UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

MACHINE LEARNING PROJECT

Predict whether a video game will be a blockbuster or not

Niccolò Morabito
Md Jamiur Rahman Rifat

June 2022

Abstract

Predicting the success of an activity such as video games, movies, books of the entertainment industry is very troublesome but also an important aim to achieve. For this project, we tried to predict whether a videogame will be a blockbuster or not based on the review score. Our pipeline is robust enough to deal with data/information leakage as we have preprocessed the train and test dataset separately. Five machine learning models were tested with four experimental setups, tuning HPs for each of them with cross-validation. The best-obtained model has been finally tested on unseen data to get the final evaluation. The negative class has good results since it reaches an f1-score of 0.88. On the other hand, the results in the positive class are worse (the recall cannot exceed 50% because of the high number of negative values), mainly depending on the imbalance between the target classes. In the future, we will exploit some methodologies such as custom weight values and oversampling to improve the performance of our model.

1. Introduction

Video games are a popular medium in the entertainment business that bears a significant economic impact. In our project, we want to predict whether a video game will be a blockbuster or not. To address this, we are using the dataset collected by Dr Joe Cox from 2004 to 2010[1], containing 1212 instances with 36 features. This dataset contains originally 36 attributes, both numerical features (like Sales of Price), and categorical features (like Genres, Publishers, etc.).

The idea is to train a machine learning model able to classify a new video game as a potential blockbuster according to the score that is assigned through reviews.

2. Related Works

Trying to predict whether a product will be a success or not is always something critically important and attractive for companies in general. When it comes to media, several projects already focused on identifying possible blockbusters to invest in. For movies, Grybov et al. tried to train several classifiers from two big movie datasets (The International Movie Database and The Numbers) reaching an accuracy of 65% [1]. Also, another algorithm has been trained from Wikipedia pages to classify movies as flops or blockbusters [2]. Finally, also Google has created an application that predicts box office revenue previous to opening weekend based on the search volume of the movie's trailer [3].

For video games, the effort focused on what are the characteristics that can influence the success of a video game the most, showing that blockbuster video games are more likely to be released by one of the major publishers and that the games with higher quality are significantly more likely to sell a greater number of units than those of lower quality [4]. In this project, we will try to continue this line of research trying to add prediction to the analyses.

[1] https://corgis-edu.github.io//corgis/csv/video_games/

3. Procedures

3.1 Creating separate Train and Test datasets

To test our model independently and free from any biases we have split our dataset into completely different train and test datasets, where the train data contains 909 instances and the test dataset contains 303 data instances. The ratio between the train and test datasets is 75:25. The overall architecture diagram of our solution is illustrated in figure 1. The whole process is scalable and deployable for machine learning services also.

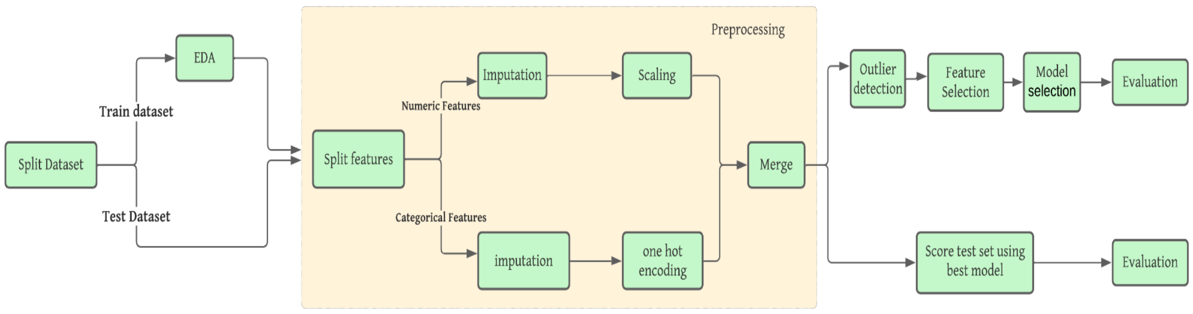

Figure 1: Solution architecture of our pipeline

3.2 Preprocessing

3.2.1 EDA

For the **Exploratory Data Analysis** part, we first looked at some statistics about the features to find evident errors or outliers in the numerical features (we will discuss the categorical and the boolean features later). In particular, looking at the maximum and the mean values for each column we can notice acceptable values that do not show any particular problem (e.g. the `Features.Max Players` column has a maximum value of 8). Most of the columns have 0 as the minimum value, which could also be considered an actual value and an acceptable minimum value. However, most of the columns have 0 for more than 25% of instances, which can make us think that those values should be considered missing and not actual values (in Figure 2, this is evident looking at the plots of columns like `Length.All PlayStyles.Median`, `Length.All PlayStyles.Rushed`, `Length.Main Story.Rushed`, etc.). That is why, in Section 3.22, we imputed these values using the closest neighbours.

