Gabriel Fischer
Raphaël Rebêche

Group 4
2025-2026

Final Report of DALAS Project : Movie recommendation algorithm

January 2026

1. Introduction

a. General introduction

The Letterboxd platform allows its users to browse the entire catalog of released and upcoming movies. Its biggest advantage is its social networking aspect, enabling users to add each other as friends, see the movies their friends have wishlisted, and view the ratings their friends have given to the movies they've watched.

However, this platform has one drawback: with the sheer number of movies available, it can sometimes be difficult to know which ones to watch. When you arrive at the main page of your favorite streaming platform, it instantly offers recommendations based on what you like and watch most. This isn't the case with Letterboxd, which, while a vast catalog and social network, doesn't offer recommendations based on the movies users have watched and rated on the platform.
This is where our project comes in. Our goal is to address this shortcoming and offer users a movie recommendation algorithm based on their preferences.

b. Challenges

Our main challenges were as follows:

- How will we retrieve the movie pages and all the data they contain from Letterboxd?

- What is the best way to represent our data? How should we convert it so that descriptors like "Brad Pitt," "Science Fiction," or "2015" are comparable?

- How will we retrieve data about users, the movies they have watched, and the ratings they have given them?

- How will we provide recommendations consistent with the movies the user has preferred?

c. Objectives

Faced with these challenges, we decided to approach the project as follows.

We decided to create an embedding space for the platform's first 100,000 movies in the popular tab.
A movie embedding space is a finite-dimensional vector space in which each movie in the catalog is represented by a dense numerical vector derived from the mean of embeddings of the tokens describing the movie, such as genres, metadata, and associated semantic information. The goal of this representation is to encode these characteristics in a compact and usable form.
The geometric structure of the space is designed so that the distance or similarity between two vectors reflects the semantic proximity between the movies: movies sharing similar characteristics are close in the space, while movies with different content or themes are far apart. This space thus provides a common basis for performing comparison operations, searching for near neighbors, and similarity analysis across the entire catalog.

The next step is to perform this same embedding across all the movies the user has saved in their profile to create their average vector. This average vector is weighted by the rating the user gave each movie, so that their favorites have more weight in the vector than those they disliked.
This average vector, thus created, will be projected onto the previously constructed vector space. We can then calculate the distance between our average vector and all the movies in the database to recommend movies to the user that they haven't seen, across all genres, that are closest to their average vector and therefore closest to their preferences.

To assess the relevance of our recommendations, we split the set of movies the user watched into two sets. This was done to create a training set, allowing us to generate the average user rating vector, and a validation set to compare the user's rating of the movie with the recommendation rating our algorithm provided for that movie. We then conducted various statistical tests and visualizations to evaluate the relevance of our model.

2. Collecting the data

a. Searching for an API

In the first place, we searched for an official Letterboxd API which could help us to facilitate the data collecting process. And we found it. However its use cases were rather limited and it could only be used if the platform allowed us to, by motivating our demand in an email. After searching online, we found that this process could be

quite long to be potentially denied access at the end, so we decided to use classic scrapping on the platform's web pages.

b. Explanation of the scrapping process

The scraping phase of the project was divided into two main phases: we had to first collect the list of movies that we wanted to study and then we had to collect the data for each of those pages.

The first phase of the scraping part was to collect all the links to the movie pages. We have done so by browsing the pages of the most popular movies on the platform of all time[1]. Since each of those pages contain 72 movies (thus 72 links), we decided to scrap the first 1400 of these pages for a total of 100 800 pages. This first phase took 6-8 hours to fully finish.

For the second phase, we had to, in the first place, decide which type of data to collect and how much of it. After some tests, we decided that we would collect the first ten links in each of those categories:

- Casting,
- Directors,
- Studios and countries of production,
- Spoken languages,
- Genres (19 existing), themes and mini themes associated[2],
- Release date.

We decided to only take the first ten links of each category for mainly one reason: in some movies, the *cast* section of the page contained sometimes as much as a hundred actors including minor characters and extras, furthermore, in this same section, the actors are sorted from the most important roles to the lesser ones. So by taking only the first ten we ensured that we only kept the main actors, those for which the movie is known for. We applied this rule of only the first ten links to all categories collected to only keep the more relevant data. In addition, we decided not to collect the exact year of release but rather its decade. This choice was motivated by the fact that most audiences express preferences based on the decade level; as people tend to refer more to "1980s movies" rather than "1986 movies".

---

[1] The exact way popularity is calculated on Letterboxd is unknown, however it has been speculated that the more popular movies are those which combine high ratings and engagement on the platform. This has a tendency to put recent movies, which are more rated and commented about, on the first few pages, but this effect is limited over the total number of movies studied.

[2] We can see those types as going from larger sets to smaller ones. For example, genres are the largest groups like `science-fiction` or `romance`, Themes are more precise like `crude-humor-and-satire` or `relationship-comedy` and mini themes are very specific (ex: `thriller-cops-suspense-killer-twist`)

The data was then stored by creating a list of strings for each movie. These strings are composed of a prefix to identify which kind of data it was representing followed by the last last part of the URL on the platform (ex: `GENRE_science-fiction` or `LANGUAGE_portuguese`), this list of attributes was stored in a dictionary where the keys were the last part of the direct URL of the movie page to facilitate its access.
This second scrapping phase was the longest, as it took two days and a half to complete.

c. Encountered problems

During the scrapping phase, we were confronted with two types of pages: those where there was multiple movies displayed and the regular movie pages. For the first type of pages, we were required to wait a non negligible amount of time for the required elements to load, this made for most of the time of the first phase of scrapping. This was not the case for the regular movie pages where all the data that we wanted was loaded directly. To ensure that no data was lost or missed during the second phase and to prevent being blocked from Letterboxd for too many requests, we put in place a system which recorded on disk small parts of the data every 5000 movies in a json file, all files were then merged. Furthermore, all movies that were not correctly scrapped were stored in an error file to try to rescrap later[3]. Finally, the scraping process randomly stopped for a long time to prevent being blocked.

d. Results

At the end of the first scrapping phase, we ended with a total of 100 770 links. This can be explained by some of them seem to have been scrapped multiple times during the first phase[4]. And of these 100 770, 16 have seen their pages removed between the two phases. Bringing the total of 100 707 movies, this accounts for $99.908\,\%$ of the expected number of movies successfully scrapped with 391 783 unique attributes.

3. Vectorization and embedding

To interpret these movies, we used the gensim `Word2Vec` model to convert our strings into a machine interpretable space. This model takes in input a list of sentences (here our lists of attributes) and converts it into a vector of numeric values. This allows us to create an embedding space where a vector is assigned to each word (here our attributes). In this model, words used in a similar context (here, attributes often present in the same movie) will be more similar. This can be seen by

[3] This has proved to be useful since almost a tenth of all movies ended in this error file and, as we saw, only 16 were left in the end.
[4] Since the first phase took multiple hours, it is likely that the platform has re-sorted the list of the most popular movies during the scrapping process.

using a function proposed by the gensim library to view the most similar attributes. `Word2Vec` offers a number of parameters to adjust the model. For our final model, we decided to represent all of our words into a 128-D space. The training parameters for our models are:

- a vector size of 128,
- a window of 100[5],
- an attribute must be present at least 5 times in the dataset to be part of the model
- a negative of 5, which means that each word will be compared to 5 not existing pairs to make sure that it is not making a link between two words
- a sample of 10^{-4} to reduce the importance of recurring values (such as `COUNTRY_usa` or `LANGUAGE_english`)
- The use of Skip-Gram
- and an epoch of 100

The choice of only getting the words that appear 5 times or more in the dataset reduces our total number of attributes to 49 801.

