
Institute of Statistics, Biostatistics and Actuarial Sciences

# LELEC2870: Machine learning

IMDb dataset: Predicting a film's gross revenue

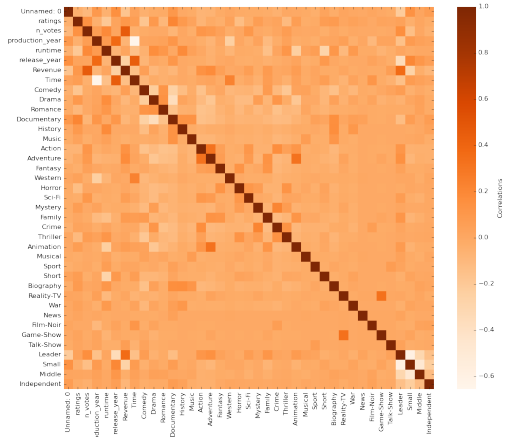
*Author:*
WARNAUTS Aymeric (DATS2M- 87031800)
LALIEU Justin (DATI2M- 74961800)

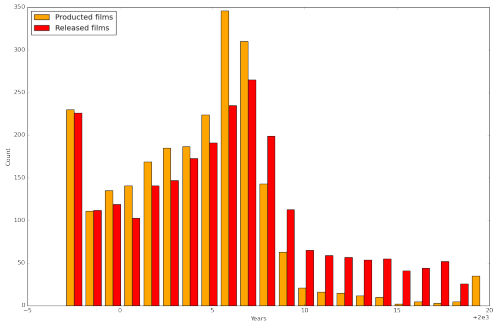
*Professor:*
VERLEYSEN Michel
LEE John

2022-2023

# 1 Introduction

As machine learning methods can be used efficiently for classification and regression tasks, we will start our travel through some of techniques and architectures from the literature to try to implement a model for the prediction of movie's revenue in the USA. To this end we will work on the well known International Movie Database which is one of the richest databases. In fact we will be dealing with categorical, numerical, text and embedding features and this will ask for a huge data processing at each step of our implementation. Moreover we will tackle some issues as the way we could manage all this features or not in linear regression, K-nearest neighbors, Multi-layer perceptron or the Gradient Boosting Regression. In each part of this report we will discuss the chosen parameters, the retained features and maybe some improvements of the methods used.
But first of all, let's summarize our data visualization as a starting point to our analysis. In fact, as we can already remove the binary *is adult* feature as it seems to be attributed to the same value for all observations, and thus will not be useful to construct our predictions, the *Unnamed: 0* will not to be given the same procedure as we display the correlations between our features.



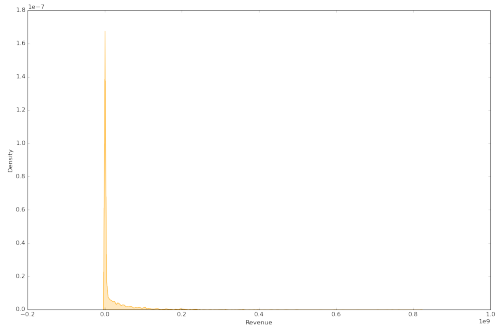

Let's point out that as we wanted to display as much information as possible we already show the interactions between transformed features such as the *Genre* and *Studio* for our correlation matrix, we will explain our procedure in the next section. Let's point out that the correlation of the studio's levels is clearly due to the bad first dummy encoding we have done for this feature. You can visualize it more clearer in our attached Notebook but we can already denote some significant correlations between features such as *production year* and *release year* but looking at the count barplot we decided to add a supplementary feature that manage the difference between the two values named *time*. In fact as we conclude that the number of produced films decreases in the past **15** years and stays below the number of released films, this feature may better capture the decreasing spreading of the cross-behaviour of this two features (see appendices). As we look at the *Studio* feature we can already point out that bigger studios (which produce more films) are linked to bigger revenues, a good categorisation of this feature will be helpful. What's already interesting is that the *Revenue* is highly positively correlated with the *n votes* feature, which seems to make sense as it increases the visibility of the film. Finally, we denote high positive correlations between related *Genres* as **Reality-TV** and **Game-Show**, or **Action** and **Adventure**.
Especially, what has been to be checked is the correlation matrix of our embeddings. In fact, strong correlated evaluations raise a question on the relevance of performing these, as we could only keep one evaluation among the correlated ones, reducing the size of such vectors and thus speed up optimization of hyper-parameters. So we display the two matrix in the appendices and we point out such correlations for the *text embeddings* that's a vector of size **768** obtained by the *description* to a **BERT** language model. We will explain this procedure later in the Notebook.

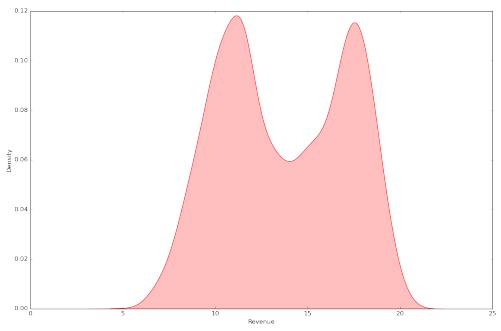
## 2 Pre-processing of features

As the stability of our chosen procedures depends on the pre-processing of the dataset, this will be huge and essential part of our engineering analysis and implementation. In fact several features have to be transformed before going further. What missing values concern we will choose to drop rows with missing values in a specific column. The advantages of doing so are that we do not have to impute any values, meaning that we are still not making any assumptions with our models and we only use sampled data, keeping **3628** film observations.
Thus, as the *Genres* feature is a list of maximum 3 genres that fit the movie best, we will be dealing with different lengths of vectors. The best way to deal with such a feature is the **One Hot** or **dummy** encoding instead of the label one. Thus we will create a new binary column for each genres that will take **1** value if the genres is in the observation list and **0** otherwise. Even if it increase the dimentionality of our dataset, it's a common way to keep information of categorical data while transforming it into numerical data. But as we have seen in our already preprocessed correlation matrix, this creates multicolinearity and thus we will keep an eye on it in the following sections.
The second main feature to be transformed is the *Studio* one as it's clear that big studios as **Warner Bros** or **Universal** will generate more film with high gross revenues. Thus as the size of the studio seems to be a good discriminant parameter, we will run a little aggregating algorithm to group this feature as [**Independent**, **Small**, **Middle**, **Leader**] for studios attributed to respectively [$\mathbf{1}$, $\mathbf{1 < n \leq 40}$, $\mathbf{40 < n \leq 100}$, $\mathbf{100 < n \leq 200}$] where $\boldsymbol{n}$ is the number of films in the category. We will display the revenue summary statistics in the following table to asses for the good cut of our implementation, but you can find **histogram, densities or boxplots** in the appendices and Notebook provided:

