Predictive Modeling Using Gradient Boosting Algorithms for Movie Metadata Analysis

SUBMITTED BY

NAME:SHREYA S PATIL

SEMESTER:VI

ORGANIZATION:M S RAMAIAH INSTITUTE OF TECHNOLOGY

INTRODUCTION

In our analysis of movie metadata, we chose to employ Gradient Boosting algorithms, specifically GradientBoostingRegressor and GradientBoostingClassifier. This decision was driven by several factors that make Gradient Boosting a suitable choice for predictive modeling in this context.

Gradient Boosting offers a powerful ensemble learning technique that combines the predictions of multiple weak learners, typically decision trees, to produce a strong predictive model. This approach allows us to effectively capture complex relationships and patterns in the data, enhancing our ability to predict movie release years and classify movie genres based on metadata features.

Moreover, Gradient Boosting algorithms are flexible and robust, capable of handling both regression and classification tasks with high predictive power. They are less prone to overfitting compared to individual decision trees.

The selection of Gradient Boosting algorithms for our movie metadata analysis reflects their ability to deliver accurate predictions, handle diverse data types, and facilitate efficient model optimization. Through this approach, we aim to extract valuable insights from movie metadata and contribute to informed decision-making in the movie industry.

DATASET OVERVIEW

The dataset used in our analysis contains comprehensive metadata information about a diverse range of movies. It encompasses various numerical and categorical features that provide insights into different aspects of each movie. Key attributes include:

Numerical Features:

- Number of critic reviews
- Duration of the movie
- Director Facebook likes
- Actor 1 Facebook likes
- Gross revenue
- Number of voted users
- Total Facebook likes of the cast
- Number of faces on the movie poster
- Number of user reviews
- Budget of the movie
- Actor 2 Facebook likes
- IMDB score
- Aspect ratio of the movie
- Number of Facebook likes for the movie

Categorical Features:

- Color (e.g., color or black and white)
- Language of the movie
- Country of production
- Content rating (e.g., PG-13, R, etc.)

These features collectively provide a rich source of information about each movie, allowing us to explore various patterns and trends in movie metadata. The target variables of interest are the release year of the movie and its associated genres, which are represented as binary indicators for different genres.

DATA PREPROCESSING

The data preprocessing steps in the provided code involve preparing the dataset for modeling by handling missing values, scaling numerical features, and encoding categorical features. Let's break down each step:

Handling Missing Values:

```
# Handling missing values
numerical_columns = data.select_dtypes(include=['number']).columns
categorical_columns = data.select_dtypes(exclude=['number']).columns

data[numerical_columns] = data[numerical_columns].fillna(data[numerical_columns].median())
data[categorical_columns] = data[categorical_columns].fillna(data[categorical_columns].mode().iloc[0])
```

- First, the columns with numerical data types are identified using select_dtypes(include=['number']), and columns with categorical data types are identified using select_dtypes(exclude=['number']).
- Missing values in numerical columns are imputed with the median of each column using SimpleImputer(strategy='median').
- Missing values in categorical columns are imputed with the most frequent value of each column using SimpleImputer(strategy='constant', fill_value='missing').

Feature Engineering:

```
# Feature engineering
# Create interaction features
data['budget_duration_interaction'] = data['budget'] * data['duration']

# Genre Frequency
genre_frequency = data['genres'].str.get_dummies(sep='|').mean()
data['genre_frequency'] = data['genres'].apply(lambda x: sum([genre_frequency[g] for g in x.split('|')]) / len(x.split('|')))

# Director's Hit Rate
director_hit_rate = data.groupby('director_name')['imdb_score'].mean().fillna(0)
data['director_hit_rate'] = data['director_name'].map(director_hit_rate)

# Lead Actor's Fame
lead_actor_fame = (data['actor_1_name'].map(data.groupby('actor_1_name')['actor_1_facebook_likes'].mean()) +
                   data['actor_2_name'].map(data.groupby('actor_2_name')['actor_2_facebook_likes'].mean())) / 2
data['lead_actor_fame'] = lead_actor_fame.fillna(0)

# Budget-to-Gross Ratio
data['budget_to_gross_ratio'] = data['budget'] / data['gross']
```

- Interaction features are created by multiplying the 'budget' and 'duration' columns to generate a new feature called 'budget_duration_interaction'.
- Genre frequency is calculated by splitting the 'genres' column, creating dummy variables using str.get_dummies(sep='|'), and taking the mean of each genre's occurrence across movies.
- Director's hit rate is computed by grouping the data by 'director_name' and calculating the mean 'imdb_score' for each director. Missing values are filled with 0.

- Lead actor's fame is calculated as the average of the Facebook likes of actor 1 and actor 2 for each movie. Missing values are filled with 0.
- Budget-to-gross ratio is computed by dividing the 'budget' column by the 'gross' column.

