UNIVERSITÀ DI PISA

Department of Computer Science

Master degree in Data Science

Data Mining: Foundations: Project report

IMDb Dataset

Di Prizio Emily
Pietriccioli Alice

Academic year 2024/2025

Contents

1 Data Understanding and Preparation **1**
- 1.0 Dataset . . . 1
- 1.1 Data Semantics and Distribution of Variables Statistics . . . 1
- 1.2 Assessing data quality . . . 3
  - 1.2.1 Missing Values . . . 3
  - 1.2.2 Outliers . . . 4
  - 1.2.3 Duplicates . . . 5
- 1.3 Pairwise correlations and eventual elimination of variables . . . 5

2 Clustering **6**
- 2.0 Normalization . . . 6
- 2.1 Analysis by Centroid-based Methods: K-means . . . 6
  - 2.1.1 Bisecting K-means . . . 7
  - 2.1.2 X-means . . . 7
  - 2.1.3 K-mode . . . 7
- 2.2 Analysis by density-based clustering: DBSCAN . . . 8
  - 2.2.1 OPTICS . . . 9
- 2.3 Analysis by hierarchical clustering: Agglomerative Clustering . . . 9
- 2.4 Final Discussion . . . 11

3 Classification **12**
- 3.1 K-Nearest Neighbors . . . 12
  - 3.1.1 Rating Prediction . . . 12
  - 3.1.2 TitleType Prediction . . . 13
- 3.2 Naive Bayes . . . 13
  - 3.2.1 Rating Prediction . . . 14
  - 3.2.2 TitleType Prediction . . . 14
- 3.3 Decision Tree . . . 15
  - 3.3.1 Rating Prediction . . . 15
  - 3.3.2 HasAwards Prediction . . . 16
- 3.4 Final Discussion . . . 17

4 Pattern Mining and Regression **18**
- 4.1 Regression . . . 18
  - 4.1.1 Simple Linear Regression . . . 18
  - 4.1.2 Simple Non Linear Regression . . . 18
  - 4.1.3 Multiple Regression . . . 19
  - 4.1.4 Multivariate Regression . . . 19
  - 4.1.5 Final Discussion . . . 20
- 4.2 Pattern Mining . . . 20
  - 4.2.1 Apriori . . . 20
  - 4.2.2 FP-Growth . . . 21
  - 4.2.3 Association Rules . . . 21

1 Data Understanding and Preparation

1.0 Dataset

The *IMDb Dataset* contains information about movies, TV shows, and other forms of visual entertainment, along with their ratings, which are generated by the online community.
Each record includes key details such as the original title, release year, runtime, and the number of user and votes. Additionally, the dataset offers insights into important aspects like awards, nominations, and user reviews, as well as statistical ratings, ranging from the best and worst ratings to the total number of critic reviews. It also encompasses metadata such as the country of origin, the number of images and videos, and the title's genre. The dataset is updated as of September 1, 2024.

1.1 Data Semantics and Distribution of Variables Statistics

The dataset consists of 23 features describing 16431 records, with each row representing a unique entry. The variables' most important features are described below:

- **Categorical**: `originalTitle`; `rating` which is an ordinal one (ten classes) represents the scale of liking; `worstRating` represents the lowest rating on a scale from 1 to 10; `beastRating` represents the highest rating on a scale from 1 to 10; `titleType` represents different categories of titles such as 'tvEpisode', 'videoGame', 'movie', 'tvSeries', etc.; `canHaveEpisodes` binary, represents whether the title can have episodes, True or False; `isRatable` binary, represents whether the title is ratable, True or False; `isAdult` binary, represents whether the title is for adults, with 0 indicating no and 1 indicating yes; `countryOfOrigin` represents a list of countries; `genres` represents a list of genres.

- **Numerical**: `startYear`; `endYear`; `runtimeMinutes`; `awardWins`; `numVotes`; `totalImages`; `totalVideos`; `totalCredits`; `criticReviewsTotal`; `awardNominationsExcludeWins`; `numRegions`; `userReviewsTotal`; `ratingCount`.

In this dataset, some variables have fixed values that do not vary. In particular, `worstRating` always takes the value of 1, `beastRating` is always set to 10, and `isRatable` is always True.
Looking at the dataset, it is evident also that some variables exhibit a strong imbalance in their values. In particular, we noticed a highly imbalanced distribution in certain columns, such as the binary variable `isAdult` and the discrete variable `awardWins`. Specifically, `isAdult` has a predominance of values equal to 0 (97.41%), indicating that the majority of titles in the dataset are not classified as adult content, with only a small percentage of titles falling into this category. Similarly, the `awardWins` variable is characterized by a strong imbalance, as most titles have not won any awards (72.86%), with only a limited number of titles having actually received recognition. Similarly, the `totalVideos` variable has a value of 0 in 90.20% of the cases, whereas `totalImages` is only 20.02%. The `criticReviewsTotal` variable also shows a notable imbalance, with 69.62% of the titles having no critic reviews. The `awardNominationsExcludeWins` variable exhibits a similar trend, with 87.80% of titles having no nominations excluding wins. The binary variable `canHaveEpisodes` indicates whether a title can have episodes, and in this case, 90.27% of titles are classified as no (i.e., 0). Finally, the `userReviewsTotal` variable shows that 56.17% of titles have no user reviews.
This observation is also reflected in the analysis of skewness and kurtosis, which show a similar pattern. For example, variables like `numVotes`, `totalCredits`, `totalVideos`, and `userReviewsTotal` exhibit high skewness, indicating that most titles have few votes, credits, videos, or reviews, while some have exceptionally high values. The kurtosis in these variables is also high, suggesting that the data is concentrated in a few titles with extreme values. On the other hand, variables like `awardWins` and `totalImages` show high skewness but more moderate kurtosis, indicating that while there is a concentration of high values in a few titles, the outliers are less pronounced. In general, variables with high skewness and kurtosis have a skewed distribution, with most titles having low values and a few having extremely high values.

The charts we have chosen to include depict those that best capture the data distribution, and we will describe them in the following sections. It is also important to note that the graphical analyses shown in the following images were conducted before performing data cleaning for missing values, outliers, and possible duplicates, as will be discussed in the following subsections.

- The histogram 1.1 shows the distribution of `ratings` across different ranges, with the frequency of each rating range represented by the height of the bars. The majority of the records have ratings in the ranges (6,7] and (7,8], indicating that most of the data points fall within these mid-range values. This suggests that, in the dataset, ratings are generally more concentrated around the central intervals of this variable, with fewer records in the lower and higher rating intervals.

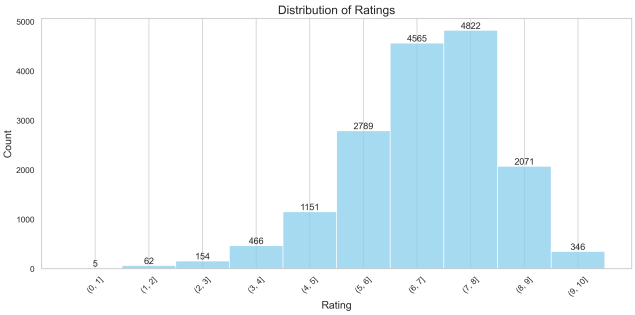


Figure 1.1: Distribution of Ratings

- The bar chart 1.2 illustrates the distribution of `originalTitle` across different genres. To conduct this analysis, the genres column (which contained multiple genres per title) was first split into individual genre values using the `str.split()` function. Then, the `explode()` method was applied to transform each genre into its own row, enabling a count of films based on individual genres. The bar chart shows the frequency of each genre, with the height of the bars representing the number of films associated with each genre. This visualization provides a clear view of how the films are distributed across various genres, highlighting the most popular genres in the dataset. The genre with the highest number of records is "Drama", which appears as the most frequent genre in the dataset.

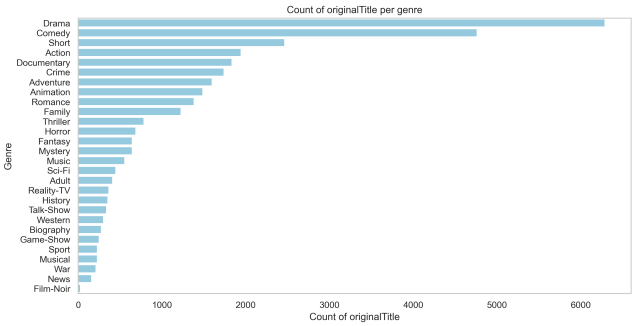


Figure 1.2: Distribution of Original Title per Genre

- The histogram 1.3 shows the distribution of titles across different `startYear` intervals. The distribution exhibits a left-skewed pattern, with a very low number of records in the early years (particularly in 1878) and a steady increase as the years progress. As the years advance, the number of titles rises rapidly, peaking in the most recent years (near 2024). This indicates a substantial growth in the number of releases over time, with the distribution becoming heavily concentrated in the more recent years. To calculate the optimal number of bins, the `Sturges' rule` was applied, which is a commonly used method to determine the number of intervals for histograms based on the size of the dataset. The formula for `Sturges' rule` is: $k = 1+log_2(n)$ where k is the number of bins and n is the number of records in the dataset.

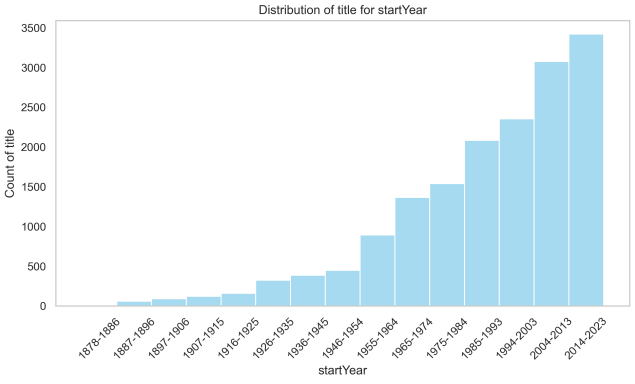
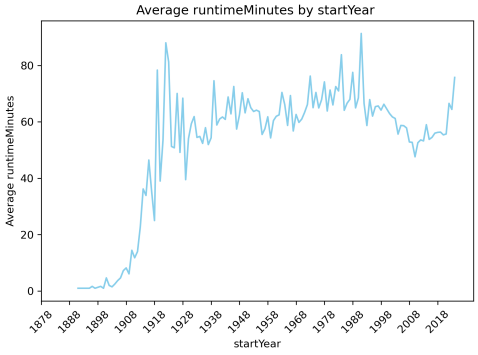
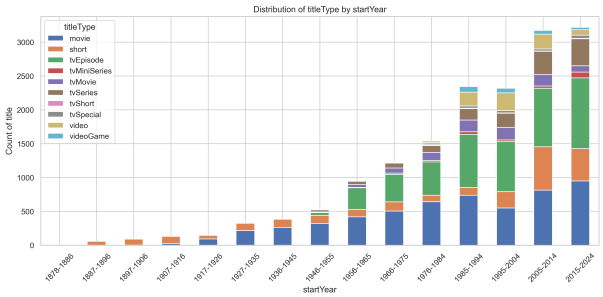


Figure 1.3: Distribution of Original Title per Year



(a) Distribution of Runtime per Years



(b) Distribution of Title Types per Years

Figure 1.4: Statistical Distributions of Runtime and Title Types

- The two charts in 1.4 provide insights into the trends in `runtimeMinutes` and `titleType` over the years. The first chart, 1.4a, illustrates the average runtime per year, showing how the average movie runtime has evolved from 1878 to 2024. This line chart helps to visualize how movie durations have changed over time, with particular attention to early and recent trends. The second chart, 1.4b, displays the distribution of title types by decade. This stacked bar chart highlights the number of different title types (e.g., movies, short films, TV shows) across various decades. By categorizing titles by their type and year, we can observe shifts in the kinds of content produced over time.
  As seen in 1.4a, the average runtime in the early years (especially around 1878) is quite low. This can be explained by the data in 1.4b, which shows that the majority of titles in the early years were of the "short" type. Moreover, based on the trends observed in both charts, it is likely that there is a strong correlation between runtime and title type.

