Predicting a Movie's Gross Revenue

Project Report

Olgu Altintas

23.12.2022

Project Description:

A dataset that contains information about movies is given and a model that is able to predict the gross revenue of a movie is requested.

Aim of the Report:

The aim of this report is to explain the working flow of the project and justify the operations and decisions in the project.

What is the working flow?

The code consists of five main parts:

1. Importing the Data and Libraries
2. Exploratory Data Analysis
3. Data Preprocessing
4. Applying Models
5. Submission

Tools:

Only programming language that is used in this project is python. Libraries that are used in this project are pandas, numpy, matplotlib, seaborn and scikit-learn

Outputs:

Outputs of this project are the code file with .ipynb extension, the submission file with .csv extension and the project report.

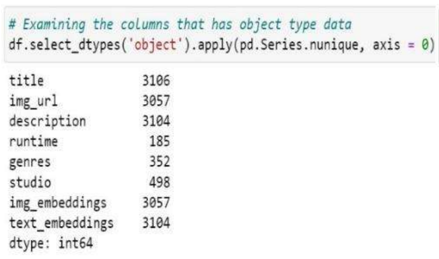
Working flow?

1-) Importing the Data and Libraries

This part is the first and the most basic part of the coding pipeline. Necessary libraries for are imported and the csv files are read.

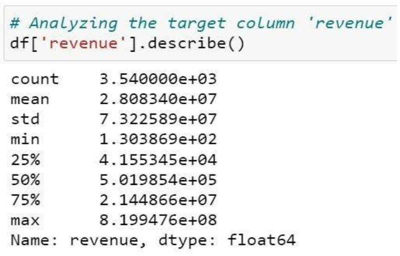
2-) Exploratory Data Analysis

The aim of this part is to understand the data frame that has been provided. In this part, multiple analyses and visualizations are done in order to obtain an idea about the data.



```
# Examining the columns that has object type data
df.select_dtypes('object').apply(pd.Series.nunique, axis = 0)
```

```
title              3106
img_url            3057
description        3104
runtime             185
genres              352
studio              498
img_embeddings     3057
text_embeddings    3104
dtype: int64
```

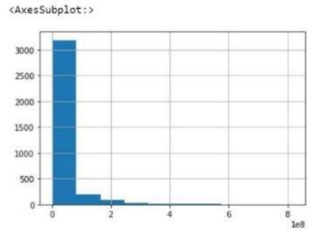


```
# Analyzing the target column 'revenue'
df['revenue'].describe()
```

```
count    3.540000e+03
mean     2.808340e+07
std      7.322589e+07
min      1.303869e+02
25%      4.155345e+04
50%      5.019854e+05
75%      2.144866e+07
max      8.199476e+08
Name: revenue, dtype: float64
```

```
# Visualizing the target column
df['revenue'].hist()
```

```
<AxesSubplot:>
```



3-) Data Preprocessing

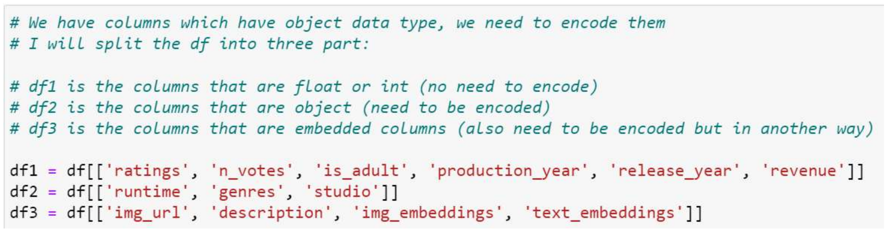
The data preprocessing part is the whole process that is done in order to obtain the final versions of the data frames that can be used in models. This part consists of four subsets.

At first glance, I thought the title column was irrelevant to this task so I dropped it before any other data processing operation.

3.1-) Label Encoding

Data that can be used in models must include only numerical values. However, the dataset that is given for this task has both numeric and non-numeric data. Those non-numeric data need to be encoded in order to be used in a model.

There were seven columns that needed to be encoded ('runtime', 'genres', 'studio', 'img_url', 'description', 'img_embeddings', 'text_embeddings'). I have split them into two data frames because the encoding operation that is going to be applied to them was different.



