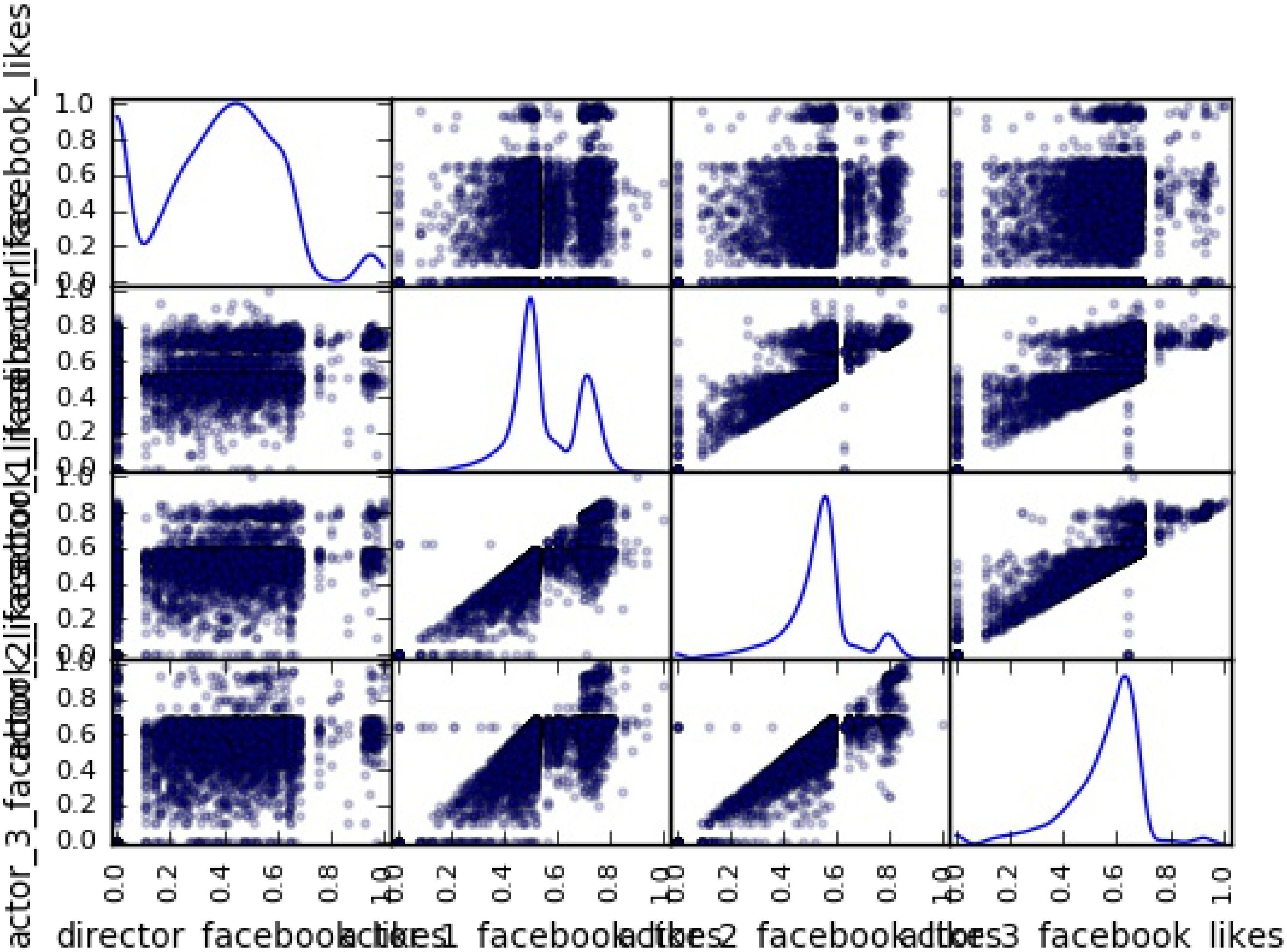
Next, we will examine some of the continuous variables that will be submitted to Principal Components Analysis in the modelling steps.

