UNIVERSITY OF BUCHAREST

FACULTY OF
MATHEMATICS AND
COMPUTER SCIENCE

Big Data

BENCHMARKING CLASSIFICATION MODELS ON MOVIE RATINGS

Project Report

Students

Alexandra-Cristina Nicolae - 407

Roxana-Adela Enășoaie - 411

Teodor-Ioan Coconu - 407

George-Daniel Movilă - 411

Cristian-Alexandru Soare - 411

Project P14

Bucharest, June 2024

Introduction

This project aims to compare five classification algorithms: Random Forest, Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Multinomial Naive Bayes, on an aggregated dataset containing information about movies. The primary goal is to evaluate the performance of these algorithms on the complete, preprocessed dataset. Additionally, the project assesses their performance on three datasets obtained by applying dimensionality reduction techniques: Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Truncated Singular Value Decomposition (TruncatedSVD) on the preprocessed dataset.

The dataset, before preprocessing and data cleaning, includes columns such as: *releaseYear*, *runtimeMinutes*, *genres*, *lifetimeGross*, *averageRating*. The *averageRating* column represents the rating of the movie and will serve as the target variable for classification. The ratings will be mapped into two classes: "bad" and "good" movies, based on the distribution of the ratings.

Gathering the Data

The Official IMDB Repository

The primary source of information for this project is the IMDB (Internet Movie Database) repository, which provides a rich dataset suitable for non-commercial use. The structure of the files and details can be accessed from the IMDB Developer site [3]. For this project, the following, very large, TSV files were used:

1. `title.akas.tsv.gz`
2. `title.basics.tsv.gz`
3. `title.crew.tsv.gz`
4. `title.principals.tsv.gz`
5. `title.ratings.tsv.gz`

To create a comprehensive dataset for the analysis, several steps of feature engineering were performed using these files:

1. **Title Basics**: The `title.basics.tsv.gz` file provides fundamental information about movies, including these features: *isAdult, releaseYear, runtimeMinutes, genres.* This file serves as the backbone of our dataset, providing crucial metadata

for each film. Also, the feature *sameYearTotalMoviesReleased*, which means the number of movies that were released in the same year as the current sample, was calculated based on this file.

2. **Title Ratings**: The `title.ratings.tsv.gz` file includes user ratings for movies. It contains the *averageRating, numVotes* columns. This data is essential for creating the target variable, where movies are classified as "good" or "bad" based on their average ratings.

3. **Title Crew**: The `title.crew.tsv.gz` file lists the directors and writers associated with each movie. This information helps in understanding the impact of the crew's experience on movie ratings. The number of movies directed by a director are derived from this file (the *director_nrOfMovies* feature).

4. **Title Principals**: The `title.principals.tsv.gz` file contains information about the key cast and crew members of each movie, including actors, actresses, and other significant contributors. This data is used to derive features like the number of actors involved in a movie or the number of employees (the *nrOfEmployees, nrOfActors* features).

5. **Title Akas**: The `title.akas.tsv.gz` file provides alternative names for movies and additional release details. This file is used to calculate the total number of different releases for a movie, such as a different release in another language (the *nrOfReleases* feature).

The data from these files was merged to create a unified dataset. The following steps outline the feature engineering process:

- **Data Merging**: The datasets were merged based on common identifiers such as `tconst`, which uniquely identifies each title in the IMDB database.
- **Handling Missing Values**: Missing values were identified and handled appropriately. For instance, missing runtime minutes were excluded from the start.
- **Feature Creation**: New features were created from the existing data, such as `nrOfReleases` (number of releases), `director_nrOfMovies` (number of movies directed by a director), `sameYearTotalMoviesReleased`, and `nrOfActors and nrOfEmployees` (number of actors in the movie and the number of total employees).
- **Target Variable**: The `averageRating` column was transformed into a categorical variable, with ratings mapped into "good" and "bad" categories based on their distribution.

By carefully processing and combining these files, a robust dataset was constructed, providing a solid foundation for further enrichment.

Additional Features

To enhance the dataset, we introduced a new feature called *lifetimeGross*, representing the total domestic gross revenue of the movies. This information was sourced from three different databases:

- **Boxofficemojo Dataset**: The primary source for the *lifetimeGross* feature was the Boxofficemojo dataset [1], which can be accessed at Boxofficemojo. This dataset was concatenated with the IMDB data using the movie titles and release years. Although there were movies with duplicate titles, approximately 12.5k out of 13k titles were unique. Therefore, there are a few different movies with the same title and release year, but the same value for *lifetimeGross* but the number of those was too small.

- **Wikidata**: Another significant dataset was obtained from Wikidata [4], which is a free and open knowledge base that acts as central storage for the structured data of its Wikimedia sister projects including Wikipedia. The data was extracted using the following SPARQL query:

  ```
  SELECT DISTINCT ?item ?imdbID ?domesticGross WHERE {
    ?item wdt:P31 wd:Q11424;
          wdt:P2142 ?domesticGross;
          wdt:P345 ?imdbID.
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[
     AUTO_LANGUAGE],en". }
  }
  ```

  Listing 1: SPARQL query for retrieving data on Wikidata

  This query fetched the domestic gross revenue for movies, identified by their IMDB IDs. The merging process was straightforward as we had the IMDB ID from Wikidata, which enriched the dataset with an additional 3k samples.

- **OMDB API**: The final source was the OMDB API [2], accessible at OMDB API. For movies that were not included in the merged dataset from the previous sources, we made multiple API calls using the available IMDB IDs to fetch the *lifetimeGross* data. This approach added another 700 samples to the dataset, and the merging was again seamless since the API calls were made using the IMDB IDs.

By integrating these external datasets, we significantly enriched the original IMDB dataset with comprehensive financial data, enhancing the robustness and utility of our analysis.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) was conducted to gain insights into the dataset and prepare it for further processing and modeling. The EDA process involved checking for missing values, examining the counts of categorical features, identifying outliers in numerical features, and analyzing the distribution of the target variable before mapping it into two classes.

1. Checking for Missing Values

The initial step in our EDA was to check for missing values in the dataset. After thorough inspection, we found that there were no missing values in any of the columns. This ensured that our dataset was complete and ready for further analysis without the need for imputation or data cleaning related to missing values.

2. Checking the Categorical Feature

We examined the bin counts of the categorical feature *genres*. This feature provides insight into the various genres represented in our dataset. By plotting the bin counts (Figure **1**), we observed that the last five genres are quite unrepresented:

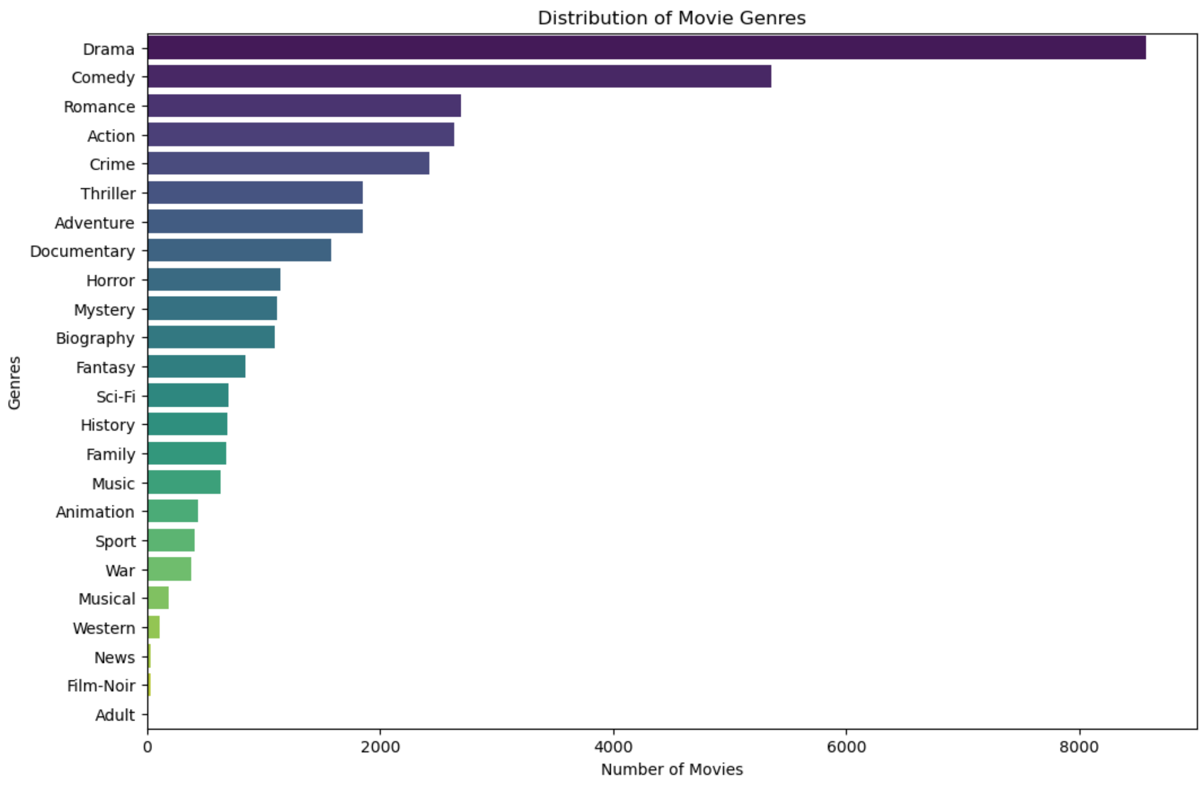

Figure 1: Distribution of Movie Genres

The last five genres have the following numbers of samples:

- Musical: 186
- Western: 113
- News: 27
- Film-Noir: 27
- Adult: 8

3. Checking for Extreme Values

Outliers in numerical features can significantly impact the performance of machine learning models. We identified and analyzed outliers in various features using box plots. Here are the observations:

- **isAdult**: This feature has only 4 values with 1 and the rest are 0s.

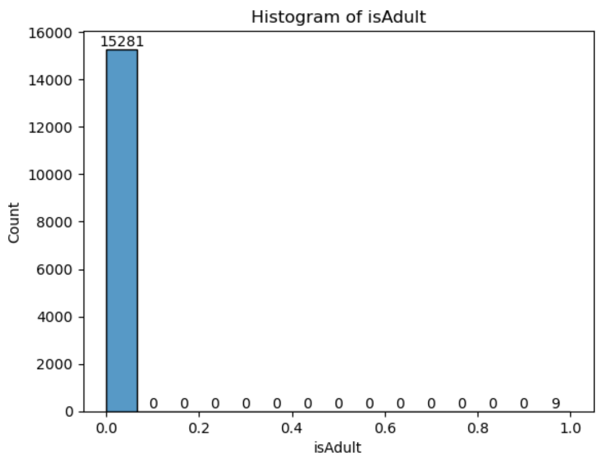

Figure 2: Histogram of isAdult

- **releaseYear**: No unusual values were observed.

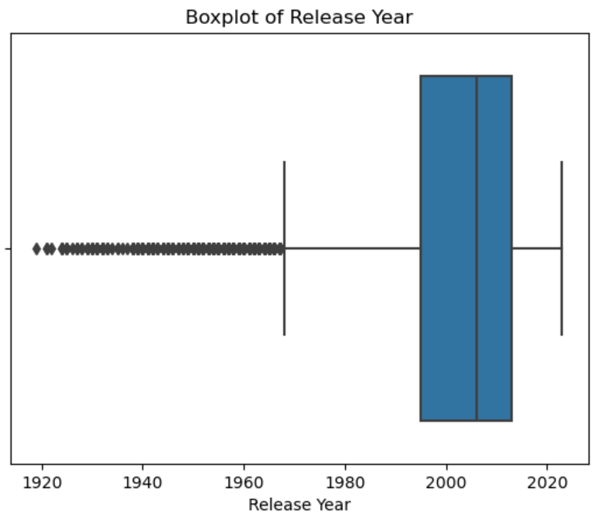

Figure 3: Boxplot of Release Year

- **runtimeMinutes**: There are 14 movies below 50 minutes and 12 movies above 240 minutes, which may seem like anomalies.

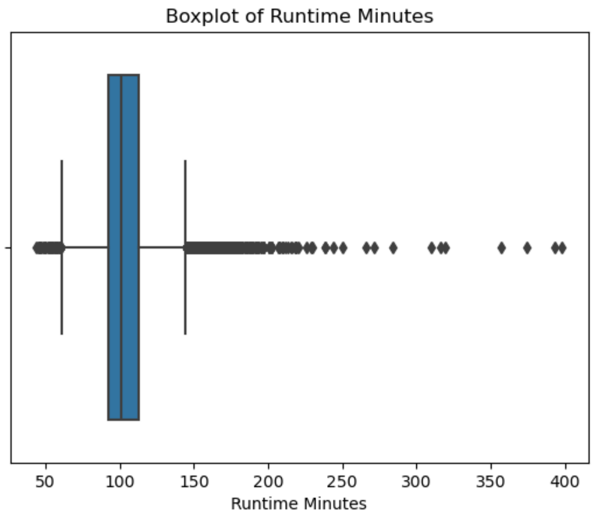

Figure 4: Boxplot of Runtime Minutes

- **lifetimeGross**: There are 58 movies with extreme incomes above $1 billion, which seem like anomalies.

