Project: Movie Recommendation Engine

Karsten Poddig

June 20, 2020

Abstract

The project described in this report is a Movie Recommendation Engine

Github Repo:
`https://github.com/KarstenPoddig/django_movie_project`

Keywords: Data Science, Data Engineering, Statistical Dashboard, Analytics, Movie Recommendation Engine, kNN-Regression, Hierarchical Clustering, Python, Django, Docker, Kubernetes, Continious Integration, Continious Development, Google Cloud Platform, SQL, PostgreSQL, JavaScript, Git, GitHub

Contents

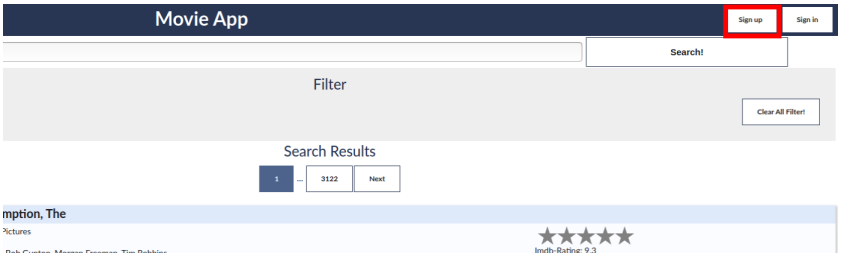

Figure 2.1: **Signing up**

1 Introduction

2 App functionalities

This section provides a quick overview of the functionalities of this app. The following functionalities can be accessed without an account:

- Search Movies
- Search for similar movies for a specific movie

The main functionalities require an account. These functionalities include

- Rate Movies
- Get Suggestions based on your ratings

Since the main purpose of this project is to provide movie suggestions I recommend to sign in if you want to explore this app. You can do this by creating an account and create your own ratings. But rating a sufficient number of movies can be time consuming, so if you want to go on fast and simply want to explore what how this app works you can simply use the following account with exisiting ratings:

username	testuser
password	password

The following subsections guide you through the app. But since the app should be somehow intuitive you can also skip these subsections.

2.1 Sign up / Sign In

2.1.1 Sign Up

Click on the button in the right top corner and enter all necessary data (figure 2.1).

