1. **Movie Recommendation Systems**

We now turn to the core of our project. In this section we apply various techniques for making movie recommendations. First, we look at simple **demographic filtering** approaches which utilize demographic user characteristics such as age or gender and do not involve any machine learning. Instead, those techniques make general recommendations to all users sharing certain attributes. Afterwards, we consider **content-based filtering** techniques which try to find movies similar to those a given user has enjoyed and propose them to the user. Moreover, we implement **collaborative filtering** methods. Rather than identifying similar films, the goal here is to group users based on their interests. Lastly, we apply a **logistic regression** and a **decision tree algorithm** to the dataset and make recommendations to individual users.

* 1. **Demographic Filtering**

A simple approach for making recommendations is to suggest the best rated or most popular (“trending”) movies in the dataset (Figure 1). While this approach is associated with low computation cost and effort, it completely ignores differences between users. Hence, this method is only appropriate for making suggestions to new users with unknown characteristics. However, there are obvious drawbacks associated with this strategy e.g. adult films may be suggested to children or French movies to English speakers.



Figure 1: Best rated and most popular movies in the dataset

A better tactic would therefore be to ask users for their age, gender, education, spoken languages etc. and tailor the recommendation to their demographic group. This way, differences in terms of demographics could be captured, however, differences between users within a demographic group would still be ignored. Unfortunately, the considered data set does not contain any demographic information on the respective users. Thus, demographic filtering beyond suggesting trending or best rated movies is not possible.

* 1. **Content-Based Filtering**

A more sophisticated approach than demographic filtering is to group movies based on features like genre, budget, cast, director, runtime or average rating. Movies similar to those a given user has liked in the past can then be identified and recommended to the user. Such content-based filtering methods include k-NN and plot-based recommenders which we will now consider in more detail.

* + 1. **k-NN Algorithm**

First, we develop a k-NN based recommendation system. For a given movie, the idea is to find the k most similar movies (“nearest neighbors”) in terms of four features: cast, director, genre and keywords. Except for the director, this information is stored in lists that contain multiple elements (see Figure 2) and we first need to convert the data into a useable format by using the unlist function in R.



Figure 2: Glance at the features of interest in the movies dataset

In the following, we calculate a similarity score between a given movie of interest which we will call our “object movie” and all other movies in the dataset which we denominate as “target movies”. For each target movie, the similarity score is calculated as follows:

where

* j ε [cast, director, genres, keywords] are the four different movie features considered
* is the number of realizations for feature j of the object movie e.g. for j = genre, the realizations for the object “Minions” would be “Family”, “Animation” and “Adventure”, thus a = 3
* is the number of realizations for feature j of the target movie e.g. for j = director, the realizations for the target “Avatar” would be “James Cameron”, thus b = 1
* is the number elements that the object and the target movie have in common e.g. for j = genre, the shared element of the object “Minions” and the target “Avatar” would be “Adventure” i.e. n = 1
* is the arbitrary weight of feature j

As an example, let us identify the 10 most similar items for “Terminator 3: Rise of the Machines”, the most rated movie in the data set, by assigning each feature an equal weight of 1. What becomes apparent when looking at the resulting recommendations, is that all movies in the dataset which were directed by Jonathan Mostow come out at the top of the list (Figure 4). This is due to the fact that the director feature consists of a single element, whereas the other features typically consist of three elements. Thus, the magnitude of the similarity score for this component is larger.

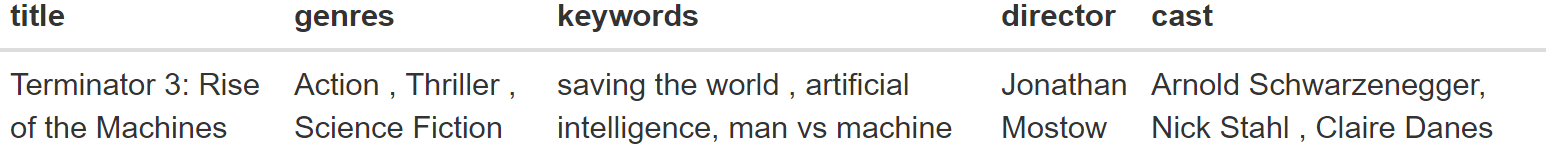


Figure 3: Movie features of Terminator 3 (2003)