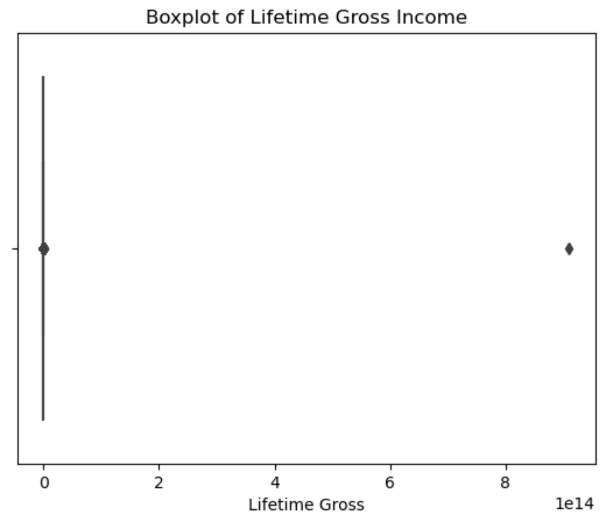

Figure 5: Boxplot of Lifetime Gross

- **nrOfReleases**: There are 57 movies with more than 80 releases, which is odd.

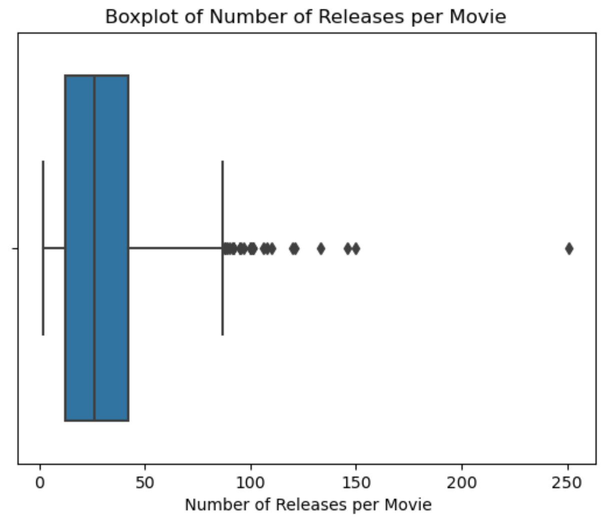

Figure 6: Boxplot of Number of Releases per Movie

- **director_nrOfMovies**: There are 79 movies where the director has directed over 500 movies, which seems too much.

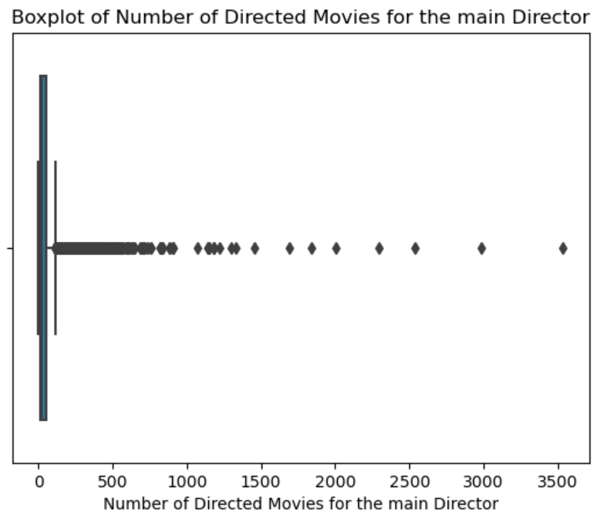

Figure 7: Boxplot of Number of Directed Movies for the Main Director

- **nrOfEmployees**: There are only 3 movies with more than 40 employees.

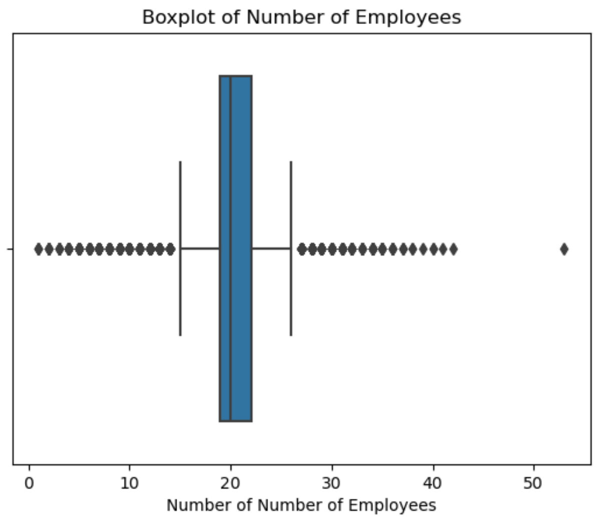

Figure 8: Boxplot of Number of Employees

- **nrOfActors**: Everything seems ok with the number of actors.

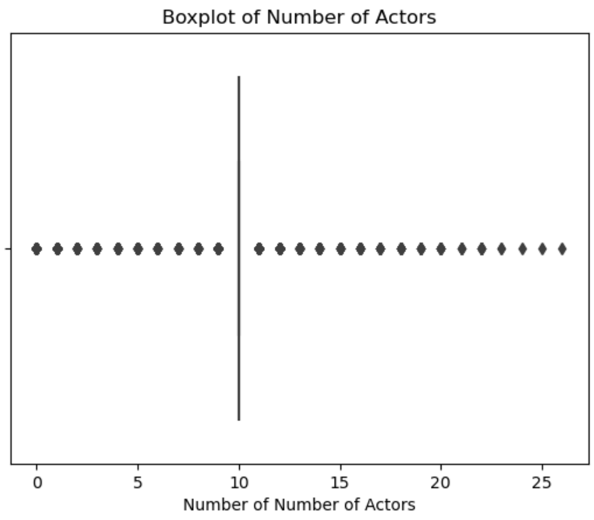

Figure 9: Boxplot of Number of Actors

- **numVotes**: There are 59 movies with extreme values above 1 million votes.

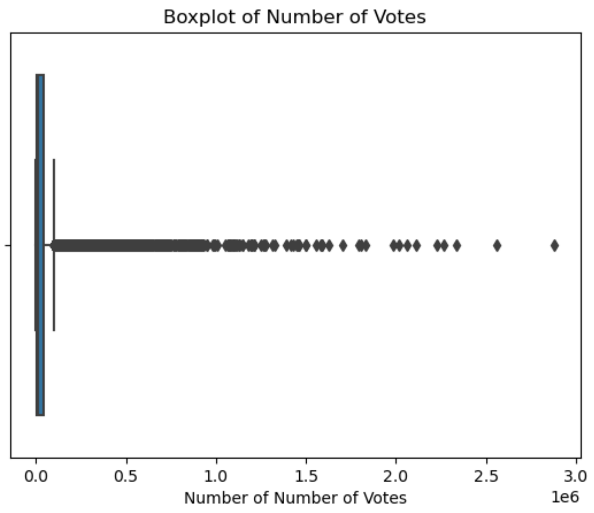

Figure 10: Boxplot of Number of Votes

- **sameYearTotalMoviesReleased**: Everything seems ok with the number of movies released in the same year.

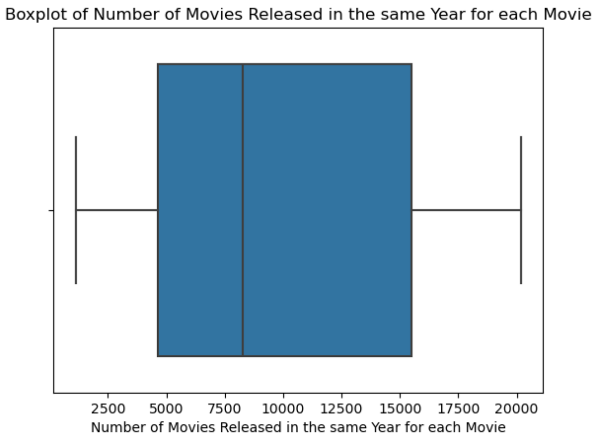

Figure 11: Boxplot of Number of Movies Released in the Same Year for Each Movie

Distribution of the Target Variable

The target variable *averageRating* was analyzed to understand its distribution before mapping it into two classes. This analysis helped in determining the threshold for categorizing movies into ”good” and ”bad” classes. By examining the histogram of *averageRating*, we observed the spread and central tendency of the ratings, which guided our decision on the mapping criteria.

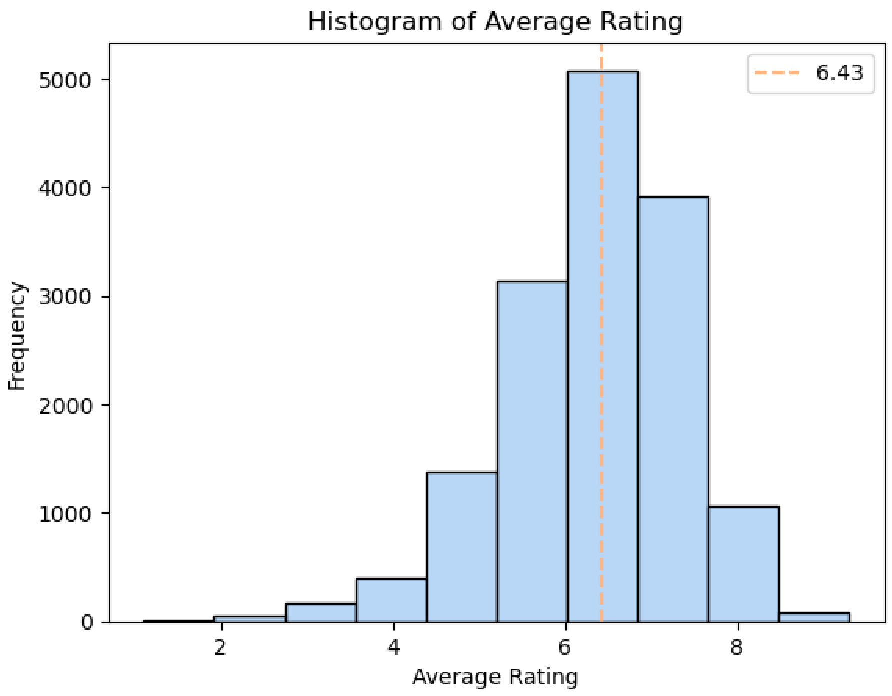

Figure 12: Distribution of Average Ratings

Correcting the Findings

Based on the observations from the Exploratory Data Analysis, several features were cleaned to mitigate the impact of outliers. Below are the plots of the features after cleaning:

Number of Votes

The number of votes was cleaned to remove extreme values above 1 million. Here is the boxplot after cleaning:

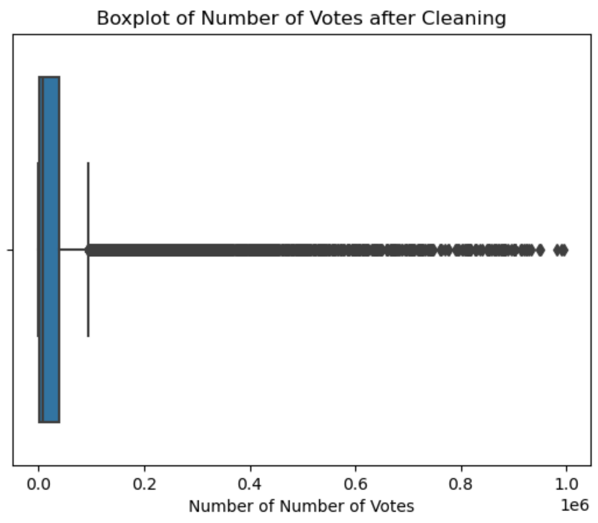

Figure 13: Boxplot of Number of Votes after Cleaning

Number of Releases per Movie

The number of releases per movie was cleaned to remove values above 80 releases. Here is the boxplot after cleaning:

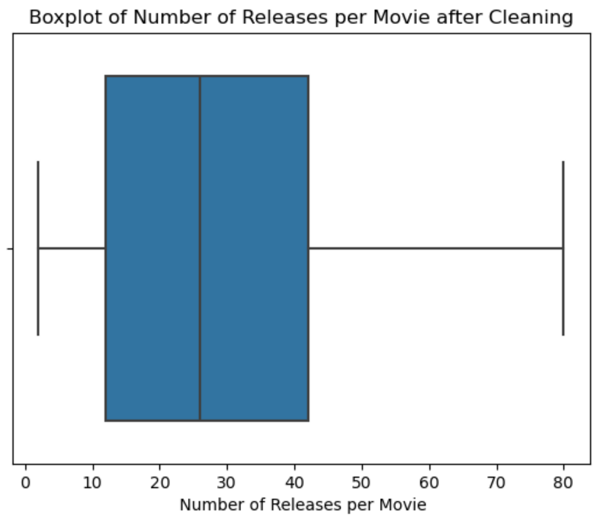

Figure 14: Boxplot of Number of Releases per Movie after Cleaning

Number of Employees

The number of employees was cleaned to remove values above 40. Here is the boxplot after cleaning:

