Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Magistrale in Scienze Statistiche

Predicting IMDb ratings: an analysis of factors influencing critical success in movies

Davide Bortoletto, Bryan Patarini, Gianmarco Rosa, Greta Schiappacasse

January 5, 2026

Contents

Section 1
Introduction

1.1 Research question

A major Hollywood film producer has commissioned our team to carry out a study with the goal of improving the artistic quality and critical reception of future movies. Rather than focusing only on box office success, the producer wants to create films that are more appreciated by critics and by movie enthusiasts.

The main goal of this project is to understand which production choices are associated with high critical ratings. In particular, the producer is interested in knowing which decisions made during the production process can improve the quality of a film. For example, which actors should be hired? Which directors are more likely to deliver successful movies? Which screenwriters and composers should be chosen? Which genres tend to receive better reviews? The producer is also interested in other creative and strategic aspects. These include whether certain words used in movie titles are more attractive to audiences, and whether targeting a specific audience category (such as movies suitable for all viewers, restricted to viewers aged 14 and above, or limited to adults only) has an impact on critical success. The explanatory goal of this analysis is to provide data driven answers to these questions, by identifying the elements that are most strongly related to higher critical ratings.

In addition to this explanatory goal, the producer has a predictive objective. They would like to estimate the expected critical ratings of one of their upcoming movies scheduled for release in the next few months: The Odyssey, directed by Christopher Nolan, which will be released in July 2026.

Section 2
Dataset construction

To meet our client's request, we needed to build an appropriate dataset. For each movie, the dataset had to contain information about the cast members, other characteristics of the film, and an overall indicator of the movie's critical rating. The code written and used to build the dataset is available in the file **dataset_creation_and_cleaning.R**.

2.1 Gathering information from IMDb bulk data

The dataset was built in several steps by combining two main sources: IMDb, one of the most well-known movie review aggregators, and Wikidata, the Wikipedia API. IMDb provides several bulk data files with information on almost every movie ever produced, available at **https://datasets.imdbws.com/**. In our analysis, we used the following files:

- **title.ratings.tsv**: contains the average rating and the number of votes for each movie;
- **name.basics.tsv**: includes basic information about people, such as actors, directors, writers, and composers;
- **title.crew.tsv**: lists the directors and writers associated with each movie;
- **title.principals.tsv**: contains the main cast and crew members for each movie;
- **title.basics.tsv**: provides general information about each movie, including the title, release year, runtime, and genres.

We filtered information from these datasets: since our commissioner is a major producer, we decided to consider only movies with at least 18k votes. We also decided to limit our analysis to movies from 1980 until 2024, since movies too old would be having a completely different cast from the cinema industry workers that are available today. On the other hand, we decided to keep deceased cast members in the dataset for the following reason. Even though the variables releated to deceased cast members are not useful from an interpretative point of view (the producer who commissioned the study will never be able to hire a deceased actor), they need to be kept in the model as control variables. Indeed, if an important but deceased cast member was removed from a successful movie in the dataset, part of the movie's success would be incorrectly attributed to the remaining younger cast members. This would make

their coefficients artificially larger. As a result, the model could produce biased conclusions and potentially lead our client to hire the wrong people.

From the file, basic information was taken about each movie:

- `tconst`: IMDb id of the movie;
- `primaryTitle`: full title in English of the movie;
- `startYear`: release year of the movie;
- `runtimeMinutes`: runtime in minutes of the movie;
- `averageRating`: average movie rating provided by IMDb, a continuous value ranging from 1 to 10. This can be seen as an indicator of the movie's overall quality and critical reception. It will be used as our response variable.

After identifying the movies, we collected information on their cast members. We selected the top 3500 actors, 300 directors, 100 writers, and 30 composers. We chose a larger number of actors and directors because we expect them to have a stronger impact on the perceived quality of a movie. We considered a larger number of actors than directors because each film usually has a large cast of actors and only one or two directors. In addition, we required that each person appears in at least six movies in our sample. This rule was used to exclude people with very limited experience in successful movies, as we assumed they would have a smaller effect on the rating, and to limit the dimensionality of the dataset.

Each cast member was represented by a dummy variable: the value is 1 if the person worked on the movie, and 0 otherwise. After checking that there were no cases of homonymy, we named each column related to a person using the format `job_name_surname`, where the job could be actor, director, writer, or composer.

Along with the main movie dataset, we created a second, smaller dataset. For each person included in the movie dataset, this dataset contains one row with their full name, IMDb ID, and date of birth. This second dataset will be important in the next part of the analysis, where we will collect additional information about the cast members.

2.2 Scraping from WikiData

While the information provided by the IMDb bulk files was already meaningful, we realized that some important variables were missing. It would have been useful to know the budget of each movie, the country where it was released, the studio that produced it, and its content rating. In addition, we wanted more information about the cast and crew. For example, we were interested in knowing whether the actors and directors were already experienced, and whether they had won major awards before the movie was released. For this reason, we decided to consult another reliable source, Wikipedia, through its API, WikiData.

The code included in the second section of the file **dataset_creation_and_cleaning.R** is used to build the queries to interrogate the API using the SPARQL language, which is the standard query language used to access Wikidata. The code writes a SPARQL query for each movie and accesses the Wikidata public API through an HTTP request. The code first tries to identify the movie using its IMDb ID, which is the most reliable method. If that fails, it retries by using the movie title. Through these queries, it retrieves information

about the production country, the production studio, the content rating (MPAA scale), the box office revenue, the budget expenses, the number of prizes won by the movie.

The responses are returned by Wikidata in JSON format, parsed in R, and cleaned so that missing or inconsistent values are handled safely. Since budgets and box office figures can be expressed in different currencies, the code also converts them into U.S. dollars using predefined exchange rates. To avoid overcomplicating the analysis, when cleaning the budget and box office variables, we chose to ignore the inflation factor, which would have varied in different countries and different time periods. This process is repeated for all movies in the dataset, with a short pause between requests to avoid overloading the Wikidata servers, and the final result is a raw dataset where each movie is associated with the data downloaded from Wikidata. Then, the raw dataset is cleaned: budget and box office values are rescaled into millions to make them more interpretable. Categorical variables (production countries, studios, and content ratings) are transformed into dummy variables, keeping only the most frequent categories (in accordance to what was done with actors, directors, writers and composers). To handle NA values in factors, a separate category for missing values is created. Finally, all these components are merged into a single clean dataset.

The final part of the scraping code of the file **dataset_creation_and_cleaning.R** collects and processes information about the cast of each movie included in the dataset. This passage is fundamental to obtain two potentially significant variables: the number of awards and, more specifically, the number of Academy awards won by the cast. To avoid data leakage, we chose to consider only the prizes won up until the year before the release of the movie. First, the code queries Wikidata using IMDb person IDs to retrieve nationality, spoken languages, and all awards won by each person, including the year of each award. Duplicate awards are removed, and both a long format (one row per award) and an aggregated format (one row per person) are created.

Finally, the code matches actors, directors, writers, and composers listed in each film (using dummy variables) with the Wikidata award data. For every movie, it reconstructs the full cast and crew, then counts how many awards and Oscars these people won before the movie's release year. These counts are finally added to the film dataset as new covariates, which can be used for prediction models.

The code used to build the queries in the SPARQL language, to build the HTTPS request and to parse the JSON files, both for movies and cast members, was written with the assistance of Artificial Intelligence and then modified by us. The specific lines of code generated by AI are highlighted in the file **dataset_creation_and_cleaning.R**. In the end, only some of the variables scraped from Wikidata will be kept in the final dataset: some of them will be removed as leaker variables, others won't be considered interesting for the analysis.

Section 3
Dataset Cleaning

The code utilised to clean the dataset is included in the file **dataset_creation_and_cleaning.R**. In this report, we briefly describe the cleaning process.

3.1 Transformation of the response

The IMDb movie rating is a numeric continuous variable that theoretically ranges from 1 (the lowest possible score) to 10 (the highest possible score). Since all the models we fit assume that the response variable can take values on the whole real line, and since we want to avoid predictions outside the theoretical support of the rating, we decided to apply a transformation to the response variable so that it takes values in $\mathbb{R}$. If x is the `averageRating` and `y` is the new transformed response variable, then the transformation used is defined as

$$\texttt{y} = \text{logit}\left(\frac{x-1}{9}\right),$$

where x is the original IMDb rating. Since two movies in our dataset had a rating exactly equal to 1, this transformation would produce infinite values. To avoid this issue, we reassigned these two movies a rating of 1.01 before applying the transformation.

3.2 Removing leaker variables

Our models are also designed to predict the critical success of a movie just before its release. This means that any variable that is not available at that time must be removed, as it would cause data leakage. We therefore identified the following variables in the dataset as leaker variables:

- `num_votes`: the number of IMDb votes is known only after the movie has been released and the average rating (our response variable) is already available.
- `box_office`: the total revenue of the movie is known well after its release.
- `num_prizes` and `num_oscars`: awards won by the movie are announced long after the movie is released in theaters.

On the other hand, the variables `awards_before` and `oscars_before` are not leaker variables: in fact, they respectively count the number of generic awards and Oscar awards won by cast members up until the year before the release of the movie: this information is available during the prediction process.

3.3 Handling NA values

After the inclusion of the dummy variables, the only columns that contain NA values are `budget_million_usd` (65% of NA values) and the two cyclical encoding variables, `time_month_sin` and `time_month_cos` (0.6% of NA values). Since only 48 out of 7,600 movies are missing the release month, and this information is likely missing at random in Wikidata, we treat this as a MCAR case. Therefore, we decided to remove the rows corresponding to these movies.

On the other hand, the `budget_million_usd` covariate is more delicate: it has a higher percentage of missing values, omitting the rows that have no budget would significantly decrease the size of our dataset. Moreover, it is likely a case of not missing at random: less successful movies are more likely to have a lacking Wikidata page. Doing imputation for this specific variable was also not recommended, since the percentage of NA values is too high. For these reasons, it was decided to convert the budget variable from a numeric covariate to a discretized variable, with a specific level for missing values. The new `budget` variable has five levels: `low`, `medium`, `high` and `veryhigh`.

3.4 Text mining on the movie titles

To enrich the dataset with other variables we include also some features related to the title of the movies. The movie titles were cleaned and normalized through a series of steps: converting to lowercase, removing numbers, eliminating common stopwords and extra spaces. Each title was then split into individual words, and only words of at least three characters were kept. These words were further reduced to their root forms using a linguistic stemming procedure. From the resulting list of root words, the 70 most frequently occurring terms were identified. For each of these top words, a binary column was added in the dataset, indicating whether that term appears in the corresponding movie title. The new variables present a large amount of zeros.

3.5 Cyclical encoding of time variables

One useful piece of information obtained from Wikidata is the full release date of each movie. In fact, movies released at the end of the year tend to be more successful because of the holiday season and are more likely to receive Oscar nominations, while movies released at the beginning of the year are often less successful. For this reason, we wanted to include seasonality in the month variable.

To do so, we used cyclical encoding for the month of release. Instead of using the month as a simple index from 1 to 12, we applied trigonometric functions with a period of 12. If `month` is the index variable of the month, the new cyclical encoding variables are defined as:

$$\texttt{month\_sin} = \sin\left(\frac{2\pi \cdot \texttt{month}}{12}\right), \quad \texttt{month\_cos} = \cos\left(\frac{2\pi \cdot \texttt{month}}{12}\right)$$

This approach correctly captures the fact that December and January are contiguous months, which a standard numeric encoding cannot represent. Both sine and cosine components are included to fully describe the 12 month periodicity: using only one of them would assign the same sine value to months that are symmetric with respect to March and April, and the same cosine value to months that are symmetric with respect to January and December.

3.6 Final dataset structure

The final dataset contains 7604 rows (movies) and 3031 columns. Most of the covariates are dummy variables that can be grouped in the following way:

- `genre_GenreName`: columns 4-25, possible genres of the movie;
- `actor_Name_Surname`: columns 26-2472, possible actors of the movie;
- `director_Name_Surname`: columns 2473-2772, possible directors of the movie;
- `writer_Name_Surname`: columns 2773-2872, possible screen players of the movie;
- `composer_Name_Surname`: column 2873-2902, possible composers of the soundtrack of the movie;
- `country_CountryName`: columns 2903-2913, possible production countries of the movie;
- `studio_StudioName`: columns 2914-2943, possible studio production of the movie;
- `rating_RatingCode`: columns 2944-2952, possible rating of the movie;
- `word_WordName`: columns 2955-3024, possible words from the title of the movie;
- `budget_BudgetLevel`: columns 3027-3030, budget levels of the movie.

The other covariates are the following:

- `time_startYear`: column 2, release year of the movie;
- `time_mese_cos` and `time_mese_sin`: columns 3025 and 3026, cyclical encoding variables of the release month of the movie;
- `prizes_before` and `oscars_before`: columns 2953 and 2954, number of Oscar awards and number of awards in general won by the cast up until the year before the movie release date.

Finally, the transformed response variable `y` is included in the first column of the dataset.

```
head(film[,c(1:4,26,2473,2773,2873,2903,2914,2945,2955,3027)])
```

```
##                   y time_startYear runtimeMinutes genre_Drama actor_Meg_Ryan
## tt0035423 0.4054651           2001            118           0              1
## tt0078935 0.1335314           1980             95           0              0
## tt0080319 0.6435502           1980            109           0              0
## tt0080339 1.0691984           1980             88           0              0
```

```
## tt0080354 0.2682640           1980             91           0              0
## tt0080360 0.6435502           1980            102           0              0
##           director_James_Mangold writer_Dan_Aykroyd composer_John_Williams
## tt0035423                      1                  0                       0
## tt0078935                      0                  0                       0
## tt0080319                      0                  0                       0
## tt0080339                      0                  0                       0
## tt0080354                      0                  0                       0
## tt0080360                      0                  0                       0
##           country_United_States studio_Miramax rating_PG word_man budget_low
## tt0035423                     1              1         0        0          1
## tt0078935                     0              0         0        0          1
## tt0080319                     1              0         0        0          0
## tt0080339                     1              0         1        0          1
## tt0080354                     1              0         0        0          1
## tt0080360                     1              0         0        0          0
```

3.7 Train-test split

To reduce the computational cost, since our dataset has a large number of columns, we decided to use a train-test approach. We randomly split the rows of the original dataset into a training set (containing about 75% of the observations) and a test set (containing about 25% of the observations). For parameter tuning, we used cross-validation on the training set, so that we could take advantage of the built-in R functions and avoid further reducing the size of the dataset.