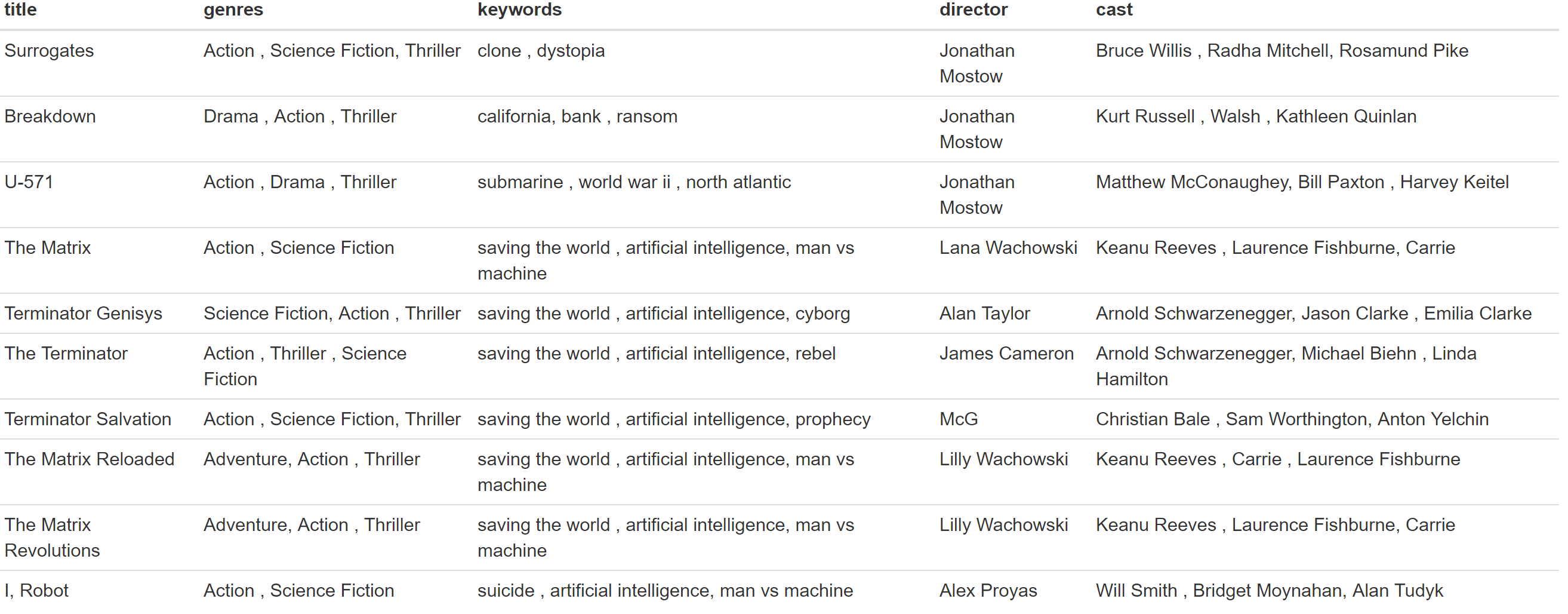


Figure 4: Movie recommendations for viewers of Terminator 3 based on k-NN with equal weights

We therefore adjust the recommendation system by decreasing the weight of the director feature in our calculation to 1/3, while keeping the weights for the other features constant. This way, the calculation of the similarity score is more balanced between the different features. While the new recommendation list still contains several movies by Jonathan Mostow, director is no longer the dominant criterion (Figure 5).

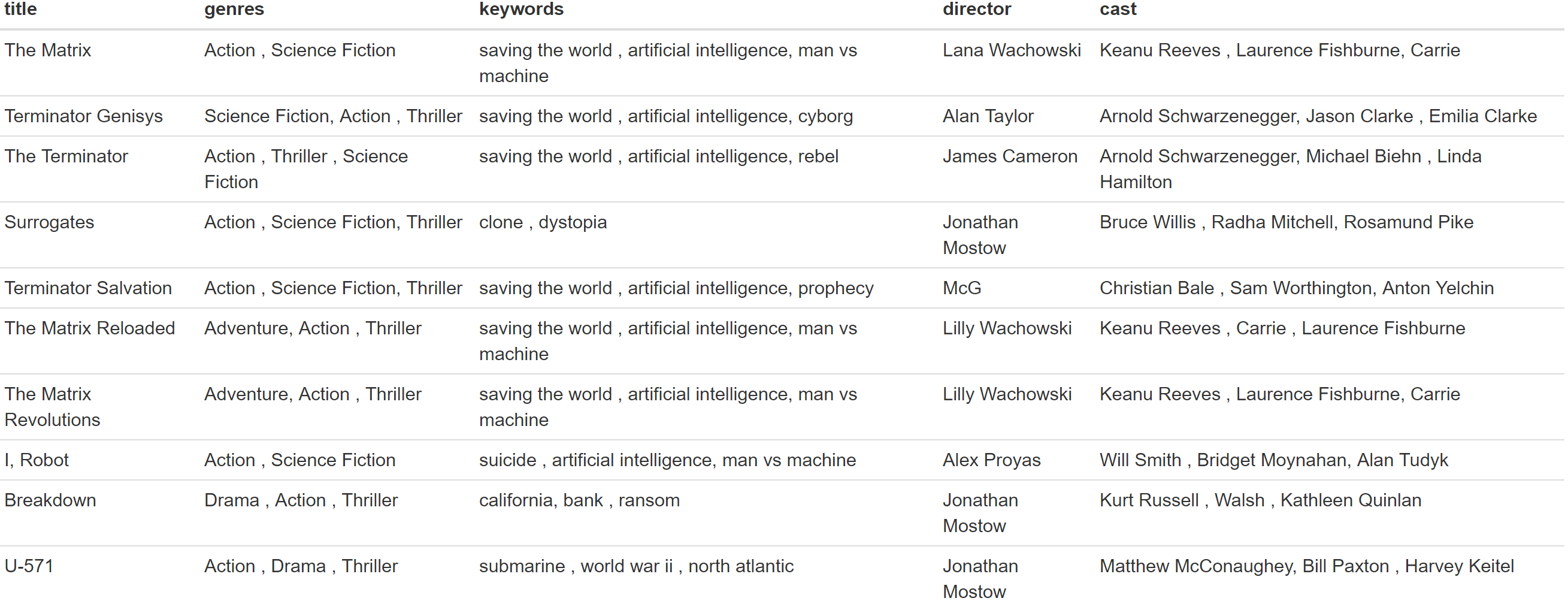


Figure 5: Movie recommendations for viewers of Interstellar based on k-NN with adjusted weights

The recommendations obtained under k-NN intuitively make sense. Furthermore, they are similar to the recommendations made by Amazon Prime (Figure 6). In order to test how well our recommendation system performs, we also look at the ratings that users who have watched and enjoyed Terminator 3 (i.e. assigned it a rating of at least 3.5) have given to the recommended target movies. The test results confirm our initial assessment, as all recommended films receive good average ratings between 2.9 and 4.2 (Figure 7).