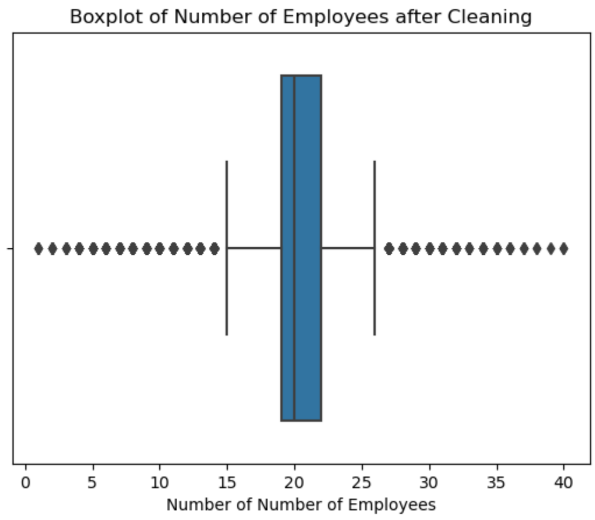

Figure 15: Boxplot of Number of Employees after Cleaning

Lifetime Gross Income

The lifetime gross income was cleaned to remove extreme values above $1 billion. Here is the boxplot after cleaning:

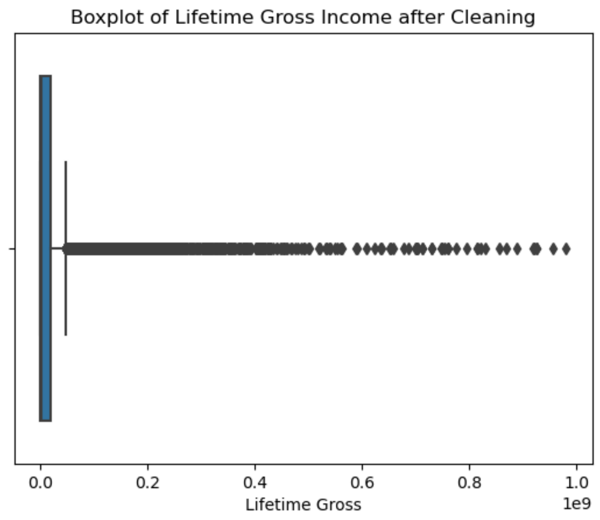

Figure 16: Boxplot of Lifetime Gross Income after Cleaning

Number of Directed Movies for the Main Director

The number of directed movies for the main director was cleaned to remove values above 500. Here is the boxplot after cleaning:

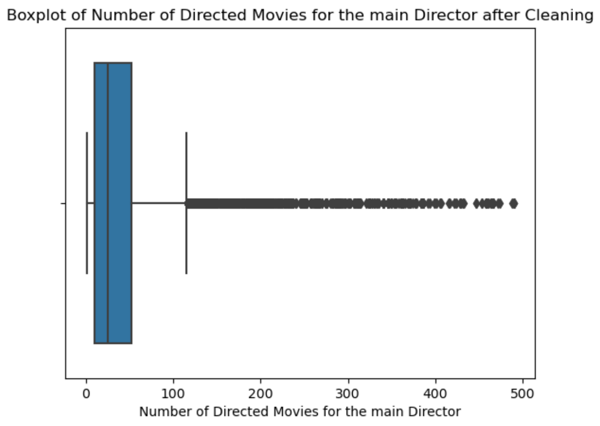

Figure 17: Boxplot of Number of Directed Movies for the Main Director after Cleaning

Runtime Minutes

The runtime minutes feature was also cleaned to remove values above 240 and below 50

Data Preprocessing

In the data preprocessing step, several transformations and encoding techniques were applied to prepare the dataset for modeling. The following steps were followed:

The target variable *averageRating* was mapped into two classes. Movies with an average rating below 6.43 were mapped to 0, indicating "bad" movies, and movies with an average rating of 6.43 or above were mapped to 1, indicating "good" movies. This threshold was determined based on the distribution of the average ratings.

The *genres* feature, which is categorical, was encoded using one-hot encoding. This process involved splitting the genres into individual elements and creating binary columns for each genre. The following steps were performed. This transformation allowed the categorical data to be used in the classification models effectively.

All numerical features were scaled using a MinMaxScaler to ensure they are on a similar scale. The columns that were scaled include:

- *releaseYear*
- *runtimeMinutes*
- *lifetimeGross*
- *nrOfReleases*
- *director_nrOfMovies*
- *nrOfEmployees*
- *nrOfActors*
- *numVotes*
- *sameYearTotalMoviesReleased*

This ensured that the numerical features had a consistent scale, which is essential for the performance of machine learning algorithms.

Dimensionality Reduction

1. Principal Component Analysis

Principal Component Analysis (PCA) is a technique used in statistics to simplify complex datasets. It can be compared to taking a high-resolution photo and reducing its quality just enough so it still shows the main features but with less details, making it easier and faster to work with. The PCA helps us identify the most important features of our data (called "principal components") by finding patterns that explain the most variance in our data. This method is super useful in many fields, like when reducing the number of variables in a dataset while keeping the essential information or in image processing and machine learning to improve efficiency and performance. So by using PCA, we essentially select the features to focus on what matters most, without paying attention to every worthless detail.

Results - PCA

First we applied the PCA using all components. We can see in the figures below that the curb for variance starts to flatten around the value of 18. Because of this, we reduced the dimensionality using a PCA with n_components=18.

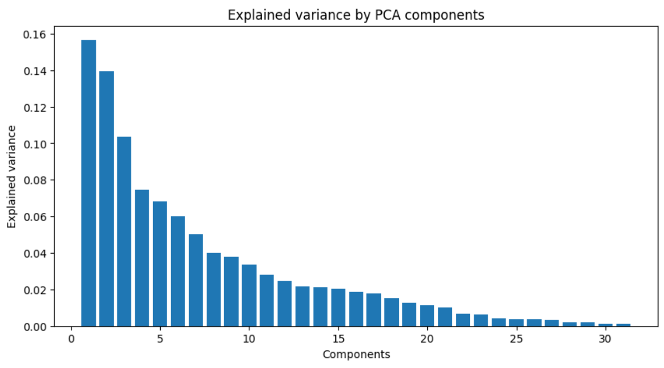

Figure 18: Explained variance by PCA components

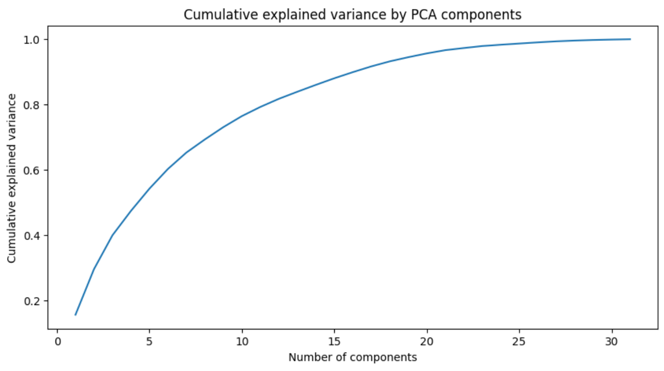

Figure 19: Cumulative explained variance by PCA components

We also outputted a list with the variance explained by each PC. By reading this list we can take only the first X principal components that sum up to a percentage. For us, if we add up the percentages for the 18 best Principal Components we will get 82.16%. This means that the rest of 13 Principal Components only hold 17.84% of Variance Explained.

Principal Component	**Percentage of Variance Explained**
PC9	0.39%
PC16	0.64%
PC31	0.66%
PC11	0.72%
PC19	0.75%
PC20	1.45%
PC25	1.66%
PC29	1.67%
PC14	1.76%
PC6	1.92%
PC23	1.95%
PC21	2.11%
PC18	2.17%
PC12	2.17%
PC2	2.21%
PC30	2.37%
PC1	2.54%
PC28	3.42%
PC26	3.43%
PC4	3.52%
PC17	3.94%
PC8	4.27%
PC5	4.56%
PC27	4.64%
PC15	5.05%
PC22	5.70%
PC13	5.81%
PC24	5.91%
PC10	6.02%
PC7	6.56%
PC3	10.04%

Table 1: Ascending list of explained variance by each principal component

In order to check that the accuracy was not sacrificed during this dimensionality reduction process, we applied the linear regression model and outputted the confusion matrix. As we can see in the matrix below, the accuracy improved to 0.72.

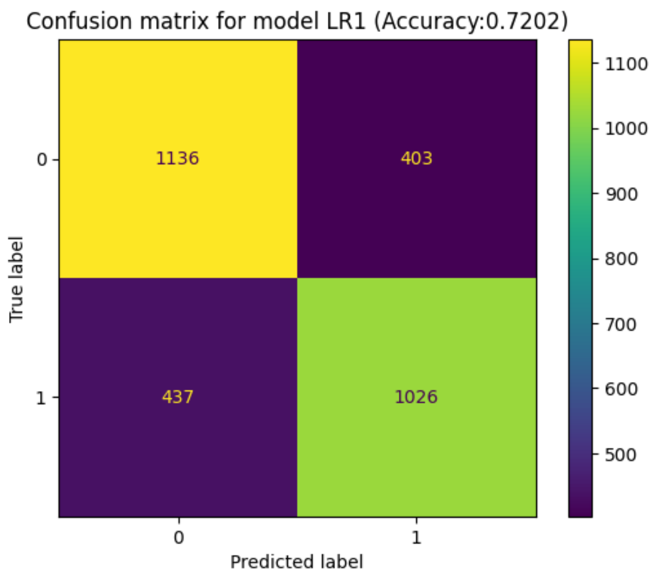

Figure 20: PCA n_components=18

2. Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a statistical method used primarily for the purpose of pattern classification and dimensionality reduction. It works by finding the linear combination of features that separates two or more classes of objects or events as distinctly as possible. It does not work as PCA, which needs directions that best represent the data in a high-dimensional space. LDA is aiming to maximize the separability between known categories. This makes it particularly effective for preparing data for machine learning models where the goal is to predict categorical outcomes.

The features for LDA take values between zero and seven (6.43 being the maximum value). In the figure below we have a visualization for these features related to their frequency.

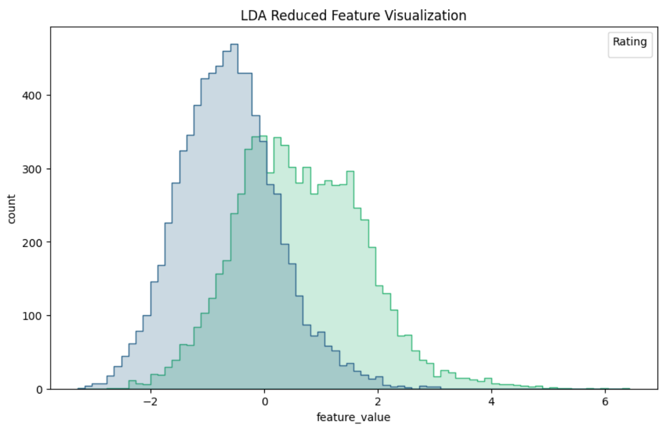

Figure 21: LDA features visualization

Considering the definition for LDA, the method fits our purpose the best as we have only two categorical classes as target. This aspect can also be seen in the results obtained after applying Logistic Regression on the reduced dataset. The results are the best we got using this model on each reduced dataset with an accuracy of 0.74.

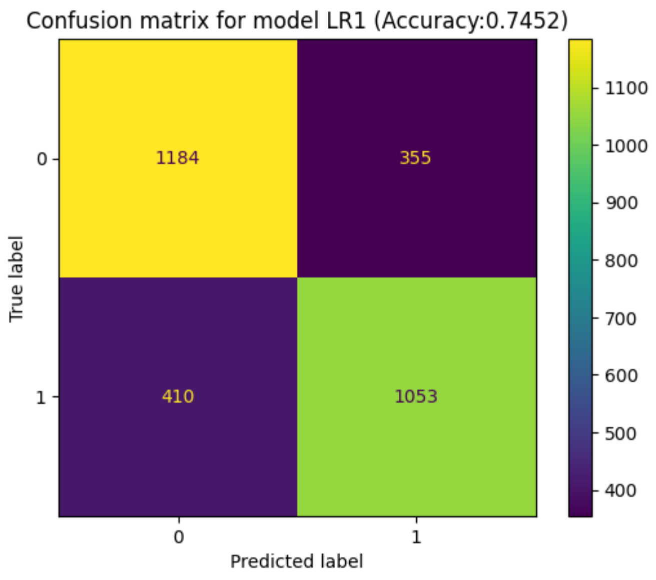

Figure 22: LDA results

3. Truncated SVD

Truncated Singular Value Decomposition (SVD) is a dimensionality reduction method based on statistics. It is an adaptation of a more general method known as Singular Value Decomposition. SVD is a matrix factorization technique that decomposes a matrix in three other matrices, where one represents the data's orthogonal row space, another the orthogonal column space, and the third a diagonal matrix of singular values that provides a measure of the importance of each dimension.