1.2 Assessing data quality

In this section, to evaluate the quality of the IMDb dataset, we need to check if there are any missing values, outliers, and also duplicated data.

1.2.1 Missing Values

Column	Missing Values	Missing Percentage
endYear	15617	95.05
runtimeMinutes	4852	29.53
awardWins	2618	15.93
genres	382	2.32

Table 1.1: Missing values, counts and percentages

Only 4 out of 23 columns contain null values, specifically the features `runtimeMinutes`, `awardWins`, `genres`, and `endYear`. In particular, as we can see from 1.1, the `endYear` column has more than 90% null values, unlike the other columns, which have null value percentages below 30%. For this reason, the `endYear` column will be removed from our dataset, as it would not make sense to replace 95% of the observations with values derived from the remaining 5%. As for the other features with null values, we will replace them with descriptive statistics derived from their respective columns:

- `runtimeMinutes`: we will replace this quantitative variable with the value 58, the median, to avoid any distortions caused by outliers, which we will address in more detail in the following subsection;
- `awardWins`: for this attribute, the value we will use to replace NaN values is 0, since approximately 75% of the titles in this dataset have not won any awards, making 0 both the mode and the median (and this increased the percentage of titles without any awards to nearly 90%, further amplifying the variable's imbalance);
- `genres`: for this variable, being categorical, we will use the mode to replace the missing values, which is the genre 'Drama'.

1.2.2 Outliers

In this section, we examine whether there are records in our dataset that show extreme values for certain variables. We exclude the variables `isRatable`, `worstRating`, and `bestRating` from our outlier analysis since, after initially examining their variability and noting that they each have only one distinct value, we deemed it unreasonable to retain them in our dataset. Therefore, from this point onward in our analysis, we will exclude these three columns from the dataset.

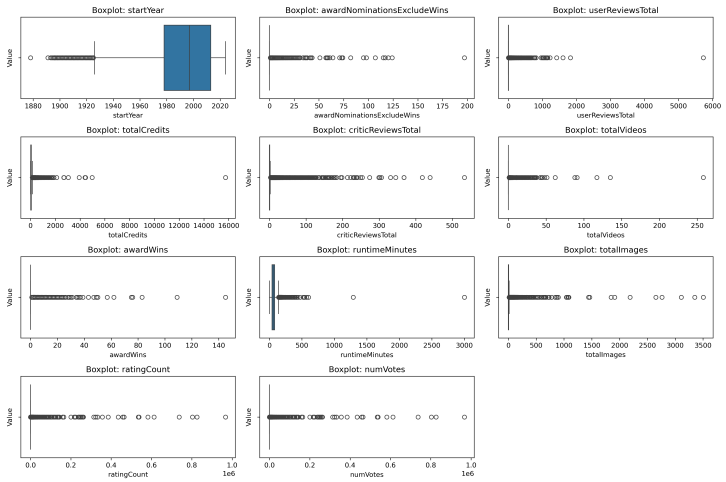


Figure 1.5: Variables' Boxplot

Now we analyze the numeric attributes' boxplots of figure 1.5:

- `runtimeMinutes`: the boxplot of this variable is highly concentrated on values below 500 minutes, except for two observations that deviate significantly, both having runtimes above 1000 minutes. For this reason, to avoid potential distortions in our analyses, we will remove these two observations with extreme runtime values from the dataset;

- `awardWins`: the majority of observations, as shown by the boxplot, which is rather flattened at 0, have not won any awards. For the award-winning titles, the number does not exceed 60 awards won, except for three titles that go beyond this threshold. Therefore, we can remove these cases of more than 60 awards won from the dataset;

- `awardNominationsExcludedWins`: the only observation we chose to remove from the dataset is the one with a value of 200 for this variable, as all others have a maximum of 125 nominations without any awards won per title;

- `userReviewsTotal`: many records have a number of user reviews higher than 0 but not exceeding 2,000 reviews per title, the only exception is a title with more than 5000 reviews, and since it is the only one above 2000, we will delete that record;

- `totalCredits`: this variable also presents several extreme values, as the boxplot is flattened at zero; anyway we have removed from the dataset the only outlier that, unlike the others with a maximum total credits value of about 5000, has a value of 16000;

- `criticReviewsTotal`: the boxplot of this variable shows that extreme observations are quite spread out; however, most of them are below 400; therefore, we will remove only the three observations that exceed this number of critic reviews per title, as we can see in its boxplot;

- `totalVideos`: even if the number of totalVideos of every record is really close to 0, there are lot of values much higher; despite this we remove the three records having a number of videos exceeding 100;

- `totalImages`, `ratingCount`, `numVotes`, `startYear`: these four variables, like the others we have analyzed so far, contain a high number of outliers; however, we will not remove any record with an extreme value, as they exhibit a high variability in values.

1.2.3 Duplicates

This dataset contains no repeated records, so each row is unique. The only variable worth analyzing for possible duplicates is the first one, `originalTitle`, which contains the names of all observations. We note that this feature has nearly 400 duplicates, meaning multiple observations can share the same title. Nonetheless, a closer look at these rows in the dataset reveals that the other features of these observations have different values, and upon analyzing the main descriptive statistics, we see that they show variance values greater than zero in most cases. For the reasons mentioned above, we believe it is meaningful to keep these rows in the dataset rather than removing them, as they refers to different records, but with the same original title.

1.3 Pairwise correlations and eventual elimination of variables

Up to this point in our data quality analysis, we have removed only a small number of observations and mainly four columns, so the analysis of correlation between variables should now be more accurate. In the correlation table provided here, we can assess whether there are variables that are correlated with each other. Looking at this matrix (figure 1.6), we immediately notice that there are two variables

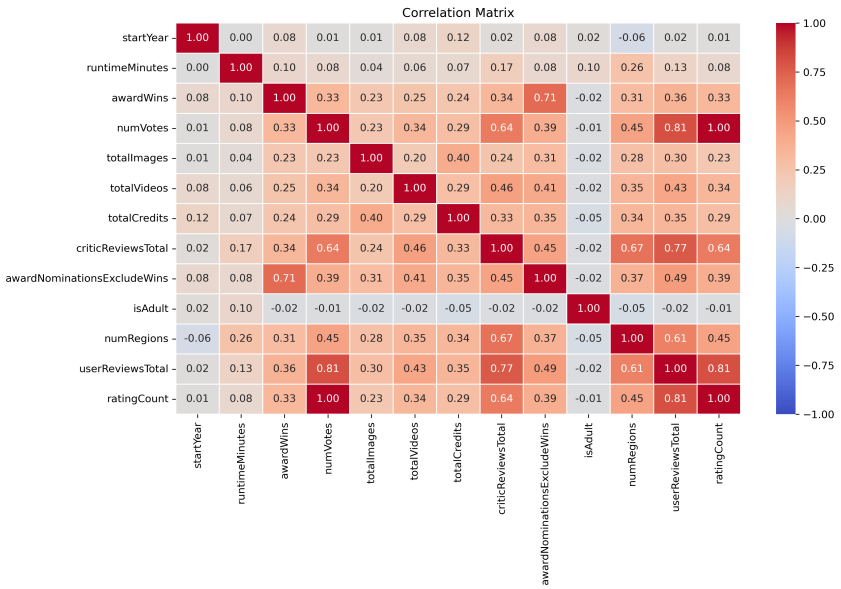


Figure 1.6: Correlation Matrix.

with maximum correlation, as they are equal to 1: `numVotes` and `ratingCount`. Therefore, we can proceed to remove one of them from the dataset (and we chose `numVotes`). We also observe that these same variables are correlated at 81% with `userReviewsTotal`. Other correlations above 70% between features include a 0.77 correlation between `userReviewsTotal` and `criticReviewsTotal`, and finally, a 0.71 correlation between `awardWins` and `awardNominationExcludeWins`. However, despite these high correlation values, we have chosen to retain these variables in our dataset for now. There are also three variables in our dataset that exhibit very low correlations with all other variables: `startYear` and `isAdult`, which, as observed in the correlation matrix, have values very close to 0. Another weakly correlated variable is `runtimeMinutes`, which, however, shows a correlation of 0.26 with `numRegions`, while all other correlation values do not exceed 0.17.
To summarize, at the end of our data understanding, preparation, and correlation analyses, we decided to exclude the following variables from our dataset for the analyses we will conduct in the subsequent chapters: `numVotes`, `isRatable`, `worstRating`, `bestRating`, and `originalTitle` (as it is a variable that identifies observations and provides no utility for the analyses). However, throughout the analyses in the following chapters, we will modify some variables to better suit the specific needs of each analysis.

2 Clustering

After the data understanding (chapter 1), we have dropped these columns: `endYear`, `bestRating`, `worstRating`, `isRatable`, `numVotes`, `originalTitle` for different reasons discussed before, and now to compute a clustering analysis we will only consider the quantitative variables, to see which will give us the best results. In summary, for the analysis in this chapter, we will exclude binary and nominal variables in the main cases (with the only exception that we will see later in the subsection); as a result, our dataset now consists of 16416 records and 11 feature.

2.0 Normalization

To normalize the data, we used `z-score normalization` (which standardizes the data by removing the mean and scaling to unit variance) and `min-max scaling` (which scales the data to a fixed range, typically [0, 1]). We then plotted the normalized data using both methods, alongside the original data, to check if the two normalization techniques produced significantly different results. However, the normalization outcomes were very similar for both methods. Therefore, we examined the presence of outliers, but found no extreme outliers that could have skewed the results of the two normalization techniques. We then proceeded to perform clustering on the normalized data using both `z-score` and `min-max` scaling. Interestingly, we observed that `min-max` scaling produced better results in terms of `SSE` and the `Silhouette score`. We chose to use `min-max` scaling to normalize the data mainly because of its direct and linear effect on the range of the data. Although `z-score` normalization is better suited to handle outliers, in our case, `min-max` scaling produced better results during the clustering phases.

