HW2 : Final report : Success Movie

Clément Bernard and Maxence Brugeres

January 17, 2020

Introduction

Problem

Definition It is hard for movie industry to know whether a movie will have success or not. Nevertheless, it seems that some companies have found a way to attract spectators. But there are still cases where high budget films are flopping and vice versa.
The first one is to determine what are the parameters that influence the most the popularity of a movie. We want to highlight how the gender, duration, cost or cast of a movie are important for its success.

What is there and the limitations One way to know the success of a movie is the advertises made for it. The more the public sees and hears about the movie, the higher is the chance for them to go to see it. Nevertheless, the success isn't guaranteed. We can't rely only on that parameter.
Then, another way to judge the future success would be critical reviews or social media chatter. It will have an impact on the reputation of the movie and then on the number of viewers. But we can't rely only on these features too. There are too many cases where the critics aren't related to the success. We can see that there are some parameters that come in mind when we are thinking and searching about movie success.
Nonetheless, there are not really a magic tool that will say this specific movie will be appreciated. Our goal is to take into account all the data we can collect in order to give the best prediction for the success of a movie.

Data

We used a dataset from Kaggle. This dataset comes from TMDb which is one of the most popular source for movie contents. It contains around 4800 different movies and 30 features. These features are divided into 2 csv files. One for the general information about the movie (budget, gender, language, title, . . .) and the other one for more details (like the casting).

Pre-processing

All the pre-processing work was explained in the previous report. Here is a reminder of what we get at the end of this work :

```
Entrée [2]:  1 data.columns

Out[2]: Index(['budget', 'keywords', 'revenue', 'cast', 'crew', 'year', 'month_1',
               'month_2', 'month_3', 'month_4', 'month_5', 'month_6', 'month_7',
               'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'Drama',
               'Comedy', 'Thriller', 'Action', 'Adventure', 'Romance', 'Crime',
               'Science Fiction', 'Family', 'Fantasy', 'Horror', 'Mystery',
               'Animation', 'History', 'War', 'Music', 'Documentary', 'Western',
               'genres_others', 'grade', 'ratio', 'oscar_score',
               'United States of America', 'United Kingdom', 'Germany', 'France',
               'production_countries_others', 'language_de', 'language_en',
               'language_es', 'language_fr', 'language_others', 'language_zh',
               'short_time', 'medium_time', 'quite_long_time', 'long_time'],
              dtype='object')
```

Figure 1: Features of our dataset

Modifications made One of our issue was about these three features : keywords, cast and crew. We decided to compute the grade for each time that an element of these categories occured. Then, we computed the average and created a new feature that computes the average of the grade for each element of a given movie.
Nevertheless, something wrong happened. We did it before splitting our dataset into the two subsets for training and testing. In this effect, the grade of the training set had access to the grade of the testing set because it was computed with all the dataset. We can't afford this phenomenon that doesn't fit the reality : we shouldn't have information about the test when we train our models.
In this effect, we modified this part of the code. We computed the average grade of the keywords, crew and cast only for the training set on the first time. It will give the values for the training set.
For the testing set values, as the testing set will be available after the training set, it isn't wrong to consider that we have the training set at disposal when we got the testing set. So, to compute these three features, we computed it with the training set and the testing set (all the data).

Models

Before using different type of models, we thought about the consistency of our data. Indeed, we thought that it doesn't seem relevant to compare together movies that aren't from the same genre.
In this effect, we'll first try with the full dataset, and then split it into genre. We'll only keep the first 8 genres because we don't have enough data to train it with the other genres (the data size is less than 100 of movies for the last occured genres).
Here is the repartition of the genres :

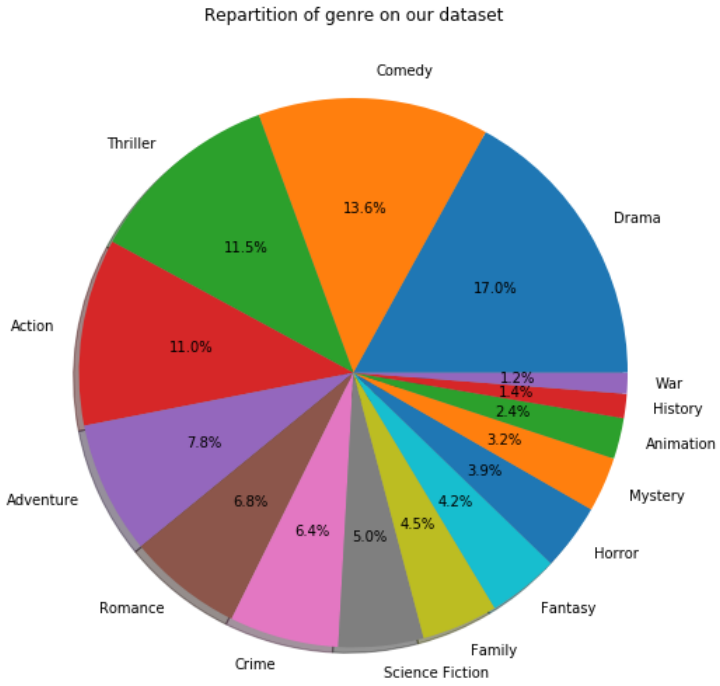


Figure 2: Repartition of the genre on our dataset

Then, we plotted the matrix of correlation available from the panda module. Here is our result :