Section 4
Exploratory analysis

A key aspect of our dataset is its high dimensionality and sparsity, which can affect both modeling choices and computational efficiency. To quantify sparsity, we computed the proportion of zero entries across all features in the design matrix:

```
dim(train)
```

```
## [1] 5703 3030
```

```
mean(train == 0)
```

```
## [1] 0.9942618
```

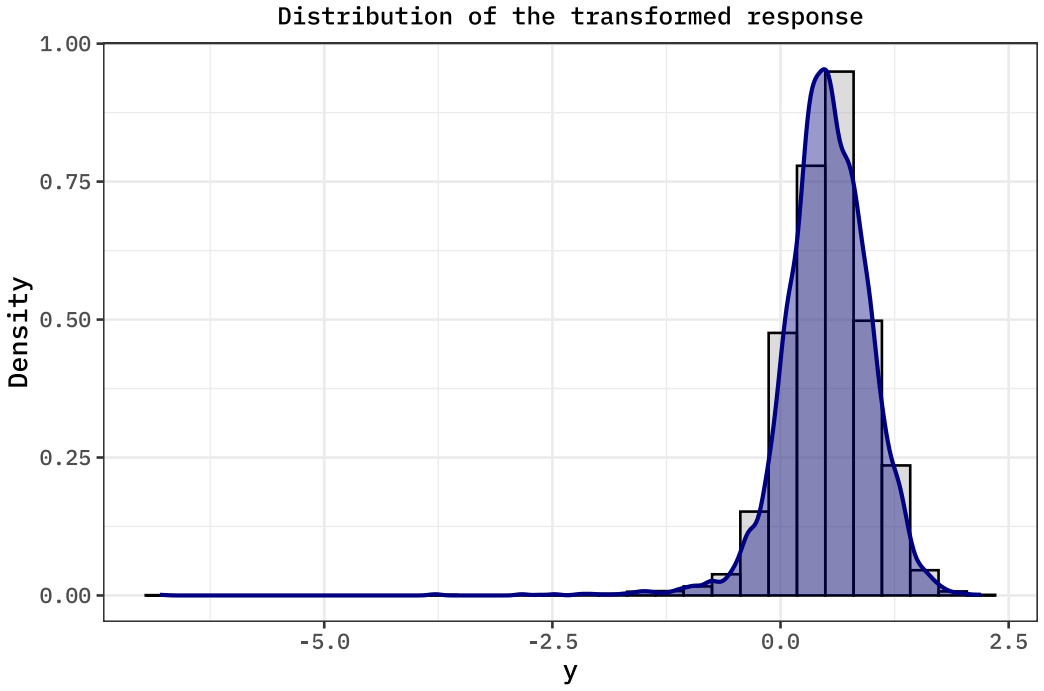
The result shows that more than 99% of the entries are zeros, indicating that the design matrix is extremely sparse. Accordingly, in the next section we adopt models that can efficiently handle sparse and high-dimensional data.

Next, we perform some exploratory analysis on the training set in order to visualize the relationship between the response variable and some covariates and marginal distribution of the predictors.

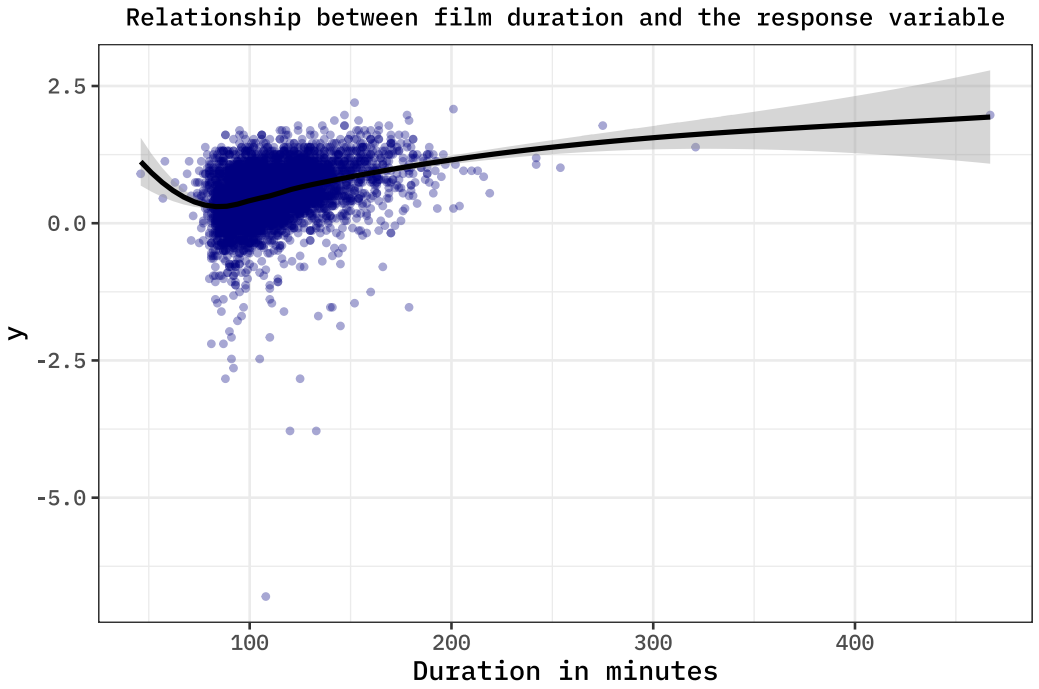
```
basicStats(train$y)
```

```
##              X..train.y
## nobs        5703.000000
## NAs            0.000000
## Minimum       -6.801283
## Maximum        2.197225
## 1. Quartile    0.268264
## 3. Quartile    0.794930
## Mean           0.507453
## Median         0.498991
## Sum         2894.002603
## SE Mean        0.006430
## LCL Mean       0.494848
## UCL Mean       0.520058
## Variance       0.235785
```

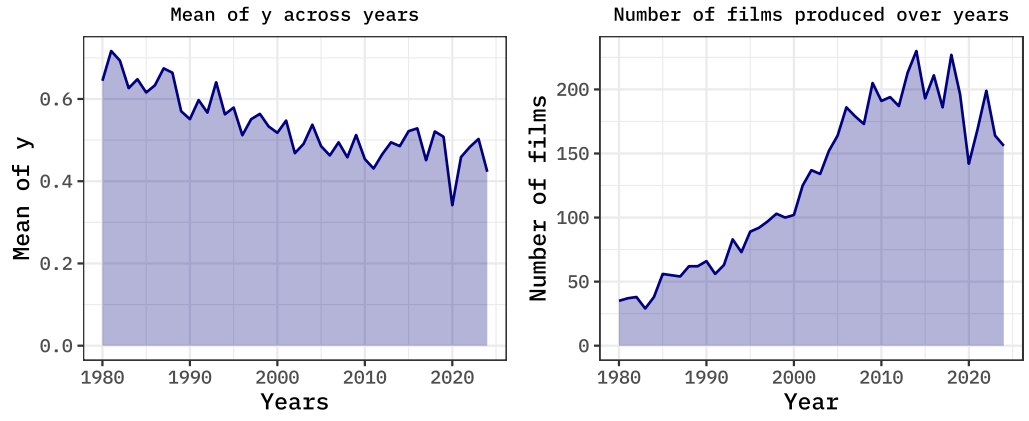
```
## Stdev          0.485577
## Skewness      -1.517083
## Kurtosis      13.857547
```



The summary statistics are coherent with the shape observed in the distribution of the transformed response variable `y`. The median (0.499) and mean (0.509) are very close, indicating a fairly balanced central tendency. The interquartile range (0.53) and the standard deviation (0.49) both point to a moderate level of variability, with dispersion largely concentrated around the center. The minimum value (-6.80) is far from the bulk of the data, confirming the presence of a long left tail. Ultimately, most values lie between the first and third quartiles ($0.27 - 0.79$).

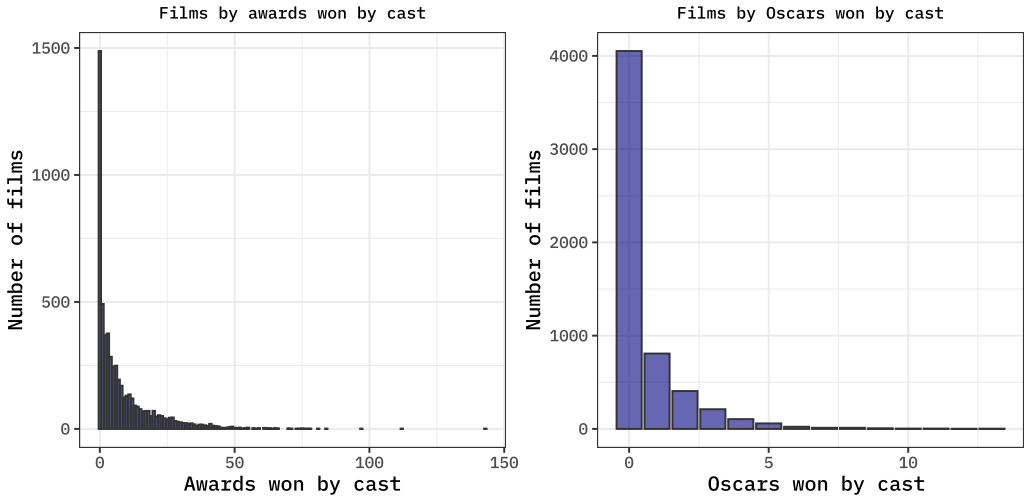


The scatter plot highlights the relationship between film duration and the response variable y. Most observations are concentrated between roughly 80 and 150 minutes, where y shows substantial variability. The smooth trend suggests a non-linear relationship; y slightly decreases for shorter durations and then gradually increases as duration grows. For longer films, the upward trend becomes more pronounced, although the wider confidence band indicates greater uncertainty due to the smaller number of observations. Overall, duration appears to be positively associated with y, but the effect is modest and not strictly linear.

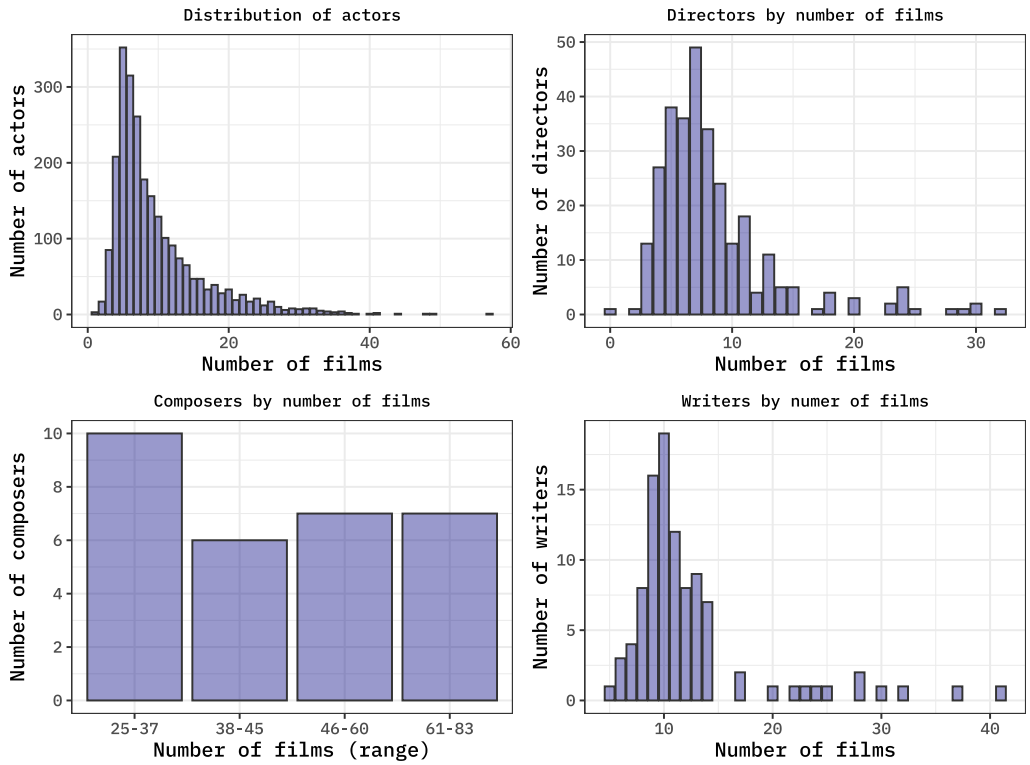


As shown in the left-hand-side panel, a clear downward linear trend in the mean value of y from 1980 to the early 2000s, followed by a period of relative stabilization with moderate fluctuations. While short term variability persists throughout the series, no sustained

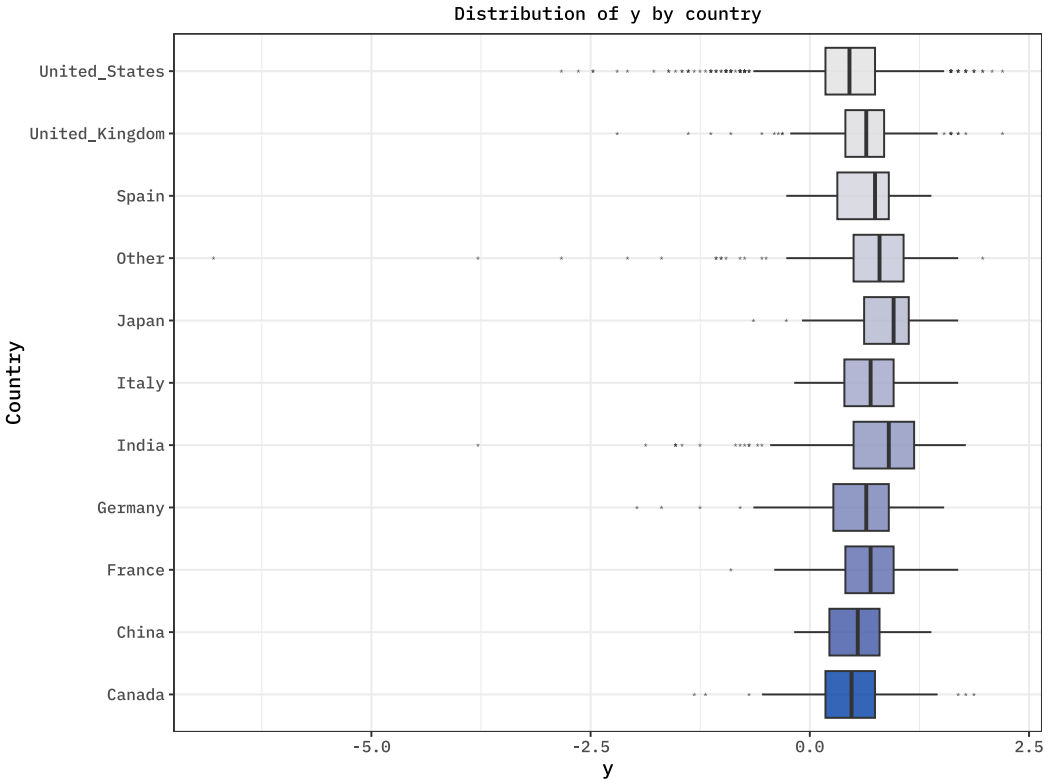
recovery to the initial levels observed in the early years is evident. The pattern suggests a long-term decline with little attenuation in later years. On the other hand, the number of films produced exhibits a strong upward trend from 1980 through the 2010s, indicating a substantial expansion of production over time. This relationship is not strictly linear, seems instead quadratic. In the most recent years, the series shows increased variability and a noticeable decline relative to the preceding peak, suggesting a contraction in film production due to the pandemic. Taken together, the two graphs indicate that the long-term increase in film production is not accompanied by a corresponding increase in the mean value of `y`. On the contrary, the transformed response declines while output expands, pointing to a potential decoupling between quantity and the underlying rating measured by `y`.