2.1 Analysis by Centroid-based Methods: K-means

K-means is a partitional clustering approach where the number of clusters, k, must be specified in advance. Each cluster is defined by a centroid (center point), and each data point is assigned to the cluster with the closest centroid. Before proceeding with the application of the K-means algorithm for clustering, we used two methods to determine the value of k: the SSE or Sum of Squared Errors and the Silhouette, as shown in Figure 2.1.

`SSE (Sum of Squared Errors)`: this method calculates the total squared distance between each data point and its assigned centroid. A lower SSE indicates better clustering, as the points are closer to the centroids.

`Silhouette`: this method measures how similar an object is to its own cluster compared to other clusters. A higher Silhouette score indicates that the points are well-clustered and distinct from other clusters.

By evaluating both the Silhouette score and the SSE, we tested different values for k, ranging from 4 to 8. After careful consideration, we chose k=6 as the final number of clusters for implementing the K-means algorithm. This value was selected because it yielded a relatively low SSE of 213.2 and a reasonably high Silhouette score of 0.36, indicating a good balance between compactness and separation of the clusters.

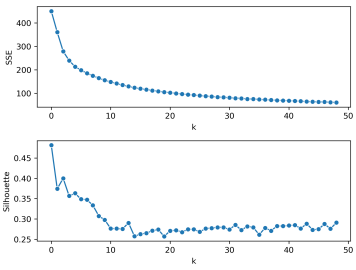


Figure 2.1: SSE and Silhouette

We first tested the clustering approach on all possible feature pairs in our dataset. For each pair, we plotted scatterplots to visually assess the relationships between the features. From these plots, we selected the two features that appeared to have the most significant clustering structure, which were `startYear` and `numRegions`. These features seemed to form well-defined groups in the scatterplots, making them the most suitable pair for clustering. After determining the optimal number of clusters (k=6) we applied the K-means algorithm to the dataset with the selected features. The scatter plot in 2.2 illustrates the clustering result. In the plot, we can observe that the clusters are generally well-defined, with only some minor overlap between the clusters. The centroids of the clusters are indicated by the red stars, which represent the mean value of the data points within each cluster. These centroids are computed by the K-means algorithm as the center of mass of each cluster, and they help visualize the general location

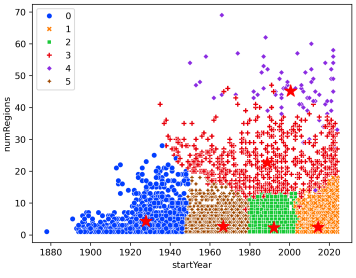


Figure 2.2: K-Means

of each cluster in the feature space. The SSE value of 213.2 and the Silhouette score of approximately 0.36 confirm the quality of the clustering. While the Silhouette score suggests that the clusters are moderately well-separated and cohesive, there may still be room for improvement. The SSE, being a measure of compactness, reflects a reasonable fit, though further refinement of the number of clusters or clustering technique could yield slightly better results.

2.1.1 Bisecting K-means

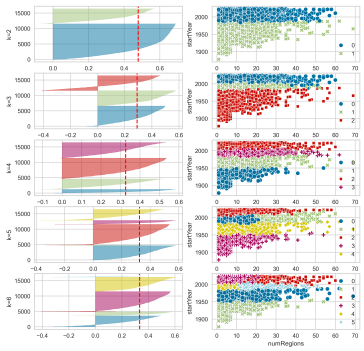


Figure 2.3: Bisecting K-Means: Silhouette Visualizer and Scatterplot

Bisecting K-Means is a variant of the standard K-Means algorithm that can produce hierarchical clustering. Unlike K-Means, where the number of clusters k is predefined, Bisecting K-Means begins with a single cluster containing all data points and iteratively splits the data into two clusters at each step. One key feature of Bisecting K-Means is that it does not use a predefined stopping criterion for bisection, meaning that it can keep splitting the data even if the resulting clusters are not necessarily useful. While this approach can be time-consuming, it can be effective in finding natural clusters, especially when there is some inherent hierarchical structure in the data. The plots in 2.3 illustrates the performance of the Bisecting K-Means algorithm on our dataset, where the number of clusters k ranges from 2 to 6. For each value of k, the plot shows both the Silhouette score and the corresponding scatterplot, which provides insight into the quality of the clustering. For instance, when applying Bisecting K-Means with k=6, the algorithm produced an SSE value of 229.38 and a Silhouette score of approximately 0.35. These values suggest that while the clustering is relatively well-formed, there is still room for improvement in terms of compactness and separation of the clusters. In comparison, when we applied the K-Means algorithm with the same selected features (`startYear` and `numRegions`), we obtained an SSE value of 213.23 and a Silhouette score of 0.36, which are slightly better than the Bisecting K-Means results. The Bisecting K-Means results are slightly worse than K-Means due to the algorithm's iterative splitting process, which can lead to the creation of singleton clusters—small, less meaningful groups. This approach may result in a slight decrease in cluster compactness and separation. Additionally, the Bisecting K-Means algorithm is sensitive to the initial selection of clusters for bisection, and it lacks an explicit stopping criterion, potentially leading to over-splitting. While the method is structured, these factors contribute to slightly worse SSE and Silhouette scores compared to K-Means.

2.1.2 X-means

X-Means is an extension of the K-Means algorithm designed to automatically determine the optimal number of clusters by using the Bayesian Information Criterion (BIC) as a splitting criterion. The algorithm recursively splits clusters using the Bisecting K-Means approach, evaluating whether the bisection improves the cluster structure by comparing the BIC of the parent cluster with the BIC of the resulting child clusters. The splitting process continues until no further meaningful splits are found, or the number of clusters exceeds a predefined maximum. For the implementation of X-Means, we used the `pyclustering` library, which provides an efficient and flexible way to apply clustering algorithms, including X-Means. As shown in Figure 2.4, the X-Means algorithm generated 20 clusters. This result is due to the large number of records in the dataset, which led the algorithm to divide even areas with little significance, creating clusters that may include noise. The algorithm attempted to group the data into meaningful clusters, but with many observations, it is possible that some clusters were generated to separate random variations or noise in the data.

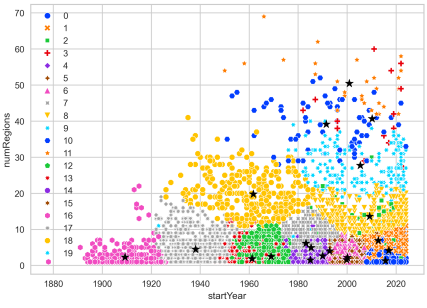


Figure 2.4: X-Means

2.1.3 K-mode

The K-Modes algorithm is a clustering technique that extends the concept of K-Means to handle categorical data. Unlike K-Means, K-Modes calculates the distance based on the number of mismatched attributes between objects. The algorithm groups objects based on the modal values (the most frequent

values) of the attributes, which serve as the "representatives" for each cluster. The algorithm follows an iterative process of assigning objects to the nearest cluster and recalculating the cluster centers until there is no change in the assignment of objects.

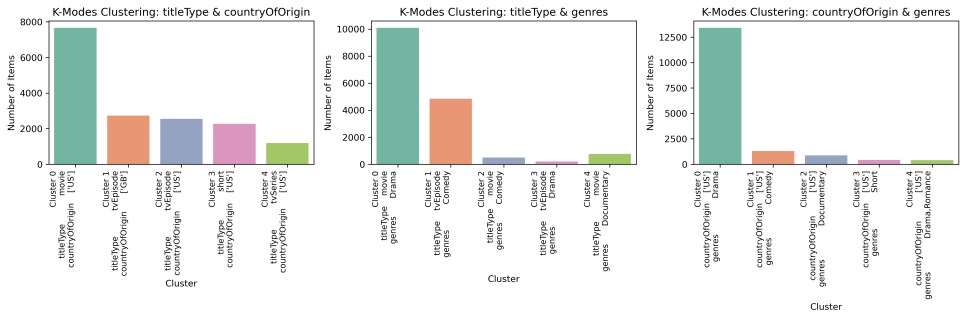


Figure 2.5: K-mode

For the K-Modes clustering, we focused on three categorical variables: `titleType`,`countryOfOrigin`, and `genres`. To begin, we transformed the data where necessary by "exploding" the columns `countryOfOrigin` and `genres`, which contained lists of strings, into individual values. This made the variables suitable for clustering. Then, we applied the K-Modes algorithm to group the data into clusters based on pairs of variables at a time. Each plot in 2.5 shows the number of items in each cluster for the selected pair of variables. From the obtained results, a clear separation emerges between movies and tvEpisodes, with films being mainly associated with the Drama genre, while tv episodes tend to group more around the Action and Adventure genres, often linked to productions from the USA. Moreover, the differences between genres play a central role in the clustering, with a marked separation between Drama movies and Comedy tvEpisodes. This suggests that the `titleType` and `genre` variables are particularly influential in determining the cluster structure, while `countryOfOrigin`, although a distinguishing factor, appears less significant in separating the groups.

2.2 Analysis by density-based clustering: DBSCAN

The algorithms discussed in this section are those based on data density, and the variables we plotted are those where the clustering division was most apparent. The DBSCAN algorithm requires two parameters to work: `eps`, which is the maximum distance to consider a point as part of the neighborhood of the considered point, and `min_samples`, the minimum number of points that must fall within the specified distance to classify a point as a core point.

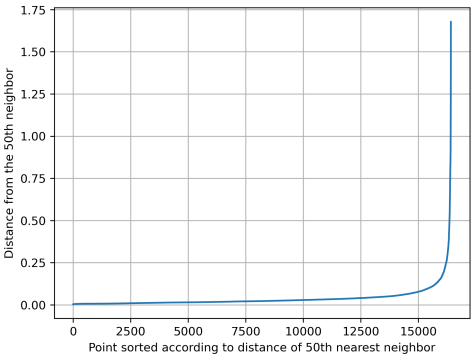


Figure 2.6: Kth Nearest Neighbor

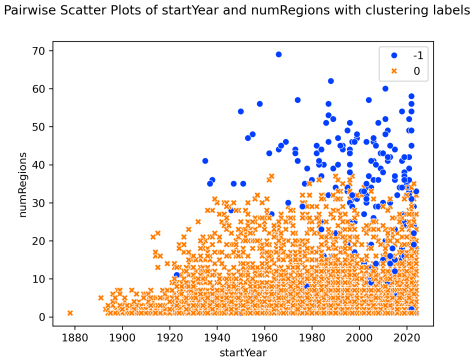


Figure 2.7: DBSCAN

The choice of the second parameter is guided by the large number of data points in the dataset, so we chose 50, also because this choice influences `eps`' value, even if in the specific case of our dataset, being very dense, we found that varying values of this parameter do not affect epsilon or the resulting clustering. For the choice of epsilon, we used the plot shown in figure 2.6 where we see the 50th nearest neighbor distance of the points, ordered in ascending order, and note that almost all observations in the dataset have 50 neighbors at an epsilon below 0.25. Therefore, we selected 0.15 as the epsilon value. As shown in the figure 2.7, the DBSCAN algorithm produces rather poor results on our dataset. Unlike the algorithms discussed in the previous section, we observe that the number of clusters is only one. The

other points (the one with label equal to `-1`) of these clusters are just 259 observations, representing the points that DBSCAN has classified as noise points (as they are not in some core points' neighborhood). The remaining observations, classified as core points or border points, fall into the unique cluster with much higher density. The poor results are likely due to the varying data density, as shown in the plot, and also because we are using high-dimensional data. Our poor clustering results are further emphasized by the fact that we cannot calculate the silhouette score for the resulting clustering. This is because having a single large cluster prevents the calculation of separation between clusters, a key component of the silhouette score. Additionally, the noise points that remain do not form a cluster, further complicating the assessment.