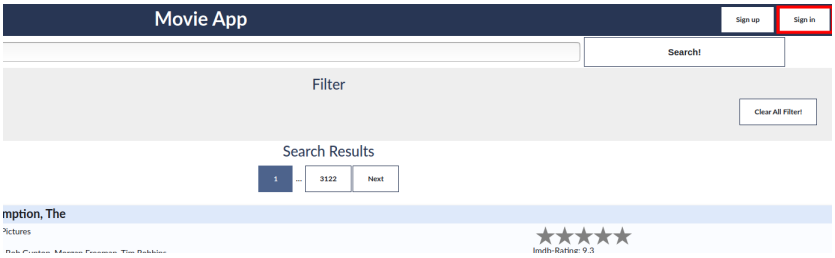

Figure 2.2: **Signing in**

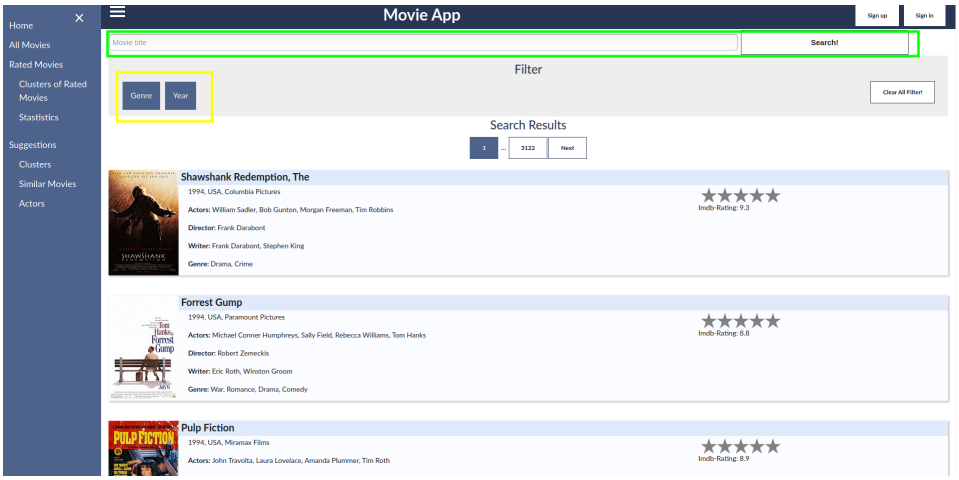

Figure 2.3: **page 'All Movies'**

2.1.2 Sign In

Click on the button in the right top corner and all necessary data (figure 2.2). Also you are redirected to the Sign In - page if you are accessing areas which require a login (like Rated Movies, etc.)

2.2 Search Movies

Click on the link 'All Movies' in the left bar. You will see a list of all movies in the database with the regarding details. To search a movie you can enter parts of title (green area) and optionally set certain filter critria (yellow area). The results are ordered decreasingly by popularity (total number of ratings) (see figure 2.3).

For example a search for movies with the term 'Bad' in the title with the Genre Action in the 2000s will look like figure 2.4

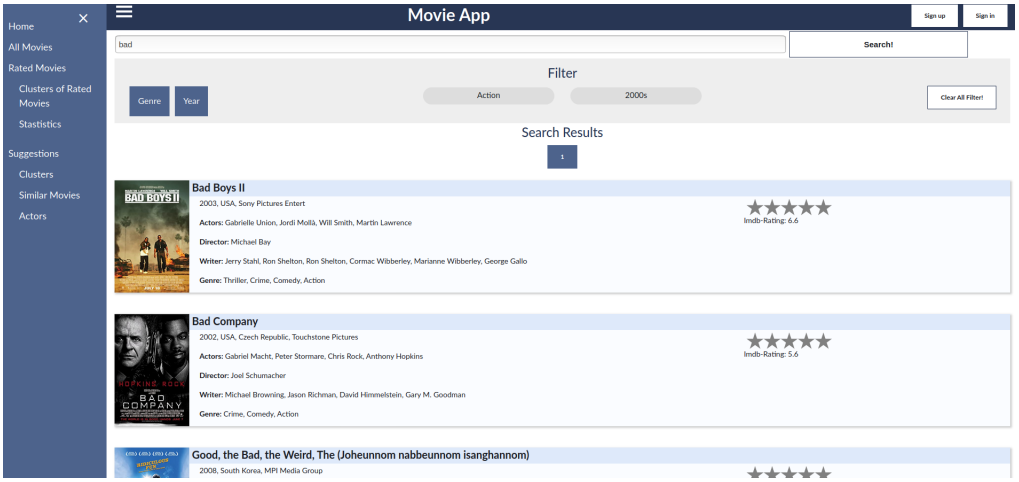

Figure 2.4: **Results for the term 'Bad' and the restrictions to 'Action', '2000s'**

Figure 2.5: **Rate a Movie**

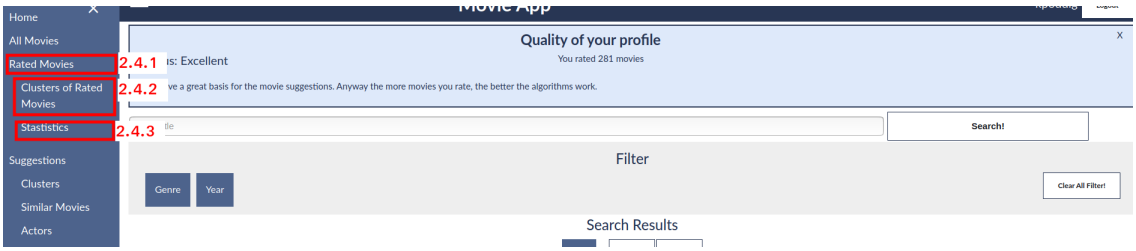

Figure 2.6: **Navigation in Rated Movies**

2.3 Rate Movies

Once you are logged in you can rate movies. Just enter the page 'All Movies', search the movie you want to rate and click on the regarding rating (red area in figure 2.5). If you choose a rating of 0 stars an existing rating will be deleted.

