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.

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

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`;

Figure 1 shows the distribution of the `averageRating` attribute.

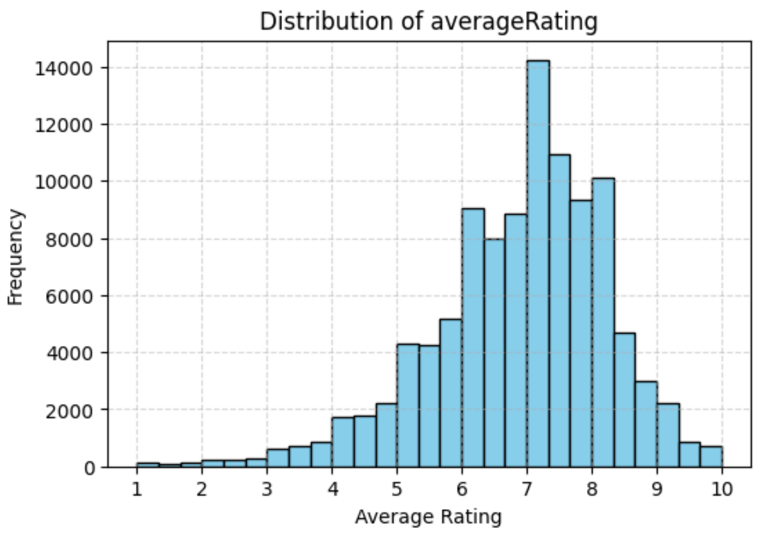


Figure 1: Distribution of the `averageRating` attribute

The distribution is Normal-like, with a peak around 7. This graph is particularly important because of the centrality of the feature in the classification and regression tasks.

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;
- `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 Finding Baseline

In this outlier analysis section, our primary objective was to identify the top outliers in the dataset using three well-established methods- `Local Outlier Factor, Isolation Forest`, and `Angle-Based Outlier Detection`. To establish a baseline for model performance, we initially employed two supervised learning algorithms- `Decision Tree` and `K-Nearest Neighbors`. To enhance the alignment between the later module tasks we defined a categorical target variable, `rating_bin`, by dividing the continuous `averageRating` into six distinct classes. The classes were defined as follows:

$$0 = [0\text{-}5);\ 1 = [5\text{-}6);\ 2 = [6\text{-}7);\ 3 = [7\text{-}8);\ 4 = [8\text{-}9);\ \text{and } 5 = [9\text{-}10]$$

The value count for each class was 12856, 19576, 37032, 49164, 25414, and 5489 respectively.
We then performed a grid search with cross-validation `(GridSearchCV)` on both models to identify the best hyperparameters. Since our dataset was large, we conducted the grid search on a 10% stratified sample to optimize computational efficiency. For K-NN, the optimal parameters were-

$$\text{metric} = \text{'manhattan'},\ \text{n\_neighbors} = 122,\ \text{weights} = \text{'distance'}$$

However, to strike a balance between accuracy and computational efficiency, we restricted our model to use `50 n_neighbors`. For Decision Tree, the optimal parameters were-

$$\text{criterion} = \text{'gini'},\ \text{max\_depth} = 10,\ \text{min\_samples\_leaf} = 4,\ \text{min\_samples\_split} = 10,\ \text{splitter} = \text{'random'}$$

We trained both models on the entire training dataset and evaluated their performance on the test set. We split our training dataset into training (80%) and validation (20%) sets. We computed the baseline accuracy without removing any outliers, which was `36% for K-NN` and `32% for Decision Tree`.

2.2 Finding Threshold

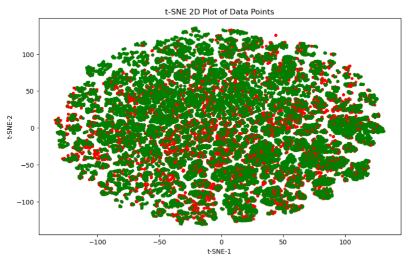
Next, we applied the three outlier detection methods to identify and remove outliers from the training dataset. We experimented with different contamination levels `(0.01, 0.05, 0.1)` to determine the optimal proportion of outliers to remove. After removing the identified outliers, we retrained both models on the cleaned training dataset and evaluated their performance on the test set. We observed that removing outliers generally improved model accuracy, but different contamination levels had no significant impact on the results as shown in table 3. Only the results from LOF with a contamination level of 0.1 showed an improvement of 1%. Therefore, we fixed the contamination level at 10% for all three methods to maintain consistency. Subsequently, we combined the results of the three methods by aggregating the outlier score and removed those rows that were flagged as outliers by at least `two out of the three methods`, which accounted for approximately `3%` of the dataset.

		LOF	**IF**	**ABOD**
1%	**KNN**	42	42	41
	DT	39	39	40
5%	**KNN**	42	42	41
	DT	39	39	40
10%	**KNN**	**43**	42	41
	DT	**40**	39	40

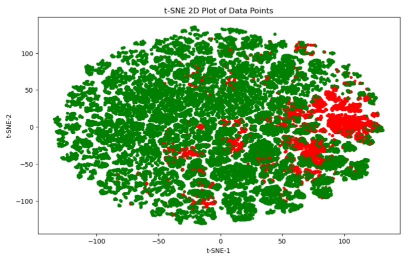
Table 3: Accuracy at different threshold

