1. Introduction

Machine Learning has numerous applications in our daily life. As it is very useful to make predictions, it will be use in this project to estimate the possible revenue of multiple films depending on their different features.

We aim to obtain the best possible predictions for the revenue minimising as much as possible the error that we had during the process. So, to make it possible, we will explain in this report how we handled the different type of features that we had as well as which models were used to make the predictions.

2. Data Engineering

Data Engineering was one of the most important parts in this project because depending on how we treated the different features that we had, the results varied hugely.

2.1 Data Pre-processing

One of the first problems that we had to solve were non-numerical features. This kind of features can be quite useful to estimate the results that we want to obtain but in order to use them, we should identify which procedure we should follow to use them properly. Among this variables we have interesting features such as studio or genres.

We handle this in two ways:

- We dropped some of them because, based on our criteria, they were not significant for the estimation (title, image_url...).
- We transformed them into numerical features. The genre's column was split in the possible genres of a film while the studio's column was transformed into numeric values.

Moving towards the numerical variables, we had two kind of problems; the embeddings and the missing values.

Text and image embeddings were problematic features so we ended up deciding that the best option was to split them in multiple columns in order to use each value of the array given separately. By this way it will be easier to handle them during our feature selection.

In order to face missing datapoints, as it was clearly represented with the feature runtime, we start thinking that dropping the missing values was a good option for us but we realized that doing that we will lose a lot of data, so we ended up replacing the NaN by the mean of the runtime's column.

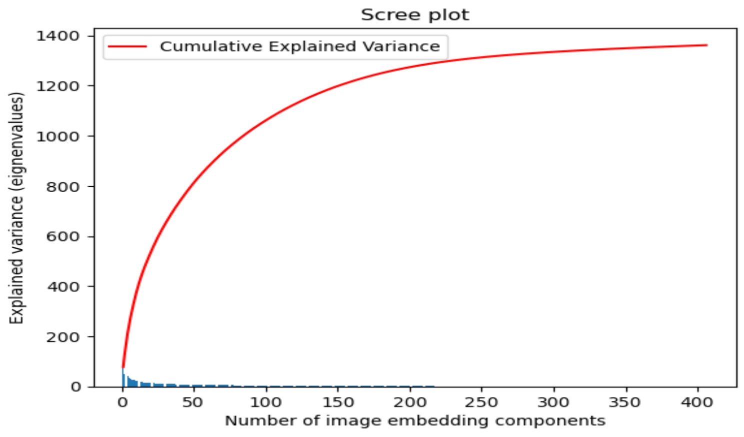
After all this changes we obtained an useful group of features (not the final ones).

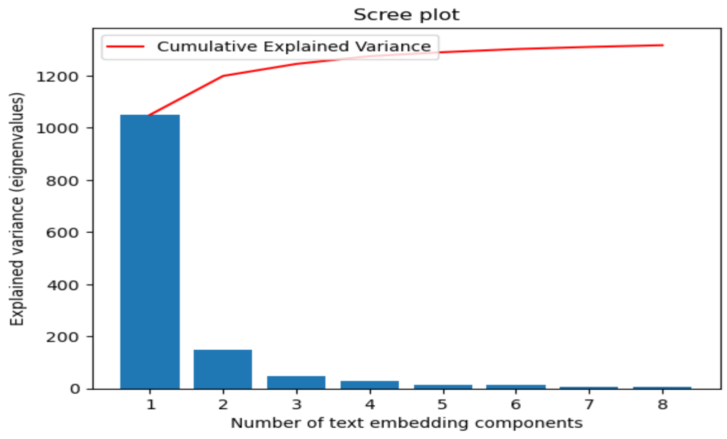
2.2 PCA

The curse of dimensionality could be a problem in the future of our project. What we thought to do was applying PCA to our features, the point was, in which ones.