2.4 View Rated Movies

At some point you maybe want to take a look at the movies you already rated. The links for the sections 2.4.1-2.4.3 are displayed in figure 2.6.

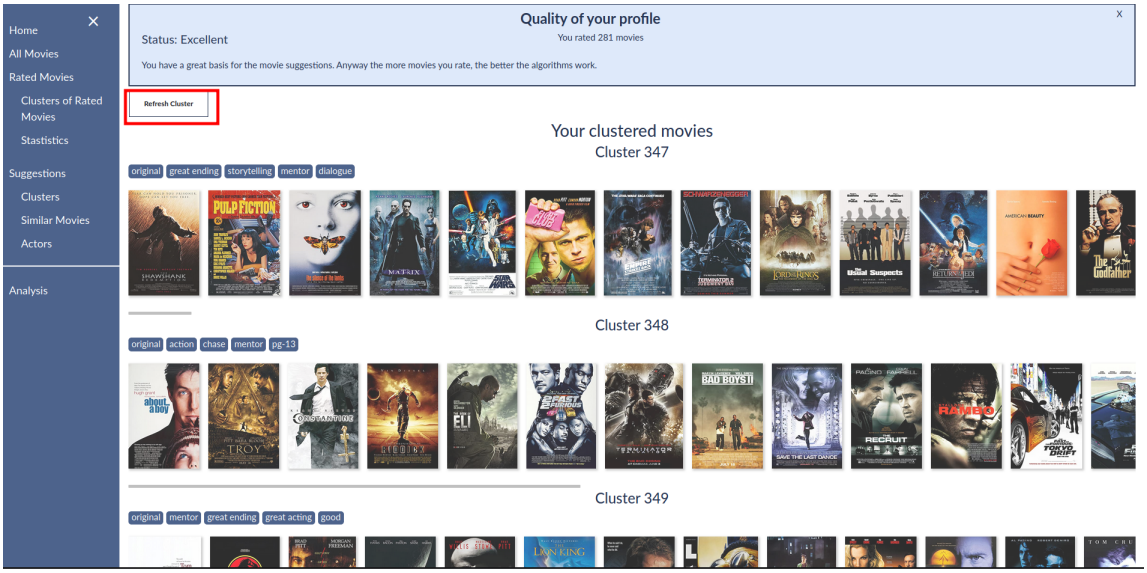

Figure 2.7: **Clusters of Rated Movies**

2.4.1 Movies with Details

This is basically the same page as 'All Movies' (section 2.2) with the exception that just rated movies are listed here. You can search and navigate through your rated movies in the same way as in section 2.2.

2.4.2 Clustered Movies

To determine better suggestions your rated movies are clustered. Take a look at section. Basically each cluster (each line) contains movies which are similar to each other. Factors which imply a similarity could be the genre, the production year, the atmosphere, actors, etc. For a better understanding what similarity in this context means I again refer to Section .... . If you want to update the clustering hit the button 'Refresh Cluster' (marked red in figure 2.7).

2.4.3 Statistics

This page provides a statistical dashboard for your ratings (see figure 2.8).

2.5 View Suggestions

Now it comes to the core functionality of the app. The following picture displays the links to the sections 2.5.1 - 2.5.3.