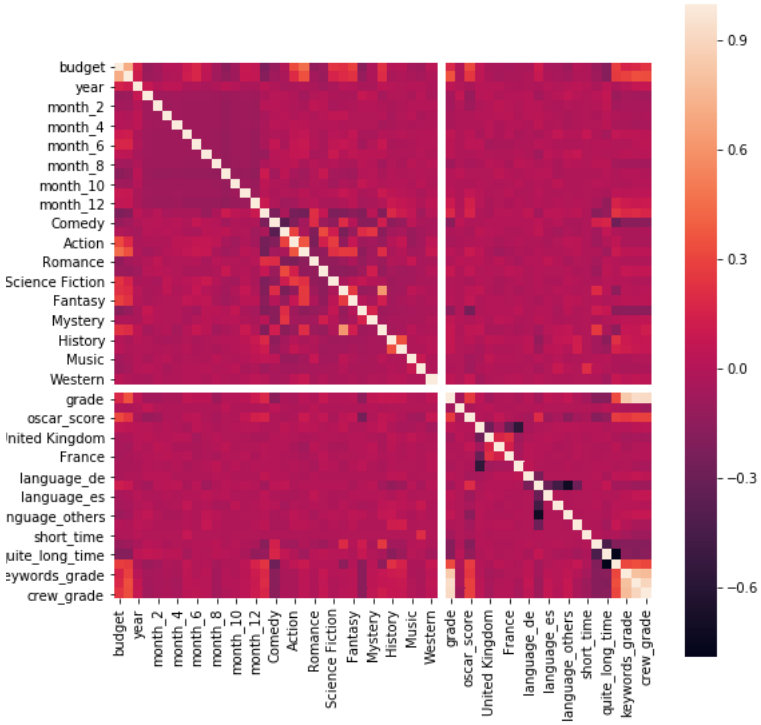


Figure 3: Correlation matrix on our features

What we might be interested are is these three columns :

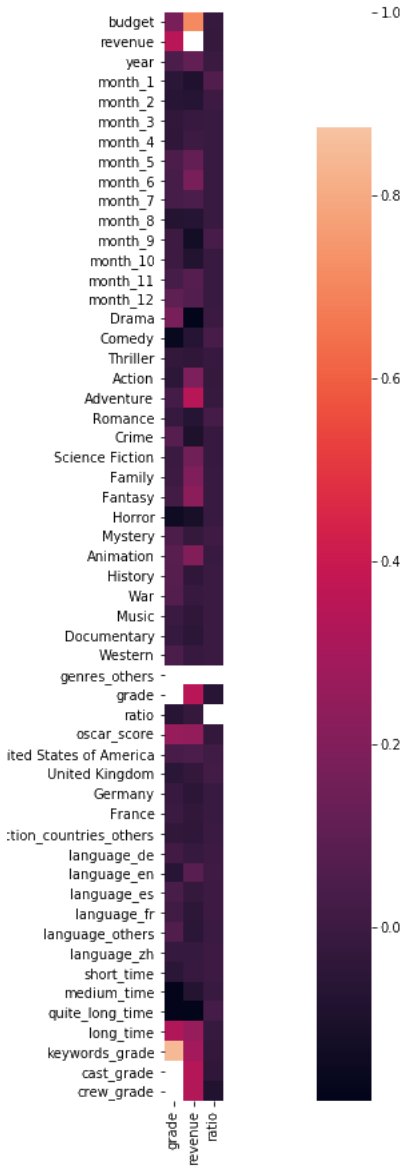


Figure 4: Correlation matrix on our our labels

We can infer these elements :

1. **Grade column** : The revenue, the oscar score, the long time, the keywords grade, cast grade and crew grade are relevant for this label. For the last one, this is logical because these features are computed using the grade.

2. **Revenue column** : The budget, the adventure, the grade, the oscar score, the long time, the keywords grade, crew grade and cast grade are relevant. Therefore we can see that the fact "being an adventure movie or not" is correlated to the revenue.

3. **Ratio column** : None of these features seem to have a lot of impact. We expected a lot from this label, but this graph show the contrary. Let's see how it we'll be with the models.

Solvers

Regression

Label selection Our labels can either be a revenue, a grade or a ratio (defined in the pre-processing). Each of these labels are continuous. That's why we started to use regressors.
Before using any regressor, we normalized our data (with a scikit function that removes the average to each point). It helps the models to converge easily (for instance with the perceptron where, for the regression version, it computes a scalar product with weights and so the values of the input has direct influence on the final output).

Regressors We used 6 regressors : **Linear regression** , **KNeighbors regressor**, **Stochastic gradient descent regressor**, **Multi layer perceptron regressor**, **Decision tree regressor**, **Random forest regressor**, **Support Vector Regression** . We created a function that test few values of the hyperparameters for each regressor. We then plotted or printed the values for each iteration made, in order to modify the final hyperparameters to avoid overfitting.

Score In order to compare each regressor, we needed a metric. We decided to use the determination coefficient, which is defined as :

$$R^2 = 1 - \frac{\sum_i^N y_i - \hat{y}_i}{\sum_i^N y_i - \bar{y}_i}$$

where y_i is the true output, $\hat{y}_i$ is the predicted one and $\bar{y}_i$ is the mean of the real output and N the size of the dataset.
We'll call this coefficient our score. The maximum score that we can obtained is 1. So we know that our goal is to be as close as possible of 1.

