17/01/2021

Prediction of the success of a Kickstarter project

Advanced methods in big data application

DEVILLARD Aurélien-FAUCOMPRE Clément-PFAU Maurice
AIX-MARSEILLE SCHOOL OF ECONOMICS

1 Table of contents

Based on a dataset of Kickstarter projects, we built new variables to better capture all the information, and, through testing of different Econometrical models and Machine Learning methods we are able to build a predictive model. This model allows us to predict the success or failure of a crowdfunding project with an accuracy of 70%.

2 Introduction

2.1 Presentation of Kickstarter

In 2020, more than 34 billion dollars have been raised through crowdfunding worldwide. This amount is increasing every year, and with it, the interest of investors for this type of project. There are several big online crowdfunding platforms, among which Kickstarter is the most important and famous one. Crowdfunding is a way to raise money from a wide range of people, by submitting an idea of a project. Anyone can submit and idea of project on the platform.

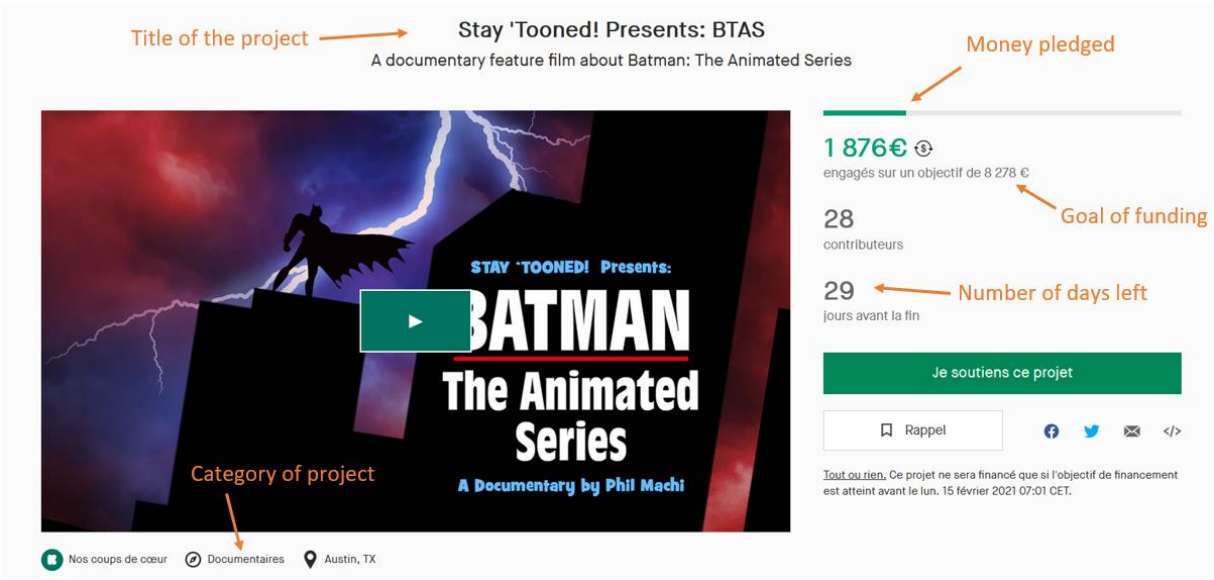


Figure 1: Example of a project

On Kickstarter, the backer can give money for a project, in exchange for a reward (usually something related to the project). The reward is chosen by the creator and depends on the amount of money given by each backer. A backer can also choose to simply donate and receive no rewards.

Finally, this platform follows an all or-nothing funding model. If the funding fails (it does not reach the amount of money asked), the money will go back to the donors, and the project is cancelled. If the goal is reached or even exceeded, all the funds will be available for the creator, and he can decide what he does with the surplus. Any kind of project can be submitted, from the making of a movie to a technologic invention.

When a creator launches a project, he cannot know in advance if it will succeed or not. What this study aims to do is providing a statistical intuition of chances for a project to success based on its characteristics, and finally create a model that can predict at the launching of the Kickstarter project, if it is going to fail or not.

