LELEC2870 - Project 2022

Predicting movie's revenues with machine learning models

Rousseau Mathieu, 67001800

UCLouvain
Belgium
23/12/2022

1 Introduction

Starting with a dataset containing data from *IMDb*[1] as well as *BoxOfficeMojo*, the goal of this project was to predict a movie's revenue in the USA with the help of machine learning models.

2 Exploratory data analysis and preprocessing

The first part of this project was to perform some data analysis to understand better our dataset and then preprocess it in order to enhance prediction results.

Our dataset does not contain that much features at a first glance. However, the principal difficultie was to manage the so-called **embeddings** that consist, for every observation, in large vectors of features (represented as float numbers). With *img_ embeddings* feature being a vector of size 2048 and *text_ embeddings* a vector of size 768. Keeping all these features would give us a very high dimensional dataset with a number of features greater than the number of observations. This situation should be avoided as it is prone to overfitting[2].

The first thing was to remove obvious unnecessary features like *title* which contains only distincts features or *img_ url* and *description* that we judged unnecessary as we already have the embeddings. We also noticed there were duplicated observations and removed them.

We then took a look at the missing values. There were only 4 values missing for the column *genre* consisting in only a tiny portion both datasets (*X1.csv* and *X2.csv*) therefore we simply dropped the observations with missing genres. However, there were respectively 264 and 106 values missing for the column *runtime* in *X1.csv* and *X2.csv*. These counting for more than 5% of our dataset, we did not take the risk of removing the observations containing these to avoid introducing a bias in our datasets. Looking at the dataset, it seemed like these values were missing randomly. We noticed that the shape of the distribution of the *runtime* feature is not too far from a Normal. As a consequence, we decided to impute the missing values with the mean of this feature.

Looking at the distributions of the other variables, we noticed that *n_ votes* and *revenues* were heavily right skewed. Because of that, simply removing outliers outside $1.5 * IQR$[3] could lead to removing a lot of datas. To fix the skewness, we took the log of these variables resulting in a much more homogenous distribution. We also noticed that the variable *is_ mature* only contained a unique value so we deleted this column. Then, we removed the outliers for the *runtime*, *production_ year* and *release_ year* features. Models like the linear regression and k-nearest neighbors are really sensible to outliers. Because the distribution of these features were more or less normally distributed, we removed the observations having a z-score[4] above the range $[-3, 3]$. We only did that for the training set so that the testing set stays representative of the reality.

The *genres* feature consists for every observation in a list of the maximum 3 genres that most accurately represent the given movie. Because there were not that much unique genres, we decided to **one-hot encode** them. However, the *studio* feature suffer of high-cardinality. Indeed, this feature consists in more than 300 unique studios so to avoid exploding the dimension of our dataset (and therefore avoid the curse of dimensionality), we replaced the less representative studios (the ones that represent less than 1% of the dataset) by a special name : "other".

To manage the embeddings, we tried to run a PCA and Kernel PCA (with a radial based function kernel) against them to keep the most significant features (those that explain the most the variance of the data points). The later has the advantage of recognizing non-linear pattern and gave slightly better results for our model. At first, we wanted to keep a certain amount of variance but we noticed

---

1. International Movie Database.
2. Situation where the model learned results of training data instead of generalizing and therefore cannot predict any new data.
3. Interquartile range.
4. Number of standard deviation with respect to the mean of the distribution.

we got better results for our model by being more drastic. We then chose to keep only the five most important principal components for each of the embeddings.

Finally, we standardized our dataset to put every features on the same scale and to have better results with the linear regression and multi-layer perceptron models (MLP). For example, in the MLP, it allows the gradient descent to converge more quickly. We ensured to only compute the transformation on the training set and applying it on the testing set to avoid data leakage. We end up with a dataset of 62 features.

3 Feature selection

Even if we ran a PCA on the embeddings, we still have a lot of features in our dataset and not enough data in order for our models to generalize well. We performed feature selection in order to decrease the number of variables in our dataset. We investigated multiple technique to select features : correlation, mutual information and recursive feature elimination. The problem with the first one is that it is only relevant for linear models since a low correlation does not mean the absence of relationship for non-linear models. We could have selected features highly correlated with the target for the linear regression but we found that even for this model we had slightly better results with mutual information. The last one on the other hand is a wrapper method and therefore use a model to compute a score and remove unnecessary features accordingly. The problem is that it is computationnaly intensive and very dependent on the chosen model.