Defining Features and Target Variables:

```
# Separate features and target variables
X = data.drop(['title_year', 'genres', 'movie_title', 'movie_imdb_link'], axis=1)
y_release_year = data['title_year']
y_genres = data['genres'].str.get_dummies(sep='|')

# Get the most probable genre for each movie
y_train_genres_labels = y_genres.idxmax(axis=1)
```

- Features (X) are defined by dropping columns such as 'title_year', 'genres', 'movie_title', and 'movie_imdb_link' that are not used for prediction.
- Target variables for release year (y_release_year) and genres (y_genres) are separated from the dataset.

Splitting Data for Prediction:

```
# Split data into train and test sets for release year prediction
X_train_release_year, X_test_release_year, y_train_release_year, y_test_release_year = train_test_split(
    X, y_release_year, test_size=0.2, random_state=42)

# Split data into train and test sets for genre prediction
X_train_genres, X_test_genres, y_train_genres_labels, y_test_genres_labels = train_test_split(
    X, y_train_genres_labels, test_size=0.2, random_state=42)
```

- The dataset is split into training and testing sets separately for predicting release year and genres using train_test_split.

Preprocessing Pipeline:

```
# Preprocessing pipeline for numerical features
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Preprocessing pipeline for categorical features
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

- Separate preprocessing pipelines are defined for numerical and categorical features using Pipeline.
- Numerical features are imputed with the median and scaled using StandardScaler.
- Categorical features are imputed with a constant value ('missing') and one-hot encoded using OneHotEncoder.

Preprocessing Data for Prediction:

```
# Preprocess the data for release year prediction
X_train_release_year_preprocessed = preprocessor.fit_transform(X_train_release_year)
X_test_release_year_preprocessed = preprocessor.transform(X_test_release_year)
```

- The preprocessing steps defined in the pipeline are applied to both the training and testing sets for release year and genre predictions.
- These preprocessing steps ensure that the data is in a suitable format for training the Gradient Boosting models for predicting release year and genres accurately.

Splitting of the Dataset:

```
# Split data into train and test sets for genre prediction
X_train_genres, X_test_genres, y_train_genres_labels, y_test_genres_labels = train_test_split(
    X, y_train_genres_labels, test_size=0.2, random_state=42)
```

- The dataset is split into training and testing sets using train_test_split.

Defining of numerical and categorical features:

```
# Define numerical and categorical features
numerical_features = ['num_critic_for_reviews', 'duration', 'director_facebook_likes',
                      'actor_1_facebook_likes', 'gross', 'num_voted_users',
                      'cast_total_facebook_likes', 'facenumber_in_poster',
                      'num_user_for_reviews', 'budget', 'actor_2_facebook_likes',
                      'imdb_score', 'aspect_ratio', 'movie_facebook_likes',
                      'budget_duration_interaction', 'genre_frequency',
                      'director_hit_rate', 'lead_actor_fame', 'budget_to_gross_ratio']
categorical_features = ['color', 'language', 'country', 'content_rating']
```

- In this, we define the numerical and categorical features extracted from the movie metadata dataset for training our predictive models.

Preprocessing Pipeline for Numerical and Categorical Features

```
# Preprocessing pipeline for numerical features
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Preprocessing pipeline for categorical features
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

To prepare the features for model training, we construct separate preprocessing pipelines for numerical and categorical features.

Numerical Features Pipeline:

- Imputation: Missing values in numerical features are filled with the median value of the respective feature.
- Scaling: Numerical features are scaled using StandardScaler to standardize their values, making them comparable across different scales.

Categorical Features Pipeline:

- Imputation: Missing values in categorical features are filled with a constant value ('missing').
- One-Hot Encoding: Categorical features are one-hot encoded using OneHotEncoder to convert them into binary vectors, representing the presence or absence of each category.

Combined Preprocessing Steps

```
# Combine preprocessing steps
preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_transformer, numerical_features),
    ('cat', categorical_transformer, categorical_features)
])
```

- The preprocessing steps for numerical and categorical features are combined into a single ColumnTransformer. This transformer applies the respective preprocessing pipelines to the corresponding feature types.

By employing these preprocessing techniques, we ensure that the numerical and categorical features are appropriately handled and transformed, ready to be used for training our predictive models.

Gradient Boosting Algorithm Architecture

Gradient Boosting is an ensemble learning technique that combines the predictions of multiple weak learners (typically decision trees) to create a stronger predictive model. The architecture of Gradient Boosting algorithms, such as GradientBoostingRegressor and GradientBoostingClassifier, follows a sequential training process:

- **Initialization:** The algorithm starts with an initial model, often a simple one like a single decision tree. This model makes predictions on the training data, which are initially quite inaccurate.
- **Loss Function:** Gradient Boosting minimizes a loss function that measures the difference between the actual target values and the predictions made by the current model. Common loss functions include mean squared error for regression tasks and cross-entropy loss for classification tasks.
- **Gradient Descent:** In each iteration (or boosting round), the algorithm fits a new weak learner to the residuals (the differences between the actual and predicted values) of the previous model. It computes the negative gradient of the loss function with respect to these residuals, indicating the direction of steepest descent.
- **Model Update:** The new weak learner is added to the ensemble in such a way that it reduces the residual error. The contribution of each model to the ensemble is determined by a learning rate parameter, which scales the predictions of each weak learner.
- **Iterative Process:** This process is repeated for a specified number of iterations or until a predefined stopping criterion is met, such as reaching a maximum number of models or achieving satisfactory performance.
- **Final Prediction:** The final prediction is obtained by aggregating the predictions of all weak learners in the ensemble. For regression tasks, this aggregation is typically a simple sum of individual predictions, while for classification tasks, it may involve averaging probabilities or using a voting mechanism.