K-folds time validation To see which parameters are the best for the regressors, we used a cross validation algorithm. Nevertheless, our dataset depends on the time (we sorted them by year from 1985 to 2016). We can't do the classic K-folds cross validation because we'll train on the future data and then test on the past. Therefore, we used a variation of K-folds validation, named Time Series Split. It will train on the first subsets and iteratively increases the size of the training subset through the time. Then we get the average through all the fold, and this is what we call our score :

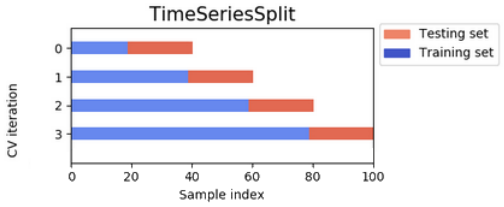


Figure 5: Time Series Split example

Training

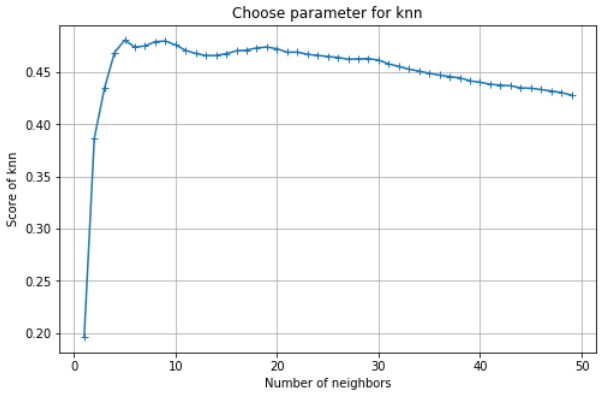
All the data

We used all the data with the K encoding for the genre in the dataset. One important point is that our dataset is larger that any of the genre dataset.

1. **With Revenue as label**
   We tried to get the best hyperparameters for these regressors.

   (a) **Linear Regression**
   We didn't use special hyperparameters.

   (b) **KNeighbors regressor**
   We increased the number of k (which means the number of neighbors we took to interpolate in order to return the output). Then, we get the value that maximizes the score.
   Here is the plot made with the training set :




```
Parameters :  {'n_neighbors': 5, 'algorithm': 'auto'}
```

Figure 6: Selection of hyperparameters for Knn

The algorithm hyperparameter is made to compute the nearest neighbors. We made it as 'auto' because we had no knowledge of the algorithms available. It is a trade-off as this value will make the algorithm to choose by itself which algorithm to choose to compute the nearest neighbors, depending on what was fitted.

(c) **Stochastic gradient descent regressor**
We decided to test either the loss function used and the regularization term (named "penalty" by scikit). The loss can have these values : "squared loss", "huber" (which is a linear loss that is said to be less focus on getting the inconsistent values), "epsilon insensitive" (the classic loss for SVC) and "squared epsilon insensitive" (same as the previous one but becomes squared loss after a certain value of epsilon). The penalty can either be "l2" (the default value), "l1" or "elasticnet" (combination of l1 and l2, which can bring to features selection contrary to l2).
Here is our result on the training set :

```
Score for different parameters :
                            elasticnet          l1          l2
squared_loss                 -3.870035   -3.878309   -3.868577
huber                        -0.469601   -0.469601   -0.469601
epsilon_insensitive          -0.469597   -0.469597   -0.469597
squared_epsilon_insensitive -254.474195 -254.479179 -254.473315
Parameters :  {'shuffle': False, 'penalty': 'l1', 'loss': 'epsilon_insensitive', 'max_iter': 1000, 'random_state':
1}
```

Figure 7: Selection of hyperparameters for SGD regressor

The parameters that maximize the score are, here, the epsilone intensive loss and the l1 penalty.

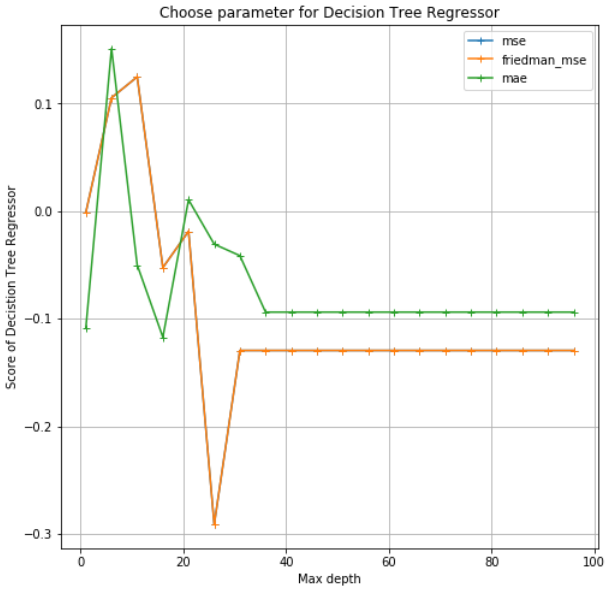
(d) **Multi layer perceptron regressor**
The settings of these hyperparameters are quite hard. There is no way to find easily and efficiently the good sizes for the hidden layers. We tried with few values with a given solver (that optimizes the weight). We fixed it a (50,1) but we know that we can miss here.
Then, for the solver, we had the choices between 'lbfgs' (uses an algorithm of the Newton family) (which is said to be well for small amount of data), 'sgd' which is the stochastic gradient descent and 'adam' (which is said to be well for large data). We don't have more details of the "adam" algorithm, we just know that this is a kind of stochastic gradient descent but optimized. All these methods were found in scikit page.
Here is our result on the training set , with the hidden layer of sizes (50,1) :

```
Score for the following solver :  ['lbfgs', 'sgd', 'adam'] [0.0090590156814839059, -237260300662.7832, -0.46953139
855036774]
Parameters :  {'shuffle': True, 'random_state': 1, 'hidden_layer_sizes': (50, 1), 'learning_rate': 'constant', 'so
lver': 'lbfgs'}
```

Figure 8: Selection of hyperparameters for MLPRegressor

We also found that with a constant learning rate, the results are better.