Scatter plot of movie credentials



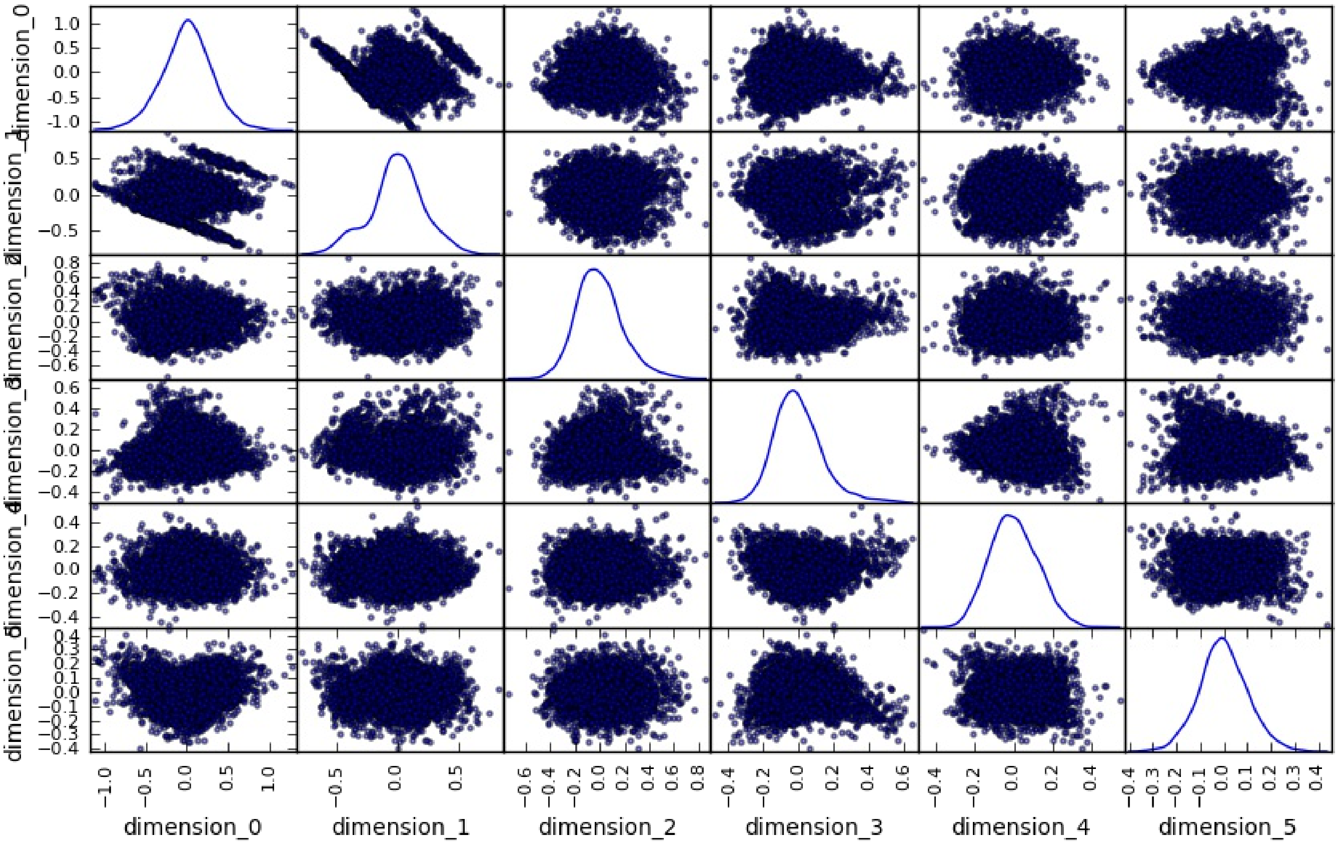
This Figure shows the distributions and covariance of three important variables related to each movie: number of critic reviews, movie duration, and gross profit, all shown after being normalised. It is clear that the number of critic reviews and gross profit are positively correlated, whereas the film duration is unrelated to these variables.