Truncated SVD, specifically, limits the number of singular values to consider by truncating or cutting off the less significant ones. This results in a compact representation of the original data.

As observed in the matrix below, our Logistic Regression model got a bad accuracy compared to the other dimensionality reduction techniques.

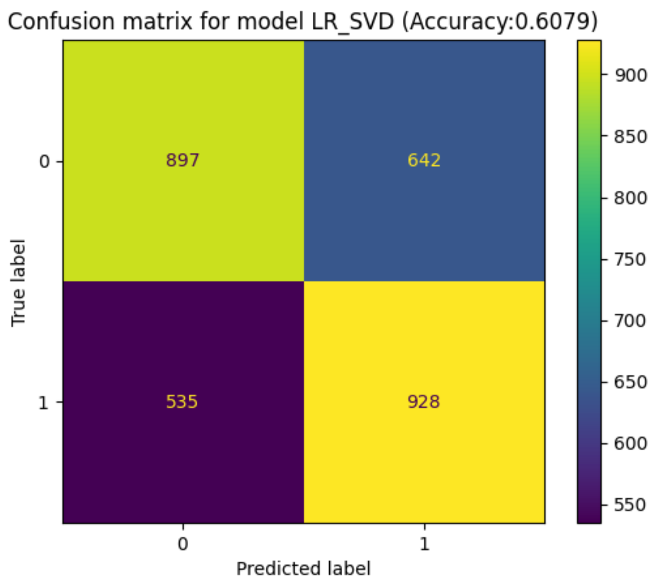

Figure 23: Truncated SVD results

4. T-distributed Stochastic Neighbor Embedding

This dimensionality reduction technique was used only as an experiment. We didn't use it further in our project.

T-distributed Stochastic Neighbor Embedding (t-SNE) is a technique used in machine learning that is primarily used for visualizing high-dimensional data. It's a dimensionality reduction technique that transforms high-dimensional spaces into two or three dimensions, which can then be easily visualized on a graph. t-SNE helps to reduce the complexity by projecting the data into a two or three-dimensional space where similar data points cluster together and dissimilar ones are placed further apart. This visualization allows for an intuitive grasp of data structures and patterns that might not be apparent in the high-dimensional space. t-SNE is particularly useful in fields like bioinformatics, image processing, and natural language processing, where it's essential to uncover hidden structures in large volumes of data. Its ability to preserve local structures and separate distinct clusters makes it a go-to method for exploratory data analysis and preliminary inspections of dataset characteristics.

For this dimensionality reduction technique we tried using different perplexities, as we can see in the figure below.

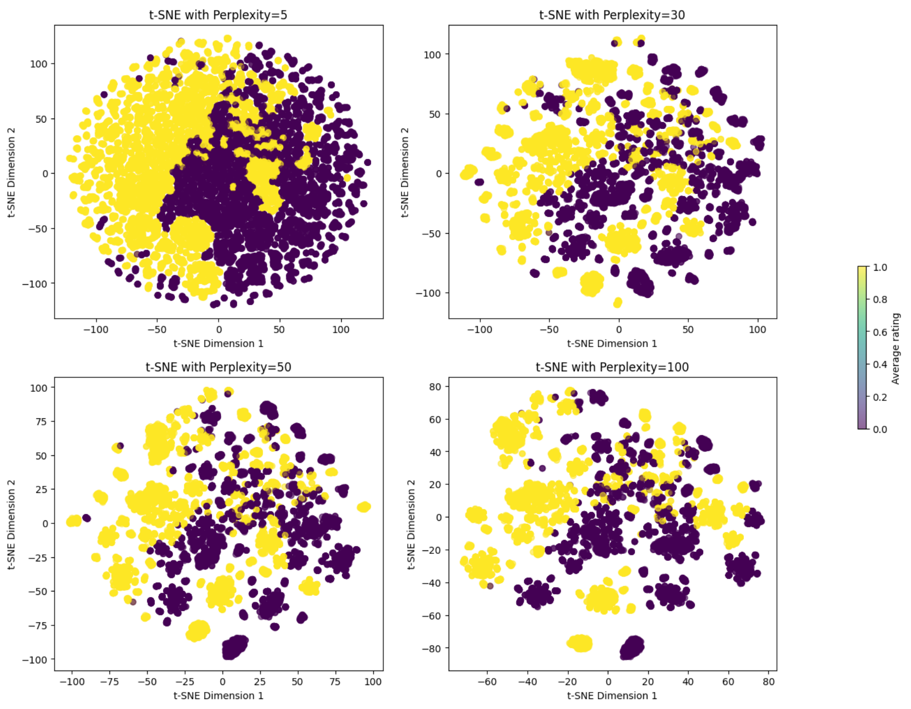

Figure 24: t-SNE with different perplexities

We can notice that for perplexity=50 the clusters become distinguishable, so we will continue with this value.

We decided to use the Barnes-Hut Method instead of the Exact Method because the Barnes-Hut Method significantly reduces the computational complexity from O(n^2) to O(n*logn). This was something we also experienced, because the Exact Method was taking a lot of time to run.

The results for this dimensionality reduction method were not as good as for PCA. The accuracy is also sacrificed because of the Barnes-Hut Method, which is a trade-off we assumed for removing the complexity of the algorithm.

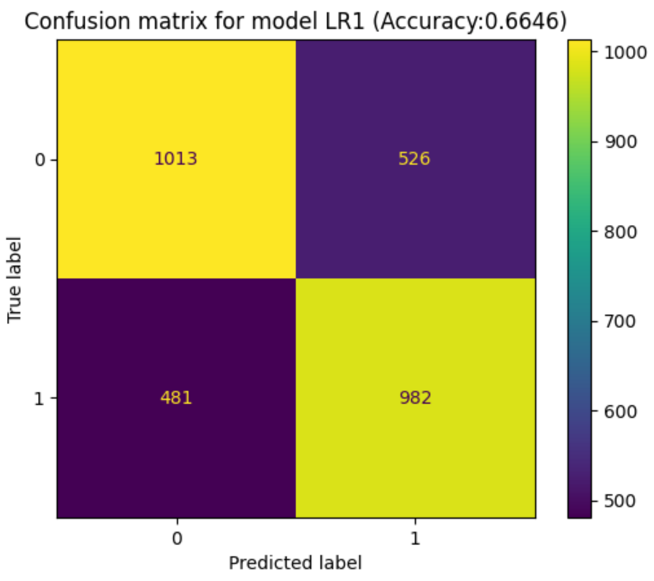

Figure 25: t-SNE perplexity=50

Model 1: Logistic Regression

The Logistic regression algorithm was employed due to its simplicity and effectiveness in handling binary and multiclass classification problems. As we tackle a multiclass classification problem, the multinomial logistic regression variant was utilized. The algorithm employs the sigmoid function to map predicted values to probabilities, thus assigning a given point to a certain class. This section details the implementation, optimization, and evaluation of the model for classifying movie ratings into three distinct categories. The focus is on the methodological approach taken to enhance the model's performance through various hyperparameter tuning techniques.

Methodology

Hyperparameter Optimization

To optimize the logistic regression model's performance, multiple hyperparameter tuning strategies were employed. These included traditional grid search with cross-validation, stratified K-fold validation, and advanced optimization using the Optuna framework.

Baseline Model

The baseline logistic regression model was trained using default parameters provided by Scikit-learn's `LogisticRegression` class. This model served as a reference point for evaluating the impact of subsequent optimizations.

Grid Search Cross-Validation
Grid search cross-validation (GridSearchCV) was employed to identify optimal hyperparameters by exhaustively searching over a specified parameter grid. The parameter grid for GridSearchCV included variations in penalty types, regularization strength, maximum iterations, and solver algorithms:

```
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'max_iter': [10, 100, 200, 500, 1000, 5000],
    'solver': ['liblinear', 'saga']
}
```

Stratified K-Fold Validation
To ensure robust model evaluation, stratified K-fold validation was utilized. This method splits the dataset into K equally sized folds while preserving the class distribution in each fold, thus providing a more reliable estimate of model performance.

Optuna Optimization
Optuna, a hyperparameter optimization framework, was used for more sophisticated tuning. The optimization process was conducted in multiple rounds:

1. **First Optuna Tuning**: Initial optimization with a defined set of hyperparameters, focusing on `penalty`, `C`, `max_iter`, and `solver`.

    - **Penalty**: {`l1`, `l2`}
    - **C**: [0.01, 10.0] (log-uniform distribution)
    - **Max Iterations**: [100, 1000]
    - **Solver**: {`liblinear`, `saga`}

    The following visualizations depict the optimization process and the importance of various hyperparameters:

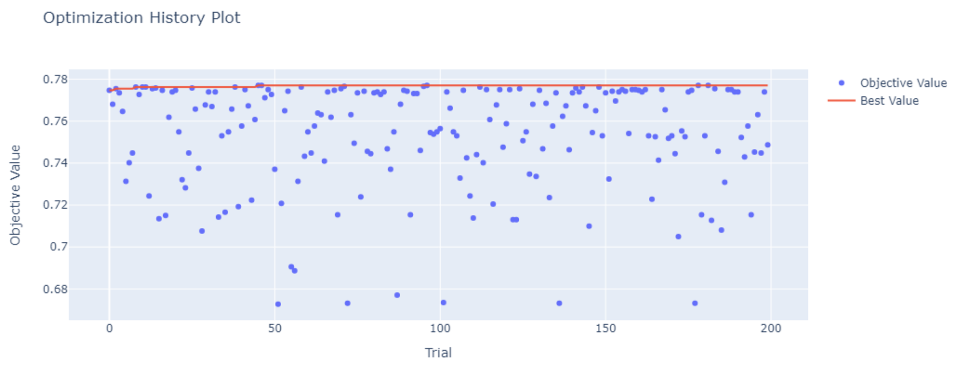

Figure 26: Optimization History Plot

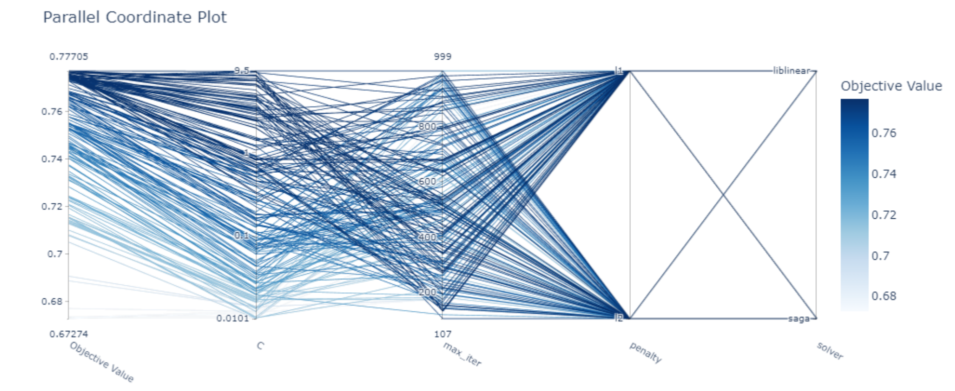

Figure 27: Parallel Coordinate Plot

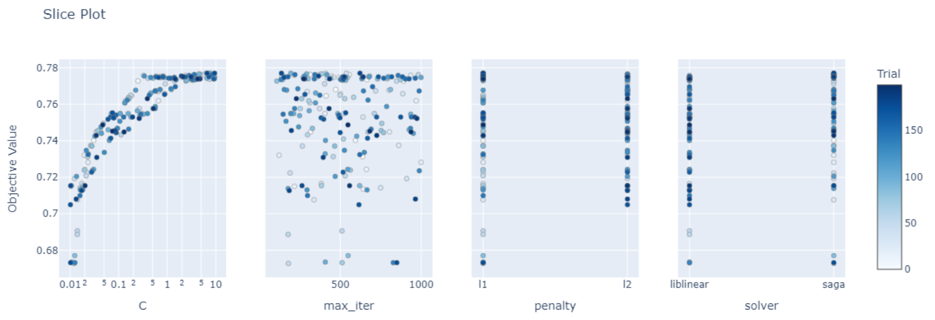

Figure 28: Slice Plot

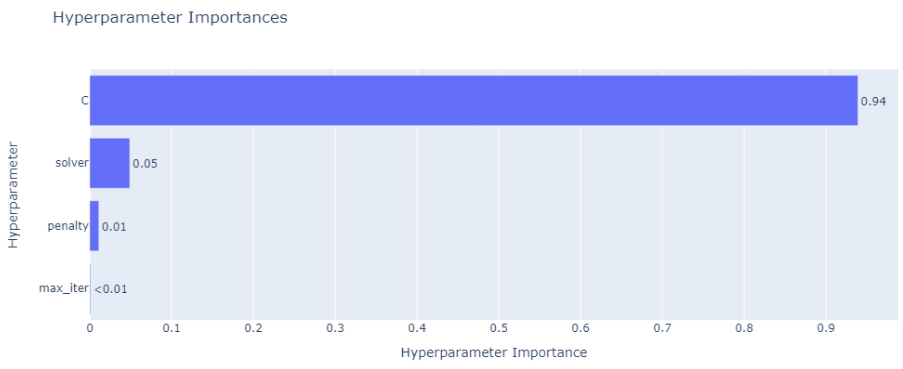

Figure 29: Hyperparameter Importance

The Optimization History Plot (Figure 26) confirms the stability and convergence of the tuning process, while the Parallel Coordinate Plot (Figure 27) and Slice Plot (Figure 28) identify the key hyperparameters and their optimal ranges. The Hyperparameter Importance plot (Figure 29) clearly establishes that C is the most influential hyperparameter, significantly impacting the model's objective value. Consequently, we focus focusing on optimizing C.