At first, we applied it in all the features we got from the pre-processing and obviously we obtained worse results. Then, we used this method just in both dataframes of all the embeddings' components. It is remarkable that as PCA is sensitive to scale, we used the RobustScaler in both dataframes. We used as parameter 0.95 what means that the number of components selected have an amount of variance explained greater than the percentage specified (see images below).

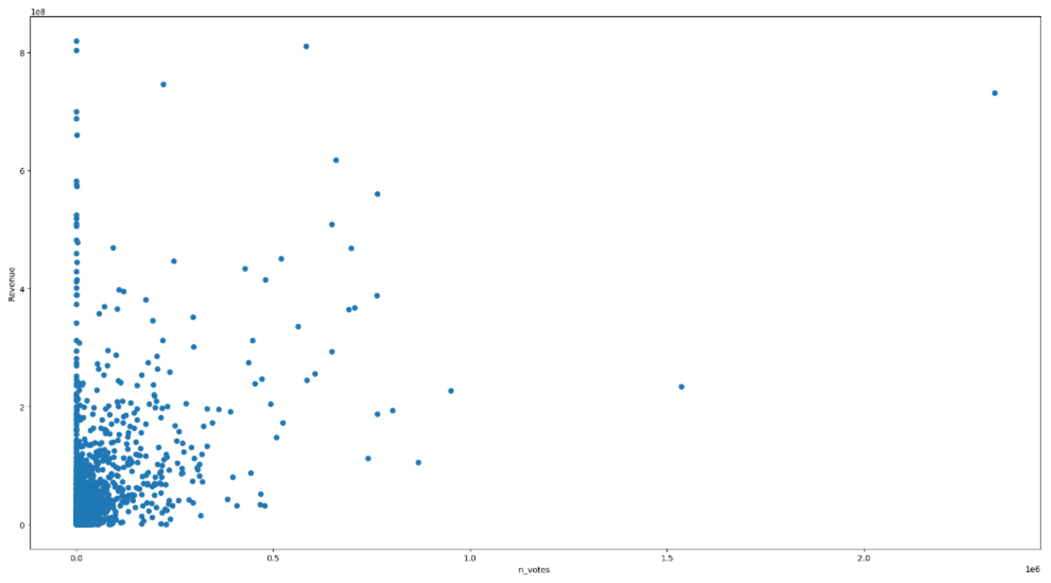
By that we finally reduced the dimensions of the embeddings and concatenated them to our pre-processed dataframe without losing much information.





2.3 **Outliers**

To manage the outliers of out dataset we assumed that one of the most related feature with the target was "n_votes" (or ratings but one come from the other). So, we decided to plot the target and the n_votes for visualizing the possible thresholds to remove the outliers. Finally, we deleted all the films with more than 500.000 votes.



2.4 **Split the dataframe**

In order to reproduce a real situation to implement our models adequately and to get a good estimator of the RMSE of each model we first split our X1 dataframe in two parts, one for training our model and the other one for testing it.

The way we split the dataset was by giving the test part a size of 20% of the whole dataset X1.

2.5 **Standarization**

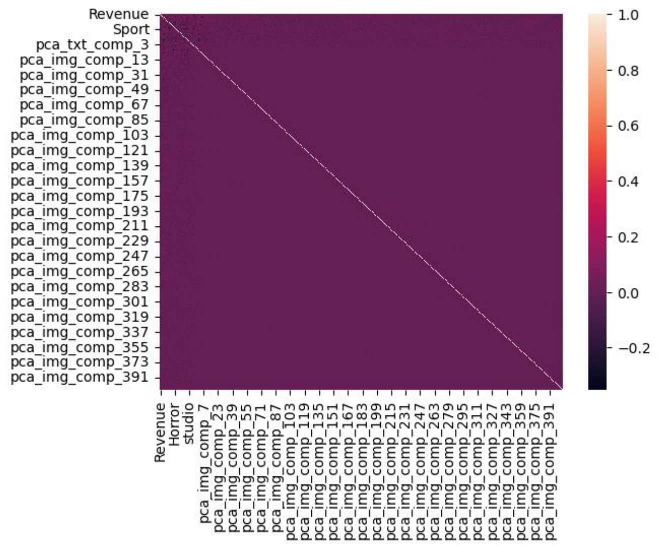
For not having future problems with some models, we standarized and scaled our training dataset and our testing dataset. At first, we used the MinMaxScaler but this led us to worse results as the StandarScaler and finally we found that RobustScaler was less sensitive to outliers so in order to be sure that we use the proper scaled dataset we used his scaler.