Moreover, for analyzing each feature more in detail, it is possible to look at the count plots in the following image.

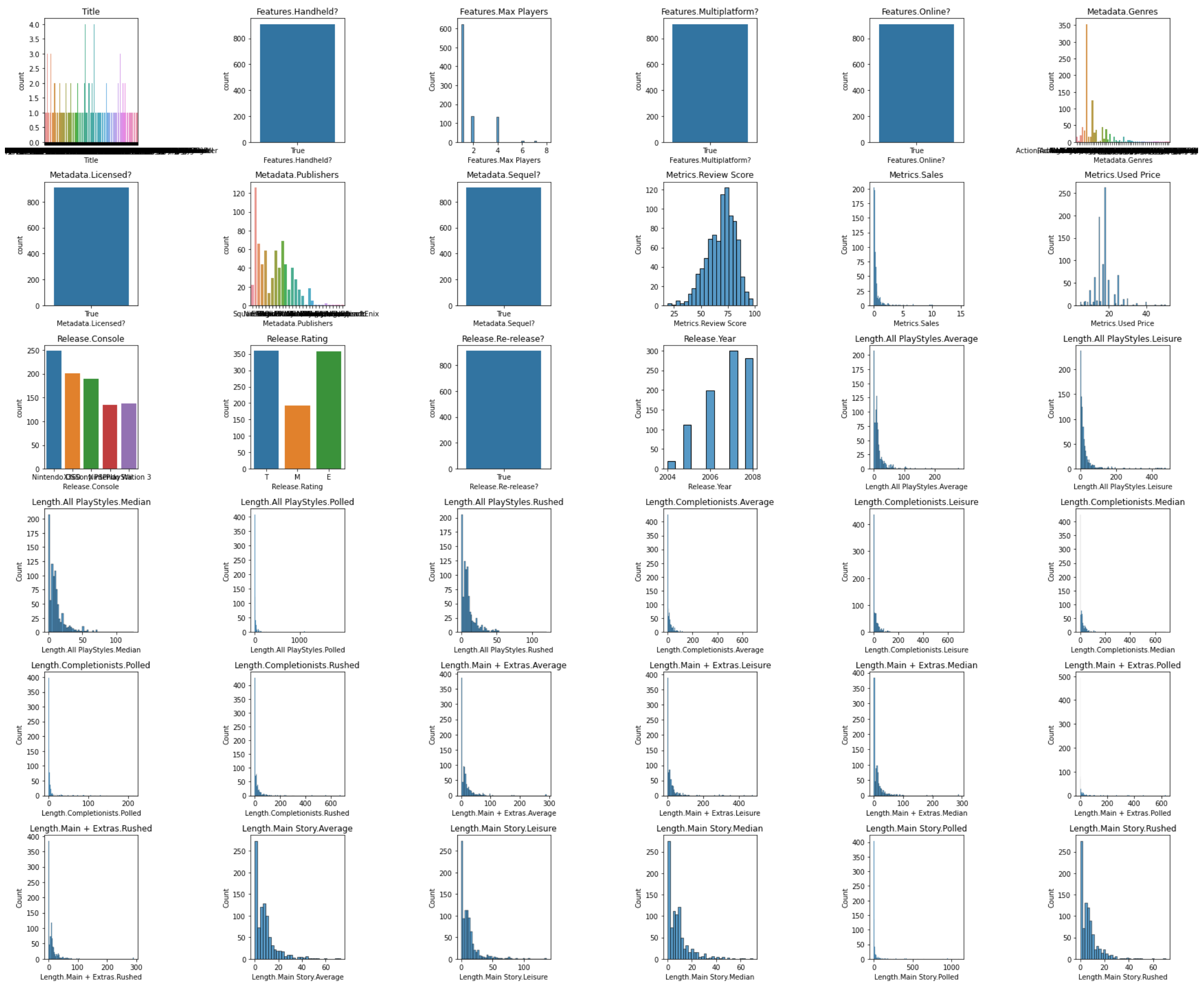

Figure 2. Distribution of values across different columns.

