UNIVERSITÀ DI PISA

Master Degree in
Data Science and Business Informatics

Data Mining 2 Report

Candidates:

Alessandro Falcetta (642129)

Riccardo Roselli (614720)

Academic Year 2024 - 2025

Contents

1 Preprocessing

This chapter describes the main preprocessing operations carried out on the dataset prior to analysis and modeling. These activities include the removal of irrelevant or redundant features, data cleaning, and the application of transformations to certain variables.

The dataset was first split using random sampling, with **80% allocated for training** and **20% for testing**. This split was performed before applying transformations and imputation in order to avoid introducing bias into the analysis results and to ensure proper generalization of the models.

Removed Features

Some variables in the dataset were removed during preprocessing because they were considered redundant, constant, or uninformative for the analysis. Specifically, variables such as `bestRating`, `worstRating`, and `isRatable` had constant or identical values across all observations and thus did not contribute to the variability of the dataset. Other features, such as `endYear` and `soundMixes`, had a high percentage of missing values (both over 60%), making them unreliable for analysis. Lastly, the variable `ratingCount` was eliminated as it duplicated the information contained in `numVotes`. Also `canHaveEpisodes` was removed as redundant.

Preprocessing of Categorical Features for Machine Learning

The preprocessing of the categorical features `countryOfOrigin`, `genres`, and `titleType` focused on transforming them into a format suitable for machine learning algorithms.

For `countryOfOrigin`, stringified lists (e.g., `['US']`) were deserialized into Python lists, and missing values (`\N`) were replaced with `pd.NA`. Country codes were standardized to resolve inconsistencies (e.g., `CSHH` $\rightarrow$ `CS`), and countries were mapped to their respective geographic regions (e.g., `US` $\rightarrow$ North America). These regions were then one-hot encoded into binary indicator variables representing regions such as `Asia`, `Europe`, and `North America`.

For `genres`, comma-separated or stringified genre lists (e.g., `Drama,Comedy`) were parsed into Python lists, and missing values were imputed based on the most frequent genre combinations within the same `titleType`, `startYear`, and `countryOfOrigin`. Genres occurring infrequently were consolidated into an `others` category. Subsequently, a multi-label encoding strategy was applied to represent the genres as binary columns (e.g., `Action`, `Drama`, `Comedy`).

For `titleType`, the categorical values were grouped into three higher-level categories: `movie`, `short`, and `Series`, based on common characteristics. A new feature, `groupedTitleType`, was created, and one-hot encoding was applied to generate binary columns for each category.

Transformations

Some features with highly skewed (positively skewed) distributions were transformed using the logarithmic transformation (**log1p**) to normalize the distribution, making it more suitable for classification and regression models. The criterion used was to apply a threshold to the skewness in the training set: distributions with skewness greater than 2 were transformed. To keep track of the transformations, the suffix "Log" was added to the transformed variables (e.g., `numVotes` becomes `numVotesLog`). Figure 1 shows the distribution of the variable `numVotes` plotted on a log-log scale, before any transformation. The trend appears approximately linear on this scale, suggesting the distribution might follow a power law. This type of distribution is frequently observed in real-world phenomena arising from complex systems and multiple interactions, such as popularity or content spread. A fit over a range of values was performed for qualitative verification (Figure 1). Furthermore, this is only one of the transformed variables,

so such a trend is not always observed, although all transformed variables are characterized by a very low mean close to the minimum value (all positive integers).

The presence of long tails and strong skewness, typical of this kind of data, makes the application of a logarithmic transformation appropriate to improve the effectiveness of predictive models, which are generally designed to handle normally distributed data.

Table 1 shows the skewness values before and after the application of the logarithmic transformation (**log1p**). The transformation significantly reduced skewness, although in some cases it still remained above the threshold used as the transformation criterion.

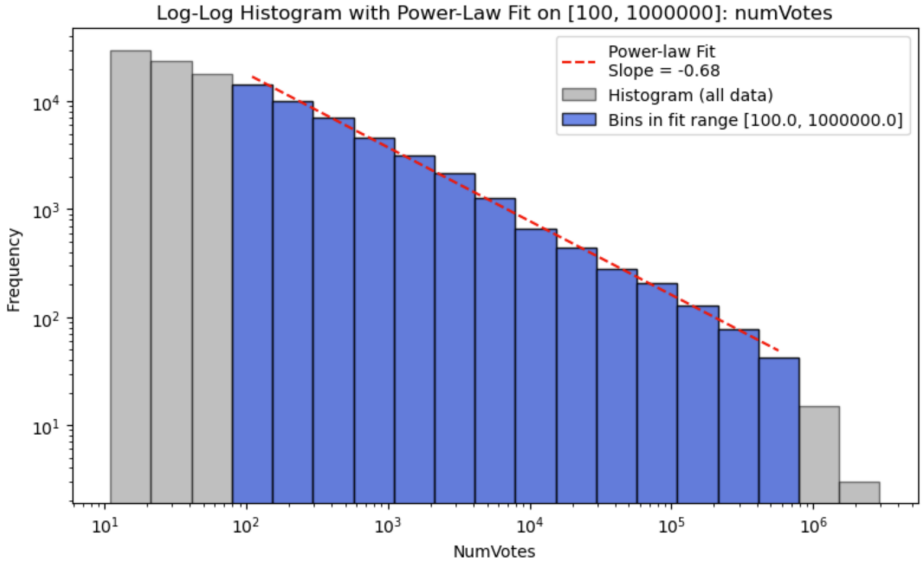


Figure 1: `numVotes` distribution in log-log scale with logarithmic binning.

Table 1: Skewness before and after the log1p transformation on test set.

Feature	Initial skewness	Post-log skewness
awardNominations...	63.04	4.71
awardWins	67.59	4.89
castNumber	56.30	-0.30
companiesNumber	20.97	0.75
criticReviewsTotal	17.77	2.88
directorsCredits	9.01	-0.15
externalLinks	13.13	1.52
numRegions	4.41	1.05
numVotes	67.29	1.42
quotesTotal	56.50	3.30
totalCredits	68.48	-0.34
totalImages	121.45	1.40
totalVideos	132.18	6.04
userReviewsTotal	73.01	2.41
writerCredits	3.95	0.31

Imputation of runtimeMinutes

After the previously described operations, the only feature still containing missing values was `runtimeMinutes`, with nearly the 30% of its entries missing. Considering it an important vari-

able, it was decided to impute it using a regressor trained on the available data. To do this, the records with missing values were set aside, and the remaining training set was used to train and validate KNN regressor using `runtimeMinutes` as the target. The features used for KNN were normalized and selected based on their importance as determined by a Decision Tree. To avoid introducing biases in later stages, features that would be used as targets in subsequent sections were excluded from the training process.

After performing a random search to estimate the best hyperparameters through cross-validation for both methods, minimizing the mean absolute error, **KNN model** was used for imputation with a **mean absolute error** of about **12 minutes**. The fact that the error is not very low suggests that the variable cannot be fully reconstructed from the others, proving that it retains non trivial information.

The model was also used to impute the missing values in the test set.

2 Outlier Detection

In this preliminary phase, an outlier detection analysis was conducted on the entire *dataframe* (not split into train/test), already subjected to *preprocessing*. The main objective was not directly related to later classification tasks but rather to compare various detection methods in terms of effectiveness and consistency. Only continuous numerical variables were considered, normalized within the interval $[0, 1]$.

2.1 Outlier Detection Methods

The following approaches were applied for outlier detection:

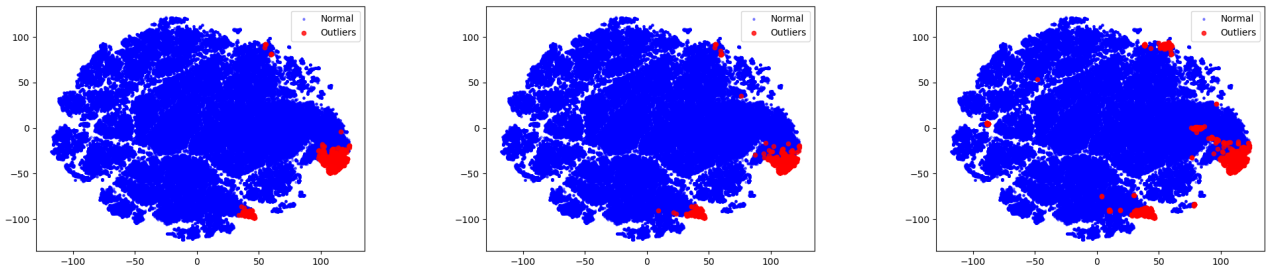
1. **Histogram-Based Outlier Score (HBOS)**: a naive method based on the histogram of variables, applied directly to all continuous features.
2. **Lower Mean Density-based Distance (LMDD) + k-means**: density estimation on centroids of a k-means partition ($k = 3000$) to reduce computational cost. The score of each centroid was equally assigned to the points belonging to its respective cluster.
3. **LMDD + split**: application of the LMDD method to 20 disjoint subsets of the dataset to reduce computational load, followed by result aggregation.
4. **1-Nearest Neighbor (1-NN)**: based on the distance to the nearest neighbor.
5. **100-Nearest Neighbors (100-NN)**: measures the average distance to the 100 nearest neighbors.
6. **Local Outlier Factor (LOF)**: applied on a PCA projection with 6 principal components, with `n_neighbors = 300`.
7. **DBSCAN**: clustering on 6 PCA components with `epsilon = 1.4` and `min_samples = 100`, optimized to identify about 1% outliers. Due to high computational cost, the method was applied to subsets (one-tenth of the dataset at a time). Overall, outliers were detected for 1.06% of the data.
8. **Isolation Forest**: a tree-based model, with `max_samples = 500` and `n_estimators = 1000`.

Except for DBSCAN, in all approaches, **the top 1%** of data points with the highest score were selected as outliers.

2.2 Dimensionality Reduction and Visualization

To reduce computational cost and focus the analysis on the most informative components, density and clustering-based methods (particularly LMDD, LOF, and DBSCAN) were applied not on the full feature space, but on a PCA projection with 6 principal components, explaining around 80% of the total variance. This choice provides a good balance between dimensionality reduction and preservation of relevant information.

Since the first three components cover only 60% of the variance, for visualization purposes, **t-SNE** was preferred, as it is a nonlinear technique that better highlights local differences in data distribution, revealing latent separations between regular points and anomalies even in two-dimensional space.



(a) HBOS (b) Isolation Forest (c) LMDD on k-means

Figure 2: t-SNE visualization of outliers according to three different approaches

2.3 Comparison Between Methods

To assess the degree of overlap among the sets of outliers identified by different approaches, a similarity matrix was computed using the Jaccard distance between outlier sets, defined as the ratio of shared outliers to the union size (for each method pair).

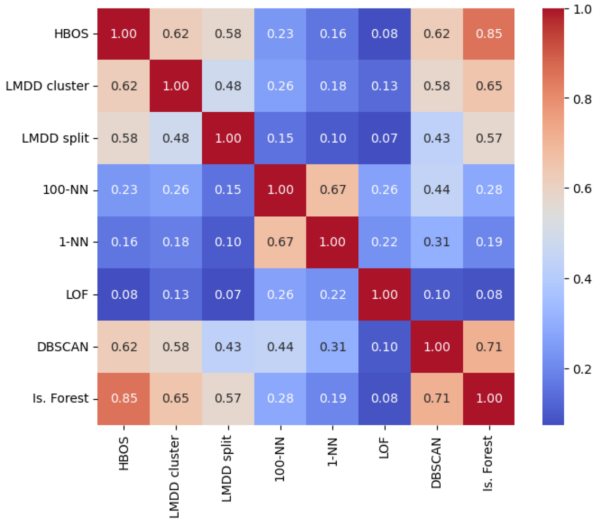


Figure 3: Similarity matrix between outlier detection methods.

Figure 3 reveals a good agreement between several methods from different families, as also seen in Figure 2. Methods relying directly on k-NN produced similar results even with significantly different values of k, but were less aligned with other methods.

Overall, the matrix highlights that the methods are not equivalent, and that the choice of approach can significantly affect the type of outliers identified, although some level of agreement always exists, suggesting that the 1% threshold might be too broad to isolate true anomalies. Many variables in the dataset exhibit skewed distributions, even after logarithmic transformations. In this context, **detected outliers** may not simply represent noisy anomalies but **potentially meaningful** or rare **observations** of interest.

Therefore, although their identification is important to understand the data structure, the removal of outliers may be inappropriate and could introduce bias, and the logarithmic transformation is sufficient to regularize the dataset. For this reason, the following analysis is conducted on the entire dataset.

3 Imbalanced Learning

This section addresses binary classification problems characterized by few records belonging to the positive class.

To maintain simple and easily interpretable models, we adopted **Decision Trees** with binary splits in both tasks, selecting the best hyperparameters through **cross-validation**. Given the strong class imbalance, the main evaluation metric is the **macro F1-score**, which improves recall on the positive class without completely sacrificing precision.