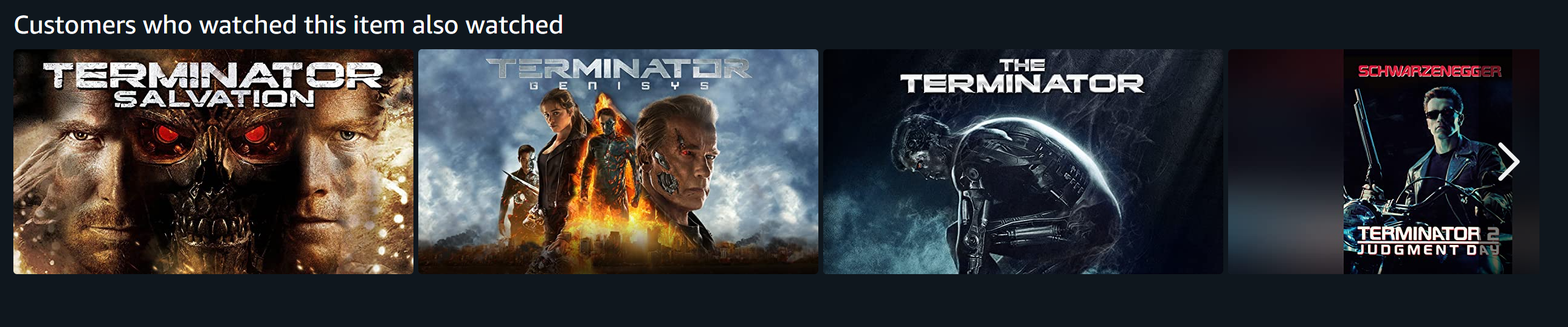


Figure 6: Amazon Prime recommendations for viewers of Terminator 3

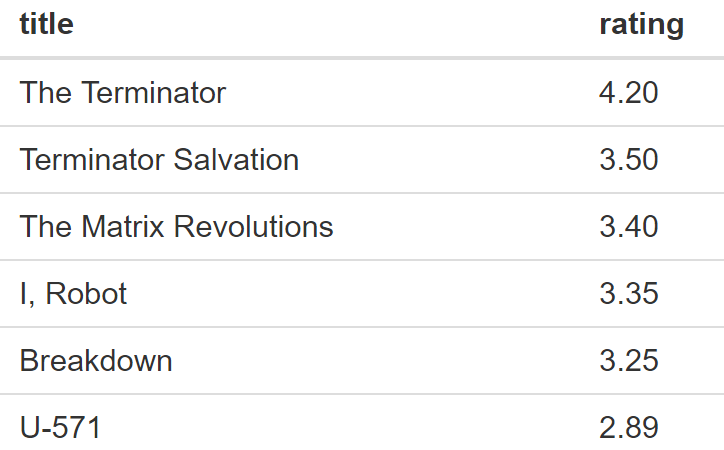


Figure 7: Average ratings for recommended movies under kNN

We conclude that the k-NN algorithm works reasonably well and could certainly be applied to users who have searched for or liked a particular movie. However, a major downside of this algorithm is that it has to be recomputed for every movie and movie recommendations under this setting are thus always based on a single item, meaning that no learning happens. In fact, we are really just screening the dataset for movies similar to the object movie. Moreover, this approach might recommend movies which a user has already watched and does not take individual user preferences into consideration but makes the same recommendation to all users.

* + 1. **Plot-based Recommender**

This movie recommendation strategy is very similar to the k-NN recommendation system. For our plot-description-based recommender, we also calculate pairwise similarity scores for all movies. Those scores are now based on the terms used in the movie descriptions. Term Frequency-Inverse Document Frequency (TF-IDF) vectors - a standard tool from text processing – allow us to convert movie descriptions into a useable format (a matrix of all movies as columns and words used in the descriptions as rows). In her tutorial Ibtesam Ahmed defines TF-IDF as follows:

*“[…] term frequency is the relative frequency of a word in a document given as (term instances/total instances). Inverse Document Frequency is the relative count of documents containing the term given by log(number of documents/documents including the term). The overall importance of each word to the documents in which they appear is equal to TF \* IDF.”*

The similarity scores for any pair of movies are then calculated by applying cosine similarity:

where:

* and are components of the TF-IDF vectors A and B
* similarity ε [0,1]

Once again, we identify the 10 most similar movies for “Terminator 3: Rise of the Machines”. As you can see, the suggestions are quite similar to those obtained under k-NN and include several movies from the Terminator series (Figure 8).



Figure 8: Movie recommendations by the plot-based recommender

Let us have a glance at the movie plots of the top 3 recommendations (which unsurprisingly are all from the Terminator series) and compare them to the description of Terminator 3 (Figure 9). Apart from the title “Terminator” and the main characters John and Sarah Connor, the movie plots share several characteristic terms like “Skynet” and “back” (as part of Arnold Schwarzenegger’s legendary “I’ll be back” catchphrase).

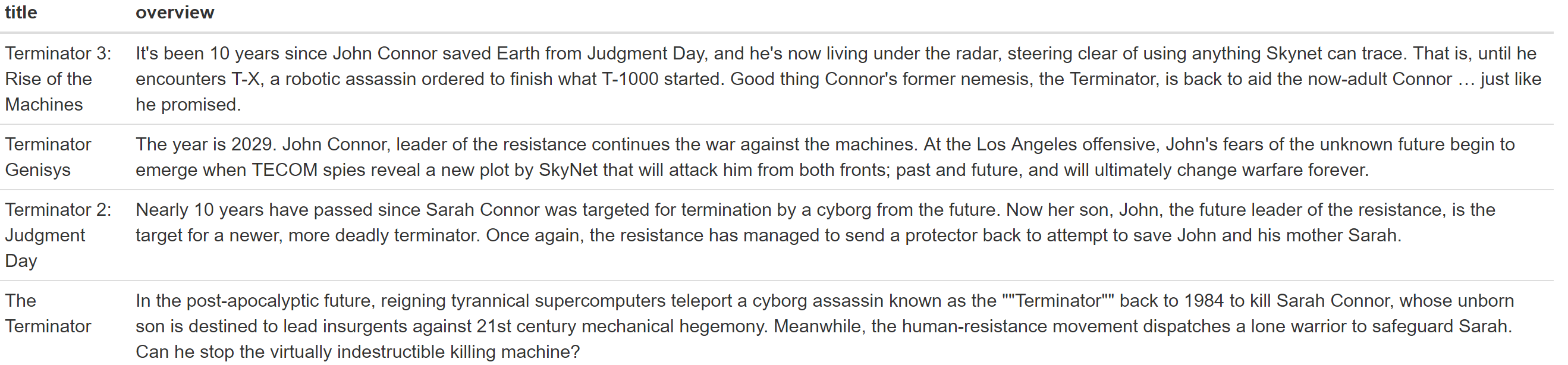


Figure 9: Comparison of Terminator movie plots

Looking at the average rating that users who have enjoyed Terminator 3 have given to the recommended movies, we find that the reviews are very positive (Figure 10). However, ratings from users who have liked Terminator 3 are available only for 3 out of the 10 suggested movies.

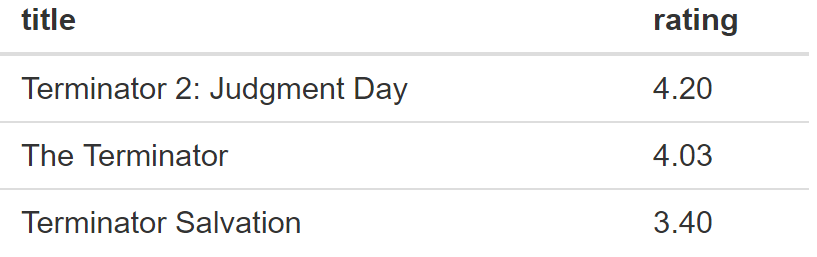


Figure 10: Average ratings for recommended movies under kNN

Thus, like the k-NN algorithm, the plot-based strategy seems to lead to good results. In fact, the outputs are almost identical. But the approach is also subject to the same disadvantages, namely that recommendations have to be computed separately for each movie and that no learning happens. Instead, this approach is basically screening the data set for similar movies, thereby completely ignoring individual user preferences with respect to genres, actors etc. and making the same recommendations to all users.