| Studios categorization | | | |
|---|---|---|---|
| Size of studio | count | mean Revenue | 95% Conf. Interval |
| Independent | 180 | 4.1 * $10^6$ | [1.02 * $10^6$ ; 7.23 * $10^6$] |
| Small | 1765 | 9.05 * $10^6$ | [7.50 * $10^6$ ; 1.059 * $10^7$] |
| Middle | 562 | 5.51 * $10^7$ | [1.89 * $10^7$ ; 2.71 * $10^7$] |
| Leader | 768 | 6.73 * $10^7$ | [6.11 * $10^7$ ; 7.34 * $10^7$] |

This time we chose to recode this feature as an ordinal categorical one to prevent multicolinearity and thus variance inflation factors that can become higher than a fixed threshold in this case.
Looking at the *Revenue* target feature, it's obvious that the relation between our dependent features and target will need to be transformed. In fact, the spreading and the magnitude of the already adjusted for inflation amount of dollars will be the subject of the well known **Box-Cox** test which gives us an optimal power transformation $\lambda = -0.0011$ and thus suggest a logarithmic variance-stabilizing transformation.
And so, as $\sigma^2 \propto (\mathbb{E}(y)) \longrightarrow y' = log(y)$, the same assumption as been checked for the *n votes* dependent feature and thus with $\lambda = -0.0015$, the same *log* transformation is applied. We display below the density of both original *Revenue* and thus the *log* transformed target:




We see that the bi-modality of our transformed target suggest us a distribution that looks like a mixture of Gaussian.

The parameter of this distribution may be estimated with the **Expectation Maximisation algorithm** available with *sklearn* package function *GaussianMixture* in order to build **General Linear models** with target distribution assumptions but this goes beyond the limits of the project, where we will be focus on other procedures.
As it's time to tackle the pre-processing of our embedding features, we first try two **unsupervised** dimensionality reduction with the **PCA** and **T-SNE** algorithm for the *text embedding* and *image embedding* respectively. In fact, this is done by reducing the number of embedding attributes while attempting to keep as much of the variation in these.
The results are provided in appendices but as it capture relatively well the explained variance with *text embedding*, as expected it's not true for the *image embedding* as on a non-correlated set the projections are very close to the original vectors. And thus it motivates the T-SNE as it uses non-linear approaches to dimensionality reduction and as such can often capture more complex relationships between our embedding attributes.
But as it has been performed we will try another approach as the methods above create new combinations of attributes from existing ones, feature extraction simply removes features, and thus further we will prefer **supervised** learning by using the *Revenue* target feature to identify the features which can increase the efficiency of our models. We will explain this procedure in the next section but the last thing to know about our pre-processing is that to this end we convert each embedding attribute of both text and image to a respective column in our dataframe. It gives us a final dataset with **3628** rows and **2855** columns.

# 3 Models comparison

## 3.1 Linear Models

First of all we will point out that since our features are correlated we need to standardize the independent ones using the *StandardScaler()* python function in order to center the features and then to reduce Structural **multicollinearity**. This has been done for both independent features and embeddings. Moreover we can investigate the **Variance inflation factors** to assess for the reliability of our regression results but we have to go further in features selection procedures as we know that correlation does not measure nonlinear relations and that correlation does not mean causality.
The last step to assess for the good fit of our procedure and to keep an eye on overfitting behaviour will be to cut our dataset in **train** and **validation** sets. To this end we will choose splitting percentages of **80/20** %.
As we run the linear model with all the features excepted the embedding we can asses for the good choice of our target *log* transformation as it increases the $R^2_{train}$ and $R^2_{test}$ from **0.38** to **0.55** and from **0.34** to **0.48** with the same number of parameters respectively. Moreover, a regression model where the outcome and at least one predictor are *log* transformed is called a **log-log linear model**. As it will be the reference error metric in this report we will compare the **root mean square error** of our models. But first as we will try to extract pertinent information from our large concatenated dataframe and in order to avoid the curse of dimensionality and risks of overfitting we will use a **filter univariate selection method** based on to the k highest *F* scores that:

- Start with a constant model, $M_0$
- Try all models $M_1$ consisting of just one feature and pick the best according to the *F* statistic
- Try all models $M_2$ consisting of $M_1$ plus one other feature and pick the best

But as we know that it usually gives lower prediction performance than wrapper methods we will use it as a first step for a quick screen and removal of irrelevant features before before applying our **wrapper** feature selection procedure. For comparison with our embedding-less previous dataset we choose a number of features $k$ parameter of **35** to compare the selected features with the original pre-processed dataset and it's surprising that with the same level of $R^2$ and $\boldsymbol{rmse}$ our function prefer **23** embedding attributes instead of some of genres attributes or features as *ratings*. But it was to be expected as we have said previously that *genres* levels where highly correlated, the same

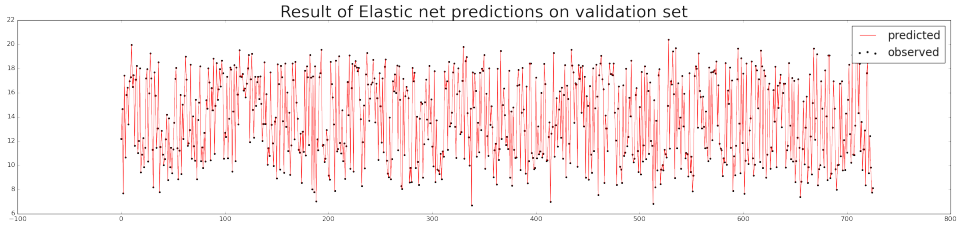
conclusion can be made for *ratings* as *n votes* is kept in the model. Finally as we do not want to choose arbitrarily the number of features to be included in the model, we choose to implement a vanilla stepwise regression with a *threshold in* and *threshold out* for features significant **p-values**. This procedure helps us to retain only **38** features over the **2854** of our high dimentional dataset.