Once we have a vector for each attribute, we want to create a vector for each movie.This can be done by making an average of all of the attributes that a movie is made of. In the end, all attributes and movies are represented in a 128-D space, which is hardly conceivable for a human.

To see proximity between attributes, we can use cosine similarity matrix (see figures, we can see that some words are closer than others. In the Languages matrix (Fig 13) for example, languages like French and Italian are as close as German and French, while an association between German and Malayalam is less common. In the Genres matrix (Fig 10), we can see that a lot of movies are drama and romance, while it is less likely to see a n animated movie about crime. We can also note that in the Decade matrix (Fig 14), the matrix seems to be quite bright around the middle. This can be explained by the fact that elements that influence a movie (like an actor, a director, a studio or a genre…) are only present for so many years and with its influence declining, new attributes are going to replace them after time. On Figure 14 which represents the models' closest attributes to France, we can see a lot of bordering countries and languages, national studios[6]. The same parallels can be made between movie vectors like on Figure 15 where we can see that *Parasite,* a korean movie, with korean actors and directors is far from all the other more "*classic*" popular movies (in english, produced in the USA…)

[5] A window determines how many attributes we want to see around the current one in the same movie. We chose an arbitrarily large one to make sure that all attributes influence the others in the same movie.

[6] It is interesting to see that while France is almost as close to Italy than Germany (and Italian and German) those two are very much less related in the model

To represent all movies in a 2D space, we decided to use UMAP (Uniform Manifold Approximation and Projection for Dimension Reduction), like PCA, it tries to condense a high dimensional space into a lower one while trying to keep meaning. In Figure 15, by taking the first country that appears among several countries, we can see many different clusters, the green and yellow one is related to English movies, while we can see clear pink, blue, black and red clusters for India, France, Germany and Japan. The same thing can be said about Figure 16 where we can again see clear clusters for different languages[7].

4. Modeling and recommendation method

a. User data retrieval

Our goal was now to retrieve data about a username in order to create their average vector. To do this, we only needed their username, which we used to access their profile page to first retrieve the number of movies they had watched.
With this initial data, we could determine the number of pages to scrape from the user's "Watched movies" tab, as these pages always contain 72 movies when complete. Therefore, if the user has watched 200 movies, we would need to scrape $\frac{200}{72} \approx 2.77$pages, meaning two pages with 72 movies each and a final page with $200 - 72 * 2 = 56$ movies.

Next, we could retrieve the links to the pages of all the movies the user has watched, as well as any ratings they may have given them on the same "Watched movies" page. This rating system ranges from 0.5 to 5 stars; however, in the HTML body of the "watched movies" page, this rating is doubled to create whole numbers ranging from 1 to 10. We refer to this as a potential rating because the platform doesn't require users to rate the movies they've watched. This forces us to make a choice regarding these unrated movies. We had two options: either exclude them from the average user rating, or include them and assign them an average rating of 5/10. The first option didn't seem right because it would have meant losing too much data on some profiles, potentially even all the data for some users who prefer not to rate the movies they've watched but simply list them. Therefore, we arbitrarily chose to keep them and rate them 5/10. This choice is debatable, but we concluded it was probably the best way to retain enough data without skewing our evaluation.

b. Vector representation of users

To represent users in vector form, we retrieved, from the links on the "watched movies" page, the tokens which we had already stored in the previous step of

[7] Note that in Figure 16, the green represents Spanish, it does not refer to the UK.

creating the database. This saved us from having to rescrape the pages of movies that the user watched unnecessarily.
If the movie a user watched isn't in our database of 100,000 movies, we ignore it and remove it from the user's watchlist. This seemingly strict policy actually has very little impact because the 100,000 most popular movies on the platform likely encompass all the movies you know, as the last movies on our list sometimes have only one or two views. movies with such few views are often YouTube shorts without a cast, director, or a very limited number of tokens available. It's a scenario that almost never occurs, but we've implemented it as a precaution.
For all the other movies, we retrieve their associated tokens and perform the embedding, in the same way we did to create our vector space.
We use the same model, which we don't need to recalculate because we stored it in a file with the .model extension. This file type is a convention used by some tools or libraries to designate a file containing a trained model. Here, it's a binary serialization file containing a Word2Vec model trained with Gensim, including the vectors, vocabulary, and parameters necessary to reload and use it in Python.
Our user vector is the same size as the movie vectors so that the comparison makes sense.
This vector is constructed as a weighted linear combination of the vectors of movies viewed by the user, with the weights determined by the ratings they assigned. The goal is to ensure that highly rated movies contribute more to the direction of the user vector than those that were less highly rated. Intuitively, a highly rated movie should have a greater influence on the user's representation than a poorly rated one.

Initially, we used linear weighting, in which the weight associated with a movie was directly proportional to its rating: for example, a movie rated 10/10 had a weight ten times greater than that of a movie rated 1/10. This approach proved unsatisfactory, as the resulting recommendations were heavily biased by the movie genres most frequently consumed by the user, regardless of their actual level of appreciation.
To correct this bias, we introduced a non-linear weighting, raising the rating to a power strictly greater than 1. We initially tested a quadratic weighting, which led to a slight improvement in performance. By progressively increasing this power—up to a value of 10 — we observed better results, with movies genuinely enjoyed by the user then having a dominant influence on the user vector and being recommended more frequently.
However, upon reflection, we found that having poorly rated movies contribute positively, even weakly, to the user vector was not always relevant. Indeed, in a semantic vector space, adding the vector of a movie that the user disliked can orient their representation vector in a direction corresponding to characteristics they reject. Therefore, for movies with below-average ratings, we chose to have them contribute to the user vector with a negative weight, which is equivalent to adding their vector in the opposite direction. This modeling aims to more accurately reflect the user's preferences and aversions in vector space.

c. Recommendation method

Now that our user vector has been created, the question was how to compare it to other movie vectors in the vector space. We chose to normalize the movie vectors and the user vector so that we could apply a simple dot product to them. The dot product between two vectors u (user) and m (movie) is defined by :
$u \cdot m = ||u||*||m||*\cos(\theta)$, where $\cos(\theta)$ is the angle between $[0,\pi]$ and is defined as $(u \cdot m) / (||u||*||m||)$.
It therefore combines two components:
- directional alignment (semantic similarity)
- the norm of the vectors (amplitude)

In other words, a movie can obtain a high score either because it is well aligned with the user's preferences or because its embedding has a high norm. Here, since our vectors are normalized, this dot product is equal to the cosine similarity: $u \cdot m = \cos(\theta)$. In this case, the comparison relies solely on directional similarity. In a Word2Vec space, the direction of vectors captures semantics much better than their norm. The dot product (after normalization) therefore seemed consistent with the model's objective.
The cosine similarity scores obtained are therefore within the interval [-1, 1], with a score close to 1 indicating a high similarity between a movie and the user vector. We simply need to sort the calculated similarity scores in descending order to return the x movies most frequently recommended by the algorithm.