* 1. **Collaborative Filtering**

In order to overcome those issues, we now implement collaborative filtering techniques which can be divided into user- and item-based filtering methods. The former method groups similar users with the help of Pearson correlation or cosine similarity and proposes movies based on what other users in the same category have liked. In contrast, item-based filtering methods take positive reviews by a given user and try to find similar movies. Like the user-based approach item-based filtering uses Pearson correlation or cosine similarity to derive the similarity between two items. Both techniques require detailed information on the user for whom the recommendation is made and cannot be applied to new users. In this paper we will focus on item-based collaborative filtering.

In a first step we construct a matrix of all users (rows) and movies (columns) in the dataset which we fill with the ratings assigned by the users to the respective movies were possible. To the remaining movie-user combinations i.e. the movies that have not been watched by an individual user yet, we assign the average movie rating. Using a latent factor model via the built-in svd function in R allows us to address sparsity issues (i.e. the occurrence of extreme values due to a small number of observations) as well as scalability issues (i.e. high computation times in large datasets) by decreasing the number of dimensions. By applying a latent feature model, we map users and movies into a latent feature space. In this feature space we only use the first three dimensions, as they typically contain almost all of the relevant information. We now have an optimization problem where the goal is to minimize RMSE, the average prediction error, in terms of the rating which a user will assign to a given movie.

We apply the recommendation system to the most active user in the dataset with ID 564 and get the following recommendations (Figure 11):

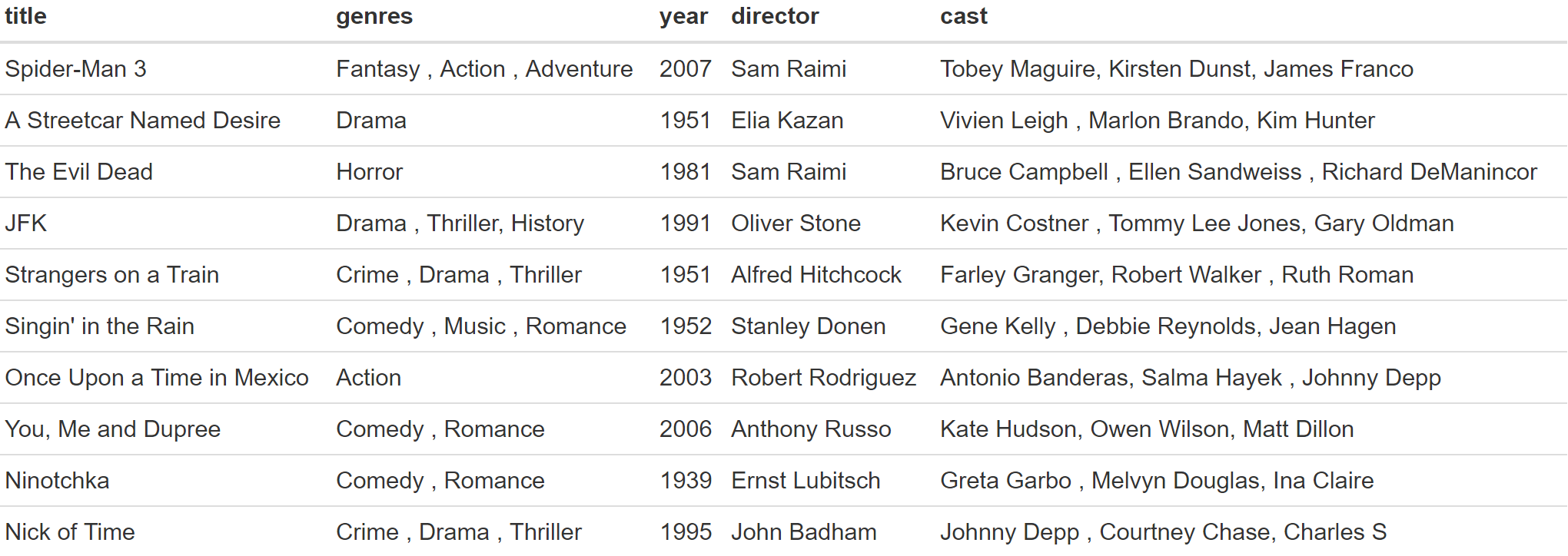


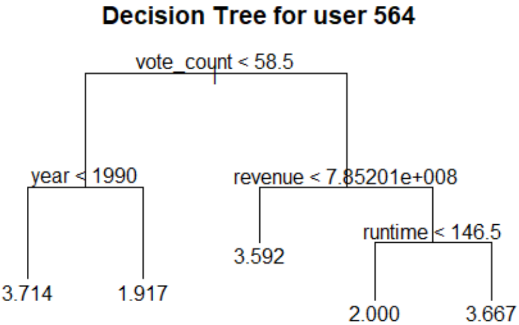
Figure 11: Recommendations based on collaborative filtering

Based on the recommendations, it seems that the user has a preference for old movies and does not have a preference for a particular genre.

Different than the content-based filtering approaches in the previous sections this recommender is learning constantly and recommendations should improve over time. However, this also means that the system will perform poorly for relatively new users. Moreover, the system is not fully transparent in how it makes recommendations.

* 1. **Decision Tree Algorithm**

Decision tree algorithms repeatedly split the feature space (in our case the set of all movies in the dataset) into two different spaces. The splitting criterion at each node is chosen in such a way that the overall loss function RSS is minimized. We build a decision tree for the most active user with user-Id 564 (Figure 12). In order to construct the tree, we use all 463 observed ratings by this particular user on movies he/she has already watched and use all numeric movie features in the dataset as decision criteria. For each resulting subset of similar movies, the algorithm thus predicts the mean rating which the user will assign to movies from this group.