But as we are still trying to avoid singularities and as with get around 2800 features, we can't even estimate the model properly, we will try to use **Ridge** and **Lasso** regularization procedures. Variables normalization is the initial procedure in regularized regression. Both the independent and dependent variables require this transformation to scale these between **[0,1]**. As Lasso works well when there is a few parameters that impact the response feature and Ridge is appreciated when at the opposite most of features influence the target, we will compare their performances. Thus we compute the efficient $\alpha$ regularization parameters by cross-validation, for the Ridge **L2 regularization** it gives $\alpha = 0.08$ and for the Lasso **L1 regularization** it gives $\alpha = 0.001$ (see appendices for the cv results).

Moreover as the **Elastic net** is appropriate when the features form groups that contain highly correlated independent features, what's to expect with our *Release Year*, *Production Year* and *Time* features looking at the **VIF** and the cross-feature plot in appendices. Therefore we choose to implement a grid search algorithm with a **3** times repeated **5** K-folds cross-validation based on the **negative mean absolute error** as evaluation of the performance of the cross-validated model on the test set. In addition we pass a dictionary with the algorithm $\alpha$'s and *l1 ratio*'s to be tested.

| Linear models performances | | | | |
|---|---|---|---|---|
| feature selection procedure | Train | | Validation | |
| | $R^2$ | RMSE | $R^2$ | RMSE |
| K-best | 0.60 | 2.11 | 0.61 | 2.17 |
| Stepwise | 0.64 | 2.02 | 0.62 | 2.14 |
| Ridge | 0.81 | 1.44 | 0.60 | 2.19 |
| Lasso | 0.64 | 2.00 | 0.62 | 2.13 |
| ElasticNet GS | 0.65 | 2.00 | 0.63 | 2.14 |
| ElasticNet CV | 0.7 | 2.05 | 0.6 | 2.22 |

Through other tests with the ElasticNet, by shifting from a grid search to a cross validation strategy or by not considering the embedding, we found other RMSE values for the validation set slightly better than the one shown in the table. However, for the consistency and conciseness of this report, we'll keep the $\alpha = \mathbf{0.001}$ and $l1_{ratio} = \mathbf{0.99}$ given by the ElasticNet alongside with the grid search because it consider less features than the other models. We display here our plot of fitted versus observed for our grid search **ElasticNet** on our validation set:




As it's clear that we get non independence in the data, such as arises from a hierarchical structure that may be explained by the *studio* feature, we will try to fit a **mixed model** to our target *Revenue* feature with respect to the *studio* feature. In fact we can imagine a random effect on both intercept and slope induced by the different levels of the *studio* feature. This is induced by the nested data we encounter after the recoding of this feature as we believe that the group membership of an observation has a significant impact on our target.

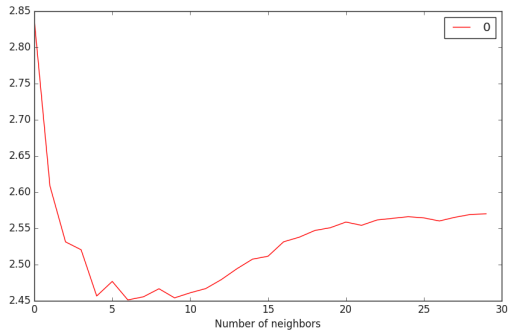
And thus our model will have the following expression:

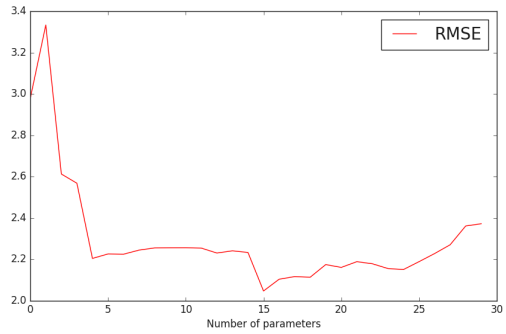
$$Revenue_{ij} = X\alpha + Z_j\beta + \varepsilon_{ij}$$

for observation $i$ in the group $j$, with $Z$ the binary BURT matrix that indicates to which group the observation owns and $\alpha$ and $\beta$ being the fixed and random coefficients respectively.

## 3.2 K-Nearest Neighbour

Here we enter in the world of the non parametric regression, as it's common to carry on the distribution of our target feature. In fact as for the non parametric regression, the target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set, then this **supervised** regression model with meta-parameters $K$ predicts the output of a sample as the mean output of the $k$ nearest neighbours in the features space. So, we will get, by fixing a number $k$ of observations once again found by cross-validated grid search, the optimal value. In fact we will run a little algorithm to compute the **root mean square error** for different $k$ values up to **30** and we will choose the value located at the elbow of our curve plot. We precise that as the *embeddings* didn't seem pertinent in our previous analysis we decided to remove them in this section. Therefore, we will display two plots that helps us to visualize the choice parameters for this algorithm:




And as it is not obvious on this plots we once again perform a grid search **5** k-folds cross-validated to find the right number of $k$ neighbours that seems to be **9**. What the number of parameters concerns we use a **forward sequential features selector** as it shows good performances in our linear regression and this gives us a number of parameters to be equal to **15**, and in addition this has been **10 k-folds** cross-validated. As we didn't want to choose arbitrarily the number of parameters we iterate over the range of our column space length to provide starting values to our algorithm. This may be considered as greedy but it works well so we keep this time consuming procedure.
Finally, as we plug in the parameters in our K-Nearest Neighbour regression model we obtain an improved **root mean square error** of **2.05**.