2.2.1 OPTICS

Using the OPTICS algorithm, which should address the problem of varying data density and is also resistant to noise points, the result should be somewhat different from the one seen with DBSCAN. In this case, the parameters required by the algorithm were the minimum number of points that must be present in a cluster to be considered as such (`min_cluster_size`), which we set to 200, then `max eps`, i.e., the maximum distance between two points for one to be considered as in the neighborhood of the other, which we set to 0.25, as in the previous algorithm and also because a non given value of it would create a unique cluster containing also all the noise points; finally `min_samples`, also set to 50. The parameters shared with DBSCAN were chosen based on the graph shown in figure 2.6, as well as the data density. On the other hand, `min_cluster_size` was chosen, after several trials, even we have tried to avoid having a single cluster as in the previous case, but at the same time, setting a very low threshold, takes as a clustering results lots of very small clusters. The clustering we obtained is very similar to the previous case; the only change is that we have only 100 points labeled as noise, while the rest are part of a single cluster. Furthermore, to understand why we obtained a single cluster, we can see in the figure 2.8 the reachability distance plot of all the points, sorted based on the value of this distance, where most of the observations have a defined value of 0.05 Therefore, the results of this second algorithm are not good for our dataset, as it tends to create either a single large cluster or many small clusters, with the majority of the points remaining outside these clusters, which are classified as outliers.

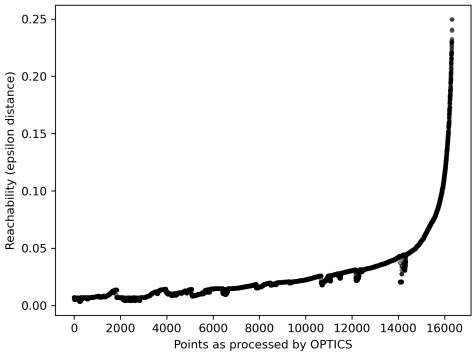


Figure 2.8: Reachability Distance

2.3 Analysis by hierarchical clustering: Agglomerative Clustering

In this section, we explore the various results of hierarchical clustering using the Agglomerative Clustering algorithm (the most common approach, which starts by treating each data point as a separate cluster and then aggregates them). We experimented with both `Manhattan` and `Euclidean` distances (except in the case of the `Ward linkage`, where `Manhattan` distance could not be applied). Both distance metrics were tested across different inter-cluster distance methods to identify the combination that could most effectively produce clusters suited to our data. Importantly, we did not predefine the number of clusters in the parameters.
In Figure 2.9, we show the dendrograms for all four cases (using `Euclidean` distance) for a selected subset of 50 data points, to ensure clarity in the representation. From the `Single Linkage` dendrogram, we immediately observe that this method is unsuitable for our data due to its tendency to group tightly packed points together. Reviewing the cluster labels assigned by the algorithm, we found only four clusters, one of which contained the majority of the data, the others just one point each. This resulted in a high silhouette score of 0.80, driven by the minimal number of clusters and a very big size of one of them. We also observed from the analyses conducted in the first chapter that this dataset contains a high number of outliers, and these can significantly distort the results of the `Minimum Linkage` method. For `Complete Linkage`, the clustering results differ significantly. The algorithm created 28 clusters, the large majority with a really low number of points, and only three have more than 1000 points each. The silhouette score was slightly lower, 0.37, as there are more clusters than in the single linkage case. But the distribution of data across clusters was far more balanced than in the other case, even if all the other clusters have a really low size, and this also is stressed by the fact that maximal methods, tends to break big clusters. Regarding the inter-cluster distances calculated with the `Group Average` method and the `Ward` method, we observed that the silhouette scores are 0.65 and 0.26, respectively. With the `Group Average` method, 4 clusters are created, but 3 out of the 4 contain only a single point, resulting in a

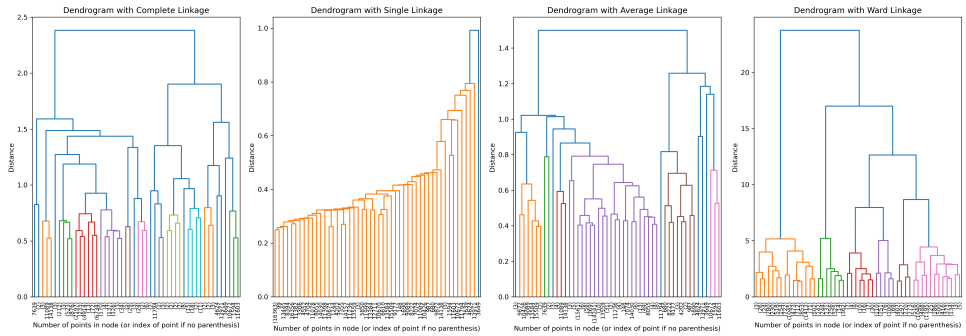


Figure 2.9: Complete, Single, Average and Ward linkage (Euclidean distance)

similar situation to `Single Linkage`. The low silhouette score with the `Ward` method is due to the high number of clusters (115), each containing between 2 and approximately 1000 points.
In conclusion, considering both the silhouette score and the need for a balanced number of clusters, the best results from Agglomerative Clustering with the `Euclidean` metric are achieved using the `Complete Linkage` method.

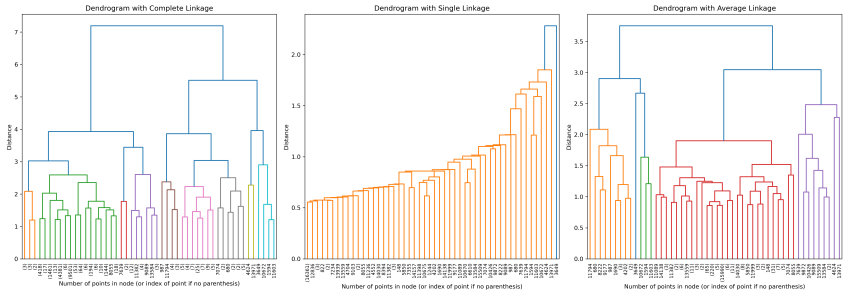


Figure 2.10: Complete, Single and Average linkage (Manhattan)

In Figure 2.10 we present the dendrograms for all three cases (using `Manhattan` distance) for a selected subset of 50 data points. The analysis reveals distinct differences in clustering behavior compared to the `Euclidean` case.
For `Single Linkage`, the algorithm creates a dominant cluster (Cluster 1) with 16,380 points, which overwhelmingly contains the majority of the dataset. Several very small clusters are also formed, each with just one or two points. This structure leads to a silhouette score of 0.70, reflecting a high degree of separation between the large cluster and the isolated smaller ones. However, the clustering remains unbalanced, with the vast majority of points assigned to a single large cluster and a small number of isolated points in the other clusters.
The `Complete Linkage` method produces even more fragmented results, with 75 clusters. Most of these clusters are small, containing only a few points (e.g., many clusters have just 1 point). The largest clusters, such as Cluster 37 (15,961 points) and Cluster 33 (145 points), dominate the structure. The silhouette score for this clustering is quite low (0.21), indicating poor overall separation of the clusters, primarily due to the large number of very small clusters and the imbalance in cluster sizes.
Finally, using the `Group Average` method, we observe a silhouette score of 0.31. This score indicates a moderate clustering structure, with several clusters containing more than 100 points, such as Cluster 37 (15,961 points) and Cluster 33 (145 points). Despite the larger clusters, there are still many small clusters with just 1 or 2 points, contributing to a lower silhouette score compared to the `Single Linkage` method. The overall distribution of points is more balanced than in the `Complete Linkage` method but still lacks clear separation in some areas.
`Manhattan` distance performs worse than `Euclidean` distance because it tends to create smaller and more isolated clusters, leading to lower separation and cohesion between groups. Additionally, the `Manhattan` metric underestimates the actual distance between points that are not aligned along the axes, which can result in less precise clustering and lower silhouette scores.

We also performed hierarchical clustering using the `Euclidean` distance with four linkage methods: `Ward`, `Average`, `Complete`, and `Single`, setting the number of clusters to 6 as we can see in Figure 2.11. This choice was made to compare the results obtained with hierarchical clustering to those obtained with k-means, which also had 6 clusters. Among the four linkage methods tested, the `Ward` method provided the best balance in cluster distribution. Despite having a relatively lower silhouette score of 0.336 compared to methods like `Single Linkage` (0.802) and `Average Linkage` (0.755), `Ward` resulted in more evenly sized clusters, with no single cluster dominating the others. In contrast, the other methods, particularly

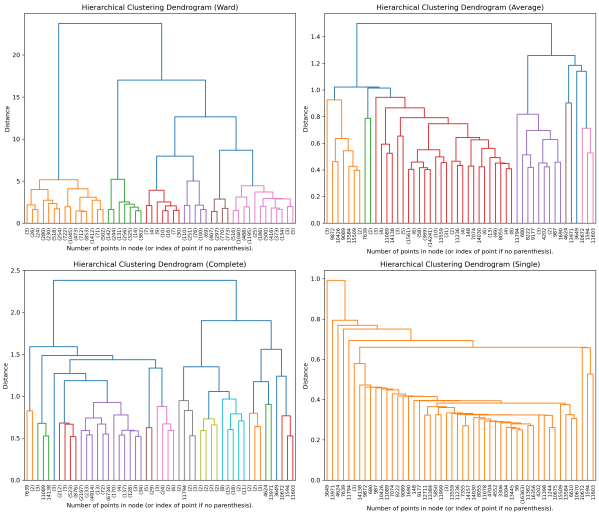


Figure 2.11: Complete, Single and Average linkage

`Single` and `Average`, created a dominant cluster (cluster 0) that contained the majority of the points, with the remaining clusters being very small (some with only 1 or 2 points). This made the distribution highly imbalanced, even though the silhouette scores were higher. The `Complete` method also showed a similar issue, with one large cluster and many smaller ones, but with a slightly better silhouette score (0.725). Therefore, `Ward` is considered the best method for ensuring more balanced clusters, making it the preferable choice despite the lower silhouette score.