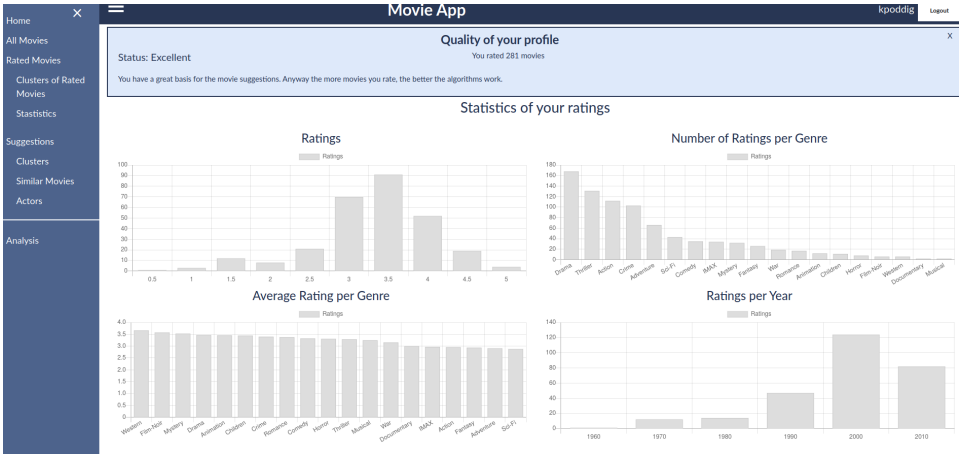

Figure 2.8: **Statistics of Rated Movies**

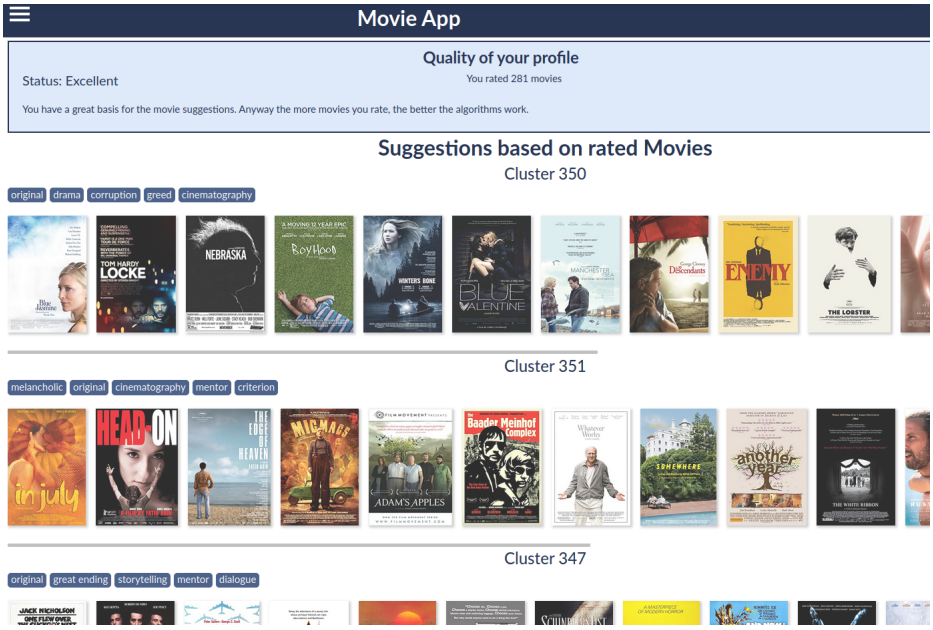

Figure 2.9: **Movie Suggestions based on Clusters of Rated Movies**

2.5.1 Suggestions based on clusters

This page shows you suggestions based on your rated movies. More specifically each row contains suggestions based on a cluster of your rated movies (see section 2.4.2). The algorithm to determine this suggestions is explained in the Data Science Part in section 3 or more specifically in subsection 3.5.

