Data Mining II Project:

Analyzing Data Insights from the IMDb Platform

Authors:

Bruno Barbieri, Chiara Ferrara, Ankit Kumar Bhagat

Academic Year 2024/2025

Contents

Introduction

The goal of this report is to illustrate the caracteristics of the given IMDb dataset, and to show key insights that can be obtained from it. In particular, the focus of many of the observations is based on aspects that could be useful for a product's creation and marketing, in order to optimize the chances of success of a product in the market.

1 Data Understanding and Preparation

TODO: distrib graphs The dataset contains 32 columns and 149531 rows of titles of different types. For each title, the dataset contains information regarding many different aspects. Table 1 lists the initial categorical features.

Feature	Description
`originalTitle`	Original title, in the original language (?)
`isAdult`	Whether or not the title is for adult
`canHaveEpisodes`	Whether the title can have episodes
`isRatable`	Whether the title can be rated by users
`titleType`	Type of the title (e.g., movie, tvseries)
`countryOfOrigin`	Countries where the title was primarily produced
`genres`	Genres associated with the title
`regions`	Regions for this version of the title
`soundMixes`	Technical specification of sound mixes
`worstRating` (ordinal)	Worst title rating
`bestRating` (ordinal)	Best title rating
`rating` (ordinal)	IMDB title rating class

Table 1: Initial categorical features of the IMDb dataset

Of the initial categorical attributes, the following were removed:

- `originalTitle`, as it did not provide particularly useful information;
- `isAdult`, as it was almost completely correlated with the *Adult* genre, so a logical OR operation was performed, and the genre only was kept; Interesting to note the fact that in our representation, being that the genres are represented through freq enc, we don't have the info
- `canHaveEpisodes`, as it was completely correlated with the title type being *tvSeries* or *tvMiniSeries*;
- `isRatable`, as it was always true;
- `soundMixes`, as it required some domain knowledge to be understood, as well as having issues with the values it contained.
- `worstRating` and `bestRating`, as they were always 1 and 10, respectively;
- `rating`, as it was obtainable from the `averageRating` continuous attribute, through a simple discretization.

Table 2 lists the initial numerical features.
Of the initial numerical attributes, the following were removed:

- `endYear`, was removed due to it not being meaningful for non-Series titles, and having around 50% of missing values for *tvSeries* and *tvMiniSeries*;
- `numVotes`, as it had a very high correlation with `ratingCount`;

1.1 Feature Engineering

- `totalImages`, `totalVideos` and `quotesTotal`, were merged through a simple sum operation into a single feature (`totalMedia`) because of their similar semantic meaning, as well as heavy right skewness;
- `awardWins` and `awardNominations`, were merged with the same procedure as above into `totalNominations`, since they represented the same concept;

Feature	Description
`startYear`	Release year of the title (series start year for TV)
`endYear`	TV Series end year
`runtimeMinutes`	Primary runtime of the title, in minutes
`numVotes`	Number of votes the title has received
`numRegions`	Number of regions for this version of the title
`totalImages`	Total number of images for the title
`totalVideos`	Total number of videos for the title
`totalCredits`	Total number of credits for the title
`criticReviewsTotal`	Total number of critic reviews
`awardWins`	Number of awards the title won
`awardNominations`	Number of award nominations excluding wins
`ratingCount`	Total number of user ratings submitted
`userReviewsTotal`	Total number of user reviews
`castNumber`	Total number of cast individuals
`CompaniesNumber`	Total number of companies that worked for the title
`averageRating`	Weighted average of all user ratings
`externalLinks`	Total number of external links on IMDb page
`quotesTotal`	Total number of quotes on IMDb page
`writerCredits`	Total number of writer credits
`directorCredits`	Total number of director credits

Table 2: Initial numerical features of the IMDb dataset

- `userReviewsTotal` and`criticReviewsTotal`,were merged with the same procedure as above into `reviewsTotal`, since they represented the same concept;
- `regions` and `countryOfOrigin` were merged through a simple union operation. The resulting feature was then represented trhough frequency encoding on the entire list, as well as counts of the number of countries from each continent. This resulted in eight new features (six continents, one for unknown country codes, and the last for the frequency encoding);
- `genre` attribute, it was observed that each record contained up to three genres, listed in alphabetical order—indicating that the order did not convey any semantic information about the title. To represent this information, three separate features were created, each corresponding to one of the genres. These features were encoded using frequency encoding, sorted in descending order of frequency across the dataset. A value of 0 was used to indicate missing genres—either when no genres were present or to fill the remaining slots when fewer than three were available;
- `runtimeMinutes` had a very high number of missing values (add %). Since the feature had high relevance in the domain, it was imputed with random sampling from a interquartile range, separately for each title type.
  eventually, add description of the imputation procedure for tasks which involved titleType

castNumber, writerCredits, directorCredits with and without totalCredits; deltacredits

2 Outliers

2.1 LOF

2.2 Isolation Forest

2.3 ABOD

3 Imbalanced Learning

3.1 Undersampling

3.2 Oversampling

3.2.1 SMOTE

4 Advanced Classification

In this section, classification results are showcased for two target variables: `averageRating` (properly binned into 5 classes), and `titleType` (with 6 classes).
We then applied multiple classification models using the data as preprocessed previously (see Section ??). The first target variable is `titleType`, which includes six categories: *movie*, *short*, *tvEpisode*, *tvSeries*, *tvSpecial*, and *video*. The classes are not equally distributed, with *tvSpecial* and *video* being significantly under-represented compared to the others. Entries labeled as *videoGame* were removed from both the training and testing sets, as they were too few to be useful for classification. The remaining categories were merged into broader groups according to the following mapping: *movie* and *tvMovie* were grouped as *movie*, *short* and *tvShort* were grouped as *short*, *tvSeries* and *tvMiniSeries* were grouped as *tvSeries*, while *tvEpisode*, *tvSpecial* and *video* were left unchanged. All feature columns were standardized using a *StandardScaler*. In addition, the variable `canHaveEpisodes` was removed prior to training, since it provides direct information about the target `titleType` and could therefore introduce data leakage.