2.3 Outlier Detection Conclusion

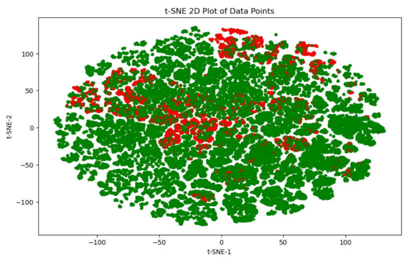
In conclusion, our anomaly detection task revealed a consistent trend: both models `KNN` and `DT` performed better than the baseline after the removal of outliers. This outcome sets the foundation for our subsequent focus on model explainability in the later sections.
From the T-SNE visualization, we observed that `LOF` predominantly identified outliers at the edges of the data distribution 2a, whereas `IF` detected them primarily within a clustered region 2b. `ABOD`, on the other hand, captured outliers both at the edges and within dense clusters 2c. This variation can be explained by the fundamental differences between the algorithms: `LOF` emphasizes anomalies in low-density regions, while `IF` isolates points distant from the main distribution. `ABOD`, which evaluates outliers based on the angular relationships of points, reflects a hybrid behavior by marking anomalies in both sparse and dense regions. Importantly, since the dataset is dominated by a single high-density region, most points lack anomalous neighbors, explaining why `ABOD` highlighted relatively fewer strong anomalies. To ensure computational efficiency, we implemented the 'fast' version of `ABOD`, which approximates angle-based detection using a specified number of neighbors rather than exhaustively computing angles across all data points.
Overall, as we can see from table 3 the removal of outliers with different threshold led to only marginal gains in predictive performance. However, the differences in detection patterns across methods provide valuable insights into the structure and density of the data distribution, reinforcing the importance of combining multiple approaches for a more comprehensive anomaly detection strategy. Therefore, we combined the results of all three methods to identify the common outliers.



(a) Local Outlier Factor



(b) Isolation Forest



(c) Angle-Based Outlier Detection

Figure 2: Comparison of T-SNE visualizations for different outlier detection methods.

3 Imbalanced Learning

We then proceeded with imbalance learning task, we used the same variable `averageRating` to categorize the data into three classes: low (0-4], medium (4-7], high (7-10]. The distribution of the classes is 4979, 67394, 72539 respectively, where the low class accounts for approx 3.5% of the dataset, which is highly imbalanced. We then applied Decision Tree classifier to the initial dataset and achieved 63% accuracy. Then we performed `RandomSearchCV` to tune the hyperparameters of the Decision Tree Classifier and achieved 67% accuracy. The best parameter were

```
random_state=42, criterion='gini', max_features=None, max_depth=None, min_samples_leaf=10,
                    min_samples_split=2, splitter = 'random'
```

Our focus was to improve the recall of the low class, so we plotted the confusion matrix to see the number of true positives. As we can see from the table 4 the number of true positives for the low class had increased with different imbalanced learning techniques. Finally, we applied different imbalance learning techniques to improve the performance of the model.

Model Name	Accuracy	True Positives	Recall
Initial DT	63	200	0.20
HyperParameter Tuned DT	67	95	0.09
SMOTE	**62**	**449**	**0.44**
ADASYN	62	439	0.43
Resample	53	576	0.56
ENN	68	346	0.21
AllKNN	66	469	0.30
RandomUnderSampler + ENN	43	1307	0.82
DBSCAN + K-Means	56	907	0.57

Table 4: Imbalanced Learning Model Performance on Low (0-4] class

3.1 Oversampling

In the oversampling techniques, we applied `Synthetic Minority Oversampling Technique (SMOTE)`, and `Adaptive Synthetic Sampling (ADASYN)` to increase the size of the minority class. Despite the different methodologies, `SMOTE` and `ADASYN`, both approaches resulted in nearly identical performance 62% accuracy and recall around 0.44-0.43 for the low class respectively 4. This highlights that while the oversampling strategy can influence which minority samples are emphasized, it does not fundamentally change the class distribution dynamics and may even introduce synthetic noise, thereby limiting overall accuracy improvements. As in our dataset both methods allowed the model to achieve similar overall accuracy and recall, indicating that the choice between these two oversampling techniques may not significantly impact performance in this specific context. The key difference lies in the sample generation strategy: `SMOTE` treats all minority samples equally during interpolation, whereas `ADASYN` focuses more on difficult-to-learn samples by adaptively weighting them according to their density in the feature space.. However, the similarity in results suggests that the dataset characteristics were such that both oversampling strategies provided comparable benefit in improving the model's sensitivity to minority classes.

3.2 Undersampling

In the undersampling techniques, we applied `resample`, `Edited EditedNearestNeighbours (ENN)`, `AllKNN`, combination of `RandomUnderSampler + ENN` and clustering based undersampling using `DBSCAN + K-Means`. The best performance was achieved by `RandomUnderSampler + ENN` with 43% accuracy and 0.82 recall for the low class. `DBSCAN` and `K-Means` achieved 56% accuracy and 0.57 recall for the low class. The results were varied and sometimes counterintuitive, making it challenging to derive a single optimal strategy. For instance, the simple `resample` method achieved an accuracy of 53% with a recall of 0.56, indicating a balanced compromise between correctly predicting the majority class and detecting minority class instances. In contrast, `ENN` prioritized cleaning noisy or borderline samples, resulting in higher overall accuracy of 68% but a significantly lower recall of 0.21, reflecting poor sensitivity to the minority class. Similarly, `AllKNN`, which removes samples misclassified by all nearest neighbors, yielded a moderate accuracy of 66% and a recall of 0.30, suggesting that while it reduces noise, it also inadvertently removes some informative minority samples. The combination of `RandomUnderSampler + ENN` achieved an extreme result, with a low accuracy of 43% but a very high recall of 0.82, demonstrating that aggressive removal of majority class instances can drastically improve detection of the minority class at the cost of overall predictive correctness. Finally, the clustering-based approach (DBSCAN + K-Means)using `DBSCAN + K-Means` produced an accuracy of 56% and a recall of 0.57, which is relatively balanced, highlighting that clustering can preserve the structure of the data while reducing the majority class. In a nutshell we can say that: simple random resampling treats all samples equally, ENN and AllKNN remove noisy or ambiguous majority samples based on neighborhood consistency, the combined `RandomUnderSampler + ENN` aggressively removes majority instances to favor recall, and clustering-based undersampling attempts to retain representative majority samples while reducing redundancy. Overall, these experiments highlight the inherent trade-off in undersampling strategies between detecting minority classes and maintaining high overall accuracy, emphasizing that the choice of method should be guided by the specific priorities of the task. For instance, in credit card fraud detection, aggressively undersampling legitimate transactions may help the model catch more fraudulent cases `(higher recall)` but can also cause more false alarms among genuine transactions texttt(lower overall accuracy). The choice of undersampling method should therefore depend on whether catching fraud or avoiding false alarms is more critical.