This can be useful in two types of situations:

- For the creator: the model created can help the creator optimizing its project before officially launching it.
- For potential donors: the model can help investors see if a project is worth looking at.

2.2 Literature review

Crowdfunding is a quite new way of financing projects, that is innovative as it does not depend on any bank or company lending, but on people that are interested in the creation of the project. The kinds of projects started on Kickstarter are very varied, as it can be projects for any domain (art, technology…) and it can need a small or huge amount of money to be accomplished, depending on the size of the project.

The success of a Kickstarter campaign depends on different parameters, starting by its quality. Mollick (2014) found that the personal network of the creator, and the project quality are very correlated with the success of the project. He found out that the number of relations the creator has on social networks was positively correlated with the chances of success. Besides, projects seem to often end-up in either of 2 situations, failure because of very few funds raised, or success with just enough money to fulfill the goal. Kindler and al. (2019) studied the impact of the "virality" of a project, to find out if the influence of backers on each other is the reason for success on Kickstarter and showed that it was not a determinant of the final success.

Etter and al. (2013) developed a model to predict the success of a Kickstarter campaign with the data available 4 hours after the launch of the project, using the amount of money pledged, and social feature. This combination of explanatory variables allowed them to reach a global accuracy of 76%.

Language can also be exploited to predict the success: Sawhney and al. (2016) are using campaign content (such as the goal, the deadline, the data of creation) and linguistic features through Natural Language Processing, allowing them to predict the final state of a project with a global accuracy of 70%. They show that the campaign done for the project, and the language are quite good predictors for success of projects, allowing to give advice to accomplish an efficient marketing campaign.

2.3 Methods used and goal of the project

In order to build a model to predict the success of a project, we use advanced machine learning methods, from logistic regressions to deep learning model combined with feature selection. From the initial database, we build new explanatory variables to try to understand the underlying relationships. We used different models and ended with a comparison of all of them to see which one gave better results and would be useful for our objective.

3 Material and methods

3.1 Presentation of the data

The data used in this project come from Kaggle (https://www.kaggle.com/kemical/kickstarter-projects). The data are collected from the Kickstarter platform, and they cover projects that have been launched between 2009 and 2018. The database initially contains 378 661 observations of projects. After some cleaning of the data (details in the section below), a selection of the explaining variables and a creation of some new explanatory variables, we end up with the following variables:

Name of the variable	Type	Description	Values associated
usd_goal_adj	Numerical	Goal of the project adjusted to inflation and converted in US dollars (conversion from Fixer.io API)	Positive float
money_per_day_adj	Numerical	Goal of the project in US dollars divided by the number of days to accomplish the project	Positive float
number_of_day	Numerical	Total number of days to pledge money for the project	Positive
count_7_days	Numerical	Number of Kickstarter projects launched in the last week (7 days rolling)	Positive integer
len_name	Numerical	Length of the name of the title of the project (number of characters)	Positive integer
usd_goal_sq	Numerical	Squared goal of the project adjusted to inflation and converted in US dollars (conversion from Fixer.io API)	Positive float
time_last_proj	Numerical	Number of hours since a project was launched in the same category	Positive float
main_category	Character	Name of the main category of the project	Ex: “Film & Video”
main_category_cat	Numerical	Unique identifier corresponding to the	Ex: 6 for “Film & Video”

		main category of the project	
category	Character	Name of the sub-category of the project	Ex: “Narrative Film”
category_cat	Numerical	Unique identifier corresponding to the sub-category of the project	Ex: 53 for “Narrative Film”
country	Character	Country from where the project is launched	Ex: “US”, “GB”
country_cat	Numerical	Unique identifier from where the project is launched	Ex: 21 for “US”
currency	Character	Currency in which the money is pledged	Ex “USD”, “GBP”
currency_cat	Numerical	Unique identifier of the currency in which the money is pledged	Ex: 13 for “USD”
launched_year	Numerical	Year when the project was launched	From 2013 to 2019
launched_month	Numerical	Number of the month in the year when the project was launched	Integer: {1: January 2: February, …, 12: December}
launched_day	Numerical	Number of the day of the week in the month when the project was launched	Integer: {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
launched_hour	Numerical	Hour of the day when the project was launched	Integer between 1 and 24