We first tried to remove redundant features. The idea was to check the features that are highly corre-lated together (*above a certain treshold*) and then keep the one that had the most mutual information with the target. However we didn't find any redundant features in our dataset (at least after the PCA).

3.1 Mutual information

We chose to use the mutual information to select the features. Mutual information has the advantage over correlation to detect nonlinear relationships between variables and therefore is a filter of choice for nonlinear models. Our strategy was to minimize the redundancy between inputs variables and a maximize relevancy between inputs variables and the target variable (*minimum redundancy, maximum relevance*). The idea is that even if 2 features are highly relevant, we shouldn't add both of them to our subset of features if they are highly correlated since it would increase the model complexity and could cause overfitting.

To minimize the redundancy, we computed a normalized mutual information matrix[2] and decided to remove for each group of redundant features, the feature that share the less information with the target. As you can see on the heatmap[2] in the appendix, *release_year* shares a lot information with *production_year*. *img_feature1* shares also a significant amount of information with *text_feature0*. Looking at the mutual information of each feature with the target variable[3], we decided to remove *production_year* and *text_feature0* since they share less information with the target.

Finally, we decided to test a set of different numbers of features to select our model. We tried to keep respectively the 5, 10, 15, 20 and 30 best features that shares the most information with the target.

4 Model selection

We tried severall models : linear regression, K-nearest neightbors (KNN), multi-layer perceptron and a random forest. For each model except the linear regression, we choosed to tweak severall well chosen hyperparameters and used a searching algorithm along with cross-validation (5 folds) to find the

best hyperparameters for each model. A common searching algorithm is **Grid Search** that will test every possible combination of hyperparameters. However, it can be computationaly intensive if the search space is wide which is often the case since we don't want to miss the optimal parameters. Another approach is to use **Random Search**. With this approach, we can also draw hyperparameter configurations from distributions besides discrete sets and the random search will randomly select a combination of hyperparameters among the ones defined instead of doing an exhaustive search. It allows to explore a wide range of hyperparameter configurations with time-efficiency. Looking at the result of the random search, we can identify the area where the results are promising, tighten our ranges and perform a random search again. Although this algorithm being less computationaly intensive, it can be time consuming of running and running again the random search to find the best set of hyperparameters.
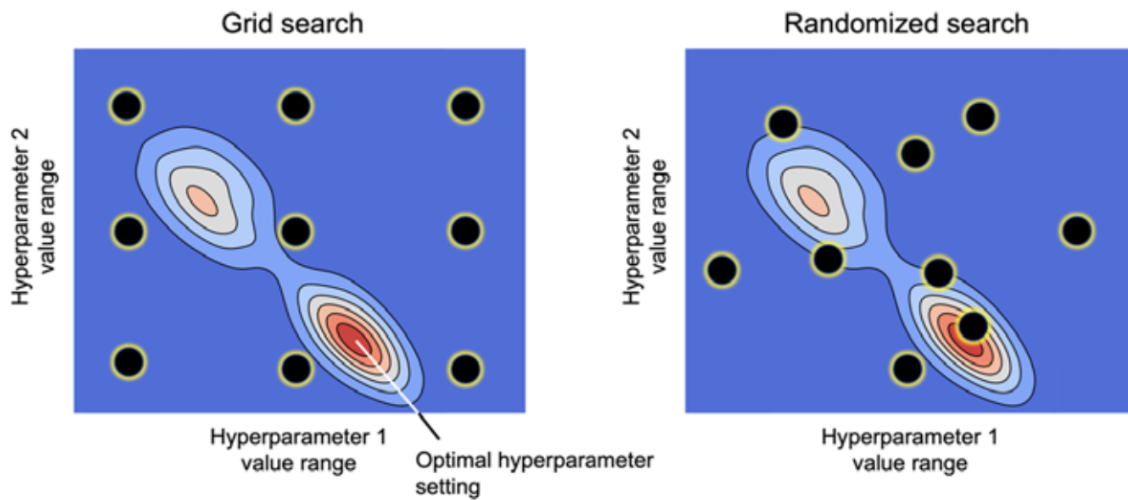

FIGURE 1 − *Comparison of grid search and randomized for sampling 9 different hyperparameter configurations*

A third approach and the one we tried in this project is to perform a **Bayesian Search** which is basically similar to random search but modelise the hyperparameters space probabilisticly to optimize the research of the best set of hyperparameters. For each model, we run a 5-fold cross-validation to avoid overfitting and we compute the validation score as well as the test score that's been computed on a part of the X_1 dataset that has not been used to train any of our models.