(e) **Decision tree regressor** :
We tried to optimize two hyperparameters : the criterion of split for each node and the maximum depth of the tree. For the criterion, we can use either "mse" (mean square error), "friedman mse" (which uses Friedman's improvement) and "mae" (mean absolute error).
We looped for each criterion and then looped again on the maximum depth. Here is our results with the training set :

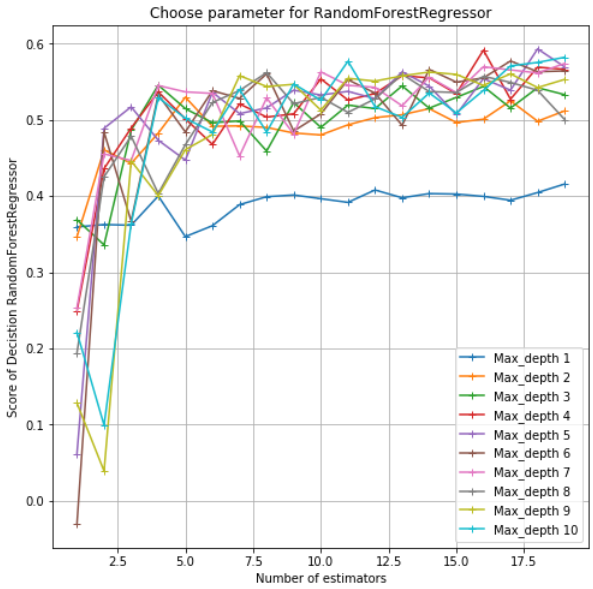



```
Parameters :  {'max_features': 'log2', 'random_state': 1, 'max_depth': 6, 'criterion': 'mae'}
```

Figure 9: Selection of hyperparameters for Decision Tree Regressor

After 40 as maximum depth, the algorithm can't improve the score because it has already treated all the features.

(f) **Random forest regressor**
Here we tried to fix the number of estimators (that is to say the number of trees in the forest) and the maximum depth. We looped on each of these values and plotted the score.
Here is the result with the training set :




```
Parameters :  {'n_estimators': 18, 'max_depth': 5, 'max_features': 'auto'}
```

Figure 10: Selection of hyperparameters for Random Forest Regressor

As we can see, the higher the maximum depth and the number of estimators are, the better is the score. Nevertheless, we don't want to overfit.
Indeed, at first view, we can think that taking the higher number of estimators and the deepest trees would lead to a better score, but it can also have a drawback : overfitting.
So, we prefered to take 10 estimators for a depth of 6.

(g) **Support Vector Regression (linear)**
We didn't add any optional hyperparameters for this regressor.

Selection of regressor Then, we compared our algorithms with the testing dataset. We kept the best parameters we obtained previously. Here is the result with the testing set :

Score for the different regressors All data, revenue

	Score
Linear Regression	0.575710
KNeighborsRegressor	0.497578
Stochastic Gradient Descent Regressor	-0.433752
Multi Layer Perceptron Regressor	0.360252
Descision Tree Regression	0.120021
Random Forest Regressor	0.730410
Support Vector Regressor	-0.116069

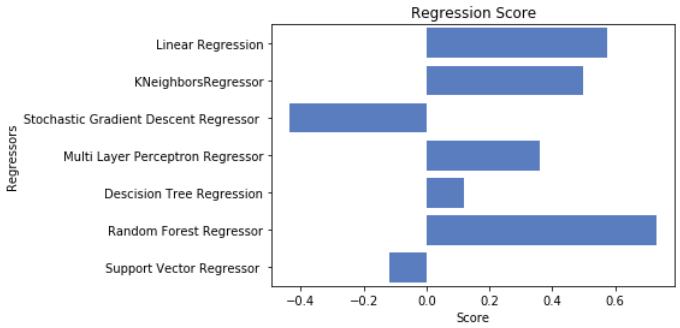


Figure 11: Score on the testing dataset

The random forest regressor works quite well with a score of 0.73. But what does that mean with our data ?
We plotted the predicted values and the real ones. We only plot the 50 first instances of our testing dataset.
Here is the result :

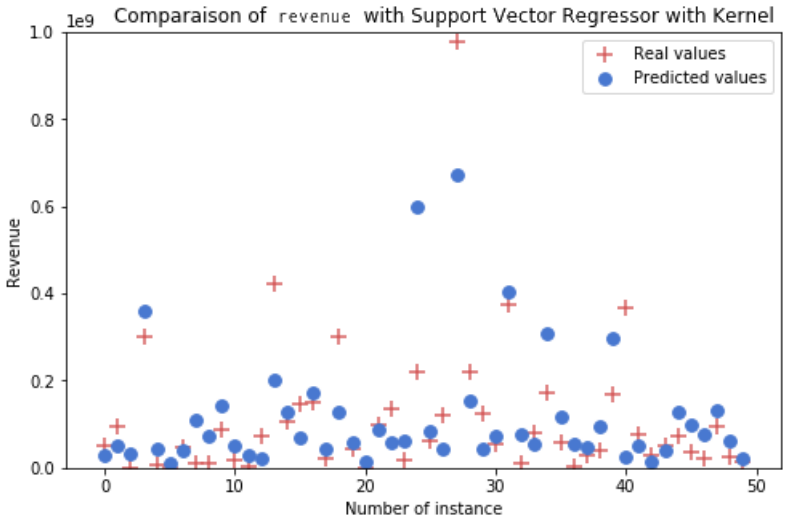


Figure 12: Comparaison of predicted and real values

The result seems to be quite good. There are no outliers except one (around the 25th instance). The score seems to give a good idea of the acceptability of the regression.