3.3 Decision Threshold

After experimenting with both increasing and decreasing the training samples, we focused on the decision tree algorithm and attempted to adjust its decision threshold and class weights in the hypertuned model. We assigned higher importance to the minority class, `low` class, using the weights obtained from the `balanced` parameter, which were

'high': 0.6659, 'low': 9.7015, 'medium': 0.7167.

So, in this way the model would penalize misclassifications of the `low` class more heavily during training. It was surprising to see that even experimenting with different values of weights and assigning very high weight to `low` class could not lead to an increase in accuracy. In an attempt to further increase the recall of the `low` class, we also experimented with adjusting the decision threshold to shift the cutoff point for predicting each class after training. We modified the decision threshold to extreme values such as [0.1, 0.9, 0.1], the model's accuracy showed only marginal changes in decimals and consistently struggled to surpass the baseline accuracy of 67%. This suggests that the limitation may not be due to the model parameters or algorithmic tuning alone, but rather is inherently linked to the characteristics of the data. Both methods share the goal of improving minority class detection, yet their similarity lies in the fact that they cannot generate additional information; they only reweigh or reassign predictions based on existing patterns. The lack of improvement indicates that the features may not provide sufficient discriminatory power to separate the classes effectively, or that the classes themselves are inherently overlapping in the feature space.

3.4 Imbalanced Learning Conclusion

In conclusion, we can say that in an attempt to strive for both high **accuracy** and high **recall**, we carried out a series of extensive experiments. While our initial `DecisionTree` model achieved a reasonable accuracy of 63%, the recall was extremely low (0.20), reflecting its bias towards the majority class. After hyperparameter tuning, the accuracy improved to 67% but recall dropped further to 0.09. This counter-intuitive outcome can be attributed to the fact that tuning optimized for accuracy rather than recall, causing the model to become even more conservative in predicting the minority class. Oversampling techniques such as `SMOTE` and `ADASYN` substantially increased recall (0.44 and 0.43, respectively) because they synthetically generated minority samples, allowing the classifier to better recognize underrepresented patterns. However, this came at the cost of accuracy (dropping to 62%), since synthetic data can blur class boundaries and make the classifier more prone to misclassifications in the majority class. Undersampling approaches such as `Resample` and `RandomUnderSampler + ENN` achieved very high recall (0.56 and 0.82, respectively), but their accuracy decreased sharply (to 53% and 43%). The reason is that by aggressively reducing the majority class, these methods forced the classifier to focus disproportionately on the minority class, which inflated recall but degraded its ability to generalize across the full distribution. To understand the trade-offs more we looked at feature importance and we found out that the feature `tvEpisode` emerged as the most significant contributor across multiple models, while other features such as `startYear`, `genre1`, `genre2`, `castNumber`, and `totalCredits` also played crucial roles. However, the relative importance of these features varied across different models, indicating that no single feature consistently dominated in all approaches. Thus we were satisfied with the results of SMOTE and ADASYN as they provided a good balance between accuracy and recall.

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 underrepresented 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 = 2.95**, **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 5, and the corresponding ROC curves are shown in Figure 3.

Table 5: Classification report for `titleType`

Class	Precision	Recall	F1-score
tvEpisode	0.90	0.88	0.89
movie	0.92	0.65	0.76
short	0.74	0.87	0.80
tvSeries	0.50	0.35	0.41
video	0.21	0.47	0.29
tvSpecial	0.07	0.52	0.12
Accuracy		0.75	
Macro avg	0.56	0.62	0.54
Weighted avg	0.83	0.75	0.78

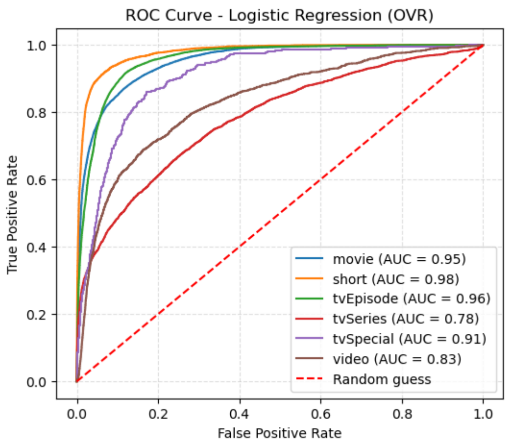


Figure 3: ROC Logistic Regression

From the results, we observe that the model achieves high performance for the `movie`, `short`, and `tvEpisode` classes, with F1-scores of 0.76, 0.80, 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 3, 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`.
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 6 and Figure 4. 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 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 distinguishing the most polarized cases, while it struggles to capture subtle differences in the middle of the rating scale.