Figure 2 Dictionary of the dataset used for our models (underlined are the created variables)

3.2 Data cleaning and creation of new features

3.2.1 Deletion of features

From the initial dataset, there are some features that we have removed. Indeed, as we want to predict the success of a project, we removed the following features that were not pertinent in the way we want to make predictions:

- The number of backers in the project (“backers”): this is not available at the launch of the project.
- The goal in local currency (“goal”): we remove it as we already have the corresponding goal in US dollar for each project.
- The amount of money pledged (columns: “pledged”,”usd_pledged”, ”usd_pledged_real”): this is not available at the launch of the project.
- The name of the project (“name”): considering the small number of characters in the title, the fact that some words are invented as they correspond to the name of the project, and that it can be a repetition of the name of the category, we decide not to use it directly. A detailed

description of the project could have been more useful but was unfortunately unavailable in our dataset.

3.2.2 Data Cleaning

To keep up with a clean dataset, we review each feature to check the integrity of the observations and delete or transform them if needed.

As we want to create a classification model, we first need to encode a binary target variable. Initially, each project is characterized by its state, which takes 5 different values (Figure 2).

Value	**Frequency (in %, rounded to 2 decimals)**
failed	52.22%
successful	35.38%
Canceled	10.24 %
Undefined	0.9 %
Live	0.7 %
Suspended	0.5 %

Figure 3: Initial target values (variable state)

We decide to drop the observations with a target variable corresponding to “undefined”, “live” and “suspended” as the total of those observations represents a neglectable part of the dataset (around 2% of it), and the target variable is not exploitable. Then, we decide to merge the “failed” and “canceled” projects together into a “failed” state: indeed, as we want to predict the success of a project, a project which is canceled is not led to its term, as the creator of the project does not receive any funds since he cancelled his project.

Finally, we encode our target variable as following: 0 for failed, 1 for successful.

The dataset is not completely balanced between the failed and successful projects, but as the imbalance is not too extreme, we decide not to rebalance the dataset and keep up with our initial observations.

Concerning the country in which the project has been launched, we find out that some observations are labeled “N,0"”. It apparently corresponds to an error from the Kickstarter website. Those observations represent only 1% of the dataset, we decide to drop them.

We also find that 4 projects are labeled with no name, and that 20 projects have an aberrant launch date, so we also drop those observations. Finally, there are no duplicates in the dataset, meaning that we have finished dropping uncomplete or redundant observations. In the end, we have **370 213** observations in the dataset.

3.2.3 Variables transformation and creation

3.2.3.1 Launching period and goal of the project

From the initial dataset, we have the exact launch date of the project, and its deadline. To use it as input for our model, we replace those initial features (launched_date and deadline) with transformations:

- We extract the year, month, day of the week and hour of launch into new columns, respectively “launched_year”, “launched_month”, “launched_day” and “launched_hour”.
- We create a new variable called “number_of_day”, which corresponds to the number of days between the date of launch of the project and the deadline. This allows us to consider the length of the funding campaign.
- We create a “money_per_day” variable: it corresponds to the goal in US Dollars, divided by the number of days during which the project is online. This new feature helps measuring the constraint induced by the period and the total amount of money needed.

3.2.3.2 Interactions between projects on Kickstarter

On top of that, we want to insert new explanatory variables, that are acknowledging not only the characteristics of the project at stake, but also the environment on Kickstarter. Indeed, if similar projects are launched during the same period, or if there are just too many projects launched in the same period, this may have an impact on the visibility of the project for the possible investors, and thus limit the number of visitors on the page of the project. To acknowledge this possible constraint, we create 2 new features:

- count_7_days: counts the total number of Kickstarter projects launched in the last week (7 days rolling).
- time_last_proj: counts the number of hours since a project was launched in the same category.