However, we made several choices regarding the recommendations to ensure they were pleasant and consistent for the user. We preferred to verify that the movies recommended by the algorithm had not already been seen by the user, as we felt it was inconsistent, and rightly so, to recommend movies the user was already familiar with. Indeed, since these movies were directly used to create the user vector, they were quite frequently recommended by the user, which partly proves that the algorithm does not produce random results. We then observed a fairly predictable but somewhat unsettling behavior: the user's preferred genre occupied almost all of the algorithm's initial recommendations. Indeed, if a user watches movies of similar genres, it's logical that the algorithm, as implemented, would recommend almost exclusively movies of the same genre. This effect is even more pronounced when the genre is specific to itself. For example, animated movies are quite distinct from other genres in the vector space, so if a user primarily watches animated movies, the algorithm would only suggest animated movies. We had two options for addressing this issue. Initially, we could choose to leave these recommendations untouched and simply offer the movies with the highest similarity score without further consideration, assuming that the user primarily enjoys this type of movie. Secondly, and this is the option we chose, we could decide to display the x most recommended movies for each of the 19 genres available on the platform. We found this choice wise because, even if it recommends movies that aren't among those with the highest similarity

score, it allows the user to either stay within their comfort zone and watch the recommended movies for their preferred genre(s), or try other types of movies they're less accustomed to watching and start with those they're most likely to enjoy. (See Figure 1)

5. Evaluation and results

a. Evaluation protocol

To evaluate our model, we chose to divide the movies watched by users into two sets: a training set and a test set. This division was preceded by a random rearrangement of the list of movies watched by the user to avoid introducing biases such as those related to the movies' release dates. We then arbitrarily divided the viewed movies into two sets, with 70% of the training set and 30% dedicated to validation. This ratio was chosen after testing several different ratios, and it seemed to us to be the most relevant for obtaining sufficient data for the test without underfitting the model.
The training set was used to construct the user vector, as described in the previous section, while the validation set was used to compare the model's predictions with the actual ratings provided by users for the different movies.

b. Qualitative Analysis of Recommendation Scores

We chose to display a boxplot for each actual score provided by the user, ranging from 1 to 10. This results in a set of boxplots showing the distribution of the algorithm's predicted ratings for each possible actual user rating. Our goal is to observe the lowest possible variance (i.e., a narrow interquartile range) and a progressive increase in the median as a function of the actual rating given by the user.
We can observe that these sets of boxplots vary in appearance and seem to be more or less correlated between the different profiles *(see Figures 2 and 3)*. We therefore need a metric, a test, to determine if there is actually a correlation between the actual score and the score provided by the algorithm.

c. Quantitative Evaluation Using Spearman’s Rank Correlation

To determine whether there is a statistically significant positive link between the actual rating and the recommendation score, we use Spearman's rank correlation coefficient, which measures the strength and direction of a monotonic relationship between two sets of values. Unlike a strict linear correlation, Spearman's rank correlation coefficient depends only on the relative order of the values: it quantifies

whether movies rated higher by the user generally appear higher in the ranking proposed by the algorithm, regardless of the exact difference between the scores. This test is particularly relevant for recommendation systems, as the primary objective is to sort movies in a way that is consistent with the user's preferences, rather than to predict each rating precisely. We can therefore extract two metrics from this test: the Spearman's rank correlation coefficient, which falls within the range [-1, 1], and the p-value of the test, which falls within the range [0, 1]. The goal is to have a Spearman's rank correlation coefficient as close as possible to 1, which would demonstrate a positive correlation between the actual rating and the predicted score. Furthermore, to study the test statistic, we would need a p-value as close as possible to 0; generally, the most common convention is to accept the test if the p-value is less than or equal to 0.05.

Different Spearman's rank correlation coefficients can be observed depending on the profile studied, as well as different p-values. Spearman's rank correlation coefficients are never very high but are nonetheless positive. Boxplots that visually appeared to show a correlation between the score and the actual rating do indeed have a higher Spearman's rank correlation coefficient than other boxplots where no correlation seemed to exist. Regarding the p-values of the tests, for tests where the boxplots appeared to be positively correlated, very satisfactory p-values close to 0 are observed. Conversely, for tests where the boxplots did not appear to follow any particular distribution, rather high p-values are observed, far from the limit set at 0.05, and sometimes even close to 1, which indicates a clear lack of correlation between the actual rating given to the user and the score predicted by the algorithm. *(See Figures 4 and 5)*

d. Impact of the Number of movies Watched on Model Performance

When the p-value is this high, it means the model poorly predicts which movies the user might like, and therefore, the recommendation algorithm doesn't work for that profile. This result bothered us a lot, but we couldn't ignore it and only consider the cases where the algorithm worked well. So, we sought to understand the reasons for these poor predictions for some profiles when the algorithm seemed to work well for others. By creating several visualizations and running several tests for different profiles, we deduced that the number of movies watched by the user must have an impact on the accuracy of our model's predictions. Indeed, we realized that overall, the model predicted movies the user might like quite poorly when the number of movies watched by the user was relatively low. This seemed quite logical, and we wanted to be absolutely sure, which is why we conducted another series of tests to validate our hypothesis. We decided to scrape a list of users from the "Popular Members" tab provided by the Letterboxd website. We therefore chose to take 30 random users on this page and perform the exact same test as before, storing their number of movies watched, as well as the Spearman rank correlation coefficient and p-value obtained from the recommendation test. We then sought to answer the

following question: "Is there a correlation between the number of movies watched by the user and the result of the Spearman rank correlation test?"

To answer this question, we produced two scatter plots and fitted two linear regression models. The first scatter plot shows the number of movies viewed versus the p-value of the corresponding Spearman's rank correlation coefficient, while the second shows the number of movies viewed versus the corresponding Spearman's rank correlation coefficient. *(See Figures 6 and 7)*

Similarly, two linear regression analyses were conducted to study the relationship between the number of movies watched by a user and the performance of the recommendation system, evaluated through Spearman's rank correlation test. The first regression examined the relationship between the number of movies watched and the p-value associated with the Spearman correlation, while the second focused on the relationship between the number of movies watched and the Spearman correlation coefficient itself. *(See Figures 8 and 9)*

The first regression revealed a statistically significant negative relationship (p-value of the regression = 0.00641) between the number of movies watched and the p-value of the Spearman test. As the size of the user's viewing history increases, the p-value tends to decrease, indicating that the correlation between the user's actual ratings and the recommendation scores becomes statistically more reliable. This result suggests that the recommendation system requires a sufficient amount of user data to produce stable and statistically meaningful rankings.

In contrast, the second regression showed no significant linear relationship (p-value of the regression = 0.425) between the number of movies watched and the value of the Spearman correlation coefficient. The coefficient associated with the number of movies watched was close to zero and not statistically significant, and the model explained only a negligible portion of the variance. This indicates that increasing the size of a user's history does not systematically lead to a stronger monotonic relationship between predicted scores and actual ratings.

Taken together, these results highlight an important distinction between statistical reliability and effect magnitude. While larger user histories improve the robustness and significance of the evaluation, they do not necessarily guarantee a higher correlation coefficient. In practice, this suggests that additional user data reduces uncertainty in the evaluation of recommendation quality, but that the intrinsic alignment between the model's ranking and a user's preferences also depends on other factors, such as the consistency of user ratings or the diversity of the movies watched.

Overall, this analysis supports the conclusion that the number of movies watched by a user primarily affects the confidence in the evaluation of the recommendation system, rather than directly driving improvements in the measured performance itself.