Scatter plot of Facebook likes



This Figure shows the distribution and covariance between the number of Facebook likes that the movie director and three principal actors have. It’s clear that the number of Facebook likes is positively correlated among the three actors, whereas the number of director facebook likes does not show a clear pattern with the actor information.

Scatter plot of PCA



This Figure shows the distributions and covariance structure among the six principal components extracted from all continuous variables in the IMDB dataset (an operation conducted during the modelling phases described below). As expected, all components are uncorrelated with each other. A useful feature to note is that the components are normally distributed, unlike some of the raw features shown in the previous Figures.

Algorithms and Techniques

This project will compare two methods for estimating users ratings for new movies: content-based filtering and a hybrid approach of content-based and collaborative filtering.

1. Content-based filtering involves a **k-nearest neighbours** algorithm. For example, take a new movie `m` and all of a user's

existing ratings `U = {rating-movie.i, rating-movie.j, ...}`. An estimate of the user's rating for `m` can be obtained by using ratings from the `k`-most similar movies to `m` rated by the user. There are a few major considerations for this algorithm:

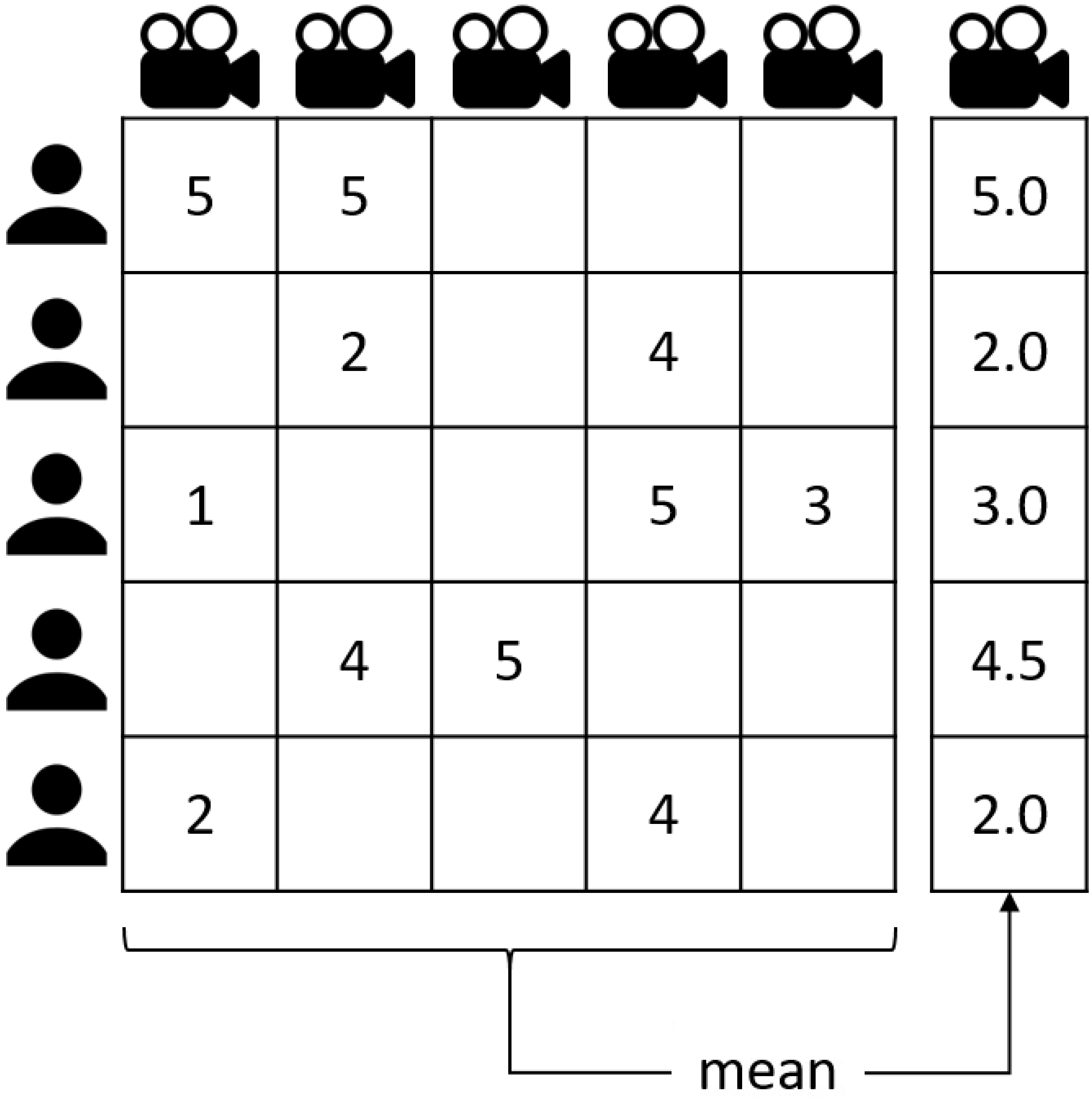
    - how similarity is computed
    - the value of `k`
    - how to combine ratings to produce an estimate

