
# Masters in Data Science and Business Informatics

Data Mining: Fundamental Project report

IMDb Dataset

Academic Year: 2024/2025

Submitted By:

| Team Members | |
|---|---|
| Ankit Kumar Bhagat | 698180 |
| Ramya Shravani Dasiga | |
| Truong Manh Nguyen | 701594 |

Submitted To:

Dino Pedreschi

Riccardo Guidotti

Andrea Fedele

Computer Science Department

## Abstract

In this report, we have the IMDb dataset, a rich repository of film and television metadata, that offers valuable insights into trends within the entertainment industry. This report applies core data mining techniques to uncover significant patterns and relationships in the dataset, with attributes such as ratings, genres, title type, number of votes and many more.

The analysis begins with a data understanding phase, where the structure of the dataset is examined, and key features are visualized using histograms, bar charts, and heatmaps. Next, a thorough data preprocessing step addresses missing values, outliers, and inconsistencies, ensuring the dataset's quality for further analysis.

To uncover latent structures, clustering methods such as K-Means, DBSCAN, and hierarchical clustering are employed, grouping films based on attribute similarities. This is followed by classification techniques, including K-Nearest Neighbors (KNN), Decision Trees, and Naïve Bayes, to predict categorical outcomes such as title type or rating class.

Association rule mining is conducted using the Apriori algorithm and FP-Growth, targeting patterns between attributes like average ratings and title types to reveal influential combinations. Additionally, regression analysis is carried out to investigate relationships between dependent and independent variables. This includes multiple linear regression, ridge regression, lasso regression as well as non-linear methods like Decision Tree Regression and KNN Regression, to better capture complex, non-linear relationships within the dataset.

This comprehensive exploration highlights the potential of data mining in extracting actionable insights from entertainment data.

# Table of Contents

## 1.Data Understanding and Data Preparation

In this section, we outlined the steps taken to gather, clean, and prepare data for further analysis. The IMDb (Internet Movie Database) dataset is a comprehensive collection of information about movies, TV shows, and more, compiled from IMDb's vast online database. Widely used in research and data science, this dataset includes various attributes such as movie titles, genres, release year, ratings, and reviews, among other details. The dataset is updated as of September 1, 2024.

### *1.1.Data Semantics*

In this subsection we interpreted the meaning of the columns of our dataset. We have used the ***train*** dataset which has 23 columns and 16431 rows. We understood its type, the values stored and got some basic insights. The summary of the results can be viewed in (table 1).

| No. | Name | Description | Data Type | Dtype |
|---|---|---|---|---|
| 1 | **originalTitle** | Original title in the original language | Text | object |
| 2 | **rating** | IMDB title rating class | Categorical => Ordinal | object |
| 3 | **startYear** | Represents the release year of a title. In the case of TV Series, it is the series start year | Numerical => Interval | int64 |
| 4 | **endYear** | TV Series end year | Numerical => Interval | object |
| 5 | **runtimeMinutes** | Primary runtime of the title in minutes | Numerical => Discrete | object |
| 6 | **awardWins** | Number of awards the title won | Numerical => Discrete | float64 |
| 7 | **numVotes** | Number of votes the title has received | Numerical => Discrete | int64 |
| 8 | **worstRating** | Worst title rating | Numerical => Discrete | int64 |
| 9 | **bestRating** | Best title rating | Numerical => Discrete | int64 |
| 10 | **totalImages** | Total Number of Images for the title within the IMDb title page | Numerical => Discrete | int64 |
| 11 | **totalVideos** | Total Number of Videos for the title within the IMDb title page | Numerical => Discrete | int64 |
| 12 | **totalCredits** | Total Number of Credits for the title | Numerical => Discrete | int64 |
| 13 | **criticReviewsTotal** | Total Number of Critic Reviews | Numerical => Discrete | int64 |
| 14 | **titleType** | The type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc) | Categorical => Nominal | object |
| 15 | **awardNominations ExcludeWins** | Number of award nominations excluding wins | Numerical => Discrete | int64 |
| 16 | **canHaveEpisodes** | Whether or not the title can have episodes | Binary | bool |
| 17 | **isRatable** | Whether or not the title can be rated by users | Binary | bool |
| 18 | **isAdult** | Whether or not the title is for adult. 0: non-adult title; 1: adult title | Binary | int64 |
| 19 | **numRegions** | Number of regions for this version of the title | Numerical => Discrete | int64 |
| 20 | **userReviewsTotal** | Total Number of Users Reviews | Numerical => Discrete | int64 |
| 21 | **ratingCount** | The total number of user ratings | Numerical => Discrete | int64 |
| 22 | **countryOfOrigin** | The country where the title was produced | Categorical => Nominal | object |
| 23 | **genres** | The genre(s) associated with the title (e.g., drama, comedy, action) | Categorical => Nominal | object |

Table 1 : Data Semantics

### *1.2. Distribution of the variables and statistics*

In this subsection, we explored the distribution of variables and key statistical metrics to understand the underlying patterns and characteristics of the data. Here,we examined measures such as mean, median, mode, min, max, and standard deviation. We gain insights into the central tendency and spread of each numerical variable, which can be seen in (table 2)

| Variables | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **startYear** | 16431 | 1991.87 | 26.12 | 1878 | 1978 | 1997 | 2013 | 2024 |
| **runtimeMinutes** | 16431 | 43.14 | 51.90 | 0 | 0 | 30 | 80 | 3000 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **awardWins** | 13813 | 0.49 | 2.97 | 0 | 0 | 0 | 0 | 145 |
| **numVotes** | 16431 | 1492.15 | 20137.71 | 5 | 15 | 36 | 148.5 | 966565 |
| **worstRating** | 16431 | 1.00 | 0.00 | 1 | 1 | 1 | 1 | 1 |
| **bestRating** | 16431 | 10.00 | 0.00 | 10 | 10 | 10 | 10 | 10 |
| **totalImages** | 16431 | 11.48 | 74.25 | 0 | 1 | 1 | 6 | 3504 |
| **totalVideos** | 16431 | 0.27 | 3.12 | 0 | 0 | 0 | 0 | 258 |
| **totalCredits** | 16431 | 61.34 | 174.02 | 0 | 16 | 34 | 65 | 15742 |
| **criticReviewsTotal** | 16431 | 2.79 | 15.41 | 0 | 0 | 0 | 1 | 533 |
| **awardNominations ExcludeWins** | 16431 | 0.56 | 3.96 | 0 | 0 | 0 | 0 | 197 |
| **isAdult** | 16431 | 0.03 | 0.16 | 0 | 0 | 0 | 0 | 1 |
| **numRegions** | 16431 | 3.55 | 5.85 | 1 | 1 | 1 | 3 | 69 |
| **userReviewsTotal** | 16431 | 7.23 | 66.50 | 0 | 0 | 0 | 2 | 5727 |
| **ratingCount** | 16431 | 1492.92 | 20145.39 | 5 | 15 | 36 | 149 | 967042 |

Table 2: Descriptive Statistics

In the table above, we can see that the variable awardWins has missing values. worstRating and bestRating has only one value. The minimum value of numvotes and ratingCount is 5 and reaching approximately to a million. Additionally, most of the data variables are highly skewed and variance, to see the shape of the distribution—whether normal, skewed, or uniform—we plotted histogram (figure 1) and some of them are:

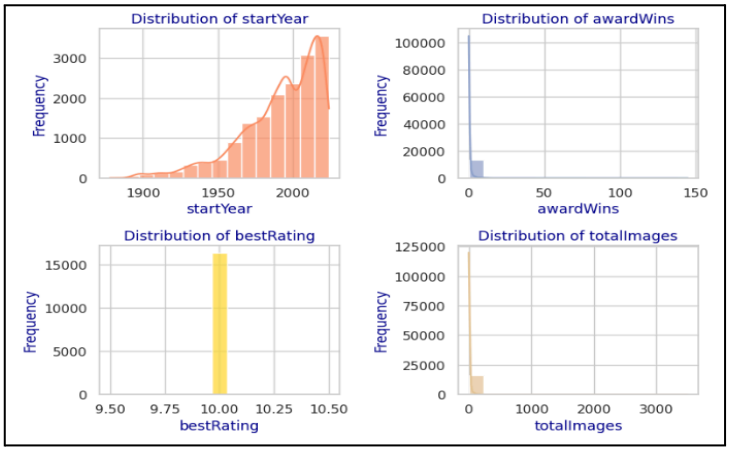



Figure 1: Data Distribution

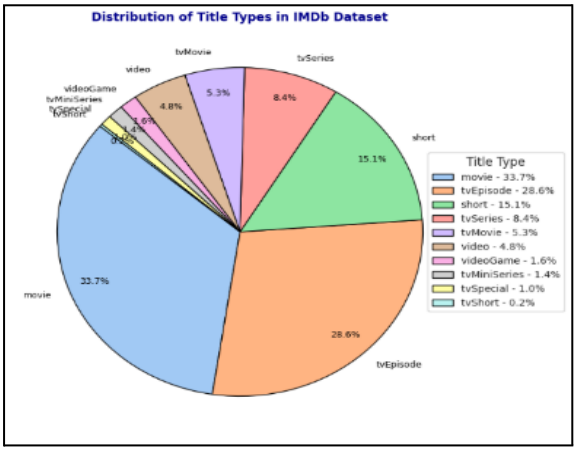



Figure 2: Distribution of titleType

To get more insights of the variables we performed some exploratory data analysis. Figure 2 display, the movie contains nearly 33% of our data. Figure 3 explains, with time the number of films released have increased. Figure 4 states, drama and comedy contains more than half of the dataset.

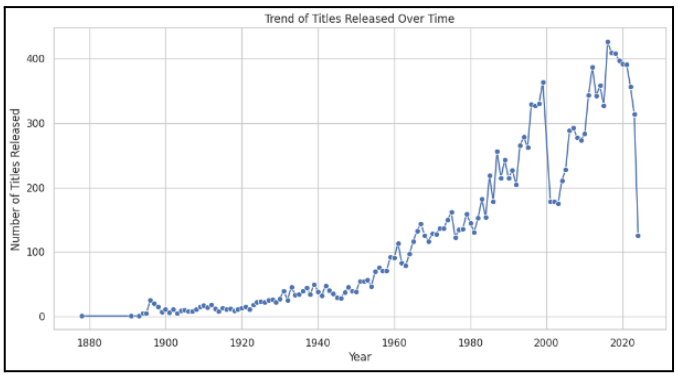



Figure 3: Number of films over the year

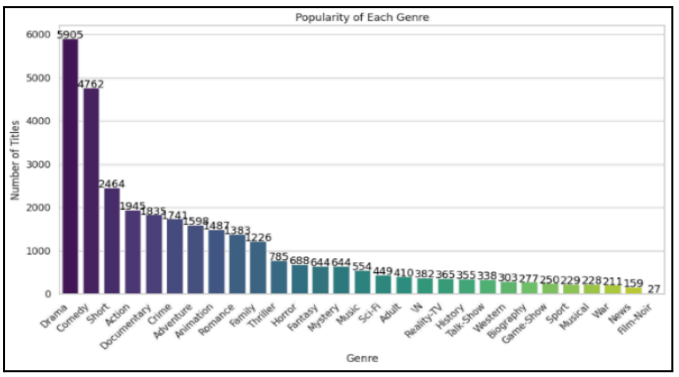



Figure 4: Frequency of genre

### *1.3. Assessing data quality*

In this section, we addressed issues like missing values, outliers, duplicates, and inconsistencies. We evaluated various aspects of our data to ensure the reliability and accuracy of our analysis and model. We manually found that the dataset is very sparse and biased. Our initial findings are mentioned in the table 3.

