Report ML/Data Science Project

Emile Descroix

Spring 2025

Contents

1 Introduction and Motivation

2 The Dataset and its Construction

The dataset used during this project contains various information on all movies nominated for the Best Picture Academy Award from 1927 to 2024. We will see what features had been chosen and what they represent, then how the dataset was built and finally a few particularities of it.

2.1 Explanation on the features

For each movie we have a total of six features giving important information which could help us predicting the winning films. We will list those features above and give a short explanation on what they illustrate.

- The first feature is the average rating, out of 5, of the movie on the iMDb website from more than 250 million monthly users. It gives us an idea of the global audience perception of the movie, and we expect higher-rated movies to perform better at winning awards.
- Then the data set contains the runtime in minutes for each movie.
- Another feature aims at quantifying how "Oscar-friendly" the movie cast is. It counts the number of actors in the cast who already won an Oscar for previous work. The awards counted are *Actor in a leading role*, *Actress in a leading role*, *Actor in a supporting role* and *Actress in a supporting role*. This feature could show whether having an already recognized cast helps in winning the Oscar.
- A similar variable aims to do the same thing for directors by counting how many *Best Director* Oscars the director won before.
- The data set also contains the production budget in dollars.
- We also have two variables stating wether or not the movie won the BAFTA or the Golden Globes best picture awards. Those two prizes are the most prestigious among the English-speaking world and can often give a trend for the rest of the award season and the Oscars ceremony.
- Finally, for each movie we have its genres from 1 up to 3 among classic cinema genres such as drama, adventure, history... This is the only categorical variable of the dataset and it needs to be preprocessed accordingly.

2.2 Construction of the dataset

The dataset is mainly based on non-commercial iMDb data sets that can be freely downloaded at this address `https://datasets.imdbws.com`. Those contains basic data on every movie register in the iMDb database such as title, runtime, genres, release date... They can be easily handled, thanks to a

unique identifier per movie shared by all datasets. To select only nominated movies they were merged to the following dataset `https://www.kaggle.com/datasets/unanimad/the-oscar-award/data`, containing the list of all nominated movies at the Academy Awards for all awards from 1927.
This merging was the first challenge faced while building the data set.. Indeed, the Kaggle dataset does not feature any identifier for the movies, so the merging was made based on the titles, converted to a "clean" version without punctuation, spaces, or capital letters, which required attention to a few details:

- The first inconvenience was movies sharing the exact same title to solve this issue the merging was made on both title and release year to avoid mixing information about homonym movies. For example, a *West Side Story* movie was nominated both in 1961 and in 2021.

- The second problem faced while merging was the difference of years between the two dataset, some movies were dated considering the general release other and their first screening in festivals for example. Allowing a 2 years interval for movies with same title solved most of the problematic cases. However, this allowed several matches for some of them and the closest one was kept when it happened.

- Moreover, for the same movie a discrepancy in title appeared for a few entries (e.g. *Star Wars* in the Kaggle dataset was listed in the iMDb database as *Star Wars : Episode IV - A New Hope*). A few solutions were tried to solve this issue such as using inclusion instead of equality or allowing a fixed number of differences for the title but none of them was efficiently covering all the different cases. The 10 titles which could not be merged using the adjustments listed above were manually found in the iMDb database and added to the dataset.

Once all datasets are merged one additional information - budget - as it is not contained in the iMDb files is added by querying wikidata. Wikidata is a collaborative database storing really various information and allowing Wikipedia information to be exploited by users. In order to query it one can use the SPARQL API allowing to retrieve the data in structured and machine-readable format. The budget is still missing for a considerable amount of entries, it is kept since it could be an important parameters to determine if the movie will win the Oscar. However, this lack of information should be handled with care and the sections to come will describe the various methods used.

2.3 Dataset particularities

This dataset has a few particularities due to either the nature of the task, the way the Oscars work or how the dataset was built. It requires attention at each step of the project, sampling the data, splitting the set, training the models or even when evaluating them. Among these aspects we have :

- The two classes are really unbalanced, within the 611 movies of the data set only 97, roughly 16%, of them won the Oscar. The rest of the movie form the other class.
- For each year we need to predict exactly one winner among the 5 or 10 nominees.
- The data set covers 97 years and criteria for Oscar winners have probably evolved through the years. It could make finding trends and making predictions harder.