2. An issue with the content-based filtering only is the sparsity of the data. For example, what if the user has only rated movies that are *dissimilar* to `m`? To solve this, collaborative filtering can be used to estimate missing data within rated movies. These estimates, in turn, can be used by the content-based filtering approach described above. Combined, this approach is a hybrid recommender system. One of the most successful collaborative filtering algorithms for estimating sparse rating data in **non-negative matrix factorisation**. This approach attempts to factorise the user-by-movie rating matrix into two smaller matrices which can be multiplied together to reproduce the original matrix. In this way, the smaller matrices can be estimated by ignoring missing values and then, when multiplied back together, produce estimates in missing cells. The major consideration for this algorithm:

    - How many latent factors to factorise, `p`. That is, if the main matrix is of dimensions `m x n`, then the two smaller matrices will be of dimensions `m x p` and `p x n`.

Benchmark

As a benchmark, the estimated rating of a new movie by a given user will be calculated as the mean of all of the user's existing ratings. This estimate represents the typical rating assigned by a user to movies, and therefore serves as an appropriate and challenging benchmark. The Figure below demonstrates this benchmark approach for a new movie and existing ratings:



5	5				5.0
	2		4		2.0
1			5	3	3.0
	4	5			4.5
2			4		2.0

The performance of this benchmark approach was established using the user-by-movie-ratings matrix via a leave-one-out cross validation. That is, for each movie (column) in the matrix, user-wise (row-wise) means were computed for all other available ratings as estimates of users' ratings for that movie. Once the entire matrix was reconstructed via this approach, the RMSE (see Metrics) was calculated for all known ratings.

The RMSE for the benchmark model was 0.9396

III. Methodology

Data Preprocessing

This section outlines the data preprocessing steps conducted on both data sets. The major steps taken in each are described in more detail above.

The following preprocessing steps were done for the **IMDB dataset**:

- IMDB movie id was extracted from the movie URL.
- Duplicate movies were removed
- All discrete variables were one-hot encoded
- Missing values were imputed as the mean for continuous variables, as the mode for one-hot variables, and as 0 for gross (as movies with a missing gross value likely did not make a profit).

- Continuous variables were transformed to adjust their distribution to be more normal. The transformations applied were to square root or log transform certain variables depending on their distribution.
- Continuous variables were normalised to range between 0 and 1.

The following preprocessing steps were done for the **MovieLens data set**:

- IMDB movie id was linked to ratings by merging separate datasets.
- Movies not appearing in the IMDB dataset were removed.
- Users who made fewer than 1000 ratings were dropped.

Implementation

Implementation was split into three main stages:

1. Developing a content-based filtering recommender for new movies using k-nearest-neighbours
2. Developing a collaborative-based filtering recommender for existing movies using non-negative matrix factorisation
3. Developing a hybrid recommender system for new movies

Details of how each step was executed is described below.

