Predicting IMDb Scores

Phase 4

Overview

This document consists of the implementation of feature engineering, model building and model evaluation in python. Implementation of the above listed steps will be discussed briefly with the necessary python code which was implemented and executed successfully in google colab.

Feature engineering

Feature engineering is the process of transforming raw data into features that are suitable for machine learning models. In other words, it is the process of selecting, extracting, and transforming the most relevant features from the available data to build more accurate and efficient machine learning models.

We have implemented certain Feature extraction process in the given dataset like extracting the columns ‘Date’ , ‘Year’ and ‘Month’ from the Premiere column that was already existing in the given dataset

```
df["Date"] = pd.to_datetime(df.Premiere)
df["Date"]
```

```
0      2019-08-05
1      2020-08-21
2      2019-12-26
3      2018-01-19
4      2020-10-30
          ...
579    2018-12-31
580    2015-10-09
581    2018-12-16
582    2020-12-08
583    2020-10-04
Name: Date, Length: 584, dtype: datetime64[ns]
```

```
df["Year"] = df["Date"].dt.year
df["Month"] = df["Date"].dt.month
print(df.head())
```

```
             Title                 Genre          Premiere  Runtime  \
0  Enter the Anime           Documentary    August 5, 2019       58
1      Dark Forces              Thriller    August 21, 2020       81
2          The App  Science fiction/Drama  December 26, 2019       79
3   The Open House        Horror thriller   January 19, 2018       94
4      Kaali Khuhi               Mystery   October 30, 2020       90

   IMDB Score          Language       Date  Year  Month
0         2.5  English/Japanese 2019-08-05  2019      8
1         2.6           Spanish 2020-08-21  2020      8
2         2.6           Italian 2019-12-26  2019     12
3         3.2           English 2018-01-19  2018      1
4         3.4             Hindi 2020-10-30  2020     10
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 584 entries, 0 to 583
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   Title       584 non-null    int64         
 1   Genre       584 non-null    int64         
 2   Premiere    584 non-null    int64         
 3   Runtime     584 non-null    int64         
 4   IMDB Score  584 non-null    float64       
 5   Language    584 non-null    int64         
 6   Date        584 non-null    datetime64[ns]
 7   Year        584 non-null    int64         
 8   Month       584 non-null    int64         
dtypes: datetime64[ns](1), float64(1), int64(7)
memory usage: 41.2 KB
```

The Feature transformation was implemented by the use of Label encoder to convert the categorical values into numeric values

```
[382] from sklearn.preprocessing import LabelEncoder
      cols=['Title','Genre','Runtime','Premiere','Language','Date','Year','Month']
      df[cols]=df[cols].apply(LabelEncoder().fit_transform)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 584 entries, 0 to 583
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Title       584 non-null    int64
 1   Genre       584 non-null    int64
 2   Premiere    584 non-null    int64
 3   Runtime     584 non-null    int64
 4   IMDB Score  584 non-null    float64
 5   Language    584 non-null    int64
 6   Date        584 non-null    int64
 7   Year        584 non-null    int64
 8   Month       584 non-null    int64
dtypes: float64(1), int64(8)
memory usage: 41.2 KB
```

The Feature selection is used to choose the most relevant features to include in the model while eliminating irrelevant or redundant ones. Therefore, we have excluded ‘Premiere’ and ‘Month’.

```
[383] x=df.drop(['Premiere','Month','IMDB Score'], axis=1)
      y=df['IMDB Score']
```

```
print(x)
print(y)
```

```
     Title  Genre  Runtime  Language  Date  Year
0      147     45       42         6   182     5
1      120    106       56        29   281     6
2      433     93       54        20   219     5
3      500     63       69         2    85     4
4      243     73       65        18   312     6
..     ...    ...      ...       ...   ...   ...
579    425     40      100         2   140     4
580    575     45       66        13     6     1
581    410     74      121         2   138     4
582    145     45       64        28   331     6
583    121     45       58         2   299     6

[584 rows x 6 columns]
0      2.5
1      2.6
2      2.6
3      3.2
4      3.4
      ...
579    8.4
```

Splitting of data

- The dataset is divided into training and test sets. The training set is used to train the model and the test set is used to evaluate the model's generalization performance.

```
[387] from sklearn.model_selection import train_test_split
      x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=42)
```

```
[313] print(x_train.shape)
      print(y_train.shape)
      print(x_test.shape)
      print(y_test.shape)

      (467, 6)
      (467,)
      (117, 6)
      (117,)
```

Model Building

By choosing a machine learning algorithm a model is build for Predicting the IMDb Scores for the given dataset

Model Training

Train the model on the training dataset. The model learns the patterns and relationships in the data during this phase.

```
[451] from sklearn.ensemble import RandomForestRegressor
      from sklearn.metrics import mean_squared_error
```

```
[452] rf = RandomForestRegressor(n_estimators=15,max_depth=15,random_state=0,criterion='squared_error')
      rf.fit(x_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=15, n_estimators=15, random_state=0)
```

Model Validation

Validate the model on the test set to assess its generalization performance. This step ensures that the model can make accurate predictions on unseen data.

```
y_pred=rf.predict(x_test)
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
RMSE:  0.845
```

Model Evaluation

Evaluation metrics for regression models are used to assess the performance of models that predict continuous numeric values. These metrics help to understand how well the regression model is making predictions and are crucial for model selection, hyperparameter tuning, and comparing different regression algorithms. Here are some common evaluation metrics for regression models:

1. Mean Absolute Error (MAE):

- Measures the average absolute difference between actual and predicted values.