| No. | Variable | Finding | Thought of Action |
|---|---|---|---|
| 1 | **originalTitle** | Can be treated as Primary key | Check for duplicacy |
| 2 | **Rating** | It's a range | Convert to average |
| 3 | **startYear** | Its year | No changes to be done |
| 4 | **endYear** | Lots of null values. Only 814 (5%) records | Remove |
| 5 | **runtimeMinutes** | 4852 (30%) rows have '\N' values | Can be treated with API |
| 6 | **awardWins** | 11971 (72%) rows have 0 as values | Transformation |
| 7 | **numVotes** | Only 1219 (7%) rows have values >1000 | Remove Outlier |
| 8 | **worstRating** | All rows have same value i.e. 1 | Remove |
| 9 | **bestRating** | All rows have same value i.e. 10 | Remove |
| 10 | **totalImages** | Only 285 rows have values >100 | Remove Outlier |
| 11 | **totalVideos** | 14821 (90%) rows have 0 as values | Transformation |
| 12 | **totalCredits** | Only 2298 (14%) rows have values >100 | Remove Outlier |
| 13 | **criticReviewsTotal** | 11439 (70%) rows have 0 as values | Transformation |
| 14 | **titleType** | Textual categorical value | Encoding |
| 15 | **awardNominations ExcludeWins** | 14427 (87%) rows have 0 as values | Transformation |
| 16 | **canHaveEpisodes** | 14832 (90%) rows have False as values | Remove/ Transformation |
| 17 | **isRatable** | All rows have same value i.e. True | Remove |
| 18 | **isAdult** | 16005 (97%) rows have 0 as values | Remove/ Transformation |
| 19 | **numRegions** | 9573 (58%) rows have 1 as values | Transformation |
| 20 | **userReviewsTotal** | 9229 (56%) rows have 0 as values | Transformation |
| 21 | **ratingCount** | Only 1997 rows have values >500 | Remove Outlier |
| 22 | **countryOfOrigin** | Textual categorical value | Encoding |
| 23 | **genres** | Textual categorical value | Encoding |

Table 3: Data Quality Check

#### *1.3.1. Errors*

In this subsection, we addressed potential data errors to ensure data integrity. We checked for issues such as UTF encoding errors and delimiter inconsistencies (e.g., spaces, commas, tabs) that could lead to column merging or misalignment. After thorough inspection, none of these errors were detected, allowing us to proceed confidently to the next step of identifying any missing values in the dataset.

#### *1.3.2. Missing values*

In this subsection, we searched for the missing values in any column. We found that *awardWins* has 2618 records (approx 16% ) missing (figure 5). As a preliminary step, we conducted an online search to verify whether the titles in our dataset had won any awards. After checking a sample of titles, we concluded that these titles had not received any awards and thus **imputed with 0**. To confirm this further, we used the OMDB API[1], ensuring the accuracy of this information before proceeding with the next steps.

#### *1.3.3. Semantic inconsistencies*

In this subsection, we addressed data semantic inconsistencies in the IMDb dataset to ensure meaningful and accurate analysis. Semantic inconsistencies occur when data is misinterpreted or does not align with expected formats, units, or categories. In the IMDb dataset, this involved '\N' in three columns – *endYear*, *runtimeMinutes*, *genres*. The *endYear* column contained very few populated rows (only 814), so we decided to remove it entirely due to its limited utility. For the *runtimeMinutes* column, a significant number of entries were missing (4,852). To address this, we leveraged the TMDB API, successfully recovering over 2,000 runtime values. For the remaining missing values, we imputed the runtime using the mean

[1] OMDb API - The Open Movie Database

runtime for the specific titleType to which each title belonged. Lastly, in the genres column, the missing values (\N) were minimal (382 entries) and were replaced with "Unknown" for clarity.

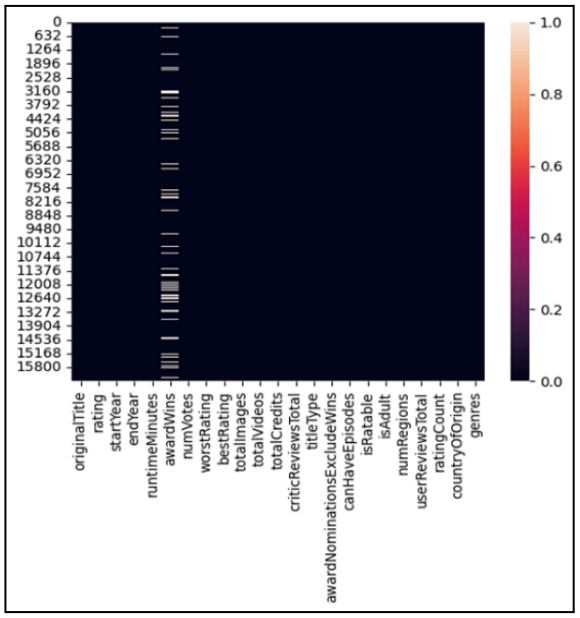



Figure 5: Heatmap of Missing Value

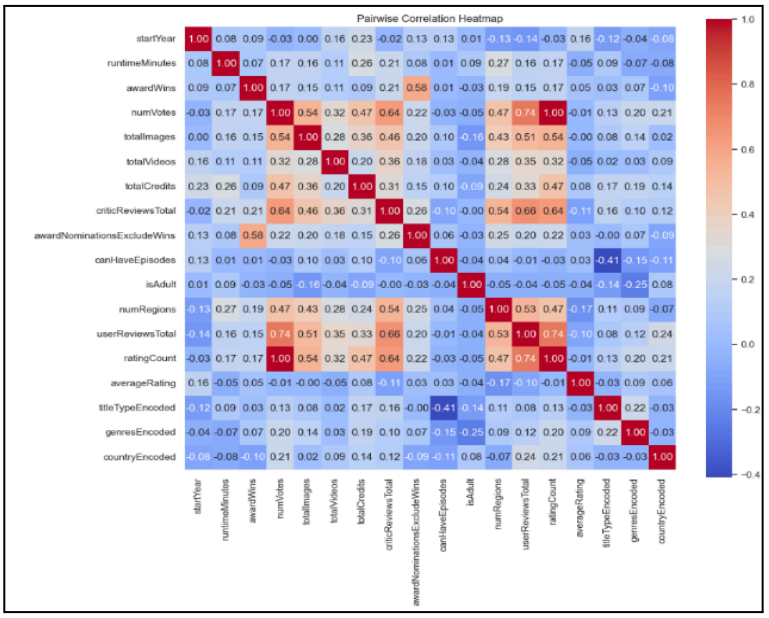



Figure 6: Heatmap of Pairwise Correlation

### *1.3.4. Outliers*

This section discusses removing outliers to enhance analysis quality. Outliers, which are data points deviating significantly from the overall distribution, can skew results or introduce bias. Using statistical methods like the interquartile range **(IQR)** and **Z-scores**, we identified and addressed these anomalies to maintain a representative and robust dataset. The **IQR** measures the range between the first quartile (Q1, 25th percentile) and third quartile (Q3, 75th percentile), with outliers defined as values:

- Below Q1 − 1.5 × IQR
- Above Q3 + 1.5 × IQR

Initially, removing outliers column-wise drastically reduced the dataset, as eliminating outliers in one column often removed rows with valuable data in other columns. To balance this, we instead calculated outlier scores for entire rows using ***Mahalanobis Distance***, preserving more data.

**Approach: Mahalanobis Distance**

- Measures the distance of a point (row) from the center of a multivariate distribution.
- Accounts for correlations between columns and is ideal for multivariate outlier detection.

This method allowed us to evaluate whether a row as a whole should be treated as an outlier or not. Using this method, we identified the most extreme outlier rows and removed the top 3%, which accounted for 493 rows, ensuring a more balanced dataset for further analysis.

### *1.4. Variable Transformations*

In this subsection, we transformed features to enhance the dataset's suitability for analysis, focusing on improving interpretability, scale consistency, and model performance. Techniques included normalization, standardization, encoding categorical variables, and applying mathematical transformations.

We started by converting the rating column into an average rating for consistency. For categorical columns (*titleType*, *countryOfOrigin*, and *genres*), we initially considered one-hot encoding and multi-hot encoding. However, this approach significantly increased the number of features due to the high cardinality: *titleType* had 10 unique values, *countryOfOrigin* had 153, and *genres* had 29. To manage this, we opted for frequency encoding, assigning categories values based on their frequency. For multi-category columns like genres, the frequencies were summed within each row. While this reduced dimensionality effectively, it could lead to overlapping representations for categories with similar frequencies, such as those in *countryOfOrigin*.

After handling duplicates, we addressed right-skewed distributions by applying a log transformation using np.log1p, which is ideal for datasets with many zeros. Finally, we normalized the dataset with a MinMax scaler to scale all features to a uniform [0, 1] range, ensuring equal feature contributions during clustering.

### *1.5. Pairwise Correlations*

In this section we visualized (figure 6) the pairwise correlation between features, we used Pearson correlation, a widely used statistical measure that quantifies the linear relationship between two variables. The Pearson correlation coefficient (r) is calculated using the formula:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}$$

Here, $x_i$ and $y_i$ are individual data points, and $\bar{x}$ and $\bar{y}$ are the means of the respective variables. The value of r ranges from -1 to 1, where:

- 1 indicates a perfect positive linear relationship.
- -1 indicates a perfect negative linear relationship.
- 0 indicates no linear correlation.

From our analysis, we observed that features such as numVotes, ratingCount, and userReviewsTotal are highly positively correlated, indicating a strong linear relationship. Conversely, canHaveEpisodes and titleTypeEncoded exhibited a strong negative correlation, suggesting an inverse relationship.

With this analysis, we concluded the dataset preparation and understanding phase. We will now proceed to the next segment: **Clustering.**

## 2. Clustering

### *2.1. Introduction and Attribute selection and standardization*

Effective clustering should be based on attributes that provide independent information, avoiding redundancy caused by high correlation. If two attributes are highly correlated, one of them should be removed to prevent redundancy. Based on the correlation matrix analysis, we have the following assessments for selecting attributes for K-means clustering:

1. **numVotes and ratingCount**: Almost perfectly correlated (correlation coefficient = 1.00). These attributes are nearly identical, and ratingCount was removed should be retained.
2. **awardWins and awardNominationsExcludeWins**: Highly correlated (0.69). Retain one key attribute, awardWins, as it reflects achievements.
3. **userReviewsTotal and numVotes**: High correlation (0.75). Retain one representative attribute.
4. **criticReviewsTotal and awardNominationsExcludeWins**: Significant correlation (0.66) but not high enough to warrant immediate removal.
5. **isAdult and other attributes**: Very low correlation with all columns (close to 0) hence, removed.
6. **runtimeMinutes**: Low correlation with other attributes. Should be retained as it represents an important characteristic (duration).
7. **totalImages and totalVideos**: Low correlation (< 0.35). Both can be retained if they provide additional information.
8. **numRegions**: Moderately correlated (0.45) with numVotes. It can be retained if regional distribution is of interest but removed.
9. **Rating and totalCredits**: Moderate correlation with other attributes but can be retained to assess additional information.

Proposed List of Attributes for Clustering: *runtimeMinutes, awardWins, numVotes, criticReviewsTotal, userReviewsTotal, totalImages, totalVideos, averageRating, totalCredits*

### *2.2. K-means*

#### *2.2.1 Identification of best k value*

We used the Elbow Method to determine the optimal number of clusters (k) by calculating the total sum of squared errors (SSE) for k values ranging from 2 to 15. The goal was to identify the 'elbow point'—where the SSE decreases more slowly afterward, indicating that increasing the number of clusters further does not result in significant improvements. This analysis revealed a clear 'elbow point' around k=4; from k=4 onward, the SSE continues to decrease but at a much slower rate than before (figure 7 left).

We determined the optimal number of clusters using the Silhouette Method, which identifies the peak Silhouette Score. While the highest score was at k = 2, other stable peaks occurred at k = 4 and k = 5 (figure 7 right).