2.4 Final Discussion

As we observed in the previous sections, the best clustering results on our dataset were obtained using centroid-based algorithms. In contrast, density-based algorithms produced a single large cluster, leaving out the points classified as outliers. For hierarchical clustering, the results from the dendrogram of the entire dataset, as expected, were confusing due to the high volume of data.

3 Classification

In this section, we will apply several algorithms to two different classification tasks. The first task involves the use of the `rating` variable, an ordinal variable with ten distinct classes, each representing a range of scores. This variable will be used for all three classification models (KNN, Naive Bayes, and Decision Tree). For the second task, different variables will be used (`titleType` for KNN and Naive Bayes and `hasAwards` for Decision Tree). To proceed with these tasks, the non-quantitative variables (`titleType`, `countryOfOrigin`, and `genres`) were converted using the `One-Hot Encoding` technique. This technique transforms categorical variables into binary columns for each distinct value. As a result, the number of columns in the dataset increased significantly.
To summarize, the columns we are now using include: `startYear`, `runtimeMinutes`, `totalImages`, `totalVideos`, `totalCredits`, `criticReviewsTotal`, `awardNominationsExcludeWins`, `awardWins`, `can HvaeEpisodes` , `isAdult`, `numRegions`, `userReviewsTotal` and all the columns derived from separating the distinct values of `countryOfOrigin`, `genres` and `titleType` (which leads to a very high number of columns). Furthermore, before proceeding with the classification tasks, we cleaned the test set following the same principles applied to the training set by removing columns, outliers, and missing values, and performing the transformation of non-quantitative variables through `One-Hot Encoding`. Before doing any classification tasks we also noted the training dataset contained more countries in the `countryOfOrigin` variable compared to those present in the test set. To address this discrepancy, we added the missing columns in the test set and filled them with 0s, ensuring no errors would occur during the implementation of one of all the classification algorithms.
This ensured that the test set was also ready to be used for evaluating the predictions made by the various algorithms, as will be explained in more detail in the following sections. The goal is to evaluate and compare the performance of each model in terms of `accuracy`, `precision`, `recall`, `F1-score`, and other relevant metrics.

3.1 K-Nearest Neighbors

3.1.1 Rating Prediction

For the KNN classification, we started by splitting the train set into a train set and a validation set using the `train_test_split` function from scikit-learn. The split was made by allocating 80% of the data for training and 20% for validation. This was done to optimize the model and test its performance on unseen data during training, before applying it to the final test set. During the model optimization phase, several tests were conducted to select the best hyperparameters. First, the relationship between the number of neighbors and the model's performance was explored. To this end, a graph was used to show accuracy and standard deviation as a function of the number of neighbors considered, within the range from 1 to 50. Subsequently, to further refine the selection of hyperparameters, the `RandomizedSearchCV` method was used. This approach allowed for a more efficient exploration of hyperparameter combinations, including the number of neighbors (ranging from 1 to 200), weights (`uniform` or `distance`), and distance metrics (`euclidean` or `cityblock`). The choice of `RandomizedSearchCV` was primarily driven by the need to reduce computation time, as using `GridSearchCV` would have required an unmanageably long computational time for our systems.
Finally, the optimized parameters were tested on the test set to evaluate the model's final performance. Initially, the base model with the default parameters was used, which returned an accuracy of 0.35. The model was then tested with the optimal number of neighbors, determined through the analysis of a graph showing the relationship between the number of neighbors and the model's accuracy. This choice resulted in a slight improvement in performance compared to the base model (accuracy of 0.36). However, the best result was achieved using the parameters selected through `RandomizedSearchCV`. In this case, the optimal parameters identified were: `n_neighbors=196`, `metric="cityblock"`, `weights="distance"`, which led to an accuracy of 0.40 on the test set.
The model's `ROC AUC` score is 0.70, indicating a good ability to discriminate between classes (see Figure 3.1). However, as shown in Figure 3.2, precision and recall are very low, especially for certain classes, such as class 1, which represents the range (0, 1]. This issue is primarily due to the significant class imbalance present in the dataset. Some classes, like class 1 and class 3, have only a few samples (1 and 52, respectively), while

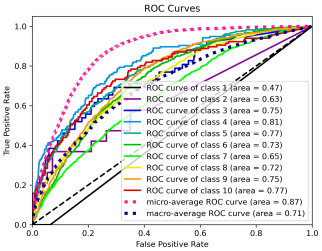


Figure 3.1: ROC Curves of Rating Prediction

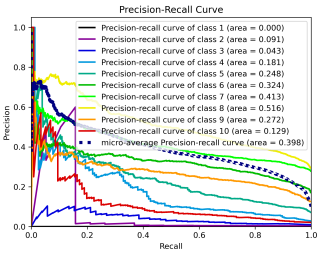


Figure 3.2: Precision-Recall Curves of Rating Prediction

others, such as class 8, have over 1600 samples. As analyzed in the distribution of the target variable in Section 1.1, this class imbalance leads to a model that struggles to predict the minority classes accurately, thereby lowering both precision and recall for those classes.

3.1.2 TitleType Prediction

For the classification of the `titleType` variable, we followed the same procedure as for the rating prediction task, including splitting the dataset into training and validation sets, optimizing hyperparameters using `RandomizedSearchCV`, and evaluating model performance with `accuracy`, `ROC`, and `Precision-Recall` curves. However, two variations of the classification were tested. In the first, we applied `One-Hot Encoding` to the `titleType` variable and performed a binary classification, selecting the `titleType_movie` column as the target. This resulted in a classification task with two classes: "movie" vs "non-movie". In the second variation, we used `Label Encoding` to treat the problem as multiclass classification, where the target variable consisted of multiple categories (e.g., movie, TV episode, short, etc.). The same model optimization steps were applied to both encoding strategies, with evaluation conducted on the test set to assess final performance. For both cases, the best results were achieved using the parameters selected through `RandomizedSearchCV`. For the binary classification on `titleType_movie`, with parameters `n_neighbors=110`, `metric="cityblock"`, `weights="distance"`, we obtained an accuracy of 0.96, a ROC AUC Score of 0.9932, and precision and recall values of 0.99 and 0.98, respectively. These excellent results may be influenced by the nature of the dataset, where "movie" is the most frequent type in the `titleType` variable, which could have favored the prediction of this class. For the multiclass classification on `titleType`, the parameters selected through `RandomizedSearchCV` were `n_neighbors=24`, `metric="cityblock"`, `weights="distance"`, with an accuracy of 0.78 and a ROC AUC Score of 0.9056. As we can observe from the Figures 3.4 and 3.5, the most frequent classes, such as class 1 (tvEpisode), class 3 (movie), and class 7 (short), have higher precision and recall values compared to class 10 (tvShort), which is much less frequent than the others and has the lowest values.

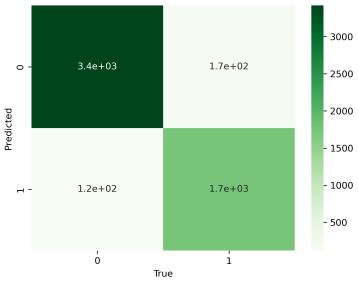


Figure 3.3: titleType_movie Confusion matrix

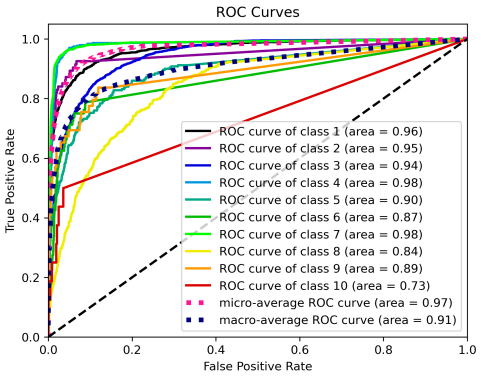


Figure 3.4: ROC Curve of titleType Prediction

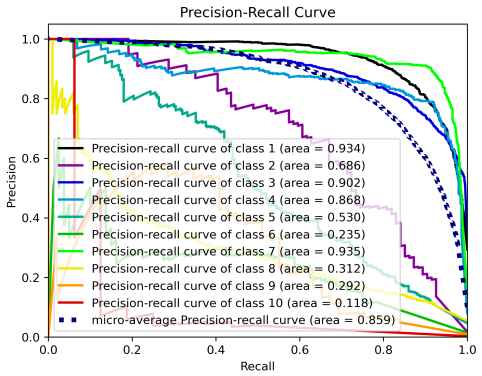


Figure 3.5: Precision and Recall of titleType Prediction

3.2 Naive Bayes

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem, assuming independence between the variables. In this study, we use two variants: `Gaussian Naive Bayes`, suitable for continuous variables with a normal distribution, and `Categorical Naive Bayes`, designed for categorical variables. In the following sections, we will apply both models to predict the `rating` and `titleType` variables in the dataset.

3.2.1 Rating Prediction

The following are the results of the `rating` predictions using the Gaussian Naive Bayes and Categorical Naive Bayes models. The table 3.6 shows the classification metrics for both models, while the heatmaps of the confusion matrices 3.7 and 3.8 provide a visual representation of the accuracy and classification errors for each model.

	Gaussian Naive Bayes	Categorical Naive Bayes
Accuracy	0.04	0.33
Precision	0.25	0.27
Recall	0.04	0.33
F1-Score	0.05	0.26

Figure 3.6: Naive Bayes Classification for Rating

The `Gaussian Naive Bayes` model shows poor overall performance, with particularly low precision, recall, and f1-score. The model struggles significantly with most ranges, especially with the range (3, 4], where both precision and recall are almost zero. Some marginal improvements are observed in the ranges (6, 7] and (7, 8], but the performance remains suboptimal overall. On the other hand, the `Categorical Naive Bayes` model performs slightly better, demonstrating a stronger ability to correctly predict the ranges, particularly in (6, 7] and (7, 8]. This model shows better precision, recall, and f1-scores in the middle ranges.