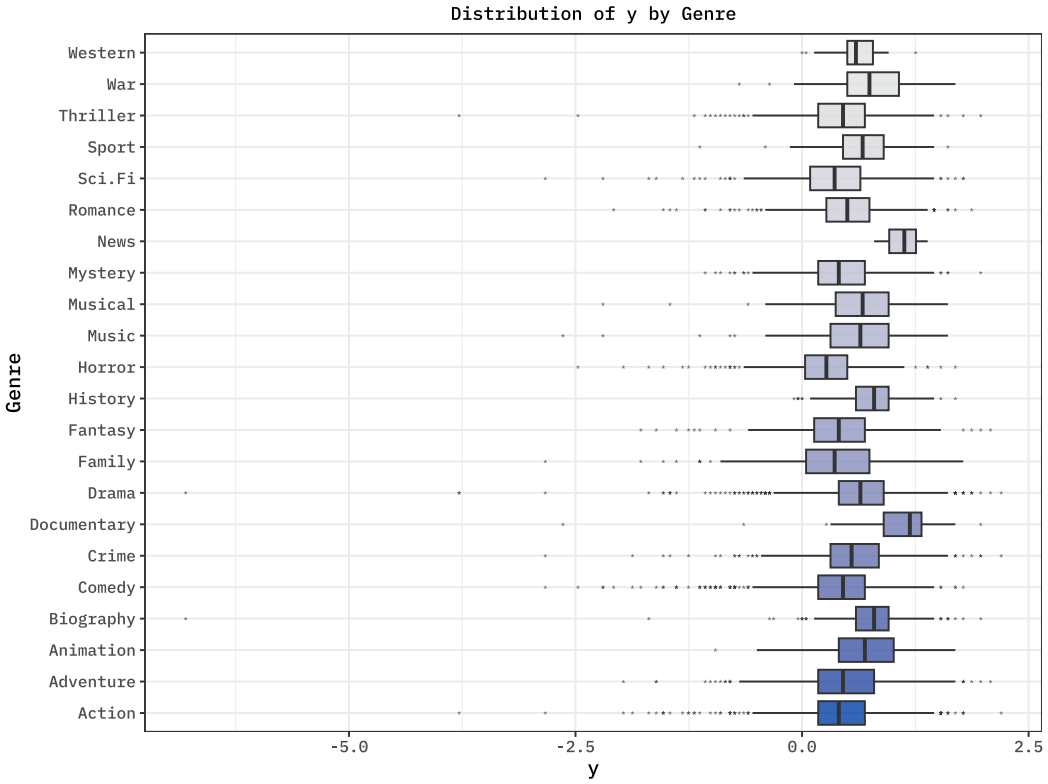
Both distributions are highly right-skewed, indicating that the majority of films are associated with casts that had won few or no awards prior to the film's release, while a small number involve highly decorated casts. The left panel shows a long tail for total awards, reflecting substantial heterogeneity in prior recognition. The right panel displays a similar but more concentrated pattern for Oscars, with most films linked to casts with zero or one previous Oscar. Importantly, all counts of awards and Oscars refer exclusively to achievements obtained before the release date of each film, in order to avoid any form of data leakage from post-release outcomes.



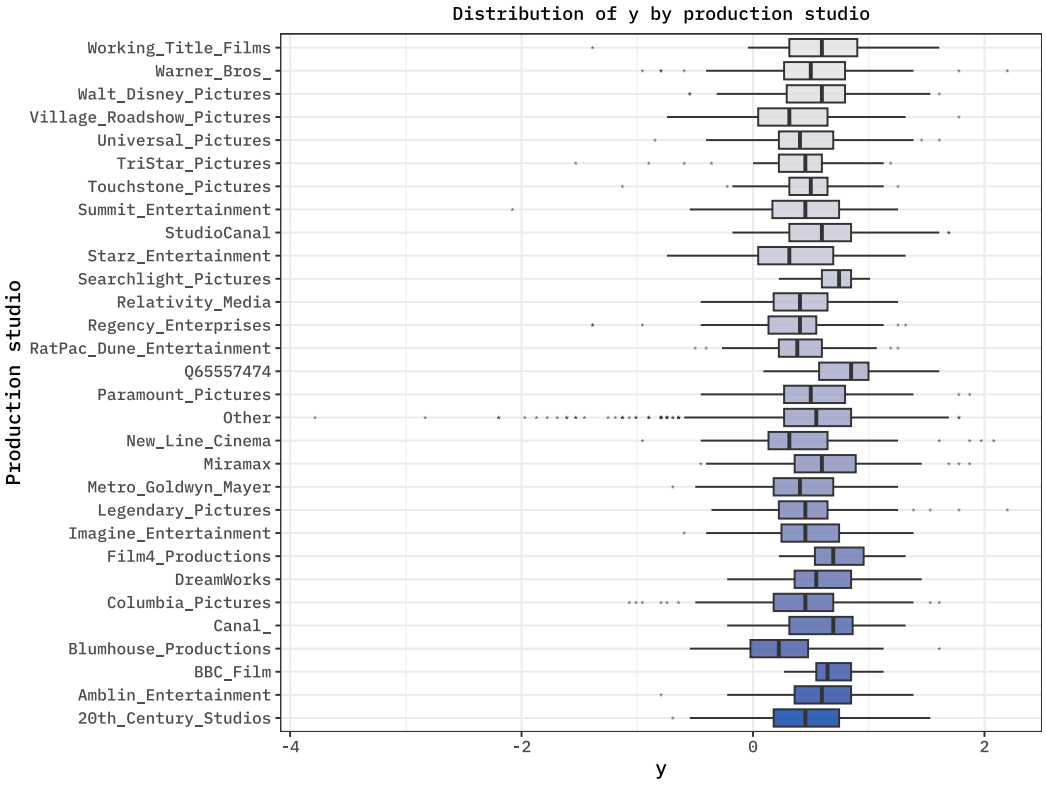
The four barplots exhibit pronounced right skewness, indicating that most individuals are associated with a relatively small number of films, while a limited minority accounts for very high output. Actors show the strongest concentration at low values, with a long tail reflecting a small group of highly prolific performers. Directors display a similar but less extreme pattern. Writers also present a right-skewed distribution, with moderate dispersion and a few high-output outliers. In contrast, composers are grouped into predefined productivity classes for graphical clarity. The distribution indicates that composers tend to be associated, on average, with a higher number of films compared to the other professional roles. In conclusion, the results point to heterogeneous productivity structures across professional roles, with inequality in output being most pronounced among actors.



The boxplot analysis of transformed IMDb scores by country reveals that `Japan` and `India` maintain the highest median values. While most nations cluster between 0.5 and 1.0, the `United States` and `United Kingdom` exhibit the greatest variability, characterized by wide interquartile ranges and numerous negative outliers. Conversely, `Spain` and `Italy` show the highest consistency with the narrowest score distributions. The `Other` category contains the most extreme negative outlier, reaching approximately -6.5. Overall, the significant overlap of boxes suggests a relatively uniform core performance across all geographical regions.



This boxplots illustrate the distribution of `y` across various film genres, revealing that most medians hover slightly above zero. `News` and `Documentary` show the highest central tendencies and narrowest spreads, suggesting more consistent values. In contrast, genres like `Action`, `Comedy`, and `Drama` exhibit significant negative outliers, indicating high variability and occasional extreme underperformance in the transformed response variable `y`. In general, most genres maintain a median near 0.5, but the significant overlap in notches (where visible) suggests that differences between many categories seems not be significant.



The boxplot illustrates the distribution of transformed IMDb scores across production studios. `Film4 Productions` and `BBC Film` exhibit the highest medians and the most concentrated interquartile ranges, indicating consistent performance. Conversely, `Blumhouse Productions` shows one of the lowest medians in the dataset. The `Other` category displays the greatest dispersion, with significant negative outliers extending toward -4. Given the extensive overlap of notches and whiskers across most studios, the differences in central tendency for many groups appear marginal.

Wordcloud

The word-cloud visualizes the most frequent terms in the dataset, with font size mapped to frequency. In other "words", the more a word is frequent in the title of films, the bigger is its size in the wordcloud plot. High-frequency keywords such as `man`, `day`, and `love` dominate the central area, representing the most common thematic elements. Smaller peripheral terms like `monster` or `secret` indicate lower occurrences.

Section 5
Regression

In this section, we present several regression models suitable for handling high-dimensional and sparse data. We first estimated LASSO (*Least Absolute Shrinkage and Selection Operator*) regression, Ridge regression, and Elastic Net using our dataset. Then, a sub-dataset was built to compute LASSO regression and Elastic Net including first order interactions. Ridge regression with interactions was not estimated due to its high computational cost. SCAD (*Smoothly Clipped Absolute Deviation*) and MCP (*Minimax Concave Penalty*) models are also presented in this section.

5.1 LASSO

We start by implementing LASSO Regression, which uses L1 penalty. The optimal tuning parameter $\lambda = 0.00783$ was selected using 8-fold cross validation. Using this parameter, 393 coefficients are set different from zero. The MSE on the test set for LASSO is 0.18282.

```
set.seed(16)
m1 <- cv.glmnet(x = X.train, y = train$y, nfolds = 8)
lasso <- glmnet(x = X.train, y = train$y, lambda = m1$lambda.min)
```

5.2 Ridge

We also implemented Ridge Regression with L2 regularization. Unlike LASSO, Ridge Regression does not perform variable selection; all coefficients remain non-zero. The regularization parameter λ is chosen via 8-fold cross-validation, resulting in $\lambda = 0.480216$. The MSE on the test set for Ridge is 0.19620.

```
set.seed(8)
ridge.cv <- cv.glmnet(x = X.train, y = train$y, nfolds = 8, alpha = 0)
ridge <- glmnet(x = X.train, y = train$y, lambda = ridge.cv$lambda.min,
                alpha = 0)
```

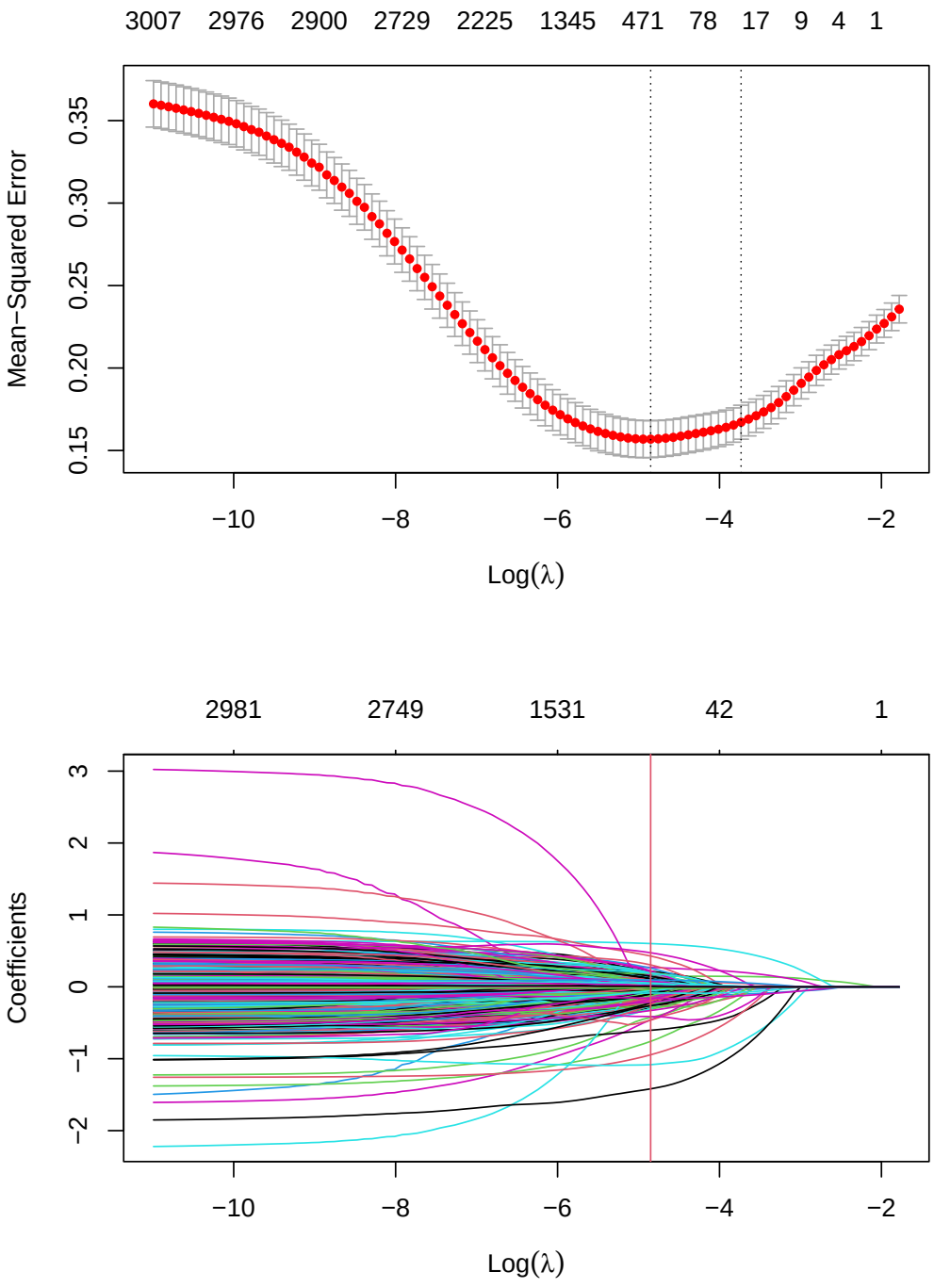


Figure 5.1: 8-fold CV MSE and coefficients paths for LASSO as a function of $\log(\lambda)$.