Also we used it with the target values of X1 (Y1) cause by that way we could perform an inverse transformation with the predicted revenues to get a right comparison between y_pred (predicted revenues) and y_true (true revenues)

2.6 **Feature Selection**

As the Data Preprocessing, the Feature Selection was a key part in our project; use the big amount of features in our preprocessed dataframe would have led us to problems later in the Model's computing.

As we implemented the Linear Model first, we decided compute the correlation matrix in order to have an idea of how correlated each feature with the target were. In the image below we can not appreciate much information, and because of the big amount of features we could not print the coefficients in the matrix. So, just by proving with used different thresholds in order to leave apart some features.



We knew that this could be one of the greatest feature selection method for the Linear model so we used it. At first we tried with the mean of all correlation's coefficients but it did not work as we just left out around 200 features. Then, we sorted the correlation coefficients in order to see the most correlated features and kept around 10 of them. This worked properly and we kept that threshold.

However, for the rest of the models, which were not linear, we had to use another feature selection method cause the correlation method explained before are not suitable for non-linear models. Then we thought to compute Mutual Information method that can be applied to non-linear relations between features. Once we computed the mutual information's coefficients and sorted them, we keep just in count the first 50 best features (obviously leaving apart “Revenue” feature).

We assumed that Mutual Information could be one of the best feature selection method for all the non-linear models we implemented as it measures general dependence, including linear and non-linear dependences between features.

3. Model’s Implementation

It is important to remark that the implementation of all the models have been similar and it is based generally in the following steps:

- Previous mentioned feature selection
- Hyperparameter (not for linear model) tuning
- Fit of the each regressor
- Prediction of the revenue
- Compute of some metrics (RMSE, explained variance score)

3.1 Linear Model

Once we selected the « relevant feautures » from correlation coefficients method we performed the fit of the regressor with the proper dataframe for one first estimation.

Once our regressor is properly train with the selected data, we predict and rescale the results to see the RMSE and if the regressor was accurate or not. We got results around 55 million of RMSE, although we don’t have a very high variance score, the model still gives uniform values for the RMSE.

3.2 KNN Model

This was the first model we had to implement that used a hyperparameter, so we decided to apply a grid search to obtain a good number of neighbours and then be able to build the best possible model. The param grid used in the search was [1,100] and for different executions we got different number of neighbours, so we estimated the best possible k, being this 14.

In general, the RMSE obtained from this model was around 60M, and in general was one of the best models in the fact that its explained variance score was around 0.22

3.3 MLP Model

For Multi-Layer Perceptron we tried to select a great param grid, for doing a accurate grid search.

As this model take a lot of time to do a greater grid search, we just worked with this param grid and with 10 folds. Trivially, we just selected the relevant features from Mutual Information method as explained before.

During all the proved situations this model gave us an RMSE more less constant or at least situated 5 million above or below from 60M. However, the explained variance score that this model gave us was totally different, it changed a lot between the situations, and this led us to do not trust much in this model as one of the better possibilities.

3.4 Non-Linear model

3.4.1 Random Forest Regressor

Surfing on internet we found this model which combines ensemble methods with the decision tree. It was hold as one of the best models in general, so we tried to implement it. The unique problem we got from this model is the big number of parameters that we could use in the param grid of the grid search, as for MLP. By looking in some examples and asking to some teacher assistant we just did the grid search with the param grid you can see at the file. (Almost 170 combinations)

The results obtained by this regressor were good enough to take in consideration as possible best model between the 5 presented. The RMSE computed was around 55M and with an explained variance of 0.2060 which in comparison with the rest is enough.