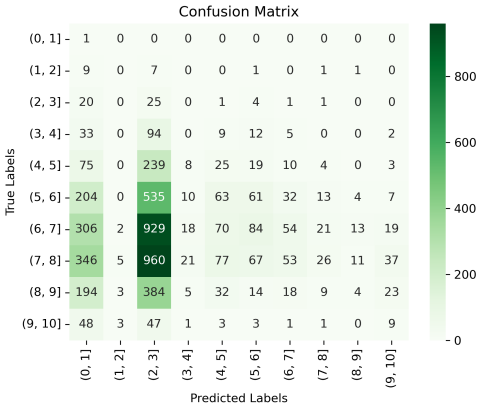


Figure 3.7: Gaussian rating

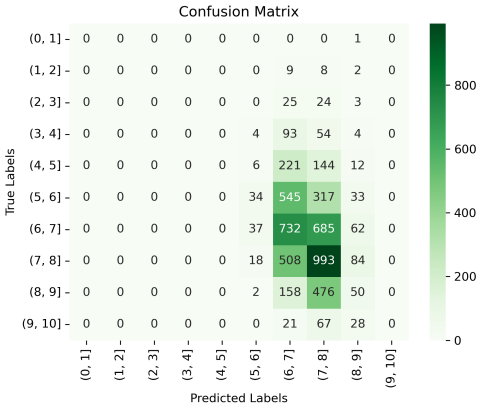


Figure 3.8: Categorical rating

3.2.2 TitleType Prediction

The following are the results of the `titleType` predictions using the `Gaussian Naive Bayes` and `Categorical Naive Bayes` models. The table 3.9 shows the classification metrics for both models, while the heatmaps of the confusion matrices 3.10 and 3.11 provide a visual representation of the accuracy and classification errors for each model. The `Gaussian Naive Bayes` model shows generally mediocre performance, with low precision and recall across many classes. In particular, the model struggles significantly with the tvMovie class (misclassified as movie) and tvSpecial, with almost zero precision and recall. The best performance is observed in the tvEpisode and movie classes. The `Categorical Naive Bayes` model performs noticeably better than the `Gaussian Naive Bayes`, with higher precision and recall, especially in the tvEpisode, videoGame, and movie classes. While it shows satisfactory results in some classes, such as tvEpisode (with a recall of 79%) and movie (with a recall of 73%), the model has difficulty correctly predicting classes like tvSeries, tvMiniSeries, and tvSpecial, where both precision and recall are significantly lower.

	Gaussian Naive Bayes	Categorical Naive Bayes
Accuracy	0.25	0.45
Precision	0.3	0.4
Recall	0.2	0.3
F1-Score	0.24	0.35

Figure 3.9: Naive Bayes Classification for titleType

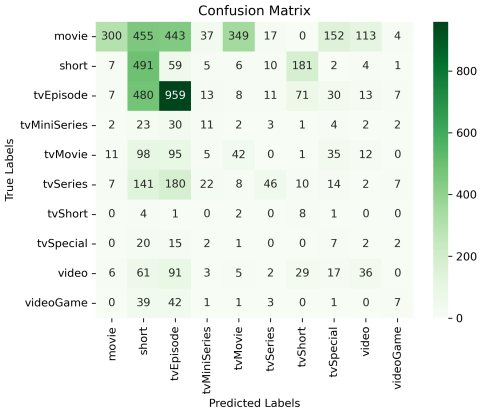


Figure 3.10: Gaussian titleType

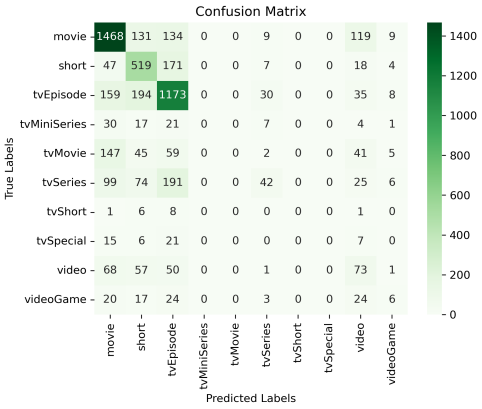


Figure 3.11: Categorical titleType

3.3 Decision Tree

3.3.1 Rating Prediction

The Decision Tree is a prediction algorithm that constructs an abstract model to predict the values of a target variable (in this case, categorical). Each leaf is selected based on specific criteria, such as Gini impurity or entropy (information gain). The final output is a tree structure, where each path represents a sequence of decisions leading to a predicted class label. First of all, to use the decision tree, we had to convert all the variables, both those we will use to predict the rating and the target variable itself, into numerical values (as stated before); specifically, for the `rating`, we decided to assign the upper bound (i.e., the included bound) as the numerical value for each interval.

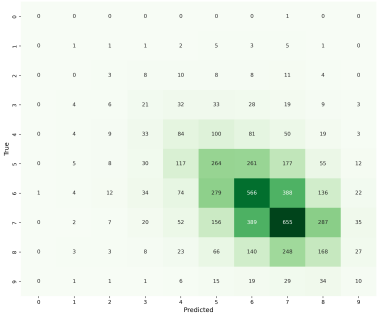


Figure 3.12: Rating Confusion Matrix

First, we obtained our predictions using all these columns listed above, without applying any hyperparameters in our decision tree, and we achieved very high accuracy on the training set, 0.99, while the accuracy on the test set was around 0.32, as we can see in the table 3.13. Furthermore, we can observe that the most frequently predicted values are those within the ranges (6,7] and (7,8], which belong to the rating classes with the highest frequency in the dataset. This is at the expense of classes with much lower frequencies, such as the rating in the range (0,1], which was predicted only once, and incorrectly.
Additionally, from the results, we inferred a clear case of overfitting of the decision tree. To justify this, as shown in image 3.14, we observe that as the number of nodes or the depth of the tree increases, the error in the prediction on the test set also increases, while there is a simultaneous improvement in accuracy on the training set.

Class	Precision	Recall	F1-Score	Support
1	0.00	0.00	0.00	1
2	0.05	0.05	0.05	19
3	0.07	0.06	0.06	52
4	0.17	0.17	0.17	155
5	0.22	0.23	0.23	383
6	0.29	0.29	0.29	929
7	0.36	0.36	0.36	1516
8	0.41	0.41	0.41	1603
9	0.23	0.23	0.23	686
10	0.10	0.09	0.10	116
Accuracy				0.32
Macro Avg				0.19
Weighted Avg				0.32

Figure 3.13: Decision Tree's results on test set

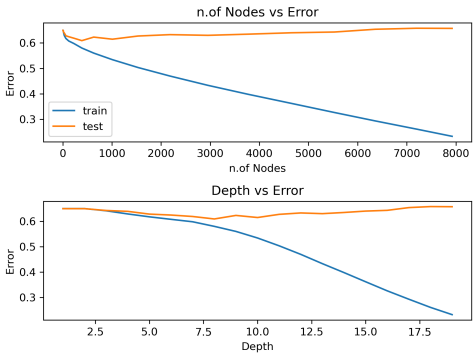


Figure 3.14: Number of nodes, Depht and Prediction Error

Therefore, we first tried to estimate the various hyperparameters of the algorithm using `RandomizedSearch` to see if the accuracy on the test set increased and also by escluding from the variables with a feature importance under 0.0001. What we obtained using the parameters from the `RandomizedSearchCV` algorithm, `min_samples_split=10, min_samples_split=10, max_depth=8` and `criterion= 'gini'`, and the results was still a very low accuracy on the test set, approximately 0.39, but also a very low accuracy on the train set, 0.41. Furthermore, we noticed by examining the report (the same type of report shown in Figure 3.13 but for the previous case) that there were no predictions for a rating of 1 in the test set. This highlights how, in the case of values that are very infrequent in the observations, the algorithm might not classify any of these values. We also tried using the pre-pruning technique (removing redundant branches that do not contribute additional information to predictions, keeping only the root node of the branch) to visualise the tree after simplifying it, as we can see from Figure 3.15, where we had to show only the nodes of his first three level of the tree for a better readability.

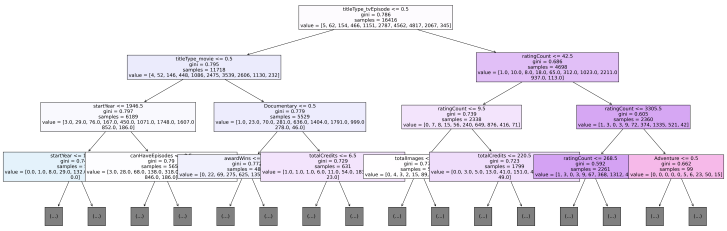


Figure 3.15: Decision tree with pre-pruning and depth=3

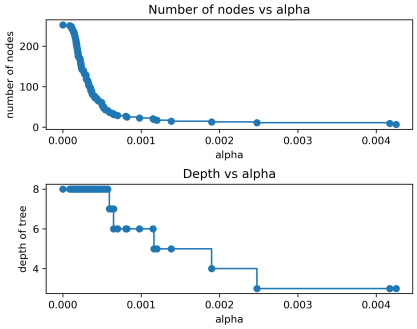


Figure 3.16: Number of nodes, depth, and effective alphas

As a final attempt then we also tested whether adding another hyperparameter, and the ones used before were set on default values, `ccp_alphas` (which prunes the decision tree by removing the weakest link nodes based on the cost-complexity criterion) could improve the accuracy of our rating prediction model. In Figure 3.16, we see that for values of `ccp_alphas` greater than 0.001, the number of nodes decreases, leading to a consistently smaller tree height. We selected the value for this parameter using the `GridSearchCV` algorithm and a value of aplhas equal to 0.00033; despite this, the model's visualised tree became more readable with a lower number of nodes if compared with the other ones, but accuracy remained similar to the one obtained without eliminating the weakest links: test accuracy of 0.37 and train accuracy of 0.42.
To summarize, we observed that our model tends to overfit, as it achieves a training set prediction accuracy of 0.99 without specifying any hyperparameters for the algorithm. However, in all other cases we examined, this accuracy drops to around 0.40 for predictions on both datasets.

3.3.2 HasAwards Prediction

Furthermore, in addition to the variable `rating`, we attempted to construct a decision tree to classify a binary variable `hasAwards` that we created by mapping a 0 to all the titles in the dataset that hasn't won any award, and with a 1 the observations that has won at least one award. The resulting variable is unbalanced because the 89% of the titles has not won any award. Then the column used for doing a prediction of this variale are the same used before, but without of course `awardWins`, and additionally the `rating` column, that we transformed using `Ordinal Encoder` (as it is an ordinal variable). We first used the algorithm with all default values and considered all columns, resulting in a training set accuracy of 1 and a test set accuracy of 0.88. Next, we explored whether the results in predicting such an imbalanced class would change by tuning the decision tree's hyperparameters using the randomize search and what we obtain is: `min_samples_leaf = 50, min_samples_split = 5 and max_depth = 5 and criterion = gini` (we also choose to use them also because by seeing the accuracy plotted with the possible value of the hyperparameters this values had an accuracy not under the 0.90).

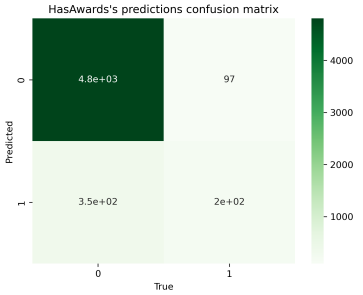


Figure 3.17: HasAwards Confusion matrix

Additionally, we selected only the columns with a feature importance greater than 0.0001. Then from using these feature what we obtain is a lower accuracy for the training set than before, but at the same time

a higher test set accuracy, in both case equal approximately to 0.92. Additionally, the resulting tree has only 6 levels of depth (see Figure 3.18) and still demonstrates high prediction accuracy on both datasets.