4.1 Logistic Regression

For the classification of the `titleType` variable, the target was transformed into numerical labels using *LabelEncoder* and all numerical features were scaled with `StandardScaler` to ensure comparability across variables. Since the problem is multi-class, we chose to present the results using the *One-vs-Rest (OVR)* approach, which trains a binary classifier for each class. We also tested the *multinomial* strategy, but it was significantly slower and produced comparable results, so OVR was selected for efficiency.
To optimize the model, a hyperparameter search was performed using `RandomizedSearchCV` with `StratifiedKFold` (5 folds), ensuring that the class distribution was preserved in each fold. The solver was set to `saga`, and the parameters tested included the regularization term C (50 values logarithmically spaced between 10^{-3} and 10^{2}), `penalty` (`l2` and `l1`), and `class_weight` (`None` and `balanced`). The hyperparameter search was performed using `f1_macro` as the scoring metric to balance performance across all classes. The best parameters selected by the search were **C = 6.16**, **penalty = l1**, and **class_weight = balanced**.
For computational efficiency, the search was conducted on a 10% stratified sample of the dataset, which was approximately representative of the original class distribution. The final model was then refitted on the full dataset using the best parameters. Evaluation on the test set was carried out using the confusion matrix, classification report, and one-vs-rest ROC curves for each class. The main performance metrics are summarized in Table 3, and the corresponding ROC curves are shown in Figure 1.
From the results, we observe that the model achieves high performance for the `movie`, `short`, and `tvEpisode` classes, with F1-scores of 0.76, 0.81, and 0.89, respectively. The model performs less well on `tvSeries`, `tvSpecial`, and `video`, likely due to lower support in the dataset. Overall, the model reaches an accuracy of 0.75, a macro-average F1 of 0.54, and a weighted F1 of 0.78. These results highlight that while the model discriminates well among the more common classes, its performance is still limited for rarer classes. This pattern is also reflected in the ROC curves shown in Figure 1, where the model clearly separates the more frequent classes, while discrimination is more limited for the less common categories.
Furthermore, coefficient analysis highlighted the main drivers for each class. Globally, `totalNomitations`, `numRegions`, and `runtimeMinutes` were most influential. For individual classes, for example, `movie` is positively influenced by `totalNomitations` but negatively by `genre3`; `short` is positively influenced by `totalNomitations` and negatively by `companiesNumber`; `tvEpisode` is positively influenced by `Europe` and negatively by `numRegions`.

Table 3: Classification report for `titleType`

Class	Precision	Recall	F1-score
tvEpisode	0.89	0.88	0.89
movie	0.92	0.64	0.76
short	0.75	0.87	0.81
tvSeries	0.51	0.35	0.41
video	0.20	0.48	0.28
tvSpecial	0.07	0.49	0.11
Accuracy		0.75	
Macro avg	0.56	0.62	0.54
Weighted avg	0.83	0.75	0.78

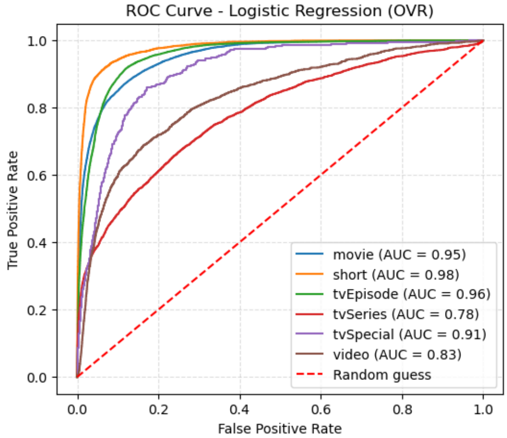


Figure 1: ROC Logistic Regression

Building upon the approach described for `titleType`, we trained a Logistic Regression model for the multi-class `rating_class` variable. All numerical features were scaled and, in addition, the categorical variable `titleType` was included as a predictor and processed via *One-Hot Encoding*.
Hyperparameters were optimized using again `RandomizedSearchCV` with `StratifiedKFold` (5 folds), performed on a 10% stratified sample of the data for computational efficiency. The scoring metric was `f1_macro`, and the search explored the same set of parameters used for `titleType`. The optimal configuration was found to be: **C = 0.08**, **penalty = l2** , and **class_weight = balanced**.
The best parameters were then used to fit the model on the full dataset, and evaluation on the test set was carried out.
The results for the `rating_class` target are reported in Table 4 and Figure 2. Overall, the model shows limited predictive ability (**accuracy = 0.27**, **macro F1 = 0.25**), with marked variability across classes. The extreme intervals, `[1,5)` and `[9,10)`, are detected with relatively high recall (0.44 and 0.57), but their low precision yields modest F1-scores. In contrast, the central ranges, which dominate the distribution, prove more difficult to classify: `[6,7)` reaches only 0.13 in F1, while `[7,8)` performs slightly better at 0.40.
Regarding ROC curves: discrimination is stronger for the extreme categories (AUCs of 0.75 and 0.76), whereas separability is weaker in the central intervals (`[6,7)`: 0.60, `[7,8)`: 0.64). This suggests that Logistic Regression tends to identify outlier ratings more effectively, while struggling to capture subtle differences in the middle of the rating scale.

