# **Introduction and Business Understanding**

**Business problem and goal**

One of the featured products in the entertainment industry is movie. Producing a movie involve huge cost. The cost ranges from 70 to 300m USD. Avatar is a good example of the largest feature film budget. Therefore, it is really important to predict the box office in order to break even or make a handsome profit.

However, there are many attributes in a movie, e.g. language, crew, producer etc. Also, there is no official model to predict the box office due to the complication. This project aims to predict the box office from various model we built that utilized different data mining tasks. We would apply a number of evaluation measures to analyze the models’ performance in order to find the most suitable model for prediction.

# **Data Understanding**

We got the movie dataset “TMDB 5000 Movie Dataset” on Kaggle[[1]](#footnote-2). The dataset recorded 4803 international movies from TMDB, a popular movie and TV database launched in 2008. The dataset originally consisted of 4803 observations and 20 variables. However, not all of the variables were meaningful for data mining and there were some missing values in the dataset. Therefore, we had to perform data preprocessing.

**Feature Selection**We first understood the meaning of variables by researching on the official website of TMDB (see Appendix Fig. 1 for the summary table of variables). There were several variables we would like to highlight. There were 3 types of movie status (released, rumored, post production) and 99.8% of the instances are released, resulting in class imbalance problems. Data mining algorithms might have poor results due to class imbalance. Considering the data of movies which had not been released might be inaccurate and incomplete, we dropped those instances. With reference to the description of variables, we decided to drop a number of variables that were inappropriate for data mining[[2]](#footnote-3).

**Missing Values Handling**We observed some movies having extremely low budget and revenue. It was hardly possible to have budget and revenue lower than USD 1000 (threshold for dropping), we believe those data were incorrect. As budget and revenue were two crucial variables and we did not want to manipulate them by assigning values, we dropped instances with budget or revenue less than 1000.

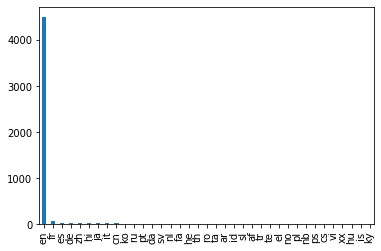
It was found that there were missing values (empty strings “[]”) in genres, production\_companies and production \_countries. As the missing values only accounted for 1.25% of the remaining dataset, we also dropped instances with these empty strings. We spotted that some values in vote\_average were 0 as no one voted for the movie (zero vote\_count). Those values were indeed missing values. Provided that the instances with zero vote\_count was only 2% of the entire dataset, we also considered dropping them.

For release\_date, we first dropped the rows with missing values. We considered that it would be meaningless and irrelevant to keep the exact released date of the movie. We believed that the release month or season might be more meaningful since factors such as inflation and seasonality should have a large effect on prediction of a movie’s revenue. We broke down that release\_date into year, month, date and quarter. We also tried to handling missing values in runtime with filling average number of it (i.e. the average length of a movie).

**One Hot Encoding**For genres, production\_companies and production\_countries, values were stored in dictionary data type which was an unordered collection of items because a single movie could have several genres and could be produced by several countries and companies. Our initial thought was to rank those companies, with top 10 companies preserved and other companies labelled as “others”. However, we realised that it was inappropriate since the “others” attribute would appear in every row. This kind of labeling did not provide any information to our data mining, therefore, we preserved all the companies to retain reliability of the data as much as we can. To enable one-hot-encoding, we extracted these values using excel’s text to columns and VBA, and created a number of columns storing these values.

We realised spoken\_languages is similar to original\_language. As they share the same information, we preserved original\_language only. Unlike the variables like genres or production\_companies, each movie would only be labelled with one language. We tried to count the number of each languages. After reviewing the ranking, we retained the top 10 languages and renamed other languages as “others”. And we performed one-hot-encoding for those 11 variables.