Step 1: content-based filtering

This step involved developing a content-based filtering recommender that acts similarly to the benchmark model (by computing the mean of ratings), but in a more informed way. Specifically, rather than computing the mean of all of a user's ratings, this approach first calculates the similarity of the new movie to each rated movie and computes the mean based on ratings from the k-most similar rated movies. This step involved

- Finding a method that best captured "similarity" among movies
- Given similarity scores, finding a value of `k` that provided the best predictions

Step 2: collaborative filtering

This step involves developing a collaborative filtering recommender for movies already in the database (which could be used by a hybrid recommender in step 3). This involved applying non-negative matrix factorization (using stochastic gradient descent to handle missing values) to the full user-by-movie rating matrix. Tuning was required to determine the optimal number of latent dimensions that best estimated ratings.

Step 3: hybrid

This step involved combining steps 1 and 2. The collaborative filtering method developed in Step 2 was used to fill in the sparse rating data. Then, the content-based recommender developed in Step 1 was used to estimate ratings for new movies.

Refinement

Step 1: content-based filtering

The similarity between movies was calculated as cosine similarity, which captures the cosine angle between the feature vectors of two movies. A challenge was to determine which variables from the IMDB dataset would work best. Various combinations of variables were tested. To validate this method, a handful of movies known well to the author (especially ones that were part of a series) were selected for analysis. When a set of variables was tried, the 5-10 most similar movies to these validation movies was examined. The optimal variable combination was determined by the author's knowledge of which movies were similar (and those known to be in the same series).

To demonstrate, below are the most similar movies to "Pirates of the Caribbean: At World's End" based on different variable sets:

- *Using all variables*: Carlos; Spy Game; Superman; Street Fighter
- *Using Genres only*: The Legend of Hercules; Indiana Jones and the Last Crusade; Warcraft; Gods of Egypt
- *Using Genre, rating, gross, cast likes*: Pirates of the Caribbean: The Curse of the Black Pearl; Pirates of the Caribbean: Dead Man's Chest; Pirates of the Caribbean: On Stranger Tides

It was clear that using all variables did not produce good results. Selecting too few (e.g., just the one-hot-encoded genre variables) was not accurate either. Rather, a combination of genre, rating, and the continuous variables produced reasonable results.

To further improve this, the twelve continuous variables were submitted to Principal Components Analysis in an attempt to reduce the dimensional space being used by the cosine algorithm. It was decided a priori that as many components would be selected as accounted for at least 90% of the variance in these twelve variables. The result was that six components accounted for precisely 90% of the variance.

At this point, the movie similarity approach was tried again using one-hot-encoded variables relating to genre and rating, and the six principle components derived from the continuous variables. Below are the results of some tests:

```
MOVIES THAT ARE MOST SIMILAR TO: Harry Potter and the Hal
f-Blood Prince

200               Harry Potter and the Sorcerer's Stone
64     The Chronicles of Narnia: The Lion, the Witch ...
144                                                 Pan
38                            Oz the Great and Powerful
374                      Percy Jackson: Sea of Monsters
193            Harry Potter and the Prisoner of Azkaban
282             Harry Potter and the Chamber of Secrets
9                Harry Potter and the Half-Blood Prince
33                                  Alice in Wonderland
```

```
106                     Alice Through the Looking Glass
Name: movie_title, dtype: object
********************************************************



MOVIES THAT ARE MOST SIMILAR TO: The Avengers


52                             Pacific Rim
112                           Transformers
53          Transformers: Dark of the Moon
86     Captain America: The Winter Soldier
65                      X-Men: Apocalypse
8                  Avengers: Age of Ultron
10      Batman v Superman: Dawn of Justice
11                        Superman Returns
27              Captain America: Civil War
17                            The Avengers
Name: movie_title, dtype: object
********************************************************



MOVIES THAT ARE MOST SIMILAR TO: The Hunger Games: Catchi
ng Fire


216                   The Day After Tomorrow
236                            The Wolverine
97                                 Inception
29                            Jurassic World
```

```
689                            Jurassic Park
187                        War of the Worlds
433                         The Hunger Games
253                                Insurgent
204    The Hunger Games: Mockingjay - Part 1
185          The Hunger Games: Catching Fire
Name: movie_title, dtype: object
********************************************************



MOVIES THAT ARE MOST SIMILAR TO: Pirates of the Caribbean
: The Curse of the Black Pearl

133                                  Wrath of the Titans
1030                  Indiana Jones and the Last Crusade
54      Indiana Jones and the Kingdom of the Crystal S...
210                                  Clash of the Titans
13            Pirates of the Caribbean: Dead Man's Chest
18           Pirates of the Caribbean: On Stranger Tides
127                                 Thor: The Dark World
1                Pirates of the Caribbean: At World's End
202     Pirates of the Caribbean: The Curse of the Bla...
21                                The Amazing Spider-Man
Name: movie_title, dtype: object
********************************************************



MOVIES THAT ARE MOST SIMILAR TO: Star Wars: Episode VI -
```

```
Return of the Jedi

2394                               Conan the Destroyer
3418                                       Logan's Run
2876                   Star Trek II: The Wrath of Khan
40                                        TRON: Legacy
1068                Journey to the Center of the Earth
2031    Star Wars: Episode V - The Empire Strikes Back
237          Star Wars: Episode I - The Phantom Menace
234       Star Wars: Episode II - Attack of the Clones
1521        Star Wars: Episode VI - Return of the Jedi
2974                Star Wars: Episode IV - A New Hope
Name: movie_title, dtype: object
********************************************************
```

