Detailed Report on Project

Rotten Tomatoes Movies Audience Rating Prediction

This project focuses on predicting the audience rating of movies using various machine learning techniques, including linear regression, Ridge regression, and Lasso regression. The dataset used is the "Rotten Tomatoes Movies" dataset.

**Table of Contents**

1. Import Libraries and Dataset Description
2. Data Preprocessing
3. Feature Engineering
4. Model Training and Evaluation
5. Hyperparameter Tuning
6. Visualization
7. Model Deployment
8. Conclusion
9. Future Work

**Import necessary Libraries**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn.model\_selection import GridSearchCV

import pickle

**Dataset Description**

The dataset contains various features of movies, including their title, genre, rating, directors, cast, and audience rating. The primary goal is to predict the audience\_rating.

**Data Preprocessing**

1. **Loading the Dataset**:

dataset = pd.read\_csv("Rotten\_Tomatoes\_Movies3.csv", encoding='latin1')

Display dataset information

print(dataset.head(5))

print(dataset.shape)

print(dataset.describe())

print(dataset.info())

1. **Handling Date Columns**: Convert in\_theaters\_date to a Unix timestamp:

if 'in\_theaters\_date' in dataset.columns:

dataset['in\_theaters\_date'] = pd.to\_datetime(dataset['in\_theaters\_date'], errors='coerce', dayfirst=True)

dataset['in\_theaters\_date'] = dataset['in\_theaters\_date'].view('int64') // 10\*\*9

1. **Dropping Unnecessary Columns**: Remove columns with the most null values:

columns\_to\_drop = ['critics\_consensus', 'writers', 'studio\_name', 'on\_streaming\_date', 'runtime\_in\_minutes', 'tomatometer\_status', 'tomatometer\_rating', 'tomatometer\_count']

dataset.drop(columns=columns\_to\_drop, inplace=True)

dataset.dropna(inplace=True)

**Feature Engineering**

1. **One-Hot Encoding**: Encode categorical columns into binary columns:

categorical\_columns = ['movie\_title', 'movie\_info', 'rating', 'genre', 'directors', 'cast']

dataset\_encoded = pd.get\_dummies(dataset, columns=categorical\_columns)

1. **Scaling Numerical Columns**: Standardize the numerical features:

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

numerical\_columns = dataset\_encoded.select\_dtypes(include=[float, int]).columns

dataset\_encoded[numerical\_columns] = scaler.fit\_transform(dataset\_encoded[numerical\_columns])

**Model Training and Evaluation**

1. **Splitting the Dataset**: Separate features and target, then split the data:

X = dataset\_encoded.drop('audience\_rating', axis=1)

y = dataset['audience\_rating']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

1. **Dimensionality Reduction with PCA**: Reduce the number of features:

from sklearn.decomposition import PCA

pca = PCA(n\_components=100)

X\_reduced = pca.fit\_transform(X)

1. **Training Linear Regression Model**:

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X\_train, y\_train)

1. **Evaluating the Model**: Calculate MAE, MSE, and R²:

y\_pred = model.predict(X\_test)

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"MAE: {mae}, MSE: {mse}, R²: {r2}")

**Hyperparameter Tuning**

1. **Ridge and Lasso Regression**: Tune hyperparameters using GridSearchCV:

from sklearn.linear\_model import Ridge, Lasso

from sklearn.model\_selection import GridSearchCV

ridge = Ridge()

ridge\_params = {'alpha': [0.1, 1.0, 10.0, 100.0]}

lasso = Lasso()

lasso\_params = {'alpha': [0.1, 1.0, 10.0, 100.0]}

ridge\_grid = GridSearchCV(ridge, ridge\_params, cv=5, scoring='r2')

ridge\_grid.fit(X\_train, y\_train)

lasso\_grid = GridSearchCV(lasso, lasso\_params, cv=5, scoring='r2')

lasso\_grid.fit(X\_train, y\_train)

best\_ridge = ridge\_grid.best\_estimator\_

best\_lasso = lasso\_grid.best\_estimator\_

1. **Evaluating Ridge and Lasso Models**:

y\_pred\_ridge = best\_ridge.predict(X\_test)

y\_pred\_lasso = best\_lasso.predict(X\_test)

mae\_ridge = mean\_absolute\_error(y\_test, y\_pred\_ridge)

mse\_ridge = mean\_squared\_error(y\_test, y\_pred\_ridge)

r2\_ridge = r2\_score(y\_test, y\_pred\_ridge)

mae\_lasso = mean\_absolute\_error(y\_test, y\_pred\_lasso)

mse\_lasso = mean\_squared\_error(y\_test, y\_pred\_lasso)

r2\_lasso = r2\_score(y\_test, y\_pred\_lasso)

print(f"Ridge - MAE: {mae\_ridge}, MSE: {mse\_ridge}, R²: {r2\_ridge}")

print(f"Lasso - MAE: {mae\_lasso}, MSE: {mse\_lasso}, R²: {r2\_lasso}")