**Normalization**For popularity, there was a relatively wider range in values (875) than other numerical variables like runtime (297) and vote\_average (8.5). Thus, we performed feature normalization for popularity to prevent attributes with wide ranges outweigh those with narrow ranges. We would test whether popularity or popularity\_norm would give a better performance in modelling.



**Data Type**At the end of the data preparation, we carefully checked the data types of every column. We mapped the actual data types with data types in Python (See Appendix Fig. 2 for the data type of each variable) and altered them to the correct data types if needed.

# **Model Building – Tree Model**

After understanding and cleaning the data in section 2, further adjustment on the data is needed for classification tree purposes since some format in pandas may not be available.

**Handle independent variable**

**Remove irrelevant variables for prediction**. Some variables would not be used in the prediction phase in reality before the movie was released or just released, for instance, ‘popularity’ and ‘vote average’. They could be the target variable.

**Categorical variables.** There are around 3k features in the cleaned data set. This is because there are many unique values for some features. The 3k features are the result of one hot encoding them. We trimmed them to generalize the model and to reduce the run time during model construction and prediction.

* Original language: Movies that are not top 5 in terms of frequency would be classified as ‘others’. The 5 most frequently used languages(out of 25) dominated 98% of the data set. The 2% was thus renamed to “others”. One hot encoding was done to create a total of 6 columns: ‘lang\_es','fr':'lang\_fr','ja':'lang\_ja','zh':'lang\_zh','other':'lang\_other'
* Genre: Some movies might have multiple genres. For simplicity, we aggregated the number of different genres, find the 5 most frequent genres (out of total 19) and do one hot encoding (“1”: movie belongs to one or more of the Top 5 genres; “0”: the opposite of “1”). The result showed that 94% movies belonged to “1”.
* Production country: Same as Genre, 1 movie could have more than 1 production country. For simplicity, we aggregated them and did value\_count in order to find the top 5 production country (out of 61). One hot encoding is done and “1” represents movie produced in at least one of the top 5 countries. The result showed that 97% of movies belonged to “1”.
* Production company: Same as the above, 1 movie could have more than 1 production companies. Similar treatment as genre and production country was done. Since there are more than 1000 companies in total, we chose the top 10 companies. The result shows that 49% movies involve one of the top 10 companies.

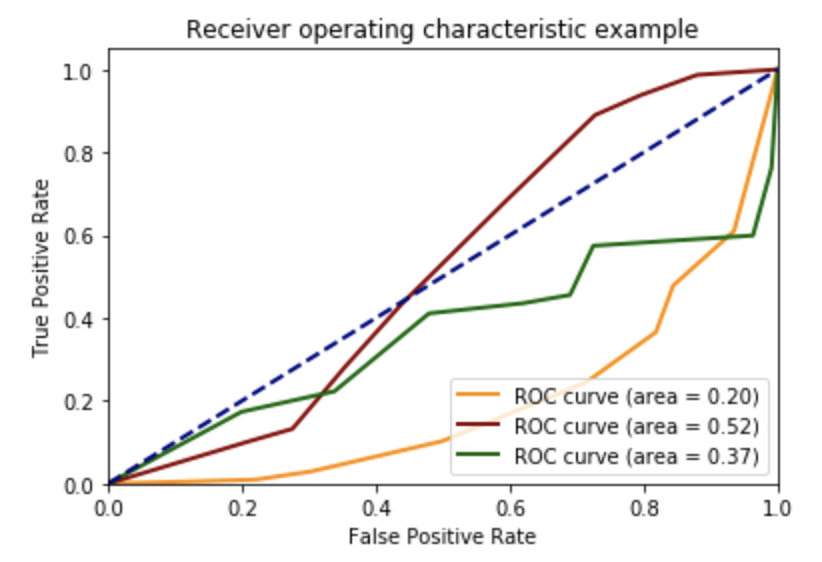
Last but not least, redundant columns are removed to obtain a cleaner data frame.