6. Discussion and conclusion

One of the project's main strengths was the construction of a large-scale semantic embedding space for movies based on rich and extensive data. By using Word2Vec on a corpus of approximately 100,000 movies with around 400,000 unique attributes, the model successfully captured meaningful semantic relationships between movies, genres, people, and production contexts.
Another strong point of our project was the way in which we managed to vectorize a user. The representation of a user as a weighted combination of the movies they have watched, taking into account both positive and negative preferences, enables the model to encode not only what a user likes, but also what they explicitly dislike. The introduction of non-linear and signed weighting improves the expressiveness of the user vector and helps mitigate the dominance of frequently consumed but weakly appreciated content.
We believe that the model evaluation strategy was also a strength of the project because we separated a user into a test and validation set, created explicit visualizations, and conducted tests to validate or refute our expectations. We were able to question our assumptions and statistically observe that the model was far from perfect, and we succeeded in describing one of the model's main biases: the number of movies watched by a user.
This is therefore the biggest flaw that we can find in our model. The analysis clearly shows that the algorithm struggles to produce reliable recommendations for users with a small number of watched movies. In such cases, the user vector is poorly estimated, leading to unstable rankings and non-significant evaluation results.
Other, smaller flaws or arbitrary choices we made can be discussed. Notably, the decision to center the movie ratings and process them exponentially. This choice was made after numerous optimization tests, and we found it yielded the best results for our model.
Finally, the evaluation focuses exclusively on offline statistical metrics. While Spearman’s rank correlation provides valuable insight into ranking consistency, it does not directly measure user satisfaction or recommendation novelty, which are critical aspects of real-world recommendation systems.

In conclusion, this project demonstrates that a content-based recommendation system leveraging a large-scale semantic embedding space can provide meaningful personalized movie recommendations. By representing both movies and users in the same vector space and incorporating weighted preferences, the model captures nuanced user tastes and generates recommendations aligned with individual preferences. However, the analysis also highlights inherent limitations, particularly in cold-start scenarios where users have few watched movies, and the reliance on offline statistical metrics rather than direct user feedback.

Overall, this project lays a strong foundation for improving movie recommendations. Future work could explore including more user behavior data and evaluating the system directly with real users. Despite its limitations, the project shows that embedding-based representations can effectively capture user preferences and provides useful insights for building and testing recommendation algorithms.

7. Appendix

```
=== ACTION ===
hanna                       0.853069
knight-and-day              0.852449
mission-impossible-iii      0.847251
the-international           0.843614
volcano                     0.843256

=== ADVENTURE ===
hanna                              0.853069
mowgli-legend-of-the-jungle        0.848522
mission-impossible-iii             0.847251
ai-artificial-intelligence         0.844496
the-international                  0.843614

=== ANIMATION ===
beowulf-2007                       0.832132
cars-2                             0.807260
sgt-stubby-an-american-hero        0.807077
rango                              0.805154
battle-for-terra                   0.804645

=== COMEDY ===
knight-and-day                   0.852449
the-brothers-grimm               0.837889
the-instigators                  0.835225
men-in-black-international       0.835039
role-play-2023                   0.833200
```

Figure 1 : Example of recommendations provided by the algorithm for a user. Here, the recommendations for the first four genres—"action", "adventure", "animation", and "comedy" are visible

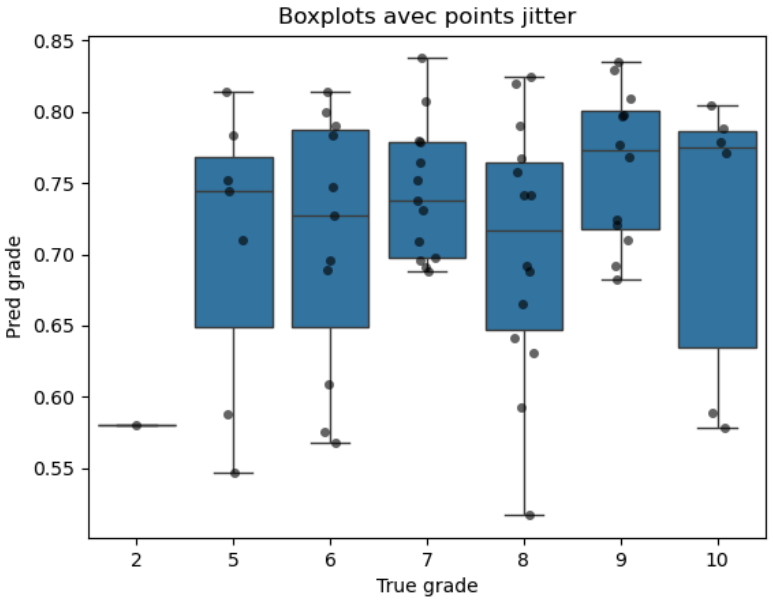

Figure 2 : An example of a boxplot that does not appear to show a correlation between predicted and actual grades

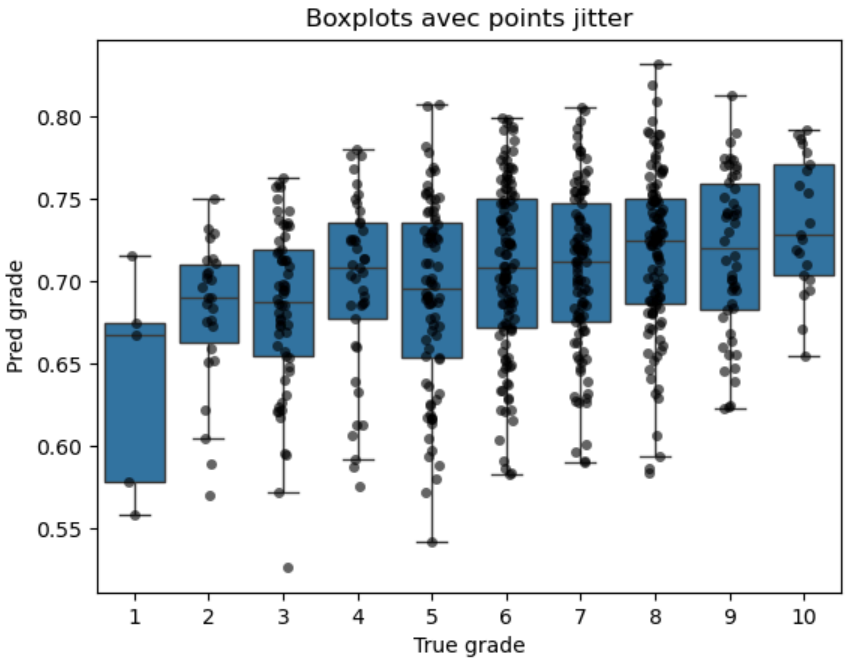

Figure 3 : An example of a boxplot that appears to show a positive correlation between predicted and actual grades

```
Spearman rho = 0.155, p-value = 0.2199
```

Figure 4 : Value of Spearman's correlation coefficient and p-value of the test associated with Figure 2

```
Spearman rho = 0.232, p-value = 0.0000
```

Figure 5 : Value of Spearman's correlation coefficient and p-value of the test associated with Figure 3

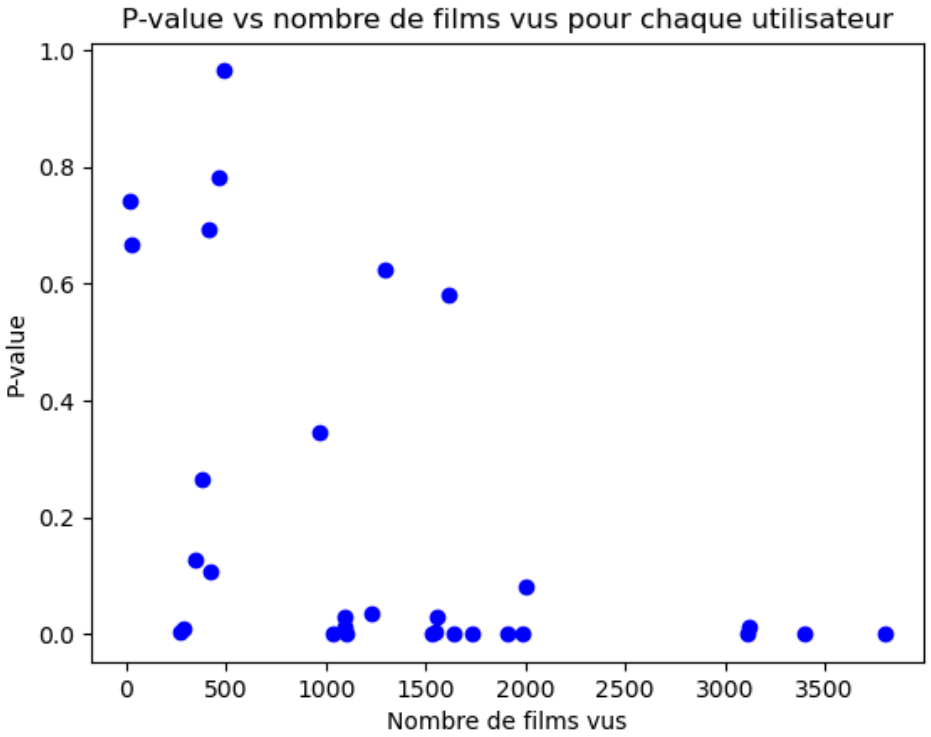

Figure 6 : Scatter plot of the p-value of the Spearman correlation test as a function of the number of movies viewed by the user