Table 6: 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

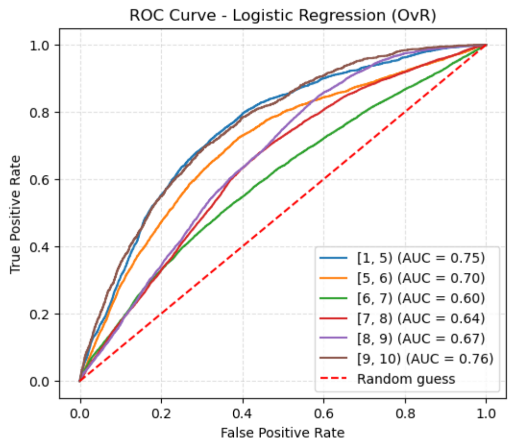


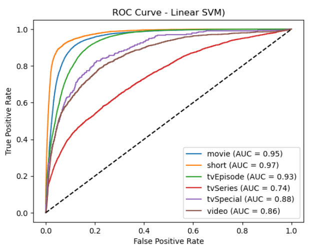
Figure 4: ROC curves for `rating_class` (OvR)

4.2 Support Vector Machines

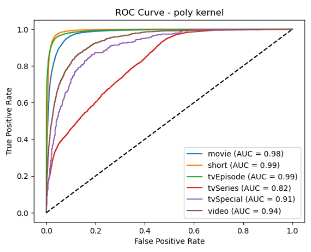
We applied Support Vector Machines (SVM) to the `titleType` classification task. Both linear and non-linear kernels were explored in order to evaluate how decision boundary complexity influences predictive performance. The first experiment used a Linear SVM trained on the full dataset. A grid search 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. All kernels were tested with C ranging from 0.01 to 100 and γ set to `scale` or `auto`. For the polynomial kernel, the degree was varied between 2 and 4 and `coef0` set to 0 or 1. For the sigmoid kernel, `coef0` was also explored at 0 and 1.
The best configuration for each kernel, reported in Table 7, 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`. 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. The corresponding ROC curves (Figure 6b) show high separability for all classes, with AUC values of 0.99 for *movie*, *short*, and *tvEpisode*, and 0.95 for *tvSeries*, *tvSpecial*, and *video*, confirming that the balanced model achieves good discrimination for the minority categories too.
Then an analysis of the support vectors was conducted. 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 but was more evenly distributed across classes, indicating a more complex but fairer decision function.
Table 7 summarizes the main results, including the parameters used for each kernel and the corresponding test performance.
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.

Table 7: Comparison of SVM models on the IMDb classification task.

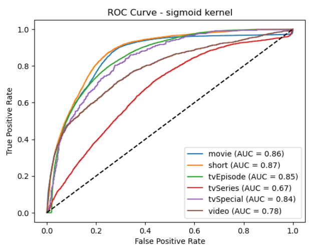
Model	**Best Params (main)**	**Test Accuracy**	**Macro F1-score**
Linear SVM	$C = 100$, $max\_iter = 1000$	0.81	0.45
RBF kernel	$C = 10$, $\gamma =$ scale	0.90	0.64
Polynomial kernel	$C = 10$, degree=3, $\gamma =$ auto	0.90	0.64
Sigmoid kernel	$C = 0.1$, $\gamma =$ auto	0.65	0.36
RBF (balanced)	$C =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



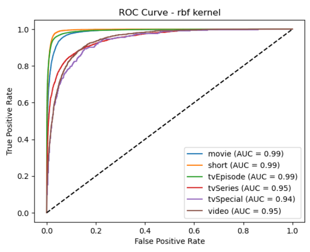
(a) Linear SVM



(b) Polynomial kernel



(c) Sigmoid kernel



(d) RBF kernel

Figure 5: ...

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)*).
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 tasks, while being optimized via Stratified Randomized Search with 5-fold cross-validation over a predefined hyperparameter space.

Table 8 shows the best hyperparameters found for Random Forest and AdaBoost on the `averageRating` task.

For Random Forest, a relatively high maximum depth as well as low values for minimum samples per split and leaf indicate that the individual trees are quite complex.

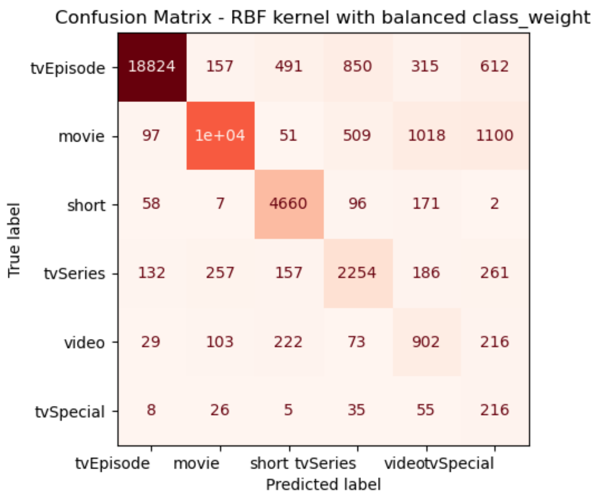
For AdaBoost, the base estimator is again characterized by high complexity. The learning rate is moderate, leading to reasonably fast learning.

Random Forest	
`n_estimators`	97
`max_depth`	19
`min_samples_split`	2
`min_samples_leaf`	6
`max_features`	0.91
`criterion`	gini
`class_weight`	None
AdaBoost	
`n_estimators`	67
`learning_rate`	0.45
`estimator__max_depth`	17
`estimator__min_samples_split`	7
`estimator__min_samples_leaf`	1

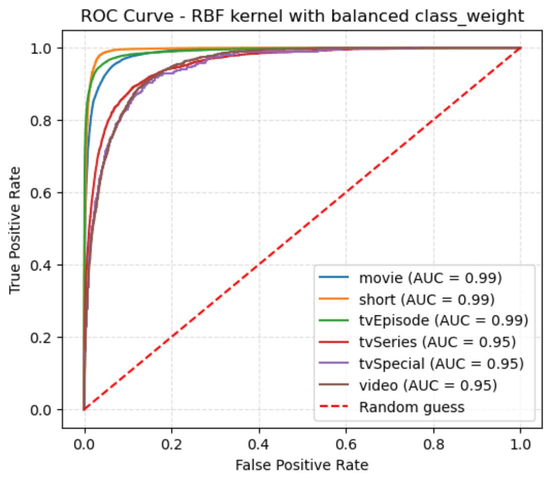
Table 8: Hyperparameters found for ensemble models.

