**Project Report**

**GitHub URL**

<https://github.com/PauStan/Assignment-Specialist-Certificate-in-Data-Analytics->

**Abstract**

Movie critics vote (from 1 to 10) on movie titles all the time. Individual votes are then aggregated and summarised as a single rating, visible to anyone interested in watching a film. Using Python this project used two types of machine learning models ( Decision Tree Regressor and Random Forest Regressor) to try and predict a movies vote based on budget, revenue, popularity, runtime, and vote count.

**Introduction**

I chose this project use case as I have always been interested in the movie industry. In today’s world, the film industry is becoming more and more competitive. According to (IMBd), the average number of films produced every year is 2577. I was interested to see if certain parameters could be used to predict a movie vote rating. A model like this could help some of these film companies and directors to predict a vote rating before filming begin. Therefore, they could optimise the vote rating by adjusting to the model predictions. The project could potentially help film investors improve the rating of their movie and lower risks.

**Dataset**

For this project I used the (The Movie Database, 2017). This dataset was put together by Kaggle generated from The Movie Database API. I chose this data source as TMDb had a large data set put together by a community of users. TMDb has become the premiere spurce for this type of data. The data set contains information about 10,000 movies collected from The Movie Database (TMDb), including user ratings and revenue

There are two datasets used, tmdb\_5000\_credits and tmdb\_5000\_movies. The first one contains 4803 observations with the following columns:

Movie id, Title, Cast and Crew

The second dataset contains 4803 observations with the following columns:

Budget, genres, homepage, id, keywords, original language original title, overview, popularity, production companies, production countries, release date, revenue runtime, spoken languages, status, tagline, title, vote average and vote count.

**Implementation Process**

**1./2. Real-world scenario/Importing Data**

The first thing to do was import all the necessary libraries to be used.

import pandas as pd

import numpy as np

import re

from ast import literal\_eval

from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier

from sklearn.linear\_model import LinearRegression

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

from sklearn.metrics import mean\_absolute\_error

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

Next, I loaded in the data set using pandas read\_csv() utility.

df\_movies = pd.read\_csv("E:/Paul/UCD/tmdb\_5000\_movies.csv")

1. **Analysing Data**
   1. **Merge DataFrames**

It could now check the head of the data to get some information on the contents of the data frame. I started by returning the first three rows.

df\_movies.head(3)

I decided to load another data set to add more data to the current data frame.

df\_credits = pd.read\_csv("E:/Paul/UCD/tmdb\_5000\_credits.csv")

I checked the shape of the array, looked for duplicates and the data types of both data sets.

df\_credits.head(3)

print(df\_credits.shape)

print(df\_movies.shape)

sum(df\_credits.duplicated())

sum(df\_movies.duplicated())

df\_credits.dtypes

df\_movies.dtypes

I then decided to merge both datasets on shared a column call ‘id’ and ‘movie\_id’.

df= df\_movies.merge(df\_credits,on='id')

Before merging I also changed the name of some columns in the credits data set.

df\_credits.columns = ['id','title2','cast','crew']

I did this so the ‘id’ column name in both data sets would match. It also ensured the duplicate title column in the credits data had a unique name.

I returned the first three rows of the newly merged dataframes and checked the shape of the new array to ensure everything looked correct. Using the describe or info commands you can get more information on the dataset. It is important to do this to understand the dataset you are working with.

df.head(3)

df.shape

df.describe()

* 1. **Replace missing values or drop duplicates**

I noticed that some columns would not be needed as part of my analysis, so I dropped these.

df = df.drop(columns = ['homepage', 'title','title2', 'status', 'spoken\_languages'])

I then checked to see if there were any missing values in the dataset.

null = df.isna().sum()

print("Number of null values in the dataset are null: ", null)

I discovered that there were eight hundred and forty-four missing values for tagline. This seemed like a lot of rows to remove from the data set, so I decided to replace the missing values with ‘No Tagline’.

df.tagline.fillna('No Tagline', inplace=True)

There were also three missing values for overview, one for release date and two for runtime. This accounted for an extremely low percentage of the overall data, so I decided to drop the six null values.

df.dropna(axis=0, inplace=True)

* 1. **Making use of Regex to extract a pattern in data**

I also noticed that the genre column contained a list of dictionaries. I was only interested in the value of the key ‘name’ so decided to extract this using regex.

pattern = r"[^a-zA-Z\_]+"

def extract(column):

rows\_list = []

for row in column:

string = re.sub(pattern=pattern, repl=' ', string=row, flags=re.IGNORECASE)

rows\_list.append(string.replace('id', '').replace('name', '').split(' '))

return list(rows\_list)

The function uses a regular expression to subtract any non-string character and replaces it with a white space. It then replaces ‘id’ and ‘name’ with an empty string.