**Handle target variable**

We would like to predict the revenue given the features. Since we were to do classification problem, the target variable needs to be categorical. We used quantile-based discretization function (qcut) and bin revenue into 3 bins – “0”,”1”,”2” and where “2” represents the highest revenue and “0” the lowest revenue among the dataset. This became a multiclass classification problem.

**1st attempt: Decision Tree**

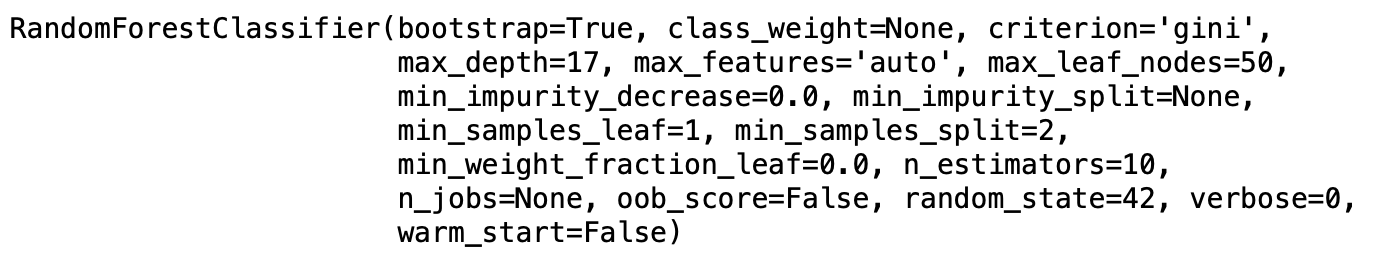
Originally, we planned to do decision tree. We performed the standard procedures, namely, train-test split, cross validation, grid search CV, in order to make the prediction. To seek the highest performance, we also tried to change the parameters, such as test size, max\_leaf\_node, min\_impurity\_decrease. The best result we got for AUC is near or lower than the random classifier, which is not satisfactory. Some possible reasons are unstable tree structure, imbalanced features which make prediction difficult.



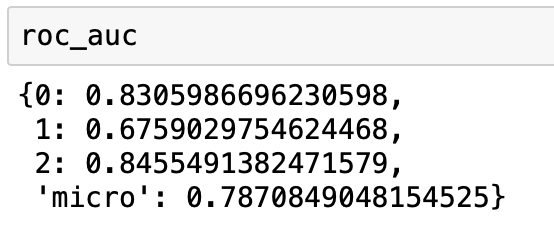
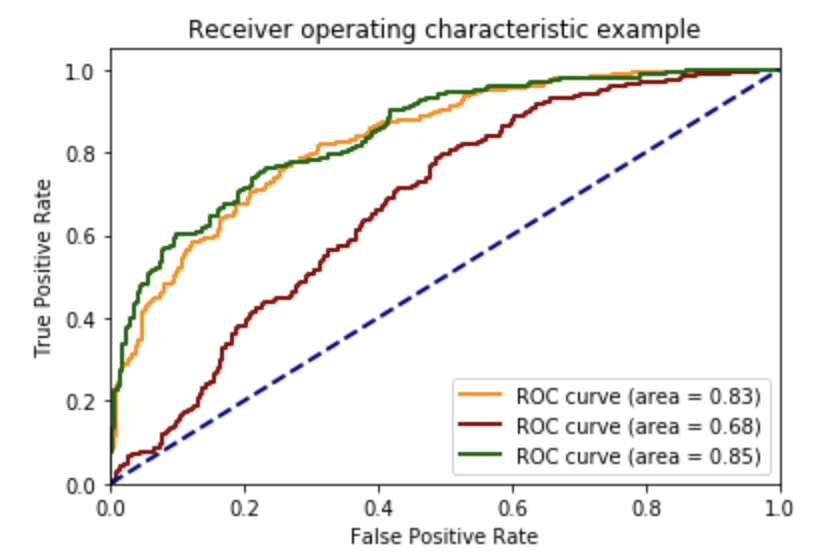
**Random Forest**

In view of the poor result and the shortcomings of a decision tree. We chose to try again with random forest. A random forest is a meta estimator that fits multiple decision tree classifiers on various subsets of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size, but the samples are drawn with replacement if bootstrap=True (default)[[3]](#footnote-4). It is expected that random forest will reduce variance part of error, achieve higher performance, especially for unstable tree structure. It is fast to train. It is useful when the dataset has many outliers, missing values or skewed data, like the movie data set.