  
Figure 12: Decision Tree for user 564

Overall, four criteria are used to form the tree (vote count, year, revenue and runtime). Interestingly, popularity and weighted rating are not used as factors, indicating that public opinion does not have a strong impact on what this user likes. The decision tree shows that the user on average assigns the highest rating to older movies that were released before 1990 and have received less than 59 votes. We use the decision tree to predict the rating the user would give to those movies in the dataset he/she has not watched yet and recommend the movies with the highest predicted rating. One might also say that we search for movies with the two characteristics described above which the user seems to like the best. From the total set of 4497 unwatched movies we obtain a list of 255 movies for which we predict a rating of 3.714 (Figure 13).

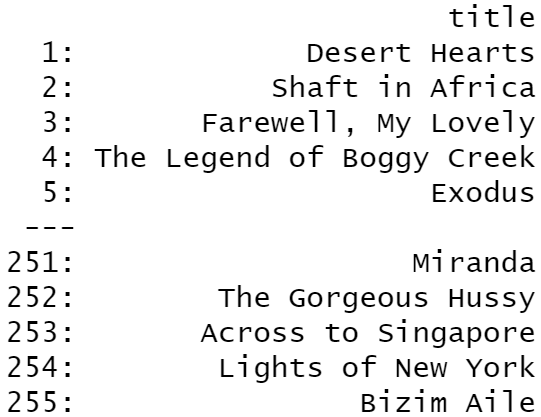


Figure 13: Decision Tree Recommendation

But how accurate are those recommendations? In order to answer this question, we randomly split the observed ratings by the selected user into a training (70%) and a testing (30%) data set and recompute our tree. Note that the resulting decision tree differs dramatically from the previous version (Figure 14). One of the major downsides of using decision trees is that they are extremely sensitive to changes in the training data and may look entirely different even for only slightly modified training data sets. Thus, we cannot make inferences as to whether our original tree makes good suggestions.

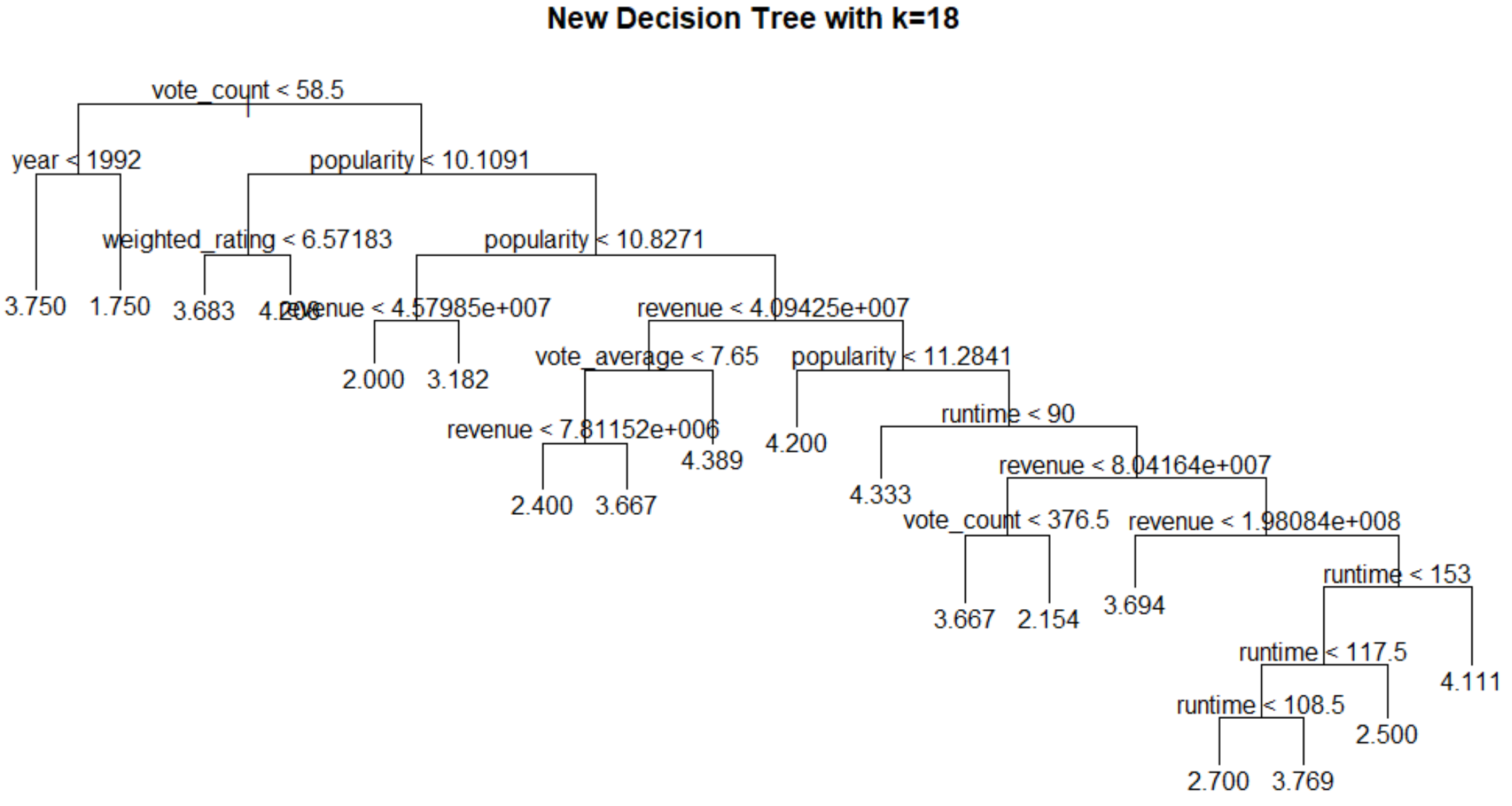


Figure 14: New decision tree w/o pruning

However, we can investigate whether decision trees in general are a good tool for making recommendations. The root-mean-square error (RMSE) for the testing data i.e. the average absolute prediction error based on the new decision tree amounts to 1.41. Given a standard error of the ratings in the test data set of only 1.14, the model performs poorly and one could more accurately predict the ratings in the test data set by simply always guessing the mean rating the user has assigned to movies from the training data set.

With 18 terminal nodes and 7 used features the new tree seems to be overfitting the training data. Therefore, we prune the tree to find the number of terminal nodes which minimizes the root-mean-squared error for the testing data. We find that a decision tree with three terminal nodes is optimal (Figure 15). However, with an RMSE of 1.13 the strategy performs only slightly better than the strategy of always guessing the mean rating observed on the training data set.