2.4 Splitting the Dataset

Once the dataset is built we can split in three different ones, training, validation and testing. The small adjustment we need to do is to keep movies from the same year in the same dataset. To reproduce a real life situation, the split is not random and we use the data from the last 15 years as a test set. This way, we are trying to predict results only with older data.

3 Exploratory Data Analysis

Before diving into model training and selection, we need to explore the data set and get to know it. In the following sections, we will do so through plots and metrics to gain preliminary information on the variables importance and first trend in the data.

3.1 Distributions and Metrics

- The first variable that we are interested in is the average rating. The plot below illustrates the difference in rating distribution between winners and nominees. While the spread of the ratings is similar, the winners distribution is slightly more to the right, indicating that winners tend to have a higher average rating. This could indicate that rating is, as we might expect, an important information for our models. Table 1, summarizing the statistic values for the ratings confirms that ratings tend to be lightly higher for winners.
- We now do the same thing for the budget. From the figure below (see Fig.2), we get two key information, first, the budget distribution is right-skewed meaning more films have a lower budget. We can spot some outliers on the left of the curve implying we might need to log-transform the data if we use it in our models. Second, while it might be counterintuitive, winners are more concentrated around lower budgets. However, the difference in distribution between winners and nominees is not this significant., budget is probably not the strongest indicators. The statistics metrics (see Table 2) confirm this trend.

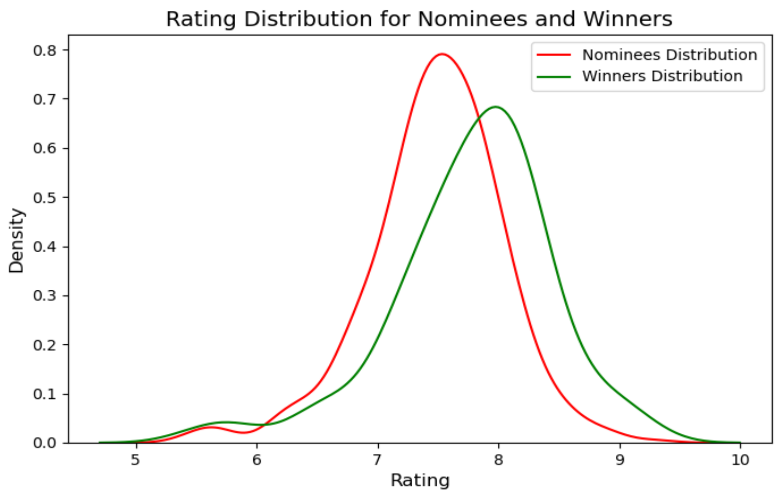


Figure 1: Distribution of the rating variable

- Finally, by doing the same kind of study on the runtime we also notice a small difference between winners and nominees. While both distributions are similar (see Fig. 3), winners tend to be slightly longer than the rest of the data set. This is once again confirmed by the metrics in the table below.

Variable	Statistic	Winners	Non-Winners	All
Rating	Mean	7.77	7.48	7.52
	Median	7.9	7.5	7.6
Runtime (min)	Mean	138.01	124.19	128.36
	Median	130	120	122
Budget (M $)	Mean	21.42	31.01	28.86
	Median	13.2	14	14

Table 1: Summary statistics (mean and median) for rating, runtime (in minutes), and budget (in million dollars) by group

- We now inspect the relation of other awards wins, BAFTA and Golden Globes, and the Oscar win. On the heatmap below we can see, that winning awards have indeed great influence on the Oscar results. As an example, 63% of movies winning both the BAFTA and Golden Globes for best drama won the Oscar just after. On the other hand, the rate of Oscar winners among movies missing on the three other awards drops to 8.2%. The bottom line is 0 simply because it is impossible to win both Golden Globes prizes.