## 3.3 Neural networks

In this section we will build neural network models with one and many layers and compare our results. While it's less interpretative than our previously build models, it does have the universal approximation property that allows for non-linear function estimations. In fact, according to the universal approximation theorem, a single hidden-layer neural network can approximate any measurable function arbitrarily well, provided the number $K$ of hidden units is sufficiently large. As for our previous construction, for the **NN preprocessing** categorical features are modeled as **dummy** or **ordinal** numeric features. We will implement a **multi layer perceptron** with the python *Keras* API and choose to perform a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments, known under the name of **ADAM**. We choose this procedure as it's *"computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters"* according to ***Kingma et al., 2014***.
What the architecture of our model concern, we display the diagram in the appendices. We choose three independent **input** layers for the two embeddings and the numerical recoded and scaled features. What embedding concern, we use two **flatten** layers to reshape them. Then we concatenate all the inputs and pass them in our computational **dense** layers.

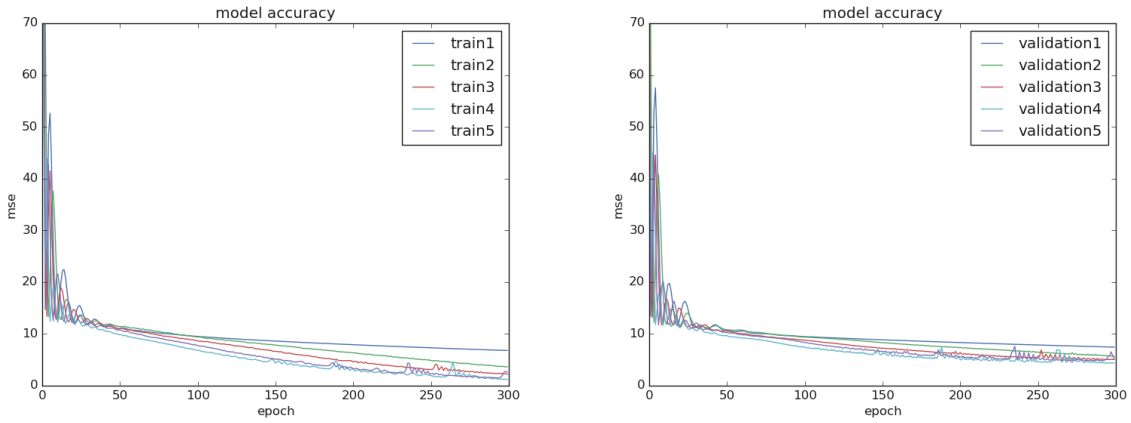
We will add a dropout layer with the fraction of the input units to drop that consists in randomly setting a fraction rate (**rate = 0.3** choosen) of input units to 0 at each update during training time, which helps prevent overfitting. Moreover, we choose a batchsize of **2000** as the larger the batch, the better the approximation and samples in a batch are processed independently, in parallel.
Even if it is very hard to attain the global minimum of the loss and therefore to get the theoretically zero coefficients, more generally, we get the loss function (with $\Omega$ the set of parameters):

$$R(\Omega) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y_i, \hat{y}_i^{net})$$

And as the optimal weights are found by minimizing $R(\Omega)$ such that $\Omega = argmin_\Omega \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y_i, \hat{y}_i)$, adjusting the vector of weights $\Omega_t$ by a small step in the opposite direction of the gradient, and working under the **mean square error** function we obtain the following results:

| Multi layer perceptrons performances | |
|---|---|
| | Validation RSME |
| $K_1$=4000 | 2.97 |
| $K_1$=4000, $K_2$=200 $K_3$=50 | 2.68 |
| $K_1$=4000, $K_2$=500 $K_3$=50 | 2.53 |
| $K_1$=10000, $K_2$=500 $K_3$=50 | 2.41 |
| $K_1$=10000, $K_2$=2000 $K_3$=200 | 2.32 |

We choose **300** maximum steps for the training of the neural networks with a threshold for the partial derivatives as the ADAM optimizer converges very fast. We display the convergence plots for the train and the validation set:




## 3.4 Gradient Boosting with regression Trees

Gradient Boosting is a sequential method that combines sequentially weak predictive models into a more powerful one, typically solved by gradient descent:

$$f_{GB}(x) = argmin_f \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i))$$

where $\mathcal{L}$ is a penalty function as a function of mean square errors. We then use $y_i$ to evaluate our $RMSE$. Thus, an iterative algorithm to locally improve a current approximation is performed by minimizing the in-sample loss. In fact, we perturb $f_m$, at iteration $m$, so that it leads to a maximal decrease in this loss, choosing a small step of size $\rho > 0$ so that. In practice, we use a Histogram-based Gradient Boosting Classification Tree (or HGBM) which is much faster than a classic GBM for big dataset. Here, the loss function is calculated based on the least square,

the learning rate is equal to **0.001** and the maximum depth of the HGBM tree is **8**. Moreover as we perform **L2 regularization** as we have already seen in the features selection before, and with a maximum number of iteration equal to **15000**. And finally, we obtain a root mean square error of **1.55** for our *log* transformed target. Moreover, this machine learning algorithm allows us to plot features impact on the target. This will be displayed trough 2D and 3D **Partial Dependence Plots** in appendices. We can clearly recognize some expected effects as for the *n votes* feature our for the *studio* categorization and as we clearly see that the *Unnamed: 0* feature have significant impact when it increases we keep it in our final model.

### 3.4.1 Calibration for relevant subgroups

We will split the train and test set into groups, and use the predictions to asses if the model is well calibrated. To do so, we will use the **Chi-Squared** test that is $\boldsymbol{H_0}$ : identical observed and expected Revenues given by the test statistic:

$$\hat{C} = \sum_{g=1}^{G} \frac{(O_g - E_g)^2}{E_g} = \sum_{g=1}^{G} \frac{(R_g \times n_g - \hat{R}_g \times n_g)^2}{\hat{R}_g \times n_g}$$

This statistic is computed through groups discriminated by the levels of the following features **studio** and that gives a total number of groups $\boldsymbol{G = 4}$ displayed on the next plots.

| Multi layer perceptrons performances | | |
|---|---|---|
| | Observed $\mu_g$ | Predicted $\mu_g$ |
| Independent | 11.19 | 10.85 |
| Small | 12.11 | 12.12 |
| Middle | 13.61 | 13.75 |
| Leader | 17.05 | 17.03 |

And thus looking at the sample sizes obtained in the preprocessing section of this project we can compute the test statistic $\hat{\boldsymbol{C}} = \mathbf{15.26}$ and as we compute the **Chi-Squared** test we can say that our model is well calibrated trough studios.