I updated the genre column by running this extract function.

df['genres'] = extract(df['genres'])

When I checked the updated genre column there were some white spaces that would now need to be removed.

df['genres'][0]

I did this with another function that stripped the white spaces.

def strip(col):

column\_list = []

for row in col:

row = [x.strip(' ') for x in row if x]

column\_list.append(row)

return column\_list

I updated the genre column with this function and checked it to make sure everything looked correct.

df['genres'] = strip(df['genres'])

df['genres'][0]

1. **Python**
   1. **Define a custom function to create reusable code**

The keyword column had similar issues, so I reused the above functions to rectify this.

df['keywords'] = extract(df['keywords'])

df['keywords'] = strip(df['keywords'])

* 1. **Dictionary or Lists**

The cast and crew columns also had data that was difficult to interpret. Both were a list of strings, so I converted both to dict.

features = ["cast", "crew"]

for feature in features:

df[feature] = df[feature].apply(literal\_eval)

* 1. **NumPy**

I then created functions to get the cast and director names.

def get\_director(x):

for i in x:

if i["job"] == "Director":

return i["name"]

return np.nan

def get\_list(x):

if isinstance(x, list):

names = [i["name"] for i in x]

if len(names) > 3:

names = names[:3]

return names

return []

df["director"] = df["crew"].apply(get\_director)

features = ["cast"]

for feature in features:

df[feature] = df[feature].apply(get\_list)

1. **Machine Learning**

Next, I wanted to demonstrate machine learning using the budget and revenue columns by implementing linear regression. I started by splitting the dataset into feature and target variables.

X = df['budget'].values

y = df['revenue'].values

Both the feature and target variables arrays need to two-dimensional so there was a requirement to reshape both.

X = X.reshape(-1,1)

y = y.reshape(-1,1)

By specifying a train\_size of 0.8, I wanted to put 80% of the data into the training set, and the rest of the data into the test set.

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=.8, test\_size=.2,random\_state=100)

The next step was to create a linear regression model and fit it using the existing data.

lm = LinearRegression()

lm.fit(x\_train,y\_train)

To obtain the predicted response, I used lm.predict.

y\_predict = lm.predict(x\_test)

I wanted to check the accuracy of my model so used lm.score

print(f'Train Accuracy {round(lm.score(x\_train, y\_train)\* 100,2)}%')

print(f'Test Accuracy {round(lm.score(x\_test, y\_test)\* 100,2)}%')

I went on to look at using a Decision Tree Regressor to predict a target variable. I started this by selecting the target variable and calling it y.

y = home\_data['vote\_average']

I then created a data frame call X. It holds the predictive features

feature\_names = ['budget', 'revenue', 'popularity', 'runtime', 'vote\_count']

X = df[feature\_names]

In the next step, I split the dataset into the training set and the test set. For this I used a test size of 0.25 which means that 25% of the data rows will only be used as test set and the remaining rows will be used as training set for building the model.

train\_X, val\_X, train\_y, val\_y = train\_test\_split(X, y, test\_size=0.25, random\_state=1)

I assigned the DecisionTreeRegressor class from sklearn.tree to the variable ‘vote\_model’. Then fit the X\_train and the y\_train to the model by using the .fit function.

vote\_model = DecisionTreeRegressor(random\_state=1, max\_depth =3)

vote\_model.fit(train\_X,train\_y)

In this step, I predicted the results of the test set with the model trained on the training set values using the .predict function**.**

val\_predictions = vote\_model.predict(val\_X)

I then compared the **‘Actual Values**’ and ‘**Predicted Values’**.

comparison\_dict = {"Actual":val\_y.head(), "Prediction":val\_predictions[0:5]}

comparison\_df = pd.DataFrame(comparison\_dict)

comparison\_df["Error"] = abs(comparison\_df["Actual"] - comparison\_df["Prediction"] )

comparison\_df

To calculate the mean error and mean absolute error I used the following

comparison\_df["Error"].mean()

mean\_absolute\_error(val\_predictions, val\_y

I was interested to see if I could reduce the mean absolute error using hyperparameter tuning. I created a function that returns the mean absolute error and a four loop to apply different max depth values to the function.