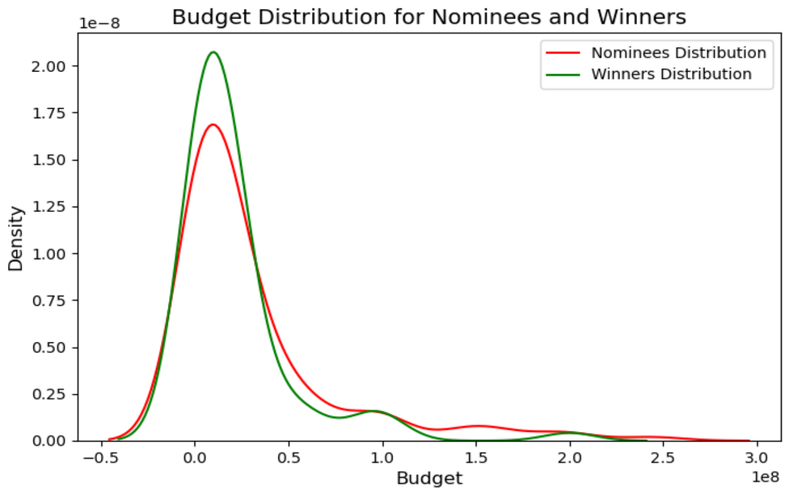


Figure 2: Distribution of the budget variable

- Finally, we want to study the repartition of genres. As a reminder, each movie has a list from 1 up to 3 genres. The following bar plot displays the repartition of genres among winners comparing it to the nominees one. We can see, (cf. Appendix) that to win the Oscar or even to be nominated a film almost has to be a drama. The top genres are the same among winners and nominees and with similar proportions, 85-90% for dramas, 30% for romance and 20% for comedy. A more interesting aspect is over and under representation among winners which could give us an hint on genres being favored or not. Two noticeable examples of over representation ar war and sports movies respectively jumping from 7 to 17% and from 1 to 4%. On the other hand genres such as action, mystery or fantasy have an apparent drop in winners representation. Sci-Fi and animation movies even disappear from the winners list.

3.2 Preliminary clustering and PCA visulisation

3.2.1 Clustering

After using the pre-processing methods described in the next section, we can do unsupervised clustering to detect groups in our data with hope that they gave us insight about win prediction. To do so, we used the k-means algorithm in our dataset and chose the most appropriate k-value using silhouette score. For each point, the score is based on the average distance with the rest of its cluster,

$$a(i) = \frac{1}{|I_k|-1} \sum_{j \in I_k,\, j \neq i} d(x^i, x^j)$$

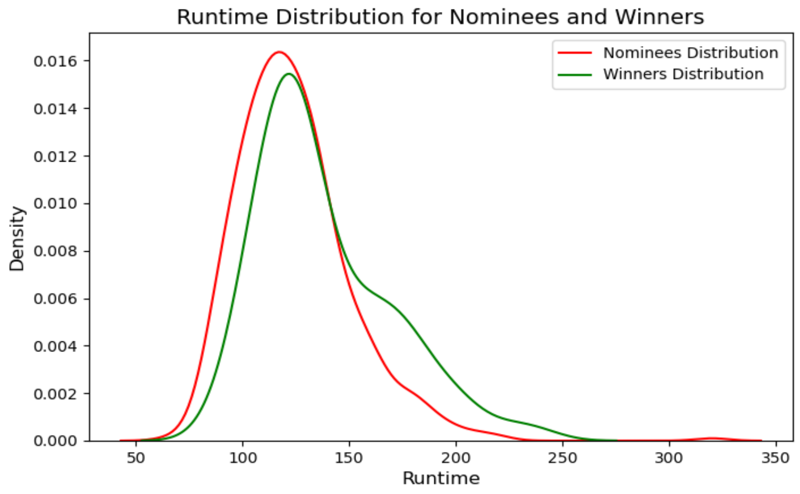


Figure 3: Distribution of the runtime variable

and the average distance with the closest different cluster,

$$b(i) = \min_{k' \neq k} \frac{1}{|I_{k'}|} \sum_{i' \in I_{k'}} d(x^i, x^{i'})$$

This gives us the silhouette score for point i,

$$s_{sil}(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

The silhouette score for the full data set explain how well the data have been classified and range from -1, worst classification to 1, best classification.

$$S_{sil} = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|I_k|} \sum_{i \in I_k} s_{sil}(i)$$

For our dataset, we have the following results, (see Fig.5), suggesting we should use $k = 2$.

3.2.2 PCA Visualization

To easily visualize our clusters and compare them to the winners repartition we use a classic dimension reduction PCA with 2 components. This yields the following plots, While the clustering algorithm seems to find a structure in the data it does not correspond to the outcome we want to predict. This could be explained by several reasons:

- The transformation through PCA may not capture some non-linear relationships.