## 3.5 Choice of the best model and predictions

As we have to compute the prediction for the new target removed *X2.csv* data file, we first have to clean it. In fact as this dataset contains missing values for both the *runtime* and *genres* features we will choose two different strategies. For the *runtime* feature we simply replace the **NA** by the feature mean value and what *genre* concern we pass the *description* into a classification algorithm to retrieve it from this feature. As we have seen before, the **Histogram-based Gradient Boosting Regression Tree** seems to give us better results in term of ***RMSE*** and it motivates us to use this procedure for our predictions.

# References


Brownlee, J. (2019, November). *How to Choose a Feature Selection Method For Machine Learning.* Retrieved 2022-12-22, from https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

Brownlee, J. (2020, December). *Histogram-Based Gradient Boosting Ensembles in Python.* Retrieved 2022-12-22, from https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/

Charan, R. (2020, July). *Expectation Maximization Explained.* Retrieved 2022-12-22, from https://towardsdatascience.com/expectation-maximization-explained-c82f5ed438e5

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction* (Second ed.). Springer. Retrieved from http://www-stat.stanford.edu/~tibs/ElemStatLearn/

*In Depth: Principal Component Analysis | Python Data Science Handbook.* (n.d.). Retrieved 2022-12-22, from https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

Melit Devassy, B., & George, S. (2020, June). Dimensionality reduction and visualisation of hyperspectral ink data using t-SNE. *Forensic Science International*, *311*, 110194. Retrieved 2022-12-22, from https://www.sciencedirect.com/science/article/pii/S0379073820300566 doi: 10.1016/j.forsciint.2020.110194

Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General)*, *135*(3), 370. Retrieved 2022-12-22, from https://www.jstor.org/stable/10.2307/2344614?origin=crossref doi: 10.2307/2344614

*NLP — Machine Learning, Statistiques et Programmation.* (n.d.). Retrieved 2022-12-22, from http://www.xavierdupre.fr/app/mlstatpy/helpsphinx/c_nlp/index.html

Parhi, R., & Nowak, R. D. (2021). Banach space representer theorems for neural networks and ridge splines. *Journal of Machine Learning Research*, *22*(43), 1–40. Retrieved from https://arxiv.org/abs/2006.05626 doi: 10.48550/ARXIV.2006.05626

Ridgeway, G. (2007). *Generalized boosted models: A guide to the gbm package*. Retrieved from https://moodle.uclouvain.be/pluginfile.php/173575/mod_resource/content/0/gbmExplanations.pdf

*Régression non linéaire avec un réseau de neurones |.* (n.d.). Retrieved 2022-12-22, from https://lucidar.me/fr/neural-networks/curve-fitting-nonlinear-regression/

*sklearn.linear_model.ElasticNet.* (n.d.). Retrieved 2022-12-22, from https://scikit-learn/stable/modules/generated/sklearn.linear_model.ElasticNet.html

*sklearn.linear_model.Lasso.* (n.d.). Retrieved 2022-12-22, from https://scikit-learn/stable/modules/generated/sklearn.linear_model.Lasso.html

*sklearn.linear_model.Ridge.* (n.d.). Retrieved 2022-12-22, from https://scikit-learn/stable/modules/generated/sklearn.linear_model.Ridge.html

*Visualizing Embeddings With t-SNE.* (n.d.). Retrieved 2022-12-22, from https://kaggle.com/code/colinmorris/visualizing-embeddings-with-t-sne

# 4 Appendices

## 4.1 Embeddings correlation matrix

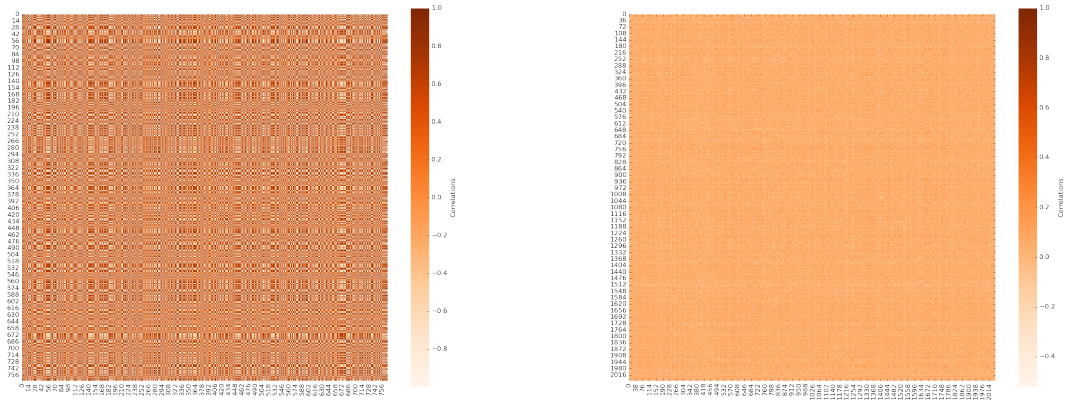
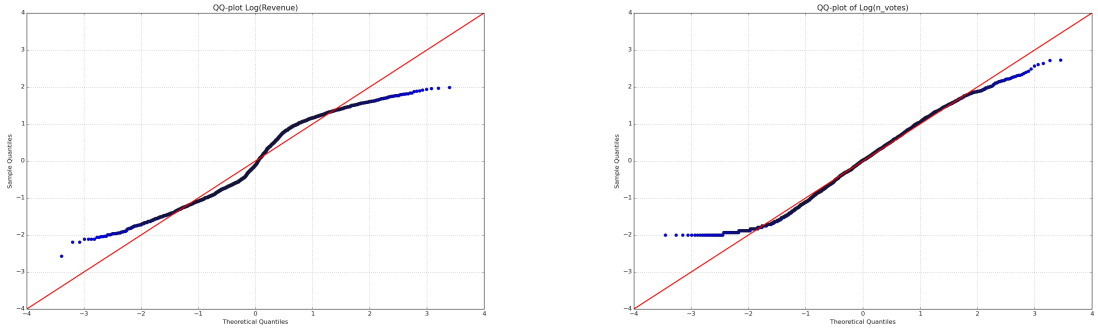