These results were excellent. For example, in the final test shown, the most similar movies to the fourth Star Wars movie are the five other movies in the same series.

Using the similarity metric defined in the stages above, the k-nearest neighbours approach described above was used to estimate movie ratings. As with the benchmark model, leave-one-out cross validation was conducted to derive performance metrics. The model was fit using five values of `k`: 10, 20, 30, 40, and 50. The results were as follows:

Using 10-most similar movies… RMSE = 0.8787
Using 20-most similar movies… RMSE = 0.8717

Using 30-most similar movies… RMSE = 0.8712

Using 40-most similar movies… RMSE = 0.8717

Using 50-most similar movies… RMSE = 0.8732

In all cases, the RMSE was better than the benchmark model, indicating that this was a useful approach to take. Comparing the results for different values of `k`, 30 was determined to be the most suitable.

Step 2: collaborative filtering

The collaborative filtering algorithm involved the use of non-negative matrix factorisation. Unlike content-based filtering, collaborative filtering cannot be used to predict new movie data. Instead, it is examined here as a potential method for filling in sparse rating data. This method attempts to find two matrices that, when multiplied together, best reproduce the original user-by-movie rating matrix. Specifically, given an original matrix of dimensions `n * m`, non-negative matrix factorization finds two matrices `n * p` and `p * m` where, in this case, `p` represents a number of latent dimensions. Stochastic gradient descent is used to iteratively sample available data (ignoring missing ratings) in order to derive the two matrices. The two complete matrices can be multiplied together to closely reproduce the original ratings as well as provide estimates for missing cells.

This component was refined via k-fold cross-validation. Specifically, a number of values for `p` (latent dimensions) were chosen. Then, cells in the original data were randomly assigned to one of five “folds”. For each `p` and fold, the following was done:

- Set all cells in the fold to missing.
- Conduct non-negative matrix factorization on the matrix.
- Multiply the resulting matrices together to reproduce the data.
- Calculate the RMSE for cells in the fold.

For each value of `p`, this is done once for each fold. The mean RMSE for the five folds is then computed as an indicator of the performance of `p`.

Values for `p` that were first tested were 10, 40, 70, and 100. This yielded the following results:

RMSE for 10 latent dimensions: 0.7597
RMSE for 40 latent dimensions: 0.8356
RMSE for 70 latent dimensions: 0.8911
RMSE for 100 latent dimensions: 0.9327

It was clear that lower values of `p` were performing better. The analysis was redone to test values of 2, 4, 6 and 8. The results were:

RMSE for 2 latent dimensions: 0.7662
RMSE for 4 latent dimensions: 0.7534
RMSE for 6 latent dimensions: 0.9822
RMSE for 8 latent dimensions: 0.7576

Based on these results, 4 latent dimensions were chosen as the best option for estimating the sparse rating data.