The two models use a different number of estimators, with Random Forest employing a larger ensemble. This is likely due to the fact that Random Forest builds trees independently, while AdaBoost adds weak learners sequentially, focusing on correcting previous errors.
Another interesting aspect is the distribution of feature importances. Both models assign most of the importance to the same set of features:

- `ratingCount`



(a) Confusion Matrix RBF balanced



(b) ROC RBF balanced

Figure 6: Confusion Matrix and ROC Curve for the SVM (kernel rbf and balanced class weight) on the `titleType` classification task.

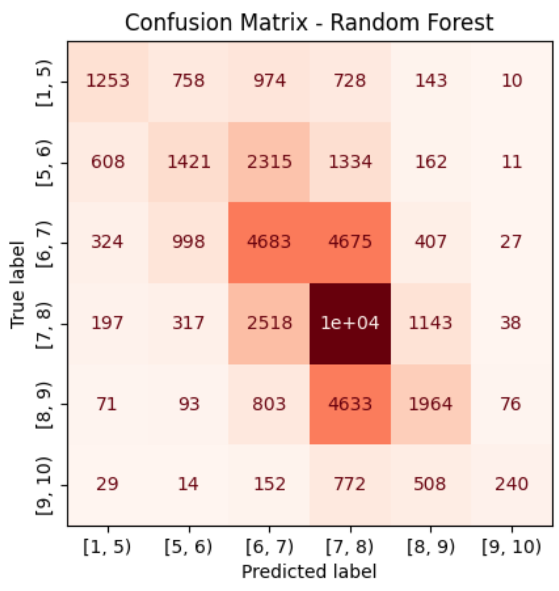
- `startYear`
- `runtimeMinutes`
- `deltaCredits`

Random Forest assigns about 50% of the total importance to these features, spreading it evenly among them. AdaBoost assigns about 75%, 41 of which is attributed to `ratingCount` and `startYear`.
Table 9 summarizes the classification report for both models. Although the performances of the two models are similar, Random Forest outperforms AdaBoost in all metrics but Macro-averaged Precision. The weighted averaged metrics, here not reported, also give the edge to Random Forest.

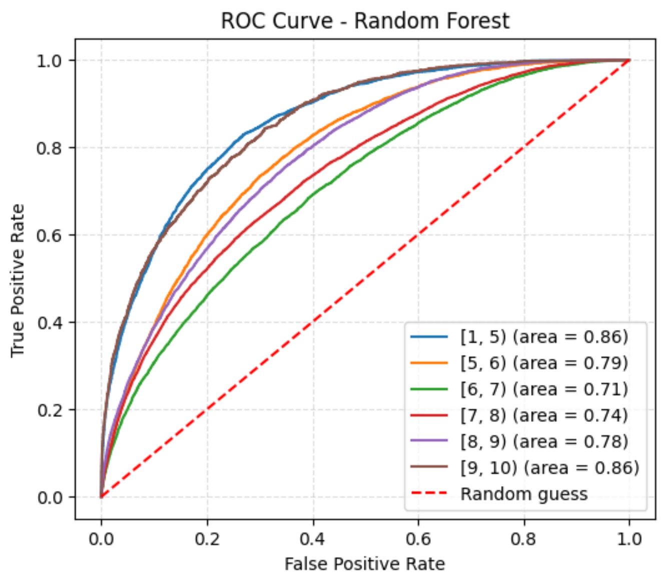
Model	**Accuracy**	**Macro Precision**	**Macro Recall**	**Macro F1-score**
Random Forest	0.45	0.47	0.35	0.37
AdaBoost	0.43	0.49	0.33	0.35

Table 9: Performances of the models on the `averageRating` classification task.

Performances of both models are limited, and the results show that the different classes are not well separated. This can be observed in Random Forest's confusion matrix in figure 7a. The matrix also shows that many of the misclassifications occur within adjacent classes, which is expected given the ordinal nature of the target variable. In particular, a lot of confusion occurs between the most represented classes, `[6,7)`, `[7,8)` and `[8,9)`. This is likely due to the fact that these classes cover a wide variety of titles that perform similarly in terms of ratings. Additionally, as seen in graph 1, Many of the titles have ratings that fall close to the boundaries between these classes, making it more difficult for the model to distinguish between them.



(a) Confusion Matrix - Random Forest



(b) ROC Curve - Random Forest

Figure 7: Confusion matrix and ROC Curve for Random Forest on the `averageRating` classification task.

Figure 7b shows the ROC curve for AdaBoost. AUCs are generally high, with the lowest ones being 0.70 and 0.72, although these correspond to the highest-represented classes. This suggests that the models are better at separating the extreme classes, while the more frequent intermediate ones are more difficult to distinguish.

Table 10 shows the best hyperparameter configurations found for Random Forest and AdaBoost on the `titleType` task.

For Random Forest, the trees are quite deep and complex, with low minimum samples per split and leaf.

For AdaBoost, the learning rate is very low, leading to slow, incremental learning. The base estimator is a decision tree with considerable depth and higher minimum samples per split and leaf than Random Forest's trees, indicating that each weak learner is slightly less complex.