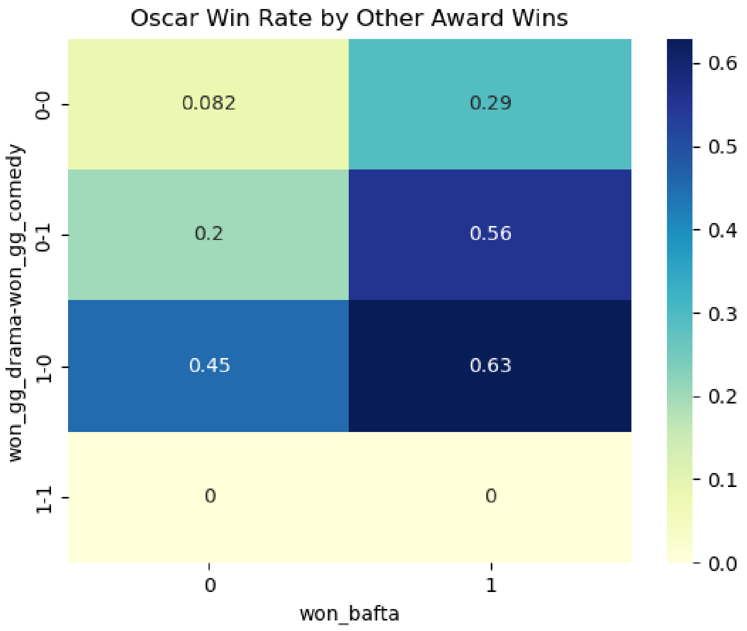


Figure 4: Repartition of genres among winners vs nominees

- K-means might not be the most suitable algorithm for this precise task
- The features used may no be sufficient to discriminate the winners from the nominees.

4 Pre-Processing

Considering the nature of the features and the particularities listed previously, we need to apply some usual pre-processing methods to our data before trying to use clustering and training models.

- To encode genres, we use a classic one hot encoding preserving the full information using `scikit-learn` `MultiLabelBinarizer`.
- For the budget, we have several options. Since the data is missing from a lot of entries, we could delete those to train the models on a complete dataset. However, this would considerably reduce the size of our training set and could cause a drop in performance. In order to try to keep the larger training set possible, we can replace the missing budgets by reasonable values as the mean or the median of movies released ± 1 year window. We will try those different solutions and compare results on the testing

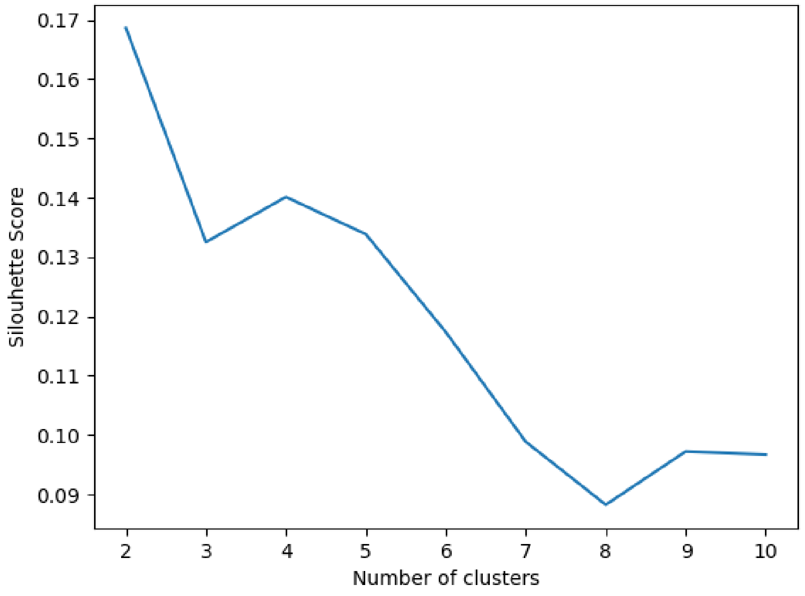


Figure 5: Silhouette scores for different values of k

set. Accordingly to what we observed in the previous section, we will use a logarithmic transformation of the budget to handle outliers.