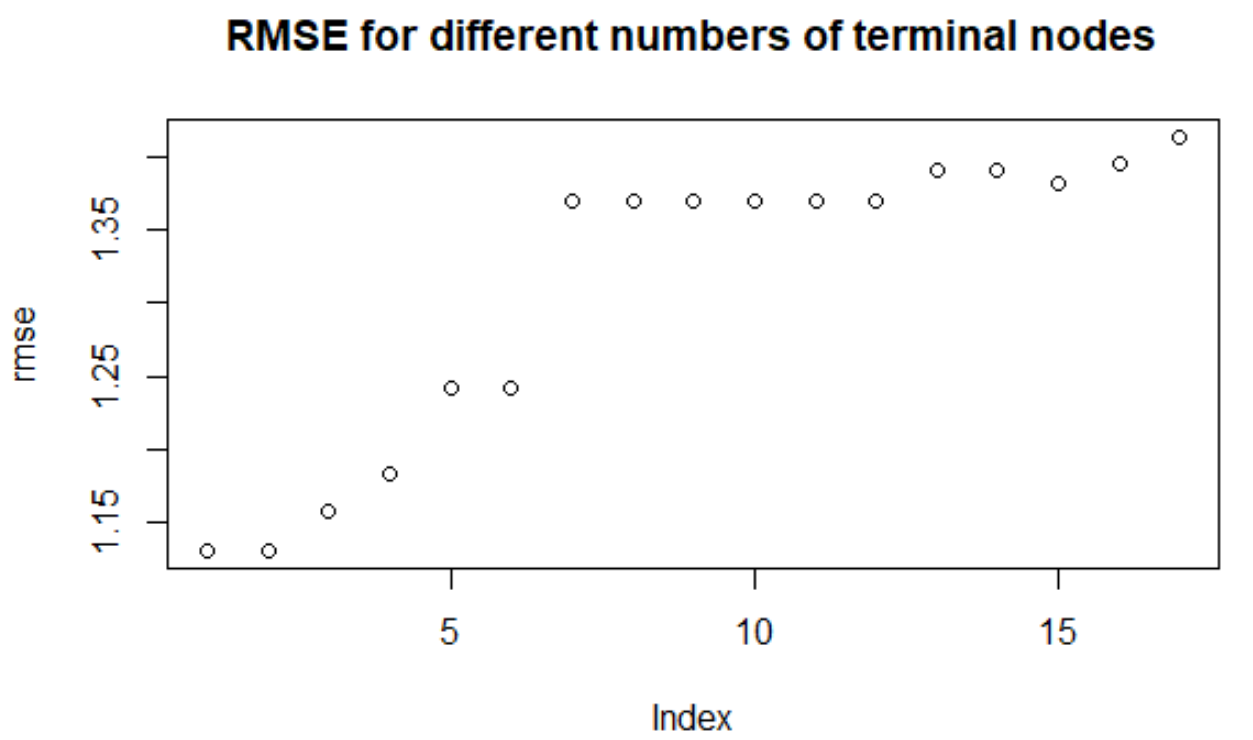


Figure 15: RMSE for different numbers of terminal nodes

We conclude that decision trees are very intuitive and easy to compute. Also, they take into account user preferences with respect to movie characteristics. However, they appear to be overly simplistic and lead to inaccurate predictions. Moreover, decision trees may overfit the training data and are extremely sensitive to changes in the training dataset.

* 1. **Logistic Regression**

Another approach for making movie recommendations is to use a logistic regression and predict the likelihood with which a given user will like a movie based on its features. Once again, we consider the most active user with ID 564 and train our regression model using all 463 reviews by this user. Since logistic regressions can only work with binary dependent variables, we introduce a like variable which equals 1, for perfect ratings (i.e. 5 out of 5) and 0 otherwise. As our independent variables we use the numeric movie features: budget, popularity, revenue, runtime, vote average, vote count and year. The regression summary statistics show that only revenue and vote count are statistically significant predictors of whether a user will like a movie at the 10% significance level (Figure 16). Interestingly, revenue has a negative sign, indicating that the selected user does not have a preference for blockbusters. Moreover, the negative coefficient for “year” – even though it is not statistically significant - confirms our finding from the decision tree recommendation system that the user seems to like older movies. In order to eliminate the effect of multicollinearity, we also perform a second regression where we drop budget and vote count which - as we have shown above - are highly correlated with revenue. We observe that the statistical significance of the revenue variable disappears. However, the negative signs for revenue and popularity persist.