Again, the two models use a very different number of estimators, with Random Forest employing a much larger ensemble. For feature importances, both models assign over half of the importance to `runtimeMinutes`, indicating a high dependence on this feature for classification.

Random Forest	
`n_estimators`	42
`max_depth`	19
`min_samples_split`	4
`min_samples_leaf`	3
`max_features`	0.74
`criterion`	gini
`class_weight`	`None`
AdaBoost	
`n_estimators`	11
`learning_rate`	0.03
`estimator__max_depth`	14
`estimator__min_samples_split`	18
`estimator__min_samples_leaf`	10

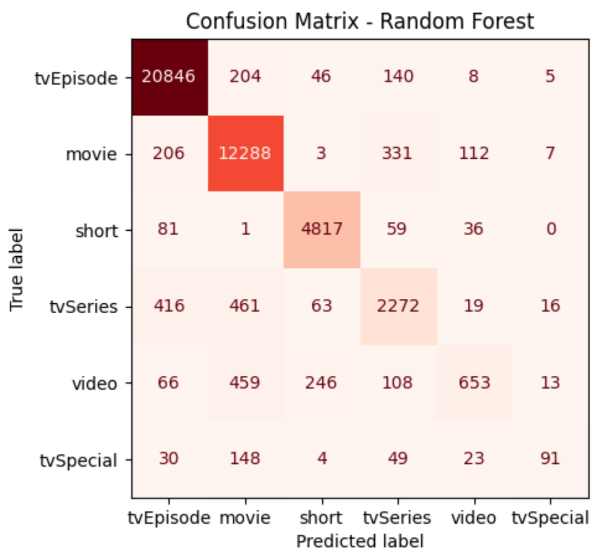
Table 10: Hyperparameter search space for ensemble models.

Table 11 summarizes the classification report for both models.

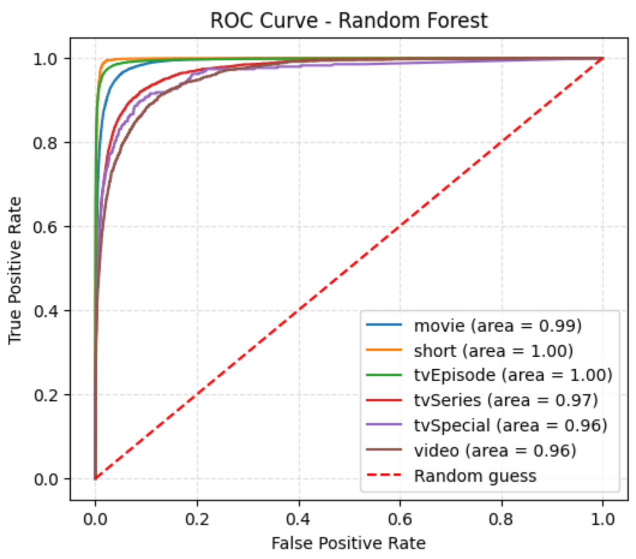
Model	**Accuracy**	**Macro Precision**	**Macro Recall**	**Macro F1-score**
Random Forest	0.92	0.84	0.71	0.75
AdaBoost	0.92	0.81	0.70	0.73

Table 11: Performances of the models on the `titleType` classification task.

Figures 8a shows the confusion matrix for Random Forest. Consistently good performances can be observed across the most represented classes. `video` and `tvSpecial` are the classes that cause the most problems, since they are often misclassified as the more frequent classes. The class `movie` attracts most of these misclassifications, likely due to similar durations of the products in these categories. Likely due to the same reason, `video` is also often misclassified as `short`.



(a) Confusion Matrix - Random Forest



(b) ROC Curve - Random Forest

Figure 8: Confusion matrix and ROC Curve for Random Forest on the `titleType` classification task.

The ROC curve shows high AUCs for all classes, with the lowest ones being for the aforementioned less represented classes, both achieving 0.96. This indicates that the model is very good at maintaining a low false positive rate for these, while correctly identifying the more represented classes.

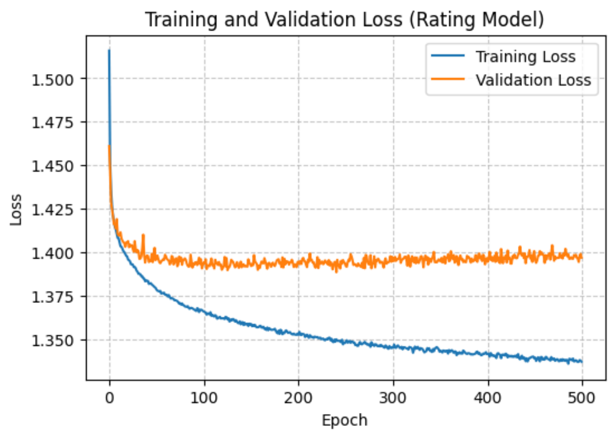
4.4 Neural Networks

For both classification tasks, a feedforward neural network was implemented.