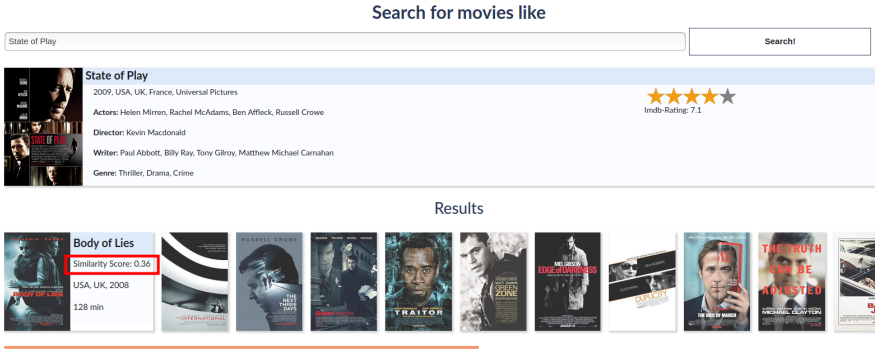

Figure 2.10: **Similar Movies to 'State of Play'**

2.5.2 Search Similar Movies

Let's say you liked a certain movie and want to find similar movies. This page does exactly this. Just enter the title of the movie in the search area. Enter the title, click search and wait a few seconds. The movie you searched is displayed as in section 2.2 (All Movies) and 2.3 (Rated Movies). The list of similar movies is shown in a row. Each movie shown in this row is displayed in a more compact way. By clicking on the wallpaper some basic information are displayed (see the following figure 2.10). This inherits the similarity to the movie you searched (marked with a red frame).

2.5.3 Suggestions based on Actors

This page creates suggestions based on the actors which you liked best in average. There are five rows with suggestions, each with movies from one actor which you didn't rate (figure 2.11).

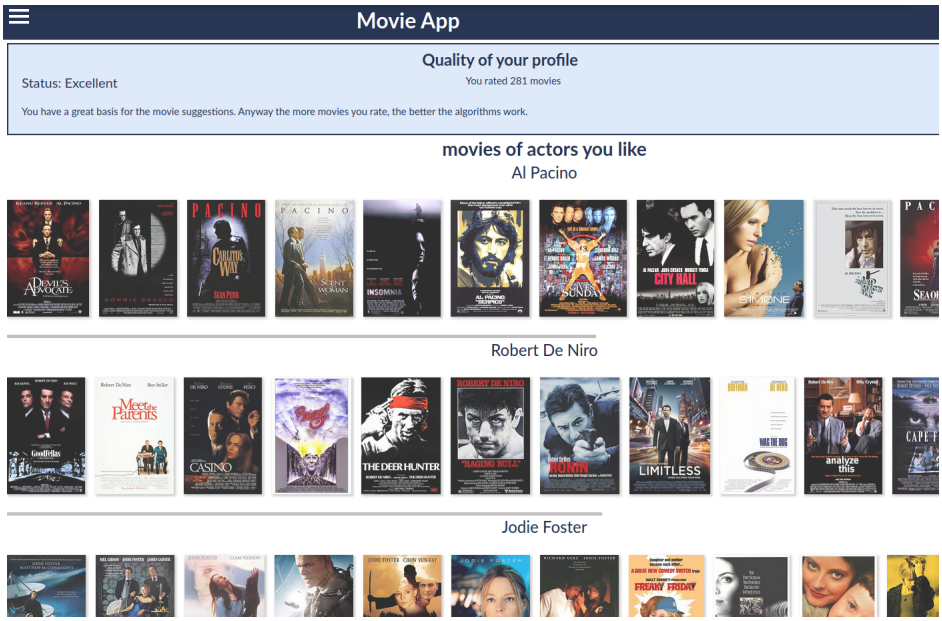

Figure 2.11: **Suggestions based on Actors**

3 Algorithms (Data Science Part)

In this section the algorithm on which the movie recommendation engine are explained.