Again, it is important to note that these RMSE scores are based on estimating ratings for movies that other users have rated. Thus, although they are superior to the RMSE scores obtained for the benchmark and content-based approaches, this algorithm does not, by itself, solve the problem being addressed: to estimate scores for new movies.

Step 3: hybrid

The hybrid model did not involve any refinement and will, therefore, be reported in the results section below.

IV. Results

Model Evaluation and Validation

Testing the final hybrid recommender involved the following steps:

- Randomly sample a proportion of movies to hold out as “new” movies. This proportion was set to 0.2.
- Using the remaining data, use the collaborative filtering algorithm to fill in all missing rating data.
- For each “new” movie, use the content-based filtering approach to estimate a rating for each user.

The final RMSE of this model was 0.9784. It seems that this is worse than the benchmark model. This means that the use of true ratings (done by the content-based filtering algorithm alone) was superior to their combined use with ratings estimated via the collaborative filtering method.

Thus, the best-performing and final approach is the k-nearest-neighbours content-based filtering recommender system. Given that the RMSE was established using leave-one-out cross-validation, we can feel confident that these results reflect a robust ability for the model to generalise to new data.

In summary, the results indicate that the final model is a reasonable solution to the problem (theoretically and empirically) and that it aligns with the expectation of supporting personalised movie recommendations. The results can be considered robust given the size of the data and method of cross-validation used. The final model performance indicates that the accuracy of predictions made on future data can be trusted to be better than ratings obtained through the use of simple heuristics.