First, we defined the features and target variable. Features are [budget, runtime, year, month, date, quarter, lang\_en, lang\_es, lang\_fr, lang\_ja, lang\_other, lang\_zh, Top5Genre, Top5Country, Top10Company]. The target variable is the revenue which was divided into 3 possible classes based on quantile as mentioned above. Second, we split the dataset for training and testing purposes. Test size of 0.2 gave the best performance. Third, Grid Search CV was done to seek the best hyperparameter. We tried max\_depth from 1 to 51 and max\_leaf\_nodes of [5,10,20,50,100]. The best parameters returned are 17 and 50 respectively. Other parameters per below. Fourth, we train the model with RandomForestClassifier with the best parameters (below). Then, we use the model to make predictions on the test set.



After building the model, we tried to change different parameters or even the dataset to adjust the model for a higher performance. The larger the area under curve (AUC) of ROC, the better the model. Below ROC curve is the performance of the model we built:



Orange= “0”, Q1 of the revenue, lower rev

Red: “1”, Q2 of the revenue, medium revenue

Green: “2”, Q3 of revenue, higher revenue

Q: Quantile

Below is a table recording the performance of the model on difference changes. Each change is independent on the others.

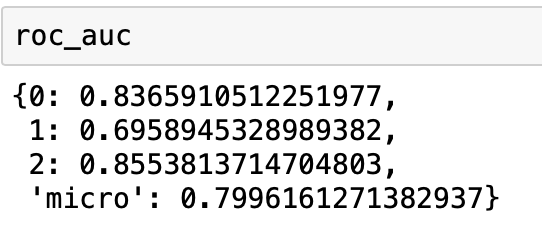
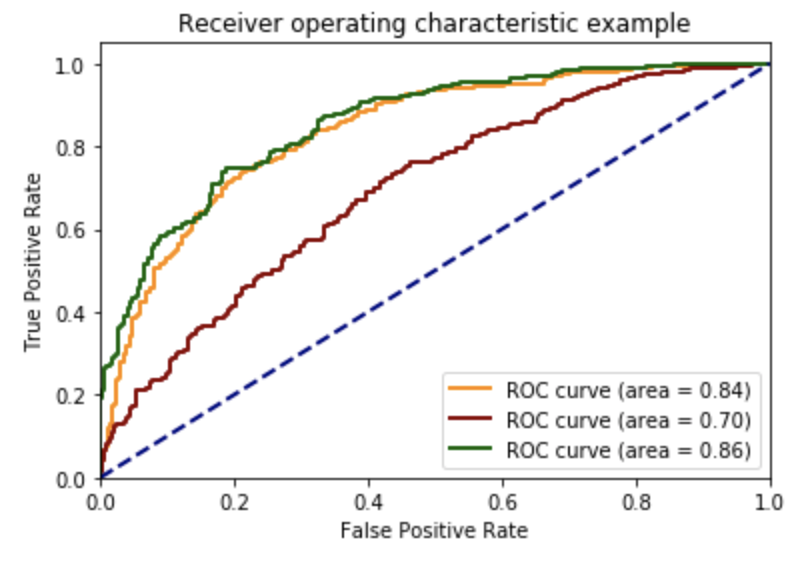
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **ROC\_AUC** | | | |
| **Changes** | **Q1 of rev/low rev**  **(Orange line)** | **Q2 of rev/medium rev**  **(Red line)** | **Q3 of rev/ high rev**  **(Green line)** | **Micro** |
| Model built  - Test size=0.2  - With Grid Search CV (depth 17, leaf node: 50)  - n\_estimator: 100 | 0.83 | 0.68 | 0.85 | 0.79 |
| Test size = 0.25  Test size = 0.4  Test size = 0.6 | 0.81  0.80  0.80 | 0.65  0.64  0.65 | 0.84  0.84  0.83 | 0.78  0.77  0.77 |
| Without Grid Search CV | 0.80 | 0.64 | 0.81 | 0.76 |
| Best max depth; max leaf nodes  9;20  7;50  11;150  47;20 | 0.81  0.82  0.81  0.81 | 0.63  0.64  0.65  0.64 | 0.83  0.83  0.84  0.83 | 0.77  0.77  0.77  0.77 |
| n\_estimator=500  n\_estimator=250  n\_estimator=100  n\_estimator=50  n\_estimator=10 | 0.83  0.83  0.83  0.83  0.82 | 0.68  0.68  0.68  0.68  0.66 | 0.85  0.85  0.85  0.84  0.84 | 0.79  0.79  0.79  0.79  0.78 |
| Dataset with 3k features | 0.84 | 0.70 | 0.86 | 0.80 |