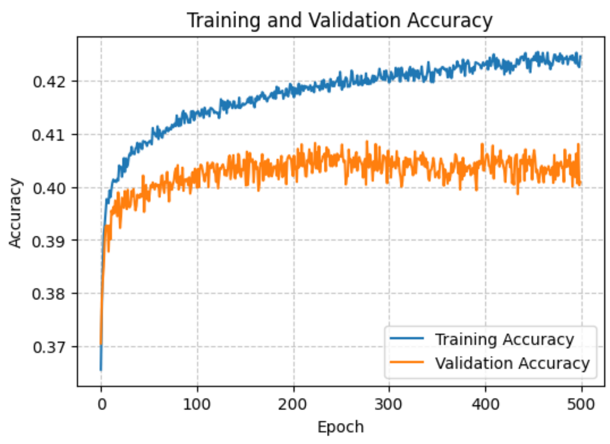
For the `averageRating` task, the architecture consists of an input layer, six hidden layers with ReLU activation, forming a diamond shape (increasing and then decreasing number of neurons), a dropout layer (rate 0.2) to prevent overfitting, and an output layer with softmax activation for multi-class classification.
The model was compiled with the Adam optimizer, categorical cross-entropy loss, and accuracy as the evaluation metric. No early stopping was used, in order to observe the full training dynamics. Training was set to last for 500 epochs, with a batch size of 64 and a validation split of 20%.
Figures 9a and 9b show the training and validation loss and accuracy over epochs. The training loss shows a steady decrease, while the validation loss stops improving after around 100 to 150 epochs. A similar trend is observed in the accuracy plot, although the end of the the rise can be found after around 200 epochs. Neither graph shows particular signs of overfitting, but the stagnation of the validation metrics suggests that the model could benefit from further regularization. Experiments were made with different dropout rates and configurations, as well as different training batch sizes and learning rates, in an attempt to improve generalizaiton, but this remained the best configuration found.

	Precision	Recall	F1-score
Macro avg	0.39	0.29	0.29
Weighted avg	0.39	0.40	0.36
Accuracy			0.40

Table 12: Classification performances for the neural network on the `averageRating` classification task.

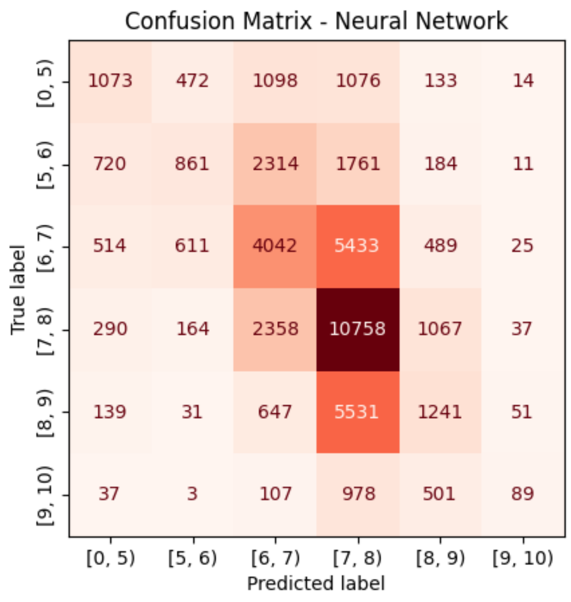


(a) Loss - Neural Network



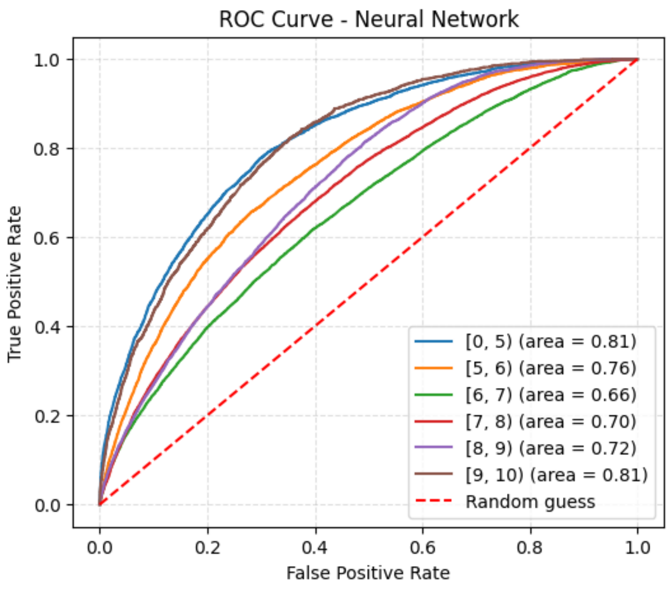
(b) Accuracy - Neural Network

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



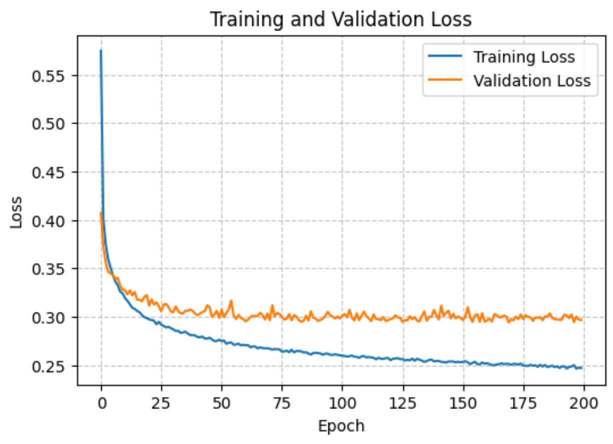
	[0, 5)	[5, 6)	[6, 7)	[7, 8)	[8, 9)	[9, 10)
[0, 5)	1073	472	1098	1076	133	14
[5, 6)	720	861	2314	1761	184	11
[6, 7)	514	611	4042	5433	489	25
[7, 8)	290	164	2358	10758	1067	37
[8, 9)	139	31	647	5531	1241	51
[9, 10)	37	3	107	978	501	89