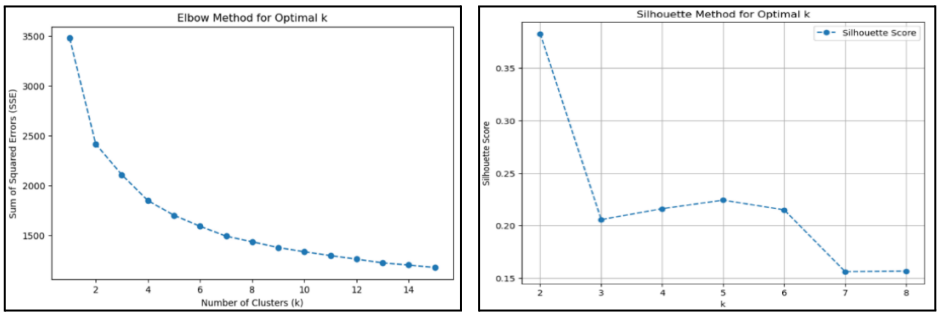



Figure 7: Finding optimal k by Elbow Method (left) and Silhouette Method (right).

| K - value | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| **Silhouette score** | 0.3826 | 0.2056 | 0.2160 | 0.2241 | 0.2149 | 0.1560 | 0.1566 |
| **Calinski-Harabasz score** | 7018.6863 | 5192.0465 | 4695.6307 | 4172.4051 | 3792.3898 | 3548.5224 | 3250.7267 |
| **David-Boundlines score** | 1.2569 | 1.6942 | 1.5630 | 1.4731 | 1.5048 | 1.5697 | 1.6392 |

Table 4: K-means clustering evaluation with metrics

Further analysis with the Calinski-Harabasz Index (higher values indicating better cluster separation) and the Davies -Bouldin Index (lower values indicating better-separated clusters) supported k = 4, as the most optimal choice (table 4). Compared to k = 2, k = 4 provides more detailed data segmentation, allowing for the identification of smaller, distinct groups.

### *2.2.2 Characterization of the obtained clusters*

The left figure 8 illustrates the K-means clustering results with k = 4, supported by PCA, in a two-dimensional space. The data points are clearly distributed into four clusters, each representing distinct characteristics based on the input attributes.

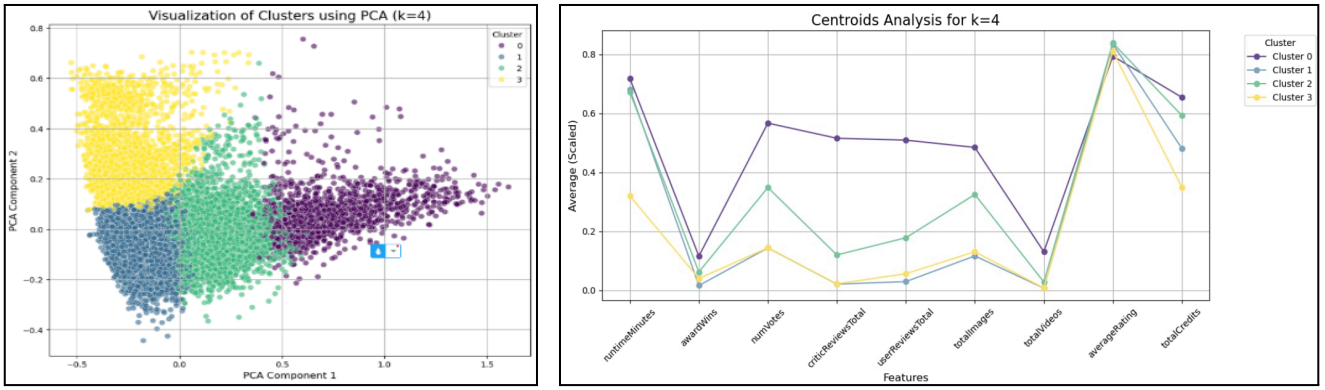



Figure 8: Clustering obtained with K-Means on the complete dataset (k=4).

The right figure 8 illustrates the centroid analysis for each cluster, highlighting the main characteristics of each group:

- **Cluster 0**: Represents the group with the highest ***runtimeMinutes*** (film duration) and the highest number of ***awardWins*** (achieved awards). The main feature of this cluster is its very high values for ***totalImages*** and ***totalVideos***, indicating that the objects in this group are heavily invested in visual and video content.
- **Cluster 1**: Characterized by ***awardWins*** at a moderate level and ***criticReviewsTotal*** at a low level, suggesting that this group contains objects receiving moderate critical acclaim. However, this group has relatively low numbers of ***numVotes*** and ***userReviewsTotal***.
- **Cluster 2**: Consists of objects with average ***runtimeMinutes***, but attributes such as ***numVotes***, ***userReviewsTotal***, and ***totalCredits*** are high, along with the highest ***averageRating***. This suggests that this cluster focuses on films with high popularity among audiences.
- **Cluster 3**: This cluster scores high in ***averageRating*** and has the highest number of ***awardWins***, but other attributes, such as ***totalImages*** and ***totalVideos***, are low. This cluster may represent high-quality films that do not require extensive investment in visual and video content and are highly regarded by prestigious award committees.

*Figure 9* visualizes the results of K-means clustering with the selected attributes, while *Table 5* shows the number of observations in each cluster.

| Cluster | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Number of Observations | 1385 | 2436 | 8023 | 4094 |

Table 5: Number of Observations in Each Cluster

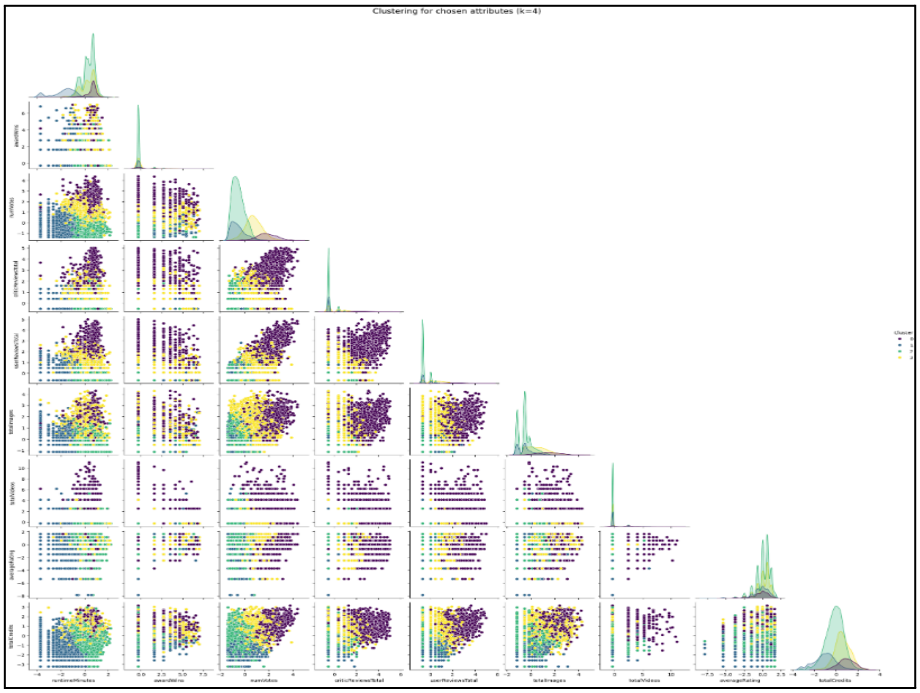



Figure 9: Clustering for chosen attributes (k=4).

## 2.3. DBSCAN

### *2.3.1. Determining eps and min points*

To determine the optimal starting range for the ***epsilon*** (ε) value in DBSCAN clustering, we used the Elbow method on the normalized dataset. The minimum samples were set to 18, calculated as twice the number of dimensions (9 attributes × 2). For ε, we computed the average distance between each point and its 18 nearest neighbours and plotted these k-distances in ascending order. The optimal ε value, identified as the point of maximum curvature on the graph, was 4.8438, as shown in *Figure 10*.

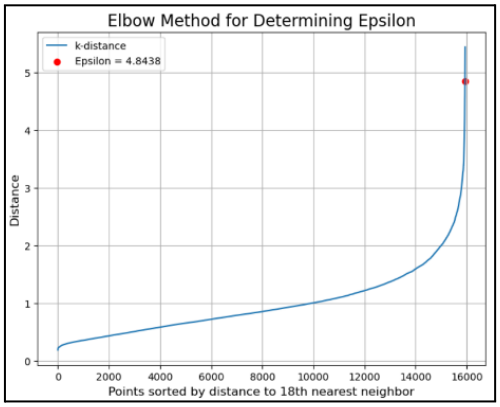



Figure 10: The best eps value by applying the Elbow method.

### *2.3.2 Clustering analysis*

After executing the algorithm with the specified parameters, we observed that there is a single large cluster (Cluster 0) and no other clusters, or no other remaining points being classified as noise hence, the Silhouette value could not be calculated. If the data does not have high-density clusters and is only distributed randomly or uniformly, DBSCAN cannot identify clear clusters. As a result, most of the data will be grouped into a single cluster (figure 11).

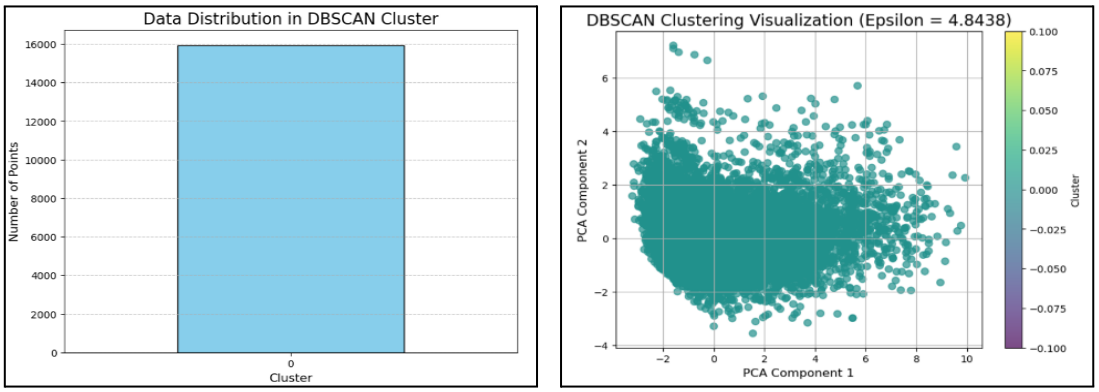



Figure 11: Data distribution and clustering obtained with DBSCAN on the complete dataset

### *2.4. Hierarchical Clustering*

#### *2.4.1. Parameter and distance function*

Before beginning agglomerative hierarchical clustering, we needed to choose a distance metric. In this case, we applied four methods: *Single*, *Complete*, *Average*, and *Ward*, using various metrics such as ***Euclidean Distance*** and ***Manhattan Distance***. We also applied the ***Minmax scaler*** before running the hierarchical algorithms.

#### *2.4.2. Dendrograms analysis*

Except for the Ward method with Euclidean distance, all other combinations produced results with many disordered clusters, making visualization challenging due to the uneven distribution of elements among the groups. The Ward method with Euclidean distance yielded a suitable clustering result (figure 12) for numerical values with Silhouette Score: 0.2636.

**Cluster 1 (Blue Line)** - This cluster likely represents films that are widely recognized by the public but lack high quality. These films tend to generate mixed reviews and low public ratings while receiving fewer awards from critics. Suitable for popular or average-quality works.

- **Highest values:** runtimeMinutes, numVotes, criticReviewsTotal, userReviewTotal, totalImages, totalVideos, totalCredits
- **Average values:** awardWins are at a medium level, not exceptional
- **Lowest value:** averageRating

**Cluster 2 (Orange Line)** - This cluster likely represents entities that win many awards and receive high ratings from a smaller audience. These works are critically acclaimed but may not be widely popular (as reflected by lower vote counts and reviews but outstanding averageRating). Likely represents artistic works recognized by experts but less mainstream.