**Visualization:**

* Scatter plot of actual vs. predicted values: The scatter plot of actual vs. predicted values helps you see how close your predictions are to the actual values. Ideally, the points should lie along the red line, indicating perfect predictions.

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred, alpha=0.5)

plt.title('Actual vs. Predicted Values')

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red') # Line of perfect prediction

plt.show()

* Residual plot : The residual plot shows the differences between the actual and predicted values. Ideally, residuals should be randomly scattered around zero, indicating no obvious patterns.

residuals = y\_test - y\_pred

plt.figure(figsize=(10, 6))

plt.scatter(y\_pred, residuals, alpha=0.5)

plt.title('Residual Plot')

plt.xlabel('Predicted Values')

plt.ylabel('Residuals')

plt.hlines(y=0, xmin=min(y\_pred), xmax=max(y\_pred), color='red') # Horizontal line at 0

plt.show()

* Distribution plot of residuals : This plot helps you check if the residuals follow a normal distribution, which is an assumption of linear regression.

plt.figure(figsize=(10, 6))

sns.histplot(residuals, kde=True)

plt.title('Distribution of Residuals')

plt.xlabel('Residuals')

plt.ylabel('Frequency')

plt.show()

**Model Deployment**

1. **Saving the Model**:

import pickle

with open('model.pkl', 'wb') as file:

pickle.dump(model, file)

1. **Creating Flask API**:

from flask import Flask, request, jsonify

import pickle

import numpy as np

app = Flask(\_\_name\_\_)

with open('model.pkl', 'rb') as file:

model = pickle.load(file)

@app.route('/predict', methods=['POST'])

def predict():

data = request.get\_json(force=True)

prediction = model.predict(np.array([data['features']]))

return jsonify({'prediction': prediction.tolist()})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=False)

Conclusion

In this project, we developed and evaluated a machine learning model to predict audience ratings for movies using a dataset from Rotten Tomatoes. We followed a comprehensive pipeline that included:

1. Data Preprocessing:

* Loading and inspecting the dataset.
* Handling missing values by dropping columns with a high percentage of null values and rows with any null values.
* Converting date columns to a numerical format for easier handling.

2. Feature Engineering:

* One-hot encoding categorical features to convert them into a numerical format suitable for machine learning algorithms.

3. Scaling and Dimensionality Reduction:

* Standardizing numerical features to have zero mean and unit variance.
* Reducing the dimensionality of the dataset using Principal Component Analysis (PCA) to retain the most informative features.

4. Model Training and Evaluation:

* Training a Linear Regression model.
* Evaluating the model using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R² score.
* Implementing hyperparameter tuning for Ridge and Lasso regression models to improve performance.

5. Visualization:

* Creating scatter plots, residual plots, and distribution plots to visualize the model's performance and residuals.

Future Work

While the current model provides a good starting point for predicting movie audience ratings, there are several areas for future improvement:

1. Feature Selection:

* Investigate additional features that might influence audience ratings, such as social media mentions, critic reviews, or audience demographics.

2. Advanced Models:

* Experiment with more advanced machine learning models like Random Forest, Gradient Boosting, or Neural Networks to potentially improve prediction accuracy.

3. Hyperparameter Tuning:

* Implement more sophisticated hyperparameter tuning techniques like Bayesian Optimization or Random Search to fine-tune model parameters.

4. Cross-Validation:

* Use cross-validation to ensure the model's robustness and generalizability to new, unseen data.

5. Deployment:

* Deploy the model in a production environment using a robust framework like Flask combined with a WSGI server like Gunicorn, ensuring it can handle real-world requests efficiently.

6. Real-Time Predictions:

* Integrate the model into a web application or API to provide real-time predictions based on user input.

By addressing these areas, we can enhance the accuracy and usability of the model, providing more reliable predictions for movie audience ratings.