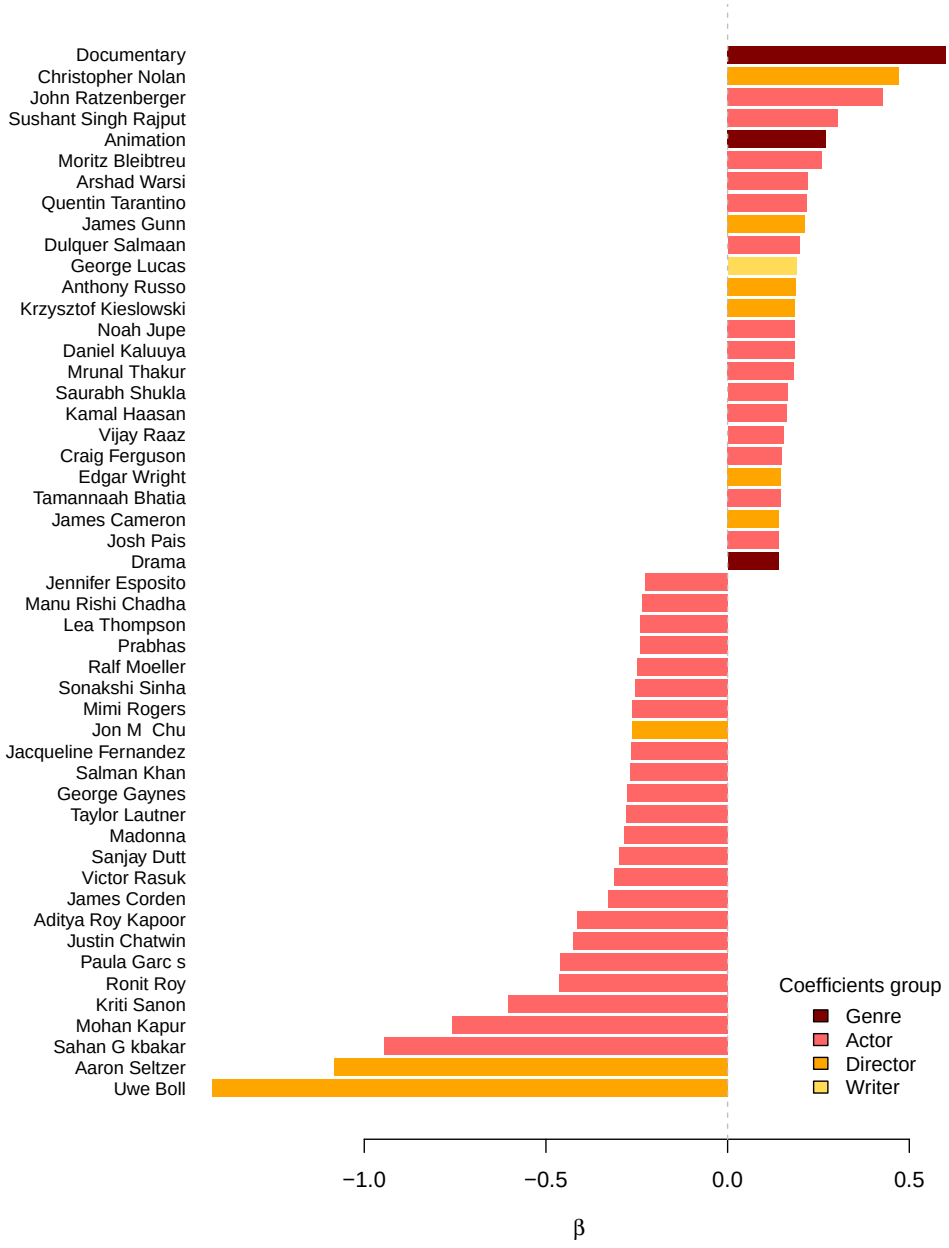


Figure 5.2: Top 50 coefficients (by absolute value) estimated by LASSO, colored by feature group.

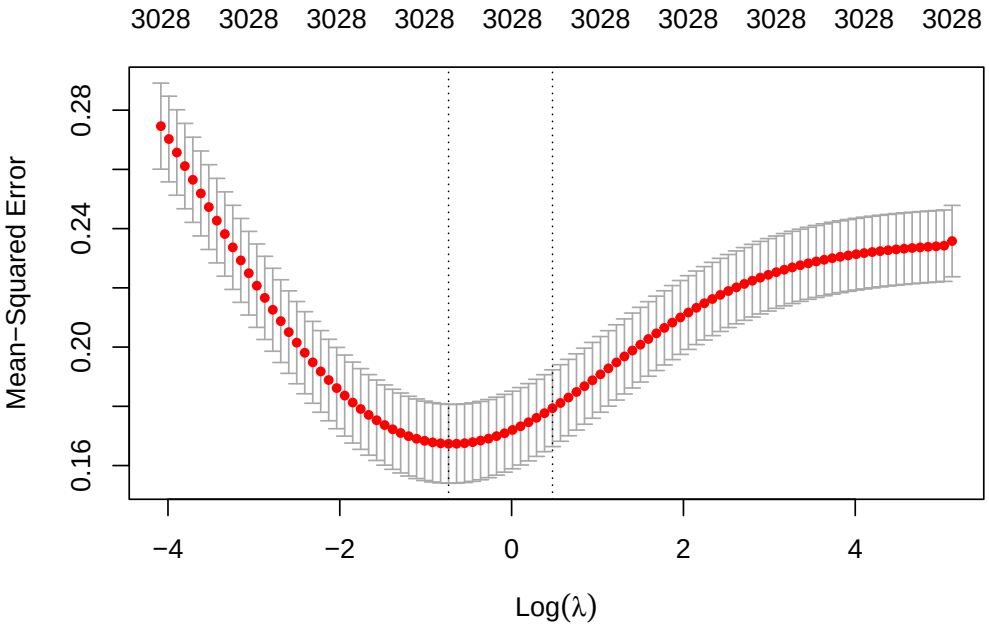


Figure 5.3: 8-fold CV MSE for Ridge as a function of $\log(\lambda)$.

5.3 Elastic Net

Elastic Net was also estimated on the data. We selected α and λ via cross validation. The parameter α was set to 0.65, using a 4-fold cross validation, trying multiple values of the parameter. Using this value of α, λ was chosen via 8-fold cross validation and set equal to 0.01321. The MSE on the test set for Elastic Net is 0.18414.

```
set.seed(123)
alpha <- c(0.1, 0.25, 0.45, 0.65, 0.85, 0.95)
err <- NULL
for(i in 1:length(alpha)){ # CV for alpha
  print(i)
  a <- alpha[i]
  EL_tmp <- cv.glmnet(x = X.train, y = train$y, nfolds = 4, alpha = a)
  err[i] <- EL_tmp$cvm[EL_tmp$lambda == EL_tmp$lambda.min]
}

set.seed(88)
alpha_best <- alpha[which.min(err)]
elastic.net <- cv.glmnet(x = X.train, y = train$y, alpha = alpha_best,
                         nfolds = 4) # CV for lambda
elastic.net <- glmnet(x = X.train, y = train$y, alpha = alpha_best,
                      lambda = elastic.net$lambda.min)
```

Elastic net set some coefficients equal to zero and shrink others. In this case 346 coefficients are set different from zero. Looking at Figure 5.4, we observe that most of the coefficients align with those estimated by the LASSO regression. The documentary genre remains associated with the largest coefficient, while Ratzenberger and Nolan continue to rank among the top contributors. Similarly, the negative coefficients show a clear agreement with the LASSO estimates.

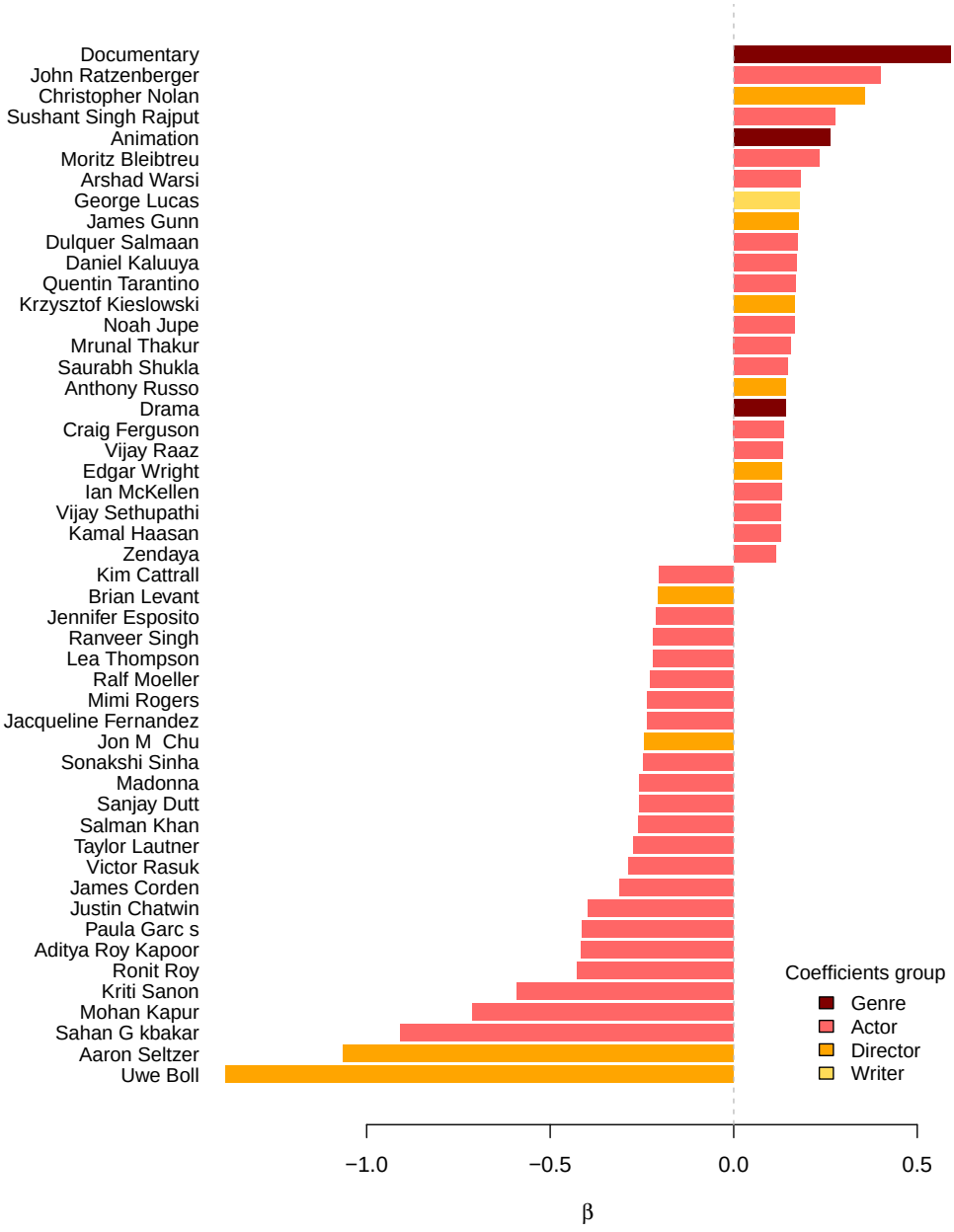


Figure 5.4: Top 50 coefficients (by absolute value) estimated by Elastic Net, colored by feature group.