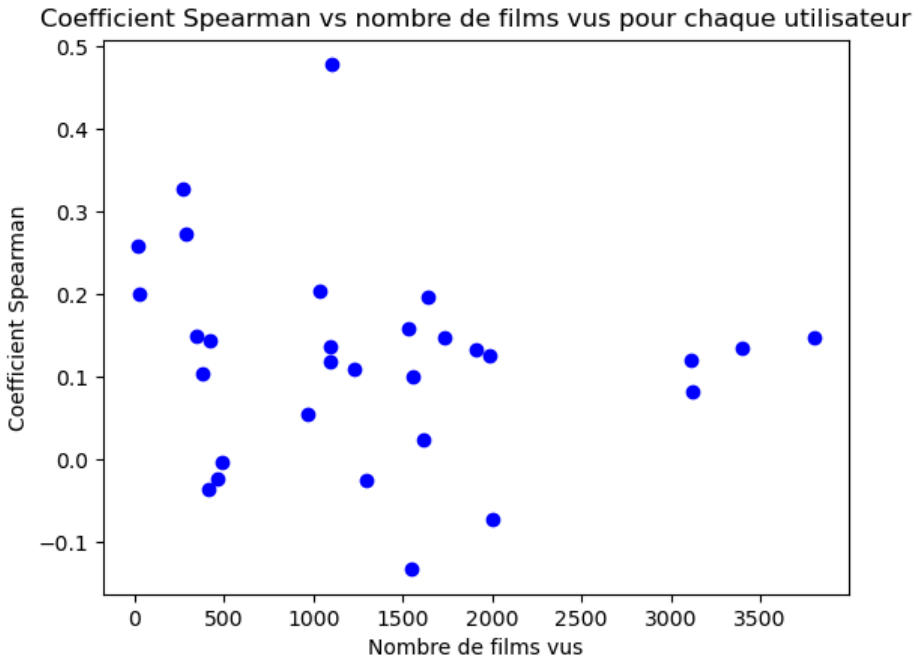

Figure 7 : Scatter plot of the Coefficient Spearman of the Spearman correlation test as a function of the number of movies viewed by the user

```
==============================================================================
Dep. Variable:                      y   R-squared:                       0.237
Model:                            OLS   Adj. R-squared:                  0.209
Method:                 Least Squares   F-statistic:                     8.684
Date:                Fri, 09 Jan 2026   Prob (F-statistic):            0.00641
Time:                        18:43:04   Log-Likelihood:                -2.5415
No. Observations:                  30   AIC:                             9.083
Df Residuals:                      28   BIC:                             11.89
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.4007      0.083      4.816      0.000       0.230       0.571
x1            -0.0001   5.01e-05     -2.947      0.006      -0.000    -4.5e-05
==============================================================================
Omnibus:                        3.617   Durbin-Watson:                   1.365
Prob(Omnibus):                  0.164   Jarque-Bera (JB):                3.255
Skew:                           0.760   Prob(JB):                        0.196
Kurtosis:                       2.457   Cond. No.                     2.77e+03
==============================================================================
```

Figure 8 : Results of the linear regression associated with Figure 6

```
==============================================================================
Dep. Variable:                      y   R-squared:                       0.023
Model:                            OLS   Adj. R-squared:                 -0.012
Method:                 Least Squares   F-statistic:                    0.6554
Date:                Fri, 09 Jan 2026   Prob (F-statistic):              0.425
Time:                        18:43:04   Log-Likelihood:                 21.253
No. Observations:                  30   AIC:                            -38.51
Df Residuals:                      28   BIC:                            -35.70
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1461      0.038      3.882      0.001       0.069       0.223
x1         -1.836e-05   2.27e-05     -0.810      0.425   -6.48e-05    2.81e-05
==============================================================================
Omnibus:                        3.464   Durbin-Watson:                   1.713
Prob(Omnibus):                  0.177   Jarque-Bera (JB):                2.101
Skew:                           0.348   Prob(JB):                        0.350
Kurtosis:                       4.094   Cond. No.                     2.77e+03
==============================================================================
```