Table 4: Classification report for `rating_class`

Class	Precision	Recall	F1-score
`[1, 5)`	0.20	0.44	0.27
`[5, 6)`	0.26	0.32	0.29
`[6, 7)`	0.39	0.08	0.13
`[7, 8)`	0.48	0.34	0.40
`[8, 9)`	0.25	0.22	0.24
`[9, 10)`	0.09	0.57	0.16
Accuracy		0.27	
Macro avg	0.28	0.33	0.25
Weighted avg	0.35	0.27	0.27

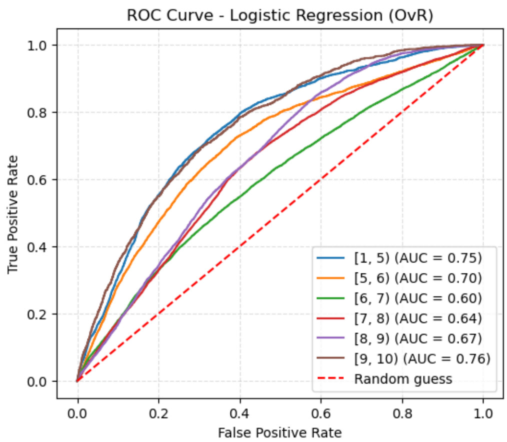


Figure 2: ROC curves for `rating_class` (OvR)

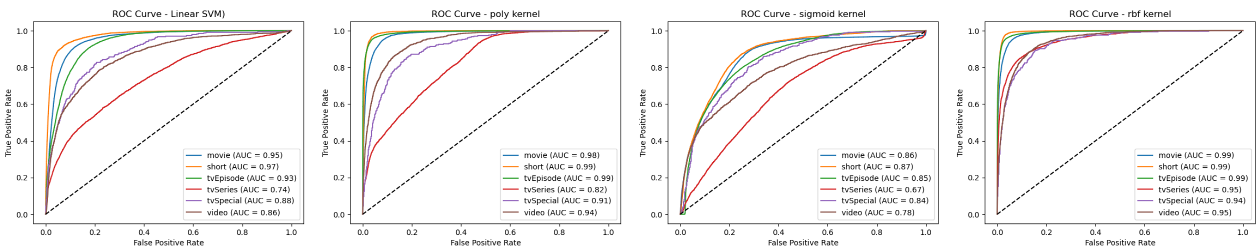
4.2 Support Vector Machines

We applied Support Vector Machines (SVM) to the `titleType` classification task. Both linear and non-linear kernels were explored in order to evaluate how decision boundary complexity influences predictive performance. The first experiment used a Linear SVM trained on the full dataset. A grid search with five-fold cross validation was carried out on the parameters $C \in \{0.01, 0.1, 1, 10, 100\}$ and $max\_iter \in \{1000, 5000, 10000\}$. The optimal

configuration, with $C = 100$ and $max\_iter = 1000$, achieved a test accuracy of 0.81. While precision and recall were high for majority classes (*movie*, *short*, *tvEpisode*), the classifier failed on *tvSeries*, *tvSpecial*, and *video*, indicating that a linear decision boundary is insufficient for this problem.
Non-linear kernels were then evaluated. A grid search was first performed on a stratified 10% subset of the training set to efficiently explore a wide range of hyperparameters for each kernel, since a full search on the complete dataset would have been computationally prohibitive. For the RBF kernel, C was varied from 0.01 to 1000 and γ between `scale` and `auto`. The polynomial kernel was tested with C from 0.01 to 100, degree 2–4, γ as `scale` or `auto`, and `coef0` 0 or 1. The sigmoid kernel was explored over C 0.01–100, γ `scale`/`auto`, and `coef0` 0 or 1. Remember to fix C!
The best configuration for each kernel, reported in Table 5, was then retrained on the full dataset and evaluated on the test set. Both RBF and polynomial kernels reached approximately 0.90 test accuracy, substantially outperforming the linear baseline and sigmoid. The RBF kernel was selected as the reference non-linear model due to slightly more stable results and improved recall on the under-represented classes.
ROC curves were used to evaluate class separability showing excellent separation for majority classes, although minority categories remained problematic.
I will change the text and explain the figures better.
To address class imbalance, the RBF kernel was retrained with `class_weight=balanced`, which penalizes misclassification of under-represented classes. This model reached a slightly lower overall accuracy of 0.84, but recall for *tvSpecial* and *video* improved, providing a more equitable classification across categories. Confusion matrices (Figure 4) illustrate that *tvSpecial* and *video* ... Analysis of the support vectors confirmed this effect. In the unbalanced RBF, nearly all points of minority classes became support vectors, while in the balanced model the total number of support vectors increased and was more evenly distributed across classes, indicating a more complex but fairer decision function.
Table 5 summarizes the main results, including the parameters used for each kernel and the corresponding test performance.

Table 5: Comparison of SVM models on the IMDb classification task.

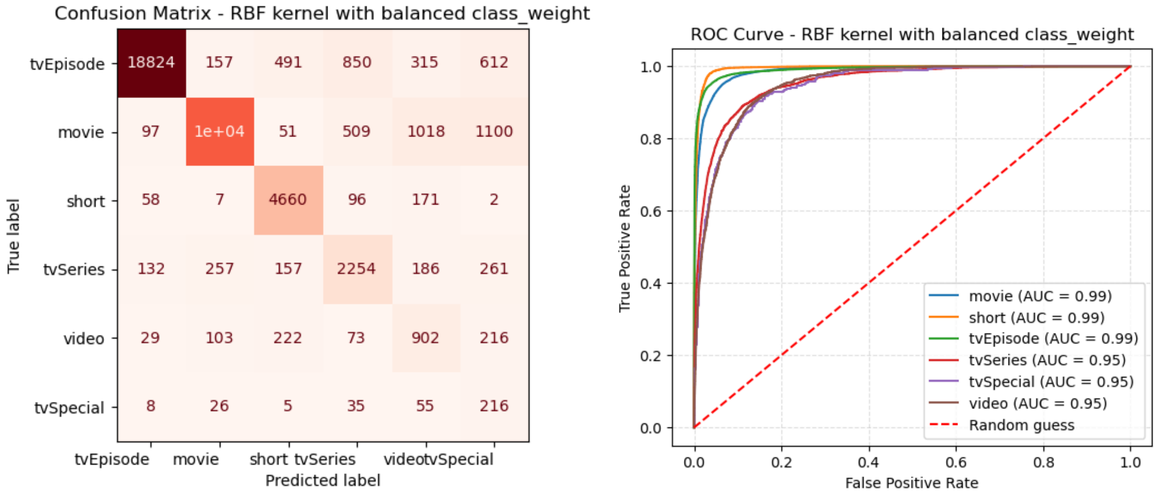
Model	**Best Params (main)**	**Test Accuracy**	**Macro F1-score**
Linear SVM	$C = 100$, $max\_iter = 1000$	0.81	0.45
RBF kernel	$C = 10$, $\gamma =$ scale	0.90	0.64
Polynomial kernel	$C = 10$, degree=3, $\gamma =$ auto	0.90	0.64
Sigmoid kernel	$C = 0.1$, $\gamma =$ auto	0.65	0.36
RBF (balanced)	$C = 10$, $\gamma =$ scale, balanced	0.84	0.65