5.4 MCP

In order to compute MCP, we tested four values of the γ parameter: 2.8, 3, 3.2, and 3.5. No significant differences were observed in the cross-validation error across these values, so we chose $\gamma = 3$, the default option. The value of the parameter λ is chosen via cross validation, and set equal to 0.01871. The model estimated on the training set selects 29 coefficients. The MSE on the test set for MCP is 0.19532. Figure 5.5 reports cross-validation error graph and the coefficients paths for MCP.

```
set.seed(100)
id_cv <- sample(1:5, nrow(train), replace = T)
lambda_try <- c(seq(0.003, 0.025, length.out = 50),
                seq(0.025, 0.6, length.out = 65))
set.seed(1)
cv.mcp3 <- cv.ncvreg(X = X.train,y = train$y,
                     folds = id_cv,
                     lambda = lambda_try, gamma = 3)
mod.mcp.3 <- ncvreg(X = X.train,y = train$y,
                    lambda = cv.mcp3$lambda.min)
```

5.5 SCAD

Even for SCAD, the parameter γ did not impact much on CV-error. We decided to set it equal to 3. λ parameter was then chosen via cross validation and set equal to 0.02096. The model on the training set sets 28 coefficients different from zero and has a MSE of 0.1988 on the test set. Figure 5.6 reports cross-validation error graph and the coefficients paths for SCAD. The optimal solution lies in the non convexity area.

```
set.seed(100)
id_cv <- sample(1:5, nrow(train), replace = T)
lambda_try <- c(seq(0.003, 0.025, length.out = 50),
                seq(0.025, 0.6, length.out = 65))
set.seed(1)
cv.scad3 <- cv.ncvreg(X = X.train,y = train$y,
                      penalty = 'SCAD', folds = id_cv,
                      lambda = lambda_try, gamma = 3)
mod.scad.3 <- ncvreg(X = X.train,y = train$y,
                     lambda = cv.scad3$lambda.min)
```

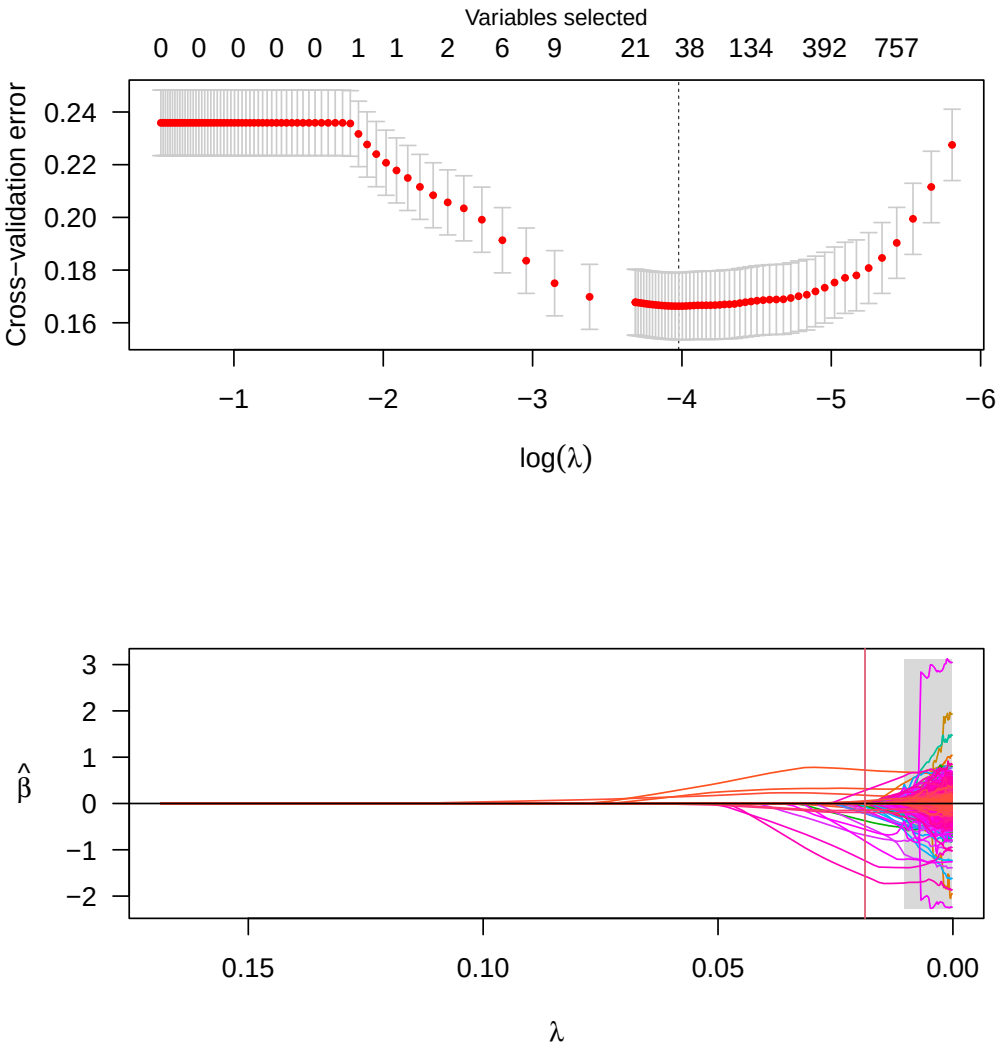


Figure 5.5: 5-fold CV MSE and coefficients paths for MCP as a function of $\log(\lambda)$.

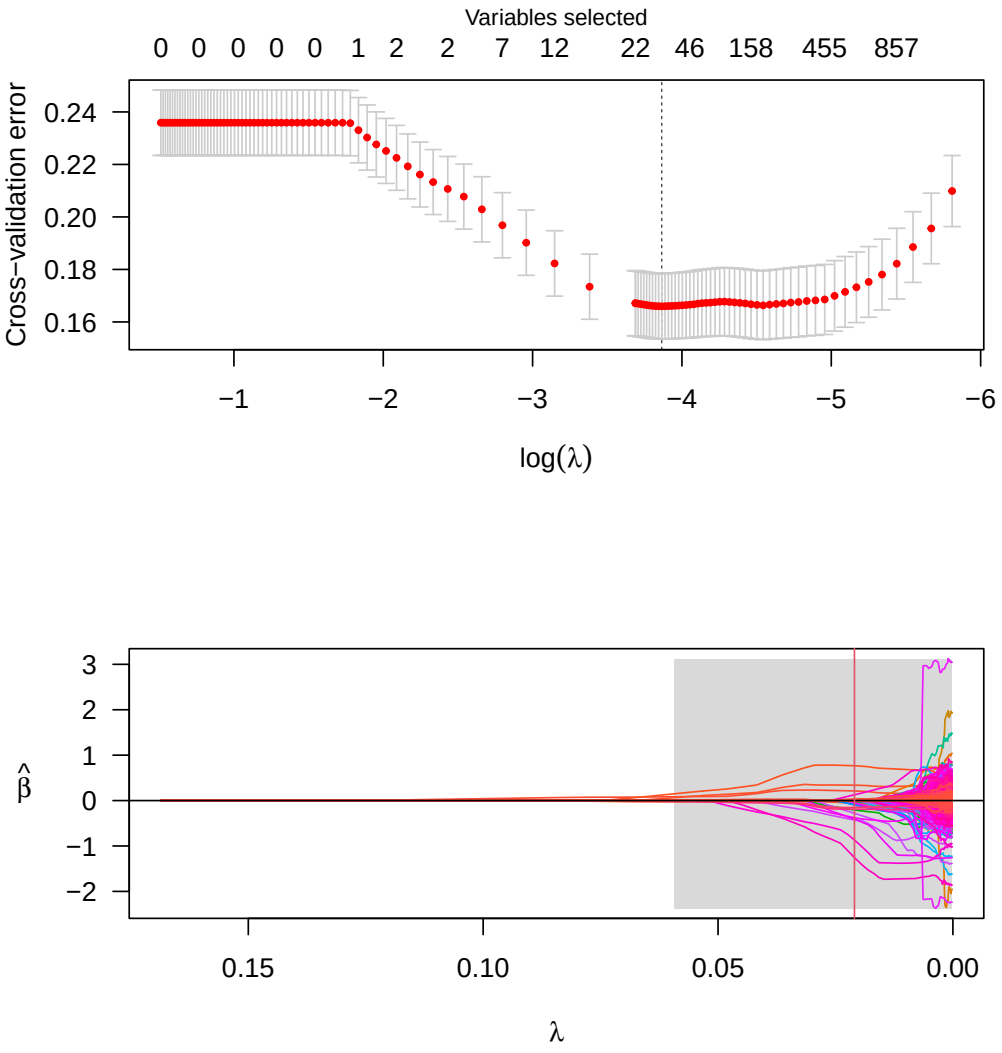


Figure 5.6: 5-fold CV MSE and coefficients paths for SCAD as a function of $\log(\lambda)$.

5.6 Models with interactions

Taking into account the potential impact of collaborations, we consider several models that allow for first order interaction effects. Due to the large dimensionality of the dataset, we restrict attention to a selected subset of variables from both the training and test sets. Specifically, we selected 480 actors, 130 directors, 30 writers, 25 composers, and 10 words with most appearances across all observations. In particular, we implement LASSO and Elastic Net regressions.

For the LASSO model, the regularization parameter λ is selected via cross-validation. When estimated on the training set, the resulting model shows a test-set MSE of 0.19065. This value is higher than the MSE obtained by the LASSO model without interaction terms, indicating that the inclusion of second-order interactions does not improve predictive performance in this case. We will discuss this result further.

Using cross validation for both, α and γ, we estimated the Elastic Net on the training set. The fitted model shows a test-set MSE of 0.1901.

Looking at the mean squared errors, we can clearly see that these two models perform worse than LASSO regression and Elastic Net without interactions. This result was expected, considering that the number of actors and writers present in the sub-dataset is small. It would be interesting to estimate the models on the complete dataset using more advanced computational resources in order to capture the effects of all interactions.

5.7 Group LASSO

Since our covariates are naturally divided into groups, we decided to fit a group LASSO model. Unlike the standard LASSO, the L1 penalty is applied to the L2 norm of groups of coefficients, rather than to individual coefficients. As a result, as the tuning parameter λ increases, the norm of an entire group of coefficients can be pushed exactly to zero. This means that all coefficients belonging to that group will be set to zero, allowing for selection at the group level rather than at the level of single variables.

The following groups are considered:

- Group 1: covariates referring to movie genres;
- Group 2: covariates referring to actors;
- Group 3: covariates referring to directors;
- Group 4: covariates referring to screenwriters;
- Group 5: covariates referring to composers;
- Group 6: covariates referring to production countries;
- Group 7: covariates referring to studios;
- Group 8: covariates referring to content ratings;
- Group 9: covariates referring to prizes won by the cast before the release of the movie;
- Group 10: covariates referring to words present in the movie titles;

- Group 11: covariates referring to time encoding;
- Group 12: covariates referring to the budget levels;
- Group 13: singlet group composed by the covariate `runtimeMinutes`.

Since the group LASSO model is a computationally heavy model, after several attempts, we decided to reduce the number of columns. In particular, we considered the 150 actors, the 80 directors, the 20 writers, the 10 composers, the 15 words, the 10 studios, the 5 content ratings, the 7 countries and the 10 movie genres with the highest number of appearances. The train set and test set obtained from this secondary dataset have been split using the same seed as the original dataset, in order to maintain comparability with the other models.

```
col_names = colnames(train2)[-1]

# groups creation
group = rep(NA, ncol(train2)-1)
group[grep("^genre_",    col_names)] = 1
group[grep("^actor_",    col_names)] = 2
group[grep("^director_", col_names)] = 3
group[grep("^writer_",   col_names)] = 4
group[grep("^composer_", col_names)] = 5
group[grep("^country_",  col_names)] = 6
group[grep("^studio_",   col_names)] = 7
group[grep("^rating_",   col_names)] = 8
group[c(which(col_names=="oscars_before"),
        which(col_names=="total_awards_before"))] = 9
group[grep("^time_",   col_names)] = 10
group[grep("^word_",   col_names)] = 11
group[grep("^budget_",   col_names)] = 12
# singlet group
unassigned_idx = which(is.na(group))
group[unassigned_idx]=13
```

5.7.1 Model regularization

The optimal value of the tuning parameter $\lambda = 0.00019$ was identified minimizing the prediction error via 4-fold cross-validation on the reduced train set, utilizing a user selected grid of 30 values of λ. Despite having reduced the number of columns and having considered a small number of values of lambda, the `cv.gglasso` function still struggled to fit models for really small values of λ, thus the tolerance of the algorithm had to be decreased to 1e-4.

```
library(gglasso)
lambda.values <- 10^seq(-5, 0.3, length = 30)
set.seed(1)
cv.grouplasso <- cv.gglasso(x = X.train2, y = train2$y,
                            group = group,eps = 1e-4,
                            lambda = lambda.values,
                            nfolds = 4)
```

After having identified the optimal value of lambda, we fit the model on the whole train set,

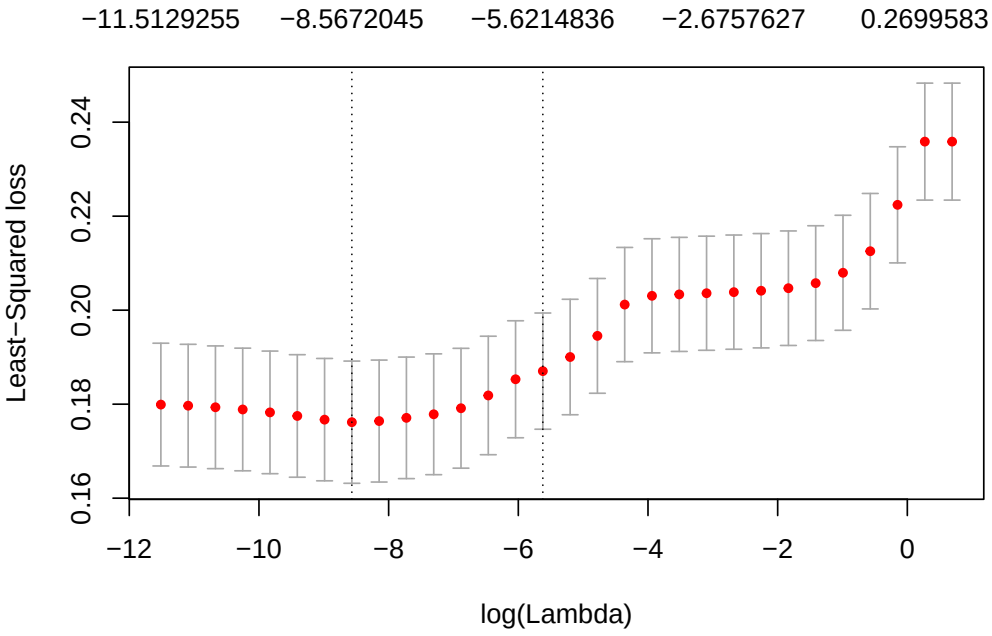


Figure 5.7: 4-fold CV MSE as a function of $\log(\lambda)$.

and we compute the prediction error on the test set. Unsurprisingly, the prediction error of this model (MSE = 0.20571) is worse than the prediction error of the LASSO model: in fact, the optimal lambda is small enough that all of the groups are included, thus there is no sparsity. This suggests us that all groups contain at least some important variables and it's better to include all of predictors rather than exclude the whole group. Moreover, the prediction error of the group LASSO model is higher than that of the ridge model, since the number of columns had to be reduced in order to fit this specific model.