Various resampling techniques were also explored, and the best results are reported (in case of equal performance, the most efficient method is presented). Optimal hyperparameters were determined via Random Search, consistently applied across all techniques, with the only adjustment being the scaling of parameters sensitive to normalization on the training set. The search focused on the optimal combination of **max_depth**, **min_samples_split**, and the splitting **criterion** between entropy and Gini. The **minimum number of samples per leaf** was fixed at **3**, while **5** for oversampling methods, to reduce overfitting risk.

All continuous and boolean variables (including those defined during preprocessing via one-hot encoding) were used in these tasks.

3.1 Binary Rating

The first objective concerns rating prediction: the goal is to identify whether a title has a rating less than or equal to 3, which accounts for about 3% of the total.

Below are the best hyperparameters and the results obtained with the original training set and after applying resampling techniques. The values explored for `max_depth` ranged from 3 to 25, including the option of unlimited depth, while min_samples_split was varied between 10 and 200.

Sampling (train size)	Hyperparameters (depth,split,criterion)	F1-macro	Recall	Precision
None (104,529)	19, 50, entropy	0.52	0.04	0.13
ENN (96,571)	11, 20, gini	0.61	0.22	0.26
ADASYN (202,237)	16, 20, gini	0.53	0.35	0.09

Table 2: Comparison of sampling techniques and model performance (decision tree)

The best result was obtained using the **Edited Nearest Neighbours (ENN)** technique, which improved both performance and model simplicity. ENN works by removing training instances that disagree with the majority of their nearest neighbors, reducing noise and refining class boundaries. Analyzing feature importance, the most relevant variables are genres like animation, comedy, and adventure, although no particularly effective simple rules emerge. Overall, performance remains limited: precision is low and recall only moderate, even with oversampling techniques like **ADASYN**, which generates new synthetic minority class examples by focusing on the most difficult-to-classify observations. This highlights the inherent difficulty of classification tasks under strong class asymmetry and limited positive representation.

3.2 isAdult

The second objective is to determine whether a title is suitable for a minor audience, using the `isAdult` variable, which has a positive class ratio of about 2%.

Since the "adult" genre provides information nearly equivalent to the `isAdult` variable, genre features were excluded from the model to make the problem less trivial. The assumption

is that such information may not be available at imputation time, and that genre is not essential to reconstruct it.

For this variable, some particularly informative features were identified by the decision tree, enabling the creation of simple models with high F1-score, even without resampling techniques. For this reason, during the random search, tree depth was limited to 20 without compromising performance. Below are the best results obtained, and Figure 4 shows an example of a shallow decision tree among the top performers.

Sampling (train size)	Hyperparameters (depth,split,criterion)	F1-macro	Recall	Precision
None (104,529)	13, 50, entropy	0.90	0.80	0.79
Tomek links (104,393)	13, 50, entropy	0.91	0.82	0.84
SMOTE (205,226)	14, 100, entropy	0.85	0.89	0.59

Table 3: Comparison of sampling techniques and model performance (decision tree)

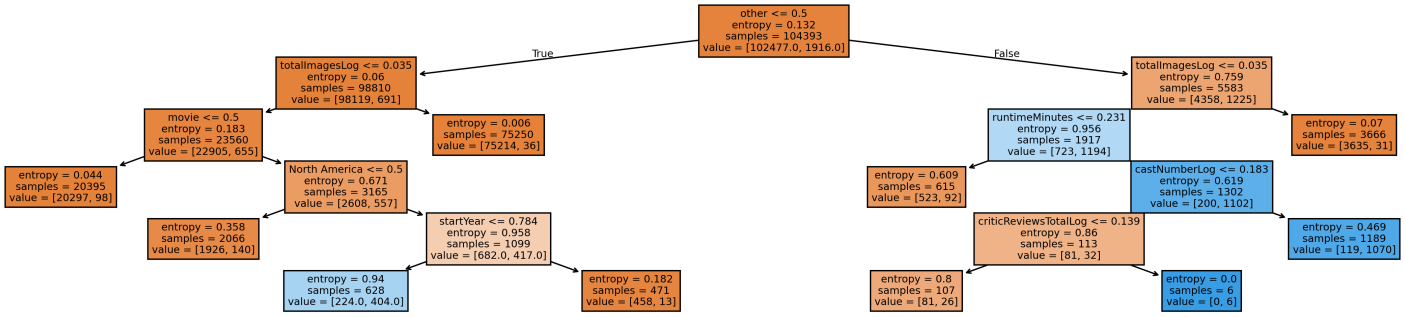


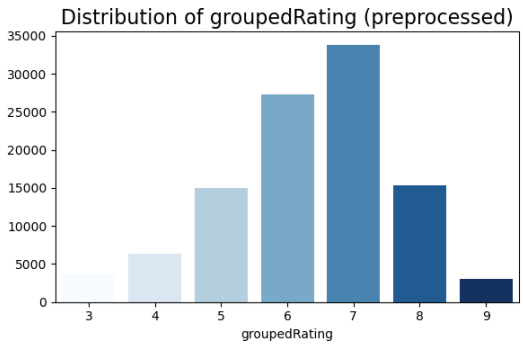
Figure 4: Pruned decision tree (Tomek links)

The best results were achieved with **Tomek links**, which removed only a small portion of the majority class by identifying pairs of closely located examples from opposite classes and deleting the one from the dominant class. This cleaned the class boundaries and slightly improved overall performance using the same hyperparameters as the original model. Oversampling via **SMOTE**, on the other hand, introduced a bias that significantly increased recall at the expense of precision, by generating new synthetic minority class examples through interpolation among neighbors in the feature space, while maintaining similar model complexity.

Using the **pruned tree** in the figure, reduced to a depth of 5, still yields a **recall of 0.77** and a **precision of 0.74**. Such simple relationships are easily interpretable and could potentially be used to manually identify suspicious cases or classification errors, even without computational support.

4 Advanced Classification

Advanced classification explores a range of machine learning algorithms, from simple linear models to complex ensembles, to uncover patterns that basic methods may miss. The objective of this chapter is to assess how effectively these models predict a newly defined target variable, **groupedRating**, which maps IMDb's original **1–10** scores into seven ordinal classes: **[3, 4, 5, 6, 7, 8, 9]**. This grouping was chosen to reduce granularity in the tails of the distribution, where data is sparse, while preserving the original classes in the central range.



Despite this transformation, the class distribution remains imbalanced, with a majority of titles clustered around ratings **6** and **7**, while extreme classes like **3** and **9** represent only a small fraction of the data. This imbalance poses a challenge for most classifiers, particularly in recognizing and correctly classifying the rarest classes. Therefore, our evaluation goes beyond overall accuracy to also consider how well each model identifies under-represented ratings.

To ensure a fair and consistent comparison, all classifiers were implemented using a standardized pipeline that included **model validation** and **training**, **prediction**, **evaluation** using a classification report and confusion matrix, and multiclass ROC-AUC analysis.

Hyperparameter tuning was performed using a validation set consisting of 20% of the training data, ensuring that model selection decisions were based on unseen data and not biased by the test set. For each classifier, we discuss the most influential hyperparameters, explaining why the selected configurations performed best in this context. Any parameters not explicitly addressed were left at their default values, as defined in the official documentation.

4.1 Logistic Regression

Logistic Regression served as our starting point, offering a transparent, fast benchmark on the groupedRating task. Using the default **regularization** setting **C = 1** and the **lbfgs solver**, and after experimenting with both stronger and weaker regularization, the model achieved **0.35 accuracy** overall but revealed clear limitations when examined class by class. The classification report showed strong precision and recall on the well-populated middle classes (particularly ratings 6 and 7), yet **0.00 recall** on the rare edge classes 3 and 9. The confusion matrix confirmed that most titles rated 3 were misclassified to 5 or 6, and those actually rated 9 fell back into 8, illustrating the model's tendency to favor majority classes under linear assumptions. Because logistic regression lacks a non-linear decision boundary, it struggled to classify correctly the "corners" of the rating distribution. In this context, its key hyperparameter (the regularization strength C) remained at its default, since increasing model complexity through weaker regularization did not meaningfully improve detection of under-represented classes and only demonstrated overfitting on the abundant mid-range data.

4.2 Support Vector Machines

Support Vector Machines seek decision boundaries that best separate classes: **Linear SVC** does so with a straight hyperplane, while the full **SVC** variant can leverage non-linear kernels (**RBF**, **polynomial**) to warp the feature space. Using default **C = 1** and **l2** regularization (and **gamma = 'scale'** for non-linear kernels), Linear SVC achieved **0.35 accuracy** and a **weighted F1 of 0.32**, while SVC edged up only to **0.36 accuracy**. In both cases, precision and recall remained strong for well-represented mid-range ratings 6 and 7, but recall on the rare classes 3 and 9 lingered at **0.01**, with most true 3s mapped to 5 or 6 and true 9s to 8. Tuning C tighter or looser failed to improve these corner cases, and adding RBF or polynomial kernels only introduced marginal gains at a dramatic increase in training time. These results underscore that, under severe class imbalance and overlapping feature distributions, even flexible kernels cannot meaningfully outperform simpler linear boundaries, making SVMs an impractical choice for this particular multi-class prediction task.

4.3 Neural Network

The **Multilayer Perceptron (MLPClassifier)** captures complex non-linear relationships by stacking fully connected layers of neurons trained via backpropagation using the **Adam solver** by default.

In our final setup, we adopted a tapered three-layer architecture of **(128, 64, 32) units** with **ReLU activations**, paired with **early stopping**. The ReLU activation offers fast convergence without the vanishing-gradient issues of tanh or logistic, while early stopping proved especially important: by halting training once validation performance stopped improving, it not only prevented overfitting on the dominant mid-range classes 6 and 7 and encouraged the model to discover the rare classes, but also significantly reduced training time by several minutes.

This configuration achieved an overall **accuracy** of **0.38** and a **weighted F1-score** of **0.37**. Notably, it was the first model to register non-zero **recall** for the rare edge classes: **0.10** for class 3 and **0.02** for class 9, with **F1-scores** of **0.16** and **0.04**, respectively. The model also achieved an **F1-score** of **0.51** for class 7, correctly predicting over 8000 instances. The confusion matrix, though still imperfect for class 9, showed a more balanced error distribution across all classes compared to earlier models. Extreme ratings were less frequently absorbed into central bins like 6 or 7, reflecting the network's improved sensitivity and ability to learn richer, non-linear feature combinations across the entire rating spectrum.

We also experimented with deeper "diamond-shaped" architectures like (32, 64, 128, 64, 32) or up to five layers such as (256, 128, 64, 32, 16), and tested alternative solvers like lbfgs or sgd paired with constant and adaptive learning rates. However, these configurations either converged much more slowly, overfitted the mid-range classes without improving recall on the under-represented ones, or even degraded the overall model's performance. These experiments confirmed that moderate depth with the Adam solver and early stopping together provide the most effective trade-off between accuracy, generalization, and computational efficiency in this imbalanced multi-class task.

4.4 Bagging Classifier

Bagging Classifier is an ensemble method that builds multiple models on different bootstrap samples of the data and aggregates their predictions to reduce variance. By default, each estimator sees all features at every split and is trained on a random subset of rows, yielding a diverse set of learners whose majority vote improves stability and accuracy.

In the final BaggingClassifier, we configured **300 estimators**, each built on a **DecisionTreeClassifier** with **max_features** equal to **the square root of the total number of features**. This moves feature subsampling into the base estimator itself, reproducing Random Forest's per-split feature randomness, ensuring that every tree split considers only a random subset of features, rather than the full set. With this setup, the BaggingClassifier achieved **0.45 accuracy** and a **0.42 weighted F1-score**. But since this behavior is essentially identical to **Random Forest**'s one, a deeper description is provided in the Random Forest section below.

We also tried using BaggingClassifier's own max_features parameter (which samples features once per tree) and found quite similar results which instead lacked the extra randomness you get at each split. Additionally, we experimented with LinearSVCs as base estimators, but the computational cost increased without meaningful accuracy gains.

4.5 Random Forest

Random Forest is also a bagging-based ensemble, but it introduces a second layer of randomness by sampling a subset of features at every split in each tree, whereas BaggingClassifier

samples features once per tree. This per-split feature subsampling further decorrelates the trees, enhancing robustness against overfitting and enabling the model to capture diverse patterns across both common and rare classes.

In our final Random Forest, the best parameters were still the same found in BaggingClassifier: **300 decision trees**, measured splits by **entropy** to maximize information gain (rather than the default Gini impurity), and allowed **unlimited depth** so that each tree could fully partition the feature space and capture subtle interactions. We kept the default **minimum split size** of **2** samples and **leaf size** of **1** sample, ensuring that trees could grow until pure leaves were reached. Parallelizing across all CPU cores kept training time practical.