Figure 3: Correlation of text and image embeddings

## 4.2 Quantiles of *log* transformed Revenue and n votes



## 4.3 Barplot and boxplot of studio sizes (*log*-Revenue)

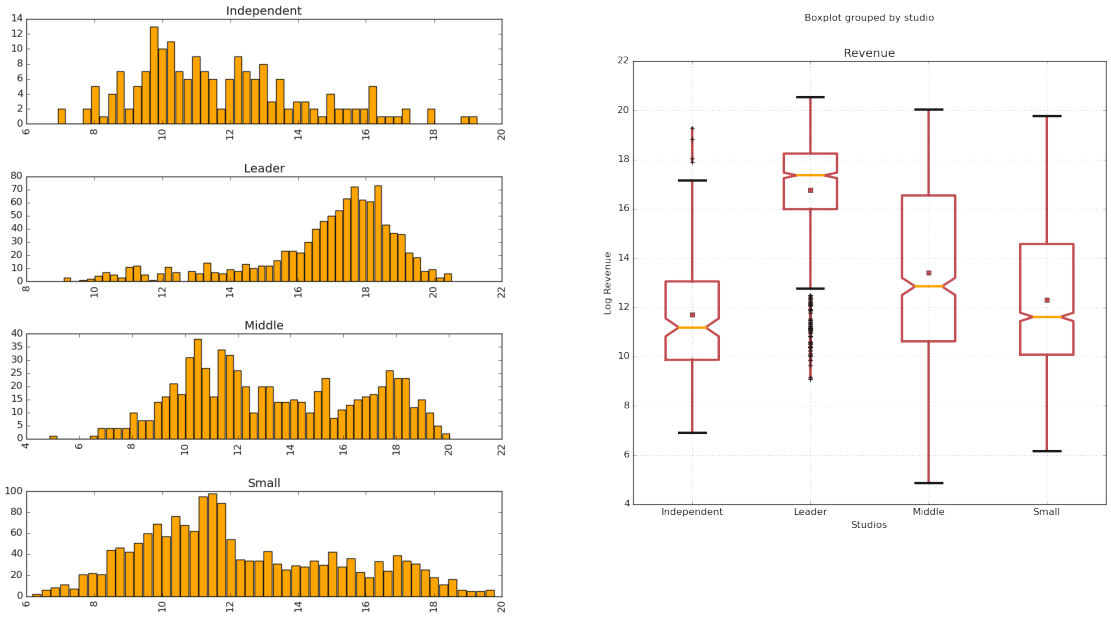



Figure 5: Revenue attributions in new recoded studios

## 4.4 2D and 3D PCA for text embedding

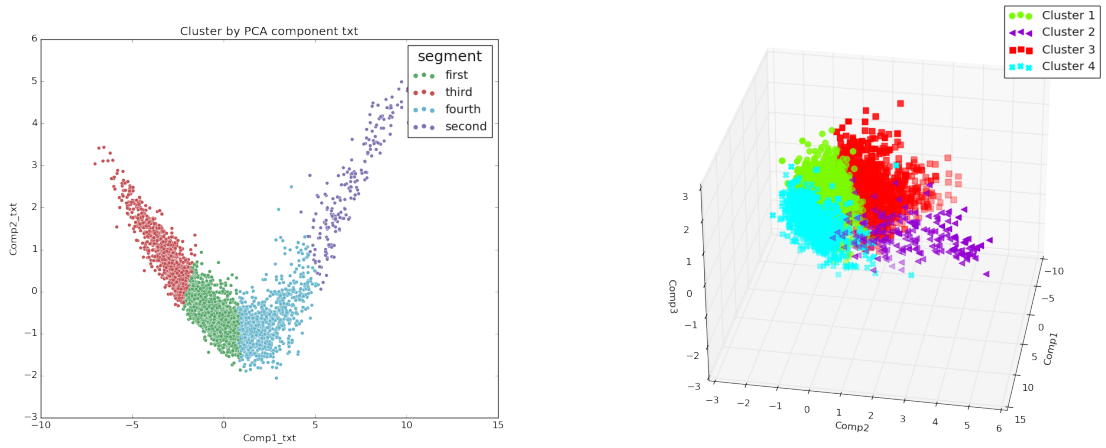



Figure 6: K means clustering with inertia optimization

## 4.5 2D T-SNE for Image embedding

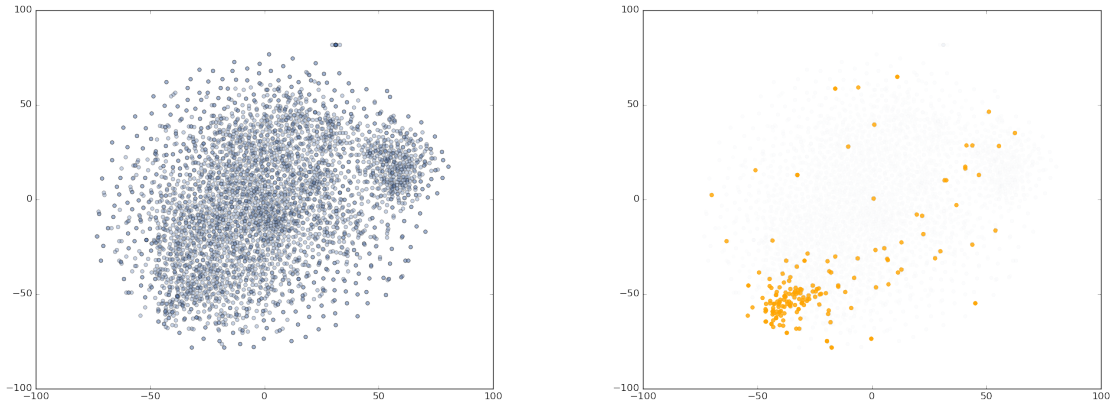