- **Highest value:** awardWins and averageRating shows a very high average, significantly outperforming other clusters.
- **Other attributes:** All other factors remain at medium levels.

**Cluster 3 (Green Line)** - This cluster likely represents entities that are not prominent, unpopular, or receive little attention from both professionals and the public. Suitable for average or poorly performing works.

- **Lowest values:** Most attributes (*runtimeMinutes*, *awardWins*, *numVotes*, etc.) have the lowest average across clusters.
- **Chart tendency:** Concentrated around negative values or near zero.

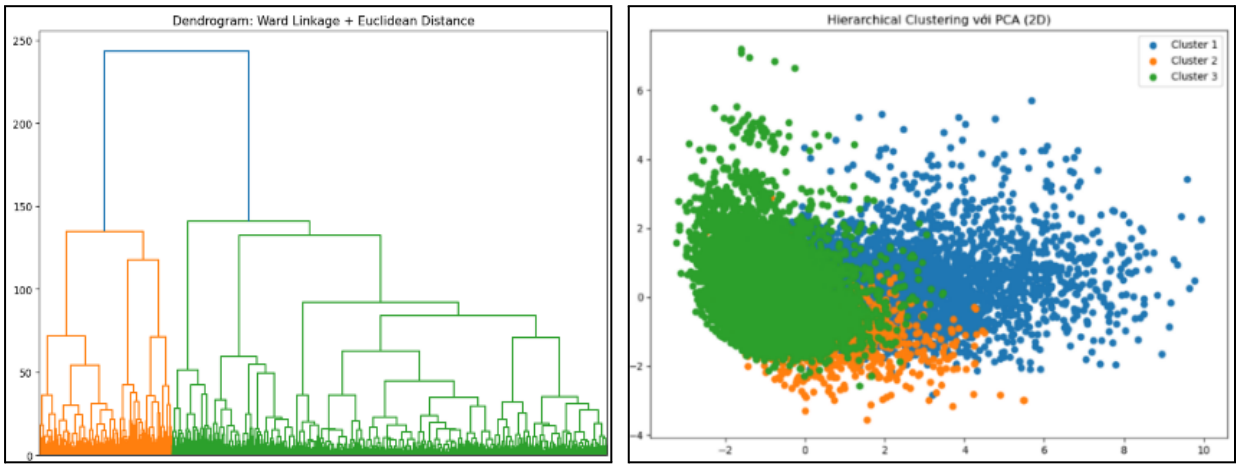



Figure 12: Clustering obtained Ward-Euclidean distance from the complete dataset

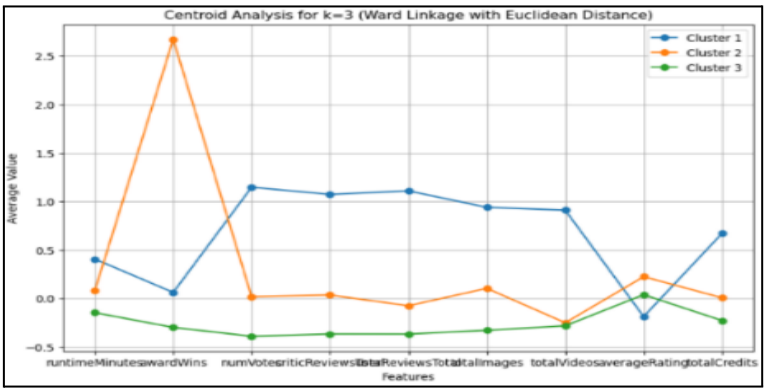



Figure 13: Centroid Analysis with Ward-Euclidean distance

These insights provide a clearer understanding of each cluster's distinctive attributes (figure 13) and their potential relevance to different audience groups or evaluation contexts.

### *2.5 Evaluation and comparison of clustering approaches*

- **K-Means Clustering** - Applied with k = 4. Data distribution showed challenges in clear separation across clusters. Centroid Analysis revealed ambiguous distinctions between clusters in several attributes. Achieved Silhouette Score = 0.2160.
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** - Applied with eps=4.84388 and MinPts=18. It formed only one cluster, which implies that the algorithm struggled to differentiate meaningful groupings in the dataset resulting in poor performance in this context.
- **Hierarchical Clustering** - Applied the Ward method with Euclidean distance to define 3 clusters. It achieved clear separations between clusters, as seen from the attributes. Outperformed the other methods in visualizing and understanding the data structure. Achieved Silhouette Score: 0.2636, better than K-means and DBSCAN.

In conclusion, considering the dataset's characteristics and Silhouette Scores, hierarchical clustering emerged as the most effective method. With a Silhouette Score of 0.2636, it demonstrated better-defined clusters and clearer separations, especially when analyzing relationships between attributes. Additionally, its Dendrogram visualization provided valuable insights into how clusters were formed and their connections, enhancing the interpretability and overall understanding of the data distribution. This makes hierarchical clustering the preferred approach for this analysis.

## 3. Classification

Classification is a supervised machine learning technique, where data is categorized into predefined classes or labels. In this section, we classify our dataset using two separate target variables: **titleType** and **averageRating** using **K-Nearest Neighbors (KNN), Decision Tree** and **Naive Bayes** methods. The test dataset is prepared consistently with the training set, including handling missing values, feature selection, and normalization. The target variable is evenly distributed across both sets. Model performance will be evaluated using **Accuracy**, **Precision**, **Recall**, **F1-score**, and the **ROC curve**, with average F1-scores and ROC curves calculated using Macro and Micro averaging methods.

### *3.1 K-Nearest Neighbor (KNN)*

We initiated classification with the K-Nearest Neighbors (KNN) algorithm, which labels points based on the majority vote of their K nearest neighbors, identified by distance. Using only numerical attributes as input (as in the clustering section, see *Section 2.1*), we focused on ***titleType*** and ***averageRating*** as the **target variables**. The numerical ***averageRating*** was discretized into three groups: **low** (0.5–<3.5), **medium** (3.5–<6.5), and **high** (6.5+).

#### *3.1.1 Hyper-Parameter Tuning*

To determine the optimal K, we applied ***Grid Search*** with K values ranging from 1 to 50 (step size 2, using odd numbers to avoid ties). We tested weights (***uniform***, ***distance***) and distance metrics (***manhattan***, ***euclidean***). The best parameters for

- ***titleType*** {'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}.
- ***averageRating*** {'metric': 'manhattan', 'n_neighbors': 47, 'weights': 'distance'}.

#### *3.1.2 Confusion Matrix and Evaluation*

For ***titleType,*** the KNN model achieved an overall accuracy of 0.82, (figure 14 left) performing well in multi-class classification. Popular classes like *movie* (91%), *short* (87%), *tvEpisode* (95%), and *tvSeries* (100%) had high recall and F1-scores due to clear features, large sample sizes, and low noise. For instance, *tvSeries* was easily distinguishable by the feature *canHaveEpisodes = 1*, while *tvEpisode* had stable *numVotes* and *averageRating*. Features like *runtimeMinutes* and *ratingCount* helped accurately

classify *movie* and *short*. However, from confusion matrix (figure 15 left) we can infer that *tvMovie* was often misclassified as *movie* (54%) or *tvEpisode* (20.7%), and *video* was frequently mistaken for *movie*, *short*, or *tvEpisode* (56.3%) due to overlapping features, data imbalance, and local similarity in the feature space. Misclassification was exacerbated by limitations in the feature set and possible data labeling errors. Classes with little data, such as *tvShort*, *tvMiniSeries*, and *tvSpecial*, showed poor performance. We also evaluated our model using the ROC curve (*Figure 16 left*). Through the ROC curve, we can identify some high-performing classes, with the top 3 being: *tvSeries* (area = 1.00), *short* (area = 0.98), and *tvEpisode* (area = 0.98). The Macro-average AUC is 0.88. These results are quite consistent with the F1-score results.

For ***averageRating***, the KNN model achieved an overall accuracy of 0.77 (figure 14 right), indicating a good performance but not yet truly excellent for a classification problem with only 3 classes. Furthermore, this overall test accuracy might also be inflated due to the dominance of the High class. The confusion matrix (figure 15 right) shows that the KNN model performs best on the *High* class, achieving 92% accuracy. The *Low* class has zero performance across all metrics, likely due to severe data imbalance (only 69 instances in the test set). The *Medium* class is frequently misclassified as *High* (61.91%) due to uneven class distribution. Although the *High* and *Medium* classes have similar proportions (44.02% and 44.91%), overlapping feature space boundaries lead to confusion. We also evaluated our model using the ROC curve (*Figure 16 right*). Through the ROC curve, we can identify some high-performing classes, with the top 3 being: *High* (area = 0.79), *Medium* (area = 0.78), and *Low* (area = 0.70). The Macro-average AUC is 0.76. These results are quite consistent with the F1-score results.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| movie | 0.83 | 0.91 | 0.87 | 1795 |
| short | 0.88 | 0.87 | 0.88 | 760 |
| tvEpisode | 0.83 | 0.95 | 0.88 | 1597 |
| tvMiniSeries | 0.75 | 0.08 | 0.14 | 75 |
| tvMovie | 0.50 | 0.19 | 0.28 | 299 |
| tvSeries | 0.86 | 1.00 | 0.92 | 409 |
| tvShort | 0.00 | 0.00 | 0.00 | 16 |
| tvSpecial | 0.38 | 0.12 | 0.19 | 48 |
| video | 0.57 | 0.36 | 0.44 | 222 |
| videoGame | 0.62 | 0.17 | 0.27 | 92 |
| accuracy | | | 0.82 | 5313 |
| macro avg | 0.62 | 0.46 | 0.49 | 5313 |
| weighted avg | 0.80 | 0.82 | 0.80 | 5313 |

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| High | 0.79 | 0.92 | 0.85 | 3819 |
| Low | 0.00 | 0.00 | 0.00 | 69 |
| Medium | 0.64 | 0.38 | 0.48 | 1425 |
| accuracy | | | 0.77 | 5313 |
| macro avg | 0.48 | 0.44 | 0.44 | 5313 |
| weighted avg | 0.74 | 0.77 | 0.74 | 5313 |

Figure 14: Classification Report for KNN – titleType (left), averageRating (right)

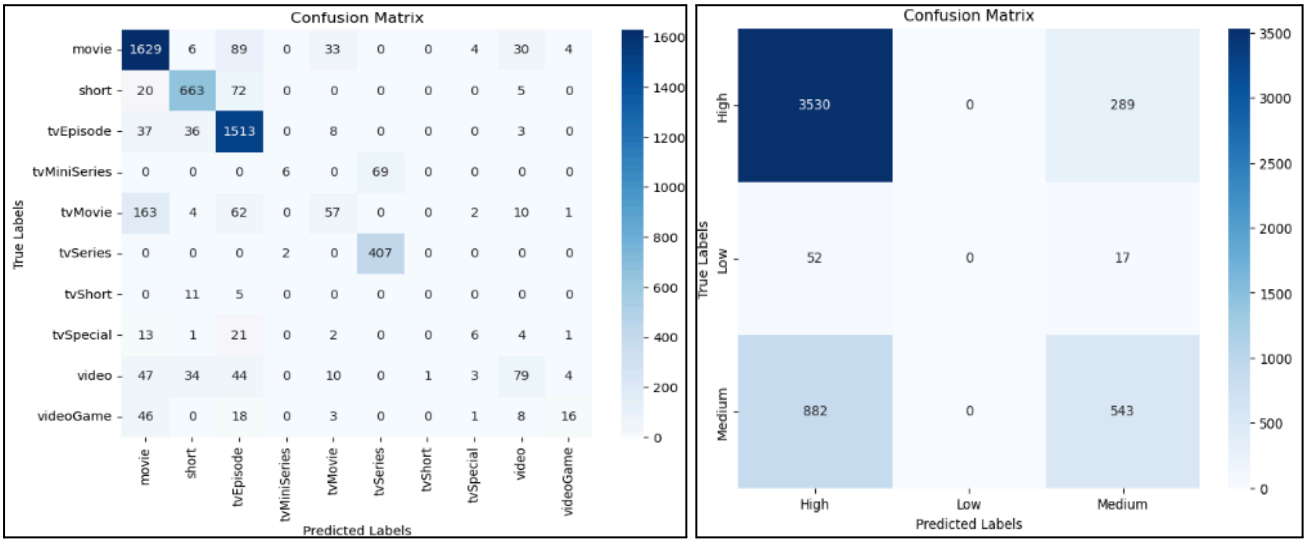



Figure 15: Confusion matrix for KNN – titleType (left), averageRating (right)