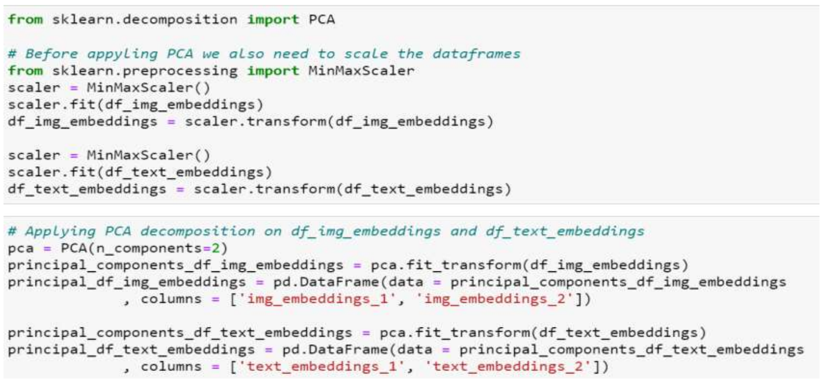
```
# We have columns which have object data type, we need to encode them
# I will split the df into three part:

# df1 is the columns that are float or int (no need to encode)
# df2 is the columns that are object (need to be encoded)
# df3 is the columns that are embedded columns (also need to be encoded but in another way)

df1 = df[['ratings', 'n_votes', 'is_adult', 'production_year', 'release_year', 'revenue']]
df2 = df[['runtime', 'genres', 'studio']]
df3 = df[['img_url', 'description', 'img_embeddings', 'text_embeddings']]
```

I used LabelEncoder of the scikit-learn library for the df2. Using a for loop, I have successfully encoded the columns 'runtime', 'genres' and 'studio'.

For df3, I needed another approach because there were embedding columns. I needed to reduce the multidimensionality of embedding columns.

First, I dropped 'img_url' and 'description' columns because embedding vectors were created using those columns in models so the information in them is already provided in the embedding vectors.

I applied PCA decomposition to them and reduced the multidimensionality. When I am done, I obtain two vectors corresponding to each embedding vector. I used scikit-learn library for PCA decomposition. Before applying PCA decomposition, data needed to be scaled so I used a min-max scaler.



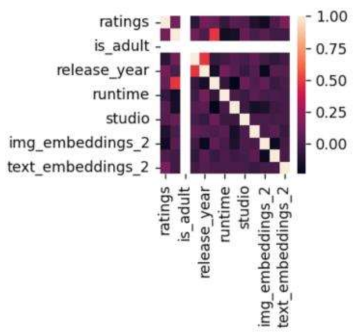
```
from sklearn.decomposition import PCA

# Before applying PCA we also need to scale the dataframes
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df_img_embeddings)
df_img_embeddings = scaler.transform(df_img_embeddings)

scaler = MinMaxScaler()
scaler.fit(df_text_embeddings)
df_text_embeddings = scaler.transform(df_text_embeddings)
```

```
# Applying PCA decomposition on df_img_embeddings and df_text_embeddings
pca = PCA(n_components=2)
principal_components_df_img_embeddings = pca.fit_transform(df_img_embeddings)
principal_df_img_embeddings = pd.DataFrame(data = principal_components_df_img_embeddings
              , columns = ['img_embeddings_1', 'img_embeddings_2'])

principal_components_df_text_embeddings = pca.fit_transform(df_text_embeddings)
principal_df_text_embeddings = pd.DataFrame(data = principal_components_df_text_embeddings
              , columns = ['text_embeddings_1', 'text_embeddings_2'])
```

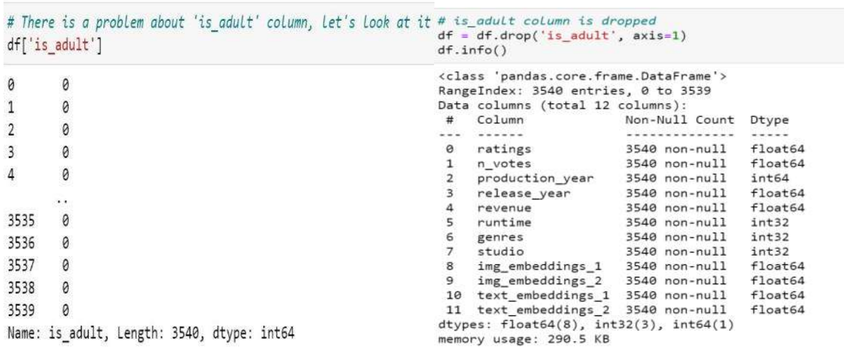
3.2-) Feature Engineering

Having transformed every column of the data frame into numerical columns, feature engineering needed to be done in order to decide which columns are going to be used in the final data frame that is going to be used in models.

I started this task by looking at the correlation heatmap using the seaborn library.