This configuration delivered as expected the same results as BaggingClassifier with **0.45 accuracy** and a **0.42 weighted F1-score**, with **0.14 recall** on both rare classes 3 and 9, with an **F1-score** of **0.23** on those extremes for the first time. The largest class, rating 7, correctly predicted over 10000 times, achieved **F1** of **0.57**, demonstrating how Random Forest's combination of deep, diverse trees of entropy-based splits can both generalize on abundant mid-range ratings and isolate the rare classes.

Alternative configurations reinforced our setup's effectiveness. Reducing the ensemble to 100 trees led to higher variance across predictions: despite growing to full depth, the smaller ensemble failed to adequately average out individual tree noise, leading to poor predictions. Conversely, increasing the ensemble to 1000 trees but restricting each tree's depth to 3 or 5 produced forests that were too shallow resulting in underfitting, as small trees could not model the complexity of the IMDb feature space. Neither extreme of few full-grown trees nor many shallow ones matched the performance of our 300 trees with unlimited depth setup.

4.6 AdaBoost Classifier

Boosting is an ensemble technique that builds learners sequentially, with each new model focusing on the errors of its predecessors. Unlike bagging, which trains learners independently on random subsets to reduce variance, **AdaBoost** reduces bias by iteratively reweighting misclassified samples so that subsequent learners pay more attention to the "hard" cases.

Among all trials with various base estimators, simply comparing the metrics revealed that the best performance was achieved by configuring AdaBoost with **10 boosting rounds** and a **learning rate of 1.3**, using a **Random Forest** of **100 trees** with **entropy splits** as the base learner. The 10 rounds parameter controls how many times the algorithm reweights and retrains a new forest; too few rounds limit bias reduction, while too many risk overfitting. We found that 10 provided steady gains on rare classes without runaway complexity. The learning rate of 1.3 determines how much each new forest's vote counts: higher values accelerate learning but can overfit, so 1.3 struck a good balance, amplifying focus on misclassified extremes without destabilizing the ensemble. These choices leveraged Random Forest's strong baseline performance and relatively fast training: each boosting iteration refines the ensemble by emphasizing examples the forest struggled with.

This setup delivered **0.45 accuracy** and a **0.43 weighted F1-score**, with **0.15 recall** on both class 3 and class 9 and an **F1-score** of **0.24** on those extremes, slightly surpassing the pure Random Forest's performance. Class 7 remained at **F1** of **0.57**, correctly identifying over 10 000 instances. The confusion matrix (figure 5) showed a clear uptick in true positives for the edge ratings, while sustaining high precision on the mid-range bins.

We also experimented with heavier boosting configurations: 10 rounds of 500-tree forests, which became very slow and ultimately overfit, and 100 rounds of 10-tree forests, which required prohibitive runtimes as boosting compounds computational cost across rounds. Using simple decision tree stumps as the base estimator still delivered respectable results but could not match the combined strength of boosted forests. Even if embedding one ensemble within another feels unconventional, it delivered the best metrics; however, the computational over-

head was only partly justified by the slight accuracy gains, suggesting diminishing returns beyond this configuration.

4.7 Gradient Boosting and Histogram-Based Gradient Boosting

Gradient boosting is a sequential ensemble method that builds trees to correct the residual errors of preceding models, effectively minimizing a chosen loss via gradient descent. While the classic **GradientBoostingClassifier** uses exact split-finding on continuous features, its histogram-based counterpart (**HistGradientBoostingClassifier**) first buckets feature values into discrete bins, vastly speeding up split selection on large datasets without sacrificing accuracy. We trained both implementations but chose to focus on HistGradientBoostingClassifier because it delivered better results in a fraction of the time.

For our final configuration, we used **1 000 boosting rounds**, a **learning rate of 0.1**, and **no depth limit**. Increasing the number of iterations from the default 100 allowed the model to steadily refine its fit under a modest learning rate, reducing bias on the multi-class task without overfitting. Leaving max_depth unconstrained let each tree capture complex interactions in the IMDb features.

This setup achieved **0.42 accuracy** and a **0.38 weighted F1-score**, with **F1** of **0.20** on class 3 and **F1** of **0.17** on class 9, and still correctly predicting over 10 000 instances of class 7.

We also evaluated a broad range of boosting rounds, from as few as 100 up to 3 000 iterations, taking advantage of HistGradientBoosting's rapid training. While pushing beyond 1 000 rounds yielded only marginal improvements in F1 for the rare classes, dropping below 500 rounds began to underfit the data. This confirmed that 1 000 boosting rounds hit the optimal compromise between performance and efficiency in our setup.

4.8 XGBoost

XGBoost (**eXtreme Gradient Boosting**) is an optimized gradient-boosting framework that uses both first and second order gradient information plus regularization to enhance speed and generalization.

In our final model, we used **300 estimators** to balance variance reduction and training time, a **learning rate** of **0.3** (the default, which ensures rapid yet stable convergence), and **unlimited depth** so each tree could fully explore complex feature interactions. We set **gamma = 0.0** (no minimum split gain) and **L2 regularization (lambda = 1)** to control overfitting, while **tree_method='auto'** enabled the very fast histogram-based split algorithm. The combination of **objective='multi:softmax'** and **eval_metric='mlogloss'** directly optimizes and evaluates the multi-class log-loss for our seven rating classes prediction.

This configuration achieved **0.41 accuracy** and a **0.39 weighted F1-score**, with **F1** of **0.22** on class 3 and **F1** of **0.15** on class 9. Class 7 fell below 10 000 correct predictions, resulting in a lower F1 than previous models.

We also tried lower learning rates like 0.1 or 0.05 and capped tree depths to 6 or 8, but these settings either slowed convergence without improving edge-class recall or produced underfitted trees unable to capture the IMDb dataset's complexity.

4.9 LightGBM

LightGBM is a gradient boosting framework that grows trees leaf-wise rather than level-wise, allowing for deeper and more precise splits with improved accuracy and lower memory usage. It also leverages histogram-based binning, making it exceptionally fast and efficient, especially on large datasets.

For our final LightGBM model, we used **300 estimators**, a **learning rate of 0.1**, and **unlimited depth**, allowing each tree to capture the full complexity of the IMDb features. We set **boosting='gbdt'** for traditional gradient-boosted decision trees, **objective='multiclass'** with **metric='multi_logloss'** to directly optimize the seven classes prediction, and **is_unbalance=True** to automatically adjust for our skewed class frequencies. These parameters provided a solid balance of learning capacity, bias reduction, and built-in handling of class imbalance without extensive additional tuning.

This setup achieved **0.42 accuracy** and a **0.38 weighted F1-score**, with **F1** of **0.19** for both class 3 and class 9. Notably, it was the only model to correctly predict over **11 000 instances** of class 7 while maintaining strong balance across neighboring ratings.

4.10 Advanced Classification Conclusion

In conclusion, our advanced classification experiments reveal that ensemble methods combining deep, non-linear partitioning with mechanisms to emphasize difficult examples excel at handling the IMDb rating prediction task. In the table we focus on the overall performance metrics and on the F1-scores of classes 3 and 9 as the most under represented classes to assess each model's ability to recover rare ratings.

Model	Accuracy	Weighted F1	F1 (3)	F1 (9)
AdaBoostClassifier	0.45	0.43	0.23	0.24
RandomForestClassifier	0.45	0.42	0.22	0.23
BaggingClassifier	0.45	0.42	0.22	0.21
HistGradientBoosting	0.42	0.38	0.20	0.17
LGBMClassifier	0.42	0.38	0.19	0.18
XGBClassifier	0.41	0.39	0.22	0.15
CatBoostClassifier	0.41	0.38	0.17	0.11
GradientBoosting	0.40	0.34	0.11	0.08
MLPClassifier	0.38	0.37	0.16	0.04
Linear SVC	0.35	0.32	0.02	0.00
Logistic Regression	0.35	0.32	0.01	0.00

AdaBoost, built with Random Forest as its base estimator, delivered the top accuracy and F1-scores due to its sequential bias correction: by upweighting misclassified titles at each iteration, it forced the ensemble to focus on rare edge cases without compromising accuracy on the well represented mid-range ratings. Importantly, layering boosting atop bagging introduces significant complexity and computational overhead for only marginal gains; creating a "Boosted Random Forest" ensemble is neither generally advisable nor cost-effective despite its slight edge in our experiments.

Random Forest, with its entropy-based splits and unlimited depth, followed immediately. Its true strength lies in combining bootstrap aggregation and per-split feature subsampling to uncover both common and rare patterns efficiently without overfitting.

Gradient boosting variants, including **HistGradientBoosting**, **LightGBM**, **XGBoost**, and **CatBoost**, offered efficient runtime and solid performance producing consistent overall metrics, but their log-loss objectives and regularization strategies were less successful in lifting extreme classes, with little variance across implementations as shown in the table.

The **MLPClassifier** was the first model to register non-zero recall for both classes 3 and 9, confirming that neural architectures can extract subtle, non-linear signals. However, without dynamic reweighting or targeted error correction, it could not match the predictive power of tree-based ensembles.

Finally, **SVC variants** and **linear models** consistently underperformed. Their rigid decision boundaries were unable to capture the nuanced structure of the IMDb data, especially under severe class imbalance. Sparse classes remained inseparable from the dense mid-range clusters.

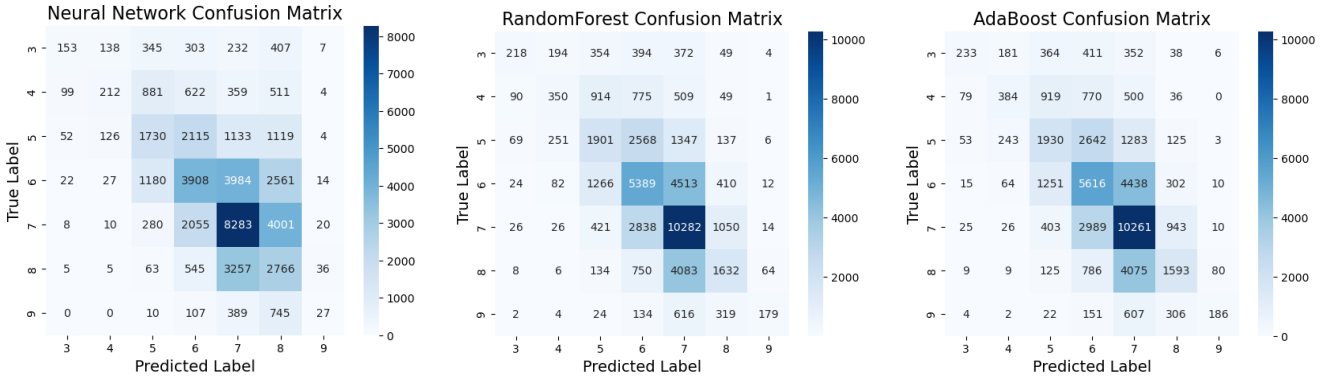


Figure 5: Confusion Matrices

To complement performance metrics, Figure 5 compares **confusion matrices** for the top three performing model families (Neural Network, Random Forest, and AdaBoost), highlighting how ensemble-based methods distribute errors more evenly, especially for edge classes.

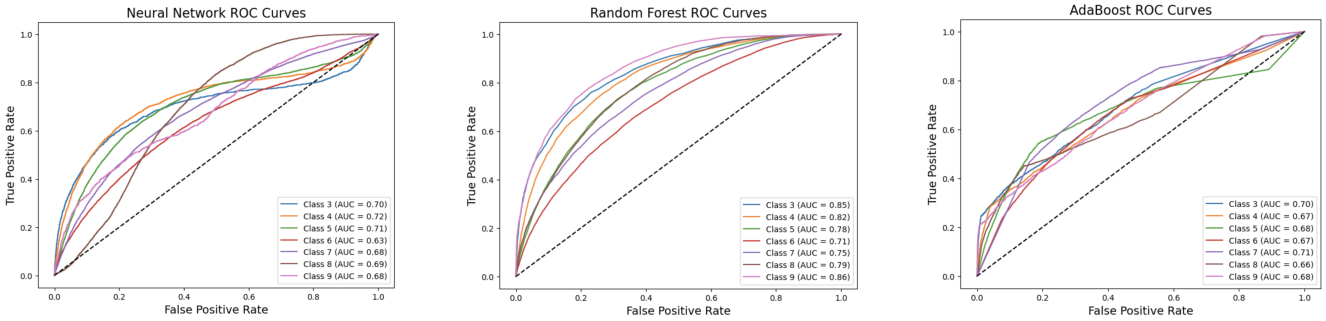


Figure 6: ROC Curves

Figure 6 overlays their **multiclass ROC curves**, providing deeper insight into probability calibration. Although AdaBoost achieved the highest F1 and accuracy at the 0.5 threshold, its boosted votes concentrate on that single decision point, making it sharp at classification time but poorly calibrated across thresholds. This leads to less well-spread probability estimates and flatter ROC curves. In contrast, Random Forest uses soft voting across hundreds of deep trees and produces smoother, better-calibrated scores that rank true positives above false positives more consistently across all thresholds, resulting in higher AUC despite slightly lower final accuracy. This underscores that while AdaBoost excels at the final prediction boundary, Random Forest offers superior ranking quality.