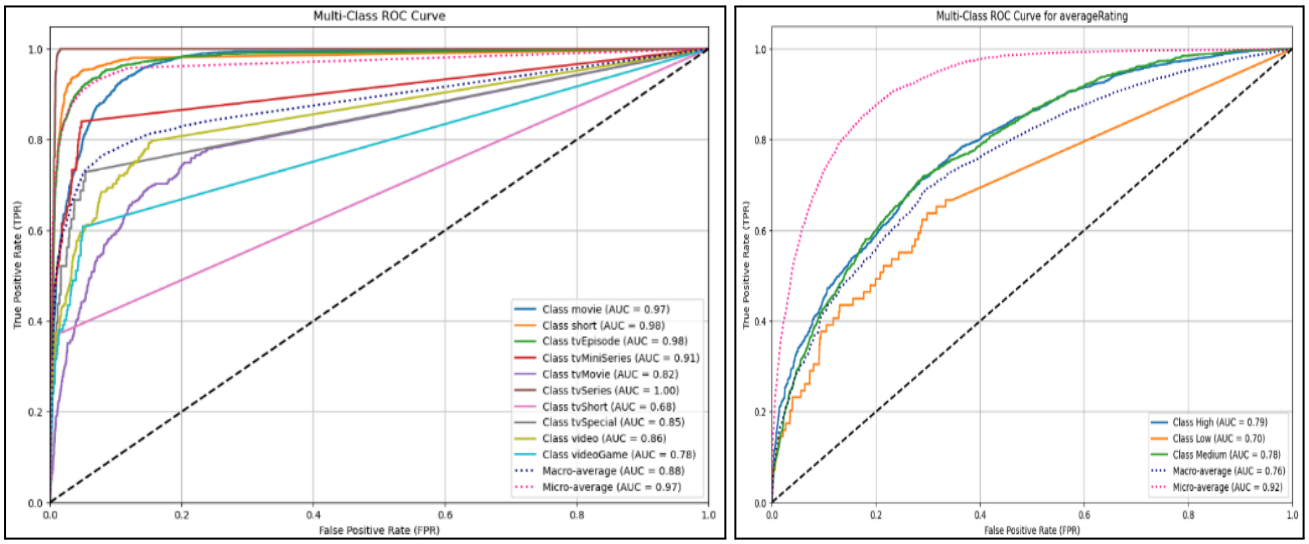



Figure 16: ROC curve for KNN – titleType (left), averageRating (right)

### *3.2 Decision Tree*

In this section, we implemented Decision Tree Classifier on two target variables: ***titleType*** and ***averageRating***. In which we have 10 distinct classes for ***titleType*** and for ***averageRating*** we divided into five classes to reduce the data imbalance good (4680), aboveAverage (4440), average (2717), veryGood (2336), and low (1765). We initially applied Decision Tree classifier with default parameters, followed by fine-tuning to improve accuracy. To address class imbalance, we used **SMOTE (Synthetic Minority Over-sampling Technique)** to generate synthetic samples for underrepresented classes, but this reduced accuracy, leading us to abandon the approach. We also attempted dimensionality reduction through **wrapper method with backward elimination**, removing irrelevant features but this failed to improve accuracy. After these techniques proved ineffective, we moved on to the next model.

#### *3.2.1 Hyper-Parameter Tuning*

To determine the optimal parameters, we applied ***Grid Search*** and **Randomized Search** with min_samples_split, min_samples_leaf, max_depth, criterion. The best parameters for

- ***titleType*** {'min_samples_split': 6, 'min_samples_leaf': 15, 'max_depth': None, 'criterion': 'gini'}.
- ***averageRating*** {'min_samples_split': 8, 'min_samples_leaf': 5, 'max_depth': 10, 'criterion': 'gini'}.

#### *3.2.2 Confusion Matrix and Evaluation*

For ***titleType****,* the Decision Tree Classifier achieved an overall accuracy of 0.78 (figure 17 left), performing well in multi-class classification. Popular classes like *movie* (91%), *short* (76%), *tvEpisode* (87%), and *tvSeries* (99%) had high recall and F1-scores due to clear features, large sample sizes, and low noise. Feature importance analysis (figure 18 left) identified *runtimeMinutes* and *canHaveEpisodes* as key features for distinguishing *titleType*, with *numRegions* and *genresEncoded* offering moderate insights. Features like *totalVideos*, *totalImages*, *isAdult*, and *userReviewsTotal* had low importance and were removed, but this did not improve accuracy. Then, we plotted confusion matrix (figure 19 left) where we compared the actual class with the predicted class. The model performs well for dominant classes such as *movie* and *tvEpisode*. For example, *movie* has 1638 correct predictions, and *tvEpisode* has 1377 correct predictions. Smaller classes such as *tvMiniSeries* and *tvShort* are often misclassified due to class imbalance or overlapping features with larger classes. We plotted a multi-class ROC curve (*Figure 20 left)*, with each curve representing a specific class and the diagonal line indicating random guessing (Area Under the Curve AUC = 0.5). AUC measures model performance for each class, with values closer to 1 indicating better discrimination. Classes like *tvSeries* (0.99), *movie* (0.95), and *tvEpisode* (0.95) showed excellent performance. *Short* (0.90), *tvMiniSeries* (0.91), and *video* (0.85) had high AUCs but room for improvement. Conversely, *videoGame* (0.61) and *tvSpecial* (0.70) had the lowest AUCs, highlighting difficulty in distinguishing these classes.

For **averageRating**, the Decision Tree Classifier initially achieved 30% accuracy, which improved to 36% (figure 17 right) after fine-tuning. All the classes performed poor in all the metrics with values ranging from 20-50% approx. Feature importance analysis (figure 18 right) revealed that the model's low accuracy was due to the lack of strong predictive features. Key features like startYear (0.15) and titleTypeEncoded (0.20) had limited impact, while others like runtimeMinutes (0.07), numVotes (0.07), and genresEncoded (0.08) contributed minimally. We removed the least important features (totalVideos, awardWins, awardNominationsExcludeWins, canHaveEpisodes, and isAdult). Overall, no feature strongly correlated with averageRating, leading to random model outputs. Then we plotted the confusion matrix (figure 19 right) showed that diagonal values represent correct predictions (true positives), while off-diagonal values indicate misclassifications. For instance, *aboveAverage* was often confused with *good* (548 times). The model performed best on classifying *good* (861

correct) but struggled with distinguishing adjacent ratings, like *aboveAverage* vs. *good*. The ROC curve (figure 20 right) showed varying performance across classes: aboveAverage (AUC = 0.58) had the lowest AUC, indicating poor separability; average (AUC = 0.64) and good (AUC = 0.66) showed moderate performance; low and veryGood (AUC = 0.71) performed the best, but still not ideal.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| movie | 0.78 | 0.91 | 0.84 | 1795 |
| short | 0.77 | 0.76 | 0.76 | 760 |
| tvEpisode | 0.86 | 0.87 | 0.87 | 1597 |
| tvMiniSeries | 0.62 | 0.11 | 0.18 | 75 |
| tvMovie | 0.27 | 0.20 | 0.23 | 299 |
| tvSeries | 0.86 | 0.99 | 0.92 | 409 |
| tvShort | 0.00 | 0.00 | 0.00 | 16 |
| tvSpecial | 0.09 | 0.04 | 0.06 | 48 |
| video | 0.60 | 0.33 | 0.42 | 222 |
| videoGame | 0.29 | 0.02 | 0.04 | 92 |
| accuracy | | | 0.78 | 5313 |
| macro avg | 0.51 | 0.42 | 0.43 | 5313 |
| weighted avg | 0.75 | 0.78 | 0.76 | 5313 |

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| aboveAverage | 0.34 | 0.38 | 0.36 | 1474 |
| average | 0.31 | 0.23 | 0.26 | 909 |
| good | 0.40 | 0.55 | 0.46 | 1569 |
| low | 0.37 | 0.25 | 0.29 | 585 |
| veryGood | 0.34 | 0.20 | 0.25 | 776 |
| accuracy | | | 0.36 | 5313 |
| macro avg | 0.35 | 0.32 | 0.33 | 5313 |
| weighted avg | 0.36 | 0.36 | 0.35 | 5313 |

Figure 17: Classification Report for Decision Tree Classifier – titleType (left), averageRating (right)

```
startYear 0.04599606159168468
runtimeMinutes 0.4754596380236425
awardWins 0.005192856059878699
numVotes 0.0076966384208236064
totalImages 0.03100030929964906
totalVideos 0.0033627173202763296
totalCredits 0.05140660231565569
criticReviewsTotal 0.0038955394874972643
awardNominationsExcludeWins 0.004874342977565126
canHaveEpisodes 0.17145821370155473
isAdult 0.01419491648150638
numRegions 0.06742804837209804
userReviewsTotal 0.0022239628301747474
ratingCount 0.012092694355897162
averageRating 0.010610969843714745
genresEncoded 0.05739929987844014
countryEncoded 0.035707189039941016
```

```
startYear 0.14976391661877594
runtimeMinutes 0.07389443601902743
awardWins 0.027715721862017484
numVotes 0.07344278760586274
totalImages 0.02843777770249299
totalVideos 0.0
totalCredits 0.07540998789727474
criticReviewsTotal 0.030783035707570378
awardNominationsExcludeWins 0.013958486770755289
canHaveEpisodes 0.0068077577016739345
isAdult 0.008481383593827056
numRegions 0.0384764818211185
userReviewsTotal 0.03605983304498102
ratingCount 0.08515761711662198
titleTypeEncoded 0.2054404790599914
genresEncoded 0.07933713900492834
countryEncoded 0.06683315847308076
```

Figure 18: Feature Importance for Decision Tree Classifier – titleType (left), averageRating (right)