2. **Second Optuna Tuning**: Expanded the range for `C` and introduced the `elasticnet` penalty, which is compatible *only* with the `saga` solver and includes an `l1_ratio` parameter.

   - **Penalty**: {`elasticnet`}
   - **C**: [0.01, 100.0] (log-uniform distribution)
   - **L1 Ratio**: [0, 1] (uniform distribution)

3. **Third Optuna Tuning**: Adjusted the `C` interval to a uniform distribution, given its critical importance in model performance.

   - **Penalty**: {`elasticnet`}
   - **C**: [0.01, 100.0] (uniform distribution)
   - **L1 Ratio**: [0, 1] (uniform distribution)

Results

Model Performance Evaluation

The performance of the logistic regression models was evaluated using standard metrics: accuracy, precision, recall, and F1 score. The evaluation metrics for each model variant are presented in the following table:

Model	Accuracy	Precision	Recall	F1 Score	Weighted F1
LR_baseline_untuned	0.7418	0.7421	0.7418	[0.758, 0.724]	0.7413
LR_baseline_CV1_best	0.7468	0.7472	0.7468	[0.763, 0.729]	0.7463
LR_baseline_skf_best	0.7465	0.7468	0.7465	[0.762, 0.728]	0.7460
LR_baseline_OPT1	0.7475	0.7479	0.7475	[0.764, 0.729]	0.7469
LR_baseline_OPT2	0.7468	0.7472	0.7468	[0.763, 0.729]	0.7463
LR_baseline_OPT3	0.7468	0.7472	0.7468	[0.763, 0.729]	0.7463

Table 2: Model Performance Metrics

As it can be seen, the best model was found during the first Optuna tuning.

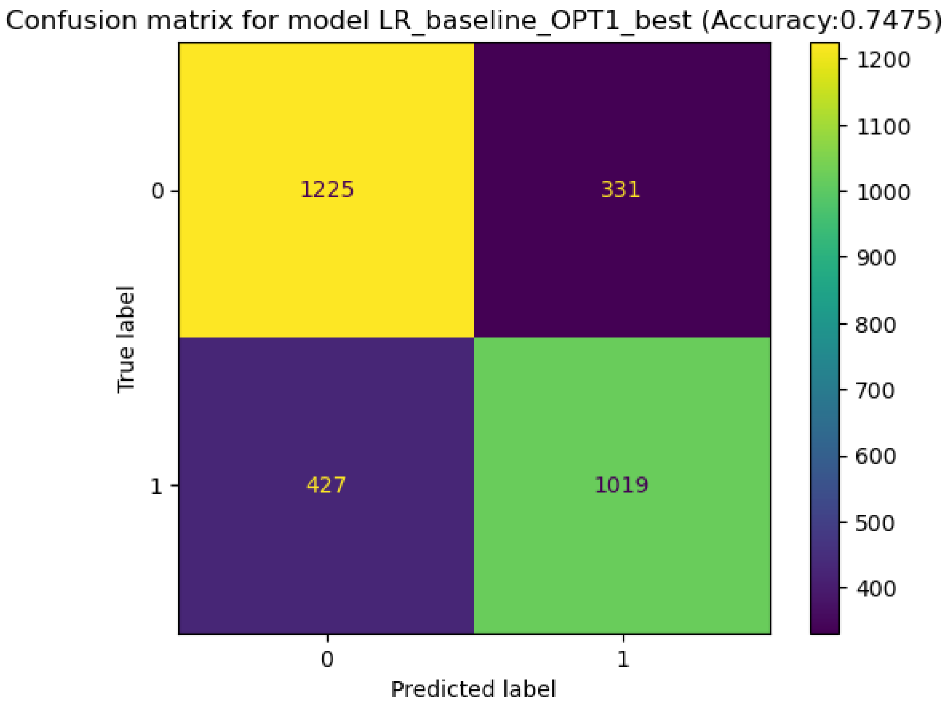

Figure 30: Confusion matrix for model LR_baseline_OPT1_best

The confusion matrix in Figure 30 indicates that the model performs well in predicting the '1' class, with 1019 correct predictions. However, it struggles with the '0' class, showing 331 misclassifications. The model also has errors in predicting the '1' class, with 427 instances misclassified as '0'. The model's hyperparameter configuration is shown below:

Best Hyperparameters:
{ `penalty`: 'l1', `C`: 1.0653730304965756, `max_iter`: 963, `solver`: 'saga' }

The optimized logistic regression models demonstrated significant improvements over the baseline model. The utilization of grid search, stratified K-fold validation, and advanced hyperparameter tuning with Optuna yielded substantial performance enhancements. Among the optimized models, LR_baseline_OPT1_best and achieved the highest accuracy, precision, recall, and F1 scores, indicating their superior performance.

Logistic Regression with Dimensionality Reduction The following table shows the performance achieved using all the presented tuning methods but with reduced features:

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Weighted F1**
LR_PCA_untuned	0.7212	0.7211	0.7212	[0.731, 0.711]	0.7211
LR_PCA_CV1	0.7205	0.7205	0.7205	[0.730, 0.710]	0.7204
LR_PCA_skf	0.7205	0.7205	0.7205	[0.730, 0.710]	0.7204
LR_PCA_OPT1	0.7268	0.7268	0.7268	[0.737, 0.716]	0.7267
LR_PCA_OPT2	0.7272	0.7272	0.7272	[0.738, 0.716]	0.7270
LR_PCA_OPT3	0.7219	0.7218	0.7219	[0.731, 0.712]	0.7218
LR_LDA_untuned	0.7435	0.7436	0.7435	[0.755, 0.731]	0.7433
LR_LDA_CV1	0.7435	0.7436	0.7435	[0.755, 0.731]	0.7433
LR_LDA_skf	0.7425	0.7429	0.7425	[0.755, 0.728]	0.7421
LR_LDA_OPT1	0.7445	0.7450	0.7445	[0.758, 0.730]	0.7441
LR_LDA_OPT2	0.7435	0.7436	0.7435	[0.755, 0.731]	0.7433
LR_LDA_OPT3	0.7435	0.7436	0.7435	[0.755, 0.731]	0.7433
LR_SVD_untuned	0.6079	0.6092	0.6079	[0.604, 0.612]	0.6078
LR_SVD_CV1	0.6079	0.6092	0.6079	[0.604, 0.612]	0.6078
LR_SVD_skf	0.6079	0.6092	0.6079	[0.604, 0.612]	0.6078
LR_SVD_OPT1	0.6079	0.6092	0.6079	[0.604, 0.612]	0.6078
LR_SVD_OPT2	0.6079	0.6092	0.6079	[0.604, 0.612]	0.6078
LR_SVD_OPT3	0.6079	0.6092	0.6079	[0.604, 0.612]	0.6078

Table 3: Model Performance Metrics

Based on the results presented in the tables and the performance metrics, we can draw several conclusions regarding the effectiveness of logistic regression models enhanced by dimensionality reduction techniques.

Impact of Dimensionality Reduction

- **PCA** reduced models showed a slight decrease in performance compared to the baseline models without dimensionality reduction. For instance, the best PCA-tuned model (LR_PCA_OPT2) achieved an accuracy of 0.7272, which is lower than the best untuned baseline model. This suggests that while PCA can help in reducing the feature space, it may also lead to loss of information critical for classification in this context.

- **LDA** reduced models performed better than PCA-reduced models and were closer to the performance of the baseline logistic regression models. The best LDA-tuned model (LR_LDA_OPT1) achieved an accuracy of 0.7445, which is very close to the baseline logistic regression model's performance. LDA seems to be more effective than PCA in retaining the discriminative information necessary for classification.
- **Truncated SVD** reduced models showed significantly lower performance compared to both the baseline and the other dimensionality reduction techniques. The accuracy for SVD-reduced models remained constant at 0.6079 across all tuning methods. This indicates that truncated SVD may not be suitable for this particular classification problem, possibly due to its inability to retain sufficient variance in the data.

In summary, while hyperparameter optimization significantly improved the logistic regression model's performance, the choice of dimensionality reduction technique played a crucial role. LDA emerged as the most effective method for this classification problem, while truncated SVD was the least effective.

Model 2: Random Forest

Random Forest is an ensemble learning method widely used for both classification and regression tasks. It belongs to the family of decision tree-based algorithms and operates by constructing a multitude of decision trees during training and outputting the mode (classification) or mean prediction (regression) of the individual trees. Random Forest is known for its robustness, scalability, and ability to handle high-dimensional data.

Key Concepts

1. Decision Trees
    - Random Forest is built on the foundation of decision trees, which are hierarchical structures that recursively partition the data based on feature values.
    - Each internal node represents a decision based on a specific feature, and each leaf node represents a class (in classification) or a predicted value (in regression).
2. Ensemble Learning

- Random Forest employs an ensemble of decision trees, combining their outputs to improve overall accuracy and generalization.
- The ensemble approach helps mitigate overfitting and enhances the model's stability.

3. Randomization
   - The ”random” in Random Forest comes from the introduction of randomness during both tree construction and prediction.
   - Random subsets of the data (bootstrap samples) and random subsets of features are used to train each tree.
4. Voting or Averaging
   - For classification tasks, the final prediction is determined by a majority vote of the individual tree predictions.
   - For regression tasks, the final prediction is the average of the individual tree predictions.

Limitations

1. Lack of Interpretability
   - The ensemble nature of Random Forests makes it challenging to interpret individual trees and understand the decision-making process.
2. Computational Complexity
   - Training multiple decision trees can be computationally expensive, especially for large datasets.

Results

Results with the default parameters were:

- Accuracy: 0.7724
- F1 score: [0.78351823 0.76026676]

The model was fine-tuned using GridSearchCV and the parameters with the highest score are as follows: **{'bootstrap': True, 'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}**. The results of this optimization are presented in the table below:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.77	0.79	0.78	1556
1	0.77	0.75	0.76	1446
Accuracy	0.7728			
Macro Avg	0.77	0.77	0.77	3002
Weighted Avg	0.77	0.77	0.77	3002
Best Accuracy	0.7728			
Average Classification Report	**Precision:** 0.7728 **Recall:** 0.7720 **F1-Score:** 0.7720 **Accuracy:** 0.7728			

Table 4: RF - Best and Average Classification Reports

Next, we conducted a comprehensive evaluation of the model using **alternative-methods**. These alternative methods imply that we take the optimized models (found during the grid search), sort them by average classification report, and make predictions to find the best accuracy. The best models found with alternative methods are presented below:

Place	**Parameters**	**Accuracy**
	bootstrap, max_depth, min_samples_leaf, min_samples_split, n_estimators	
0	True, 30, 1, 10, 200	0.7728
1	True, 30, 1, 10, 200	0.7726
2	True, 30, 1, 10, 200	0.7725
3	True, 30, 1, 10, 200	0.7724
4	True, 30, 1, 10, 200	0.7723
5	True, 30, 1, 10, 200	0.7722
6	True, 30, 1, 10, 200	0.7721
7	True, 30, 1, 10, 200	0.7720
8	True, 30, 1, 10, 200	0.7719
9	True, 30, 1, 10, 200	0.7718

Table 5: RF - Top 10 Hyperparameter Configurations and Test Accuracy

As expected, running the validation on the same training data has an accuracy of 0.99. This is happening because we have a complex model, RandomForest, that makes decisions based on some features - we have a small number of 35 features. If we minimize the number of decisions/ the maximum depth, the accuracy is descending, as shown in Figure 31.