3.1 Notation

Let $U = \{u_i\}_{1 \leq i \leq m}$ be the set of all users. The set of all movies is denoted with $M = \{m_j\}_{1 \leq j \leq n}$. The set R contains all tuples of users and movies for which a rating exists, i.e.

$$R = \{(u, m) \mid \text{if user } u \text{ rated movie } m\} \subseteq U \times M.$$

If user u rates the movie m (and therefore $(u, m) \in R$) the rating is denoted by r_{u_i, m_j}. The set off all ratings is then $\{r_{u,m}\}_{(u,m) \in R}$.

$$r \quad : \quad R \to \{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$$

3.2 Similarity and Distance

The most important part to determine movie suggestions is the concept of similarity of movies. As a similarity measure we use the cosine similarity.

Short Excourse: Cosine Similarity

Let $x, y \in \mathbb{R}^n$. Then the cosine similarity of these vectors is defined as

$$\text{similarity}(x, y) := \frac{\langle x, y \rangle}{\|x\| \|y\|} = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n x_i^2}},$$

where $\langle x, y \rangle$ is the scalar product (or inner product) of the two vectors and $\|x\|$ is the euclidian norm (l2 norm) of the vector x. The cosine similarity is in general a value between -1 and 1. The highest similarity is 1 and is achieved if and only if x and y there is a $\lambda > 0$ such that $y = \lambda x$.

We will apply the concept of the cosine similarity in the following way. Let $m, m' \in M$ be two arbitrary movies. The vector for each movie is the vector of ratings. If a user didn't rate a movie then the value in the vector is defined as zero. Written technically that means

$$\bar{r}_m = \begin{pmatrix} \bar{r}_{u_1, m} \\ .. \\ \bar{r}_{u_m, m} \end{pmatrix}, \text{ where } \bar{r}_{u_i, m} := \begin{cases} r_{u_i, m} \text{ if } (u_i, m) \in R \\ 0 \text{ else} \end{cases}.$$

So now we get our similarity of movie m, m' as

$$\text{similarity}(m, m') := \frac{\langle \bar{r}_m, \bar{r}_{m'} \rangle}{\|\bar{r}_m\| \|\bar{r}_{m'}\|}.$$

Notice: Since all values in the vectors which appear here are greater or equal zero the similarity measure is always between zero and one. This makes the ongoing computations more comfortable, since we don't have to deal with negative similarites. The more parallel the vectors $\bar{r}_m, \bar{r}_{m'}$ are,

the higher the is the similarity of the movies m and m'.
Speaking freely: If two movies in general are rated similar by the same users then the movies are considered as similar.
Example: In section 2.5.2 it is shown that we can search the most similar movies

3.3 Prediction of Rating

Now it comes to the prediction of ratings: Let's say there is a user $u \in U$ and some movie $m \in M$ which is not already rated by u, that means $m \notin M_u$ (as defined in section 3.1. M_u is the set of all movies which u did already rate).
The prediction $\hat{r}_{u,m}$ is simply the average value of the rating $(r_{u,m})_{m \in M}$ weighted by similarity. Of course the intuition of this approach is more similar movies to m should have a higher impact on the prediction.
Furthermore there are at most the 15 movies included. Using just a low number of movies for the prediction yields better results than using all movies. The number $k = 15$ is chosen because heuristically for this value the evaluation metrics has shown the smallest error for the predictions (see subsection 3.6).
This algorithm is known as the k Nearest Neighbour Regression (with $k = 15$).

Expressed in a more technical way: Let $u \in U$, $m \notin M_u$. Then

$$\hat{r}_{u,m} := \sum_{i=1}^{\min(15,|M_u|)} \left(\frac{\text{similarity}(m^{(i)}, m)}{\sum_{i=1}^{\min(15,|M_u|} \text{similarity}(m^{(i)}, m)} \right) r_{u,m^{(i)}},$$

where $(m^{(1)}, m^{(2)}, ...) \in M_u$ are the movies rated by u sorted by the similarity to m. That means $(\text{similarity}(m^{(i)}, m))_i$ is decreasing.