(a) Linear SVM (b) Polynomial kernel (c) Sigmoid kernel (d) RBF kernel

Figure 3: ...

We also applied SVM to the `averageRating` classification task. The same kernels and hyperparameter search strategies were used as for `titleType`.
We then applied the same methodology to the `rating` classification task.
A Linear SVM was first tested using the same hyperparameter grid as before. The best configuration ($C = 100$, $max_iter = 1000$) achieved only 0.37 accuracy and a macro F1-score of 0.28, confirming (perchè confirming?) that a linear decision boundary is inadequate for this task.
We therefore moved to non-linear kernels. As for the previous experiment, a grid search on a stratified 10% subset of the training set was conducted. The optimal configurations were $C = 10$, $\gamma =$ auto for the RBF kernel, $C = 10$, degree=2, $\gamma =$ scale, coef0 $= 1$ for the polynomial kernel, and $C = 0.1$, $\gamma =$ auto, coef0 $= 0$ for the sigmoid kernel. These models were then retrained on the full dataset.
Results were generally lower than in the `titleType` task, with the RBF and polynomial kernels both reaching around 0.42 macro F1, while sigmoid performed substantially worse (macro F1 ≈ 0.36). Confusion matrices showed that the models correctly identified the most populated bins, but failed on the tails (*[1,6)* and *[9,10)*).



(a) Confusion Matrix RBF balanced

(b) ROC RBF balanced

Figure 4: Confusion Matrix and ROC Curve for the SVM (kernel rbf and balanced class weight) on the `titleType` classification task.

Finally, to mitigate imbalance, the RBF kernel was retrained with $class\_weight = balanced$. This increased recall on minority classes, at the cost of a slight drop in overall accuracy, producing a fairer but more complex decision function. However, even the balanced model struggled to capture the extremes of the rating distribution.

4.3 Ensemble methods

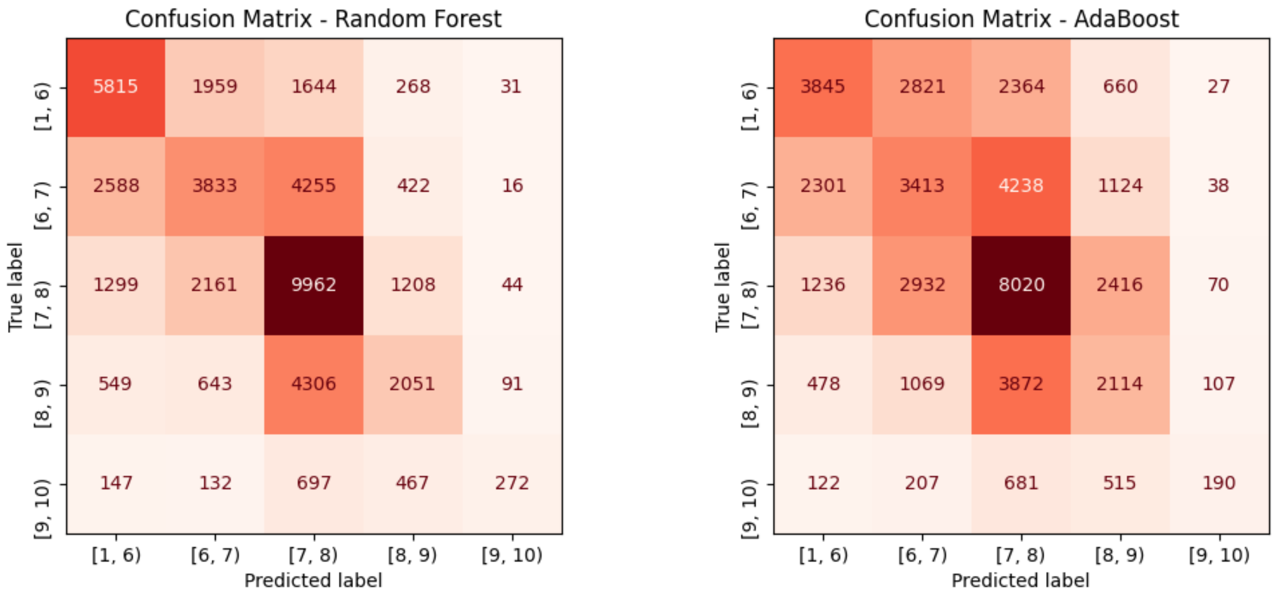
Boosting and Random Forest models were trained on the classification, while being optimized via Stratified Randomized Search with 5-fold cross-validation over a predefined hyperparameter space.

For the `averageRating` classification task, the best hyperparameters for the Random Forest model were: `n_estimators=42`, `max_depth=19`, `min_samples_split=4`, `min_samples_leaf=3`, `max_features=0.74`, `criterion='gini'`, `class_weight=None`.
For the AdaBoost model, the best hyperparameters were: `n_estimators=56`, `learning_rate=0.47`, `estimator__max_depth=` `estimator__min_samples_split=16`, `estimator__min_samples_leaf=16`.
The table below summarizes the classification report for both models.