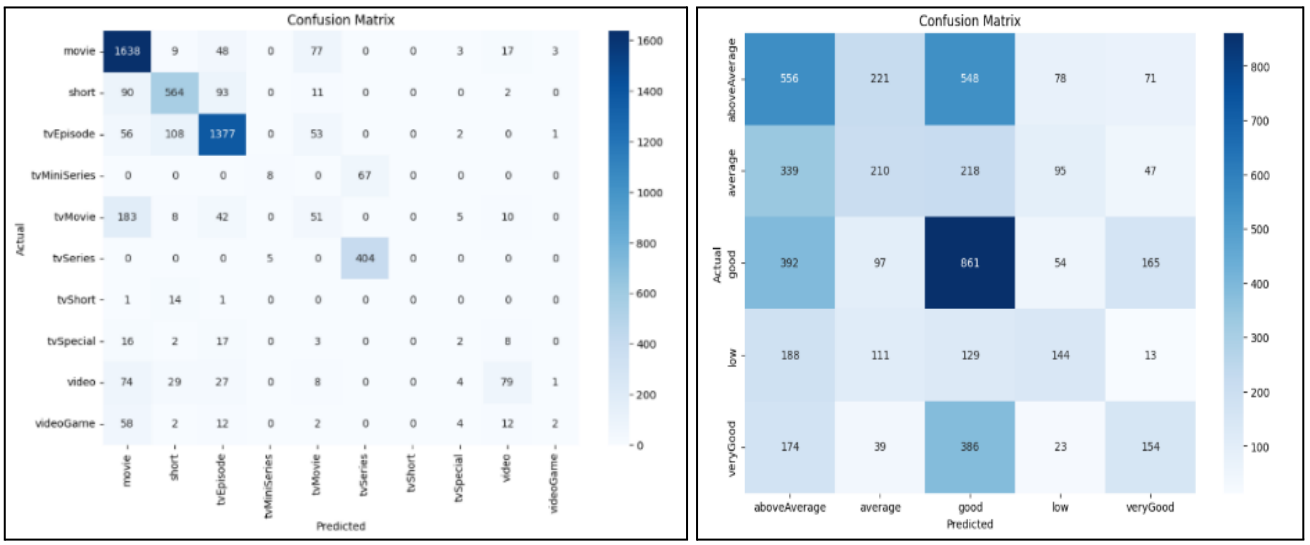



Figure 19: Confusion Matrix for Decision Tree Classifier – titleType (left), averageRating (right)

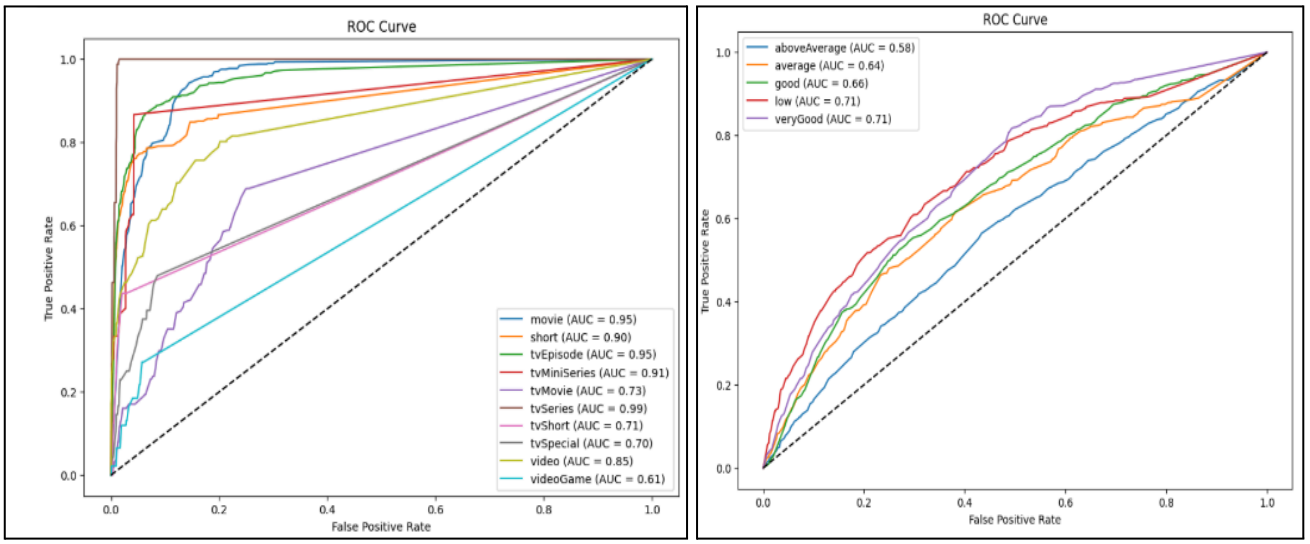



Figure 20: ROC curve for DecisionTree Classifier – titleType (left), averageRating (right)

### 3.3 Naïve Bayes

In this section, we implemented Gaussian Naive Bayes Classifier as the same approach as Decision Tree Classifier on two target variables: titleType and averageRating. We initially applied Naive Bayes classifier with default parameters, followed by fine-tuning to improve accuracy. To address class imbalance, we used SMOTE to generate synthetic samples for underrepresented classes, but this reduced accuracy, leading us to abandon the approach. We also attempted dimensionality reduction through wrapper method with backward elimination, removing irrelevant features but this failed to improve accuracy. After these techniques proved ineffective, we stopped our experimentation.

#### *3.3.1 Hyper-Parameter Tuning*

To determine the optimal parameters, we applied Grid Search and Randomized Search with var_smoothing and priors. The best parameters for

- **titleType** {'var_smoothing': 0.18420699693267165, 'priors': None'}.
- **averageRating** {'var_smoothing': 0.022229964825261957, 'priors': [0.3, 0.3, 0.3, 0.05, 0.05]}.

#### *3.3.2 Confusion Matrix and Evaluation*

For **titleType**, the GNB Classifier achieved an overall accuracy of 0.71, (figure 21 left) performing well in multi-class classification. Classes like movie, tvSeries, tvEpisodes, short had higher precision, recall and F1-score. Feature importance analysis (figure 22 left) highlighted that averageRating, startYear, canHaveEpisodes are key features in performing the classification. Then we plotted the confusion matrix (Figure 23 left) which revealed that the model performed strongest in classifying tvEpisodes (1,463 correct predictions) and tvSeries (408 correct). However, the visualization showed significant confusion between movies and tvEpisodes, with 719 movies being misclassified as tvEpisodes. The model struggled particularly with distinguishing between similar content types, demonstrating a tendency to over-predict the tvEpisode category across multiple classes. We plotted a multi-class ROC curve (figure 24 left) showing classification performance across categories. tvMiniSeries demonstrated excellent performance (AUC = 0.98), followed by videoGame and movie (both AUC = 0.90). tvMovie and tvSeries achieved good results (AUC = 0.73 and 0.85 respectively), while tvShort showed lower discrimination ability (AUC = 0.63). All categories performed above the random classification baseline (AUC = 0.5).

For **averageRating**, the GNB classifier initially performed poorly with default parameters, achieving 0.18 accuracy and low precision, recall, and F1-scores. Fine-tuning via GridSearchCV identified optimal parameters (var_smoothing: 1.0 and priors: [0.3, 0.3, 0.3, 0.05, 0.05]), improving accuracy to 0.33 (figure 21 right). Feature importance analysis (figure 22 right) highlighted titleTypeEncoded (24.35), startYear (22.53), and runtimeMinutes (12.97) as key features, leading to the removal of less impactful ones like isAdult, awardWins, and totalVideos to refine the model. The confusion matrix (figure 23 right) shows the model performed best at predicting good ratings (714 correct predictions), but frequently misclassified good as veryGood (437 cases) and aboveAverage as good (428 cases). The model shows particular difficulty distinguishing between adjacent rating categories, suggesting challenges in separating similar rating levels. The ROC curves (figure 24 right) show varying classification performance across rating categories. The veryGood class achieved the best performance (AUC = 0.68), followed

by low (AUC = 0.66). Average and good ratings showed identical performance (AUC = 0.64), while aboveAverage performed closest to random classification (AUC = 0.53). All categories except aboveAverage performed notably above the random baseline (AUC = 0.5).

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| movie | 0.75 | 0.82 | 0.78 | 1795 |
| short | 0.77 | 0.71 | 0.74 | 760 |
| tvEpisode | 0.64 | 0.81 | 0.72 | 1597 |
| tvMiniSeries | 0.00 | 0.00 | 0.00 | 75 |
| tvMovie | 0.00 | 0.00 | 0.00 | 299 |
| tvSeries | 0.85 | 1.00 | 0.92 | 409 |
| tvShort | 0.00 | 0.00 | 0.00 | 16 |
| tvSpecial | 0.00 | 0.00 | 0.00 | 48 |
| video | 0.28 | 0.22 | 0.25 | 222 |
| videoGame | 0.57 | 0.04 | 0.08 | 92 |
| accuracy | | | 0.71 | 5313 |
| macro avg | 0.39 | 0.36 | 0.35 | 5313 |
| weighted avg | 0.64 | 0.71 | 0.67 | 5313 |

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| aboveAverage | 0.31 | 0.33 | 0.32 | 1474 |
| average | 0.33 | 0.10 | 0.15 | 909 |
| good | 0.33 | 0.69 | 0.45 | 1569 |
| low | 0.35 | 0.09 | 0.14 | 585 |
| veryGood | 0.34 | 0.04 | 0.07 | 776 |
| accuracy | | | 0.33 | 5313 |
| macro avg | 0.33 | 0.25 | 0.23 | 5313 |
| weighted avg | 0.33 | 0.33 | 0.27 | 5313 |

Figure 21: Classification Report for GNB Classifier – titleType (left), averageRating (right)

| | Feature | Importance |
|---|---|---|
| 14 | averageRating | 28.401255 |
| 0 | startYear | 23.653186 |
| 9 | canHaveEpisodes | 13.303399 |
| 16 | countryEncoded | 12.442802 |
| 1 | runtimeMinutes | 10.830986 |
| 15 | genresEncoded | 9.967168 |
| 6 | totalCredits | 6.541357 |
| 13 | ratingCount | 1.185008 |
| 3 | numVotes | 1.183181 |
| 4 | totalImages | 0.920751 |
| 11 | numRegions | 0.358483 |
| 12 | userReviewsTotal | 0.258252 |
| 7 | criticReviewsTotal | 0.120001 |
| 8 | awardNominationsExcludeWins | 0.056585 |
| 2 | awardWins | 0.043123 |
| 10 | isAdult | 0.041752 |
| 5 | totalVideos | 0.032454 |

| | Feature | Importance |
|---|---|---|
| 14 | titleTypeEncoded | 24.346148 |
| 0 | startYear | 22.526105 |
| 1 | runtimeMinutes | 12.965715 |
| 16 | countryEncoded | 12.953167 |
| 15 | genresEncoded | 10.589019 |
| 6 | totalCredits | 9.530245 |
| 13 | ratingCount | 1.789553 |
| 3 | numVotes | 1.787286 |
| 4 | totalImages | 1.219857 |
| 11 | numRegions | 0.473588 |
| 12 | userReviewsTotal | 0.437980 |
| 7 | criticReviewsTotal | 0.270271 |
| 9 | canHaveEpisodes | 0.098500 |
| 8 | awardNominationsExcludeWins | 0.083155 |
| 2 | awardWins | 0.077823 |
| 5 | totalVideos | 0.069924 |
| 10 | isAdult | 0.018387 |

Figure 22: Feature Importance for GNB Classifier – titleType (left), averageRating (right)

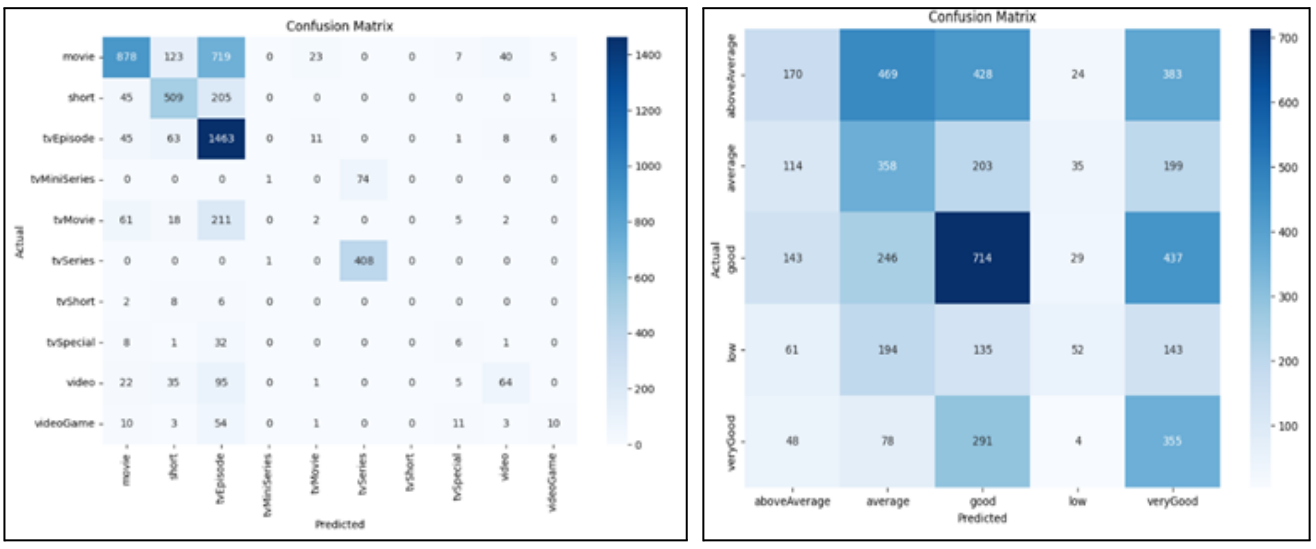



Figure 23: Confusion Matrix for GNB Classifier – titleType (left), averageRating (right)