3.4 Clustering of Rated movies for a user

The next step is the clustering of rated movies. What is the purpose for this step? At first one could think to simply compute the rating predictions for all movies and then suggest the movies with the highest predicted values to the user. But in reality often people like different kind of movies. Depending on your mood, etc. you maybe want to watch a movie of a certain genre, topic, athmosphere and so on. Therefore the rated movies will be clustered and then for each cluster suggestions will be determined.
The clustering algorithm to compute these clusters is the AgglomerativeClustering provided by the package scikit-learn. This algorithm needs the information which distance should be used to cluster the objects. Here we will use the option 'precomputed' and provide a matrix which is based on the similarity matrix in section 3.2 $(\text{similarity}(m, m'))_{m,m' \in M}$. The number of clusters also needs to be set at the beginning.

Creation of distance matrix

The distance between the objects is often simply defined as the inverse of the similarity. But in this case there is a problem with this approach:
Let's take two movies. First we take m_1 is 'Independence Day'. The second movie m_2 is 'Tinker Tailor Soldier Spy'. Then we enter these movies in the similarity view (section 2.5.2) and check

Figure 3.1: **Similar Movies to 'Independece Day'**

Figure 3.2: **Similar Movies to 'Tinker Tailor Soldier Spy'**

the results in figures 3.1 and 3.2. As you can see the similarities for m_1 are much higher then for m_2. Anyway the similarity search works good for both movies. This is an example for the following relation: The similarities for popular movies are in general much higher then for less popular movies. To illustrate this relation in a more representive way figure 3.3 shows a scatter plot with number of ratings of each movie and the summed up similarities to all other movies. If one would simply take the inverse of the similarity as the distance the result is one big cluster including all blockbuster and some very small cluster with relatively unpopular movies.
Therefore here we take the following approach: Instead of focusing on the absolute similarity values the distance is determined by the position in the similarity list. That means to compute the distance between the movie m_1 and the movie m_2 we identify which position m_2 has in the ordered list of movies (ordered decreasingly by the similarity - $(\text{similarity}(m_1, m))_m$). So if m_2 is for example on the fifth position in the list of most similar movies to m_1 we have the value 5. To put more weight on the very low positions and less weight on very high positions the logarithm is applied to this value. Roughly speaking it should be important for the clustering if m_2 is the 1th

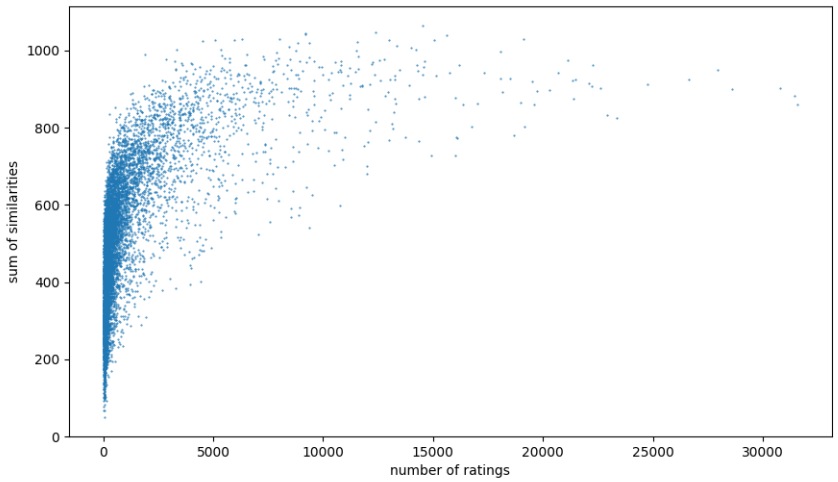

Figure 3.3: **Scatter plot for number of ratings per movie and summed up similarities to other movies**

or 10th most similar movie to m_1, but not if it is the 10001th or 10010th most similar movie. This is guaranteed by applying an increasing function with decreasing derivative. Other choices for this behaviour could be the function $x^{-\frac{1}{2}}$ or functions which are also used for the activation in neural networks (i.e. sigmoid, relu, softmax).