Overall, methods that both diversify decision boundaries and explicitly correct misclassified records are the most effective strategies; yet, taking computation versus gain into account, **Random Forest** emerges as the best general-purpose solution for this imbalanced, multi-class rating prediction task.

5 Explainable AI

Explainable AI (XAI) refers to a suite of techniques and tools designed to dive deep in the "black-box" of complex machine learning models, revealing why they make the predictions they do. In our IMDb rating prediction task, high accuracy alone is not enough: we also need

to understand which features drive decisions, how confident the model is in borderline cases, and what minimal changes could steer a misclassification back on track.

Building on top of our previously discussed advanced classification experiments, we now turn to applying XAI techniques (**SHAP**, **LIME**, **LORE** and **Counterfactual Explainer**) on three standout black-box models drawn from distinct algorithmic families: **Neural Network**, **Random Forest**, and **AdaBoost**. We selected these because they delivered the strongest predictive performance, while also embodying fundamentally different learning paradigms.

To further deepen our understanding of how each model handles the most challenging regions of the rating spectrum, we shift our focus to a specific subset of severely misclassified records. Rather than analyzing common and often explainable slips between neighboring classes, such as predicting a 7 instead of an 8 or 4 instead of 3, we deliberately target the most extreme edge-case errors, where a **true class 3** is **misclassified as class 9**, or vice versa. By applying LIME, LORE, and Counterfactual Explainers to these dramatic **3** $\leftrightarrow$ **9 misclassifications**, we aim to uncover what pushes the model so far off track. To operationalize this focus, we examined the misclassified records and specifically searched for cases that were misclassified across different model families: by scanning all **true-3** $\rightarrow$ **pred-9** errors, we isolated three shared records that both **Random Forest** and **AdaBoost** misclassified. These "shared misclassification targets" expose a structural blind spot that neither a bagging-based model nor a boosting-based model could overcome.

Below, we display each film's original metadata (before preprocessing) and emphasize the features that SHAP analyses as most influential. These snapshots will ground our subsequent local XAI deep dives with LIME, LORE, and Counterfactuals.

Table 4: Sample Subset of Media Titles with Metadata

Indexes	rating	startYear	runtimeMin.	numVotes	totalCredits	titleType	genres
16293	(1, 2]	2021	34	13	7	tvEpisode	Action, Adv., Comedy
18528	(1, 2]	1993	60	7	3	video	Adult
24362	(0, 1]	2018	120	16	3	tvSpecial	Talk-Show

5.1 SHAP

SHAP (**SHapley Additive exPlanations**) is a unified framework for assigning each feature an importance value by approximating Shapley values. It treats the model as a black box and estimates how much each feature contributes on average to pushing the prediction away from a baseline.

In our global analysis we employ **KernelExplainer** on each of the three models using identical settings to ensure a fair model-agnostic comparison. It only requires access to the model's predict_proba function, making it applicable to any learner. To keep this study tractable, we first distilled our full training set into **1000 k-means centroids** with the provided **shap.kmeans** method as the background distribution, then drew a stratified sample of **50 test records per class** (350 total) so that rare ratings 3 and 9 remain well represented and equally spread across classes without incurring prohibitive runtime.

The resulting global SHAP bar plots reveal a striking consensus: `runtimeMinutes` and the binary `Series` flag are the most influential features across all three models, followed closely by `movie`, `startYear` and `numVotesLog`. Beyond those five, which appeared consistent across all models, features like `totalCreditsLog` and `numRegionsLog` appear in slightly different orderings but never challenge the top tier. Even Random Forest's own **feature_importances_** method mirrors this ranking exactly, just without the `Series` and `movie` features, validating that SHAP faithfully recovers each model's native importance. The remarkable alignment of fea-

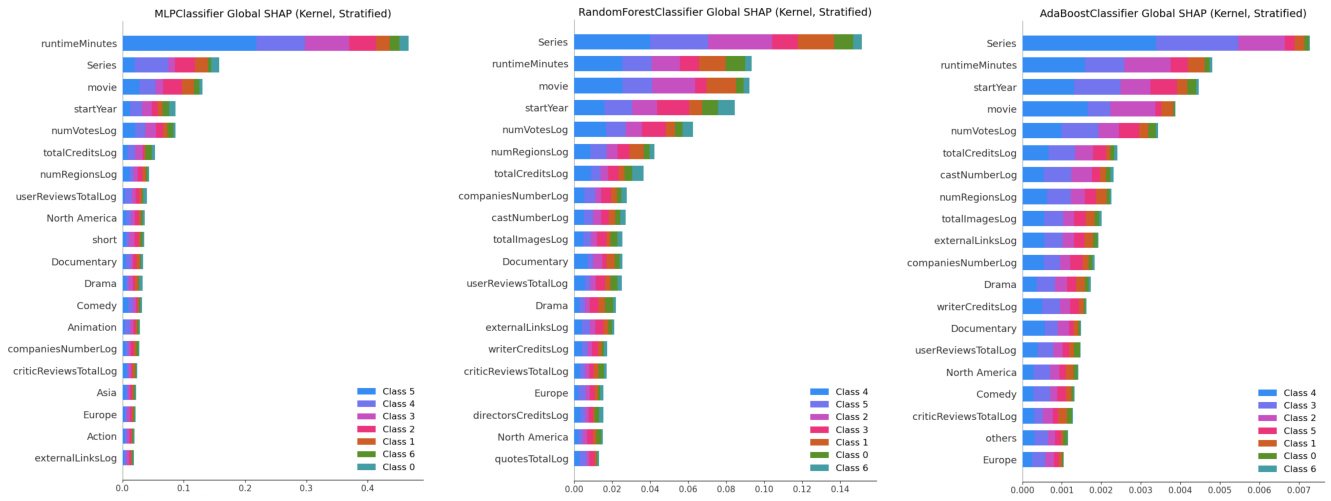


Figure 7: SHAP

ture rankings, despite using three distinct algorithms, underscores that these attributes carry the bulk of predictive signal for our seven-class groupedRating task.

5.2 LIME

LIME (**Local Interpretable Model-agnostic Explanations**) approximates any classifier's decision boundary in the neighborhood of a single instance by fitting a simple, interpretable surrogate model. Essentially, it perturbs an input around its observed values, observes the black-box's predict_proba outputs, and then reveals which features push the prediction toward one class versus another.

Building on our shared misclassification targets, we applied LIME to the three "extreme corner" indices (16293, 18528, 24362) presented before, focusing specifically on the dramatic **true = 3** $\rightarrow$ **pred = 9** flips in both Random Forest and AdaBoost. For each instance, the LIME report presents: the full prediction probability distribution across all seven classes; a "Why 9?" panel showing features that positively or negatively contribute to pushing the local surrogate toward class 9 and a "Why not 3?" panel contrasting those same features against the true class; and a concise table of the top weighted features with their actual values.

Across all examined records, index **16293** yielded no substantive insights: its tiny probability mass and flat feature weights left the surrogate model essentially undecided.

Index **24362**, by contrast, showed overwhelming confidence in the wrong label: for Random Forest the class 9 probability reached **0.36** (with all other classes well below), and AdaBoost displayed a similar distribution. These LIME plots confirm that both ensembles "knew exactly what they were doing", incorrectly.

The most revealing case was index **18528**. As shown in figure 8, both models placed class 9 at the top, but class 3 still appeared in the runner-up slot, with AdaBoost in particular nearly splitting its probability mass between the two. The "Why 9?" and "Why not 3?" charts for this record show only modest feature contributions: none of the usual top-tier drivers from SHAP (`runtimeMinutes`, `Series`, `startYear`) dominate, and instead lower ranked attributes such as `totalCreditsLog`, `isAdult`, and `castNumberLog` register small but opposing pushes. These explanations underscore that 18528 has prediction probabilities close to the decision boundary for both models, making it the prime candidate for the deeper LORE rule-extraction and counterfactual analyses that follow.

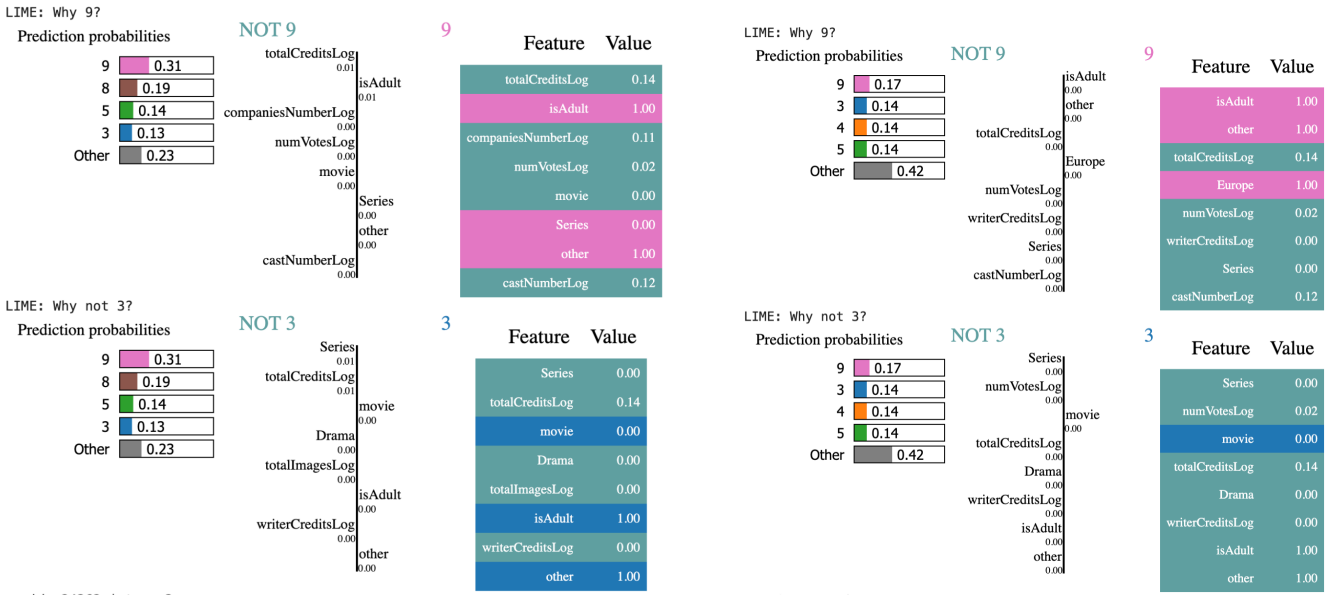


Figure 8: LIME for index 18528 (Random Forest and AdaBoost)

5.3 LORE

LORE (**LOcal Rule-based Explanations**) uncovers the decision logic of a black-box classifier by generating simple, human-readable "IF–THEN" rules that govern a single prediction (local rule) and minimal alternative rules that would flip the decision (counterfactual rules). Despite its computational cost, LORE offers nice insights into exactly which feature thresholds drive those extreme misclassification.

To explore the fragile **true = 3** $\rightarrow$ **pred = 9** boundary at index 18528, we fit LORE on our models using a small, randomized neighborhood with **size=100** due to its high computational time and a few counterfactual generations **ngen=10** for tractability. The tables below distill the core findings.

LORE Results for AdaBoostClassifier:

Class	Rule Conditions
Why 9?	Series $\leq$ 0.50, totalCreditsLog $\leq$ 0.15, castNumberLog $\leq$ 0.13, totalImagesLog $\leq$ 0.03, companiesNumberLog $\leq$ 0.14, externalLinksLog $\leq$ 0.05, other $>$ 0.50, userReviewsTotalLog $\leq$ 0.04, writerCreditsLog $\leq$ 0.14, numVotesLog $\leq$ 0.12
Would become 6 if	Series $\leq$ 0.50, totalCreditsLog $\leq$ 0.15, castNumberLog $\leq$ 0.13, totalImagesLog $\leq$ 0.03, companiesNumberLog $\leq$ 0.14, externalLinksLog $>$ 0.05
Would become 7 if	Series $\leq$ 0.50, totalCreditsLog $\leq$ 0.15, castNumberLog $\leq$ 0.13, totalImagesLog $\leq$ 0.03, companiesNumberLog $\leq$ 0.14, externalLinksLog $\leq$ 0.05, other $>$ 0.50, userReviewsTotalLog $\leq$ 0.04, writerCreditsLog $\leq$ 0.14, numVotesLog $>$ 0.12

LORE Results for RandomForestClassifier:

Class	Rule Conditions
Why 9?	Series $\leq$ 0.50, numVotesLog $\leq$ 0.03, castNumberLog $\leq$ 0.13, totalImagesLog $\leq$ 0.03, companiesNumberLog $\leq$ 0.14, other $>$ 0.50, startYear $>$ 0.76, writerCreditsLog $\leq$ 0.14

Would become 5 if	Series ≤ 0.50, numVotesLog ≤ 0.03, castNumberLog > 0.25, startYear ≤ 0.79, directorsCreditsLog ≤ 0.32, quotesTotalLog ≤ 0.19, movie ≤ 0.50, awardWinsLog ≤ 0.06, Australia/Oceania ≤ 0.50, userReviewsTotalLog ≤ 0.28

These rule sets confirm that both models hinge on tightly clustered thresholds of low-level engagement features (vote counts, cast size, credits, and title type flags), rather than the dominant predictors identified previously by SHAP. The counterfactuals reveal minimal tweaks that would steer the prediction back from 9 toward adjacent ratings. This concise, symbolic view sets the stage for our final counterfactual deep dive with the FAT-Forensics Counterfactual Explainer.