- Calculation: (1/n) Σ |actual - predicted|

2. Mean Squared Error (MSE):

- Measures the average of the squared differences between actual and predicted values.

- Calculation: (1/n) Σ (actual - predicted)^2

3. Root Mean Squared Error (RMSE)**:

- It is the square root of the mean squared error.

- Calculation: √MSE

4. **R-squared (R2) Score**:

- Measures the proportion of the variance in the dependent variable that is predictable from the independent variables.

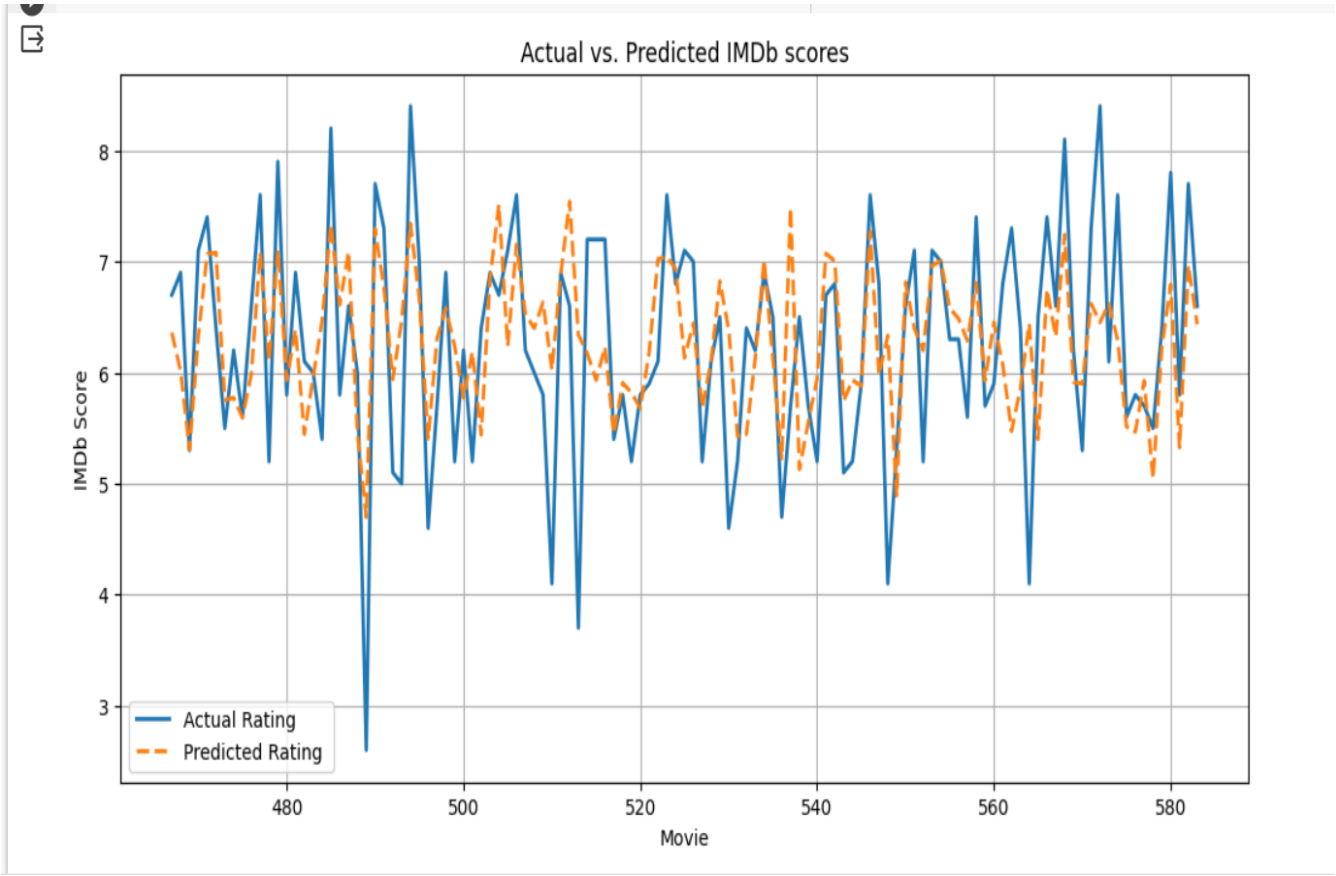
- Calculation: 1 - (MSE(model) / MSE(mean))

```
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)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