The three features that cannot be easily visualized because of the huge number of possible values are Title, Genres and Publishers. The former column is a unique identifier which does not contain relevant information, therefore it cannot be deleted. For Genres and Publishers, please refer to the section for encoding categorical features, where the two columns are analyzed in greater detail.

Moreover, we can notice that the Year column is not relevant. All the 900 instances are distributed over 5 years (from 2004 to 2008) and the new instances, since they will represent more recent games, are expected to have a different value for that. Other columns that do not contain relevant information are the boolean ones:

- Features.Multiplatform?
- Metadata.Sequel?
- Features.Handheld?
- Metadata.Licensed?
- Features.Online?
- Release.Re-release?

In fact, it is easily visible that their count plots contain only one possible value (True) for all the instances.

Finally, plotting these features can be also useful to study the distribution of the numerical features. Metrics.Review Score, which is the target feature, seems to have almost a normal distribution, and similarly for Metrics.Used Price. All the other numerical features have instead a positively skewed distribution.

After the deletion of the aforementioned features, the remaining features are either numerical (23) or categorical (4), plus the numerical target feature.

3.2.2 Imputation and scaling

Only Metadata.Publishers originally contains null values. Since it is a categorical feature and it is going to be one-hot encoded, the missing values will be managed simply by having 0 in all the deriving columns. The numerical features that contain null values after our preprocessing are imputed using KNN imputer, where the missing value is calculated based on its neighbours.
We also standardized features by removing the mean and scaling to unit variance. The standard score of sample x is calculated as:

$$z = (x - u) / s$$

where u is the mean of the training samples and s is the standard deviation of the training samples.

3.2.3 Visualization and deletion of outliers

Sometimes a dataset can contain extreme values that are outside the range of what is expected and unlike the other data. These are called outliers and often machine learning modelling and model skill, in general, can be improved by understanding and even removing these outlier values. Some notable methods to detect outliers are Inter Quartile Range (IQR), Standard Deviation, and Z score. In our method Z score value is used to detect the outliers and by observing the distribution from Figure 3. we have chosen 10 as the cut off threshold. That means any data points whose z score is more than 10 are regarded as an outlier and removed from our training dataset.

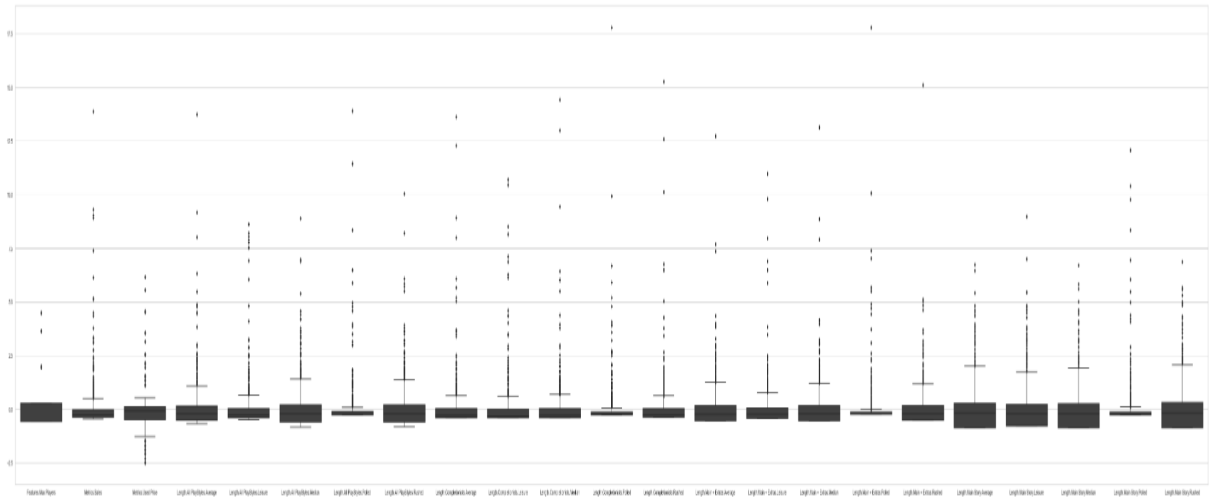

Figure 3. Box plot of each features based on z-score.