5.4 Counterfactual Explainer

The **FAT-Forensics Counterfactual Explainer** searches the feature space for the smallest numeric adjustments that would change a model's decision to a specific target class.

Unlike LORE, which generates counterfactual rules across adjacent classes and in our case failed to surface class 3 directly, here we explicitly forced the explainer to target the true class 3 for our target record. Given the heavy computational demands of this search, we configured the explainer with a moderate **step_size** of **0.2** and capped the number of simultaneous feature changes at **3** for the **max_counterfactual_length** to avoid an explosion of candidate solutions.

Despite this careful tuning, both Random Forest and AdaBoost yielded the same single-feature recommendation: **castNumberLog: 0.118** $\rightarrow$ **0.200**.

In other words, a modest increase of cast size is sufficient to reclassify the instance from 9 back to 3. While this insight is quite actionable, it also highlights the limits of purely numeric counterfactuals in capturing the full nuance of these extreme misclassifications.

With SHAP, LIME, LORE and Counterfactual Explainers each illuminating different facets of our models' logic, we arrive at a final understanding of why and how these black box learners stumble on the rarest edge cases. These extreme misclassifications appear to stem from the models' inherent limitations when faced with very sparse and variable data, addressing rare feature combinations that the learners simply haven't seen enough of to handle reliably.

6 Regression

Although the classification approach yielded non-trivial results, it relies on the discretization of `averageRating`, a continuous fature that we have available. As previously mentioned, the adopted evaluation metrics did not account for the ordinal nature of the target. For this reason, we decided to reframe the problem using advanced regression techniques, leveraging the same datasets and features used in the classification task. As done before, target values below 4 were clipped to 4. To make the results more interpretable, all evaluation metrics are computed using the unnormalized target variable.

6.1 Random Forest

Given that Random Forest performed best in the classification task with a reasonable computational cost, it was the first regression model evaluated. Hyperparameter tuning was performed via random search, using the **friedman_mse** splitting criterion. To mitigate overfitting, the **minimum number of samples** required to split an internal node was set to **5**. Mean Squared Error **(MSE)** was used as the main evaluation metric, along with **5-fold cross-validation**.

The explored hyperparameters included the maximum depth of decision trees (from 10 to 20, as well as unlimited depth) and the number of estimators (ranging from 10 to 300). The best configuration included **100 estimators with no depth limit**.

Table 6: Random Forest performance

Metric	Test	Cross Validation
MSE	1.108	1.008 ± 0.005
R^2 Score	0.364	0.417 ± 0.004
MAE	0.780	0.743 ± 0.004

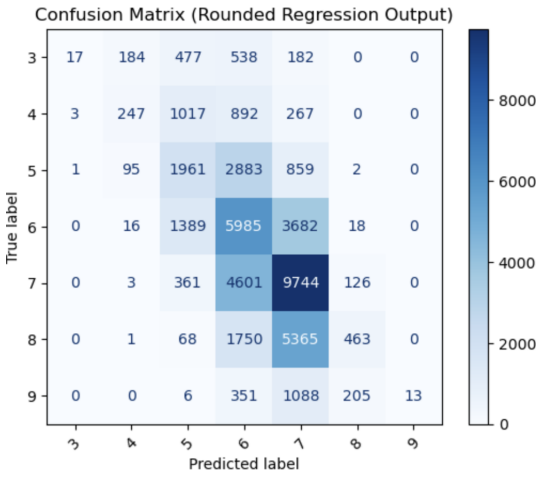


Figure 9: Confusion matrix (discretized output)

Cross-validation metrics show slightly better performance than those on the test set, likely due to overfitting. Despite exploring several simpler configurations, no comparable alternatives were found. Therefore, this model was retained.

To compare regression with the previous classification task, predictions were discretized and a confusion matrix was generated (Figure 9). As expected, the regression model made fewer large errors—misclassifications far from the diagonal are significantly reduced. However, the model tends to be conservative, rarely predicting extreme ratings, resulting in lower recall for such values.

6.2 AdaBoost

To address the limitations of Random Forest, AdaBoost with **decision trees as base estimators** was also evaluated. Due to computational constraints, hyperparameter tuning was conducted using a single validation split (20% of the training set), meaning no standard deviation is reported. Several combinations of learning rate, number of estimators, and tree depth were tested. The best result, in terms of MSE on the validation set, was achieved with **50 estimators, a maximum depth of 10, minimum samples split of 5, and learning rate of 0.1**.

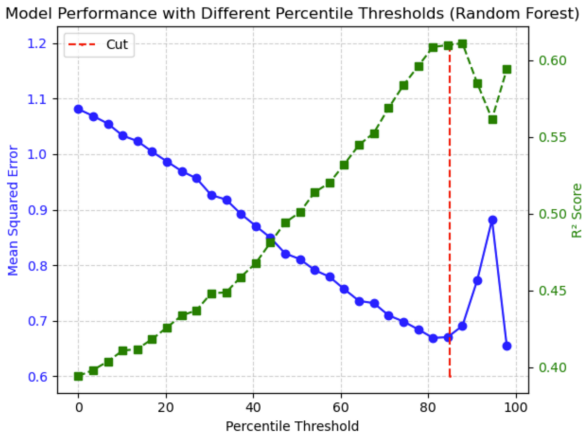
Metric	Validation	Test
Mean Squared Error	1.187	1.067
R^2 Score	0.319	0.387
Mean Absolute Error	0.829	0.788

Compared to Random Forest, AdaBoost performs slightly worse overall. Interestingly, test performance exceeds validation performance, which may indicate that the model lacks generalization and that performance differences are influenced by data sampling. Like Random Forest, AdaBoost rarely predicts extreme ratings. Since the qualitative behavior is similar, the corresponding confusion matrix was omitted for brevity.

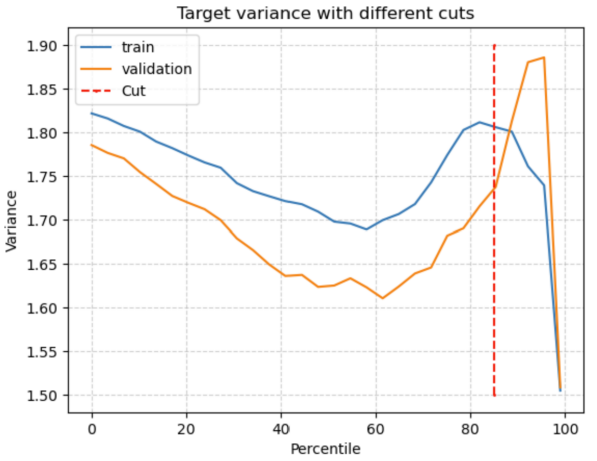
6.3 Restricting Regression to Popular Titles

Given the limitations of the previous approaches, we explored the hypothesis that model performance could improve when restricting the regression task to a more homogeneous subset of the data. Most titles in the dataset received very few ratings, leading to high statistical noise, especially for low vote counts (see Figure 1). Conversely, titles with many votes are generally more relevant and consistent in rating behavior, making predictions more meaningful.

After testing weighted regression approaches with limited success, we evaluated performance as a function of a threshold on the number of votes. Random Forest was trained and validated on increasingly restricted datasets, and we also tracked the variance of the target variable to ensure the problem did not become trivial.



Validation scores at increasing `numVotes` thresholds



Rating variance at increasing `numVotes` thresholds

Performance improves markedly as the threshold increases, while rating variance remains relatively stable until the dataset becomes too small. The red dashed line, slightly above the 80th percentile, marks the selected threshold. At this point, performance is maximized and variance is preserved, while retaining approximately **20% of the original dataset**. The model was then retrained on the entire filtered training set and evaluated on the corresponding test set.

Table 7: Random Forest performance on selected titles

Metric	Test	Validation
MSE	0.543	0.684
R^2 Score	0.672	0.604
MAE	0.537	0.601

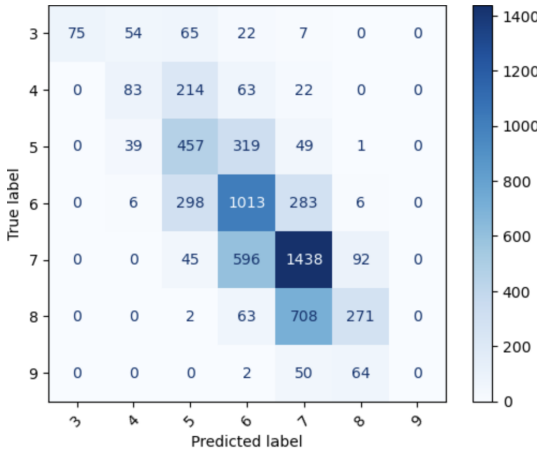


Figure 10: Confusion matrix for filtered dataset

Interestingly, the test scores are again better than the validation scores, suggesting the model could benefit from more training data even within this filtered subset. Nonetheless,

the model continues to struggle with extreme values—ratings above 9 were never predicted. While this analysis is not exhaustive, it highlights that redefining the problem scope can yield substantial performance gains. Depending on the application, tailoring the dataset may prove more beneficial than pursuing more complex models. In any case, predicting average ratings remains a challenging task, due to both the intrinsic variability of the target variable data and the possibility that the training data is not sufficient to improve the model further. Notably, the release year emerged as the fifth most important feature in the final model, hinting at the relevance of **temporal dynamics** in user behavior.

7 Time Series Analysis

This section focuses on the analysis of daily box office revenues for a selection of well-known films extracted from the IMDb dataset. These data differ from those used in the previous analyses: here, the focus is exclusively on films, and in addition to revenue, the average rating and genre for each title are also available.

For each time series, the film genre and average rating are provided, with the rating also available in a discretized categorical form. This discretized rating will be used as the target variable for subsequent analysis, similarly to what was done with the tabular data.

The dataset contains 1,134 time series, each representing the daily revenues for the first 100 days following the film's release date.

It was later discovered during analysis that the data actually correspond to the first 50 days and are partially synthetic.

7.1 Data Understanding and Preprocessing

In order to effectively compare the relevant characteristics of the time series without losing useful information from other variables, several transformations were explored. Since film revenues can vary greatly, a logarithmic transformation was applied to all series to emphasize relative rather than absolute monetary differences.

Subsequently, each series was normalized by dividing by its maximum value, allowing comparison of their shapes independently of scale. It was verified that the Spearman correlation between the average revenue and the average rating is very low, indicating that normalization does not significantly alter the relationship with the ratings, which are the classification target.

A common minimum value (e.g., zero) was not imposed on the series because the relative minimum (i.e., the ratio between the minimum and maximum revenue within each series) shows a linear correlation of 0.45 with the average rating. Fixing the minimum value would have resulted in the loss of potentially relevant information for the analysis.

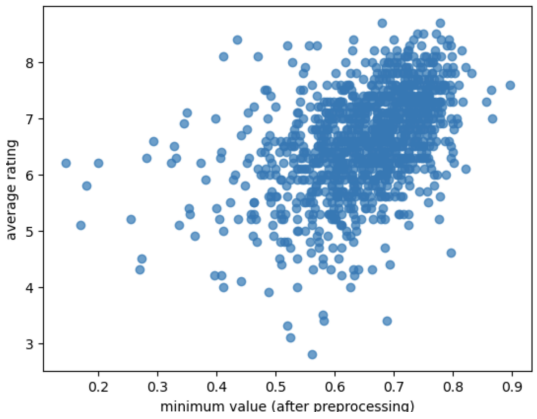


Figure 11: Scatterplot of minimum value (after preprocessing) vs average rating

After these preprocessing steps, the analysis moved to the frequency domain using the Fast Fourier Transform (FFT) to investigate periodic trends in the signals. To do so, the frequency spectrum was computed for all time series, and the magnitudes of the positive frequency components were summed to

obtain the average spectrum.

Besides the large low-frequency components due to the main trend, noticeable biweekly peaks emerged both from the frequency analysis and from the average of the signals. These peaks persist in the average because the typical release day for films is Friday, causing these peaks to be in phase across series.