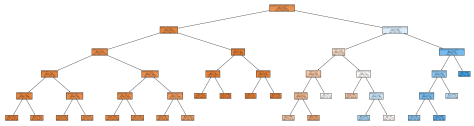


Figure 3.18: Complete Tree for predicting HasAwards

Even when observing the Precision-Recall Curve and the ROC Curve of Figures 3.19 and 3.20, we notice that values equal to 0 are predicted with high accuracy: the area under the ROC Curve for this class and also the one under the Precision-Recall Curve are over 0.85. However, for values equal to 1 (titles that have won at least one award), the model's ability to correctly identify this class is significantly lower: the area for this class is 0.88 in the ROC Curve but only 0.536 in the Precision-Recall Curve.This result highlights a clear bias of the model towards class 0, indicating a challenge in handling the prection of class 1. We can see how the decision tree is influenced by the presence of a dominant class. Specifically, it predicts a disproportionately high number of 0s, also when the true value of the observation is 1, as we can see from the confusion matrix of the predicted values in figure 3.17, as the majority of `originalTitle` entries in the dataset have not won any awards. From these results, we finally state that when predicting a highly imbalanced binary variable, even though the algorithm's prediction may disadvantage the minority class the accuracy remains consistently high in both the test and training sets, and this holds true even when changing the variables to include or modifying the parameters of the decision tree.

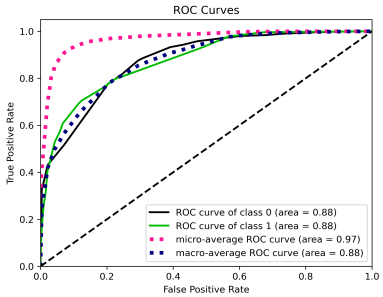


Figure 3.19: HasAwards ROC Curves

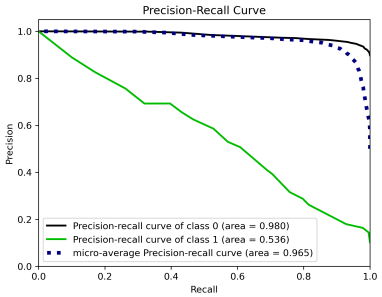


Figure 3.20: HasAwards Precision-Recall

3.4 Final Discussion

After analyzing all the predictive models for estimating the `rating` variable discussed in the previous subsections, we decided to summarize the best results obtained with the various algorithms in Figure 3.21 for a comprehensive comparison.
As shown, the accuracy results are quite similar, ranging from a maximum of 0.40 to a minimum of 0.33. Therefore, we can conclude that no method produced significantly different results from the others.

RATING PREDICTION		
	PARAMETERS	ACCURACY
K-Nearest Neighbors	Randomized Search	0.40
Naive Bayes	Categorical	0.33
Decision Tree	Randomized Search	0.39

Figure 3.21: Comparison of classification performance

4 Pattern Mining and Regression

4.1 Regression

In this section, we will explore various regression techniques to identify the most suitable coefficients for constructing a robust regression model for predicting the values of a newly added column, `averageRating`, included in both the training set and the test set (a continuous variable that we have used to replace the ordinal variable `rating`). Before building the various regression models, we examined whether there were any correlations among the quantitative variables in our dataset. We observed that `averageRating` is the only variable showing correlation levels with the other variables that are very close to zero. This could explain the poor regression results obtained.
In the next subsections we will use different algorithms for performing a simple regression, a multiple regression and a multivariate regression (both with linear and non linear regressors).

4.1.1 Simple Linear Regression

The regression algorithms we will use to estimate the regression coefficients are: LinearRegression, Lasso and Ridge. The first employs the OLS (Ordinary Least Squares) estimation method, which minimizes the sum of squares of the differences between the actual values and the predicted values. The other estimation methods in order to avoid a possible model over-fitting on the training dataset, if coefficients are too large,adds a penalty parameter (which can be specified as a hyperparameter) to the loss functions to penalize large coefficients: Ridge's Loss function = OLS + alpha * summation (squared coefficient values) and Lasso's Loss function = OLS + alpha * summation (absolute values of the magnitude of the coefficients).
As a first step, we attempted to predict the values of `averageRating` using `criticReviewsTotal` (a choice guided by the correlation between the first variable and all the others). The results obtained with the four regression algorithms mentioned above were poor. For each regression line, the slope was very close to 0 (`Lasso regressor: 0`; `LinearRegression and Ridge: 0.0004`) , and the intercept was 6.71. This means that regardless of changes in the independent variable, the predicted value of averageRating will always remain constant. Therefore, linear methods did not yield an adequate predictive model for the variable, likely due to the extremely low correlation between the two variables and the scattered values, as shown in Figure 4.1 (where we choose to represent the `LinearRegression` line of the train dataset, to better see how this line tried to fit the datas).
Further evidence of this is provided by the low fit metrics, which are identical across all four cases: an R^2 of 0, indicating that the model does not explain any of the variability in the rating values, and also that the value of the intercept is really close to the mean of all the `averageRating`'s values. Additionally, the MSE of 1.83 and MAE of 1.06 highlight the magnitude of prediction errors, with MAE representing the average error in the predicted valuesAdditionally and both of which remain notably low.

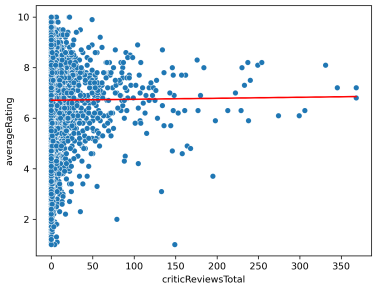


Figure 4.1: Scatter plot with LinearRegression on train dataset

As we observed, we obtained the same results from all four regressors. Therefore, we then tried setting a different alpha penalty value, different from the default so we determined this value by testing different levels of it by using the `GridSearchCV` algorithm, and we obtained a value of alpha equal to 100, but our result in terms of coefficients and metrics remain the same also in this case.

4.1.2 Simple Non Linear Regression

The nonlinear regressors we are now testing to predict `averageRating` are the `DecisionTreeRegressor`, which is equivalent to the decision tree for continuous variables, with the only difference being that, in the case of an impure leaf, the predicted value will be the mean of all the values in that leaf. Additionally, we are testing the `KNN regressor`, which makes predictions based on the average of the target values of the k nearest training samples to the query point.
First, we apply both algorithms with all default hyperparameters to predict `averageRating` using `criticReviewsTotal`, to assess whether there is any improvement in how the predicted values fit the data. The fit metrics obtained from the decision tree are also very poor, with an R^2 of -0.009, MSE of 1.85, and MAE of 1.07. As for the results of the `KNN regressor`, they are even worse: the R^2 is negative, at -0.264, and also the MSE and MAE are even bigger to those of the decision tree (MSE: 2.317 and

MAE: 1.247). This continues to demonstrate that the independent variable fails to explain the variability of the dependent variable effectively, as we can also see by Figure 4.2 where the regression models are not able to capture the datas' `averageRating`'s values.
Subsequently, we worked on tuning some hyperparameters for both regressors, using `GridSearchCV` to obtain the values to test. For the `KNN regressor`, we kept the distance metric as the default Euclidean metric and experimented with the `weights` parameter (which determines if all neighbors contribute equally to the prediction or if closer ones have a higher influence) but we eventually left it as uniform (default); and we set `n_neighbors` to 50 (the number of nearest neighbors considered for prediction). Despite these changes, the `KNN regressor` produced fit results nearly identical to those already obtained, showing no significant improvement. Afterward, we attempted to modify some parameters, `min_sample_split`, `min_sample_leaf` and `max_depht` (already seen in the `Decision Tree` for classification, section 3.3) of the `Decision Tree Regressor` using `GridSearchCV` but even in this case, the results obtained in terms of goodness-of-fit of the predictions to the actual values did not change. In fact, the only metric that changed was R^2 which became even closer to 0, while the other two metrics remained unchanged.

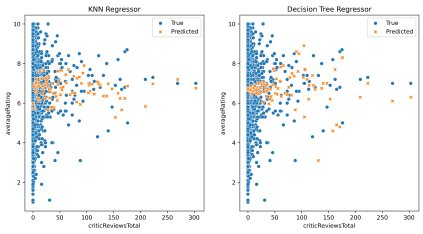


Figure 4.2: Scatter plot with Non linear regressor on test dataset

4.1.3 Multiple Regression

In this section, we used not only `criticReviewsTotal` as the independent variable but also added other variables with which `averageRating` had a slightly higher correlation (though still very close to zero): `awardNominationsExcludeWins`, `TotalCredits` and `awardWins`. Even in the case of multiple regression, the results obtained were unsatisfactory . The linear regressors produced metrics identical to those of simple regression, while the non-linear regressors performed even worse. The R^2 values tended toward -0.5 (-0.19 for the `KNN regressor` and -0.31 for the `Decision Tree regressor`), and the MSE and MAE values slightly increased. We therefore conclude that adding independent variables to our regression model did not result in any improvement in the accuracy of the predictions.

4.1.4 Multivariate Regression

In this final section, we added another target variable, `ratingCount`, alongside `averageRating`, and analyzed which independent variables improved the predictions.As a first attempt, we used the following two features as independent variables to predict the two target variables: `awardNominationsExcludeWins` and `userReviewsTotal`. This choice was guided by the high correlation previously noted during the data understanding phase between `ratingCount` and `userReviewsTotal`. Our results showed an overall improvement for all the regressors considered in the previous sections. The three linear regressors reached the same R^2 value, now explaining 27.9% of the variability in the data (a more satisfactory result compared to the previous cases). However, this improvement came at the cost of a significant increase in the values of MSE and MAE, which reached 22,346,688 and 785, respectively, for `LinearRegressor` and `Ridge`, with slightly lower values for `Lasso`.
This excessive increase in these variables is likely due to the different measurement scale of `ratingCount` and the high dimensionality of the data. Using the two non-linear regressors, we also observed an improvement in the R^2 results, achieving non-negative values. The `KNN regressor` reported a performance of 0.24 (see Figure 4.3 for the predictions on test set), which is close to the results of the linear regressors, while the `Decision Tree regressor` achieved an R^2 of 0.12. Furthermore, a similarity with the linear regressors' results is that there was also an increase in the other two metrics: the MAE was approximately 340 (half of that of the linear regressors), and the MSE reached levels similar to those previously reported for both regressors. Finally, we tested whether adding `totalCredits` (selected due to its correlation with the two target variables) to the independent variables would result in more accurate predictions. However, the regressors' performance worsened compared to the previous case. Specifically, the `decision tree` once again reported a negative R^2, while for the `KNN regressor`, R^2 decreased to 0.20. The linear regressors achieved the same R^2 results as before but showed a slight increase in MAE, which reached approximately 800 for all three methods. The MSE, however, remained similar to the previous results across all regressors.