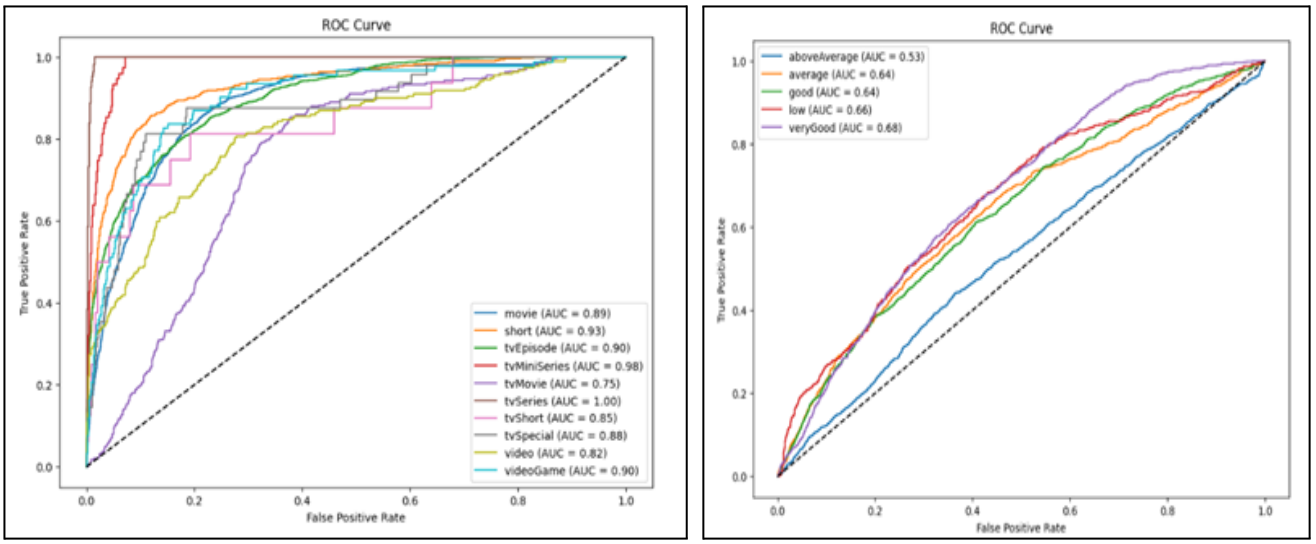



Figure 24: ROC curve for GNB Classifier – titleType (left), averageRating (right)

## 4. Pattern Mining and Regression

### *4.1 Pattern Mining*

This section explored association rules to uncover hidden patterns, focusing on *averageRating* and *titleType*. Frequent patterns were extracted with varying support thresholds, and meaningful rules were derived to predict the target variable and applied to the test dataset.

#### *4.1.1. Attribute Selection and Binning*

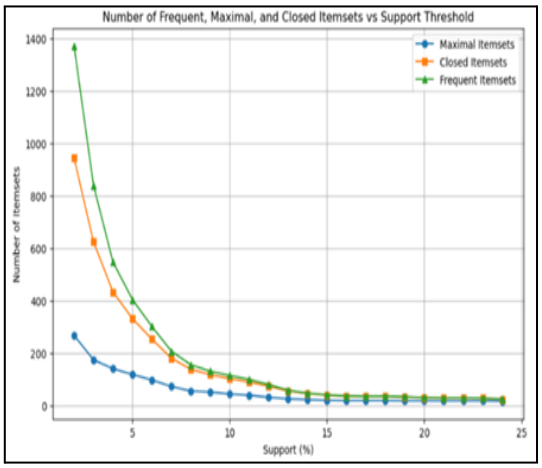
Firstly, we selected the most informative attributes, which include *runtimeMinutes*, *numVotes*, *totalImages*, *totalCredits*, *ratingCount*, *averageRating*, and *titleType*, primarily for the following reasons:

- **Representativeness and Discriminative Power**: These attributes represent core characteristics of the content, helping to identify patterns related to quality, popularity, or scale.
- **Ease of Processing and Mining**: These attributes all have clear numerical or categorical values, making them suitable for pattern mining algorithms.
- **Avoiding Redundancy:** We did not select attributes like awardWins, numRegions, or genresEncoded to reduce complexity and data duplication.
- **Interpretability:** These attributes have clear meanings, making the implementation of association rules more practical. For example, "Movies with a runtime longer than 120 minutes and an average rating above 7.0 tend to have higher overall ratings."
- **Data Completeness and Balance:** The remaining attributes exhibited imbalance within the dataset.

The dataset, derived from the Data Understanding phase, consisted of 15,939 training rows. The **IQR** method was used to group *runtimeMinutes*, *numVotes*, *totalImages*, *totalCredits*, and *ratingCount* into balanced data quantities. The original data for titleType was retained, while averageRating was divided into three groups (*High*, *Low*, *Medium*), following the approach from the **KNN** section.

#### *4.1.2 Frequent Items*

The extraction of frequent patterns and association rules was performed using the Apriori algorithm. After some testing, we decided to consider only itemsets with a minimum length of 2 elements to focus on the most significant ones. *Figure 25* shows the distribution of the number of extracted itemsets with three types of frequent items (frequent, maximal, closed). As expected, increasing the support value leads to a decrease in the resulting number of itemsets.




| support | itemsets |
|---|---|
| 0.271113 | ((-0.001, 0.107]_Ratings, (-0.001, 0.107]_Votes) |
| 0.193814 | ((-0.001, 0.107]_Ratings, (-0.001, 0.117]_Images) |
| 0.194002 | ((-0.001, 0.107]_Votes, (-0.001, 0.117]_Images) |
| 0.182457 | ((-0.001, 0.415]_Credits, (-0.001, 0.117]_Images) |
| 0.150897 | ((-0.001, 0.545]_Runtime, (-0.001, 0.117]_Images) |
| 0.228385 | (High, (-0.001, 0.117]_Images) |
| 0.259066 | (Medium, (-0.001, 0.117]_Images) |
| 0.182018 | ((-0.001, 0.117]_Images, tvEpisode) |
| 0.232150 | ((0.107, 0.195]_Ratings, (0.107, 0.195]_Votes) |
| 0.246706 | ((0.195, 0.341]_Votes, (0.195, 0.34]_Ratings) |
| 0.249090 | ((0.341, 1.0]_Votes, (0.34, 1.0]_Ratings) |
| 0.152968 | (High, (0.545, 0.675]_Runtime) |
| 0.151964 | ((0.675, 0.747]_Runtime, Medium) |
| 0.162191 | (movie, (0.747, 1.0]_Runtime) |
| 0.204417 | (High, tvEpisode) |
| 0.231459 | (movie, Medium) |
| 0.193751 | ((-0.001, 0.107]_Ratings, (-0.001, 0.107]_Vote... |

Figure 25: Distribution of the number of extracted itemsets

Initially, we set a support threshold of 20%, resulting in frequent itemsets that were both closed and maximal due to the dataset's characteristics and the high threshold. To enhance pattern discovery, we reduced the support threshold to 15%. Table 34 summarizes the dataset patterns. Using both FP-Growth and Apriori methods yielded identical results, but with no significant speed advantage for FP-Growth, likely due to the dataset's modest size.

### *4.1.3 Association Rules*

After generating the frequent itemsets, we decided to extract rules with a minimum support value of 15% and a minimum confidence of 60%, leading to *Table 6*. These rules provide concise insights into the relationships between features, aiding decision-making and the potential refinement of predictive models.

| Antecedents | Consequents | Support | Confidence | Lift | Conviction |
|---|---|---|---|---|---|
| (0.107, 0.195]_Ratings | (0.107, 0.195]_Votes | 0.23215 | 0.997843 | 4.293632 | 355.7822 |
| (0.107, 0.195]_Votes | (0.107, 0.195]_Ratings | 0.23215 | 0.99892 | 4.293632 | 710.564 |
| (0.195, 0.341]_Votes | (0.195, 0.34]_Ratings | 0.246706 | 0.998984 | 4.046201 | 741.0554 |
| (0.195, 0.34]_Ratings | (0.195, 0.341]_Votes | 0.246706 | 0.999238 | 4.046201 | 987.7402 |
| (0.341, 1.0]_Votes | (0.34, 1.0]_Ratings | 0.24909 | 1 | 4.011578 | 7.51E+08 |
| (0.34, 1.0]_Ratings | (0.341, 1.0]_Votes | 0.24909 | 0.999245 | 4.011578 | 994.4535 |
| (-0.001, 0.107]_Votes, (-0.001, 0.117]_Images | (-0.001, 0.107]_Ratings | 0.193751 | 0.998706 | 3.682874 | 563.3807 |
| (-0.001, 0.107]_Ratings | (-0.001, 0.107]_Votes, (-0.001, 0.117]_Images | 0.193751 | 0.714484 | 3.682874 | 2.822953 |
| (-0.001, 0.107]_Ratings | (-0.001, 0.107]_Votes | 0.271113 | 0.999769 | 3.68168 | 3148.338 |
| (-0.001, 0.107]_Votes | (-0.001, 0.107]_Ratings | 0.271113 | 0.998383 | 3.68168 | 450.6213 |
| (-0.001, 0.107]_Ratings, (-0.001, 0.117]_Images | (-0.001, 0.107]_Votes | 0.193751 | 0.999676 | 3.68134 | 2250.168 |
| (-0.001, 0.107]_Votes | (-0.001, 0.107]_Ratings, (-0.001, 0.117]_Images | 0.193751 | 0.713494 | 3.68134 | 2.813851 |
| (0.747, 1.0]_Runtime | movie | 0.162191 | 0.667959 | 2.016656 | 2.014144 |

| | | | | | |
|---|---|---|---|---|---|
| tvEpisode | High | 0.204417 | 0.694078 | 1.576712 | 1.829857 |
| (-0.001, 0.107]_Ratings | (-0.001, 0.117]_Images | 0.193814 | 0.714715 | 1.407703 | 1.725584 |
| (-0.001, 0.107]_Ratings, (-0.001, 0.107]_Votes | (-0.001, 0.117]_Images | 0.193751 | 0.714649 | 1.407573 | 1.725185 |
| (-0.001, 0.107]_Votes | (-0.001, 0.117]_Images | 0.194002 | 0.714418 | 1.407117 | 1.723786 |
| (0.545, 0.675]_Runtime | High | 0.152968 | 0.615035 | 1.397154 | 1.454144 |
| (-0.001, 0.415]_Credits | (-0.001, 0.117]_Images | 0.182457 | 0.694531 | 1.367947 | 1.611562 |
| movie | Medium | 0.231459 | 0.698807 | 1.344956 | 1.595069 |
| tvEpisode | (-0.001, 0.117]_Images | 0.182018 | 0.618023 | 1.217258 | 1.288776 |

Table 6: Association Rules

### *4.1.4 Target Variable Prediction*

We will apply the following rule to the test set: tvEpisode => High (support 0.204417, confidence 0.694078). Using the rule, we designated the "averageRating" attribute as the target variable, with the antecedent being "tvEpisode". The confusion matrix is described in Figure 26, and the accuracy is reported as 91%.