This finding was rather unexpected, as weekly trends would have been more intuitive, prompting further investigation. Notably, the observed frequency appeared to be halved, suggesting the possible presence of interpolated data between real observations.

To explore this hypothesis, the average time series across all samples was first differentiated, revealing that successive points, specifically transitions from odd to even indexed days, tended to have very similar derivatives. This consistent behavior suggests a smooth, nearly linear trend between those points, which is atypical of raw observational data but characteristic of synthetic data generated through interpolation.

Based on this observation, it was hypothesized that a data augmentation process involving linear interpolation may have been applied. To test this, for each unprocessed time series and for each pair of "odd" day indices (e.g., first–third, third–fifth, etc.), the relative difference was calculated between the average of the outer pair and the actual value of the intermediate (even) day. If the intermediate point had been generated by linear interpolation, this deviation would be minimal.

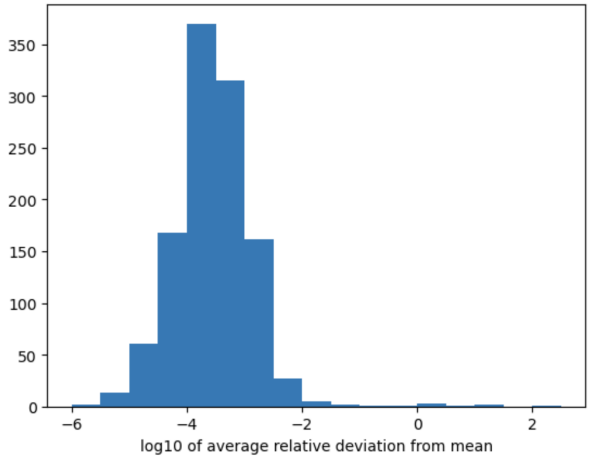


Figure 12: Logarithm of average relative deviation from the mean of supposed synthetic data.

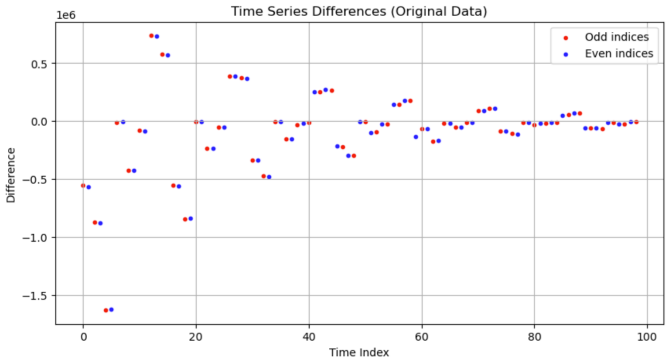
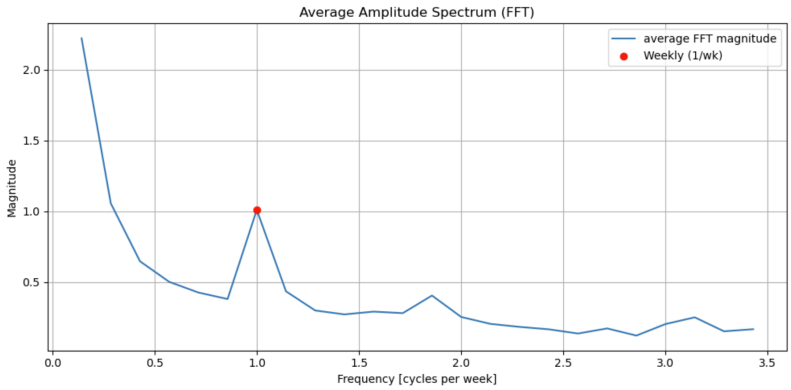


Figure 13: Differentiated average signal. Consecutive values show similar behavior, suggesting interpolation.

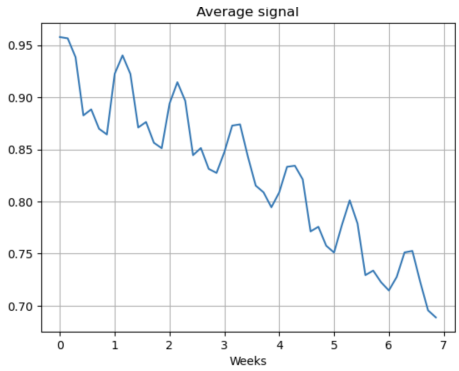
Except for about ten time series (many of which were later identified as outliers), the deviation was below 1%, suggesting that, aside from minor noise, the data on even-numbered days are indeed synthetic or, at the very least, strongly dependent on neighboring values. As a result, these even-day data points were removed without affecting the previously observed patterns, and the frequency analysis was repeated.

These findings support the hypothesis that the dataset includes synthetically generated intermediate values, likely introduced to artificially increase temporal resolution.

Repeating the frequency analysis revealed, as expected, a prominent peak at the weekly frequency. This confirms that the underlying periodicity, initially obscured by the presence of synthetic data or long-term trends, corresponds to a weekly cycle, which aligns with the typical behavior expected in this type of time series.



Average Amplitude Spectrum



Average signal (only odd days)

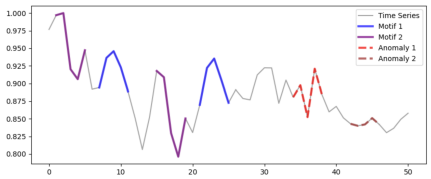
From now on the analysis will be performed only on the data of odd days, comprising 50 days for each time series.

7.2 Motif and Discord Discovery

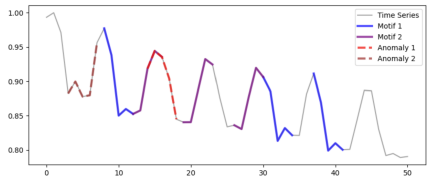
The analysis of motifs and discords allows for the identification of, respectively, the most frequently repeated and the most uniquely anomalous subsequences within time series. This helps reveal recurring or exceptional local patterns that may be overlooked in a global analysis.

In this study, a sliding window of length 5 was applied to detrended signals for motif and discord detection. This choice was made to avoid detecting trivial patterns related to the dominant trend, common across most sequences, and instead focus on more meaningful local variations.

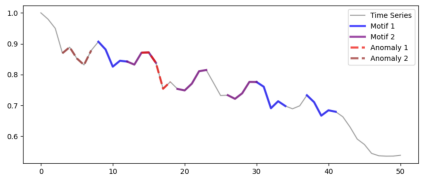
This approach primarily identified motifs associated with weekly revenue peaks, which are typical in the cinema industry. However, it did not reveal any substantial differences across the various rating classes. Figure 14 shows examples of motifs detected on a representative time series for different rating categories.



High Rating



Medium Rating



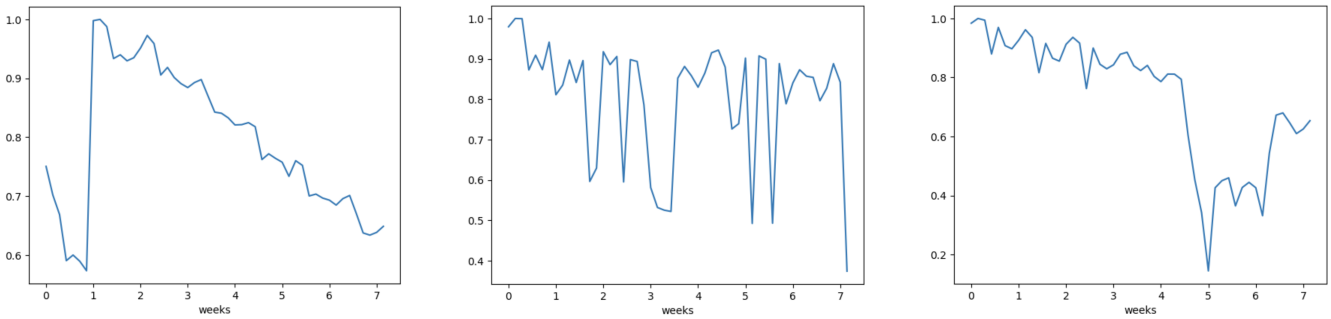
Medium-Low Rating

Figure 14: Motif and discords on different rating categories.

More interesting results emerged from the analysis of anomalies, as illustrated in Figure **??**. For this analysis, Euclidean distances were computed between each time series and the dataset average. To prevent the dominant trend (which is mostly linear) from disproportionately influencing the distance (since Euclidean distance is highly sensitive to differences in scale or offset) the main trend was first removed from each series through linear detrending.

Time series with a distance greater than three standard deviations from the mean were considered outliers. A total of 15 anomalous sequences were identified and subsequently excluded from further analyses to preserve the reliability of the results.

Unlike the approach used for discord discovery, which focuses on identifying unusual subsequences within a single series, this method compares entire time series against each other, providing a complementary perspective that is particularly useful for detecting sequences that deviate significantly from typical behavior.



Double Trend (10) Unstable trace (2) Sudden Change (3)

Figure 15: Different types of Instance Outliers

In 15, representative examples are shown for each identified type of outlier, with the number of time series indicated in the caption. The detected anomalies can be attributed to various factors: different film release dates depending on location, the presence of missing or discontinuous data, or unique characteristics of certain sequences that make them significantly different from the typical behavior observed in the others. However, no significant distinguishing features were found in terms of rating or genre.

An analysis at the level of individual data points was not performed, as there is no solid criterion to reliably characterize them as anomalies. Overall, the time series appear to be regular when evaluating the maximum variation between consecutive days for each sequence (excluding anomalous ones).

7.3 Clustering

Before proceeding with the classification phase, a clustering analysis was carried out to explore the presence of natural groupings within the time series data, independently of the provided labels.

Two distinct approaches were adopted: K-means with DTW (Dynamic Time Warping) distance, to account for temporal variability across the series; and hierarchical clustering applied to the first five principal components obtained via PCA on features extracted with TSFRESH, in order to reduce dimensionality while preserving the temporal structure.

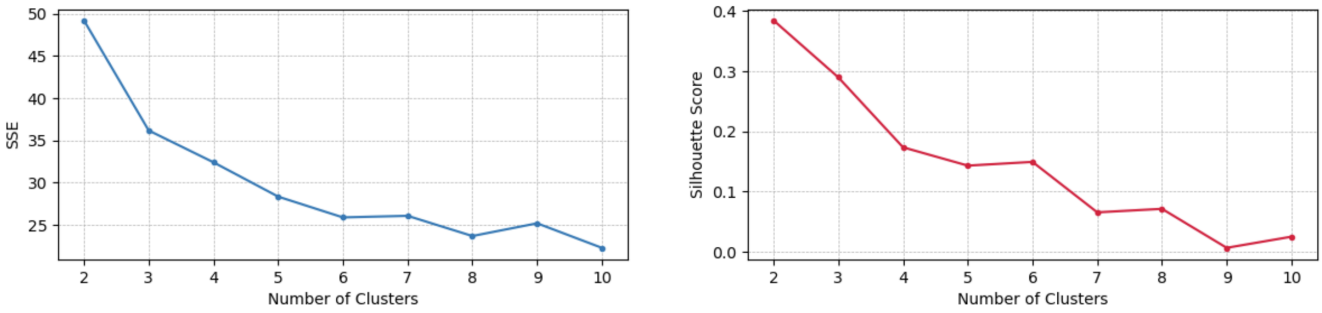


Figure 16: K-means SSE and Silhouette Score for different k (DTW with window 0.05).

In the case of K-means, k = 2 was selected as the optimal number of clusters, since higher values led to a clear decrease in the silhouette score, indicating lower internal consistency within the groups.

No significant improvements were observed when using longer window sizes for the DTW (Dynamic Time Warping) distance: the results remained largely unchanged, likely because the typical weekly fluctuations in box office revenue are mostly in phase across the different time series.