- Finally, we will normalize the data using `scikit-learn` `StandardScaler`

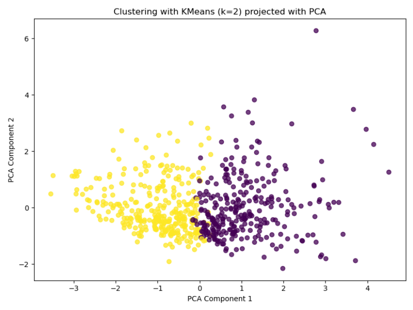
5 Model Training and Selection

The data set is now complete and well preprocessed, we have a few indication on how the data behave and which variables could be useful for our predictions. Hence we can start our supervised learning. The following sections will discuss the different evaluations strategies we can adopt and the models we used and how we built them. For more advanced techniques, we will explain how they work and why they are suitable for our task.

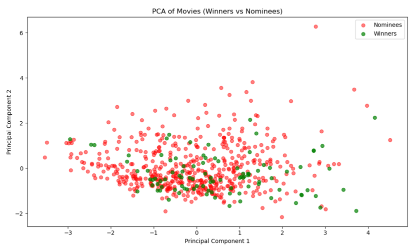
5.1 Evaluation Strategy

5.1.1 Evaluating the models

Considering our task and dataset we have a somewhat different case of classification. Indeed, we are restricted to exactly one winner per year and wish to



(a) PCA projection of clustering using K-Means with k=2



(b) PCA projection of the data set with winners and nominees

Figure 6: Comparison of clusters and winners/nominees repartition

rank the nominees from the same year. Hence, we need to establish what makes a model effective. We can check the following accuracies,

- The most natural way is "Top-1 accuracy", for each year our predicted winner is the nominee with the highest probability to win,, and then we just compute the proportion of well-predicted winners.
- However, we can also compute "Top-k accuracy", typically with $k = 3$, to get the proportion of years for which the actual winner was in our k-first predictions. This allows for near-misses and makes the evaluation more flexible.
- We can also use the classic scores provided by the `scikit learn` functions, but those scores might be misleading since the models could classify more or less winners than what is allowed (e.g., multiple winners for the same year).

5.1.2 Cross Validation

If we want to perform cross validation we encounter two issues with our data set, the classes are imbalanced and we don't want movies from the same year to be separated, they need to be all either in the training or the testing set. To solve the imbalance we could use a stratified k-fold preserving the same proportion in both training and testing sets. However, we can avoid using it by splitting our data with a "Group K-Fold". Group K-Fold ensures that entries belonging to the same group, here the same year, stay in the same fold. It solves both of our problems since it ensures proportion indirectly.

5.2 Logistic Regression

5.2.1 Baseline Model

The first model we try is a logistic regression. We adjust the predictions to fit our task, instead of taking the model raw prediction, for each year we select the entry with the highest probability to be the winner all other are predicted non-winner. We run a first model with all the features and without any regularization to have a results baseline. For a 1000 runs of this model on random train-validation splits we get an average top-3 score accuracy of 81% and an average top-1 score of 46%. These results are quite encouraging but without any regularization the model might overfit the training set data. In the next sections we will try to upgrade the model and to make it more robust through feature selection, regularization and hyper parameters tuning.

5.2.2 Hyper parameters Tuning

In logistic regression we can choose two hyper parameters, the type of penalty L1 (Lasso) or L2 (Ridge) we can also combine both with the Elastic Net regularization. In our case L1 showed poor results compared to L2 and caused the model to not converge on some data splits. So we can stick to L2 penalty and we have to tune the regularization strength to find an optimal value. On the plot below, see Fig. 7, we can see both top-1 and top-3 accuracies for different value of C, the inverse of the regularization strength. The error bars also display the standard deviation, giving us an idea of the variance for each model. To compute the accuracies we cannot use the built in `GridSearch` function since it does not support our custom scorer. Instead for each value of c we fit the model on a 100 different data splits and we compute the average score on the validation set. As we can see, the best value for C is between 0.1 and 1 and it outperforms the model with a strong regularization, on the right of the plot, closer to a plain logistic regression. Additionally, this regularization could prevent overfitting which could happen with our baseline model and perform better on the testing set. With more precise tests, the optimal value for c seems to be $C = 0.2$, we can keep it and use it for our final logistic regression model.