A problem in 'is_adult' column can easily be observed by looking at the heatmap. Each element of the 'is_adult' column is zero. Therefore, I have dropped it.



```
# There is a problem about 'is_adult' column, let's look at it
df['is_adult']
```

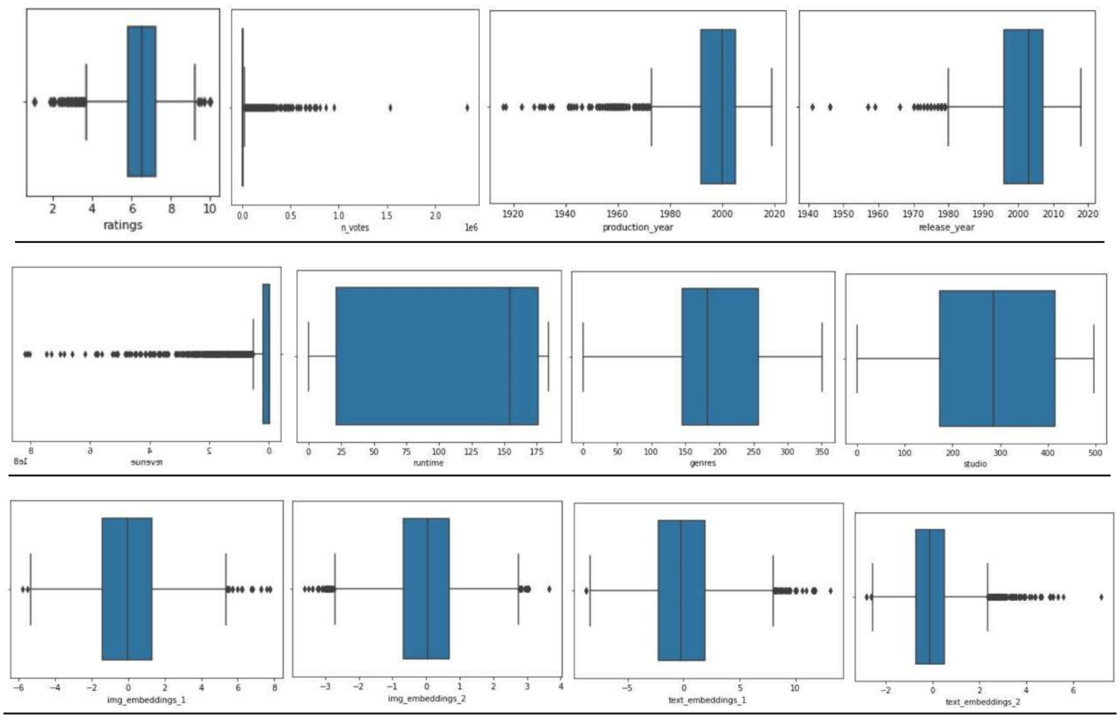
```
0       0
1       0
2       0
3       0
4       0
       ..
3535    0
3536    0
3537    0
3538    0
3539    0
Name: is_adult, Length: 3540, dtype: int64
```

```
# is_adult column is dropped
df = df.drop('is_adult', axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3540 entries, 0 to 3539
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ratings           3540 non-null   float64
 1   n_votes           3540 non-null   float64
 2   production_year   3540 non-null   int64  
 3   release_year      3540 non-null   float64
 4   revenue           3540 non-null   float64
 5   runtime           3540 non-null   int32  
 6   genres            3540 non-null   int32  
 7   studio            3540 non-null   int32  
 8   img_embeddings_1  3540 non-null   float64
 9   img_embeddings_2  3540 non-null   float64
 10  text_embeddings_1 3540 non-null   float64
 11  text_embeddings_2 3540 non-null   float64
dtypes: float64(8), int32(3), int64(1)
memory usage: 290.5 KB
```

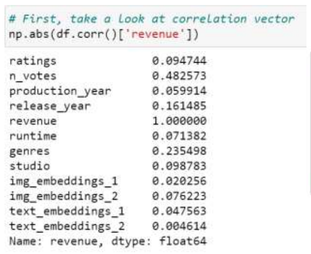
3.3-) Visualizing Outliers

Normally, it would be better to perform visualizing outliers in the exploratory data analysis step. However, data needs to be in a numerical format to visualize. Therefore, I visualized outliers after label encoding and feature engineering process. Outliers can affect the performance of the models so they have to be considered in the applying models' step.