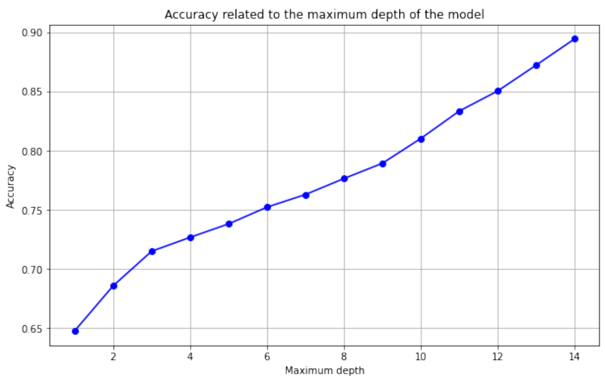

Figure 31: RF - MaxDepth vs Accuracy

Next, we conducted a comprehensive evaluation of the model on the dataset through 100 iterations with different random splits. The model is then trained and assessed in each iteration, considering distinct random seeds for data splitting. We tracked the best-performing model based on accuracy and classification report metrics across all iterations. The best classification report is shown in the table below.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.79	0.81	0.80	1521
1	0.80	0.77	0.79	1481
Accuracy	0.7941			
Macro Avg	0.79	0.79	0.79	3002
Weighted Avg	0.79	0.79	0.79	3002
Best Accuracy	0.7941			
Average Classification Report	**Precision:** 0.7945 **Recall:** 0.7939 **F1-Score:** 0.7940 **Accuracy:** 0.7941			

Table 6: RF - Best and Average Classification Reports using shuffled search

The confusion matrix of the best model is presented in Fig. 32. It indicates that the model performs well in predicting Class 1, but has significant difficulties in predicting Class 0 and some confusion between Class 1 and Class 2.

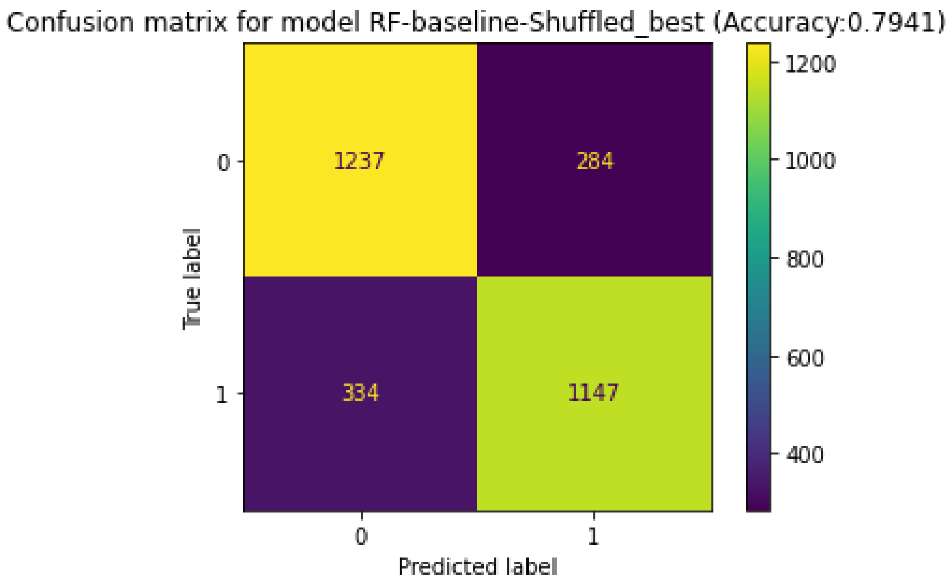

Figure 32: Confusion Matrix for Best RF

The results obtained with different dimensionality reduction methods are presented below:

Model	Accuracy	Precision	Recall	F1 Score	Weighted F1
RF_PCA_GS_CV1	0.7352	0.7355	0.7352	[0.748, 0.721]	0.7348
RF_PCA_alternatives	0.7405	0.7408	0.7405	[0.753, 0.727]	0.7402
RF_LDA_GS_CV1	0.7318	0.7318	0.7318	[0.741, 0.722]	0.7317
RF_LDA_alternatives	0.7342	0.7341	0.7342	[0.743, 0.724]	0.7341
RF_SVD_GS_CV1	0.6209	0.6246	0.6209	[0.603, 0.637]	0.6198
RF_SVD_alternatives	0.6239	0.6280	0.6239	[0.605, 0.641]	0.6226

Table 7: Random Forest Model Performance Metrics with Different Dimensionality Reduction Methods

The comparison of different dimensionality reduction techniques combined with Random Forest highlights distinct performance variations. Principal Component Analysis, when paired with Random Forest, demonstrated improved accuracy and F1 scores in alternative configurations compared to GridSearchCV. Linear Discriminant Analysis also showed slight performance enhancements with alternative methods, although the gains were marginal. Singular Value Decomposition, however, resulted in significantly lower performance metrics, indicating its unsuitability for this dataset with Random Forest. Overall, the choice of dimensionality reduction method substantially influences the efficacy of Random Forest, with PCA and LDA proving more effective than SVD.

Model 3: Multinomial Naive Bayes

Multinomial Naive Bayes is a probabilistic learning method primarily used for text classification and other categorical data tasks. It operates based on the Bayes theorem and assumes that features are conditionally independent given the class. This model is known for its simplicity, efficiency, and effectiveness in dealing with high-dimensional data.

Key Concepts

1. **Naive Bayes Theorem**
   - Multinomial NB relies on the Bayes theorem, which calculates the probability of a class given a set of features.
   - The ”naive” assumption is that all features are independent of each other, which simplifies the computation.
2. **Categorical Data Handling**
   - This model is particularly well-suited for text classification problems where features represent the frequency of words in a document.
3. **Parameter Tuning**
   - The primary parameters for Multinomial NB are `alpha` (smoothing parameter) and `fit_prior` (whether to learn class prior probabilities).

Baseline

Results Results with the default parameters were:

- **Accuracy**: 0.6762
- **F1 Score**: [0.7204, 0.6155]

The model was fine-tuned using GridSearchCV and the parameters with the highest score are as follows:

- `alpha`: 0.5
- `fit_prior`: False

Top 6 parameters (obtained with GridSearchCV) and respective test accuracy are shown in the table below.

Then, we took the best parameters and made the predictions:

- **Accuracy achieved for best parameters**: 0.7016

Place	Alpha	Fit Prior	Accuracy
1	0.5	False	0.6814
3	0.75	False	0.6814
5	1.0	False	0.6814
0	0.5	True	0.6766
4	0.75	True	0.6766
2	1.0	True	0.6763

Table 8: Multinomial NB - Top 10 Hyperparameter Configurations and Test Accuracy

Running the validation on the same training data resulted in an accuracy of 0.99. This high accuracy is due to the model overfitting the training data.

Next, we conducted a comprehensive evaluation of the model on the dataset through 100 iterations with different random splits. The model was trained and assessed in each iteration, considering distinct random seeds for data splitting. We tracked the best-performing model based on accuracy and classification report metrics across all iterations. The best classification report is shown in the table below.

Class	Precision	Recall	F1-Score	Support
0	0.67	0.81	0.74	964
1	0.75	0.59	0.66	923
Accuracy	0.70			1887
Macro Avg	0.71	0.70	0.70	1887
Weighted Avg	0.71	0.70	0.70	1887

Table 9: Best Classification Report for Multinomial NB

Best Accuracy: 0.7016

Average Classification Report:

- Precision: 0.6856
- Recall: 0.6718
- F1-Score: 0.6681
- Accuracy: 0.6761

The confusion matrix of the best model is presented in Figure 33. It indicates that the model performs well in predicting Class 0 but has some difficulties in predicting Class 1 and some confusion between the two classes.

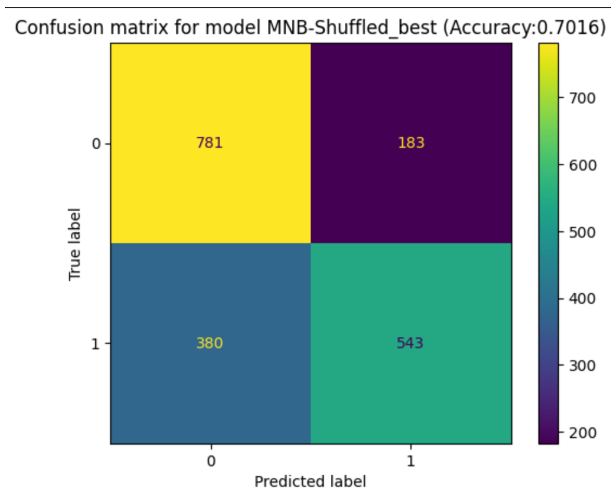

Figure 33: Confusion Matrix for Multinomial NB

The evaluation demonstrates that Multinomial Naive Bayes, while effective and simple, may require further tuning or feature engineering to improve performance, especially in cases where the naive independence assumption does not hold.

Multinomial NB with PCA

Results Results with the default parameters after applying PCA were:

- **Accuracy**: 0.5539
- **F1 Score**: [0.5956, 0.5028]

The model was fine-tuned using GridSearchCV and the parameters with the highest score are as follows:

- `alpha`: 1.0
- `fit_prior`: False

Top 6 parameters (obtained with GridSearchCV) and respective test accuracy are shown in the table below.

Place	Alpha	Fit Prior	Accuracy
5	1.0	False	0.5896
1	0.5	False	0.5896
3	0.75	False	0.5896
2	0.75	True	0.5895
4	1.0	True	0.5895
0	0.5	True	0.5894

Table 10: Multinomial NB with PCA - Top 10 Hyperparameter Configurations and Test Accuracy

Then, we took the best parameters and made the predictions:

- **Accuracy achieved for best parameters**: 0.6885

Running the validation on the same training data resulted in an accuracy of 0.99, indicating overfitting.

Next, we conducted a comprehensive evaluation of the model on the dataset through 100 iterations with different random splits. The model was trained and assessed in each iteration, considering distinct random seeds for data splitting. We tracked the best-performing model based on accuracy and classification report metrics across all iterations. The best classification report is shown in the table below.

Class	Precision	Recall	F1-Score	Support
0	0.66	0.80	0.72	1539
1	0.73	0.57	0.64	1463
Accuracy	0.69			3002
Macro Avg	0.70	0.69	0.68	3002
Weighted Avg	0.70	0.69	0.68	3002

Table 11: Best Classification Report for Multinomial NB with PCA

Best Accuracy: 0.6885
Average Classification Report:

- Precision: 0.6802
- Recall: 0.6702
- F1-Score: 0.6661
- Accuracy: 0.6712

The confusion matrix of the best model is presented in Figure 34. It indicates that the model has improved performance with PCA but still faces challenges in correctly classifying instances of Class 1.

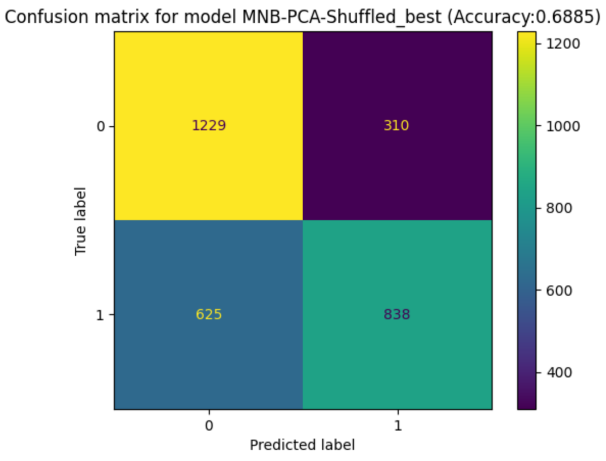

Figure 34: Confusion Matrix for Multinomial NB with PCA

The evaluation shows that applying PCA before Multinomial Naive Bayes can help reduce dimensionality and improve model performance to some extent, but further tuning and feature engineering might still be needed for optimal results.

Multinomial NB with LDA

Results Results with the default parameters after applying LDA were:

- **Accuracy**: 0.4440
- **F1 Score**: [0.5236, 0.3327]

The model was fine-tuned using GridSearchCV and the parameters with the highest score are as follows:

- `alpha`: 1.0
- `fit_prior`: False

Top 10 parameters (obtained with GridSearchCV) and respective test accuracy are shown in the table below.

Place	Alpha	Fit Prior	Accuracy
5	1.0	False	0.6237
1	0.5	False	0.6235
3	0.75	False	0.6235
4	1.0	True	0.6234
2	0.75	True	0.6231
0	0.5	True	0.6230

Table 12: Multinomial NB with LDA - Top 10 Hyperparameter Configurations and Test Accuracy

Then, we took the best parameters and made the predictions:

- **Accuracy achieved for best parameters**: 0.6925

Running the validation on the same training data resulted in an accuracy of 0.99, indicating overfitting.

Next, we conducted a comprehensive evaluation of the model on the dataset through 100 iterations with different random splits. The model was trained and assessed in each iteration, considering distinct random seeds for data splitting. We tracked the best-performing model based on accuracy and classification report metrics across all iterations. The best classification report is shown in the table below.