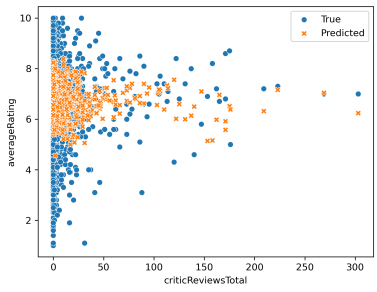


Figure 4.3: Scatter plot of KNN for multivariate regression on test set

Therefore, we can conclude that adding `totalCredits` as an independent variable did not improve the predictive performance of the analyzed regressors.

4.1.5 Final Discussion

The results obtained from these various types of regressions to predict the variable `averageRating` were not at all explanatory of the data variability, and no model produced accurate predictions for this variable. This is justified by the consistently low R^2 values we observed both in the case of simple regressions and multiple regressions.For multivariate regressions, we observed an overall improvement in the performance of all regressors compared to the types of regression previously analyzed. This improvement was likely due to the inclusion of two highly correlated variables (`ratingCount` and `userReviewsTotal`). Despite this, the higher R^2 levels were offset by significantly increased values of MSE and MAE. Consequently, even this type of regression did not yield optimal results for predicting the two target variables.
In conclusion, we acknowledge that using regression to attempt to predict `averageRating` with variables that are weakly correlated with it has resulted in inaccurate predictions and unsatisfactory outcomes.

4.2 Pattern Mining

Before proceeding with the pattern analysis on our dataset, it was necessary to make some modifications to prepare it for the application of Pattern Mining algorithms: `Apriori`, `FP-Growth` and `Association Rules`. We mapped the values of the rating field into ten distinct categories, ranging from "Very Low" to "Outstanding". Additionally, we discretized numerical variables such as `startYear` and `runtimeMinutes` by dividing them into ten bin. For categorical variables, such as `isAdult` and `canHaveEpisodes`, we transformed them into binary categories, such as "No Adult" vs "Is Adult" and "Has Episodes" vs "No Episodes". Furthermore, all other variables were reformatted to be compatible with pattern mining algorithms, ensuring that each value was easily interpretable.

4.2.1 Apriori

To apply the Apriori algorithm, we iterated over different combinations of minimum support (`supp`) and minimum number of items (`zmin`) to determine the optimal parameters. The minimum support values were tested within the range of 10 to 50, while the minimum number of items varied from 2 to the number of columns in the dataframe. We calculated the number of frequent, closed, and maximal itemsets for each combination in order to identify the values that produced the most significant results in terms of associations. At the end of the iteration, the best parameters obtained were a minimum support of 10.0 and a minimum number of items of 2. However, in order to identify more meaningful patterns and reduce the number of overly general rules, we decided to increase the minimum number of items to 4. Although this choice led to a decrease in support (for example, the itemset with the highest support for frequent itemsets, ("No episodes", "No Adult"), had a support of 87.810673) it allowed us to focus on more relevant item combinations. The goal was to obtain more robust and interpretable rules, providing a clearer understanding of the relationships between the attributes in our dataset.

As we can see from the results, in 4.4, the frequent and closed itemsets indicate that the dataset mainly consists of films and video content with no awards and no nominations excluding wins. These two attributes were already identified as strongly correlated, and this combination might suggest that the analyzed content is generally less awarded or poorly recognized. Furthermore, these contents are non-adult and without episodes, which could represent a specific type of content. The maximal itemsets, on the other hand, are more specific and include combinations of attributes such as a duration between 47-58 minutes and geographical localization in the United States.

TYPE OF ITEMSET	ITEMSET	SUPPORT
Frequent	0_awardNominationsExcludeWins, 0_awards, No episodes, No Adult	73.684211
	0_awardNominationsExcludeWins, 0_videos, No episodes, No Adult	72.264864
	0_awards, 0_videos, No episodes, No Adult	72.033382
Closed	0_awardNominationsExcludeWins, 0_awards, No episodes, No Adult	73.684211
	0_awardNominationsExcludeWins, 0_videos, No episodes, No Adult	72.264864
	0_awards, 0_videos, No episodes, No Adult	72.033382
Maximal	(47.0, 58.0]_minutes, ['US'], 0_awardNominationsExcludeWins, 0_awards, 0_videos, No episodes, No Adult	10.087719
	(47.0, 58.0]_minutes, 0_reviewsTotal, 1_regions, 0_criticReviews, 0_awardNominationsExcludeWins, 0_awards, 0_videos, No episodes, No Adult	11.701998
	['US'], 0_reviewsTotal, 1_regions, 0_criticReviews, 0_awardNominationsExcludeWins, 0_awards, 0_videos, No episodes, No Adult	11.671540

Figure 4.4: op 3 itemsets for each type (Frequent, Closed, and Maximal)

To gain a more general understanding of the dataset, we also attempted to divide all the variables into 5 categories. These bins were mapped to values such as "very low", "low", "medium", "high", and "very high". However, the results were not much different from those obtained previously. The patterns observed in the binned data were quite similar to those identified through the

itemset analysis. For example, the "very low" categories for attributes like `totalCredits`, `totalVideos`, `ratingCount`, and `totalImages` consistently appeared across all types of itemsets. Similarly, the maximal itemsets continued to highlight specific combinations, such as certain geographical locations and attributes like `startYear`, `numRegions`, and `runtimeMinutes`, but the general trends remained unchanged. However, it is worth noting that, for the frequent and closed itemsets, the support increased significantly, reaching very high values. This could be due to the binarization into only five bins, which "grouped" the records into broader classes, increasing the support. For the maximal itemsets, however, the support remained lower, not even reaching 17, suggesting that these patterns continue to represent more specific and less common combinations.

4.2.2 FP-Growth

To apply the FP-Growth algorithm, we used the same iterative approach to explore different combinations of minimum support and minimum number of items in order to determine the optimal parameters. The best results were obtained with a minimum support of 10.0 and a minimum number of items equal to 2. For the same reason, we also increased the `zmin` parameter to 4 to ensure that only the most significant itemsets were selected for analysis.

FREQUENT ITEMSET	SUPPORT
0_awards, 0_videos, No episodes, No Adult	72.033382
0_awardNominationsExcludeWins, 0_videos, No episodes, No Adult	72.264864
0_awardNominationsExcludeWins, 0_awards, No episodes, No Adult	73.684211

Figure 4.5: Top 3 itemsets found with the FP-Growth algorithm

The FP-Growth algorithm, which results are shown in 4.5, in confirms the patterns previously identified using the Apriori algorithm. The most frequent itemsets still revolve around combinations where content lacks awards, videos, and episodes, and is not designated for adult audiences.
Moreover, when applying the FP-Growth algorithm on the dataset with variables binned into 5 categories, the patterns did not change significantly. However, as with the Apriori algorithm, the support values increased, likely due to the binarization of the data into broader categories, which grouped records into larger classes.

4.2.3 Association Rules

To apply the association rules, we explored different combinations of minimum support (from 10 to 50) and confidence (from 50 to 100), calculating the number of rules for each combination to identify the most significant ones. We found that the optimal value for confidence was 50, and we set the minimum support to 10. Subsequently, we sorted the results by lift in descending order, as lift helps to identify the most statistically significant associations, highlighting attribute combinations that occur with a higher probability than would be expected under independence.

CONSEQUENT	ANTECEDENT	ABS_SUPPORT	%_SUPPORT	CONFIDENCE	LIFT
(-0.001, 17.0]_minutes	short, No episodes, No Adult	1070	6.518031	0.559039	5.495317
(-0.001, 17.0]_minutes	short, 0_awardNominationsExcludeWins, 0_awards, No episodes	1070	6.518031	0.559039	5.495317
(-0.001, 17.0]_minutes	short, 0_awardNominationsExcludeWins, 0_awards, No episodes, No Adult	1068	6.505848	0.558870	5.493656
(-0.001, 17.0]_minutes	short, 0_awardNominationsExcludeWins, 0_awards, No Adult	1068	6.505848	0.558870	5.493656
(-0.001, 17.0]_minutes	short, 0_awards, No episodes	1151	7.011452	0.558196	5.487033

Figure 4.6: Association Rules

The results of the association rules, in Figure 4.6, reveal some interesting trends about the content in our dataset. The first rule that emerges, with a particularly high lift (5.495), associates the combination of attributes: "short" "0_awardNominationsExcludeWins" and "0_awards" with the content's duration, which falls within the range `(-0.001, 17.0]` minutes. This rule can be interpreted as: "If a content is classified as 'short' (i.e., a brief content, such as a short film) and has not received award nominations (excluding awards won) nor any awards in general, then its duration will be between -0.001 and 17 minutes". This implies that content with these characteristics (short, non-awarded, and without nominations) tends to have a very short duration, defined in our dataset as between 0 and 17 minutes. In other words, these contents are likely to be short films or brief clips. In the first part of the report, we observed a possible correlation between the variable `titleType` and `runtimeMinutes`, specifically that short films (classified as "short") tend to have a very short duration. This hypothesis is now confirmed by the association rule analysis.

Analyzing `consequent`, `confidence`, and `lift`, we observe that `consequent` is primarily correlated with the content's duration, with the lowest bin of `runtimeMinutes` appearing in the first 60 values. This indicates that the duration of the content is closely linked to the associated attributes, particularly for short content. The average `confidence` of around 0.55 suggests that in about 55% of cases, if a content is classified as "short" and has not received any awards or nominations, its duration will fall within the lower 17-minute range. Finally, a `lift` of 5.49 implies that this association is 5.49 times more likely than would be expected if the attributes were independent, confirming the strength and statistical significance of the relationship.

To further deepen the analysis, we applied the same association rule extraction process to the dataset where variables were divided into 5 categories (or bins). Analyzing the first resulting record (by sorting the results in descending order of lift), we observe that the consequent of the rule is "tvEpisode" meaning that the association identifies television episodes. The antecedent includes a combination of attributes such as "Very High" (which mapped to the range (7, 8] of the rating), "US" indicating the country of origin, and several "very low" attributes, including `numRegions`, `runtimeMinutes`, `criticReviewsTotal`, `awardWins`, `totalVideos`, and `totalImages`. This suggests that the episodes in question tend to have a very low number of regions, videos, images, and awards, as well as a short duration. The confidence value of 0.684 indicates that 68.4% of the instances that match the antecedent are indeed classified as "tvEpisode". Finally, a lift of 2.389 (significantly lower than the previous analysis, where we had a lift of around 5.5) implies that the occurrence of this association is 2.39 times more likely than what would be expected under independence, still highlighting a significant relationship between the attributes, albeit with a lower probability compared to the previous results.