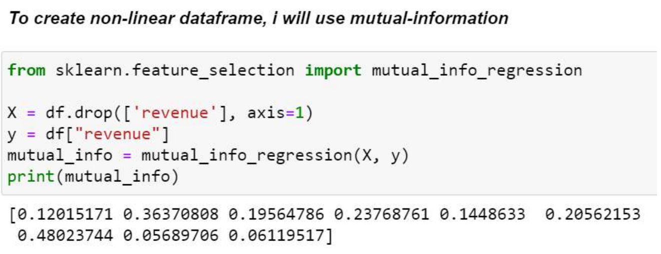
3.4-) Feature Selection

I used both linear and non-linear models for the task. Therefore, I needed two different data frames for the models because the feature selection criteria for linear and non-linear models were different. I used correlation for linear models and mutual information for non-linear models. For mutual information, I used scikit-learn library.



```
# First, take a look at correlation vector
np.abs(df.corr()['revenue'])
```

```
ratings                0.094744
n_votes                0.482573
production_year        0.059914
release_year           0.161485
revenue                1.000000
runtime                0.071382
genres                 0.235498
studio                 0.098783
img_embeddings_1       0.020256
img_embeddings_2       0.076223
text_embeddings_1      0.047563
text_embeddings_2      0.004614
Name: revenue, dtype: float64
```



To create non-linear dataframe, i will use mutual-information

```
from sklearn.feature_selection import mutual_info_regression

X = df.drop(['revenue'], axis=1)
y = df["revenue"]
mutual_info = mutual_info_regression(X, y)
print(mutual_info)
```

```
[0.12015171 0.36370808 0.19564786 0.23768761 0.1448633  0.20562153
 0.48023744 0.05689706 0.06119517]
```

After feature selection, data frames were ready and I started the model step.

4-) Applying Models

For the task, I used four models: Linear Regression, KNN, MLP and Random Forest Regression.

I used scikit-learn library for every model.

4.1-) Linear Models

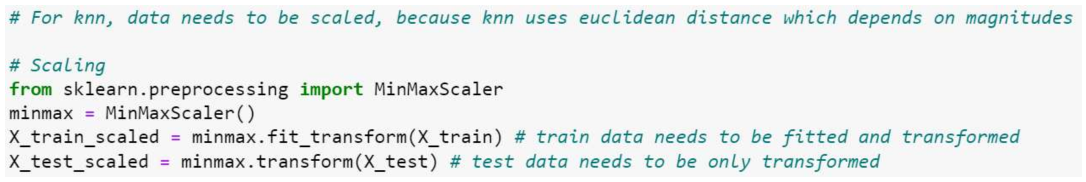
4.1.1-) Linear Regression:

Linear Regression was the only linear model for this task. I used the data frame that I created using correlation whereas I used another data frame for non-linear models which I created using mutual information.

4.2-) Non-Linear Models

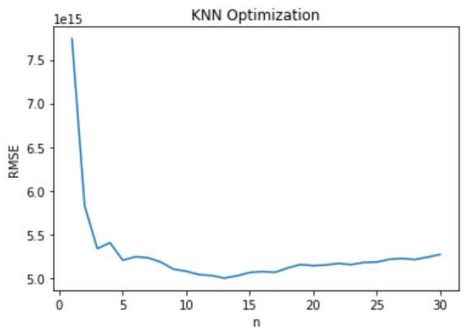
4.2.1-) KNN Regression:

For, KNN Regression data needed to be scaled because KNN uses Euclidean distance which is sensitive to magnitudes. I used min-max scaler.



```
# For knn, data needs to be scaled, because knn uses euclidean distance which depends on magnitudes

# Scaling
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()
X_train_scaled = minmax.fit_transform(X_train) # train data needs to be fitted and transformed
X_test_scaled = minmax.transform(X_test) # test data needs to be only transformed
```

I also applied grid search in order to optimize KNN and find the best choice for n.



4.2.2-) MLP Regression:

For MLP Regression, I have selected 25000 for the maximum iterations because that was the minimum number of iterations that satisfies convergence. I thought numbers bigger than 25000 could increase the risk of overfitting so I used 25000.

I also applied grid search to optimize MLP regression. For the optimized model, I used identity activation function and invscaling learning rate.

```
# Applying optimized mlp

# After several try, 25000 is found out to be the min number of max_iteration that satisfies convergence
# 25000 is the best choice for max iterations because after that, risk of overfitting increases

from sklearn.neural_network import MLPRegressor
mlp_optimized = MLPRegressor(max_iter=25000, activation='identity', learning_rate='invscaling')
mlp_optimized.fit(X_train, y_train)
predictions_mlp_optimized = mlp_optimized.predict(X_test)
```

4.2.3-) Random Forest Regression:

In the task, we are asked to apply Linear Regression, KNN and MLP regression. Last model (or models) selection was given to us.