Figure 7: 2D dimentionality reduction and location of *Animation* observations

## 4.6 K best and VIF

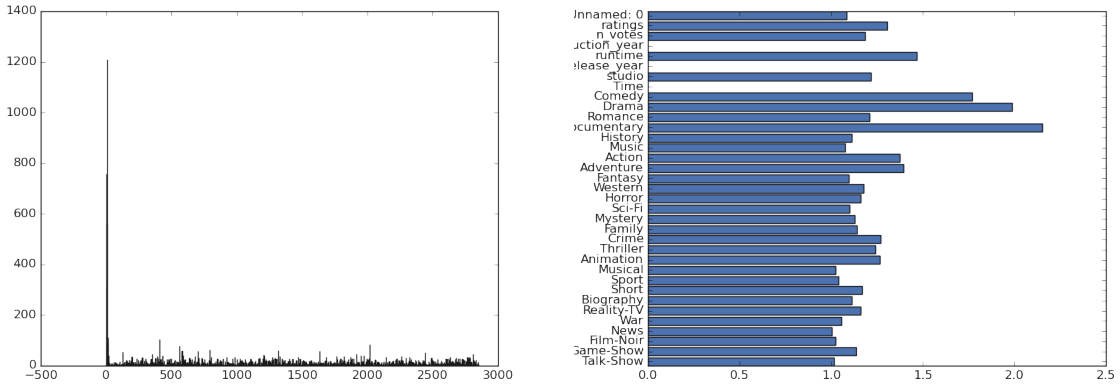



Figure 8: 2D dimentionality reduction and location of *Animation* observations

## 4.7 Ridge and Lasso

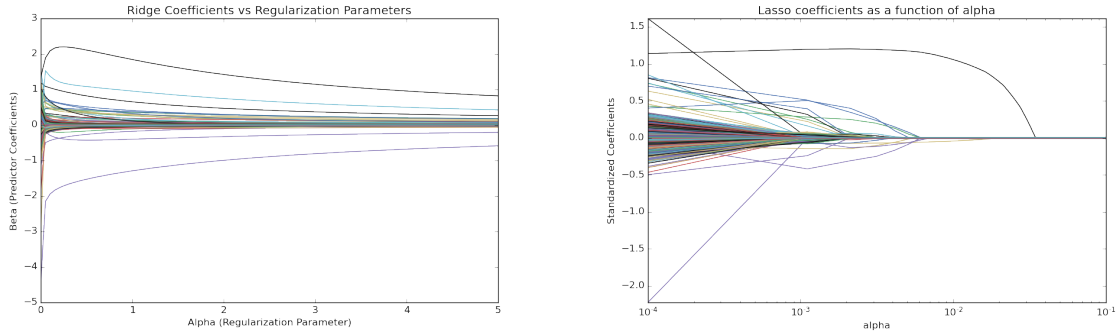



Figure 9: RIDGE and LASSO regularization's

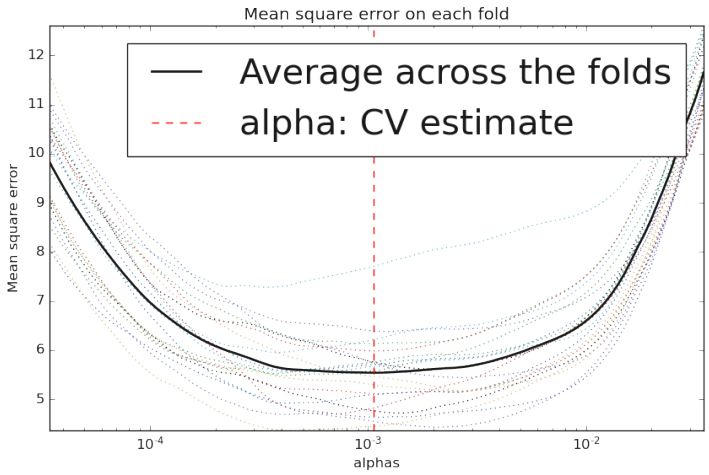
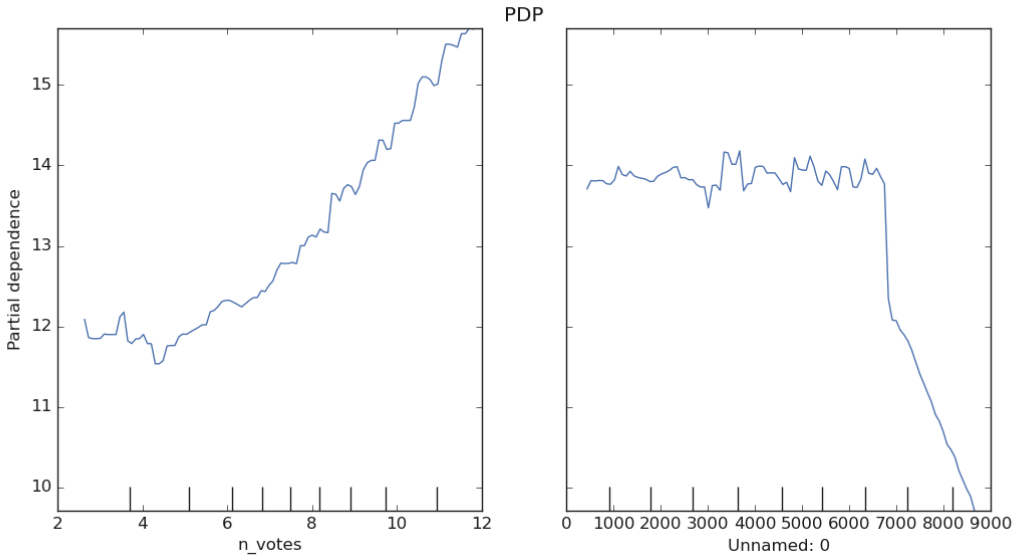