Model	Accuracy	Macro Precision	Macro Recall	Macro F1-score
Random Forest	0.49	0.51	0.41	0.42
AdaBoost	0.39	0.40	0.33	0.34

Table 6: Performances of the models on the `averageRating` classification task.

The Random Forest model outperforms AdaBoost in all metrics. The biggest difference is found in the recall. Figure 7a and Figure 7b show the confusion matrices for the models.

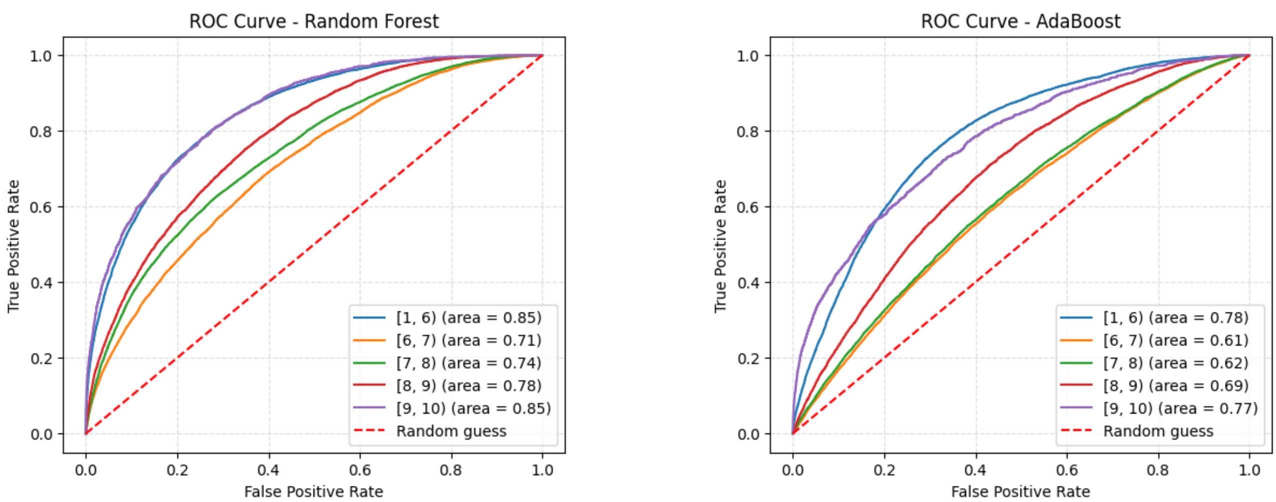


(a) Confusion Matrix - Random Forest

(b) Confusion Matrix - AdaBoost

Figure 5: Confusion matrices for the ensemle models on the `averageRating` classification task.

From these, it can be seen why the recall is much lower for AdaBoost, with regards to Random Forest: the former tends to classify less aggressively as the most represented class. It's also worth noting that Random Forest tends to assign most of the misclassifications to the adjacent classes, while AdaBoost spreads them more evenly across all classes.
Figures 8a and 8b show the ROC curves for the two models. From these representations, it can be seen



(a) ROC Curve - Random Forest

(b) ROC Curve - AdaBoost

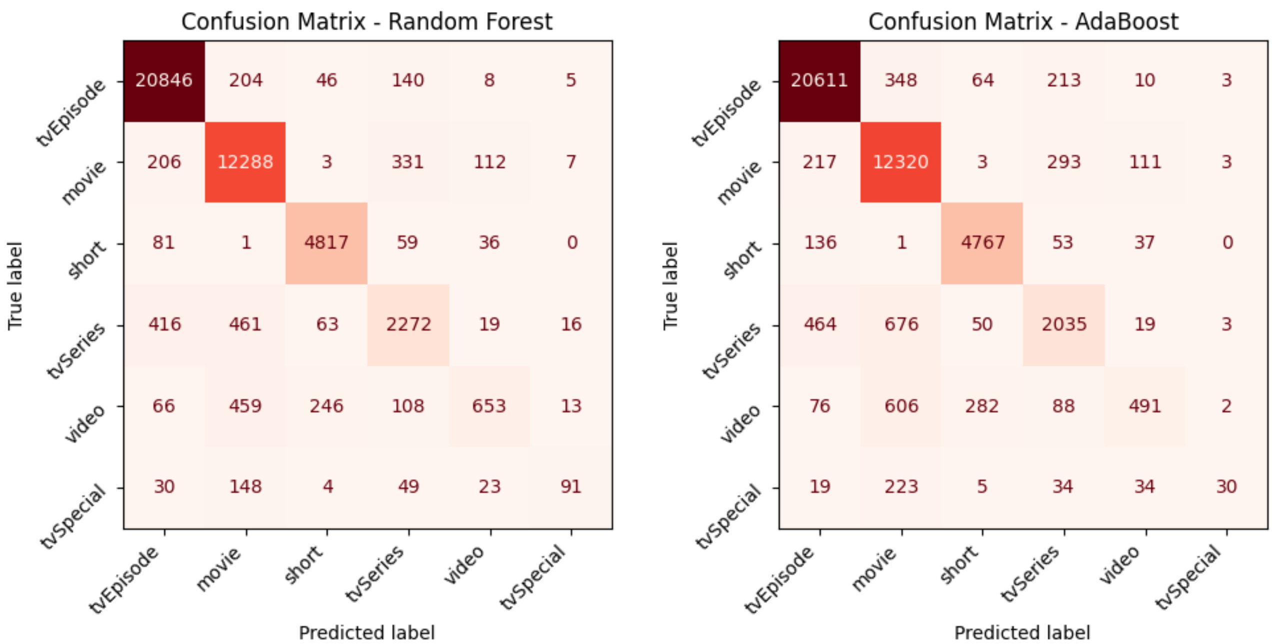
Figure 6: ROC curves for the ensemble models on the `averageRating` classification task.

that AdaBoost has poorer performances in all classes, but especially struggles with the under-represented ones, which lead to the biggest difference in Area Under The Curve (AUC).
On the `titleType` classification task, the best hyperparameters obtained for the Random Forest model were: `n_estimators=42`, `max_depth=19`, `min_samples_split=4`, `min_samples_leaf=3`, `max_features=0.74`, `criterion='gini'`, `class_weight=None`.
For the AdaBoost model, the best hyperparameters were: `n_estimators=56`, `learning_rate=0.47`, `estimator__max_depth=` `estimator__min_samples_split=16`, `estimator__min_samples_leaf=16`.
The table below summarizes the classification report for both models.