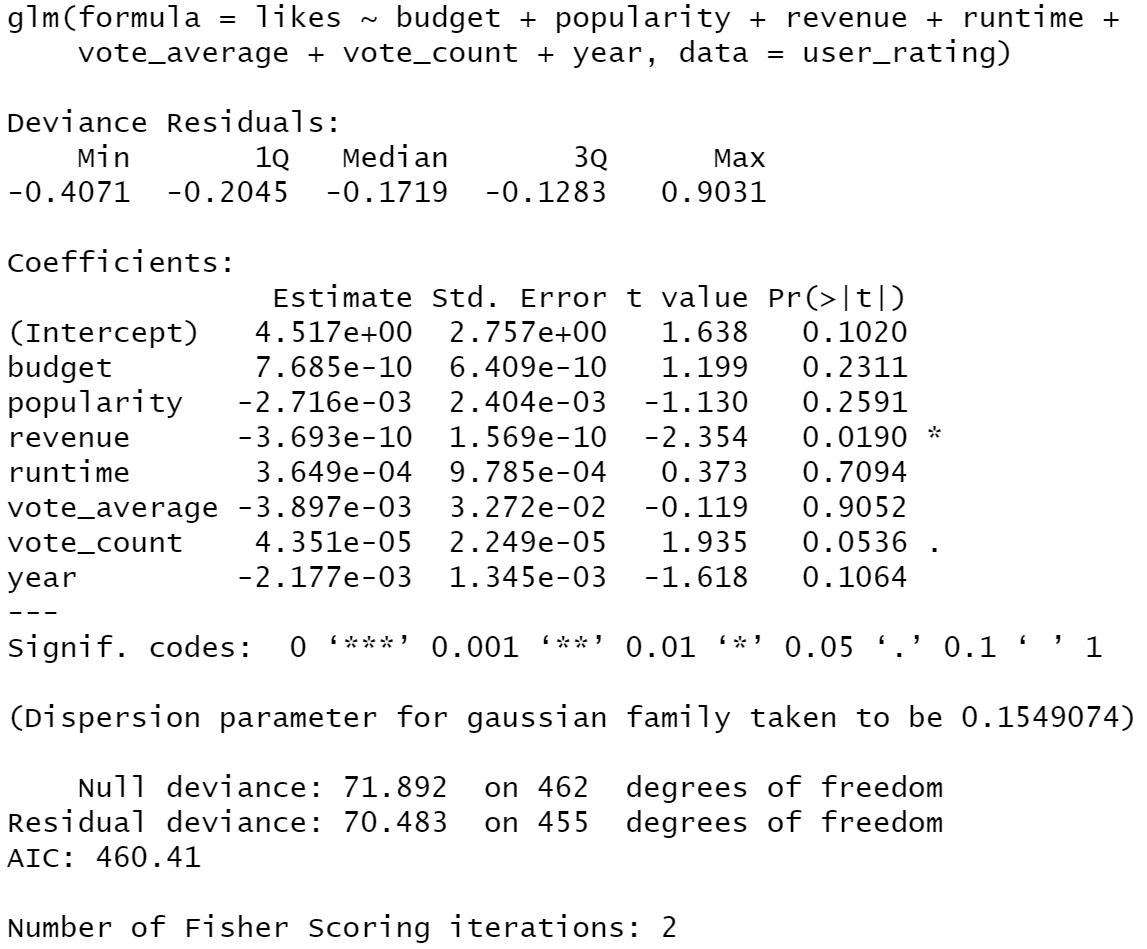
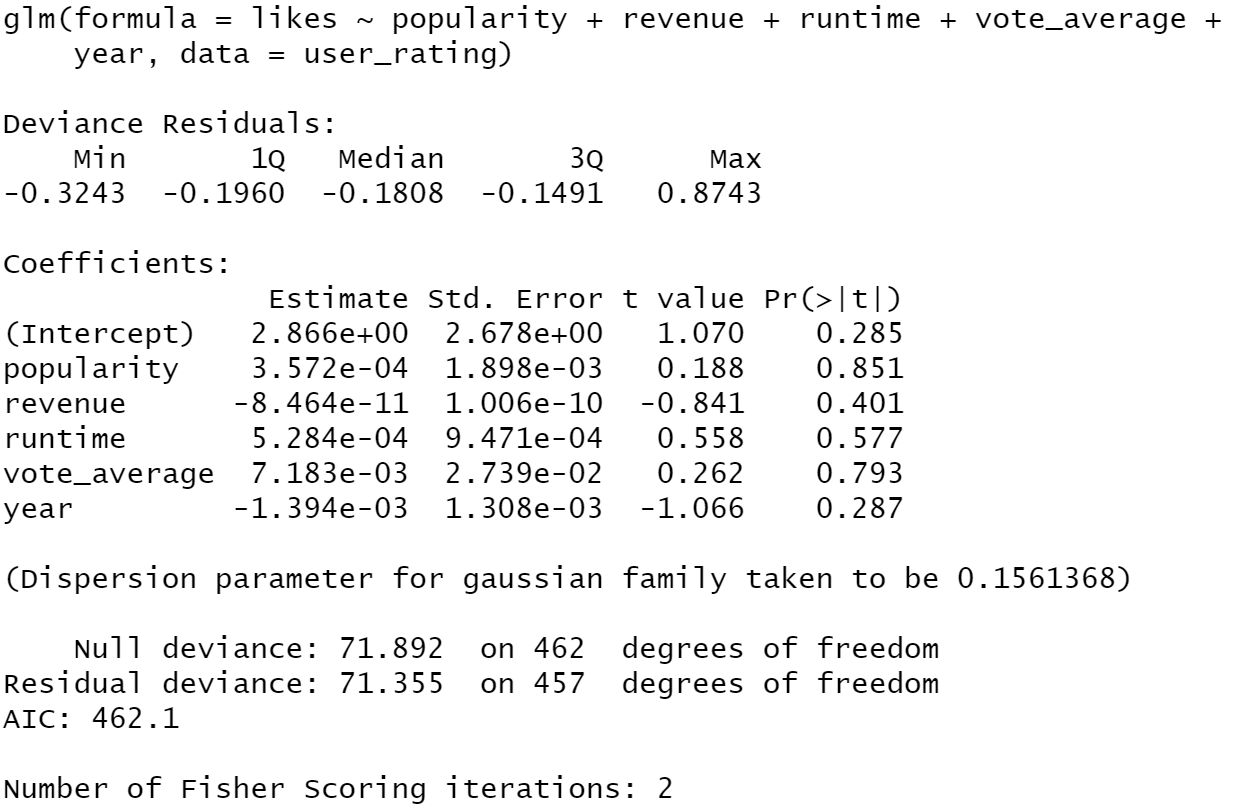
 

Figure 16: Logistic regression summary statistics for all numeric features (left) and when accounting for multicollinearity (right)

Using the coefficients obtained in the regressions above, we predict the “like” probabilities for all unwatched movies in the dataset. Our recommendation system makes movie suggestions by identifying the movies with the highest like probabilities. The recommendations based on the two regressions differ substantially. Only “The birth of a nation” from 1915 is proposed under both models. Notably, the second model only proposes older movies from before 1962 and the predicted probabilities of a perfect rating are much lower than for the first model (Figures 17, 18).



Figure 17: Movie recommendation based on a logistic regression using all numeric features



Figure 18: Movie recommendation based on a logistic regression accounting for multicollinearity

In order to test the predictive power of our logistic regression model, we again randomly split the reviews by the most active user into a training (70%) and a testing data set (30%). When looking at the coefficients and their statistical significance one realizes that the regression model obtained for the training data subset is fairly similar to the regression model before which was based on the whole dataset (Figure 19). Thus, unlike for the decision tree algorithm, we can draw conclusions with respect to the accuracy of our original model.