Figure 9 : Results of the linear regression associated with Figure 7

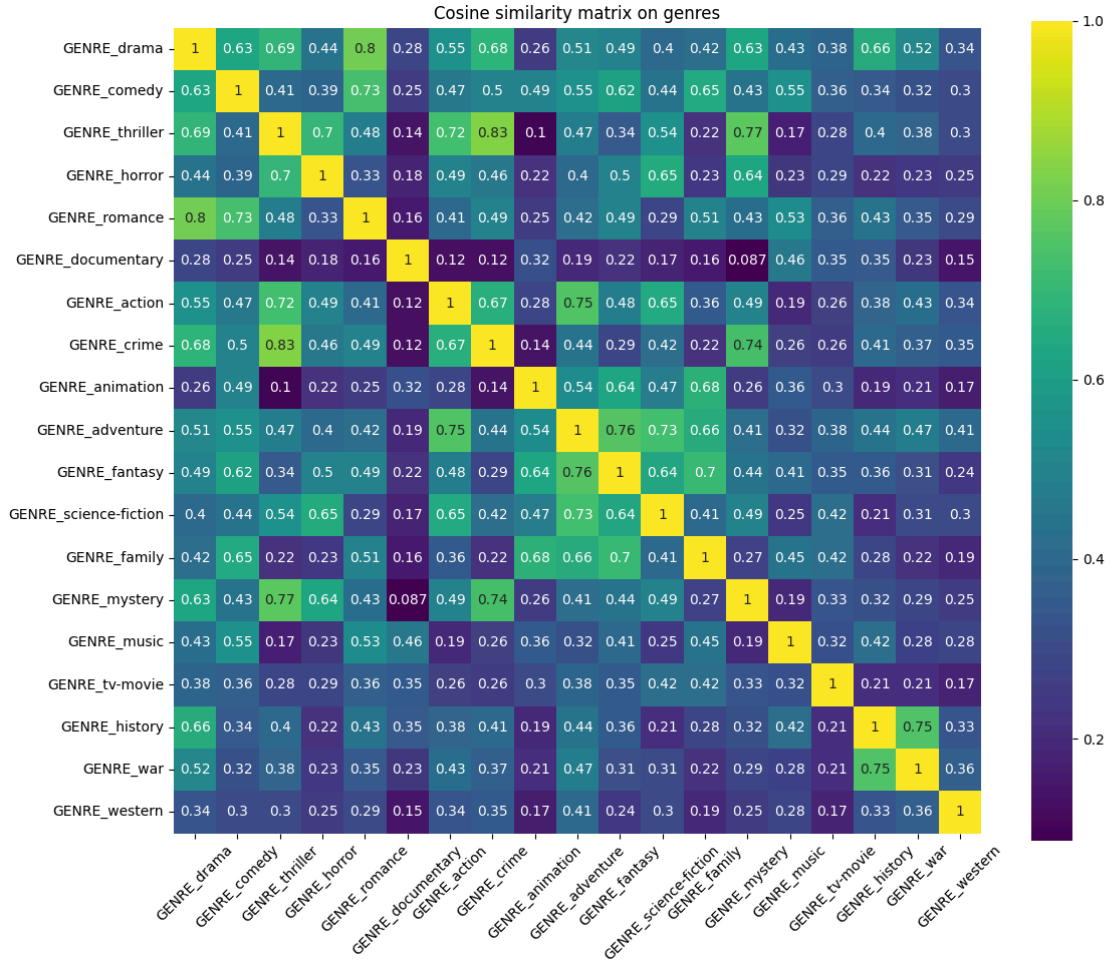

Figure 10: cosine similarity matrix between genres

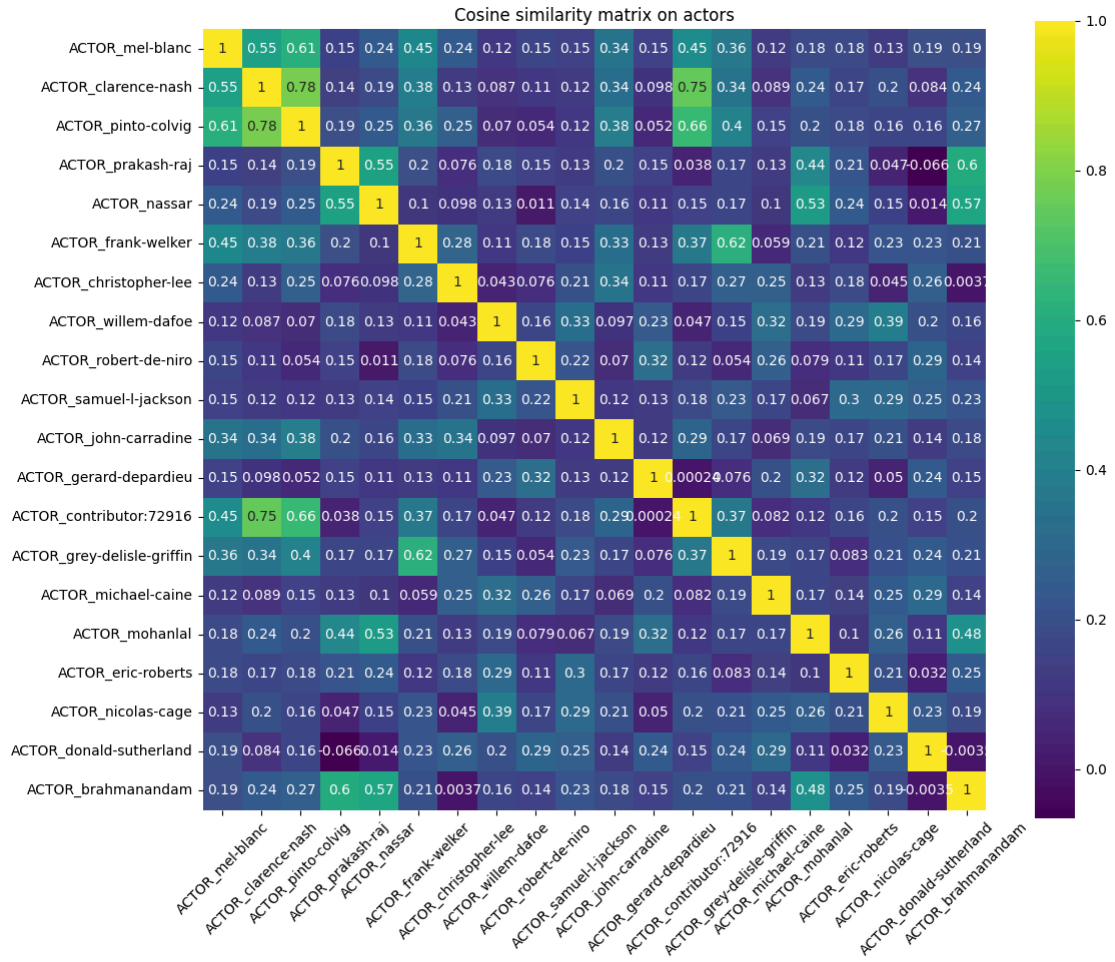

Figure 11: cosine similarity matrix between actors

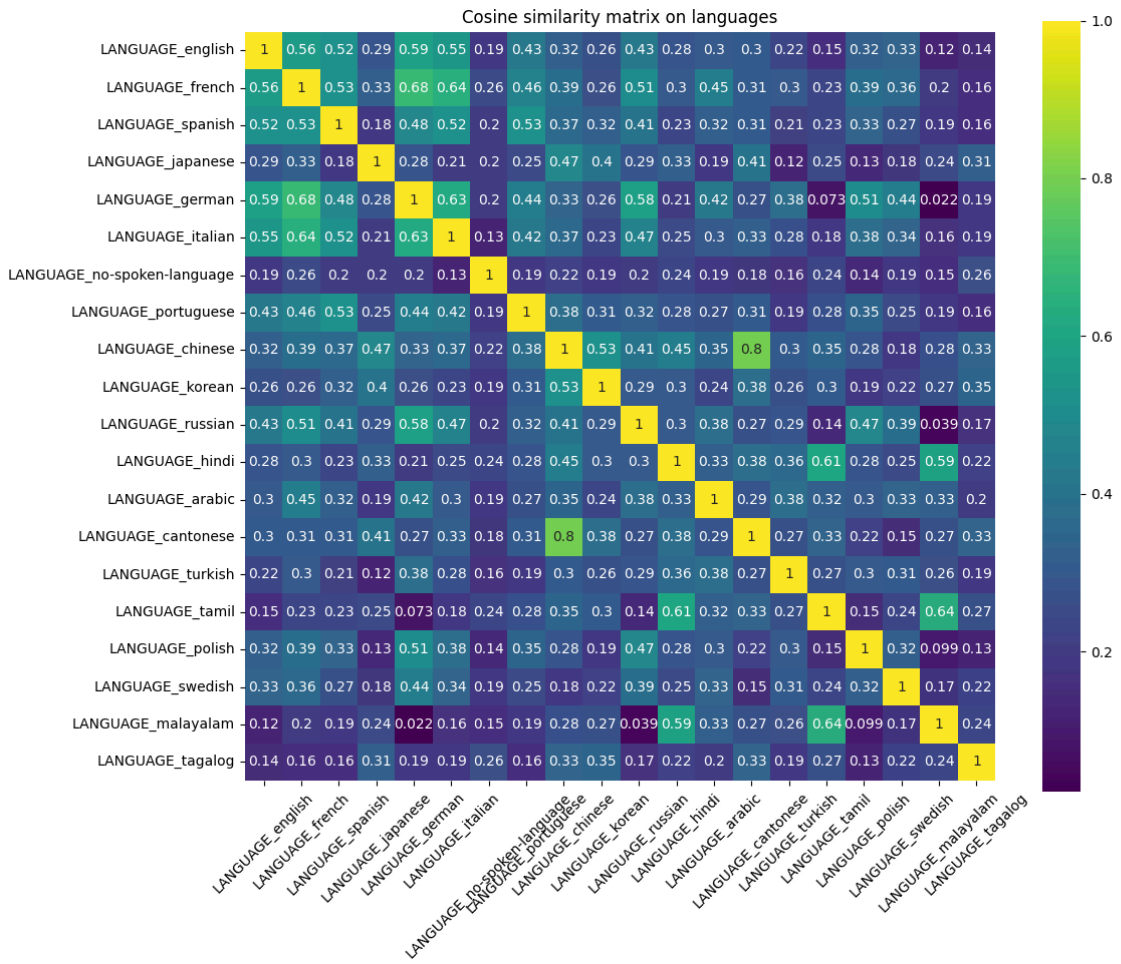