5.7.2 Interpretation of the results

The group LASSO model sets entire groups of coefficients to zero depending on the value of the penalty parameter λ. Our goal is to evaluate the importance of whole groups of variables by checking which groups remain active for different values of λ.

From the heatmap and the coefficients' paths in Figures 5.8-5.10, we can see that the most persistent groups are the movie genre, the runtime of the movies and the time-related variables, such as the year and month the film was released. On the other hand, the less persistent groups are the directors and the writers. This result is likely influenced by group size. Large groups seem to be pushed to zero more quickly, even when they include several coefficients that were found to be important by other sparse models, such as the actor group. On the other hand, smaller groups seem to remain active for larger values of the penalty, even if they contain fewer relevant variables. This behavior may be partly explained by the fact that `gglasso` automatically applies group weights equal to the square root of the group size, in order to correct for differences in group numerosity. However, since the groups in our analysis are extremely unbalanced in size, it is likely that very large groups have been over penalized.

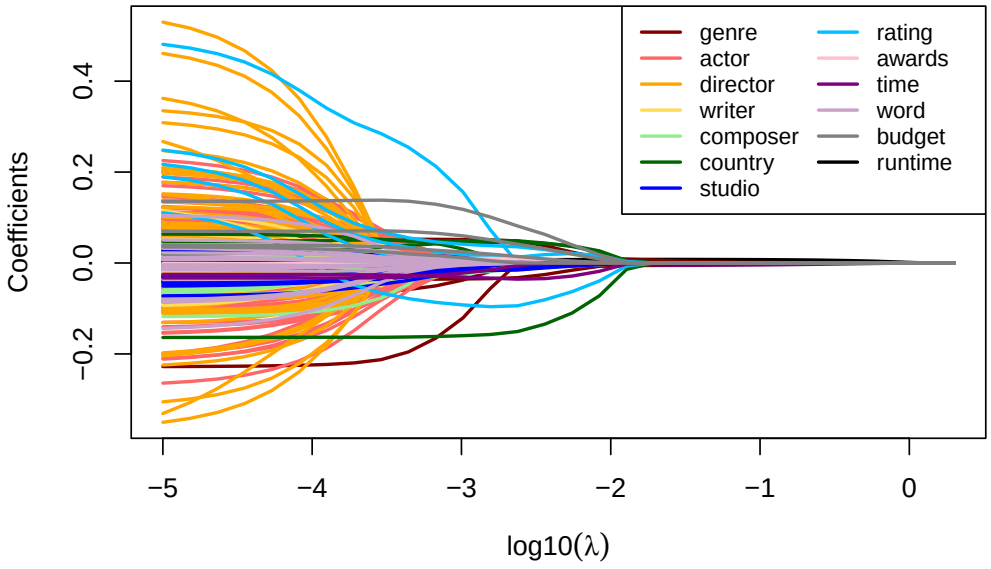


Figure 5.8: Group LASSO coefficients paths as a function of $\log_{10}(\lambda)$, colored by group.

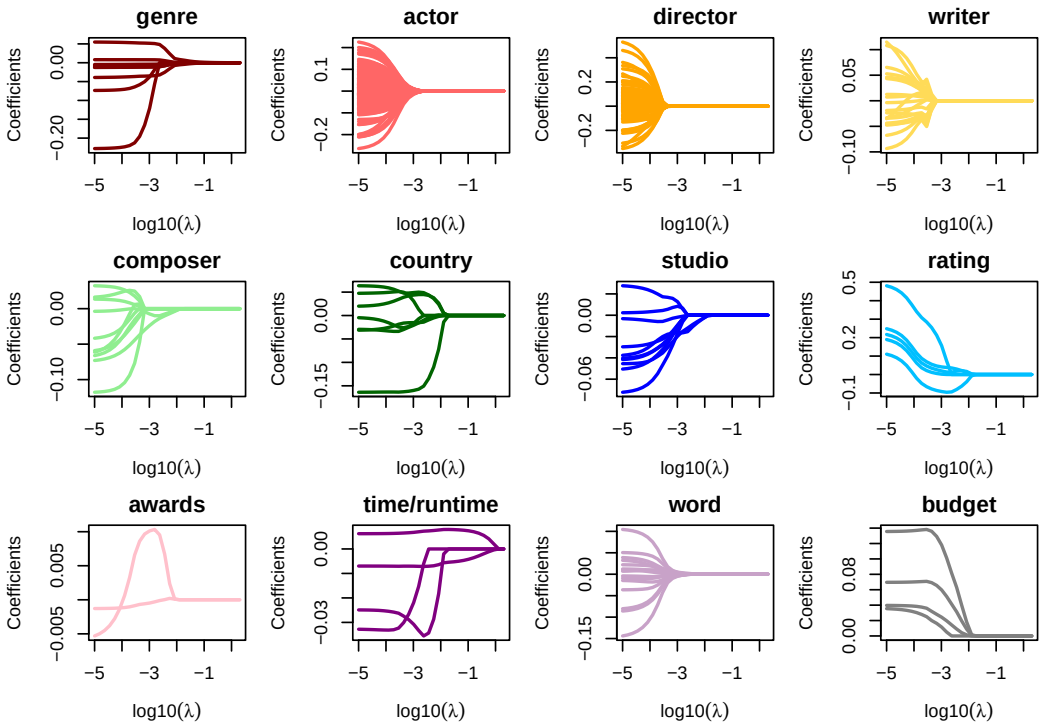


Figure 5.9: Group LASSO coefficients paths as a function of $\log_{10}(\lambda)$, divided by group.

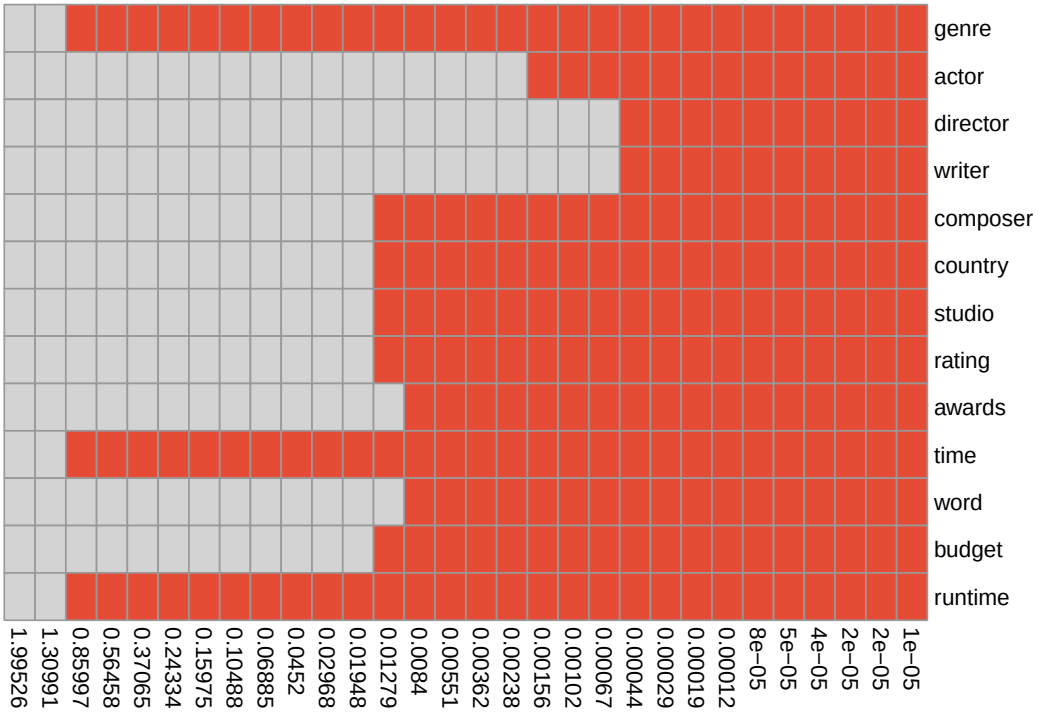


Figure 5.10: Heatmap of active groups (identified by the color red) according to Group LASSO model for different values of λ.

5.8 Regression tree

A regression tree was fitted, a non-parametric model that can automatically capture non-linear effects and interactions between predictors. Although its predictive performance is not particularly good if compared to more complex models, the tree is highly interpretable and can suggest the importance of a variable.

5.8.1 Model regularization

The optimal number of leaves J for the tree was selected using a growing and pruning procedure, aiming to balance model complexity and predictive power. First, a saturated tree is fitted, with deviance close to zero. Then, starting from the full tree, the least important leaves (i.e. those causing the smallest increase in deviance) are removed one by one.

Figure 5.9 shows how the relative cross-validated error changes with J. In this case, the deviance reaches a global minimum for J=28.

```
library(rpart)
mod.tree.sat <- rpart(y ~ . ,data = train, method = "anova",
                      control = rpart.control(cp = 1e-5,
                                              minsplit =  2,
                                              minbucket = 1,
                                              xval = 10))
```

After tuning the parameters, the regression tree was pruned, and we computed the prediction error on the test set, which is equal to 0.21695.

5.8.2 Interpretation

Variables that appear in the first splits of the tree, such as the runtime, or the drama genre are often the most influential for dividing the data. However, this does not necessarily mean they are the most important variables for explaining the overall phenomenon. A covariate can appear in multiple splits, and what really matters is the effect of a specific split point, rather than the presence of the variable in general.

From this plot, we can deduce that the runtime, some specific genres and the release year have a strong impact on the rating. On the other hand, we can see that variables like the content rating, the writers, the composers, the words in the title, the production studio and the month of release do not appear at any split.

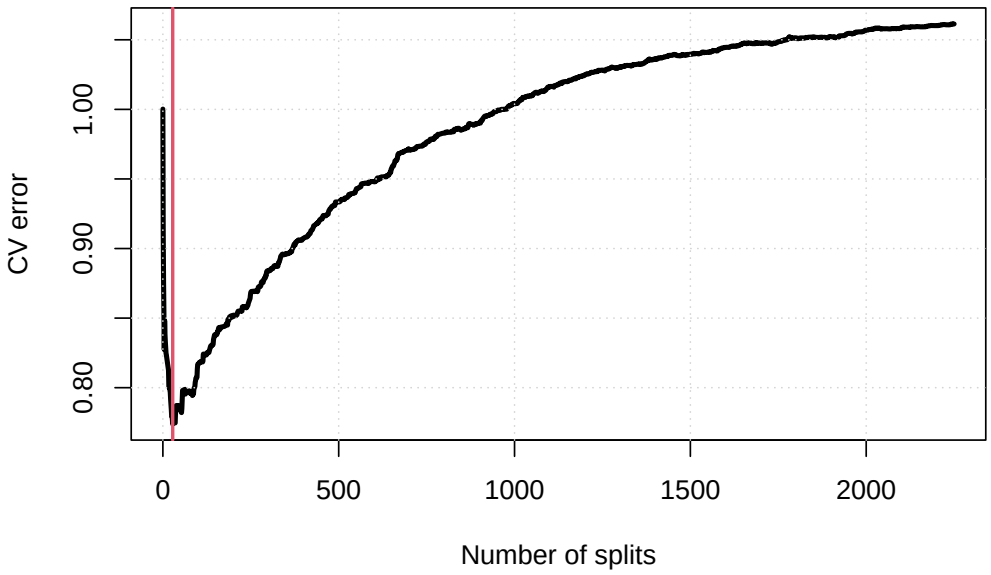


Figure 5.11: Relative (to the null model) CV-error of the regression tree as a function of the number of splits.

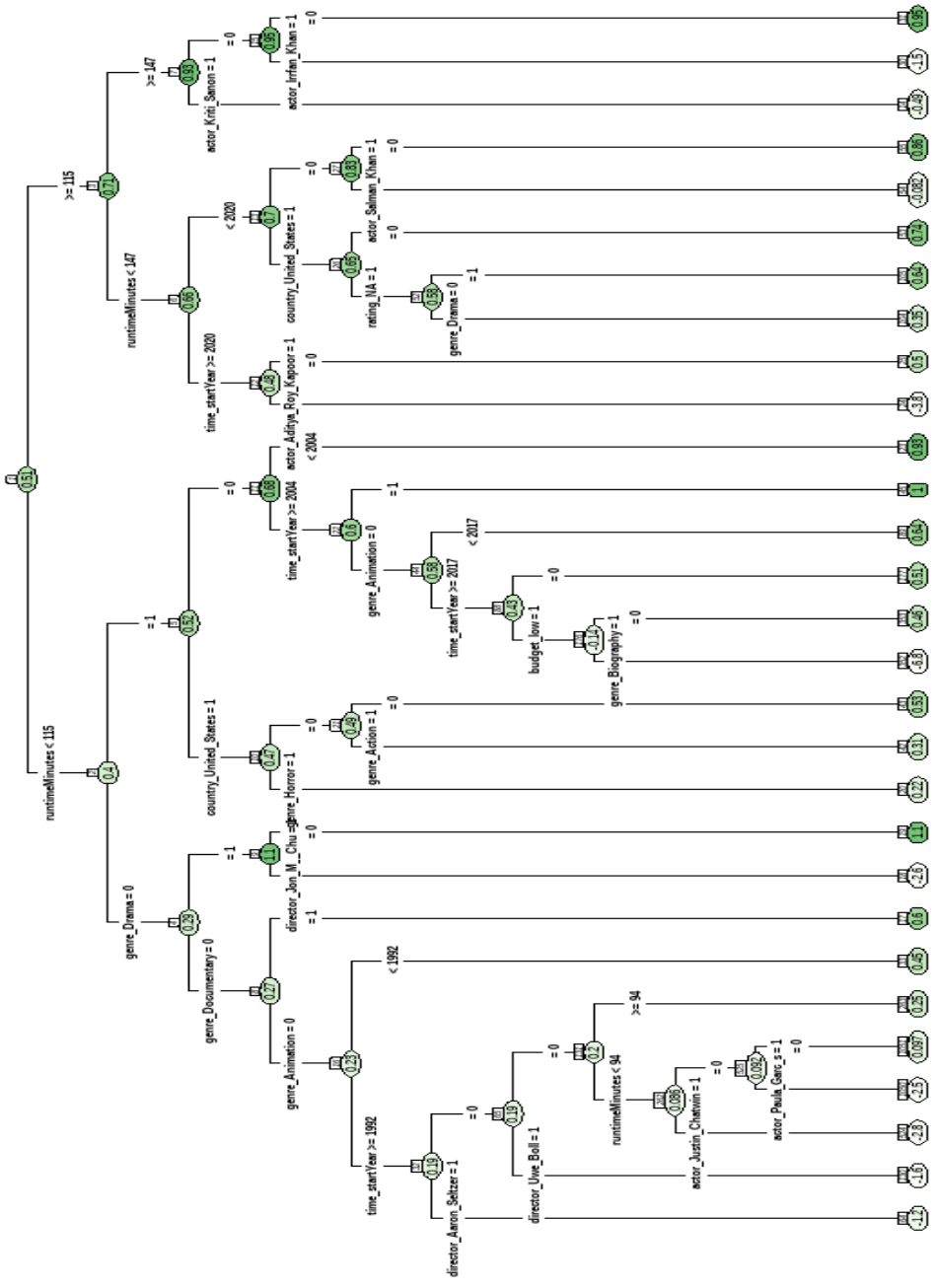


Figure 5.12: Regression tree.