3.2.4 Encoding categorical features

As mentioned before, there are 4 categorical features that need to be encoded in numerical attributes. In order to do that, we used **one-hot encoding**. The simplest way to encode a categorical feature would have been **label encoding**: each possible value of the categorical feature is simply converted to a number. In this way, there is also no possibility to have an explosion in the number of features. The problem with this approach is that the numeric values can be misinterpreted by algorithms as having some sort of hierarchy/order in them. This is quite evident in Decision Trees, which would split the data without any sense if the categorical feature is not ordinal. The only categorical feature in our dataset which has an order is the Release.Rating, but as we are going to see the explosion problem lies in the other ones.

Since we are going to substitute the column of the feature f with a set of m columns (where m is the number of possible values for f), it is important to check first the number of possible values that each categorical feature can have because if these numbers are too large, we could have too many features that would lead to an explosion in the training time and space.

- Metadata.Genres has 44 possible values
- Metadata.Publishers has 30 possible values
- Release.Console has 5 possible values
- Release.Rating has 3 possible values

Consequently, we cannot one-hot encode Metadata.Genres and Metadata.Publishers as they are. They both have a big number of possible values but looking at them it is possible to notice that values in both features can be composed by a string of more genres/publishers, separated by a comma (e.g. the instances with Metadata.Publishers='Activision,Konami' can be considered to have two publishers: Activision and Konami).

In particular, there are only 8 genres and 18 publishers and the other possible values are just concatenations of them. Therefore, we encode the Metadata.Genres column to 8 binary features and the Metadata.Publishers in 18 new features. A record with Metadata.Genres = Action,Racing / Driving,Role-Playing (RPG) will have 1 in Metadata.Genre Action, Metadata.Genre Racing / Driving and Metadata.Genre Role-Playing (RPG) columns. Finally, the obtained columns are concatenated to the original dataframe and the original categorical feature is dropped.

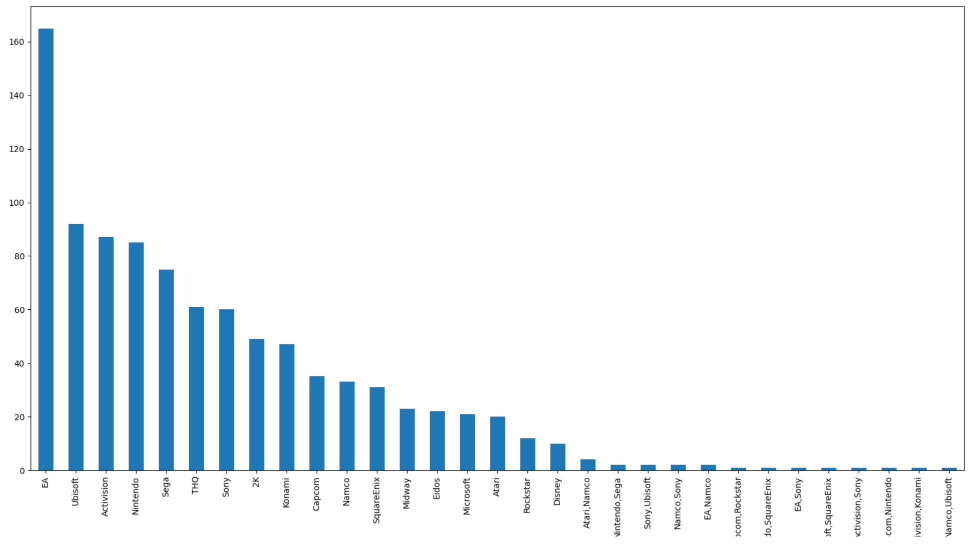

Figure 4. The histogram showing counts of unique values of Metadata.Publishers

As mentioned above, the Metadata.Publishers column contains missing values that obviously will not contribute to creating an additional column, and the records with a missing value will have all the dummy columns equal to 0; in this way, we can prevent imputation on that column that would have resulted in fictitious values.

For the remaining categorical features (Release.Console and Release.Rating) we can proceed with the classic one-hot encoding that will result in 8 new columns (5 for the consoles and 3 for the ratings).

3.2.5 Binning target variable

The name of the target feature is Metrics.Review Score. Though it is a continuous variable we want to bin it into two categories namely “blockbuster” or “not a blockbuster”. The threshold for binning is determined based on statistical intuition and domain knowledge. As the score is between 0 to 100 and from Figure 5 we can see that from the range 80 to 100 only 117 instances are there which is 7.67% of the train data. Therefore, we chose 80 to be the cutoff threshold. we are considering the instances as not blockbusters having a review score less than 80 labelled as “no” and blockbusters having a value equal and greater than 80 labelled as “yes”. The main reason is that the important business question that lies with video games is whether one will be a blockbuster or not. One of the ways to measure it is with its review score. But review score might be superficial or may not be comprehended by some users. With binary classification, we can also distinguish which are the features that contribute most to making a video game blockbuster and vice versa.