5.3 Random Forest

5.3.1 The Method

Before training and tuning our random forest model we do a small recall on what is a random forest and why it is appropriate for our task and dataset. Random Forest is an ensemble method based on decision trees. Decision trees have a few drawbacks they are unstable because a small change in the data can result in a completely different tree and are prone to overfitting. To solve these problems random forest is based on an ensemble of trees. Each tree is built on a bootstrap sample, sample with replacement, and we take one vote per tree for the final classification. In our case since we are not directly interested in

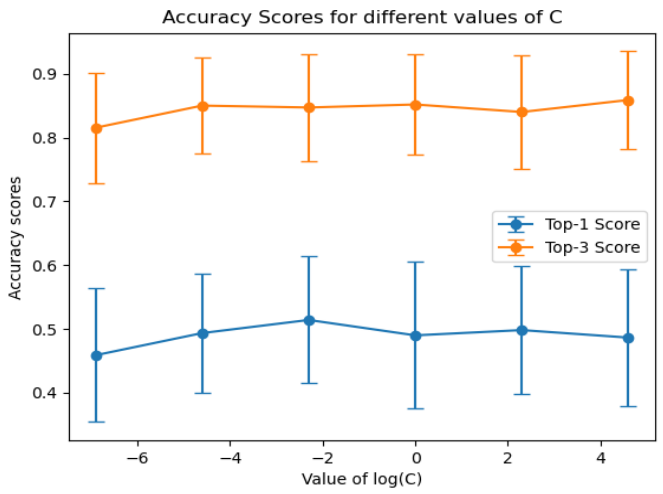


Figure 7: Accuracies for different values of regularization strength

the class but the probability since we want to take the movie with the highest probability to be the winner for each year. The probabilities outputted by a random forest are the mean of the trees probabilities. Random Forest is suitable for our dataset and task for several reasons,

- We can use the outputted probabilities for our ranking model and allow us to use our top-1 and top-3 accuracies described above.
- It is robust to overfitting thanks to the tuning of hyper parameters such as the number of trees or their depth. So we can used it on our relatively small dataset.
- It can capture non-linear relationship and interactions between features.

As for logistic regression we start with a baseline model with default parameters and we get the following results : 47% for the top-1 accuracy and 83% for top-3.

5.3.2 Hyper parameters Tuning

For Random Forest we can tune a certain number of hyper parameters we will focus on the following,

- The number of trees used in the forest, usually the more tree the better. However, while having more trees does not lead to overfitting it takes longer to train the model and we should choose a reasonable number giving satisfying results without taking to much time to train.

- We can also choose the maximum depth of the trees, this is a key to prevent overfitting. Deeper trees might give better results on the training test but could adapt poorly to new data.
- Additionally we can define the number of samples needed to split an internal node. If we set it to 5 for example any node with less than 5 samples becomes a leaf.
- Finally, we can tune the number of features to be considered when looking for the best one to split on. If we set it too high trees might be similar and if it is too low we might under fit the data.

The last two do not seem to have noticeable influence on the result for our task. We set the number of features to be considered to the standard default value which is the square root of the total number of features. To choose the best values for the maximum depth and the number of trees we can use the same method as for the logistic regression. The plots below display the average top-1 accuracy for the different pairs depth-number of trees on a hundred different training-validation splits.

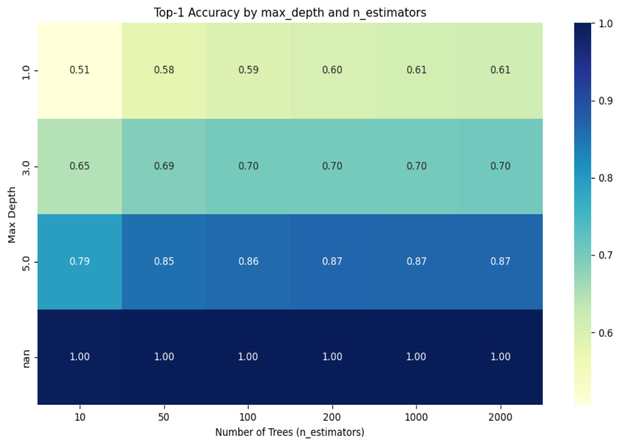


Figure 8: Top-1 accuracy on the training set for different tree depths and number of trees

On the training set we can see that models with no depth limit can learn the data perfectly and get 100% accuracy. The number of trees has the expected influence, more trees give better results. The results are totally different on the validation set. When the tree is too deep it cannot adapt to the new data and performs really poorly. The number of trees behave as expected, we need a good amount of estimators but going for more than a thousand trees does not give better results but takes way more time to train. We can settle for a maximum depth of 3 and a thousand trees which gives satisfying and stable results. Compared to the plain random forest it seems to give slightly better results, 47 to 53% in top-1 accuracy and it could adapt better to new data since we limit the depth of the trees. We have similar results to the logistic regression even though they seem more stable.