(a) Confusion Matrix - Neural Network

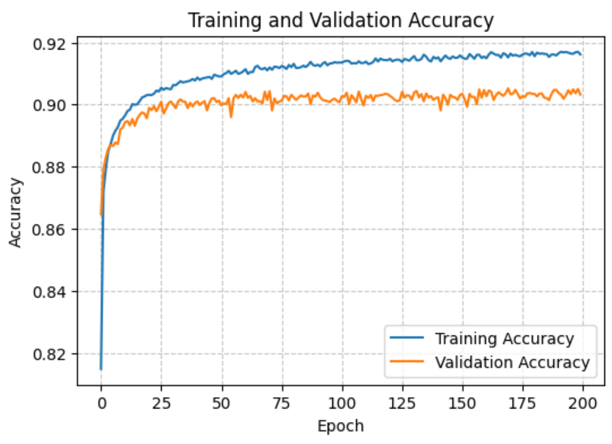


(b) ROC Curve - Neural Network

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

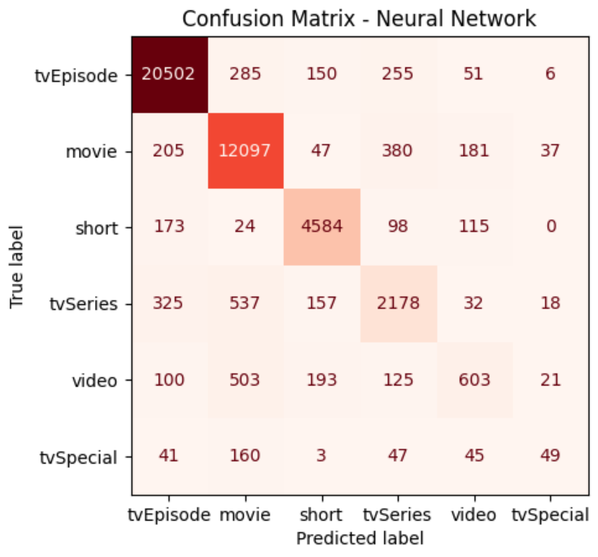


(a) Loss - Neural Network



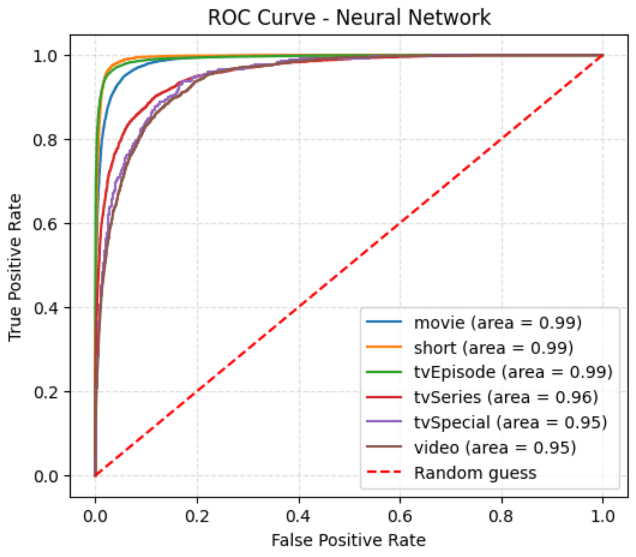
(b) Accuracy - Neural Network

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



True label \ Predicted label	tvEpisode	movie	short	tvSeries	video	tvSpecial
tvEpisode	20502	285	150	255	51	6
movie	205	12097	47	380	181	37
short	173	24	4584	98	115	0
tvSeries	325	537	157	2178	32	18
video	100	503	193	125	603	21
tvSpecial	41	160	3	47	45	49

(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.

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*.
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 $\mathtt{R}^2$ 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 13). 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 13: Performance of the Random Forest Regressor on the test set (full model vs reduced features).

Model	**MAE**	**MSE**	**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 13, 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.

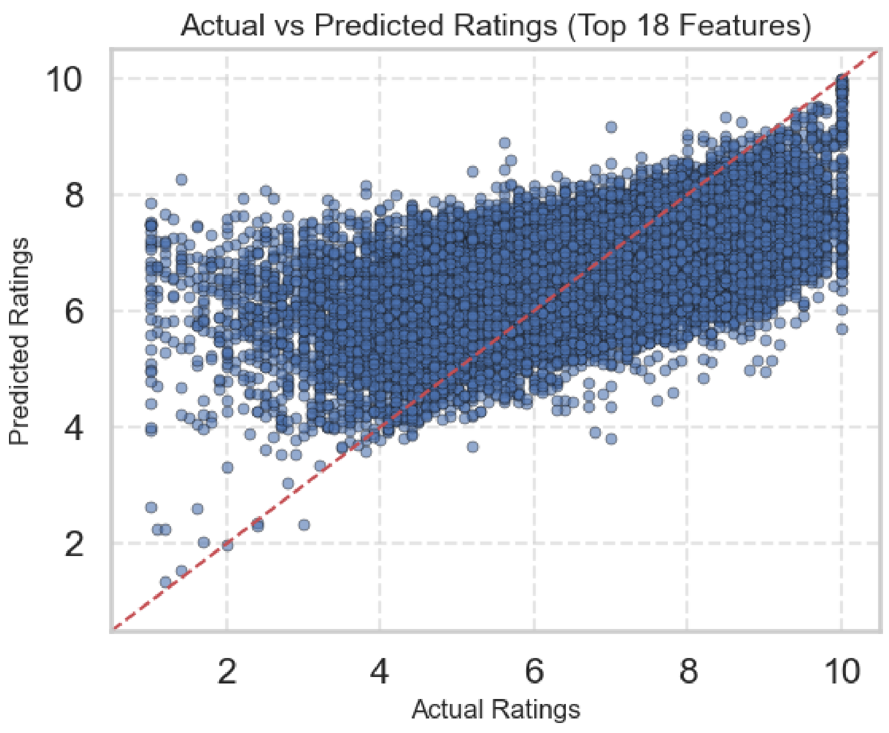


Figure 13: Actual vs Predicted `averageRating` for the Random Forest model trained on the top 18 features.

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.