```
Mean Absolute Error (MAE): 0.6575585906821325
Mean Squared Error (MSE): 0.7138135847776027
Root Mean Squared Error (RMSE): 0.8448748929738666
R-squared (R2): 0.31227950251748593
```

Visualization of Random Forest Regressor

```
df_range=df.index[-len(y_test):]

plt.figure(figsize=(12,6))
plt.plot(df_range,y_test,label='Actual Rating ',linewidth=2)
plt.plot(df_range,y_pred,label='Predicted Rating ',linestyle='--',linewidth=2)
plt.title("Actual vs. Predicted IMDb scores")
plt.legend()
plt.xlabel('Movie')
plt.ylabel('IMDb Score')
plt.grid()
plt.show()
```



Other Models we built

1. Linear Regression

```
[44] from sklearn.linear_model import LinearRegression

     lr=LinearRegression()

     lr.fit(x_train,y_train)
```

```
LinearRegression
LinearRegression()
```

```
from sklearn import metrics
y_pred=lr.predict(x_test)
rmse1 = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse1)
```

```
RMSE:  1.0
```

```python
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)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

```
Mean Absolute Error (MAE): 0.8160952869715199
Mean Squared Error (MSE): 0.9993713758390388
Root Mean Squared Error (RMSE): 0.9996856385079456
R-squared (R2): 0.03716012917302325
```

2. Decision Tree

```python
[47] from sklearn.tree import DecisionTreeRegressor
     dt=DecisionTreeRegressor(random_state=0)
     dt.fit(x_train,y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

```python
[48] y_pred=dt.predict(x_test)
     rmse2=float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
     print("\nRMSE: ", rmse2)
```

```
RMSE:  1.221
```

```
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)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

```
Mean Absolute Error (MAE): 0.8914529914529914
Mean Squared Error (MSE): 1.4903418803418804
Root Mean Squared Error (RMSE): 1.2207955931858046
R-squared (R2): -0.43586320185693217
```

3. Support Vector Regressor

```
[50] from sklearn.svm import SVR
     svr=SVR(kernel='linear')
     svr.fit(x_train,y_train)
```

```
SVR
SVR(kernel='linear')
```

```
y_pred=svr.predict(x_test)
rmse3=float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse3)
```

```
RMSE:  1.025
```

```
[52] 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)
     rmse = np.sqrt(mse)
     r2 = r2_score(y_test, y_pred)

     print(f"Mean Absolute Error (MAE): {mae}")
     print(f"Mean Squared Error (MSE): {mse}")
     print(f"Root Mean Squared Error (RMSE): {rmse}")
     print(f"R-squared (R2): {r2}")
```

```
Mean Absolute Error (MAE): 0.8163283215996394
Mean Squared Error (MSE): 1.0501367297588973
Root Mean Squared Error (RMSE): 1.0247617917149805
R-squared (R2): -0.011749523426987762
```

4. Gradient Boosting Regressor

```
[53] from sklearn.ensemble import GradientBoostingRegressor
     params={'n_estimators': 10,
             'learning_rate': 0.01,
             'criterion':'squared_error'}
     gbr=GradientBoostingRegressor(**params)
     gbr.fit(x_train,y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(criterion='squared_error', learning_rate=0.01,
                          n_estimators=10)
```

```
y_pred=gbr.predict(x_test)
```

```
[55] 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)
     rmse = np.sqrt(mse)
     r2 = r2_score(y_test, y_pred)

     print(f"Mean Absolute Error (MAE): {mae}")
     print(f"Mean Squared Error (MSE): {mse}")
     print(f"Root Mean Squared Error (RMSE): {rmse}")
     print(f"R-squared (R2): {r2}")
```

```
Mean Absolute Error (MAE): 0.7970444173047133
Mean Squared Error (MSE): 0.9984548130175558
Root Mean Squared Error (RMSE): 0.9992271078276228
R-squared (R2): 0.03804318751347302
```

5. Logistic Regression

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

data=df

data['IMDB_Score_Category'] = pd.cut(data['IMDB Score'], bins=[0, 5, 7, 10], labels=['Low', 'Medium', 'High'])

X = data[['Runtime']]
y = data['IMDB_Score_Category']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 69.23%
```

6. Polynomial Regression

```
x = data['Runtime'].values.reshape(-1, 1)
y = data['IMDB Score'].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

degree=2

poly_features = PolynomialFeatures(degree=degree)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)


model = LinearRegression()
model.fit(X_train_poly, y_train)


y_pred = model.predict(X_test_poly)


mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")


plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.title('IMDb Scores Prediction (Polynomial Regression)')
plt.xlabel('Runtime')
plt.ylabel('IMDB Score')
plt.legend()
plt.show()
```

```
Mean Squared Error: 0.9882182648225293
R-squared: 0.04790554397083957
```