Best Accuracy: 0.6925

Average Classification Report:

Class	Precision	Recall	F1-Score	Support
0	0.68	0.79	0.73	1572
1	0.72	0.58	0.64	1430
Accuracy	0.69			3002
Macro Avg	0.70	0.69	0.69	3002
Weighted Avg	0.70	0.69	0.69	3002

Table 13: Best Classification Report for Multinomial NB with LDA

- Precision: 0.6807
- Recall: 0.6704
- F1-Score: 0.6662
- Accuracy: 0.6713

The confusion matrix of the best model is presented in Figure 35. It indicates that the model has improved performance with LDA but still faces challenges in correctly classifying instances of Class 1.

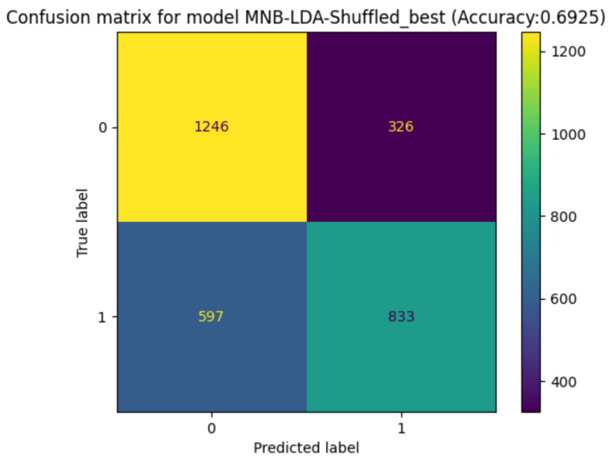

Figure 35: Confusion Matrix for Multinomial NB with LDA

The evaluation shows that applying LDA before Multinomial Naive Bayes can help improve model performance by reducing dimensionality and capturing the most significant features, but further tuning and feature engineering might still be needed for optimal results.

Multinomial NB with TruncatedSVD

Results Results with the default parameters after applying TruncatedSVD were:

- **Accuracy**: 0.5586
- **F1 Score**: [0.4925316 0.60949013]

The model was fine-tuned using GridSearchCV and the parameters with the highest score are as follows:

- `alpha`: 0.75
- `fit_prior`: False

Top 10 parameters (obtained with GridSearchCV) and respective test accuracy are shown in the table below.

Place	Alpha	Fit Prior	Accuracy
0	0.75	False	0.5696
1	1.0	False	0.5696
2	0.5	False	0.5696
3	0.5	True	0.5586
4	0.75	True	0.5586

Table 14: Multinomial NB with TruncatedSVD - Top 5 Hyperparameter Configurations and Test Accuracy

Then, we took the best parameters and made the predictions:

- **Accuracy achieved for best parameters**: 0.6942

Next, we conducted a comprehensive evaluation of the model on the dataset through 100 iterations with different random splits. The model was trained and assessed in each iteration, considering distinct random seeds for data splitting. We tracked the best-performing model based on accuracy and classification report metrics across all iterations. The best classification report is shown in the table below.

Class	Precision	Recall	F1-Score	Support
0	0.68	0.80	0.73	1572
1	0.72	0.58	0.64	1430
Accuracy	0.69			3002
Macro Avg	0.70	0.69	0.69	3002
Weighted Avg	0.70	0.69	0.69	3002

Table 15: Best Classification Report for Multinomial NB with TruncatedSVD

Best Accuracy: 0.6942

The confusion matrix of the best model is presented in Figure 36. It indicates that the model has improved performance with t-SNE but still faces challenges in correctly classifying instances of Class 1.

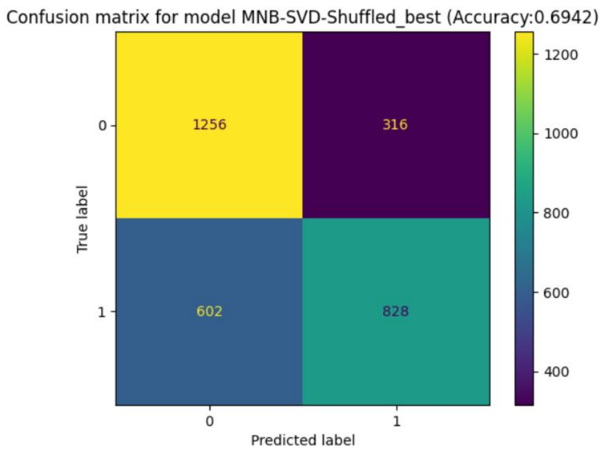

Figure 36: Confusion Matrix for Multinomial NB with TruncatedSVD

The evaluation shows that applying tSVD before Multinomial Naive Bayes can help improve model performance by visualizing and capturing the complex structure of the data, but further tuning and feature engineering might still be needed for optimal results.

Model 4: SVM

Support Vector Machines (SVM) is a versatile and robust machine learning algorithm developed for both classification and regression tasks. It is a supervised learning algorithm that aims to find a hyperplane in an N-dimensional space (where N is the number of features) that distinctly separates data points belonging to different classes. SVM is widely used in various domains, including image recognition, text classification, and bioinformatics.

Key Concepts

1. Hyperplane

- SVM works by finding a hyperplane that best separates the data points of different classes in feature space.
- In a two-dimensional space, the hyperplane is a line, while in higher dimensions, it becomes a hyperplane.

2. Support Vectors

- Support vectors are the data points that lie closest to the decision boundary (hyperplane).

- These vectors play a crucial role in determining the optimal hyperplane and are used to define the margin.

3. Margin

- The margin is the distance between the hyperplane and the nearest data point of any class.
- SVM aims to maximize this margin, leading to better generalization and improved performance on unseen data.

4. Kernel Trick

- SVM can efficiently handle non-linearly separable data by transforming the input features into a higher-dimensional space.
- Kernels, such as polynomial or radial basis function (RBF), are applied to implicitly map the data into a higher-dimensional space without explicitly calculating the transformations.

Limitations

1. Computational Complexity

- Training SVMs can be computationally expensive, especially for large datasets.

2. Sensitivity to Noise

- SVMs can be sensitive to noisy data and outliers.

3. Interpretability

- The decision boundary of an SVM may be challenging to interpret, particularly in high-dimensional spaces.

Results with the default parameters were:

- Accuracy: 0.747834
- F1 score: [0.77067555 0.71994081]

Hyperparameter Optimization

To optimize the SVM model's performance, we employed a Grid Search. The parameter grid for GridSearchCV included variations in regularization strength, kernel types, and gamma settings:

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['rbf', 'linear', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto'],
}
```

The parameters with the highest score are as follows: **{'C': 100, 'gamma': 'auto', 'kernel': 'rbf'}**. The results of this optimization are presented in the table below:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.74	0.82	0.78	1556
1	0.78	0.69	0.73	1446
Accuracy	0.7565			
Macro Avg	0.76	0.75	0.75	3002
Weighted Avg	0.76	0.76	0.76	3002
Best Accuracy	0.7565			
Average Classification Report	**Precision:** 0.7600 **Recall:** 0.7540 **F1-Score:** 0.7551 **Accuracy:** 0.7565			

Table 16: SVC - Best and Average Classification Reports

Next, we conducted a comprehensive evaluation of the model using **alternative-methods**. These alternative methods imply that we take the optimized models (found during the grid search), sort them by average classification report, and make predictions to find the best accuracy. The best models found with alternative methods are presented below:

Place	Parameters	Accuracy
	C, gamma, kernel	
0	100, auto, rbf	0.7565
1	10, scale, rbf	0.7568
2	10, scale, poly	0.7528
3	100, scale, rbf	0.7558
4	10, auto, rbf	0.7535
5	1, scale, rbf	0.7478
6	1, scale, poly	0.7498
7	100, scale, poly	0.7595
8	10, auto, linear	0.7528
9	10, scale, linear	0.7528

Table 17: SVC - Top 10 Hyperparameter Configurations and Test Accuracy

The confusion matrix of the best model using alternative methods is presented in Fig. 37. It indicates that the model performs well in predicting Class 1, but has significant difficulties in predicting Class 0 and some confusion between Class 1 and Class 2.

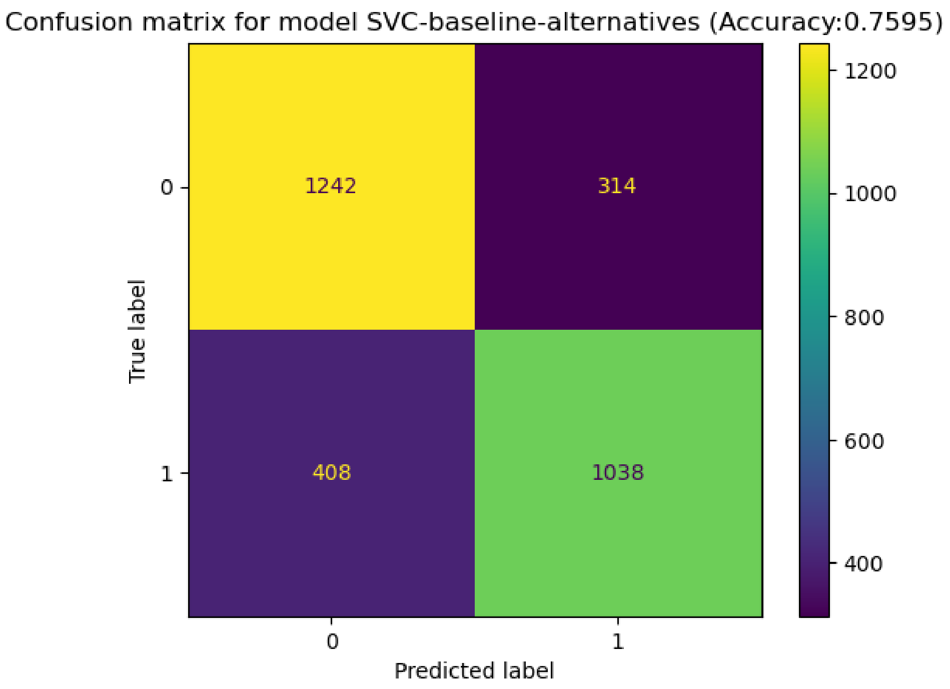

Figure 37: Confusion Matrix for Best RF

The results obtained with different dimensionality reduction methods are presented below:

The results demonstrate that LDA consistently outperforms PCA and Truncated SVD when combined with SVM for this classification task. The SVC-LDA models achieved

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Weighted F1**
SVC_PCA_GS_CV1	0.7315	0.7320	0.7315	[0.746, 0.715]	0.7310
SVC_PCA_alternatives	0.7315	0.7320	0.7315	[0.746, 0.715]	0.7310
SVC_LDA_GS_CV1	0.7428	0.7437	0.7428	[0.758, 0.726]	0.7422
SVC_LDA_alternatives	0.7442	0.7447	0.7442	[0.758, 0.729]	0.7437
SVC_SVD_GS_CV1	0.6106	0.6178	0.6106	[0.577, 0.639]	0.6073
SVC_SVD_alternatives	0.6136	0.6208	0.6136	[0.581, 0.642]	0.6104

Table 18: SVM Model Performance Metrics with Different Dimensionality Reduction Methods

the highest accuracy and F1 scores, indicating superior performance in capturing the underlying data structure. In contrast, SVC-SVD models lagged significantly, suggesting that SVD may not be well-suited for this particular problem. PCA provided moderate improvements but did not match the effectiveness of LDA. This highlights the importance of choosing appropriate dimensionality reduction techniques to enhance SVM performance.

Model 5: KNN

K-Nearest Neighbors (KNN) is a simple yet effective machine learning algorithm used for both classification and regression tasks. It is a type of instance-based learning where predictions are made based on the majority class or average of the k-nearest neighbors in the feature space. KNN is known for its simplicity and ease of implementation, making it a popular choice for various applications.

Key Concepts

1. Instance-Based Learning
    - KNN is an instance-based learning algorithm that memorizes the entire training dataset, making predictions based on the similarity of new instances to existing data points.
2. Distance Metric
    - The choice of a distance metric (e.g., Euclidean distance, Manhattan distance) is crucial in determining the similarity between data points.
    - Euclidean distance is commonly used in KNN for continuous features, while other distance metrics may be suitable for different types of data.
3. K-Nearest Neighbors

- The 'k' in KNN represents the number of nearest neighbors considered for making predictions.
- The algorithm identifies the k-nearest neighbors based on the chosen distance metric and assigns the majority class (for classification) or average value (for regression) as the prediction.

4. Decision Boundary
    - KNN does not explicitly learn a decision boundary; instead, it classifies or predicts based on the majority vote or average of neighboring instances.

Limitations

1. Computational Complexity
    - KNN can be computationally expensive, especially for large datasets, as it requires calculating distances for each prediction.
2. Sensitivity to Noise
    - KNN is sensitive to outliers and noisy data, as they can significantly impact the nearest neighbors' selection.
3. Curse of Dimensionality
    - As the number of features increases, the distance between instances tends to become more uniform, affecting the performance of KNN.

Baseline Results

The initial implementation of the K-Nearest Neighbors (KNN) classifier was conducted using the default parameters. The results are summarized below.

Default Parameters:

- Model: `KNeighborsClassifier()`

The model was fine-tuned using `GridSearchCV` with a specified parameter grid. The highest-scoring parameters from the grid search are as follows:

- **Best Parameters**:
    - `n_neighbors`: 15
    - `p`: 1

Top Hyperparameters and Accuracy: The top 10 hyperparameter configurations and their respective test accuracies are shown in Table 19.

Place	n_neighbors	p	Accuracy
8	15	1	0.709098
20	15	1	0.709098
16	7	1	0.704599
4	7	1	0.704599
21	15	2	0.701684
9	15	2	0.701684
22	15	3	0.700850
10	15	3	0.700850
23	15	4.3	0.698768
11	15	4.3	0.698768

Table 19: Top 10 Hyperparameter Configurations and Test Accuracy for KNN

Best Parameters Achieved:

- **Accuracy**: 0.7205
- **F1 Score**: [0.73789441, 0.70607785]

Upon validation on the same training data, the model achieved an accuracy of 0.99, indicating significant overfitting.

Evaluation on Test Data: A comprehensive evaluation was conducted over the dataset with 100 iterations of different random splits. The results are summarized in the classification report in Table 20.

Class	Precision	Recall	F1-Score	Support
0	0.72	0.75	0.74	1569
1	0.72	0.69	0.70	1433
Accuracy	0.7205			
Macro Average	0.7202	0.7190	0.7201	3002
Weighted Average	0.7204	0.7205	0.7201	3002

Table 20: Classification Report for KNN

	Predicted Class 0	Predicted Class 1
True Class 0	1181	388
True Class 1	451	982

Table 21: Confusion Matrix for KNN

Conclusion: The evaluation demonstrates that the K-Nearest Neighbors (KNN) model, while achieving reasonable accuracy, may still benefit from further parameter tuning and feature engineering to improve performance, especially in distinguishing between classes with some degree of confusion.

This section provides a baseline understanding of the model's performance and will serve as a reference point for further improvements and comparisons with other models or techniques.

KNN with PCA

In this section, we explore the performance of the K-Nearest Neighbors (KNN) classifier after applying Principal Component Analysis (PCA) for dimensionality reduction. The following results summarize the evaluation of the KNN model using PCA.

Implementation: The KNN model was fine-tuned using `GridSearchCV` on the dataset transformed with PCA. The results are detailed below.

Default Parameters and Accuracy:

- **Accuracy**: 0.53797
- **F1 Score**: [0.5489, 0.5265]

The model was fine-tuned using `GridSearchCV`, and the highest-scoring parameters are as follows:

- **Best Parameters**:
  - `n_neighbors`: 15
  - `p`: 1

Top Hyperparameters and Accuracy: The top 10 hyperparameter configurations and their respective test accuracies are shown in Table **??**.

Place	n_neighbors	p	Accuracy
8	15	1	0.590056
20	15	1	0.590056
21	15	2	0.585308
9	15	2	0.585308
10	15	3	0.584308
22	15	3	0.584308
11	15	4.3	0.582475
23	15	4.3	0.582475
4	7	1	0.577893
16	7	1	0.577893

Table 22: Top 10 Hyperparameter Configurations and Test Accuracy for KNN with PCA

Best Parameters Achieved:

- **Accuracy**: 0.5463
- **F1 Score**: [0.5632, 0.5281]

Evaluation on Test Data: A comprehensive evaluation was conducted, and the results are summarized in the classification report in Table **??**.

Class	Precision	Recall	F1-Score	Support
0	0.54	0.58	0.56	1505
1	0.55	0.51	0.53	1497
Accuracy	0.5463			
Macro Average	0.5465	0.5462	0.5457	3002
Weighted Average	0.5465	0.5463	0.5457	3002

Table 23: Classification Report for KNN with PCA

	Predicted Class 0	Predicted Class 1
True Class 0	870	635
True Class 1	729	768

Table 24: Confusion Matrix for KNN with PCA

Conclusion: The evaluation demonstrates that while applying PCA to the KNN model provides a moderate increase in performance, there remains room for further optimization. Fine-tuning the model and exploring other dimensionality reduction techniques may yield better results.

KNN with LDA

In this section, we explore the performance of the K-Nearest Neighbors (KNN) classifier after applying Linear Discriminant Analysis (LDA) for dimensionality reduction. The following results summarize the evaluation of the KNN model using LDA.

Implementation: The KNN model was fine-tuned using `GridSearchCV` on the dataset transformed with LDA. The results are detailed below.

Default Parameters and Accuracy:

- **Accuracy**: 0.5893
- **F1 Score**: [0.5993, 0.5787]

The model was fine-tuned using `GridSearchCV`, and the highest-scoring parameters are as follows:

- **Best Parameters**:
  - `n_neighbors`: 15
  - `p`: 1
  - `n_jobs`: -1

Top Hyperparameters and Accuracy: The top 10 hyperparameter configurations and their respective test accuracies are shown in Table 25.

Place	n_neighbors	p	Accuracy
23	15	4.3	0.621460
22	15	3	0.621460
21	15	2	0.621460
8	15	1	0.621460
9	15	2	0.621460
10	15	3	0.621460
11	15	4.3	0.621460
20	15	1	0.621460
4	7	1	0.609797
16	7	1	0.609797

Table 25: Top 10 Hyperparameter Configurations and Test Accuracy for KNN with LDA

Best Parameters Achieved:

- **Accuracy**: 0.6336
- **F1 Score**: [0.6472, 0.6188]

Evaluation on Test Data: A comprehensive evaluation was conducted, and the results are summarized in the classification report in Table 26.

Class	Precision	Recall	F1-Score	Support
0	0.65	0.64	0.65	1569
1	0.61	0.62	0.62	1433
Accuracy	0.6336			
Macro Average	0.6330	0.6331	0.6337	3002
Weighted Average	0.6338	0.6336	0.6337	3002

Table 26: Classification Report for KNN with LDA

The confusion matrix for the best model is presented in Table 27.

	Predicted Class 0	Predicted Class 1
True Class 0	1009	560
True Class 1	540	893

Table 27: Confusion Matrix for KNN with LDA

Conclusion: The evaluation demonstrates that the K-Nearest Neighbors (KNN) model with LDA achieved an accuracy of 0.6336. While this represents a moderate performance, further tuning of hyperparameters and exploring other dimensionality reduction techniques may yield better results.

KNN with Truncated SVD

In this section, we explore the performance of the K-Nearest Neighbors (KNN) classifier after applying Truncated Singular Value Decomposition (Truncated SVD) for dimensionality reduction. The following results summarize the evaluation of the KNN model using Truncated SVD.

Implementation: The KNN model was fine-tuned using `GridSearchCV` on the dataset transformed with Truncated SVD. The results are detailed below.

Default Parameters and Accuracy:

- **Accuracy**: 0.5179
- **F1 Score**: [0.5223, 0.5136]

The model was fine-tuned using `GridSearchCV`, and the highest-scoring parameters are as follows:

- **Best Parameters**:

  - `n_neighbors`: 15
  - `p`: 2
  - `n_jobs`: -1

Top Hyperparameters and Accuracy: The top 10 hyperparameter configurations and their respective test accuracies are shown in Table 28.

Place	**n_neighbors**	**p**	**Accuracy**
21	15	2	0.547319
9	15	2	0.547319
23	15	4.3	0.546404
11	15	4.3	0.546404
10	15	3	0.545737
22	15	3	0.545737
20	15	1	0.544237
6	15	1	0.544237
18	7	3	0.536573
4	7	3	0.536573

Table 28: Top 10 Hyperparameter Configurations and Test Accuracy for KNN with Truncated SVD

Best Parameters Achieved:

- **Accuracy**: 0.5363
- **F1 Score**: [0.5451, 0.5272]

Evaluation on Test Data: A comprehensive evaluation was conducted, and the results are summarized in the classification report in Table 29.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.54	0.55	0.55	1505
1	0.54	0.52	0.53	1497
Accuracy	0.5363			
Macro Average	0.5363	0.5362	0.5363	3002
Weighted Average	0.5363	0.5363	0.5363	3002

Table 29: Classification Report for KNN with Truncated SVD

The confusion matrix for the best model is presented in Table 30.

	Predicted Class 0	**Predicted Class 1**
True Class 0	834	671
True Class 1	721	776

Table 30: Confusion Matrix for KNN with Truncated SVD

Conclusion: The evaluation demonstrates that the K-Nearest Neighbors (KNN) model with Truncated SVD achieved an accuracy of 0.5363. While this represents a moderate performance, further tuning of hyperparameters and exploring other dimensionality reduction techniques may yield better results.

Comparing all of the Results

In this study, we conducted a comprehensive comparison of five classification algorithms: Random Forest, Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Multinomial Naive Bayes, using an aggregated dataset containing information about movies. Our primary objective was to evaluate the performance of these algorithms on the complete, preprocessed dataset and further assess their performance on datasets obtained by applying dimensionality reduction techniques: Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Truncated Singular Value Decomposition (Truncated SVD).

Logistic Regression demonstrated varied performance across different dimensionality reduction techniques. The baseline model achieved an accuracy of 0.7418, with an F1 score of [0.758, 0.724]. Applying PCA resulted in a slight decrease in performance, with the best PCA-tuned model (LR_PCA_OPT2) achieving an accuracy of 0.7272. LDA provided a good performance for this algorithm, with the best LDA-tuned model (LR_LDA_OPT1) achieving an accuracy of 0.7445, which is very close to the baseline model's performance. Truncated SVD showed significantly lower performance, with the accuracy for SVD-reduced models remaining constant at 0.6079 across all tuning methods. These results indicate that while hyperparameter optimization significantly improved the logistic regression model's performance, the choice of dimensionality reduction technique played a crucial role. LDA emerged as the most effective method for this classification problem, while Truncated SVD was the least effective.

Multinomial Naive Bayes demonstrated varied performance across different dimensionality reduction techniques. The baseline model achieved an accuracy of 0.6762, with an F1 score of [0.7204, 0.6155]. Applying PCA slightly improved the accuracy to 0.6885. LDA provided a good performance for this algorithm, with an accuracy of 0.6925. Truncated SVD has even better results, resulting in an accuracy of 0.6942 and F1 scores

[0.4925316, 0.60949013]. These results indicate that while all dimensionality reduction techniques offer some improvement, Truncated SVD provided the best overall performance for Multinomial Naive Bayes on this dataset.

Suppor Vector Machines demonstrated that LDA consistently outperforms PCA and Truncated SVD when combined with this model for classification tasks. The SVC-LDA models achieved the highest accuracy and F1 scores, indicating superior performance in capturing the underlying data structure. In contrast, SVC-SVD models lagged significantly, suggesting that SVD may not be well-suited for this particular problem. PCA provided moderate improvements but did not match the effectiveness of LDA. This highlights the importance of choosing appropriate dimensionality reduction techniques to enhance SVM performance.

K-Nearest Neighbors (KNN) algorithm performed best in its baseline form, achieving an accuracy of 0.7205 and F1 scores of [0.73789441, 0.70607785]. When PCA was applied, the accuracy dropped to 0.5463 with F1 scores of [0.5632, 0.5281], suggesting that PCA may not be beneficial for KNN in this context. LDA showed moderate performance improvements with an accuracy of 0.6336 and F1 scores of [0.6472, 0.6188], indicating it can be useful for enhancing KNN performance by maximizing class separability. Truncated SVD resulted in lower performance, with an accuracy of 0.5363 and F1 scores of [0.5451, 0.5272]. These results suggest that while KNN benefits from LDA, other dimensionality reduction techniques like PCA and Truncated SVD may not be as effective.

Random Forest showed variant results when combined with different dimensionality reduction techniques Principal Component Analysis, when paired with Random Forest, demonstrated improved accuracy and F1 scores in alternative configurations compared to GridSearchCV. Linear Discriminant Analysis also showed slight performance enhancements with alternative methods, although the gains were marginal. Singular Value Decomposition, however, resulted in significantly lower performance metrics, indicating its unsuitability for this dataset with Random Forest. Overall, the choice of dimensionality reduction method substantially influences the efficacy of Random Forest, with PCA and LDA proving more effective than SVD.

Final Remarks Overall, the choice of dimensionality reduction technique significantly impacted the performance of the classification algorithms. LDA emerged as the most effective method for Logistic Regression and KNN, enhancing their performance by retaining crucial discriminative information. Truncated SVD, while beneficial for Multinomial Naive Bayes, generally resulted in lower performance for other algorithms. PCA provided mixed results, being less effective for KNN but moderately useful for Logistic Regression and Random Forest.

Further research could involve additional hyperparameter tuning, exploring other dimensionality reduction methods, and combining multiple classifiers to improve predictive performance. This study provides valuable insights into the effectiveness of different algorithms and dimensionality reduction techniques for movie dataset classification, offering a solid foundation for future investigations in similar contexts.