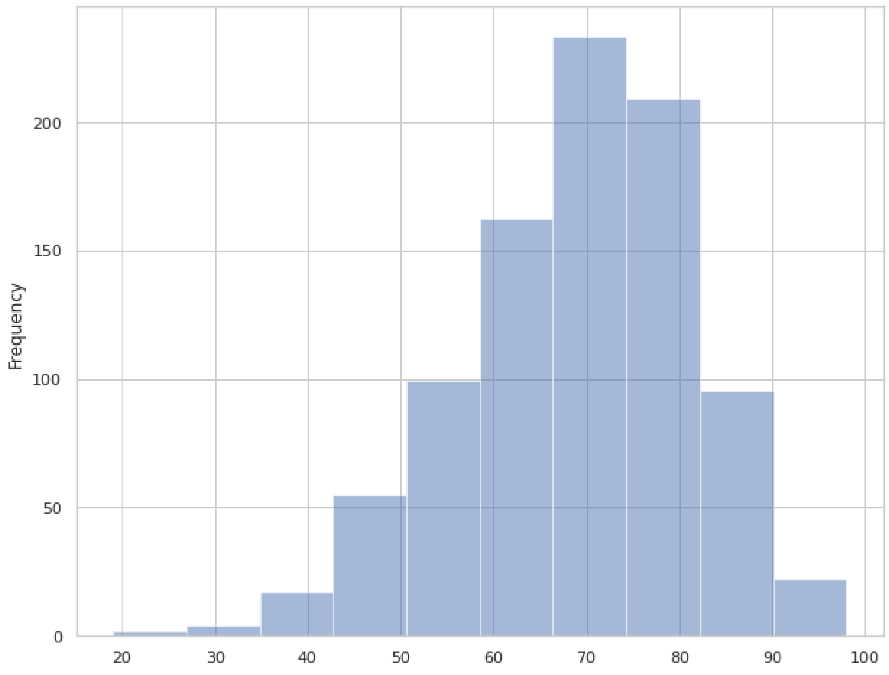

Figure 5. Histogram of the target variable “Metrics.Review Score”

3.2.6 Feature selection

3.2.6.1 Feature Importance (ANOVA)

Analysis of variance (ANOVA) [5] is a statistical technique that is used to check if the means of two or more groups are significantly different from each other. ANOVA checks the impact of one or more factors by comparing the means of different samples. We have used the scikit learn implementation of ANOVA to calculate the feature importance in our dataset. There are other methods to calculate the feature importance, such as the forward/backward selection method, and decision tree feature_importance but those are highly dependent on the machine learning models used in it. The f_classif method returns a score for each variable. In our method, we have conducted two types of experiments where at first we kept only those variables whose score is greater than 1 and in another method those whose score is greater than 6. The comparative result analysis is shown in table 1. The top 5 features found from the ANOVA are presented in Figure 6.

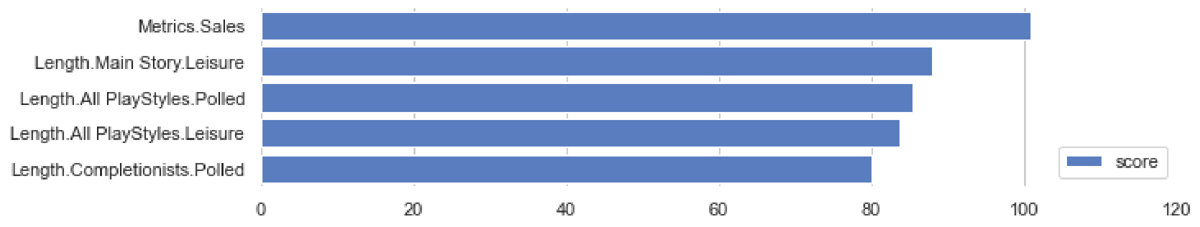

Figure 6. Top 5 Features based on ANOVA.

3.2.6.2 Feature Correlation

We have also run an experiment excluding the correlated features. In this step, the correlation value between each variable is checked in Figure 7, representing the correlation matrix.