3.2.3.3 Merging of category of projects

Each Kickstarter project belongs to a main category and a category. Thus, some categories contain just a few projects, and as we use those categories as features, it may be a problem for our model. That is why we merge some categories together to have big enough categories. We changed the category of some projects to another existing category, which belongs to the same main category. We removed the categories with less than 1000 observations by merging them. Thanks to this, we reduced the number of categories from 159 to 83 categories (see Appendix 1).

Ex: Category “Puzzles” is now assigned to category “Tabletop Games”.

3.2.3.4 Title of the project

To use the name of the project that is available in the database, we decide to measure the length of the title of the project (number of characters) and use it as a new explanatory variable, called “len_name”. This measure has been proven to be pertinent in scientific articles (see Y. Bramoullé and L. Ductor (2016)), which encourages us to try this transformation. As already explained above, the initial “name” feature is not used and then removed from the dataset.

3.2.3.5 *Non-linearities in the features*

Some of the models we plan to use do not take in account the possible non linearities in the relations, such as the Logistic regression. Consequently, we try looking at our few numerical features to see if any transformation could be helpful for the explanatory power of those models. Thanks to the plot (Fig 2) we show that the success rate regressed on the amount of money needed follows a kind of quadratic form ($y = 0.5 - 0{,}4x^2$). Based on this observation, we decide to create a variable corresponding to the square of the goal in us dollars, called "usd_goal_sq".

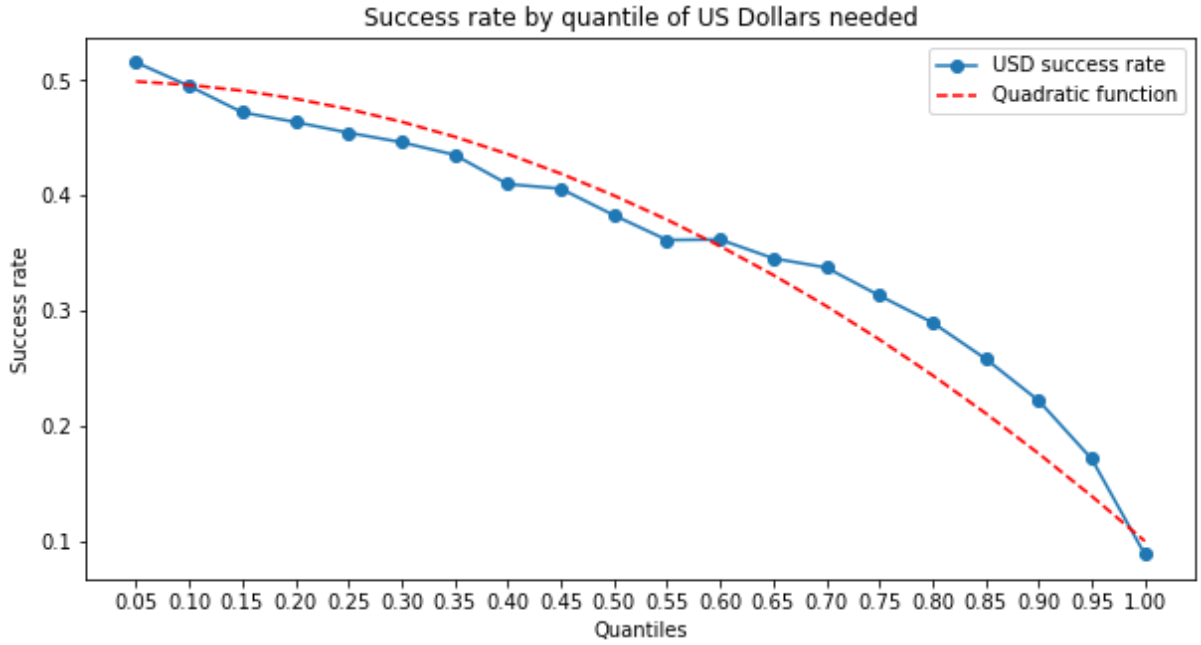


Figure 4: Plot of the success rate by quantile of US Dollars needed justifying a quadratic transformation

3.2.4 Train, test, and validation datasets

As we have quite a big dataset, to perform the training and evaluation of the model, we split randomly the final dataset into 3 partitions:

- Train data (236936 observations, 64% of total): used to train the model.
- Test data (59234 observations, 16% of total): used to compare accuracy within a model with different hyperparameters, to find the "optimal" model.
- Validation data (74043 observations, 20% of total): used to compare accuracy between the models, with data that has never been "observed" by any of the models.