Figure 12: cosine similarity matrix between languages

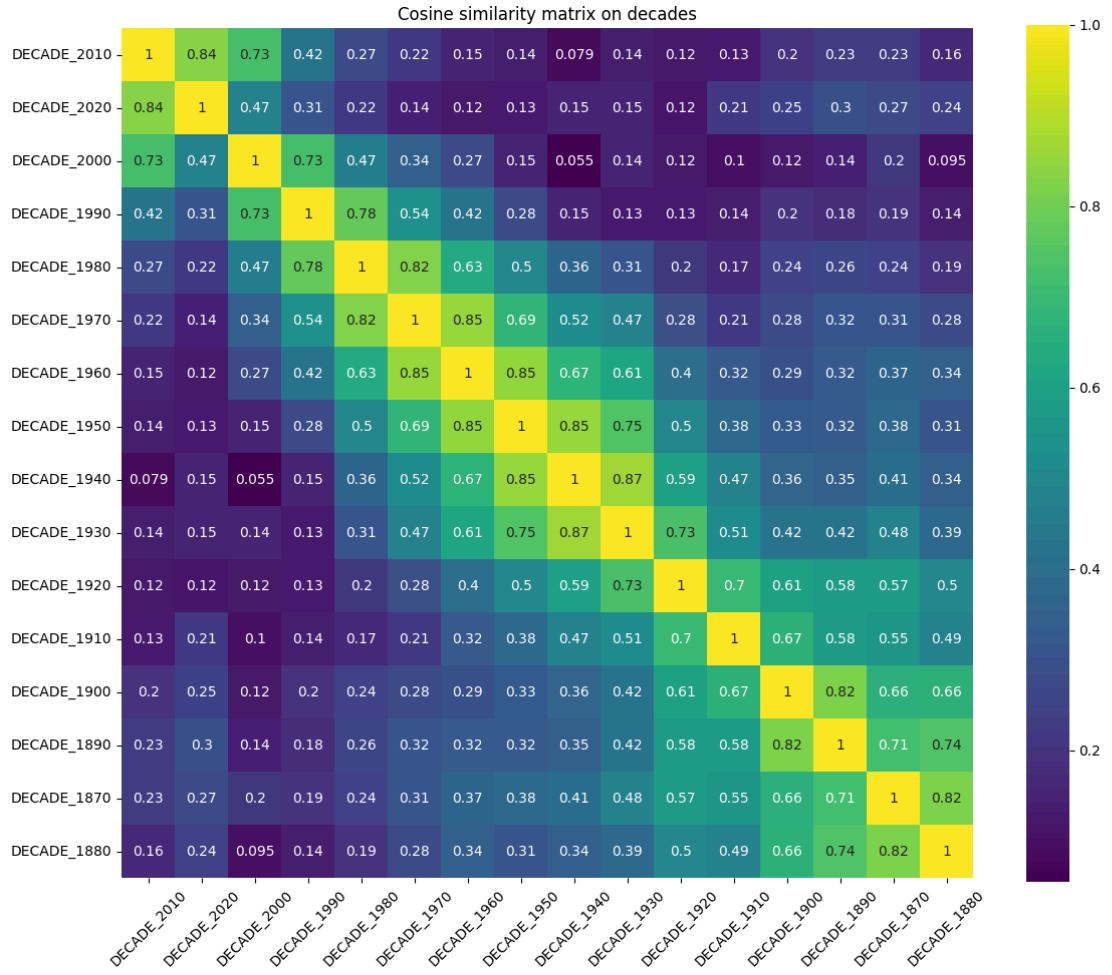

Figure 13: cosine similarity matrix between decades

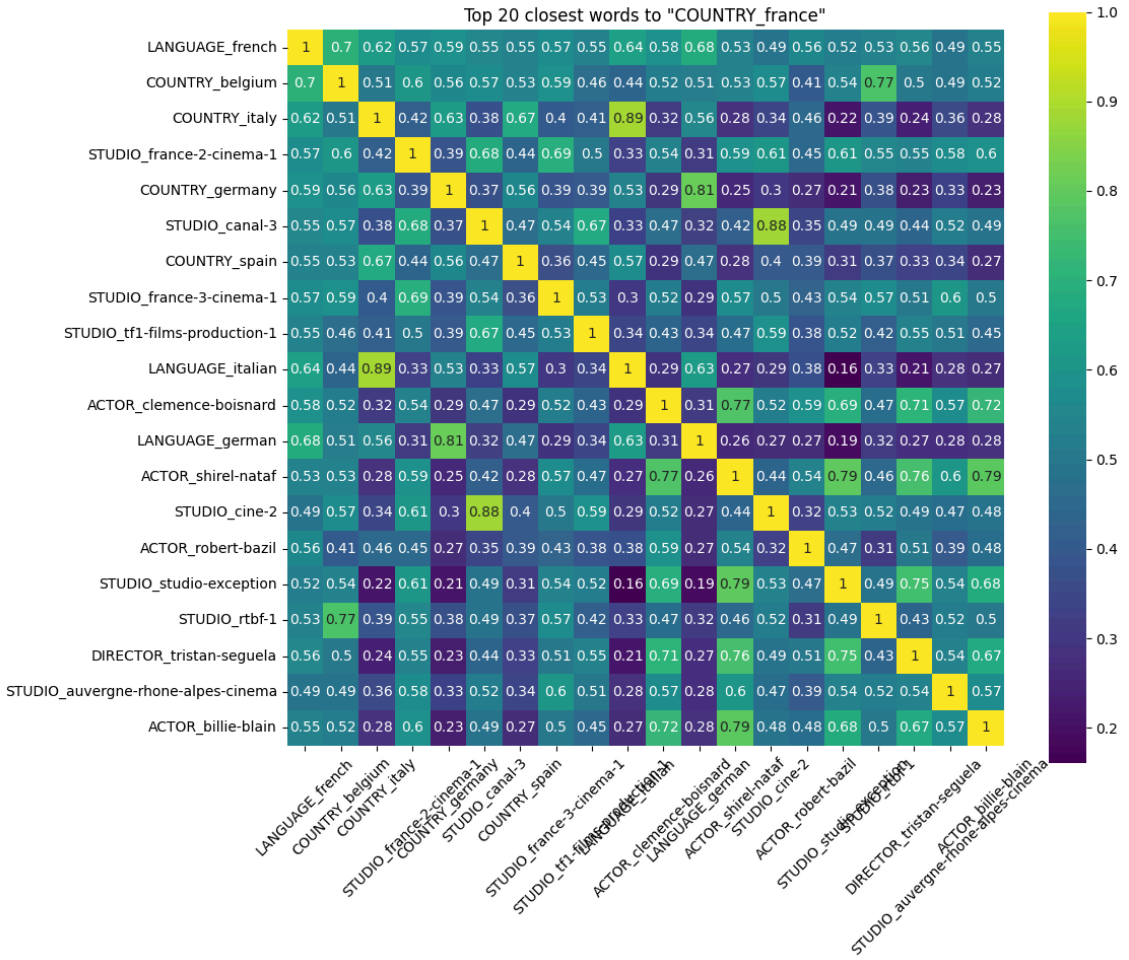

Figure 14: Top 20 closest words to the attribute “COUNTRY_france”