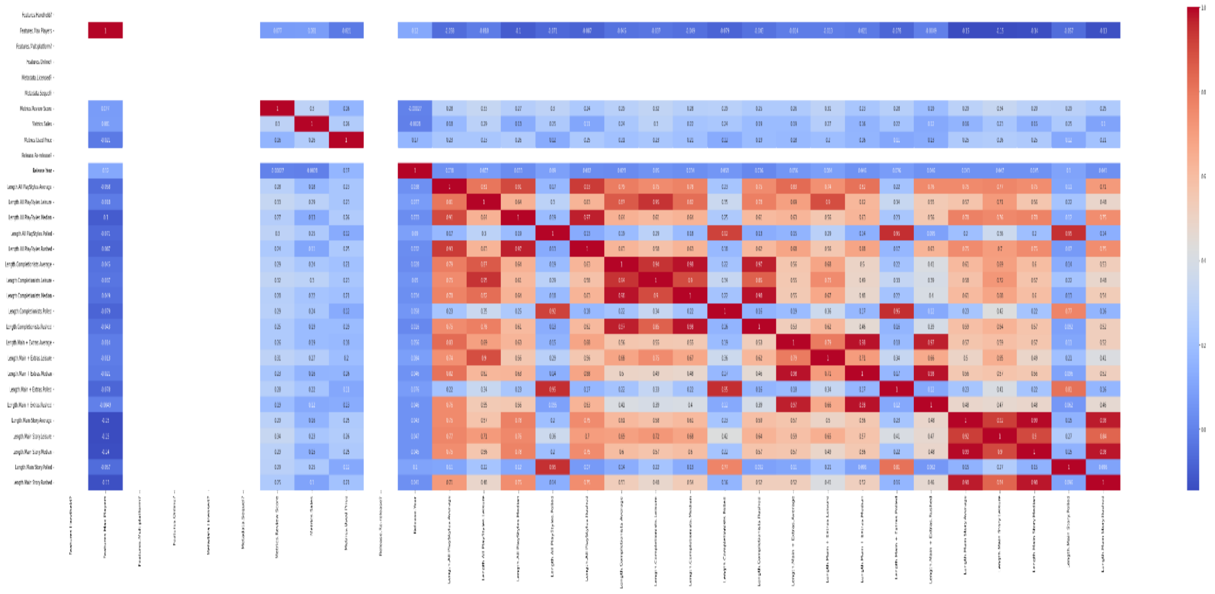

Figure 7. Correlation matrix between each variable

If two variables have a correlation value greater than 95 we consider those two variables as highly correlated and keep one of them only. The list of variables that we removed using the correlation is as follows:

['Length.All PlayStyles.Rushed', 'Length.Completionists.Median', 'Length.Completionists.Rushed', 'Length.Main + Extras.Median', 'Length.Main + Extras.Polled', 'Length.Main + Extras.Rushed', 'Length.Main Story.Median', 'Length.Main Story.Polled', 'Length.Main Story.Rushed']

3.3 Model selection and HP tuning

In order to select the best model, we trained **Decision Tree** (DT) [6], **Random Forest** (RF) [7], **Support Vector Machine** (SVM) [8], **Multi-Layer Perceptron Network** (MLPN) [9] and **XG boost** (XGB) [10] models.

Each algorithm has been run with different hyperparameters to tune the best ones. The parameters and the settings to try have been chosen according to official documentation and the theory studied during the course. The tuning is performed using cross-validation, therefore each model is trained with a part of the training set and the performances are tested with one of the different folds the training set has been split into. In case the tuning time was too long, we used a randomized search not to explore the entire grid search. That is the case of MLPNs, in which parameters are a lot and time expensive.

Other attempts were carried out regarding the feature selection. Different methods to filter out some of the attributes obtained at the end of the preprocessing were tested:

- leaving all the features;
- excluding correlated features;
- excluding non-important features (using two different thresholds).

The results for both the different algorithms and the different feature policies are summarized in Table 1, comparing the results obtained by the best models on the training set.