2. **With grade as label**

We tried to get the best hyperparameters for these regressors. We did exactly the same methods as previously.
In this effect, we don't print and explain every step we did because it wouldn't bring any more information.
The only part that has changed is the behavior of the Stochastic Gradient Descent. We decided to take the loss and the penalty function that maximize the score, but we encountered an overfitting case.

```
Score for different parameters :
                            elasticnet        l1        l2
squared_loss                  0.888055  0.888682  0.887939
huber                         0.775007  0.782603  0.773459
epsilon_insensitive           0.886539  0.886762  0.885514
squared_epsilon_insensitive   0.885848  0.886641  0.885688
Parameters :  {'penalty': 'l1', 'max_iter': 1000, 'loss': 'squared_loss', 'shuffle': False}
```

```
1 regressor_grade.print_classifier_scores(2)
2
3
```

```
Score cross validation with Stochastic Gradient Descent Regressor  0.888682491081
Score for generalization with Stochastic Gradient Descent Regressor  -20.2219958819
```

Figure 13: Score and parameters for SGD Regressor

As we can see, we got 0.89 as a score after k-fold time validation, but -20 when we tried on the testing set. Here is a typical case of overfitting.
Here is our result when we take l2 as penalty and epsilon intensitive as loss :

```
1 param_sgdr = {
2     'penalty': 'l2', 'max_iter': 1000, 'loss': 'epsilon_insensitive', 'shuffle': False
3 }
4 regressor_grade.set_param(2, param_sgdr)
```

```
1 regressor_grade.print_classifier_scores(2)
2
3
```

```
Score cross validation with Stochastic Gradient Descent Regressor  0.885514369754
Score for generalization with Stochastic Gradient Descent Regressor  0.907502555181
```

Figure 14: Score and parameters for SGD Regressor with l2 and epsilon intensitive

We've evolved from a score of -20 to a 0.90. This is hard to prevent and we'll need to be careful while using these methods of selectivity. Nevertheless, this error doesn't appear in the jupyter notebook because we fixed it. It was a special case that happened once. Now, with the "random state" equals to a fixed value SEED, this error won't happen when we run the jupyter notebook.
Here is the final result for all our regressions :

Score for the different regressors Grade, all data

	Score
Linear Regression	0.893193
KNeighborsRegressor	0.489205
Stochastic Gradient Descent Regressor	0.887757
Multi Layer Perceptron Regressor	0.857514
Descision Tree Regression	0.615341
Random Forest Regressor	0.892270
Support Vector Regressor	0.892856

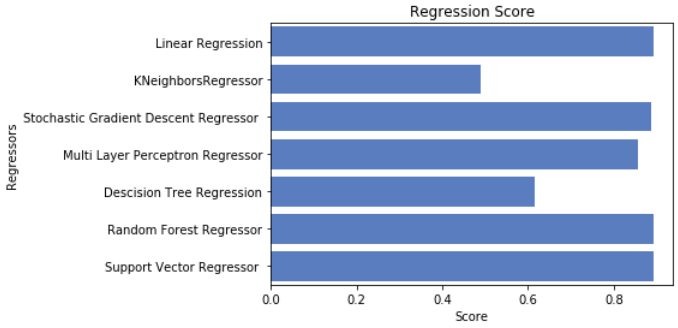


Figure 15: Score on testing set with grade as output

Then, we plotted it with our best regression algorithm, that is to say with either the linear regression or the support vector regressor. We've chosen the linear regression :

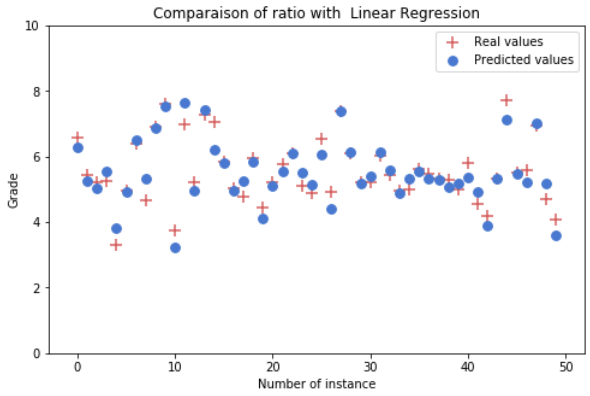


Figure 16: Real and predicted grade with Linear Regression

The result looks great too (no outliers). Our data seems to be linearly separable with the grade as an output (because the linear regression and the linear SVC have both the highest score).

3. **With ratio as label**

We tried to get the best hyperparameters for these regressors. We did exactly the same methods as previously.
All the algorithms got a bad score. The features don't allow to predict well the ratio label. This is what we've seen with the correlation matrix.
We know that now, for the rest, we won't try to guess with ratio as a label.

Score for the different regressors

	Score	All data, ratio
Linear Regression	-0.074051	
KNeighborsRegressor	-0.107209	
Stochastic Gradient Descent Regressor	-0.002969	
Multi Layer Perceptron Regressor	-0.000218	
Descision Tree Regression	-0.002970	
Random Forest Regressor	-0.005542	
Support Vector Regressor	-0.002969	

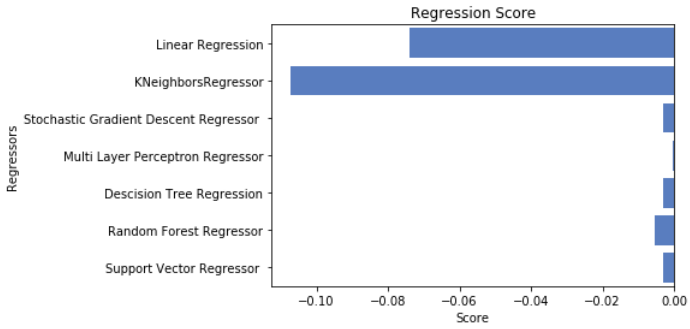


Figure 17: Score on testing set with ratio as output

We see it also when we plot it with both the real outputs and predicted ones on the test set :

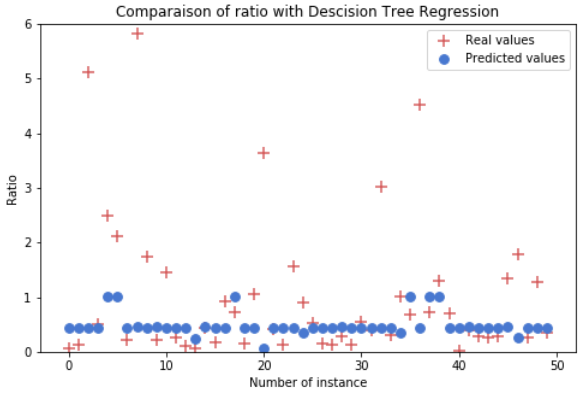


Figure 18: Real and predicted grade with Decision Tree Regression with ratio as output

By genre

We decided to train a model for each of our data splitted by genre. We took 8 different genres, the most frequent ones in our dataset, that is to say : **Drama** ,**Comedy** , **Thriller**, **Action**, **Adventure**, **Romance**, **Crime** and **Science Fiction**. We did the same methods to get the hyperparameters for our algorithms, and we only kept the best regressor with the best score on the testing set (generalization).