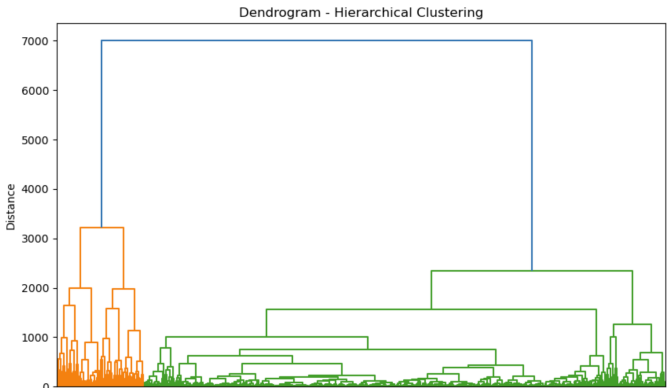
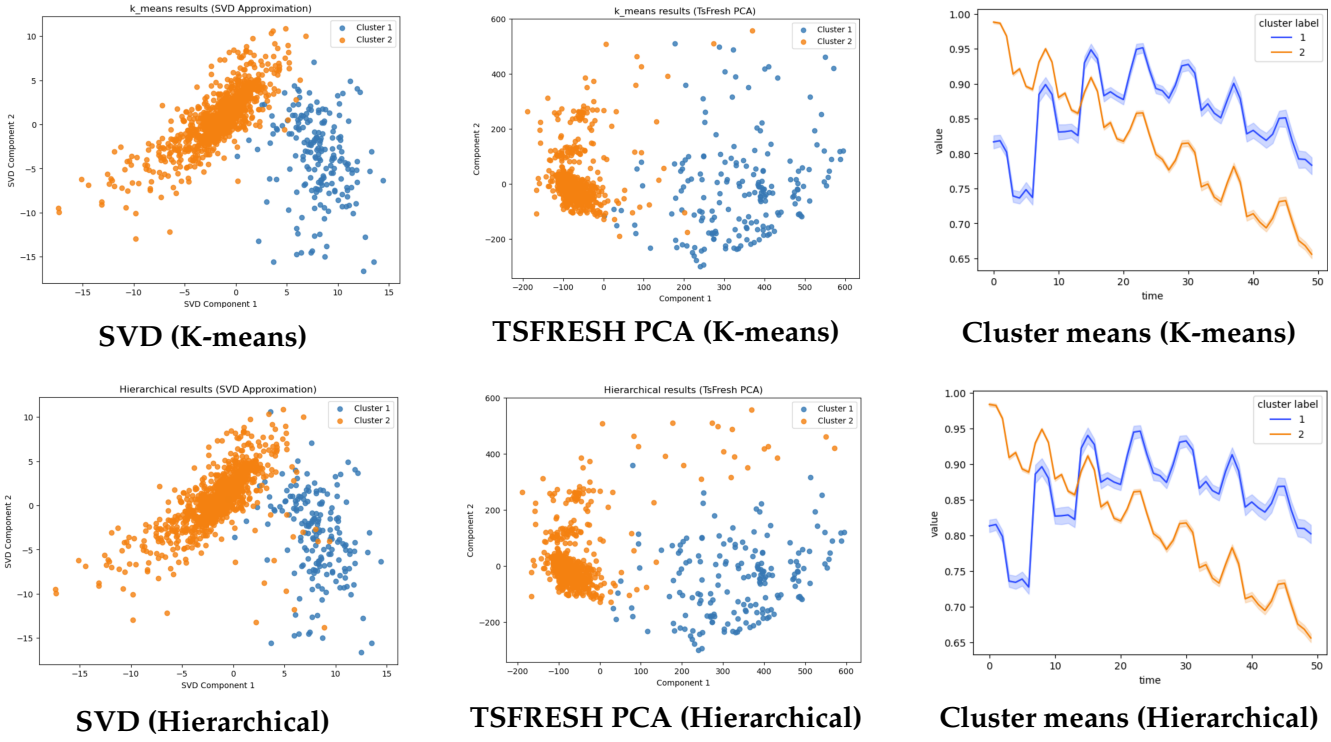


Figure 17: Dendrogram of Hierarchical Cluster on TsFresh approximation reduced with PCA.

In figure 17 the dendrogram obtained with hierarchical clustering, using the *Ward* method, in which the two clusters identified by cutting at the highest gap have been highlighted. This is the best result among the agglomerative hierarchical methods tested, producing a clear gap between the clusters.



SVD (K-means) **TSFRESH PCA (K-means)** **Cluster means (K-means)**

SVD (Hierarchical) **TSFRESH PCA (Hierarchical)** **Cluster means (Hierarchical)**

Figure 18: Comparison of clustering results. Top row: K-means — SVD projection, PCA-reduced TsFresh features, and cluster means. Bottom row: Hierarchical clustering — same structure for comparison.

Although the approaches are radically different and act on different representations, the similarity between the results is evident, both by visualizing the average of the signals and the distribution in the different projected spaces (figure 18). Both identify in a cluster the traces that assume maximum value at the beginning (cluster 2), separating them from the others. Cluster 2 is strongly characterized by a linear trend (after logarithmic transformation) and contains the majority of the series (929 for k-means, 939 with hierarchical).

The rating distribution is interesting: the cluster containing films that achieve maximum success in the weeks following release includes about a third of the films from the highest rating category, despite consisting of only a fifth of the total dataset, and has an average rating of 7.23 compared to the total average of 6.58. The result for hierarchical clustering is very similar.

A straightforward interpretation is that movies reaching peak popularity shortly after release are likely well-received by the audience. This sustained success is reflected in higher average relative income in the final days of the observation window, which correlates with higher ratings. The clustering results capture this pattern, grouping together movies with similar post-release performance and, consequently, similar ratings.

8 TimeSeries Classification

As in the previous section, supervised classification methods were also applied in this case to predict the discretized rating of the films. However, in this **Time series classification** we examined a broad collection of models to uncover temporal patterns for movie income.

Our goal is to predict a newly defined ordinal target **rating_category_ord**,

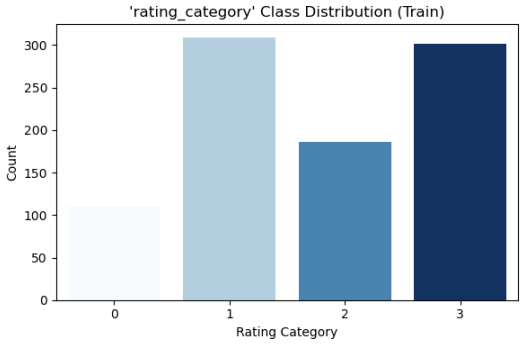


Figure 19: Target Distribution on train set

which consolidates IMDb's original five nominal tiers into four classes: 0 (Low + Medium Low), 1 (Medium), 2 (Medium High), and 3 (High).

Even after merging, class 0 remained the smallest group, but this consolidation enabled non-zero recall across different experiments. Infact, by maintaining the five original classes alone, or even applying some oversampling tecniques like ADASYN, the results for the smallest class were always poor.

The dataset was split into train and test sets (80%/20%), maintaining the proportion of the classes.

To compare 31 diverse algorithms on equal footing, we adopted a standardized evaluation workflow: hyperparameter search & training with a Randomized search over each method's parameter grid with a 3-fold stratified cross-validation, prediction on the test set, performance assessment by focusing on classification report and confusion matrix, and multiclass ROC-AUC curves.

8.1 Instance-Based Models

The classic **KNeighborsClassifier** assigns labels by majority vote among the k closest series under a chosen distance (Euclidean or Manhattan). We searched `n_neighbors` in {10, 20, 25, 30} and `metric` in {"euclidean", "manhattan"} over eight randomized trials. The best performance came with **k = 30** and **Euclidean** distance, yielding **0.50 test accuracy** and **0.45 weighted F1**. Overall, varying k or switching to Manhattan distance had minimal impact, indicating that weekly or seasonal peaks are generally well aligned.

To respect temporal distortions, we also applied **KNeighborsTimeSeriesClassifier**, a k-NN variant using Dynamic Time Warping with three window constraints (`window` = 0.1, 0.5, 0.9) and `n_neighbors` in {10, 20, 25, 30}. The optimal setup achieved **0.51 accuracy**. Despite its higher computational cost, DTW offered only a marginal gain over Euclidean k-NN, suggesting that full alignment flexibility is unnecessary for this task.

8.2 Tree-Based Models

Random Forest employs an ensemble of decision trees. We tuned the main five hyperparameters `n_estimators` (100, 200, 300, 500), `max_depth` (None, 10, 20, 30), `min_samples_split` (2, 5,

10), `min_samples_leaf` (1, 2, 4), and `class_weight` (None, "balanced"). The best configuration used **300 trees** yielding **0.49 test accuracy**.

Instead, **Random Forest + Catch22** first transforms each series into 22 summary statistics (autocorrelations, entropy measures, and other shape descriptors) using the Catch22 feature set, then applies the same Random Forest grid. The optimal setup featured **100 trees** producing **0.51 accuracy**. Incorporating Catch22 slightly boosted overall accuracy and decreased significantly the computational time.

8.3 Interval-Based Models

ProximityTree builds a decision tree by recursively splitting series based on nearest-neighbor distances. We tuned the main two hyperparameters, even though the best configuration proved catastrophically slow and yielded only **0.14 accuracy**. In practice, the cost of repeated DTW calculations at each split far outweighed any minor gains, making it impractical for our series.

CanonicalIntervalForest (CIF) samples random subintervals from each series, computes summary statistics (mean, slope, etc.) on those intervals, and fits an ensemble of trees to these localized features. Our CV grid covered `n_estimators` (100, 200, 300), `n_intervals` (None), `max_interval` (None, 7, 14), and `base_estimator` (DecisionTreeClassifier, ExtraTreeClassifier). The optimal setup used **100 trees** with **max_interval = None**, achieving **0.49 accuracy** and **0.44 weighted F1**. CIF's focus on canonical intervals helps uncover short-lived bursts (like opening-week spikes), but its randomized interval sampling can overlook longer-term trends.

8.4 Dictionary-Based Models

BOSSEnsemble leverages multiple Bag-of-Symbolic-Fourier-Approximation (BOSS) classifiers, each converting subseries into symbolic "words" and then voting across the ensemble to make a final prediction. We tuned its ensemble configuration hyperparameters `max_ensemble_size` (100, 200, 500), `min_window` (7, 14), and `alphabet_size` (2, 4). The optimal setup used **500 ensemble members**, **min_window = 14**, and **alphabet_size = 2**, resulting in **0.44 test accuracy**. While BOSSEnsemble captures repeating local patterns effectively, it can miss subtle temporal alignments and long-range trends, which limits its discrimination among similar classes.

SFAFast + KNN combines a rapid SFA transform, binning series windows into words using parameters like `word_length` (4,6,8), `alphabet_size` (2,4), `window_size` (7,12,14), with a Euclidean KNN. We tuned all SFA hyperparameters alongside `knn__n_neighbors` in {10, 20, 30} and the best configuration with **word_length = 8**, **alphabet_size = 2**, **window_size = 7**, and **n_neighbors = 10** yielded **0.49 accuracy** and **0.45 weighted F1**. This pipeline strikes a balance between capturing localized symbolic patterns and preserving overall distance-based grouping, making it one of the stronger dictionary-based models.

8.5 Shapelet-Based Classification

Shapelets are compact, highly discriminative subsequences of a time series that best separate classes by their distance to each series. In our dataset's context, they seek to capture local patterns, such as an opening weekend spike or a mid-run recovery that may distinguish one rating category from another.

We paired the **RandomShapeletTransform** with a **KNN** classifier, tuned the key RST parameters: `n_shapelet_samples` (100, 1000, 10000), `min_shapelet_length` (3), `max_shapelet_length` (8, 15, None), and `remove_self_similar` (True). The best setup constrained shapelets to **3–8 days**, retained **1000 samples**, and removed duplicates, resulting in **810 unique subsequences** and achieving **0.44 test accuracy** and **0.40 weighted F1**.

The distribution of their lengths (Figure 20) is centered between 5 and 7 days, reinforcing the presence of weekly periodicity across series. The associated information gain (IG) values, shown in Figure 20, are strongly skewed: while most shapelets provide only marginal discriminative power (IG < 0.02), a small subset achieves values up to 0.08. These rare, highly informative shapelets originate almost exclusively from movies in class 3 ("High" rating), reflecting the impact of class imbalance on the sampling process.

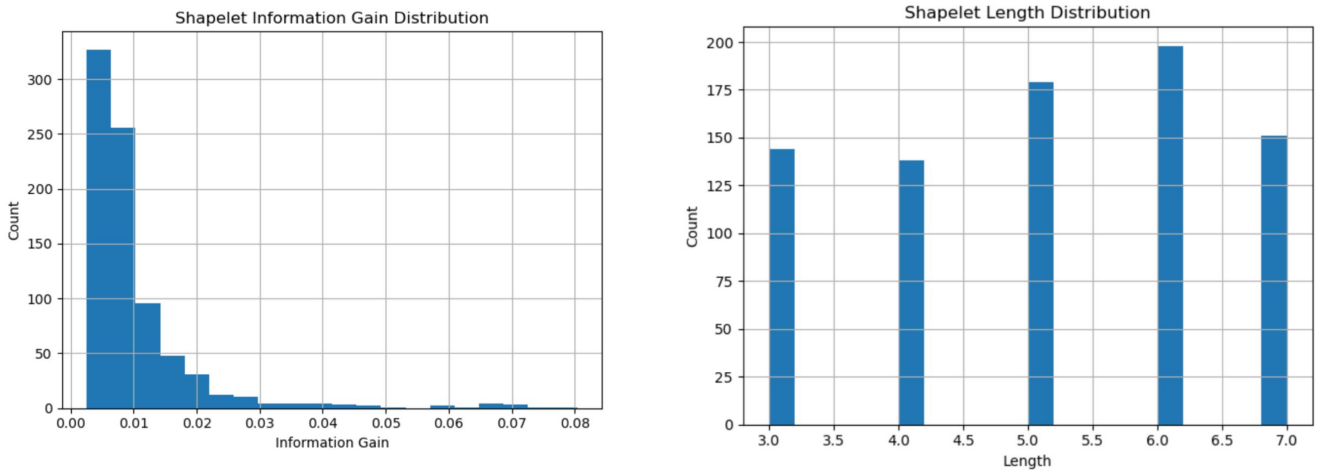


Figure 20: Summary statistics of the extracted shapelets: (left) Information Gain histogram; (right) distribution of shapelet lengths.