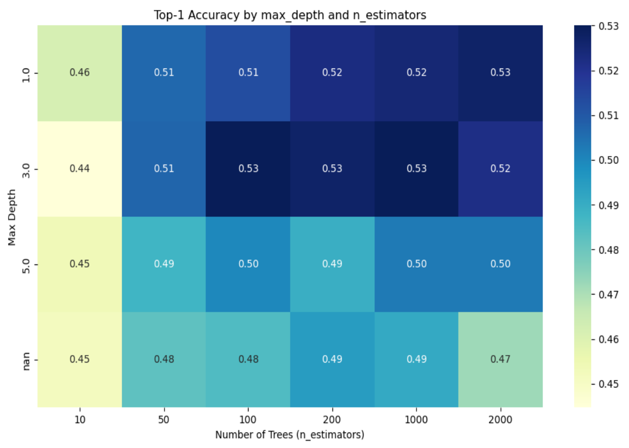


Figure 9: Top-1 accuracy on the validation set for different tree depths and number of trees

5.4 Gradient Tree Boosting with XGBoost

Finally, we will another really popular ensemble method, gradient boosting on trees. To do so we will use the `xgboost` python package a really popular an optimized implementation on gradient tree boosting algorithms. Before tuning the model and presenting results we will see how the algorithm works and what makes it efficient and popular.

5.4.1 Gradient tree boosting

Gradient tree boosting is an ensemble learning technique that builds a model by combining multiple decision trees sequentially. Unlike random forests, where trees are built independently, gradient boosting builds each new tree to correct the errors made by the previous ensemble. This is done by optimizing a loss function. The model prediction is the sum of outputs from all individual trees, each contributing to reducing the overall error. To prevent overfitting and improve generalization, we can include a regularization term that penalizes complex trees. Two additional techniques help control overfitting and improve training:

- Shrinkage (learning rate): scales the contribution of each new tree, allowing the model to learn slowly and effectively.

- Column (feature) subsampling: randomly selects a subset of features when building each tree, which reduces correlation between trees and speeds up computation.

As for pervious model we start with a plain baseline, which gives us accuracies around 48% for top-1 and 85% for top-3.

5.4.2 Hyper parameters Tuning

The xgboost classifier offers a lot of hyper parameters we can tune to mitigate overfitting and get better results. Among them some are similar to random forest such as the depth and the number of trees. Moreover, it supports L2 and L1 regularization and we can also change the learning rate. By doing a grid search on those parameters we can notice that best performing models all have around 200 estimators, the models with more do not provide better results and the training time is extended. The best learning rates are around 0.01. By fixing those parameters we can explore more precise values for depth and regularization strength. We have the following results, see Fig. 10 & 11, as expected with deeper trees and no regularization the model is almost perfect when computing the accuracy on the training set. On the validation set the

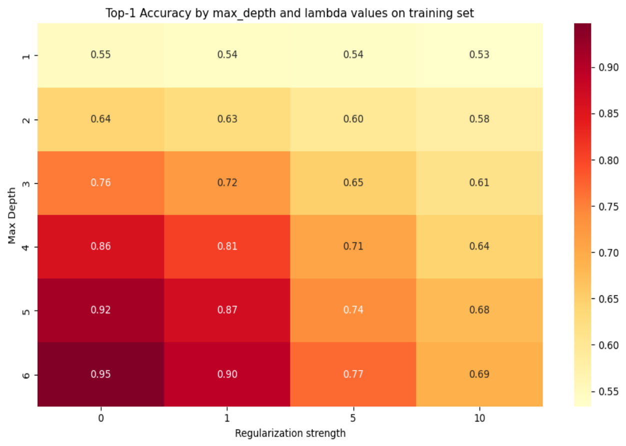


Figure 10: Top-1 accuracy on the training set for different tree depths and regularization strength

results are a bit unexpected, the model with the best score is the one with the highest tree depth and no regularization. However, we also have competitive results with a depth of 3 and a regularization strength of 5. This model will probably adapt to the new data and choosing this one may be safer.