5.9 Random Forest

A Random Forest model was fitted. This model combines a large number of regression trees, which are known to be highly unstable and variable, meaning that small changes in the data can lead to large differences in the fitted model. By aggregating many trees, this approach reduces the overall variance.

Unlike other ensemble methods, Random Forests reduce the correlation between individual trees by randomly selecting a subset of variables at each split. This leads to trees that are very different from each other. As a result, the method can effectively capture complex and non-linear relationships between the predictors and the response variable, without requiring specific assumptions about the functional form of the model.

5.9.1 Model regularization

The main tuning parameters of the trees are the number of covariates sampled at each split and the total number of trees in the forest: this last parameter does not minimize the error at a specific value, but the error tends to stabilize as the number of trees increases. The first plot shows the behavior of the out-of-bag (OOB) error (that is, the error evaluated on observations not sampled at each iteration) as the number of trees increases, for different values of the number of sampled predictors. The two values of the number of sampled predictors that (almost uniformly) minimize the out-of-bag error are 300 and 600. Since they seem to give similar results, we choose the simpler model with `mtry=300`. Moreover, the OOB error appears to stabilize after about 500 trees. Therefore, we choose this value for the number of trees.

```
library(ranger)
mtry.values <- c(3,15,40,80,150,300,600,1200,2400)
trees.values <- c(50, 100, 200, 300, 500,1000)

ERR.randfor <- matrix(NA, nrow = length(trees.values),
                      ncol = length(mtry.values))

# Grid search on mtry and trees.values
for (v in 1:length(mtry.values)) {for (a in 1:length(trees.values)) {
  cat('\n',mtry.values[v])
  mod.randfor.temp = ranger(y ~ ., data = train,
                            mtry = mtry.values[v],
                            num.trees = trees.values[a])
  ERR.randfor[a, v] = mod.randfor.temp$prediction.error
}}

best_mtry_idx <- which.min(apply(ERR.randfor, 2, min))
best_mtry <- mtry.values[best_mtry_idx]

best_num.trees <- 500
```

```r
# Final model
mod.randfor <- ranger(y ~ ., data = train,
                      mtry = best_mtry,
                      num.trees = best_num.trees,
                      importance = "impurity",
                      seed = 1)
```

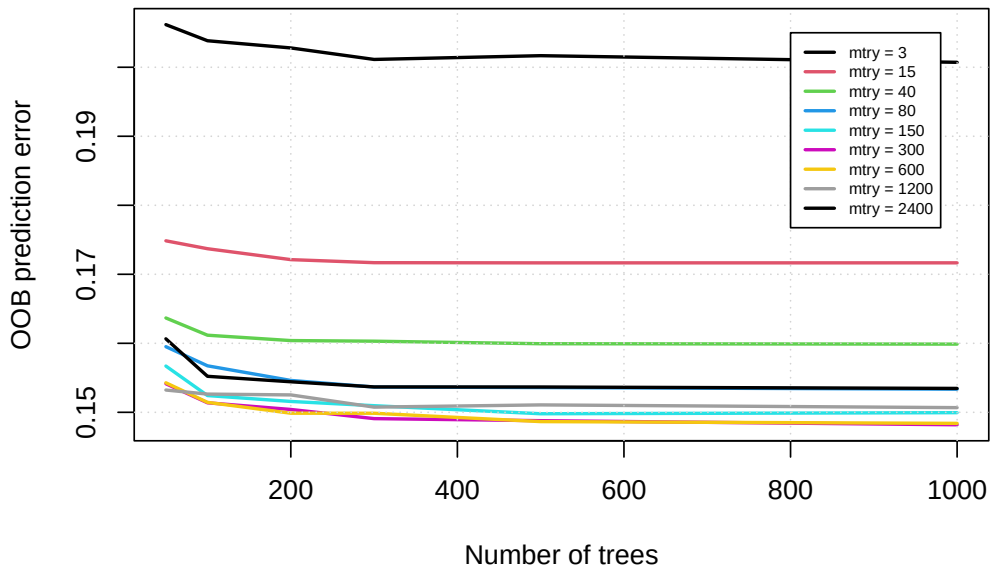


Figure 5.13: OOB prediction error as a function of the number of trees, for different values of `mtry`.

After tuning the parameters, we fit the tuned model to the whole training set, and we compute the prediction error on the test set. The mean squared error of the random forest model on the test set is 0.1757, showing that this model is one of the most effective ones thus far from a predictive point of view. However, the prediction error is only slightly better than the one of sparse and penalized linear models, such as LASSO and elastic net.

5.9.2 Interpretation

Variable importance was evaluated using the permutation method on the out-of-bag data. This method measures the increase in prediction error when the values of a predictor are randomly shuffled, breaking the relationship between that variable and the response. This allows us to identify which covariates are most important for the model. In this case, the most relevant variables are the runtime, several genres (especially the drama and horror genre), the release year, whether the movie was produced in the United States and the

number of awards won before the release of the movie by the cast.

However, this method only provides a numerical measure of importance and does not indicate the functional form (linear, quadratic, etc.) and, most importantly, the direction of the relationship between response and predictors. For example, we see that the horror genre is one of the most important predictors: one might wrongly assume that it is positively linked with the IMDb rating of the movie but, in reality, as it was seen from the previous models, it is actually strongly negatively linked.

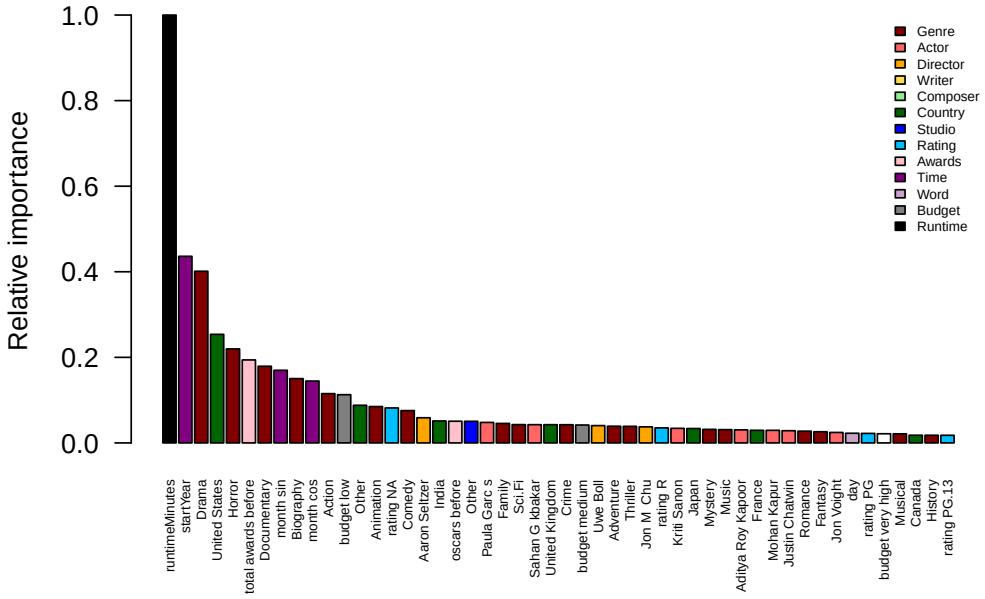


Figure 5.14: Relative variable importance - Random Forest.

5.10 Gradient tree boosting

Gradient tree boosting is an ensemble learning technique that builds a predictive model by combining a large number of weak learners. Each new tree is trained to correct the errors of the previous ones by minimizing a specified loss function through gradient descent.

XGBoost (*Extreme Gradient Boosting*) is an efficient and regularized implementation of Gradient Tree Boosting. It extends the standard framework by introducing regularization on tree complexity, shrinkage, column subsampling, and second-order optimization, improving both predictive performance and robustness. Moreover, XGBoost is specifically designed to handle sparse and high-dimensional data efficiently. Hyperparameters tuning was conducted via a grid search over three key parameters: the learning rate η, the maximum tree depth `max_depth`, and the number of boosting iterations `nrounds`. The hyperparameters' grid was

constructed using heuristic criteria. These values were combined to form a three-dimensional grid, where each cell of the 3D array corresponds to a specific triplet of hyperparameter values (η_i,$\texttt{max\_depth}_j$,$\texttt{nrounds}_k$). Model selection was carried out using cross-validation, and the optimal configuration was chosen based on the minimization of the root mean squared error. The learning rate controls the contribution of each tree to the overall model and was set to 0.05 to reduce overfitting. The parameter `max_depth` = 6 determines the maximum depth of each decision tree and therefore governs model complexity. At the end, the number of boosting iterations was set to 200. For computational reasons, hyperparameters tuning was carried out using a sparse model matrix, as recommended for XGBoost in high-dimensional contexts. Since XGBoost natively exploits sparsity [Chen and Guestrin, 2016], this approach significantly reduced computational cost and memory requirements without affecting the results. In addition, row and column subsampling were employed by setting `subsample` = 0.5 and `colsample_bytree` = 0.8. These parameters introduce randomness during tree construction, which helps reduce variance and overfitting while further improving computational efficiency.

```
# Gradient Boosting tuning

library(xgboost)
library(Matrix)

formula <- ~ . - 1

X_train_mat_tuning <- sparse.model.matrix(
  formula,
  data = train |> select(-y)
)

X_train_mat <- model.matrix(
  formula,
  data = train |> select(-y)
)

X_test_mat <- model.matrix(
  formula,
  data = test |> select(-y)
)

dtrain_tuning <- xgb.DMatrix(
  data = X_train_mat_tuning,
  label = train$y
)

dtrain <- xgb.DMatrix(
  data = X_train_mat,
  label = train$y
)
```

```r
dtest <- xgb.DMatrix(
  data = X_test_mat,
  label = test$y
)

eta_grid       <- c(0.05, 0.1, 0.2, 0.3)
max_depth_grid <- 2:6
nrounds_grid   <- c(seq(100, 900, by = 100),
                    seq(1000,19000, by = 1000))

grid <- expand.grid(
  eta = eta_grid,
  max_depth = max_depth_grid,
  nrounds = nrounds_grid
)

results <- vector("list", nrow(grid))

set.seed(123)

K <- 4
n <- nrow(X_train_mat)

fold_id <- sample(rep(1:K, length.out = n))

folds <- lapply(1:K, function(k) {
  which(fold_id == k)
})

for (i in seq_len(nrow(grid))) {

  cat("Running:",
      "eta =", grid$eta[i],
      "max_depth =", grid$max_depth[i],
      "nrounds =", grid$nrounds[i], "\n")

  params <- list(
    objective = "reg:squarederror",
    eval_metric = "rmse",
    eta = grid$eta[i],
    max_depth = grid$max_depth[i],
    subsample = 0.5,
    colsample_bytree = 0.8
  )

  cv <- xgb.cv(
    params = params,
```

```
    data = dtrain_tuning,
    folds = folds,
    nrounds = grid$nrounds[i],
    verbose = 0
  )

  rmse_last <- tail(cv$evaluation_log$test_rmse_mean, 1)

  results[[i]] <- data.frame(
    eta = grid$eta[i],
    max_depth = grid$max_depth[i],
    nrounds = grid$nrounds[i],
    rmse = rmse_last
  )
}

cv_summary <- bind_rows(results) |>
  arrange(rmse)
```

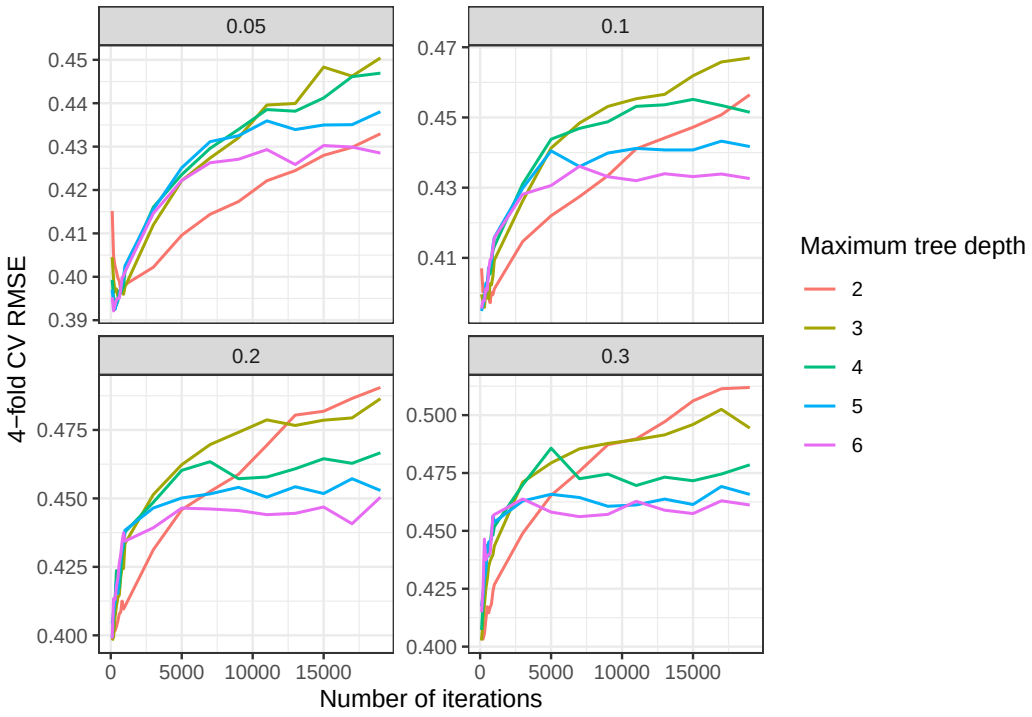


Figure 5.15: 4-fold CV RMSE for 4 different values of η and maximum tree depth, as a function of the number of boosting iterations.