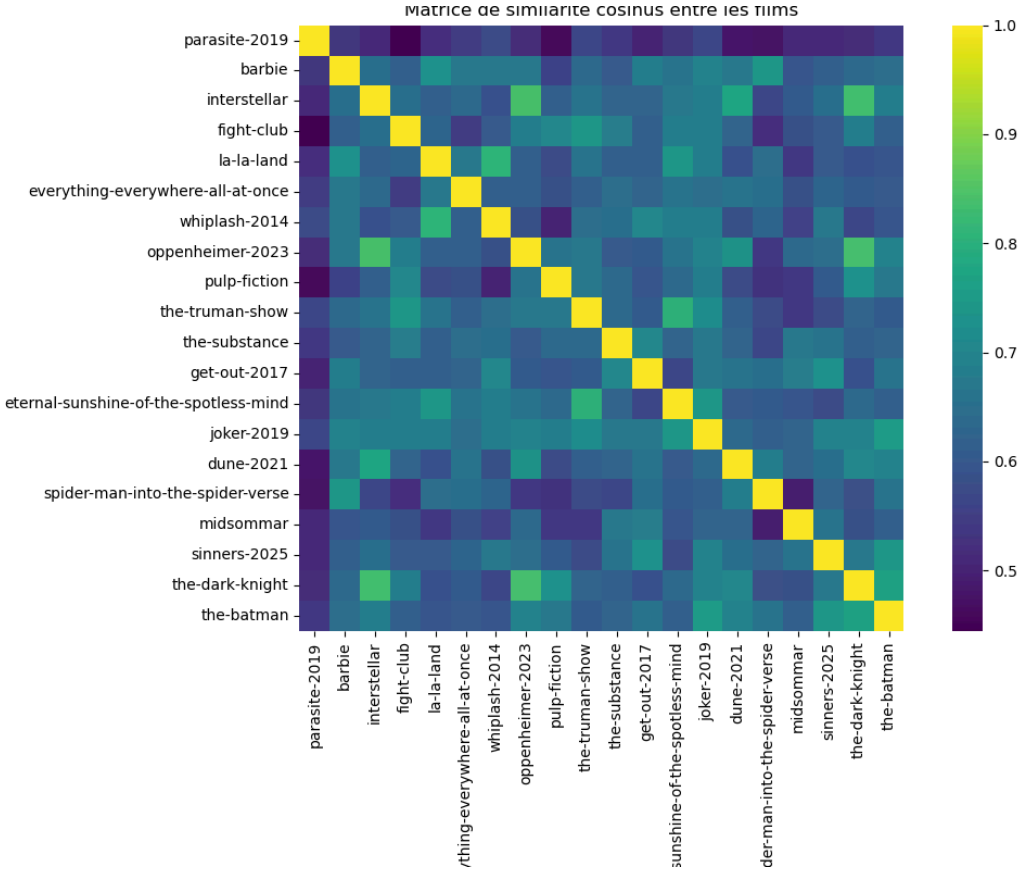

Figure 15: cosine similarity matrix with the first 20 most popular movies

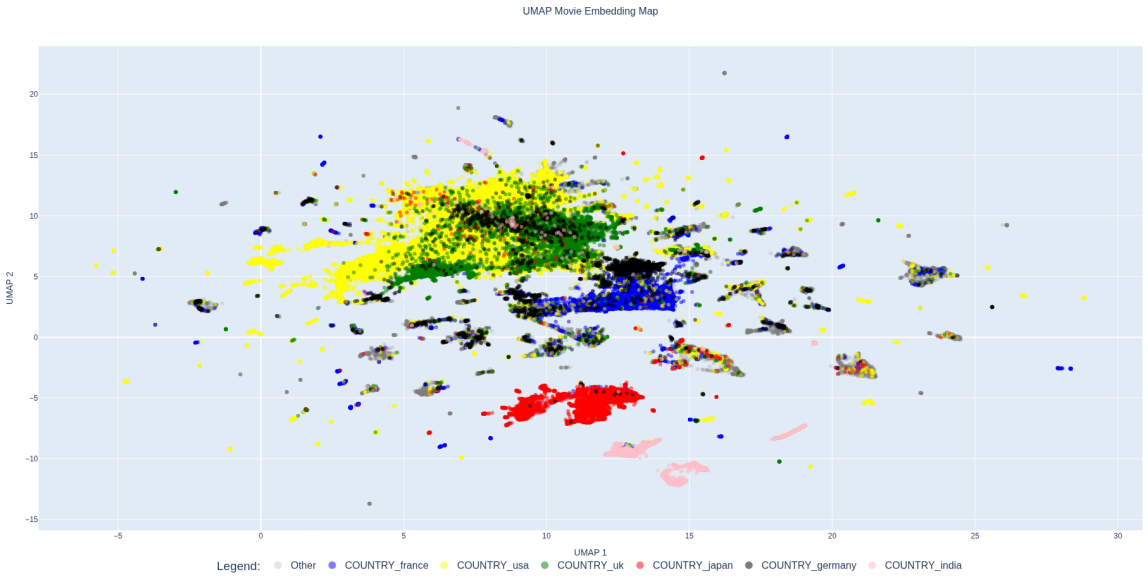

Figure 16: Umap representation of movies by country

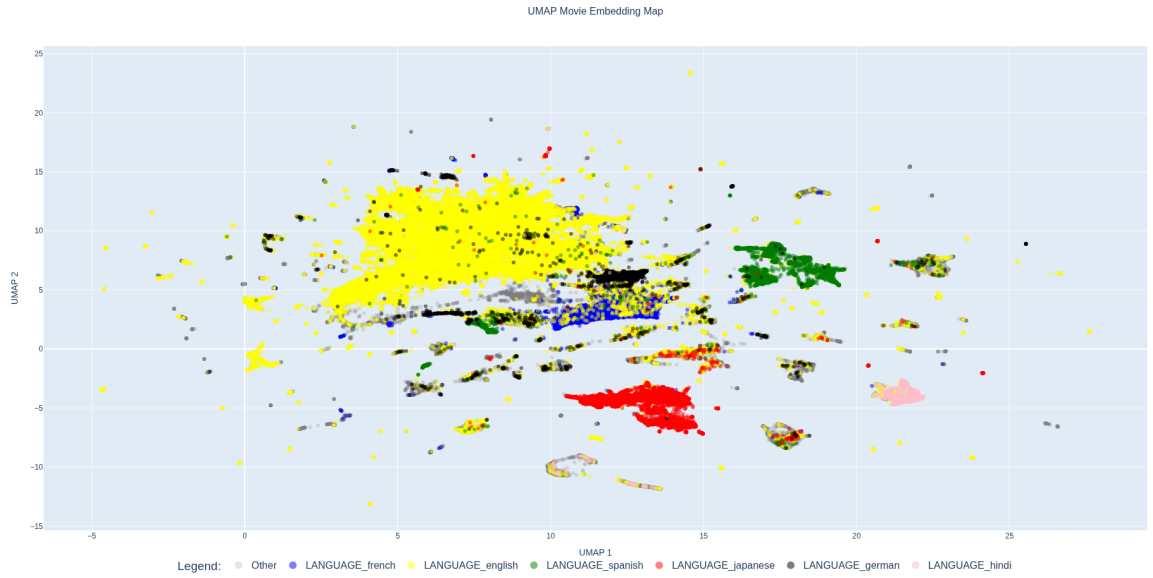

Figure 17: Umap representation of movies by language