Model	Accuracy	Macro Precision	Macro Recall	Macro F1-score
Random Forest	0.92	0.84	0.71	0.75
AdaBoost	0.90	0.82	0.65	0.68

Table 7: Performances of the models on the `titleType` classification task.

Figure 7a and Figure 7b show the confusion matrices for the models.

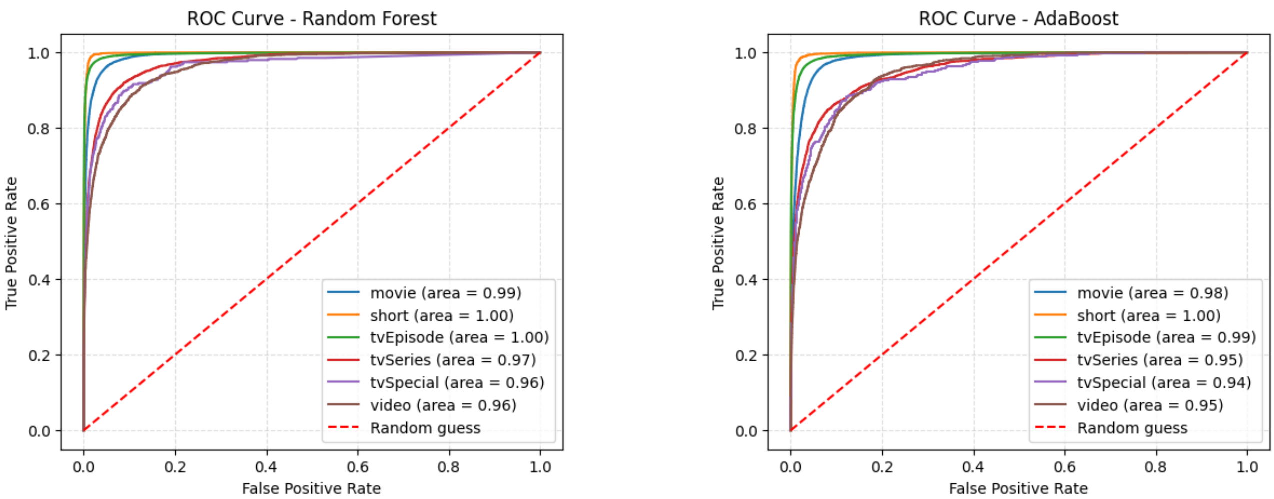


(a) Confusion Matrix - Random Forest

(b) Confusion Matrix - AdaBoost

Figure 7: Confusion matrices for the ensemle models on the `titleType` classification task.

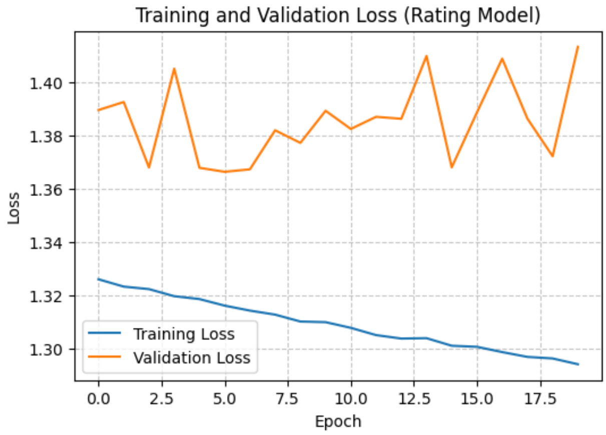
Both models perform well on the first four classes, while struggling with /texttttvideo and /textttttvSpecial, which are the most under-represented classes. In general, the two models show similar performances, with Random Forest generally outperforming AdaBoost by a slight margin. Contrary to the results, the models base their decisions on different feature importances: while Random Forest assigns over half of the importance to `runtimeMinutes`, AdaBoost spreads the importance evenly across multiple features, with the top being `runtimeMinutes` with around 15%.
The ROC curves for the two models are shown in Figure 8a and Figure 8b.



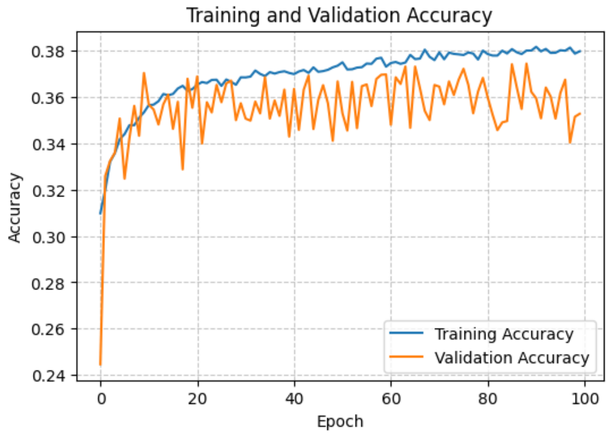
(a) ROC Curve - Random Forest

(b) ROC Curve - AdaBoost

Figure 8: ROC curves for the ensemble models on the `titleType` classification task.