1. **Grade as label**
   Here are the results that we obtained for each genre set :
   We can see that the score is around 0.8, which is quite high. Thus it may suggest that depending on the genre of the film, the models are different. This is also quite close to the best score with grade as a label when we have all our data. Splitting by genre doesn't seem to be a bad idea regarding the results.

2. **Revenue as label**
   Here are the results we obtained for each genre :

	Regressor used	Score
Drama	Random Forest Regressor	0.871455
Comedy	Linear Regression	0.801257
Thriller	Random Forest Regressor	0.874434
Action	Random Forest Regressor	0.897594
Adventure	Random Forest Regressor	0.888760
Romance	Stochastic Gradient Descent Regressor	0.859640
Crime	Stochastic Gradient Descent Regressor	0.809026
Science Fiction	Random Forest Regressor	0.868130

Figure 19: Score for each genre set with grade as label

	Regressor used	Score
Drama	Random Forest Regressor	0.517086
Comedy	Linear Regression	0.530226
Thriller	Random Forest Regressor	0.526235
Action	Random Forest Regressor	0.648901
Adventure	Random Forest Regressor	0.529079
Romance	Stochastic Gradient Descent Regressor	0.406777
Crime	Random Forest Regressor	0.511665
Science Fiction	Stochastic Gradient Descent Regressor	0.597552

Figure 20: Score for each genre set with revenue as label

The best we get for all the data with revenue as a label was 0.73 with random forest regressor. Here, the score is around 0.5. We can conclude that with revenue as a label, the splitting by genre isn't relevant.

3. **Ratio as label**
   Here are the results :

	Regressor used	Score
Drama	Descision Tree Regression	-0.000595
Comedy	Multi Layer Perceptron Regressor	-0.004933
Thriller	Support Vector Regressor	-0.013517
Action	Random Forest Regressor	-0.010150
Adventure	Multi Layer Perceptron Regressor	-0.005613
Romance	Linear Regression	-0.010491
Crime	Stochastic Gradient Descent Regressor	-0.023768
Science Fiction	Random Forest Regressor	-0.011839

Figure 21: Score for each genre set with ratio as label

We can see that the score is very low. In this effect, none of the method worked to predict the ratio.

Classification

Why classification ? Our two best outputs that we're trying to predict are continuous. In this effect, with the regression algorithms, we get a continuous output.
But when we talk about a movie, is there really important to have the exact value of a grade or a revenue ? Why don't we try to categorize our outputs ?
Knowing that a movie as a grade of 6.566767 doesn't bring more information than knowing that it has a grade between 6.5 and 7.
That's why we'll discretize our outputs and try to solve our problem with classifiers.

Classifiers We decided to use 8 classifiers : **LDA (Linear Discriminant Analysis)** , **Logistic Regression**, **Perceptron**, **Multi Layer Perceptron Classifier**, **Support Vector Machine Classifier**, **Decision Trees**, **Random Forest Classifier** and **AdaBoost Classifier**.

Accuracy We'll use here for judging the efficiency the accuracy, which is defined as the sum of the instance well categorized divided by the number of instances.

All the data

We first tried with all the data as input. We don't split here by genre.

1. **With grade as an output (20 classes)**
   We started to split our grade output with 20 classes : if the grade is between 0 and 0.5, then it will be the class 1, etc.
   Let's see how we manage to get the hyperparameters of each of these classifiers.

   (a) **LDA**
   We tried to test whether the "svd" (single value decomposition, that can deal with high number of features) or "lsqr" (least square solution) as a solver is the best.

   (b) **Logistic Regression** We set the multinomial option to true. We tried to use few solvers and to keep the best one. We chose between "newton-cg" (which deals with multinomial loss) , "sag","saga" (these two should have features with the same scale to converge) and "lbfgs". Then, we looped for each solver and kept the best one.

   (c) **Perceptron**
   We tested the regularization term ("penalty"), between "l1", "l2" and "elasticnet" (as described with the regression). We looped for each value of the penalty and kept the one that maximized the accuracy.

   (d) **Multi Layer Perceptron Classifier**
   We didn't looped on any hyperparameters. We tried with a fixed values for solver and activation parameters, and reiterate by changing the hidden layer sizes.

   (e) **Support Vector Machine Classifier** We tried to use 3 types of kernels : the polynomial, the RDF, and the sigmoid. We let the scikit module deals the value of gamma. We looped for each kernel and took the one that maximized the accuracy.

   (f) **Decision Trees**
   We looped through the maximum depth.

   (g) **Random Forest Classifier**
   Just like the regression version, we looped through the number of estimators and the maximum depth. We kept in mind that it can overfit easily, and we tried not to keep the parameters that maximized the accuracy, but parameters that had a high accuracy without having too high parameters.

   (h) **Adaboost Classifier**
   We looped on the number of simple classifiers used (i.e the number of estimators).