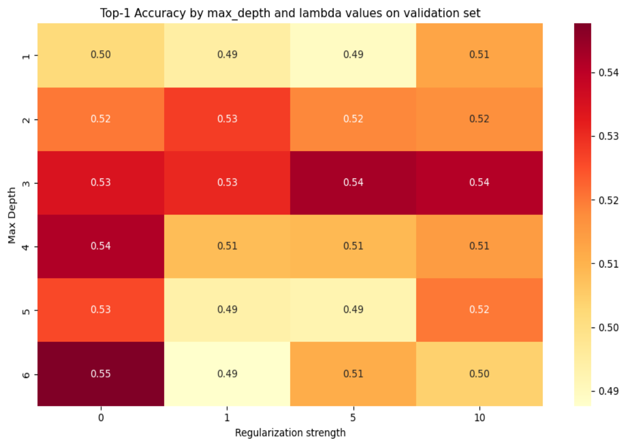


Figure 11: Top-1 accuracy on the validation set for different tree depths and regularization strength

6 Results on Testing Set

7 Limitations and Improvement

8 Appendix

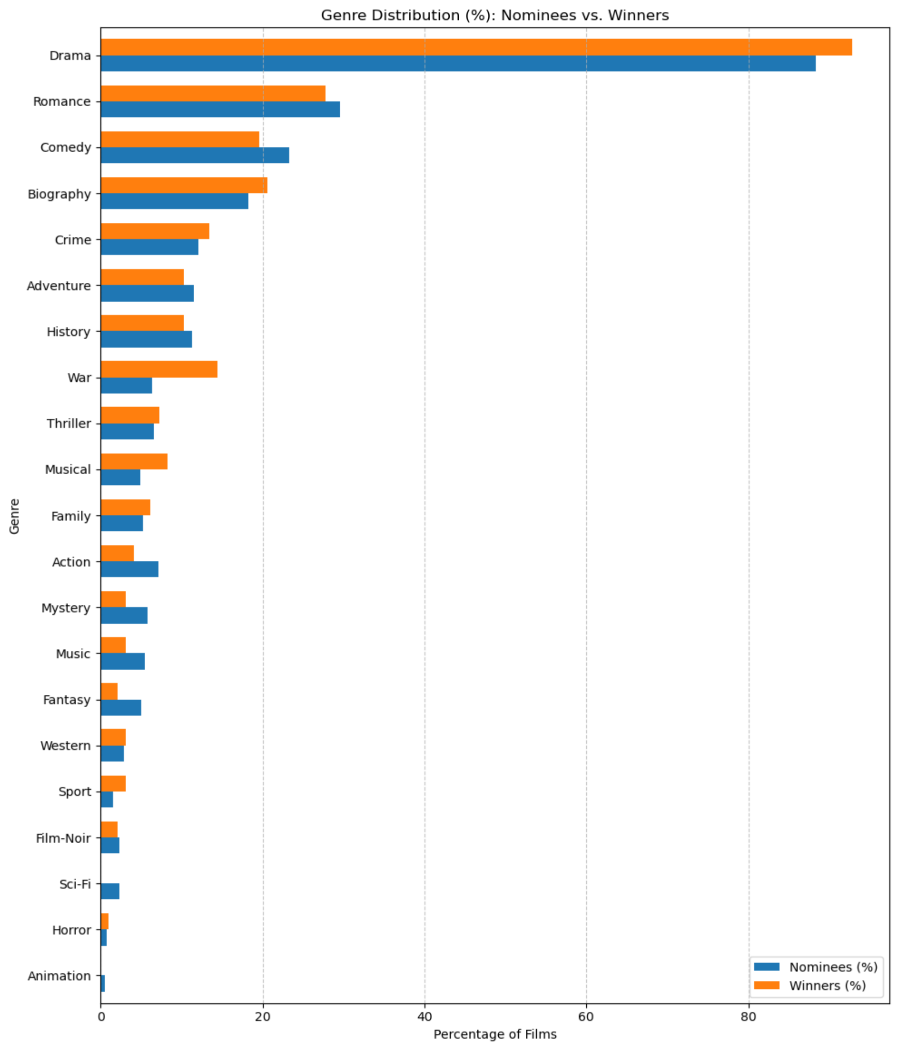


Figure 12: Repartition of genres among winners vs nominees

References

[1] https://en.wikipedia.org/wiki/Silhouette_(clustering)

[2] https://www.kaggle.com/code/vishnurapps/undersanding-kfold-stratifiedkfold-and-groupkfold

[3] Chen Tianqi et Guestrin Carlos. 2016. XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794.