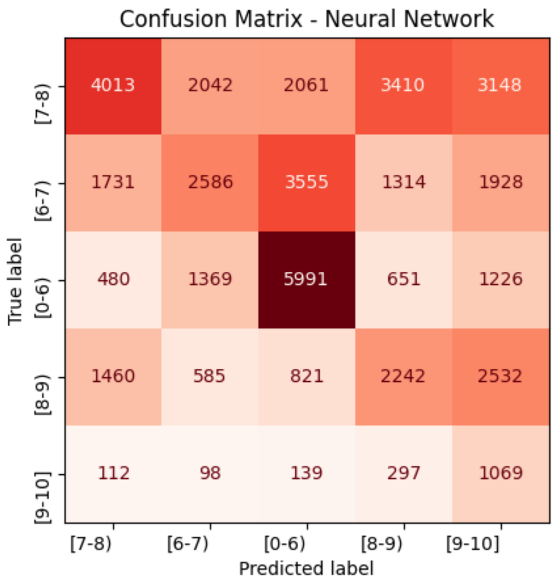


(a) Loss - Neural Network

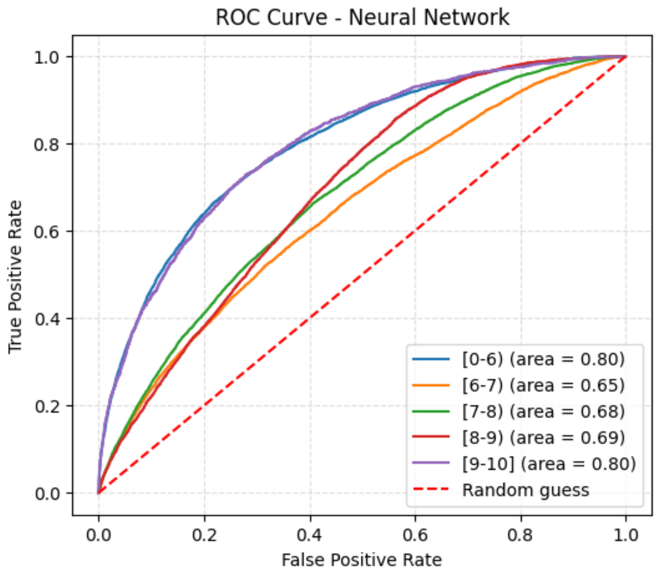


(b) Accuracy - Neural Network

Figure 9: Training and validation loss and accuracy for the neural network on the `averageRating` classification task.



(a) Confusion Matrix - Neural Network



(b) ROC Curve - Neural Network

Figure 10: Confusion Matrix and ROC Curve for the Neural Network on the `averageRating` classification task.

Again, similar performances are observed. The biggest difference seems to be found in the under-represented classes, which seem to have a bigger difference in Area Under The Curve (AUC).

4.4 Neural Networks

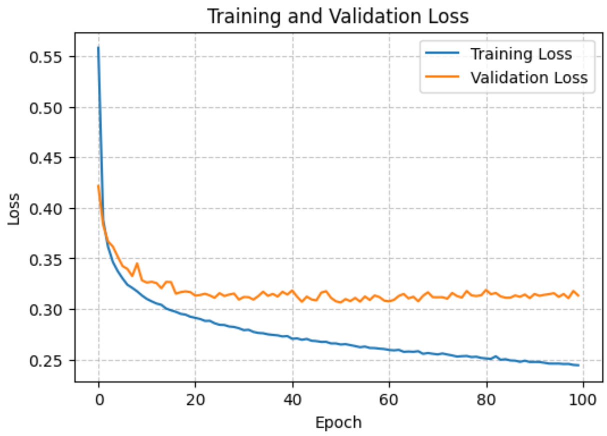
4.5 Model Comparison

5 Advanced Regression

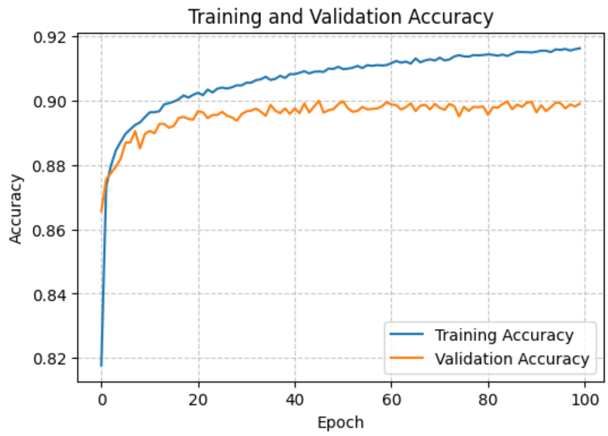
The chosen target variable is `averageRating`, which represents the average rating (on a 1–10 scale) assigned by IMDb users to each title. The exploratory data analysis showed that its distribution is approximately normal, with most titles concentrated in the range between 6 and 7.

5.1 Random Forest Regression

We applied the *Random Forest Regression* algorithm on the task. Before training the model, although not strictly necessary for tree-based models, we standardized the numerical features for consistency and transformed the categorical feature `titleType` using *One-Hot Encoding*.