Model	with all features	excluding correlated features	excluding non-important features (score less than 6)	excluding non-important features (score less than 1)
DT	0.828	0.827	0.822	0.817
RF	0.850	0.851	0.846	0.835
MLPN	0.828	0.848	0.826	0.822
SVM	0.823	0.849	0.835	0.827
XGB	0.857	0.853	0.852	0.832

Table 1. Accuracy score running on the training dataset with different configurations and algorithms

After analysing the above results, it is evident that the XGB algorithm outperformed the others. The Random Forest models tend to have similar accuracy to XGB ones in all the contexts and in the case of excluding features with importance < 1, the best Random Forest model has a slightly better performance than the best XGB one.
Moreover, it is also interesting to notice that the Decision Tree models always underperform Random Forest models.
Finally, we have selected this model as the best model and used it to predict the result in the test set. The best parameter values that we get in our experiment are:

- `colsample_bytree = 0.5` (the fraction of columns to be randomly sampled for each tree is the subsample ratio of columns when constructing each tree);
- `gamma = 1` (the minimum loss reduction required to make a split);
- `learning_rate = 0.1` (A problem with gradient boosted decision trees is that they are quick to learn and overfit training data. To overcome this a very small "learning_rate" is used in our setup);
- `max_depth = 3` (the maximum tree depth for base learners; it is used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample);
- `n_estimators = 100` (the number of rounds used for boosting);
- `reg_lambda = 0`;
- `scale_pos_weight = 1`;
- `subsample = 0.8` (the fraction of observation to be randomly sampled for each tree. Lower values make the algorithm more conservative and prevent overfitting but too small values might lead to under-fitting).

4. Results

The final best model (XGBoost model trained on all the features obtained after the preprocessing, with the parameters specified in the previous section) has been finally tested on the test set to see how it behaves with unseen data. The following figures show the confusion matrix and the values of the main metrics on the test set.

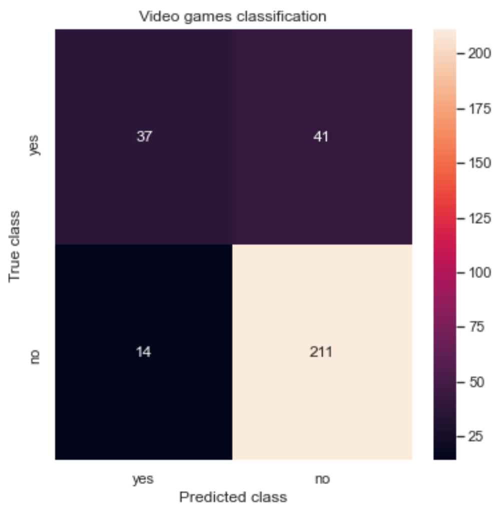

Figure 8. Classification report and the confusion matrix of XG boost classifier on the test set.

	Precision	Recall	F1-score	Support
no	0.84	0.94	0.88	225
yes	0.73	0.47	0.57	78

Table 2. Classification report on the test set

Looking at these results, it is possible to notice that the negative class has good results since reaches an f1-score of 0.88. Even though the results could be improved (especially the precision, which is 84%), we can definitely say that the problem relies on the positive class.

The model performed better for the class which has more data instances in both the training and the test set (in our case the negative class) and could not reach satisfying results for the minor class. That is why we obtained more false negatives and a recall which cannot exceed 50% for the positive class.

Naturally, the simplest explanation is that the number of positive instances in the training set was insufficient and that it made the model tend to overfit the negative instances. The unbalance between classes depends especially on the binning we decided to apply (making the positive class include only the video games with a score greater than 80/100), which actually increased the difficulty of the task reducing the number of instances for the positive class and blurring the edge between the two classes (negative class not only included very bad video games with a score close to 0 but also acceptable or valuable games with a score of 70/100).

ROC Curves summarize the trade-off between the true positive rate and false-positive rate for a predictive model using different probability thresholds. In Figure 9 the orange line representing our best model illustrates that our model is able to classify at different probability thresholds. The trade-off between the false positive and false negative rates is also a bit stable.