* 1. **Hyperparameter tuning**

def mae (max\_depth):

vote\_model\_1 = DecisionTreeRegressor(max\_depth = depths, random\_state=1)

vote\_model\_1.fit(train\_X,train\_y)

val\_predictions\_1 = vote\_model\_1.predict(val\_X)

mae\_1 = mean\_absolute\_error(val\_predictions\_1, val\_y)

return (mae\_1)

lst=[]

depth = range(1,15,1)

for depths in depth:

maes = mae(depths)

lst.append(maes)

print(depths, maes)

I found that a max depth of 7 gave the best result and created a model with this value. When I checked how accurate the model was you could see the predictions improve.

vote\_model\_2 = DecisionTreeRegressor(max\_depth = 7, random\_state=1)

vote\_model\_2.fit(train\_X,train\_y)

val\_predictions\_2 = vote\_model\_2.predict(val\_X)

mae\_2 = mean\_absolute\_error(val\_predictions\_2, val\_y)

comparison\_dict\_2 = {"Actual":(val\_y.head()), "Prediction":val\_predictions\_2[0:5]}

comparison\_df\_2 = pd.DataFrame(comparison\_dict\_2)

comparison\_df\_2["Prediction"] = (comparison\_df\_2["Prediction"])

comparison\_df\_2["Difference"] = ((comparison\_df\_2["Actual"] - comparison\_df\_2["Prediction"]))

comparison\_df\_2

I wanted to compare models, using Random Forest as another model type. I set a range of hyperparameters. Defined the model, setting the random state to 1.

max\_features\_range = np.arange(1,4,1)

n\_estimators\_range = np.arange(90,110,5)

max\_depth\_range = np.arange(5,15,1)

param\_grid = dict(max\_features=max\_features\_range, n\_estimators=n\_estimators\_range, max\_depth=max\_depth\_range)

rf = RandomForestRegressor(random\_state=1)

Grid search allowed me to test the model at the ranges I had specified above.

grid = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5)

I then fit the model and printed the best parameters.

grid.fit(train\_X, train\_y)

grid.best\_params\_

I then found the mean absolute error with these parameters.

rf\_vote\_model = RandomForestRegressor(random\_state=1,\*\*grid.best\_params\_)

rf\_vote\_model.fit(train\_X,train\_y)

pred = rf\_vote\_model.predict(val\_X)

rf\_val\_mae = mean\_absolute\_error(val\_y, pred)

print("Mean absolute error using Random Forest Model:", (rf\_val\_mae))

The mean absolute error reduced using the random forest model. This shows that it gives a more accurate result.

**Results**

1. **Visualise**

First, I decided to analyse the linear regression of budget and revenue. I was interested in seeing how investment in a movie affected revenue. I did this by creating a scatter plot of my model.

plt.scatter(X\_train,y\_train, s=20, c='b', marker='o')

plt.plot(X\_test, y\_predict, color="black", linewidth=3)

plt.xlabel('Budget')

plt.ylabel('Revenue')

plt.title('Linear Regression - Budget vs Revenue')

plt.show()

Chart, scatter chart

Description automatically generated

Figure 1 - Scatter plot of linear regression

The black line is the best fit and you can see from this graph there is a positive linear relationship between budget and revenue. This tells us as the budget of a movie increases the revenue should increase too. Using this model, we could predict y from any values of x. There are quite a lot of data points with low budget amounts and less with high budgets. This means the higher the budget the less reliable the model could be. In order to improve this model, you could analyse some of the outliers. The biggest revenue value and biggest budget value are much larger than the other data points so it would be useful to verify these. It looks like there could be some data points with a 0 value for budget or 0 value for revenue, investigating these could improve the model.