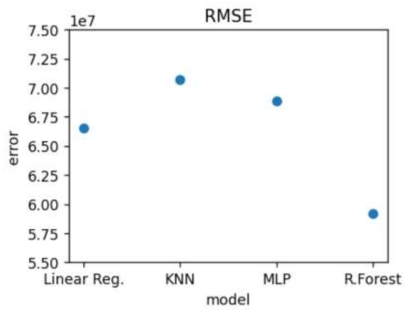
When I visualize the outliers, I observed that the data has multiple outliers that can affect the performance of the models. Linear regression, KNN and MLP are models that are sensitive to outliers. Therefore, I thought it would be great idea to apply a model that is robust to outliers. This is why I chose Random Forest Regression as my last model.

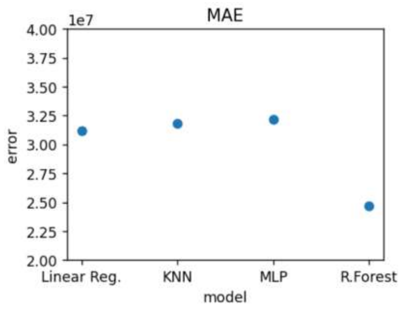
For, Random Forest Regression I also applied grid search and find optimized values for max_depth, min_samples_split, n_estimators and random_state.

4.3-) Comparison

I compared the performance of each model according to both RMSE and MAE (even though RMSE was the important one, I also wanted to use MAE in case of any different results).

```
RMSE Values for each model:
--------------------------------
Linear Regression: 66.58m
KNN Regression: 70.62m
MLP Regression: 68.92m
Random Forest Regression: 59.18m


MAE Values for each model:
--------------------------------
Linear Regression: 31.16m
KNN Regression: 31.80m
MLP Regression: 31.92m
Random Forest Regression: 24.72m
```



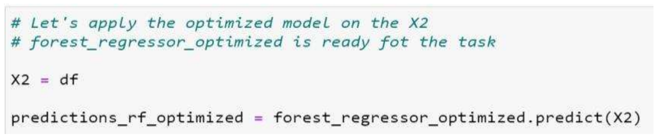


I observed that random forest regression was the most successful model for this task.

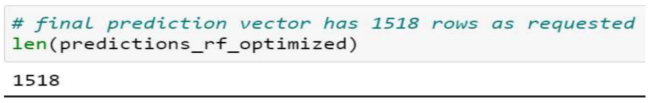
5-) Submission

I applied the same data preprocessing, label encoding, feature engineering and feature selection operations to the second csv file X2.

After that I obtained the final data frame for X2 and I applied optimized random forest model to it.



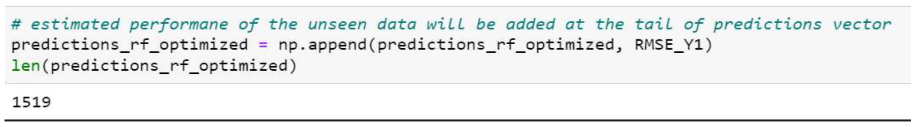
```
# Let's apply the optimized model on the X2
# forest_regressor_optimized is ready fot the task

X2 = df

predictions_rf_optimized = forest_regressor_optimized.predict(X2)
```



```
# final prediction vector has 1518 rows as requested
len(predictions_rf_optimized)
```

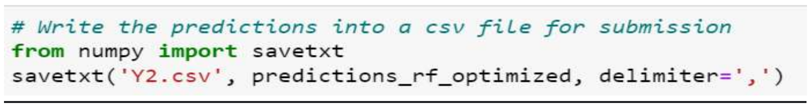
```
1518
```

RMSE value was added at the end of the vector and it is ready for submission.



```
# estimated performane of the unseen data will be added at the tail of predictions vector
predictions_rf_optimized = np.append(predictions_rf_optimized, RMSE_Y1)
len(predictions_rf_optimized)
```

```
1519
```



```
# Write the predictions into a csv file for submission
from numpy import savetxt
savetxt('Y2.csv', predictions_rf_optimized, delimiter=',')
```

Discussion:

- Random Forest Regression model was the best model for this task. I think main reason is because Random Forest is robust to outliers
- Even though MLP Regression is more complex and has a long running time, it was not the most successful model for this task. What we can observe from that is it would be wrong to say that longer the running time, better the performance
- Data visualization usually done in Exploratory Data Analysis step but it can also be done again, after some data preprocessing operations
- Dealing with embedding vectors was a hard task. Understanding the aim of statistical methods like PCA, ICA can be really helpful to solve such problems
- There are lots of tools that we can use in the coding process which makes our job a lot easier. However, main part of a machine learning project is not actually coding it. The main part is to think about the task, examine the data again and again, make decisions and comment on the outputs obtained in order to get better results in the future