Here is the result we get for the classification with 20-classes of our movies with grade as an output.

```
Score for the different classifiers        All data, grade
                            Score
LDA                      0.527629
LogisticRegression       0.534759
Perceptron               0.192513
MLPClassifier            0.292335
SVC                      0.493761
Decision Trees           0.393939
RandomForestClassifier   0.522282
AdaBoostClassifier       0.324421
```

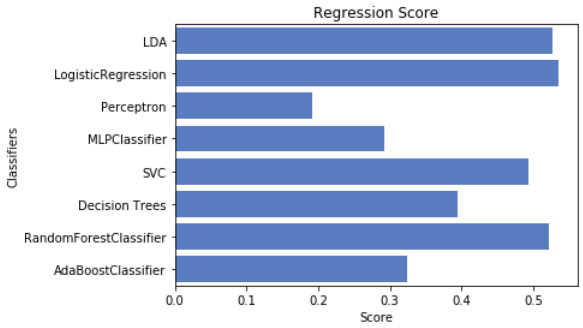


Figure 22: Accuracy on testing set with grade as 20 discretized outputs

Here is the visualization with Random Forest Classifier :

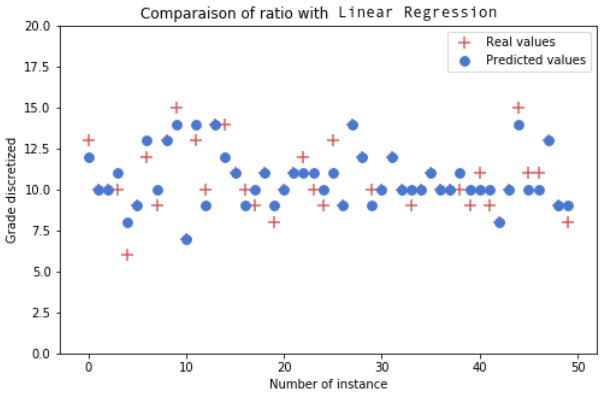


Figure 23: Real and predicted ratio with Random Forest Classifier (20 classes)

We can see that it remains some little outliers. The choice of 20 classes should have been a little too much. It has some difficulties to return sufficient and accurate values. Maybe this problem can be avoided if we split our data by genre.

2. **With grade as an output (100 classes)**
We tried first of all with 100 outputs. Nevertheless, it didn't work well, the accuracy was below 0.5. That was our initial idea.

Score for the different classifiers All data, revenue

	Score
LDA	0.124777
LogisticRegression	0.098039
Perceptron	0.051693
MLPClassifier	0.065954
SVC	0.074866
Decision Trees	0.083779
RandomForestClassifier	0.130125
AdaBoostClassifier	0.081996

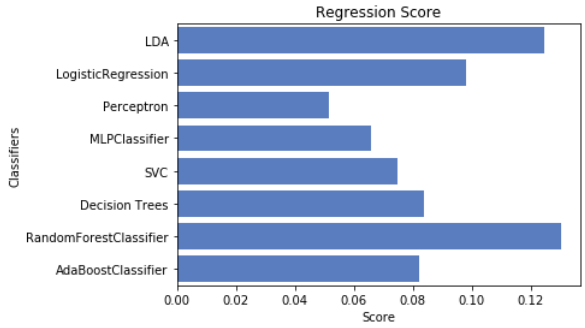


Figure 24: Accuracy on testing set with grade as 100 discretized output

Here is the vizualisation :

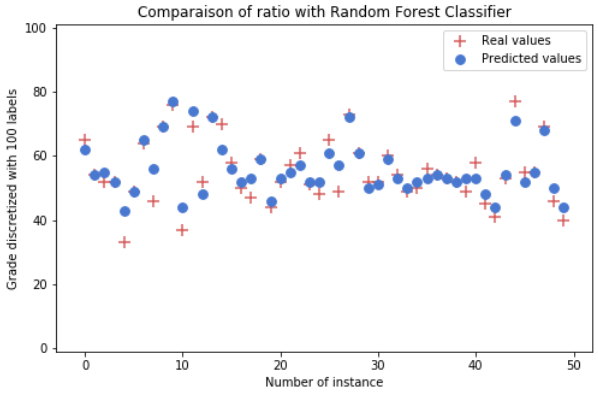


Figure 25: Real and predicted ratio with Random Forest Classifier (100 classes)

The score is low but when we plot as part of a training set it doesn't seem to have a lot of outliers. Nevertheless, our criteria remains the score.

3. **With revenue as an output (3 classes)**
We decided to split our revenue label into 3 classes that representing : low revenue, middle revenue and high revenue. To split it, we took the first quartile and the third one to restrict our classes. Here are the results we get :

Score for the different classifiers All data, revenue (3classes)

	Score
LDA	0.684492
LogisticRegression	0.707665
Perceptron	0.620321
MLPClassifier	0.675579
SVC	0.705882
Decision Trees	0.613191
RandomForestClassifier	0.677362
AdaBoostClassifier	0.691622

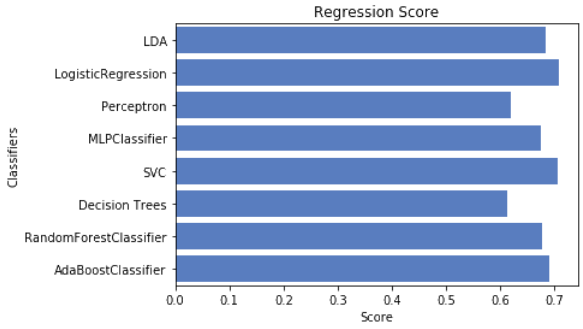


Figure 26: Accuracy on testing set with all data with revenue as 3 discretized output

By genre

As the previous part, we decided to discretize our data and to split our set by genre. We did as the previous point (in the classification with all the data) to get the hyperparameters of the algorithms.