By iteratively improving the model's predictions based on the errors of the previous models, Gradient Boosting algorithms can construct highly accurate predictive models that generalize well to unseen data. This architecture makes them powerful tools for a wide range of machine learning tasks, including regression, classification, and ranking.

MODEL TRAINING

In this , we train two Gradient Boosting models using the Scikit-learn library: one for predicting movie release years and the other for predicting movie genres.

1. Release Year Prediction

Data Preprocessing:

```
# Preprocess the data for release year prediction
X_train_release_year_preprocessed = preprocessor.fit_transform(X_train_release_year)
X_test_release_year_preprocessed = preprocessor.transform(X_test_release_year)
```

These steps ensure that our data is properly transformed and ready for training the Gradient Boosting Regressor..

Model Training:

We define the Gradient Boosting Regressor and fit it to the preprocessed training data to predict movie release years.

```
# Define XGBoost regressor for release year prediction
xgb_regressor = GradientBoostingRegressor()

# Fit XGBoost regressor for release year prediction
xgb_regressor.fit(X_train_release_year_preprocessed, y_train_release_year)
```

This trained regressor will be used to predict movie release years based on the provided features.

2. Genre Prediction

Data Preprocessing:

```
# Preprocess the data for genre prediction
X_train_genres_preprocessed = preprocessor.fit_transform(X_train_genres)
X_test_genres_preprocessed = preprocessor.transform(X_test_genres)
```

Similar to release year prediction, we preprocess the data.The same preprocessing steps are applied using the preprocessor defined earlier.

Model Training:

```
# Define XGBoost classifier for genre prediction
xgb_classifier = GradientBoostingClassifier()

# Fit XGBoost classifier for genre prediction
xgb_classifier.fit(X_train_genres_preprocessed, y_train_genres_labels)
```

A Gradient Boosting Classifier is defined and trained on the preprocessed training data.

The trained classifier is used to predict movie genres on the test data.

RESULTS

The analysis produced the following results:

```
Mean Absolute Error (Release Year Prediction) - XGBoost Regressor: 4.685647034153494
Accuracy (Genres Prediction) - XGBoost Classifier: 0.7819623389494549
```

Release Year Prediction:

Mean Absolute Error (MAE) for the XGBoost Regressor model predicting movie release years is approximately 4.686. This indicates that, on average, the predicted release years deviate from the actual release years by around 4.686 years. While there is some error in the predictions, this level of deviation suggests a reasonable approximation of release dates.

Genre Prediction:

The accuracy score for the XGBoost Classifier model predicting movie genres is approximately 0.782. This indicates that the model correctly classifies movie genres about 78.2% of the time. While not perfect, this level of accuracy suggests that the model can effectively classify movie genres based on the provided features, providing valuable insights for genre selection and audience targeting.

These results demonstrate the efficacy of the predictive models in analyzing movie metadata and making informed predictions regarding movie release years and genres.

Conclusion

In conclusion, the application of Gradient Boosting models to predict movie release years and genres based on movie metadata has shown promising results. The models exhibited reasonable accuracy in predicting both release years and genres, providing valuable insights into movie trends and characteristics.

- The Gradient Boosting Regressor achieved a Mean Absolute Error (MAE) of approximately 4.686 in predicting movie release years, indicating that the model can make reasonably accurate estimations of release dates. This insight can be valuable for understanding temporal trends in the movie industry and guiding marketing and distribution strategies.

- Similarly, the Gradient Boosting Classifier achieved an accuracy score of approximately 0.782 in predicting movie genres, demonstrating the model's ability to correctly classify genres with a high degree of accuracy. This information is essential for content recommendation systems, audience segmentation, and targeted marketing efforts.

- Feature engineering played a crucial role in enhancing the models' predictive performance, with engineered features such as interaction features, genre frequencies, and metrics like director's hit rate and lead actor's fame providing valuable context and insight into the relationships between movie attributes and target variables.

- The preprocessing steps, including handling missing values and encoding categorical features, were essential for preparing the data for model training. The use of pipelines and column transformers streamlined the preprocessing workflow, making it efficient and scalable.

Overall, the analysis of movie metadata using Gradient Boosting models provides valuable insights for various stakeholders in the film industry, contributing to more informed decision-making processes and strategies for content creation, distribution, and audience engagement.

FUTURE SCOPE

- The predictive models could be further optimized by tuning hyperparameters to improve performance.
- Additional features or alternative modeling techniques could be explored to enhance prediction accuracy.
- Exploring other aspects of movie metadata such as box office revenue prediction could provide further insights into movie success factors.