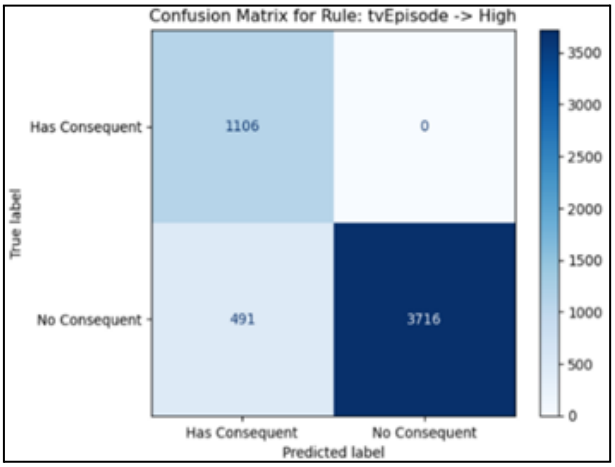



Figure 26: Confusion Matrix for Rule tvEpisode => High

### *4.2 Regression*

Regression is a supervised machine learning algorithm that predicts continuous variables, It aims to establish a relationship between the dependent variable and independent variables. We began our analysis by selecting three attributes as target variables:

- **averageRating** - Predicting whether a movie is likely to receive a high rating based on its attributes.
- **numVotes** - Estimating the level of attention a movie may receive.
- **totalCredit** - Examining the size of the production team.

The test dataset is prepared consistently with the training set, including handling missing values, feature selection, and normalization. The target variable is evenly distributed across both sets. Initially, we implemented **Multiple Linear Regression** to model the relationships. However, the model's performance was suboptimal, then we transitioned to regularized models like **Ridge** and **Lasso Regression**, which help address multicollinearity and overfitting. To capture potential non-linear relationships, we further experimented with non-linear models, including **Decision Tree Regressor** and **K-Nearest Neighbors (KNN) Regressor**. To optimize all models, we fine-tuned their hyperparameters using **GridSearchCV** and **RandomizedSearchCV**. Additionally, we analyzed **feature importance**, identified less influential features, and removed them

to streamline the models. The model performance was evaluated using error metrics such as **Root Mean Squared Error (RMSE)** and **$R^2$ score**.

To delve deeper we plotted the scatter plot of prediction results for all implemented models. It highlights that all models produce nearly similar outputs, (figure 27) but their performance varies depending on the target variable. For **averageRating**, the models frequently misclassified the ratings in the range of **5 to 8**, indicating difficulty in accurately predicting values within this range. In contrast, the predictions for **numVotes** showed an ideal fit, aligning well with the actual data points. However, predictions for **totalCredits** displayed notable discrepancies, often deviating significantly from the actual values.

Next, we utilized a **Decision Tree Regressor** to model the relationships (figure 28). For **averageRating**, the decision tree identified the root node as the **titleTypeEncoded** feature, suggesting this feature had the highest information gain for splitting the data. Similarly, for **numVotes**, the root node was determined to be **userReviewsTotal**, indicating a strong correlation between the number of user reviews and the number of votes a movie receives.

In addition, we analyzed the **feature importance** for all models to understand which features contributed the most to the predictions. For instance, (figure 29) displays the feature importance for the **KNN Regressor**, which was fine-tuned to capture non-linear relationships between variables.

Finally, **Table 7** summarizes the performance of all models using the evaluation metrics **RMSE** (Root Mean Squared Error) and **$R^2$ score**. Among all models, the **KNN Regressor** emerged as the best performer, achieving the highest **$R^2$ score**, which indicates its superior ability to explain the variance in the data. The underlying reason for the KNN Regressor's success can be attributed to its ability to capture localized patterns and non-linear relationships in the data, which other models like linear regression struggled to model effectively.

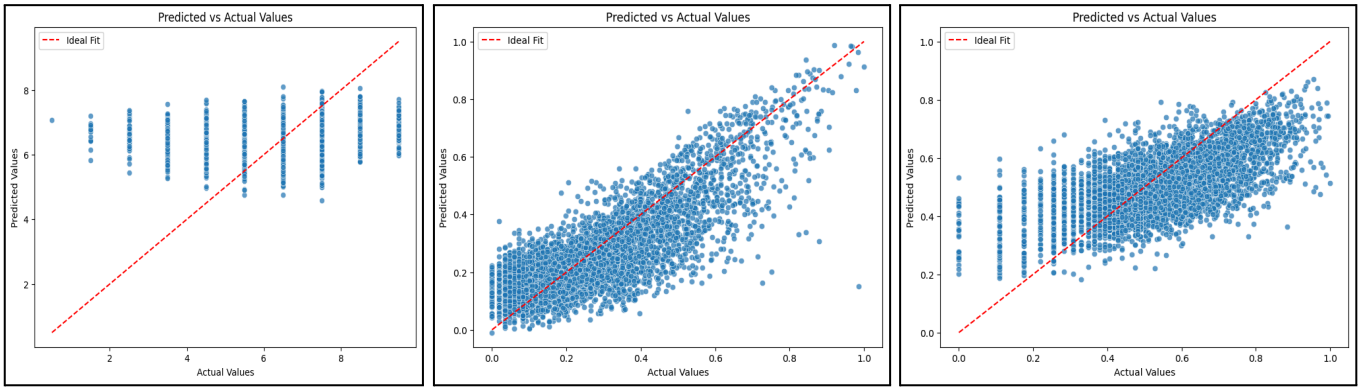



Figure 27: Predicted vs Actual Values and the ideal fit of Linear Regression

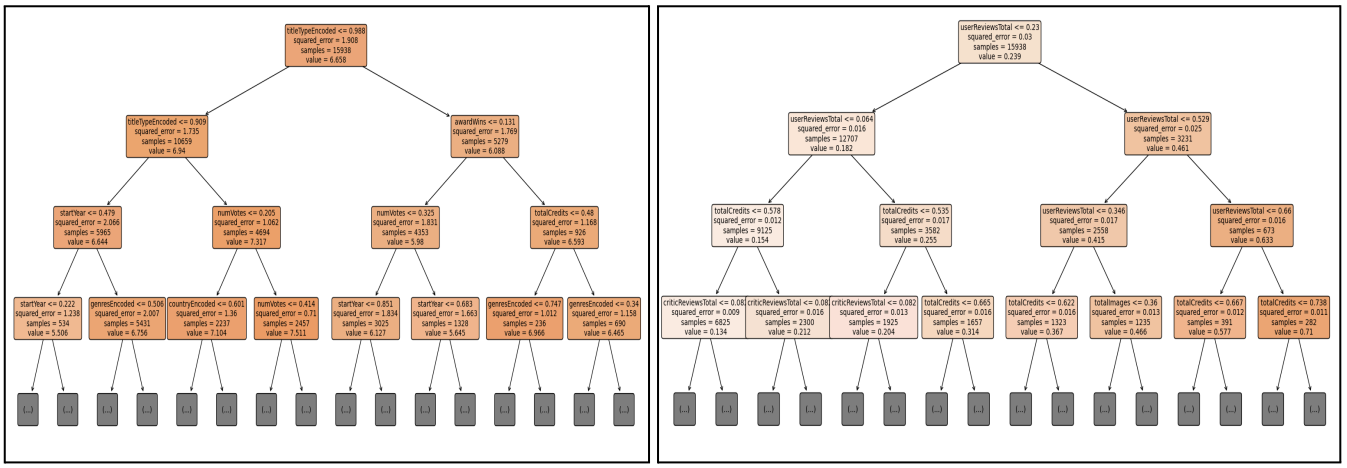


Figure 28: Decision Tree

Feature Importance for Predicting 'averageRating':

| | Feature | Importance |
|---|---|---|
| 0 | startYear | 0.181482 |
| 11 | numRegions | 0.165500 |
| 15 | countryEncoded | 0.135703 |
| 1 | runtimeMinutes | 0.083638 |
| 13 | titleTypeEncoded | 0.075309 |
| 14 | genresEncoded | 0.072067 |
| 9 | canHaveEpisodes | 0.056164 |
| 3 | numVotes | 0.051158 |
| 6 | totalCredits | 0.038258 |
| 4 | totalImages | 0.033567 |
| 12 | userReviewsTotal | 0.032274 |
| 7 | criticReviewsTotal | 0.023631 |
| 2 | awardWins | 0.018789 |
| 10 | isAdult | 0.015809 |
| 8 | awardNominationsExcludeWins | 0.011071 |
| 5 | totalVideos | 0.002533 |

Feature Importance for Predicting 'numVotes':

| | Feature | Importance |
|---|---|---|
| 11 | userReviewsTotal | 0.006729 |
| 6 | criticReviewsTotal | 0.002908 |
| 10 | numRegions | 0.002372 |
| 3 | totalImages | 0.002194 |
| 5 | totalCredits | 0.001662 |
| 15 | countryEncoded | 0.001437 |
| 0 | startYear | 0.001208 |
| 14 | genresEncoded | 0.000926 |
| 8 | canHaveEpisodes | 0.000798 |
| 1 | runtimeMinutes | 0.000656 |
| 13 | titleTypeEncoded | 0.000614 |
| 12 | averageRating | 0.000389 |
| 4 | totalVideos | 0.000252 |
| 2 | awardWins | 0.000146 |
| 7 | awardNominationsExcludeWins | 0.000143 |
| 9 | isAdult | 0.000131 |

Feature Importance for Predicting 'totalCredits':

| | Feature | Importance |
|---|---|---|
| 0 | startYear | 0.003291 |
| 3 | numVotes | 0.002054 |
| 13 | titleTypeEncoded | 0.001814 |
| 8 | canHaveEpisodes | 0.001746 |
| 15 | countryEncoded | 0.001722 |
| 14 | genresEncoded | 0.001546 |
| 1 | runtimeMinutes | 0.001354 |
| 10 | numRegions | 0.001116 |
| 4 | totalImages | 0.000840 |
| 11 | userReviewsTotal | 0.000538 |
| 6 | criticReviewsTotal | 0.000295 |
| 7 | awardNominationsExcludeWins | 0.000265 |
| 12 | averageRating | 0.000243 |
| 9 | isAdult | 0.000231 |
| 5 | totalVideos | 0.000150 |
| 2 | awardWins | 0.000097 |

Figure 29: Feature Importance of KNN regressor

| | averageRating | | numVotes | | totalCredits | |
|---|---|---|---|---|---|---|
| Model | RMSE | R2 score | RMSE | R2 score | RMSE | R2 score |
| Linear | 1.308 | 0.09 | 0.109 | 0.66 | 0.13 | 0.33 |
| Ridge | 1.309 | 0.09 | 0.109 | 0.66 | 0.13 | 0.33 |
| Lasso | 1.378 | 0.09 | 0.11 | 0.65 | 0.13 | 0.32 |
| Decision Tree | 1.25 | 0.17 | 0.11 | 0.64 | 0.13 | 0.32 |
| KNN | 1.19 | 0.24 | 0.101 | 0.71 | 0.12 | 0.45 |

Table 7: Error Metric

To conclude, our comparative analysis demonstrates that while simpler models like Linear Regression and regularized models like Ridge and Lasso fail to account for non-linear patterns, tree-based models and KNN provide a significant improvement. By identifying the right features, optimizing hyperparameters, and leveraging algorithms capable of capturing complex relationships, we successfully improved the predictive performance of the models, with KNN standing out as the most robust approach.