Here are some highlights of the adjustments.

For the hyperparameters, we tried different range for the program to seek for different best parameters. The larger the max depth and max leaf nodes, the higher the complexity of the tree. The results show that the performance of a simpler or more complex tree is similar. However, the one that gave the best performance is a relative moderate tree, i.e. not very complex or simple.

Besides, n\_estimators, i.e. the number of trees in the forest also has an effect. The default is 10 trees with average AUC=0.78. When we increased the number significantly, more trees can increase the AUC. It improves the performance specially for class=1.

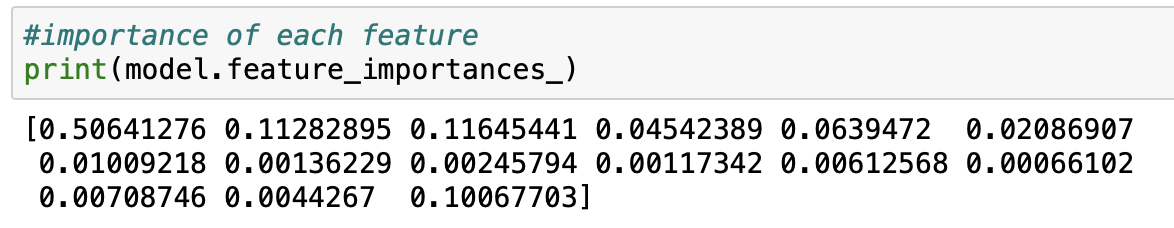
Concerning the dataset, the dataset was trimmed from around 3k features to 12 features, in order to minimize the runtime. Now, we would like to test whether the untrimmed dataset would provide more meaningful information, thus giving a better prediction. Interestingly, the performance is only slightly higher than that of the trimmed dataset. Therefore, if the predictor looks for efficiency, a trimmed data set or simple information would be sufficient for him to do the prediction.



**Conclusion for Random Forest Classification Model**

The model has a better performance in terms of predicting low or high revenue. This may be due to the fact that there are outliers and the dataset skewed to the higher revenue side. Therefore, it may be more difficult in predicting the 2nd quantile, between high and low revenue.

Using the feature\_importances function, we found that ‘budget’ is the most essential feature in this model to predict the revenue of the movie. The higher the value, the more important the feature is. Runtime, release year and production company also have certain essence.

(The values of the array sum up to 1)

As shown, random forest is a better classifier in this case because the dataset has some outliers and is skewed. Random forest is also more stable and do not have the problem of overfitting. Therefore, less step, such as overfitting, are needed to perform.