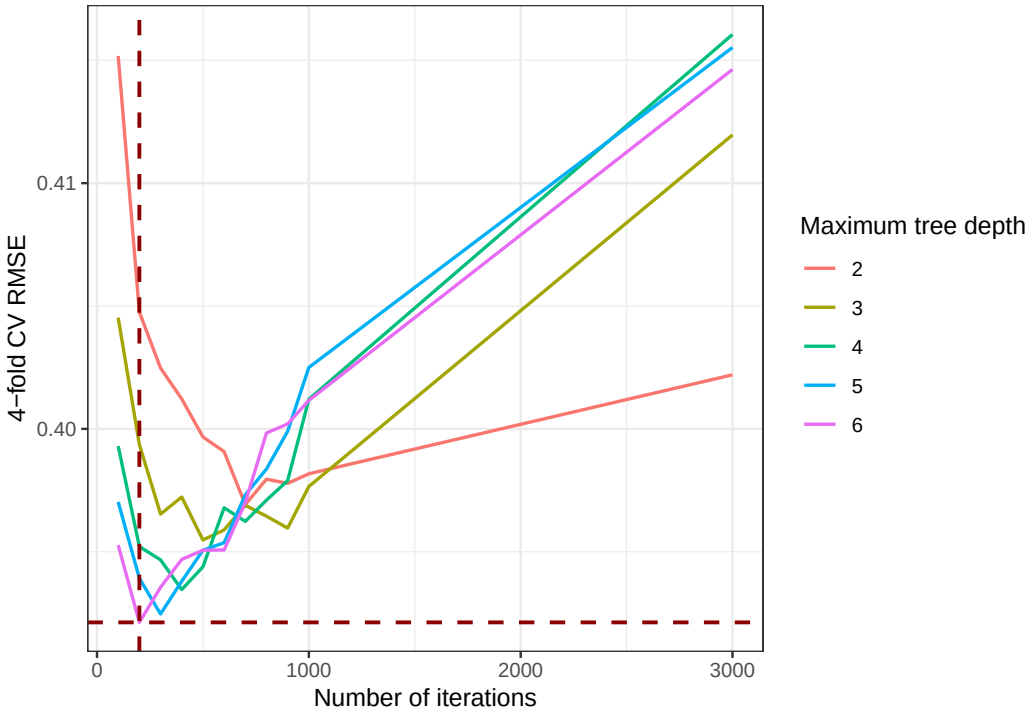


Figure 5.16: 4-fold CV RMSE for $\eta = 0.05$ and maximum tree depth, as a function of the number of boosting iterations.

```r
best <- cv_summary[1, ]

params_best <- list(
  objective = "reg:squarederror",
  eval_metric = "rmse",
  eta = best$eta,
  max_depth = best$max_depth,
  subsample = 0.5,
  colsample_bytree = 0.8
)

set.seed(1)
xgboost.tuned <- xgboost(
  data = dtrain,
  params = params_best,
  nrounds = best$nrounds,
  verbose = 0,
)
```

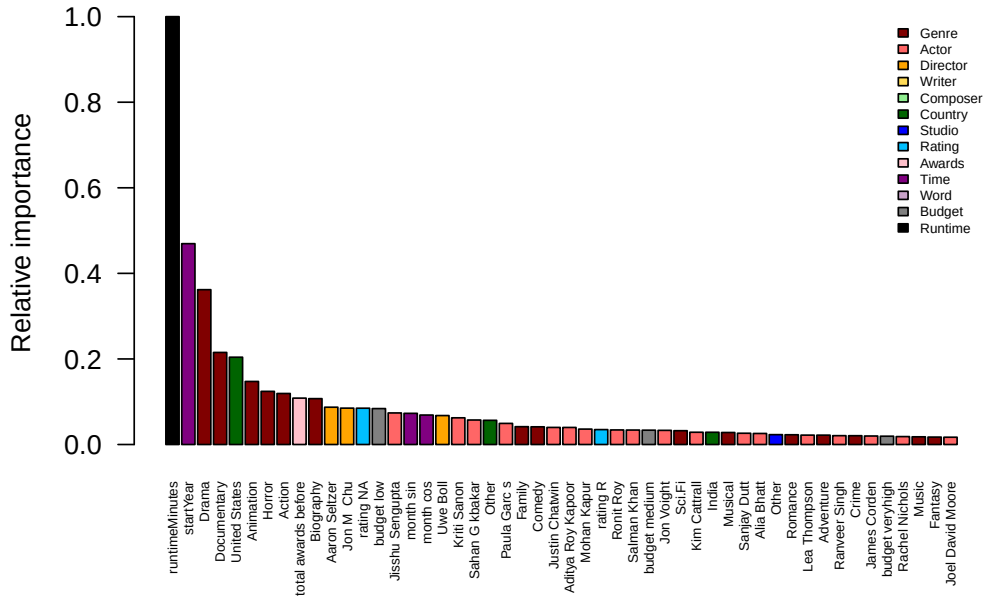


Figure 5.17: Relative variable importance - Gradient Boosting.

The final model was then estimated on the training set using the XGBoost framework and evaluated on an out-of-sample test set. Figure 5.17 shows that the most predictive variables are `runtimeminutes`, `time_startyear`, and `genre_Drama`. A comparison with the

variable importance from the Random Forest model reveals substantial agreement, with both methods identifying essentially the same variables as the most important predictors. Predictive accuracy was assessed using the mean squared error, yielding a test error of 0.17336, which indicates the best so far out-of-sample performance.

5.11 Neural Network

We implemented a feed-forward neural network with a single hidden layer and ReLU activation functions. Model selection was done via 4-fold cross-validation over a grid of hyperparameters, considering 6, 7 and 8 hidden units and a sequence of weight decay values in the interval $[10^{-4}, 10^{-2}]$.

To reduce sensitivity to random initialization and in order to avoid the convergence to local minima, multiple seeds were used within each fold. The optimal configuration selected 8 hidden units and a weight decay parameter equal to 0.000464. The final model was re-estimated on the full training set again using several random initializations, and the best-performing network was evaluated on the test set, achieving a test-set MSE of 0.1794. Better results could potentially be achieved by accounting for sparsity during neural network training. In this respect, LassoNet [Lemhadri et al., 2021] represents a suitable alternative, as it integrates feature selection into neural networks through an L1 penalization. However, in the present setting, fitting LassoNet model requires huge computational resources, which makes its application less feasible.

Section 6
Conclusions

6.1 Models comparison

We fitted several models. Some of them showed very strong predictive performance, such as the Random Forest, gradient boosting, and the neural network, but they offered limited interpretability. Other models are easier to interpret, but have weaker predictive accuracy.

Model	**MSE**
XGBoost	**0.1734**
Random Forest	0.1757
Neural Network	0.1794
LASSO	0.1828
Elastic Net	0.1841
Elastic Net with interactions	0.1901
LASSO with interactions	0.1906
MCP ($\gamma = 3$)	0.1953
Ridge	0.1962
SCAD ($\gamma = 3$)	0.1988
Group LASSO	0.2057
Regression Tree	0.2169

Table 6.1.1: Comparison of predictive performance on the test set

In this setting, simple linear models that assume sparsity perform surprisingly well. In particular, the LASSO model has a prediction error that is only slightly worse than that of much more complex and less interpretable models. This suggests that the main assumptions behind penalized linear models are reasonable: the relationship between the response and the predictors is approximately linear, and most coefficients are exactly zero.

6.2 Prediction of *The Odyssey*

From a purely predictive point of view, the best-performing model is gradient boosting. Therefore, we choose this model to predict the IMDb rating of The Odyssey (2026), the upcoming movie produced by our client, directed by Christopher Nolan and starring Matt Damon, Anne Hathaway, Robert Pattinson, Zendaya, Lupita Nyong'O and Charlize Theron among others. The movie is an adaptation of the world known poem by Homer, it will be released in theaters in July 2026 and it is rumored to be over three hours long. Our best fitting model, gradient boosting, predicts a rating of 7,62 out of 10, which lies in the higher range.

```
#information about The odyssey (2026)
x_odyssey <- c(
  time_startYear = 2026,
  runtimeMinutes = 180,
  genre_Drama = 1,
  genre_Action = 1,
  genre_Adventure = 1,
  genre_History = 1,

  actor_Matt_Damon = 1,
  actor_Tom_Holland = 1,
  actor_Anne_Hathaway = 1,
  actor_Elliot_Page = 1,
  actor_Zendaya = 1,
  actor_Robert_Pattinson = 1,
  actor_Lupita_Nyong_o = 1,
  actor_Charlize_Theron = 1,
  actor_Jon_Bernthal = 1,
  actor_Benny_Safdie = 1,
  actor_John_Leguizamo = 1,
  actor_Himesh_Patel = 1,
  actor_Will_Yun_Lee = 1,
  actor_Mia_Goth = 1,
  actor_Jimmy_Gonzales = 1,
  actor_Corey_Hawkins = 1,
  actor_Shiloh_Fernandez = 1,
  actor_Samantha_Morton = 1,
  actor_Rafi_Gavron = 1,
  actor_Bill_Irwin = 1,
  actor_Logan_Marshall_Green = 1,
  actor_James_Remar = 1,

  director_Christopher_Nolan = 1,
  writer_Christopher_Nolan = 1,
  composer_Ludwig_Goransson = 1,

  country_United_Kingdom = 1,
```

```
  country_United_States = 1,
  studio_Universal_Pictures = 1,
  rating_R = 1,
  budget_veryhigh = 1,
  total_awards_before = 300,
  oscars_before = 9
)
model_features <- colnames(test)[-1]

x_odyssey_df <- as.data.frame(
  matrix(0, nrow = 1, ncol = length(model_features))
)
colnames(x_odyssey_df) <- model_features

common_vars <- intersect(names(x_odyssey), colnames(x_odyssey_df))
x_odyssey_df[1, common_vars] <- x_odyssey[common_vars]

x_odyssey_df <- x_odyssey_df[, model_features]

X_test_mat_odyssey <- model.matrix(
  formula,
  data = x_odyssey_df
)

x_odyssey_xgb <- xgb.DMatrix(
  data = X_test_mat_odyssey
)

pred.odyssey <- predict(fit_final, x_odyssey_xgb)

pred.odyssey.orig <- plogis(pred.odyssey)*9 + 1
pred.odyssey.orig
```

```
## [1] 7.62275
```

6.3 Interpretation of the results and final recommendations

From an interpretative point of view, our results allow us to provide several strategic suggestions to our client on how to produce a critically successful movie. Overall, genre plays a key role: dramatic and realistic movies, as well as documentaries, tend to receive higher IMDb ratings, while horror and less realistic genres perform worse. Interestingly, animated movies also receive substantial critical praise. In addition, longer movies appear to be more appreciated by critics.

The choice of director is also crucial. Movies directed by well-established and critically acclaimed filmmakers, such as Christopher Nolan, Quentin Tarantino, George Lucas, Edgar

Wright, Alfonso Cuarón, Denis Villeneuve, James Cameron, Guy Ritchie, Anthony Russo, Bryan Singer, and James Gunn, tend to achieve higher ratings. When it comes to animated films Brad Bird appears to be particularly influential.

Regarding actors, several non-American actors, especially Indian actors, are strongly associated with higher IMDb ratings. Examples include Sushant Singh Rajput, Arshad Warsi, Saurabh Shukla, and Tamannaah Bhatia. This effect may be partly explained by the global nature of IMDb and the large Indian user base, which may favor actors from their own country. If our client is interested in expanding into the Indian market (which is large, growing, and becoming richer) casting well-known Indian actors could both increase IMDb ratings and help reach a market not yet fully exploited by Hollywood, potentially at a lower cost than hiring top Hollywood stars. Among Western actors, Ian McKellen, Zendaya, Mahershala Ali, Daniel Kaluuya, Jeremy Renner, and Hugo Weaving are positively associated with high ratings, while, surprisingly, most mainstream Hollywood stars do not appear among the top coefficients.

On the other hand, our analysis suggests avoiding certain actors and directors. Actors such as Taylor Lautner, Madonna, and James Corden are linked to lower ratings. In some cases, these are celebrities rather than professional actors: although popular, they are frequently associated with low-quality blockbuster films rather than critically acclaimed productions. For directors, Uwe Boll and Aaron Seltzer stand out as particularly negative examples, likely due to their history of poorly received films. Other names, such as Jon M. Chu and Alan Cumming, also appear on the negative side, even if their reputations are less extreme.

Section 7
Contributions

The contributions of each member of the group to the final project are the following:

- **Davide Bortoletto**: Gradient Boosting model, dataset creation through Wikidata scraping, dataset cleaning process, drafting of the graphs, drafting of the final Markdown document;
- **Bryan Patarini**: Gradient Boosting model, dataset creation through Wikidata scraping, dataset cleaning process, exploratory analysis;
- **Gianmarco Rosa**: LASSO model, Ridge model, Elastic Net model, SCAD model, MCP model, LASSO model with interactions, Elastic Net model with interactions, Neural Network model with PyTorch, dataset cleaning process;
- **Greta Schiappacasse**: Group LASSO model, Random Forest, Regression Tree, dataset creation from IMDb bulk files and Wikidata scraping, dataset cleaning process, drafting of the introductory and final sections of the report.

Bibliography

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794. ACM, August 2016. doi: 10.1145/2939672.2939785. URL http://dx.doi.org/10.1145/2939672.2939785.

Ismael Lemhadri, Feng Ruan, Louis Abraham, and Robert Tibshirani. Lassonet: A neural network with feature sparsity. *Journal of Machine Learning Research*, 22(127):1–29, 2021. URL http://jmlr.org/papers/v22/20-848.html.