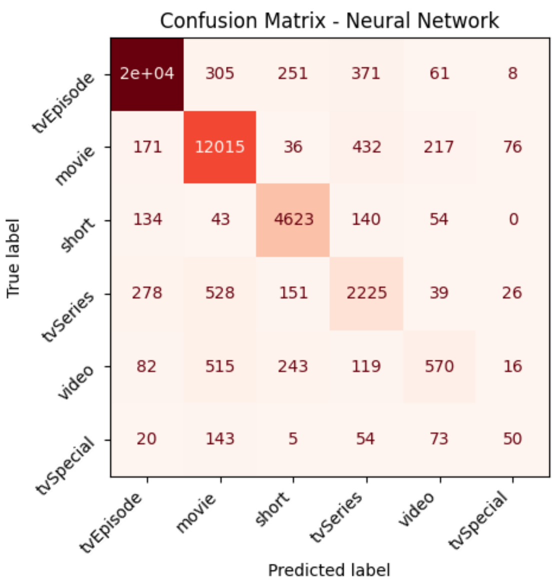


(a) Loss - Neural Network

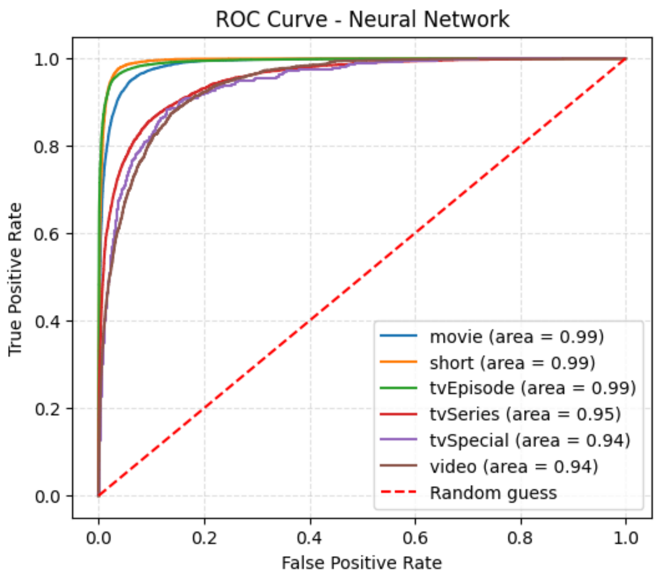


(b) Accuracy - Neural Network

Figure 11: Training and validation loss and accuracy for the neural network on the `titleType` classification task.



(a) Confusion Matrix - Neural Network



(b) ROC Curve - Neural Network

Figure 12: Confusion Matrix and ROC Curve for the Neural Network on the `titleType` classification task.

The hyperparameters were optimized using *RandomizedSearchCV* with 5-fold cross-validation, exploring different values for the number of trees (100, **200**, 300, 400, 500), the maximum depth (**None**, 10, 15, 20, 25, 30), the minimum number of samples required for a split (2, **5**, 10, 15) and for a leaf (**1**, 2, 4, 6), and the number of features considered at each split (sqrt, **log2**). The best hyperparameters found are highlighted in bold. The R^2 score was used as the evaluation metric during cross-validation.
The optimized Random Forest was first evaluated on the test set (results reported in Table 8). Subsequently, the model was retrained using only the 18 most important features identified through feature importance analysis. Feature selection was guided by a cumulative importance plot, which showed that these 18 features accounted for over 90% of the total importance, effectively reducing the dimensionality from the original 28 features without a significant loss in predictive performance.

Table 8: Performance of the Random Forest Regressor on the test set (full model vs reduced features).

Model	**MAE**	**MSE**	$\mathbf{R^2}$
Random Forest (All Features)	0.7536	1.0833	0.4033
Random Forest (Top 18 Features)	0.7550	1.0922	0.3984

The results, reported in Table 8, indicate that the Random Forest model achieves a mean absolute error below one point on the IMDb scale and explains around 40% of the variance in the target variable. Notably, the model trained on only the top 18 features performs almost identically to the full-feature model (R^2: 0.3984 vs 0.4033), showing that predictive power is concentrated in a limited subset of variables.
Feature importance analysis further highlighted that the most influential predictors include both numerical variables, such as `runtimeMinutes`, `startYear`, `ratingCount`, and `deltacredits`, and categorical variables derived from the encoding step, such as `titleType_tvEpisode`, among others.

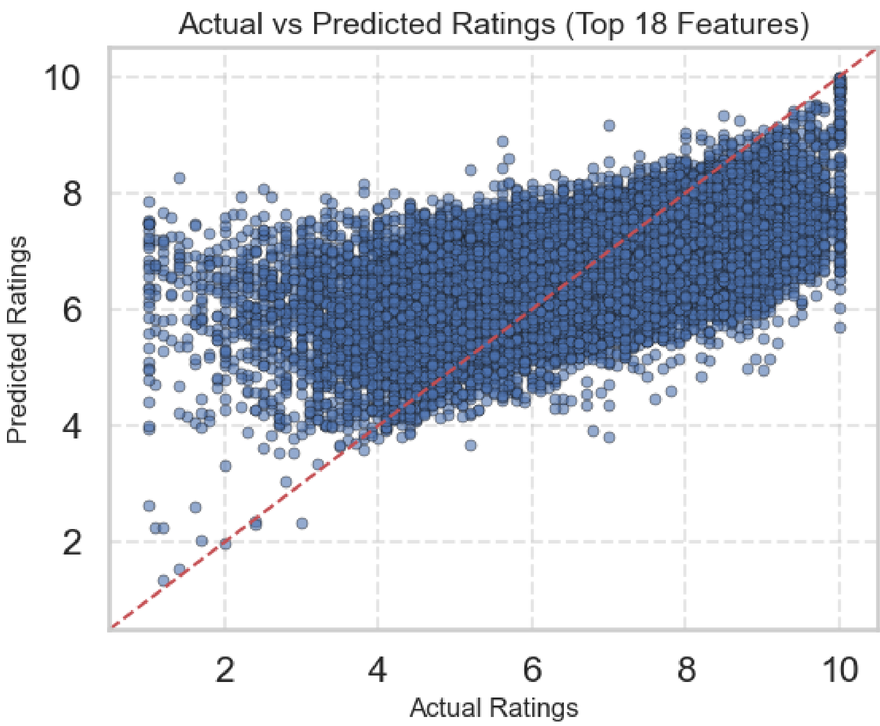


Figure 13: Actual vs Predicted `averageRating` for the Random Forest model trained on the top 18 features.

The scatter plot in Figure 13 shows that the predicted ratings roughly follow the trend of the actual ratings. Most predictions fall in the 6–7 range, consistent with the distribution of the target variable. While the model captures the general pattern, deviations occur, particularly at the extremes, which is consistent with the moderate R^2 of around 0.40. This indicates that the model explains a substantial portion of the variance, but there remains considerable unexplained variability.