3.5 Final Predictions

So we now have the algorithm for the prediction of a rating and a certain cluster of movies of a user. Which movies should we suggest based on the movies in the cluster?
At first thought it sounds reasonable to compute predictions for all ratings based on the movies in this cluster and suggest the movies with the highest predicted values. But then there occurs the following situation: Let's say a user has a cluster with mostly Animation Movies from Disney, etc. Since ratings for all movies will be computed there will also be a prediction for the rating of - let's say 'Texas Chainsaw Massacre'. The similarities between the movies in the cluster and 'Texas Chainsaw Massacre'will be quite low. Nevertheless the predicted rating is basically a weighted average which means that just the relative and not the total height of similarities are important for the prediction. This still makes sense since a prediction is simply the best estimate based on the rated movies. Therefore if the movies in the cluster which are the most similar to 'Texas Chainsaw Massacre' have high ratings, then the prediction for the rating of 'Texas Chainsaw Massacre' will also be high, even if the similarity of this movie to the movies in the cluster is low.
This example shows that it is **insufficient to solely focus on the predicted rating**. Instead the similarity of a movie to a cluster should also be taken into account.
Therefore for a user $u \in U$ and a movie $m \notin M_u$ we compute a score based on two variables:

- Predicted Rating $\hat{r}_{u,m}$

- sum of similarities, i.e.

$$\sum_{i=1}^{\min(15,|M_u|)} \text{similarity}(m^{(i)}, m)$$

  (see section 3.3).

The score is a weighted sum of these two variables. Before the variables will be normalized such that both variables are in the range between 0 and 1 (0 is the worst value and 1 is the best value). In formulas for each movie $m \notin M_u$ the score is defined as

$$\text{score}(m) \quad := \quad \lambda \, \text{score}^{\text{rating}}(m) + (1 - \lambda) \, \text{score}^{\text{similarity}}(m),$$

where the parameter $\lambda \in [0, 1]$ weights the importance of the predicted rating to the similarity and

$$\begin{aligned} \text{score}^{\text{rating}}(m) &:= \frac{1}{5} \, \hat{r}_{u,m} \\ &= \frac{1}{5} \left[\sum_{i=1}^{\min(15,|M_u|)} \left(\frac{\text{similarity}(m^{(i)}, m)}{\sum_{i=1}^{\min(15,|M_u|} \text{similarity}(m^{(i)}, m)} \right) \, r_{u,m^{(i)}} \right] \\ \text{score}^{\text{similarity}}(m) &:= \frac{1}{\min(15, |M_u|)} \left[\sum_{i=1}^{\min(15,|M_u|)} \text{similarity}(m^{(i)}, m) \right]. \end{aligned}$$

The factors $\frac{1}{5}$, $\frac{1}{\min(15,|M_u|)}$ assure the scores to be in the interval $[0, 1]$. The choice of the parameter λ is 0.5 by default.
Finally we choose the movies with the highest score as suggestions.

3.6 Evaluation of Prediction Algorithm

3.7 Discussion

4 Source of Data (Data Engineering Part)

4.1 Source of Data

MovieLen, Imdb-API Own ratings

4.2 Google Cloud SQL

5 Structure of the deployment

This section covers the basic structure of the deployment of this app. That means we will explain how the frameworks and technologies interact (like Django, Google Compute Engine, Google Cloud SQL, Kubernetes, Jenkins, etc.). This section covers not the documentation of the code (for example the code for the backend in python or the code for the frontend in html, javascript). For this purpose you can take a look at the README-file and the other files in the Github repository.