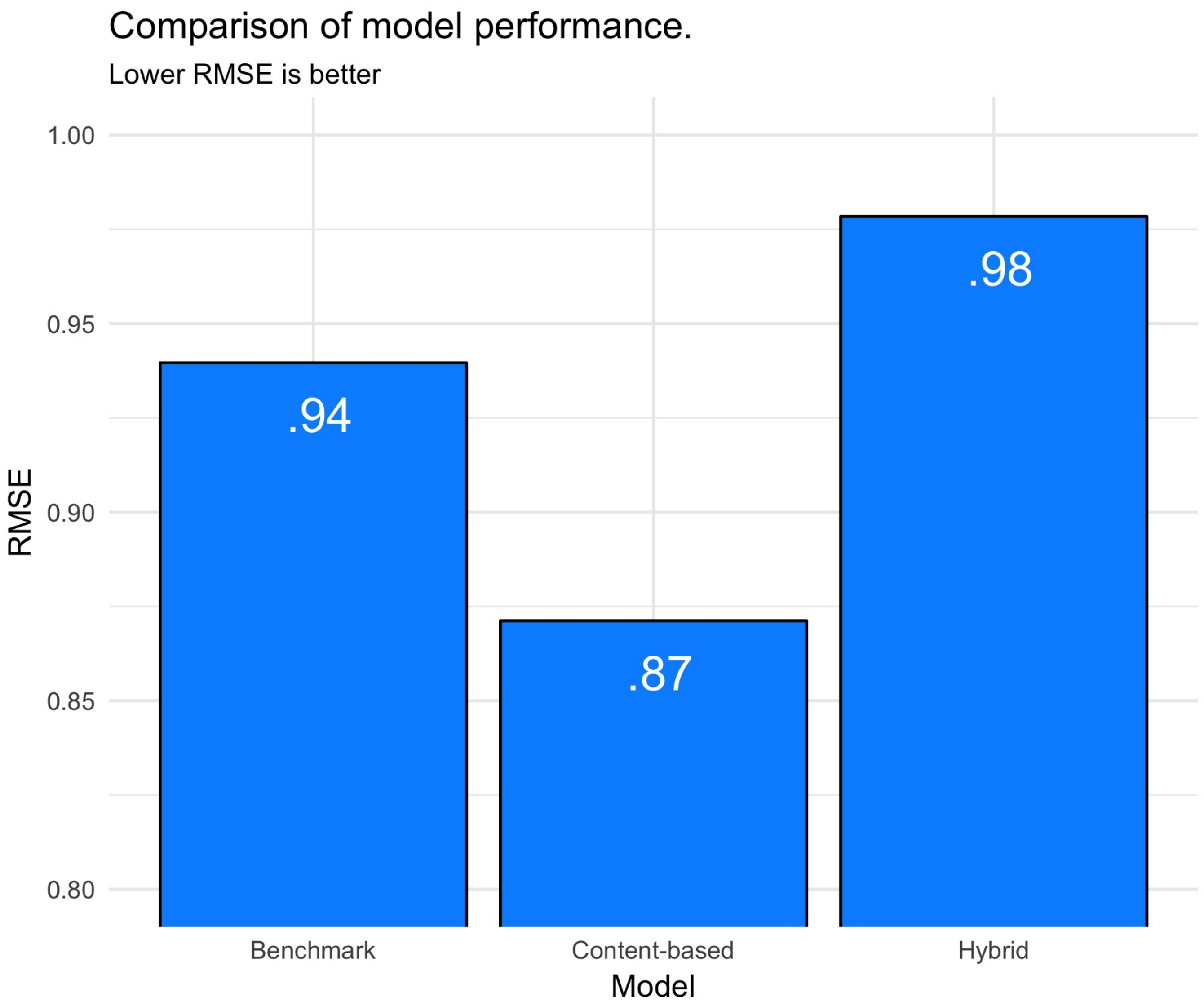
Justification

To recap, the RMSE of the benchmark model was 0.9396. The RMSE for the content-based filtering model (using 30-nearest neighbours) was 0.8712. This is a marked improvement over the benchmark model that is already fitting the data reasonably well, thus justifying its use as a method for predicting user ratings for new movies.

V. Conclusion

Free-Form Visualization

The Figure below summarises the performance of the three model solutions that were tested in this project.



It is easy to see from this Figure that the RMSE for the content-based model is markedly lower that the benchmark and hybrid models. That is, the model that best predicted user ratings for new movies involved computing the mean of ratings assigned to the 30 most-similar rated movies, with similarity being determined by cosine similarity on a specific set of movie features obtained from IMDB.

Reflection

This project presented an interesting and complex problem for me to solve. It required scraping, processing, and merging multiple data sources; and utilising, fitting, combining, testing and refining, multiple algorithms. The outcome of the many interim steps involved in this project is that a user’s prior movie ratings, and IMDB data for a new movie, can be used to predict a five-star rating of the new movie for that user.

For me, the discovery and use of non-negative matrix factorisation was most interesting. It was a novel technique to me, and I can see its broad utility, well beyond recommender systems.

Despite this useful technique, I found model refinement particularly difficult. It has been demonstrated empirically that collaborative filtering algorithms produce better rating estimates than content-based filtering. In this project, the final solution required content-based filtering, therefore making predictions particularly difficult.

I believe that the final solution is useful for making “best-guess” predictions of user ratings for new movies. Of course, the accuracy of those predictions will be far from perfect. Still, the solution provides a fast and simple approach for helping to personalise new movie recommendations that will likely improve customer experiences over and above basic benchmark approaches, until user rating data can be gathered.

Improvement

In this project, I opted to investigate a “best case” scenario, where users had each provided many ratings. This was done, in part, because the user database had to be reduced to make the computations possible on my local machine. For a real movie recommender system to be tested and implemented in services like Netflix, it would need to account for large numbers of users who may have very few ratings, possibly none.

Another avenue for improving the solution investigated here would be to source movie information from places other than IMDB. This project made use of a curated IMDB dataset. However, in reality, such data sets may not be available. It would be of value to implement general web scraping tools that could obtain movie information from multiple sources and support predictions.

---

1. Francesco Ricci and Lior Rokach and Bracha Shapira, Introduction to Recommender Systems Handbook, Recommender Systems Handbook, Springer, 2011, pp. 1-35 ↩

2. Prem Melville and Vikas Sindhwani, Recommender Systems, Encyclopedia of Machine Learning, 2010. ↩

3. R. J. Mooney & L. Roy (1999). Content-based book recommendation using learning for text categorization. In Workshop Recom. Sys.: Algo. and Evaluation. ↩