**(anything else?)**

**Appendix**

Fig. 1 Summary Table of Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Attributes** | **Description** | **Comment** | **Handling** |
| budget | budget of a movie in USD without adjustment for inflation |  | Keep |
| genres | one or more genres of a movie | More than 1 attributes in the same column | One-hot-encoding |
| homepage | link of homepage of a movie | Irrelevant to data mining | Drop |
| id | ID of movie assigned by TMDB | Irrelevant to data mining | Drop |
| keywords | brief description of a movie’s plot in few words | Can be generalised as genres | Drop |
| original\_language | original language of a movie |  | One-hot-encoding |
| original\_title | title of a movie in its original language | Multiple languages are involved | Drop |
| overview | brief description of the plot of a movie in few lines | Can be generalised as genres | Drop |
| popularity | based on user interactions (#views, #votes, favourite, watchlist) on the TMDb website  <https://developers.themoviedb.org/3/getting-started/popularity> |  | Normalization |
| production\_companies | one or more production companies of a movie | More than 1 attributes in the same column | One-hot-encoding |
| production\_countries | one or more production countries of a movie | More than 1 attributes in the same column | One-hot-encoding |
| release\_date | release date of a movie | Exact release date may not be sufficiently useful for data mining | Extract year, month, date, quarter  from release date |
| revenue | revenue of a movie in USD, without adjustment for inflation | Target variable | Drop |
| runtime | duration of a movie in minutes |  | Keep |
| spoken\_languages | available languages of a movie | Can be generalized to original language | Drop |
| status | whether the movie is released/rumored/post production | Almost all movies in the dataset is released\*;  Movies that are not yet released may have inaccurate data | Drop movies if they are not released |
| tagline | promotional text of a movie | Irrelevant to data mining | Drop |
| title | title of a movie in English | Irrelevant to data mining | Drop |
| vote\_average | average score given by users after release of a movie (out of 10) |  | Keep |
| vote\_count | number of vote given by users after release of a movie | Irrelevant to data mining | Drop |

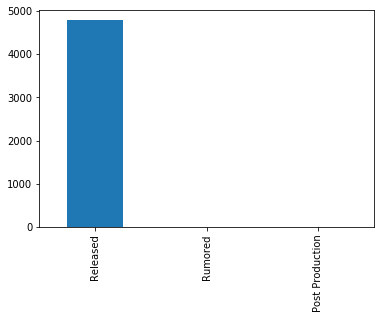
\*Status of movie before modification of dataset  
(almost all movies in the dataset is released)   


Fig. 2 Data Type of Variables

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Data type** | **Python** |
| budget | Numerical, Continuous | Float |
| popularity | Numerical, Continuous | Float |
| revenue | Numerical, Discrete | Float |
| runtime | Numerical, Discrete | Integer |
| vote\_average | Numerical, Discrete | Float |
| popularity\_norm | Numerical, Continuous | Float |
| year | Numerical, Discrete | Integer |
| month | Numerical, Discrete | Integer |
| date | Numerical, Discrete | Integer |
| quarter | Categorical, Ordinal | Integer |
| language (one-hot-encoding) | Categorical, Binary | Integer |
| genre (one-hot-encoding) | Categorical, Binary | Integer |
| production\_companies (one-hot-encoding) | Categorical, Binary | Integer |
| production\_ countries (one-hot-encoding) | Categorical, Binary | Integer |

1. [https://www.kaggle.com/tmdb/tmdb-movie-metadata#tmdb\_5000\_movies.csv](https://www.kaggle.com/tmdb/tmdb-movie-metadata" \l "tmdb_5000_movies.csv) [↑](#footnote-ref-2)
2. We dropped 9 attributes including homepage, id, keywords, original\_title, overview, spoken\_languages, tagline, title, vote\_count and status (reasons are listed in Appendix Fig. 1) [↑](#footnote-ref-3)
3. https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.GridSearchCV.html [↑](#footnote-ref-4)