Figure 10: CV LASSO parameter estimates

## 4.8 Partial Dependence Plots for interesting features



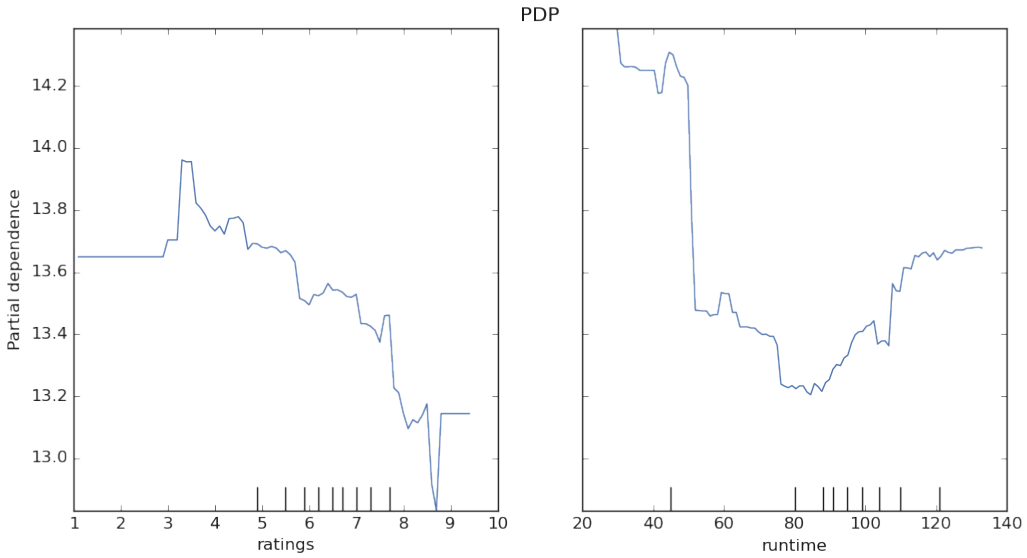



Figure 11: PDP of 4 controverted features

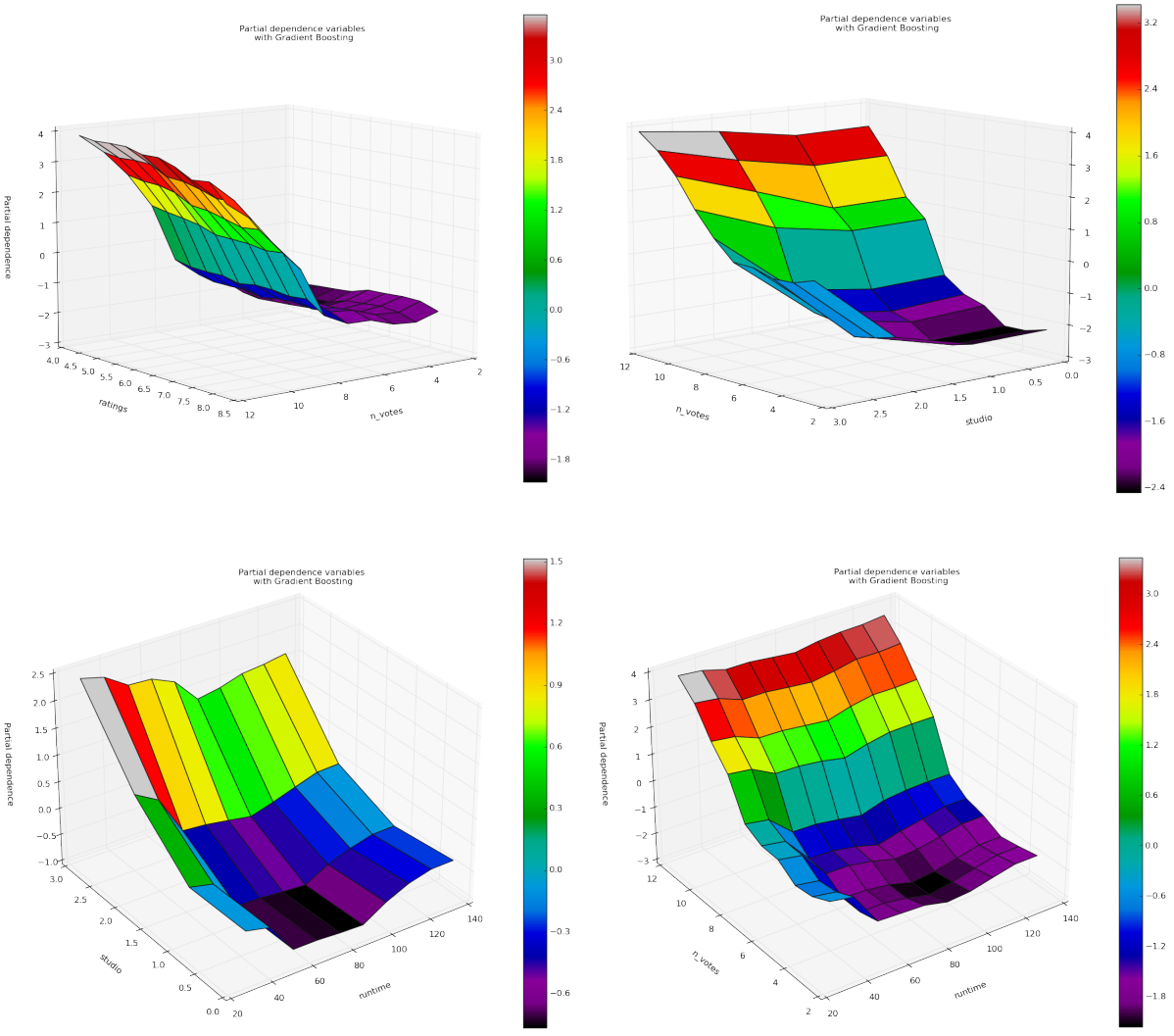



Figure 13: 3D cross features PDP