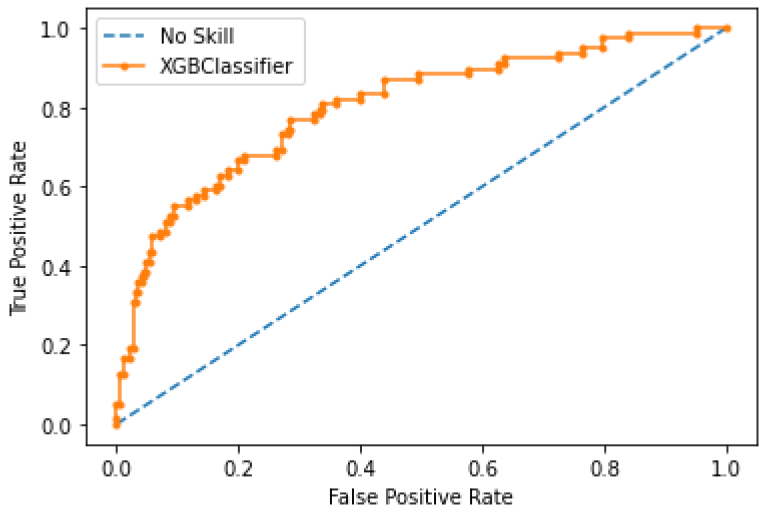

Figure 9. ROC curve on test data using XGB

Precision-Recall curve summarizes the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds. Using the graph in Figure 10, we can control the trade-off between our precision and recall based on our business problem.

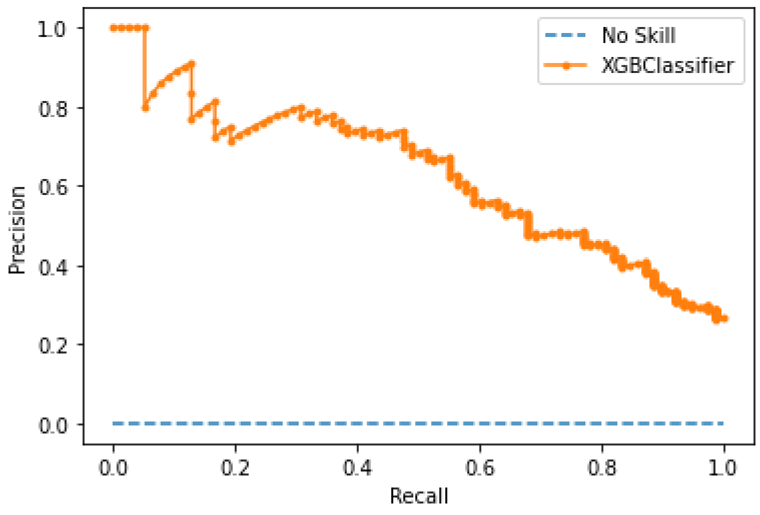

Figure 10. Precision-Recall curve on test data using XGB

ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets. Normally if the percentage of the minor class is less than 1%, we consider the dataset imbalanced. Since in our case the minor class constitutes 25.7% of all data, we used both ROC and Precision-Recall curves to evaluate our model.

5. Conclusion and Future Work

In this study we tried to formulate a regression problem into a classification model in order to better answer the business question related to the problem. Also, we used some non-ad-hoc methods like imputing the categorical variables with the one-hot encoding and took care of the multiple values of a single categorical column by setting 1 to their occurrences in one hot encoding. In all of our experimental settings, the ensemble models were found to be better in respect of others. We also figured out that the class imbalance is also causing the results to be better for the negative class and unsatisfying for the positive one.

Therefore, in the future, we would like to incorporate some methods which take care of the class imbalance problem such as custom weight values and oversampling (under-sampling, with the data at our disposal, is not possible because the number of instances is already limited). Moreover, complex neural network models are found to be useful so we would like to use a complex neural network model with more layers and nodes and other models like CNN, and RNN. The effort should focus on improving the capability of the model to predict the positive class and to reduce the number of false negatives.

References

[1] Lev Grybov et al. (2017) “Predicting a Blockbuster using Open Data Sources”. [Online] Available: https://www.andreasgeorgopoulos.com/wp-content/uploads/2017/10/Blockbuster_Georgopoulos.pdf
[2] M. Mestyan, et al. (2013). “Early Prediction of Movie Box Office Success Based on Wikipedia Activity Big Data”.
[3] K. Acuna, “Google Says It Can Predict Which Films Will Be Huge Box-Office Hits”, Business Insider, 2016. [Online]. Available:
http://www.businessinsider.com/google-study-can-predict-success-of-movies-2013-6
[4] Cox, J. (2014). What makes a blockbuster video game? An empirical analysis of US sales data. *Managerial and decision economics*, *35*(3), 189-198.
[5] St, L., & Wold, S. (1989). Analysis of variance (ANOVA). *Chemometrics and intelligent laboratory systems*, 6(4), 259-272.
[6] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning, 1*(1), 81-106.
[7] Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5-32.
[8] Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., & Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their applications, 13*(4), 18-28.
[9] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256). JMLR Workshop and Conference Proceedings.
[10] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).