3.4.2 Support Vector Regressor

As we were not very sure about the results, we were obtaining with the rest of the models we considered to implement this regressor which main element is the "kernel". You use different kernels and obtain different functions to predict the data.

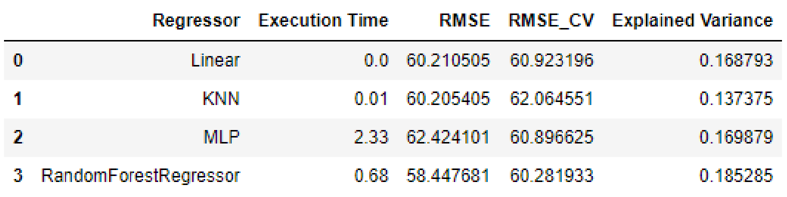
Between the kernels used, were the linear one. This means that SVR with this kernel is linear. Any way we wanted to see the results and in every execution of the file the Radial Basis Function Kernel and the Linear Kernel fought for being the best kernel with errors around 55M and explained variance over 0.20

4. Model Selection

It is important the way we built each model for the final decision. We used grid search to have the best possible hyperparameters for each execution of the project. But the RMSE estimated were just obtained from train dataset and one test data set what means that in different executions the error could vary a lot, and then our estimator would not be much robust.

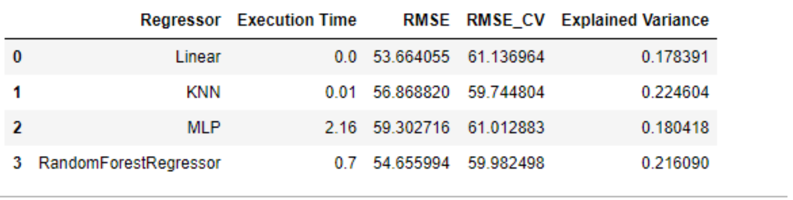
In order to get a robust estimator of each model and select the best possible regressor for the final prediction, we use our best models (taking the best hyperparameters obtained after executing several times the grid search) and complete a dataframe with some score parameters obtained from a cross validation with 10 folds. The scorer used was the explained variance, and after this cross validation we compute also the run time of each model and the mean of the RMSE.

The final dataframe with the results is below:



	Regressor	Execution Time	RMSE	RMSE_CV	Explained Variance
0	Linear	0.0	60.210505	60.923196	0.168793
1	KNN	0.01	60.205405	62.064551	0.137375
2	MLP	2.33	62.424101	60.896625	0.169879
3	RandomForestRegressor	0.68	58.447681	60.281933	0.185285

Execution(1)



	Regressor	Execution Time	RMSE	RMSE_CV	Explained Variance
0	Linear	0.0	53.664055	61.136964	0.178391
1	KNN	0.01	56.868820	59.744804	0.224604
2	MLP	2.16	59.302716	61.012883	0.180418
3	RandomForestRegressor	0.7	54.655994	59.982498	0.216090

Execution(2)

5. Conclusion

As we can see in the results, and looking on different executions we got, the best model between the four is the Random Forest Regressor. Its RMSE is more less constant between executions and is not much high, and also its explained variance is good enough (usually the better one). It could be because the other models are sensitive to outliers and probably, we still having them in some features. Multi-Layer Perceptron surprisingly has given us the worse results, maybe because it needs a high quality on the training. It is true that the Linear model was near to beat the Random Fores Regressor as it has similar results but we think that a linear regressor could not manage well with different trainings sets and maybe it reached good results cause this specific dataset. And in the case of KNN regressor we should remark that it is very sensitive to noisy, high dimensionality and large dataset.

By that we conclude by selecting Random Forest Regressor as the best model for predict the revenue of X2.