Next, I plotted the max depth and mean absolute error of the decision tree regressor model.

plt.plot(depth, lst)

plt.title('DecisionTreeRegressor - Maximum Depth Vs Mean Absolute Error ')

plt.xlabel('Maximum Depth')

plt.ylabel('Mean Absolute Error')

plt.show()

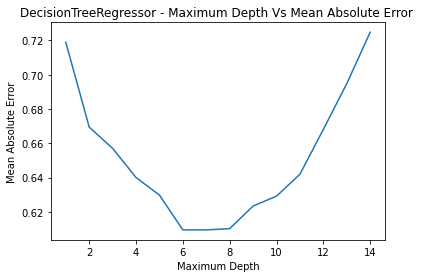


Figure 2 - Max depth and mean absolute error of the decision tree regressor model

The plot shows how the mean absolute error changes as maximum depth changes. This graph acts as a visualisation tool in selecting the correct maximum depth parameter. It helps to ensure the model has been tuned correctly. In this case it shows values for maximum depth of 6, 7 and 8 could give a smallest mean absolute error. Upon printing the results, you can see that a depth of 6 actually gives the lowest mean absolute error value.

Depth 6 Mean Absolute Error 0.6094781646031106

Depth 7 Mean Absolute Error 0.6094678050946551

Depth 8 Mean Absolute Error 0.6102125402332179

I also looked at a rectangular plot of correlation.

f,ax = plt.subplots(figsize = (10,10))

sns.heatmap(df.corr(), annot = True, linewidths=.5, fmt = '.1f', ax = ax)

plt.title('Correlation')

plt.show()

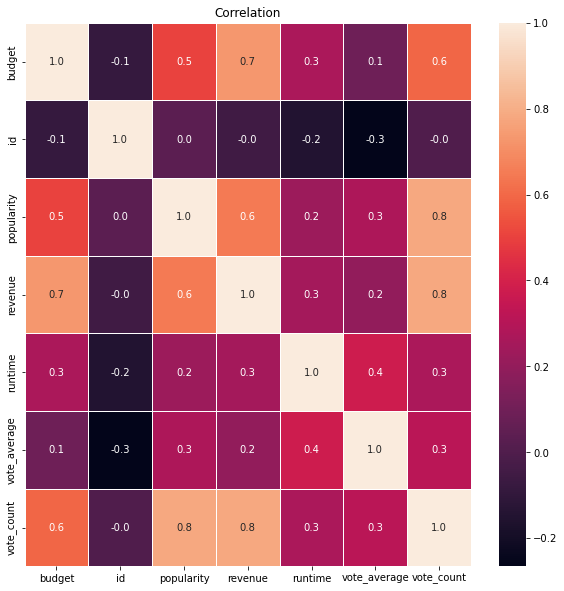


Figure 3 - Correlation between all the numerical variables

The plot shows the correlation between all the numerical variables. I used a heat map to visualise the correlation. The heat map visualises the data through variations in colour, in this case the lighter the colour the higher the corelation. From this we can see the most correlated variables. This shows us that movies with a higher revenue tend to be reviewed a lot. Similarly, budget and revenue have a strong positive correlation. Budget of a movie has negligible effect on the vote rating.

**Insights**

* The analysis seems to indicate that a movies voting score correlates positively with the parameters used (budget, revenue, popularity, runtime, vote count).
* More work could be done to yield models that fit the data better and predict a movies voting score in a more sensible and intuitive way.
* The Random Forest Regressor model produced a lower mean absolute value, indicating that it could a more accurate model.
* It is important to use hyperparameter tuning to tune your model to get the best prediction possible.
* The Random Forest Regressor model took a lot longer to run than the Decision Tree Regressor model. It would be worth exploring the value of the longer run time against overall improved accuracy of the model.
* The analysis also shows as the budget of a movie increases the revenue should increase too.
* More data cleaning could be carried out to remove outliers and 0 values for budget and revenue. This could in turn improve all models.

# **References**

The Movie Database. (2017). TMDB 5000 Movie Dataset [Data set]. <https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata>

IMDb, <https://www.imdb.com/?ref_=nv_home>