4.1 Linear regression

The linear regression has no hyperparameters to tune so we just ran a cross-validation for each set of features we decided to keep. The results show us that ...

#features	scores	
	validation rmse (\$)	test rmse (\$)
5	7.986e07	8.618e07
10	8.874e07	8.905e07
15	8.491e07	8.764e07
20	7.134e07	7.875e07
30	6.980e07	7.701e07

TABLE 1 − *Cross-validation results of linear regression model*

4.2 K-Nearest Neighbors

For the K-Nearest Neighbors model, we need to take care of adjusting the number of neighbors hyperparameter. Indeed, a value too low will induce overfitting and a value too high will induce underfitting. We ran a bayesian search with 12 iterations.

#features	scores	
	validation rmse (\$)	test rmse (\$)
5	6.357e07	7.738e07
10	6.236e07	7.404e07
15	6.172e07	7.455e07
20	6.341e07	7.281e07
30	6.386e07	7.226e07

TABLE 2 − *Cross-validation results of K-Nearest Neighbors model*

4.3 Multi-Layer Perceptron

4.4 Random Forest

A random forest is an ensemble technique that combines multiple decision trees. As a consequence, it has a better generalization performance than a single decision tree due to randomness which decrease the model's variance. Another thing to note is that it is little sensitive to outliers and do not require much hyperparameters tuning.

5 Chosen model and predictions
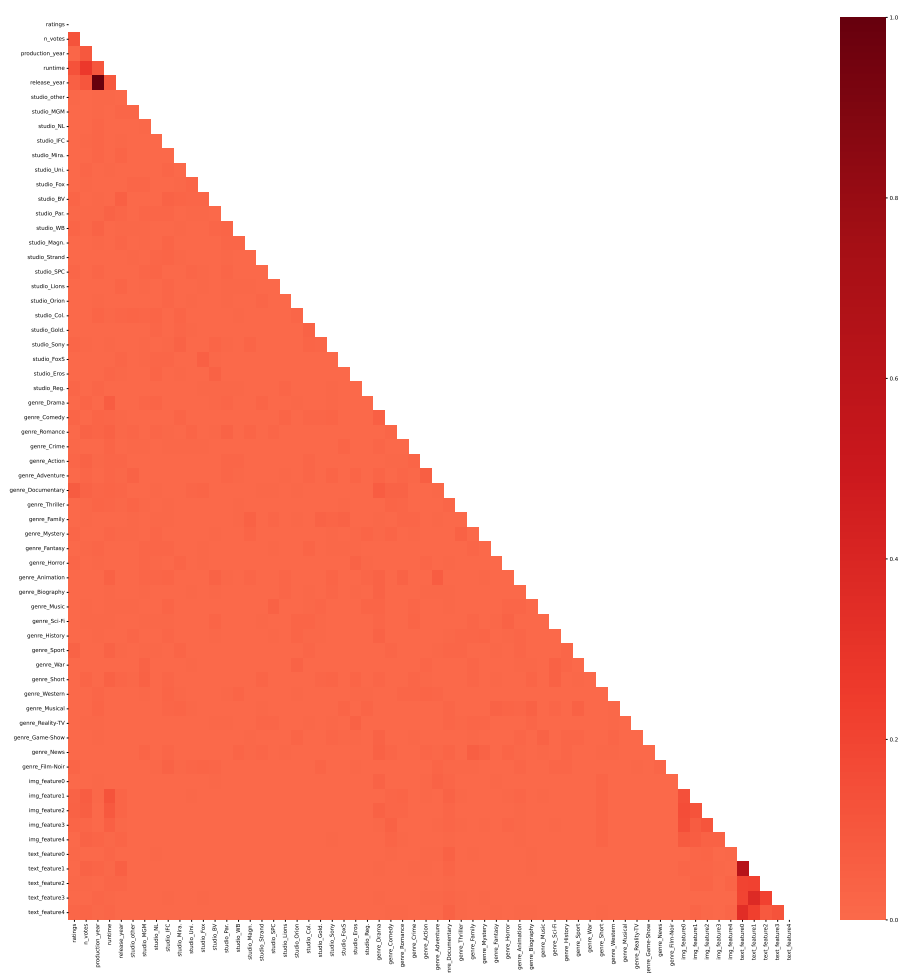
Appendix

Mutual information matrix

FIGURE 2 − *Mutual information matrix*

Mutual information with the target variable