1. **With grade as an output (20 classes)**
   Here are the results we get :

	Accuracy	Classifier used
Drama	0.517928	LDA
Comedy	0.862944	LogisticRegression
Thriller	0.848837	LogisticRegression
Action	0.855422	LogisticRegression
Adventure	0.865546	MLPClassifier
Romance	0.864583	MLPClassifier
Crime	0.865979	LogisticRegression
Science Fiction	0.853333	MLPClassifier

Figure 27: Accuracy for each genre set with grade as a discretized label (20 classes)

   We can see that for each genre the accuracy is very high (except for the Drama one) compared to the accuracy for all the data (around 0.5). We can see that the accuracy is around 0.8, which is quite high. Thus it may suggest that depending on the genre of the film, the models are different.

2. **With grade as an output (100 classes)**
   As the discretization on 20 classes worked well, what if we try to ge deeper ? Here are the results we get :

	Accuracy	Classifier used
Drama	0.143426	RandomForestClassifier
Comedy	0.065990	RandomForestClassifier
Thriller	0.069767	RandomForestClassifier
Action	0.066265	LogisticRegression
Adventure	0.050420	MLPClassifier
Romance	0.062500	MLPClassifier
Crime	0.072165	SVC
Science Fiction	0.040000	MLPClassifier

Figure 28: Accuracy for each genre set with grade as a discretized label (100 classes)

The accuracy is here very low. Our number of labels should be too high to predict correctly. Nevertheless, we only look to the accuracy. Our model can, in reality, may be not too bad if it predicts the class 62 whereas the real class is 63. As we look to the accuracy, we consider it as not good enough.

3. **With revenue as an output (3 classes)**
We saw that except for the Drama genre, the others best genres have outputs only equal to 1. In this case, we can't proceed to a classification. Our choice of discretization of the revenue may have been wrong. We have only been able to print for the drama genre :

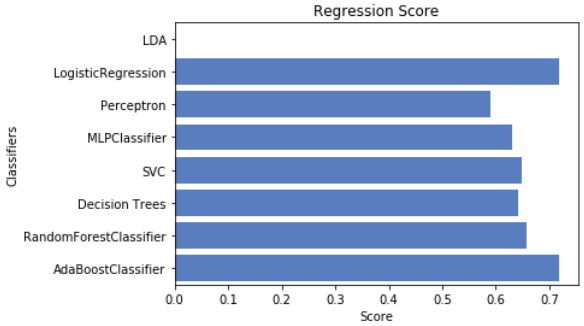


Figure 29: Score for the different algorithms of drama movies

It seems to be as well as for all the data. Only the LDA classifier is bad for this model.

Conclusion

As a conclusion, the main goal of this project was to predict whether a film will be liked by the audience or not. We have characterized this with the grade. Thus considering the best predictor we have found (linear regressor but the other models state the same result), we are now able to guide a film producer toward the success !

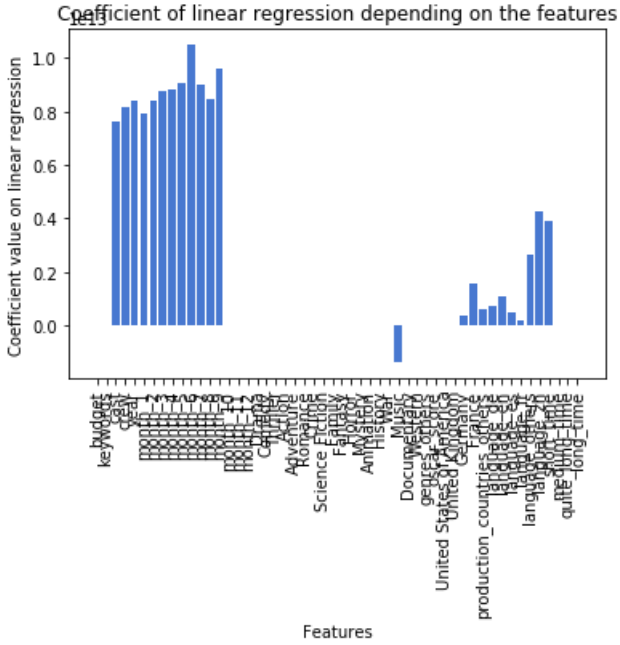


Figure 30: Values of the parameters of the linear regression for all the data with grade as an input

Here are the values that are the most important :

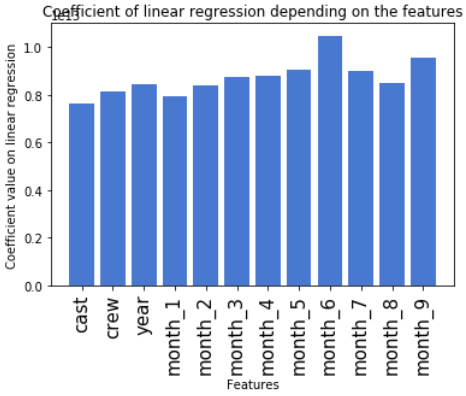


Figure 31: Highest values of the parameters of the linear regression for all the data with grade as an input

The chart below tells us two things :

1. First, concerning the weights related to the months (1-of-k encoded columns), we can see that all the weights are close except for June which is quite higher. If all the weights were the same for the months, it would have only added a bias in the computation of the grade. Still, the June weight is higher than the others so it makes a difference. Thus, dear Producer, if you want your film to be liked by the audience, release them in June.

2. As we can see, the weight of cast and crew score are quite high while the weight associated with oscar score is not. Thus, if you are a producer you should prefer trendy actors who has played in popular films than actors who have played in posh films only recognized by the oscar academy.

As a final observation, we can say that we may should have dropped the year in our preprocessing since it doesn't represent a sensible parameter correlated to the success of a film. It only adds kind of an bias. Our first thought about dividing the set into genre subsets was quite revelant. It worked well when we discretized our data into 20 labels. Keeping a linear regression on all the data is also an alternative to predict the success of a movie. The popularity of the actors, the crew and the sun of June are the key features for a movie to be liked by the public.