Despite the interpretability and compactness of shapelet-based representations, the model achieves a relatively low accuracy rather than simple knn. This is mainly due to two factors: the uneven distribution of informative shapelets across classes and the inherently local nature of the method. In particular, shapelets fail to capture global temporal dynamics, such as steady growth or decline, which appear to be more informative for rating prediction. A standard KNN classifier applied directly to the full time series, by contrast, yields superior performance, indicating that broader patterns carry more predictive power than isolated motifs.

8.6 Neural Network Models

The **MLPClassifier** captures non-linear relationships through fully connected layers. We tuned the main parameters like `hidden layer sizes`, `activation function`, `solver`, and `early stopping` and the best setup employed a three-layer architecture **(128, 64, 32)** units and **early stopping**, yielding **0.50 test accuracy** and **0.40 weighted F1**. Training was very fast and stable, but like all the next 'deep learning models' discussed, it failed to recover the smallest tiers: recall for **class 0** and **class 2** dropped to zero, highlighting its difficulty with under-represented patterns.

Convolutional networks like **CNNClassifier** learn local motifs by sliding kernels over the series. We searched over number of `epochs`, `convolutional layers`, `activation`, and `early stopping` callbacks. The optimal model ran **100 epochs** with **2 convolutional layers**, **softmax** activation and **EarlyStopping**, achieving **0.50 accuracy**. This configuration was slightly better than the MLP on minority tiers, even though still with **zero recall on class 0** and much higher computational cost.

SimpleRNNClassifier is a single recurrent layer that captures sequential dependencies. After tuning `epochs` and `unit count`, the best RNN reached **0.51 accuracy** and **0.41 weighted F1**. Its sequential memory slightly boosted overall accuracy, but it completely missed **class 2** with zero recall, indicating that simple recurrence struggles to distinguish mid range classes.

LSTMFCNClassifier is an hybrid of LSTM and convolutional blocks to combine long-term memory with local feature learning. Despite its complexity, the final model only managed **0.35**

accuracy and **0.25 weighted F1**, badly overfitting dominant patterns and failing on both **class 0** and **class 2**.

Overall, **CNNClassifier** delivered the strongest balanced performance among neural networks, but the **MLPClassifier's** rapid training time makes it a practical choice when resources are limited. All neural architectures, however, consistently struggle with at least one underrepresented class, underscoring the need for different strategies to enhance recall on rare tiers.

8.7 Kernel-Based Models

RocketClassifier applies the Rocket transform to the series, producing a high-dimensional feature vector. We tuned three hyperparameters with `rocket_transform` set to "Rocket" or "minirocket", `num_kernels` in {100, 1000, 10000}, and setting `max_dilations_per_kernel` at 32. The best configuration achieved **0.50 test accuracy** and **0.48 weighted F1**, the highest F1 across all 31 methods. This performance demonstrates MiniRocket's power to extract discriminative temporal features with minimal computation, yielding good multiclass ROC curves and solid separation.

The variant **RocketTransformer + kNN** combines the same MiniRocket features with a KNN classifier. It matched the RocketClassifier's **0.50 accuracy** but trades off some F1 performance, the hybrid benefits from kNN's non-parametric decision boundaries in the transformed space, offers an interpretable alternative that still capitalizes on Rocket's rich feature mapping.

8.8 TimeSeries Classification Conclusion

To have an overall comprehensive evaluation of all of the 31 models compared, we can see a final table summarizing all the results:

Model	Best parameters from CV	Accuracy	Weighted F1
KNN	n_neighbors: 30, metric: euclidean	0.50	0.45
KNN-DTW	n_neighbors: 20, distance: dtw, window: 0.9	0.51	0.46
Decision Tree	min_samples_split: 2, min_samples_leaf: 2, max_depth: 5, criterion: gini	0.49	0.46
DT + Catch22	min_samples_split: 2, min_samples_leaf: 4, max_depth: 5, criterion: entropy	0.48	0.42
RandomForest	n_estimators: 300, max_depth: 10, min_samples_split: 10, min_samples_leaf: 1	0.49	0.45
RandomForest + Catch22	n_estimators: 100, max_depth: 5, min_samples_split: 10, min_samples_leaf: 2	0.51	0.44
AdaBoost	n_estimators: 300, learning_rate: 1.5	0.46	0.43
HistGradient BoostingClassifier	max_iter: 100, max_depth: 10, learning_rate: 0.01	0.51	0.45
ProximityTree	max_depth: 10, distance_measure: dtw	0.14	0.07
Canonical Interval-Forest	n_estimators: 100, min_interval: 3, base_estimator: DTC	0.49	0.44
TS ForestClassifier	n_estimators: 200, min_interval: 3	0.50	0.46
ShapeletTransform Classifier	n_shapelet_samples: 10000, max_shapelet_length: None	0.41	0.38
RandomShapelet Transform + KNN	n_shapelet_samples: 1000, min_shapelet_length: 3, max_shapelet_length: 8	0.44	0.40

Model	Best parameters from CV	Accuracy	Weighted F1
ShapeletLearning ClassifierPyts	shapelet_scale: 3, n_shapelets_per_size: 0.2, max_iter: 1000, loss: crossentropy	0.41	0.30
RDSTClassifier	proba_normalization: 0.8, max_shapelets: 10000, shapelet_lengths: None	0.49	0.46
IndividualBOSS	word_length: 8, window_size: 14, alphabet_size: 4	0.36	0.36
BOSSEnsemble	max_ensemble_size: 500, min_window: 14, alphabet_size: 2	0.44	0.41
SFAFast + KNN	word_length: 8, window_size: 7, alphabet_size: 2, n_neighbors: 10	0.49	0.45
MrSEQL	symrep: sfa, seql_mode: clf	0.43	0.38
WEASEL	window_inc: 2, alphabet_size: 4, binning_strategy: information-gain	0.47	0.43
MUSE	window_inc: 2, alphabet_size: 4, bigrams: True, anova: True	0.43	0.41
WEASEL_V2	word_lengths: (7,8), min_window: 4, feature_selection: chi2_top_k	0.49	0.42
MLPClassifier	hidden_layer_sizes: (128,64,32), max_iter: 200, early_stopping: True, activation: relu	0.50	0.40
CNNClassifier	n_epochs: 100, callbacks: EarlyStopping, activation: softmax, kernel_size: 7, n_conv_layers: 2, loss: categorical_crossentropy	0.50	0.43
SimpleRNN Classifier	n_epochs: 100, callbacks: EarlyStopping, activation: softmax, units: 7, loss: mean_squared_error	0.51	0.41
ResNetClassifier	n_epochs: 100, callbacks: None, activation: softmax, loss: categorical_crossentropy	0.40	0.36
InceptionTime Classifier	n_epochs: 100, kernel_size: 40, loss: categorical_crossentropy	0.48	0.47
LSTMFCNClassifier	n_epochs: 100, lstm_size: 8, kernel_sizes: (8, 5, 3), filter_sizes: (128, 256, 128), callbacks: None, attention: False	0.35	0.25
TapNetClassifier	n_epochs: 100, callbacks: EarlyStopping, activation: sigmoid, kernel_size: (8, 5, 3), filter_sizes: (256, 256, 128), use_att: True, use_rp: True, use_lstm: True, use_cnn: True	0.33	0.18
RocketClassifier	rocket_transform: minirocket, num_kernels: 10000, max_dilations_per_kernel: 32	0.50	0.48
Rocket + kNN	num_kernels: 15000, n_neighbors: 20, metric: euclidean	0.50	0.46

The table revealed a good overall spread of performance on our four-class time series classification. As seen in the summary table, by simply watching at the performance achieved, **DTW-kNN**, **HistGradientBoostingClassifier**, and**CNNClassifier** ranked among the top three, also belonging to different families of classification models, further illustrated by their respective confusion matrices.

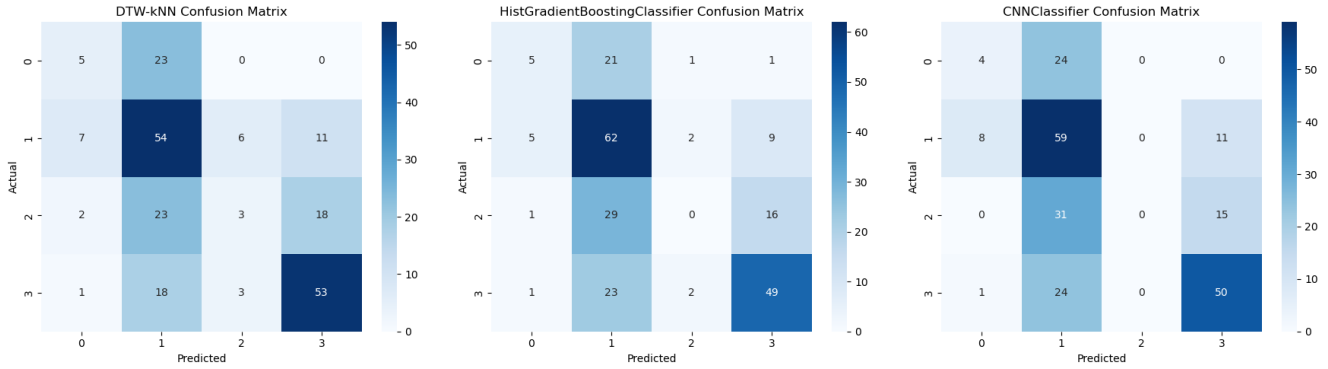


Figure 21: DTW-kNN, HistGradientBoosting and CNN classification Models

The ROC curves for our top three methods reveal a consistent pattern: even when some tiers suffer from zero recall in the confusion matrices, their binarized ROC profiles remain strong. Each model achieves high AUC on the majority classes and still demonstrates respectable separation on the under represented classes.

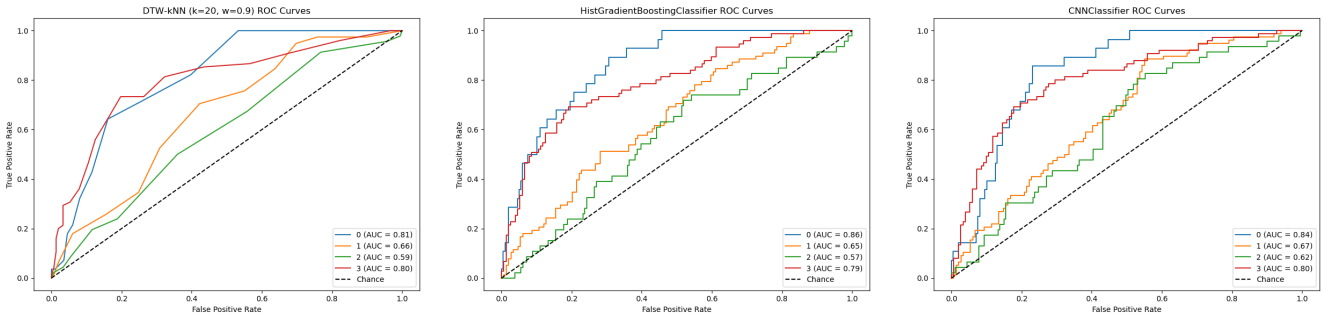


Figure 22: DTW-kNN, HistGradientBoosting and CNN ROC curves

However, when weighing both predictive quality and computational efficiency, the classic **KNeighborsClassifier** emerged as our preferred solution. Its fast fitting and inference times, coupled with crisp ROC curves, set it apart from more complex alternatives whose runtimes and complexity were less compelling. Consequently, KNN represented the best trade-off between speed, simplicity, and balanced class discrimination in our time series classification task.

9 Sequential Pattern Mining

The aim of this last section is to search for sequential patterns in the time series. To do this, the SAX approximation was used, with 5 characters and subsequences of length 7, in order to have one letter per week. However, since the series are strongly characterized by the main decreasing trend, all the patterns found are rather trivial and are not distinguished by genre or rating, except for results that are not statistically significant.

Table 9: Top Patterns with Support

Pattern	ec	ea	ed	eb	ca	dc	da	eca	cb	eda
Support	969	968	964	927	922	916	899	898	883	882

We tried to repeat the analysis on detrended signals and with other word lengths, but without obtaining significant results.