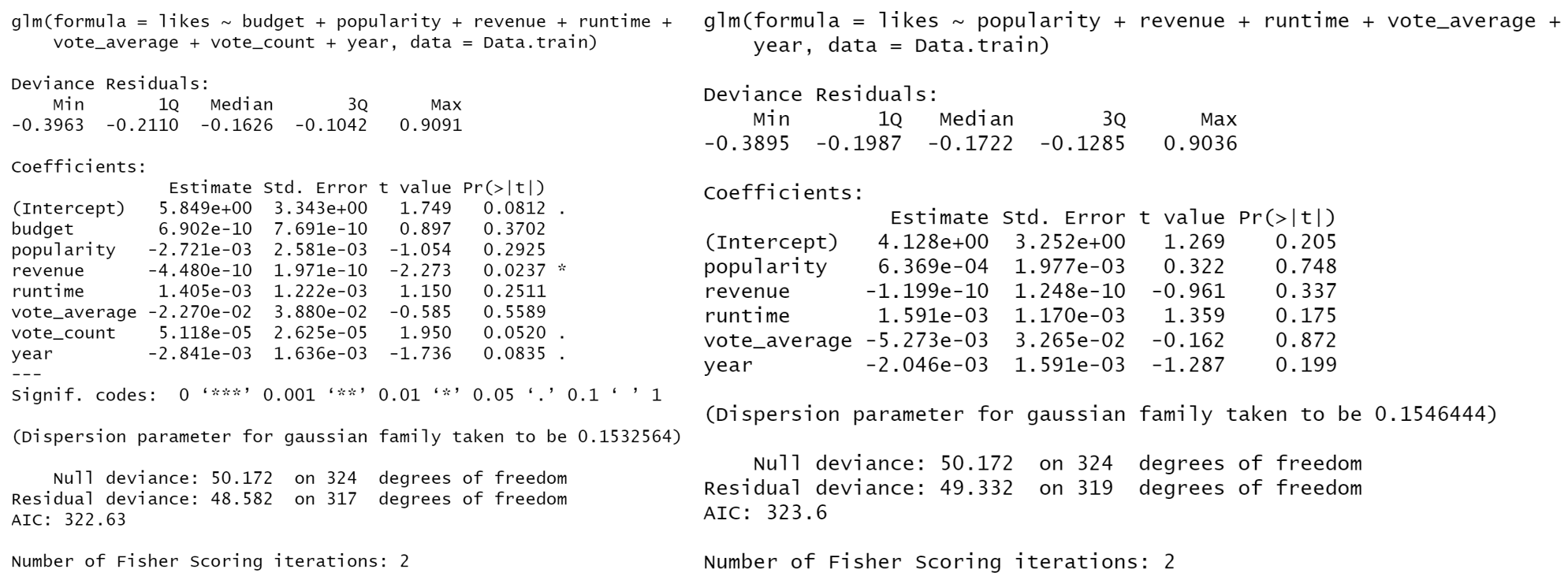


Figure 19: Summary statistics of the logistic regressions based on the training data for all numeric features (left) and when accounting for multicollinearity (right)

We predict the probability of a perfect rating for each movie in the testing dataset based on the two regression models. To compute the error rate of our regression model, we need to convert those probabilities into a binary like variable. We choose 0.165 as our decision criterion which is equal to the fraction of five-star ratings overall. Whenever the predicted probability exceeds this threshold, we assume that the user will give a movie a perfect score. The error rate is then calculated as the percentage of false predictions. For the first regression model which includes all numeric features we obtain a high error rate of 56.5%. The second model performs even worse and produces an error rate of 61.6%. However, note that the error rate drops substantially when we increase the decision criterion and converges to 21.7% (i.e. the fraction of perfect ratings in the testing dataset), as for high values of decCrit we always predict a score worse than 5.

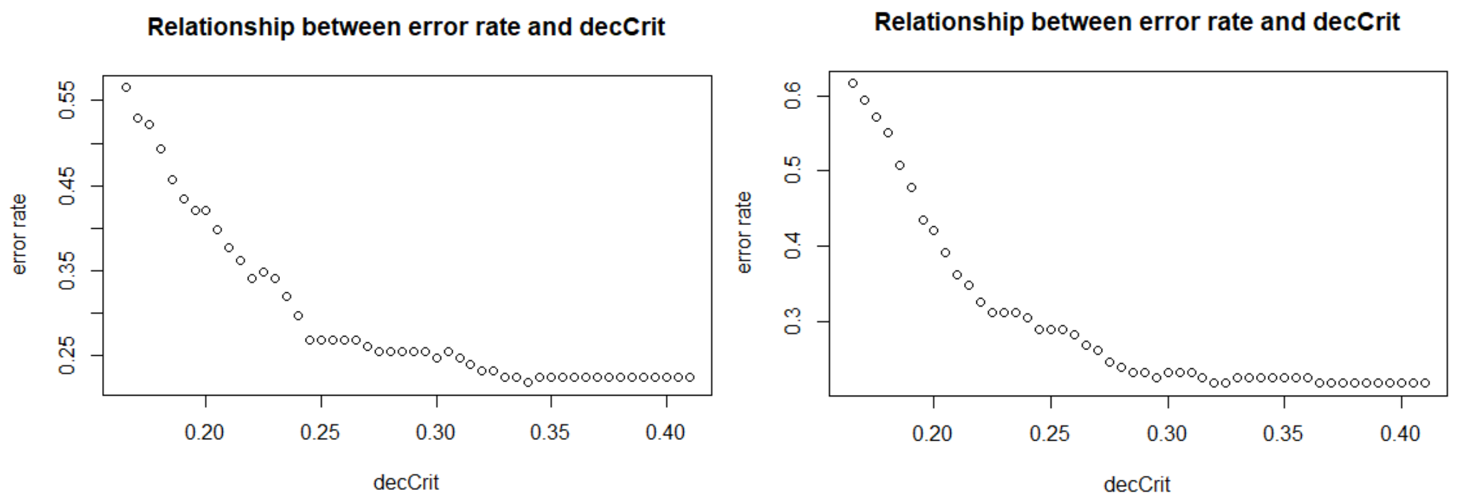


Figure 20: Relationship between the error rate and the decision criterion for the

Contrary to the k-NN and plot-based recommendation systems, the logistic regression model is constantly learning and should improve over time. It cannot be applied to new users as it requires at least one observation for training purposes. The approach is also highly susceptible to outliers when there are only a few observations. Moreover, we find that the model performs poorly even when a large number of observations is available. Thus, the logistic regression specified above is not an adequate tool for making movie recommendations and it is questionable whether features like runtime, revenue or release year are good indicators for whether a user will like a movie or not. Perhaps the regression model could be improved by including other independent variables such as dummy variables for genres, directors or main actors.

1. **Conclusion**

**Appendix**