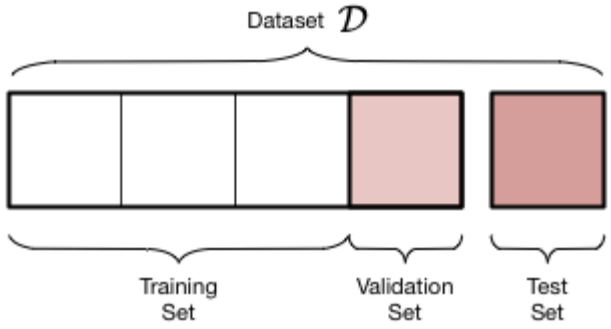


Figure 4: Illustration of the splitting procedure of the dataset

Each time randomness has been included in our code, we fixed the seed with a random state equal to 0, to have replicable results.

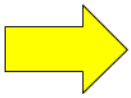
3.2.5 Adjustment of the variables

3.2.5.1 Categorical variables

It concerns the following variables: ['launched_hour', 'launched_year', 'launched_month', 'launched_day', 'category_cat', 'main_category_cat', 'country_cat', 'currency_cat']

Some models need “one hot encoded” values as input: the categorical values, that we have recoded into numerical values, need to be one hot-encoded so that the models do not consider them as continuous or ordinal variables. We then convert the categorical variables into dummy variables, which will considerably increase the number of columns.

Color
Red
Red
Yellow
Green
Yellow



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Figure 5: Example of One Hot Encoding with dummy variables

3.2.5.2 Continuous variables

It concerns the following variables: ['usd_goal_adj', 'money_per_day_adj', 'number_of_day', 'count_7_days', 'len_name', 'usd_goal_sq', 'time_last_proj']

We standardize the continuous variables of the dataset for two reasons:

- It reduces the computational costs for the computer as it deals with smaller numbers.

- Each variable has the same content and format thanks to this process: initially, the range of our continuous data is very different, for instance between the goal and the length of the name of the title. Thanks to this standardization, for the machine learning models based on distance computation, each feature is treated equally in terms of contribution.

Standardization corresponds to removing the mean and dividing by the standard deviation for each value.

$$standardized\_value = \frac{value - mean}{standard\_deviation}$$

We use the StandardScaler procedure available in sklearn. Once this is done, each feature has a mean of zero and a standard deviation of one.

3.3 Feature Selection with Principal Component Analysis (PCA)

Principal Component Analysis is a data analysis method used to reduce dimensions when a dataset is made up of too many features which might lead to worse models. PCA allows us to measure the part of the variance that is explained by each component. Each component is made up of linear combinations of the different features.

To compute the principal component analysis, the first step is to compute the covariance matrix, and then to compute the eigenvectors and eigenvalues of the matrix.

Each eigenvector is a component made up of linear combinations of the features, and thus, we can analyze how much of the variance is explained through each of the vectors.

We perform PCA on our dataset and try to find how many components are needed to explain at least 85% of the variance. We use the One Hot encoded dataset, which we standardize beforehand as well. We use the functions from the SciKit Learn package from Python.

The results we get show that, with 47 components, we are able to explain 85.06% of the variance.

	principal component	explained variance ratio
0	1	0.125239
1	2	0.095697
2	3	0.090515
3	4	0.045901
4	5	0.038609
5	6	0.021664
6	7	0.021264

Figure 6: First principal components

We can see above the most important components, and the ratio of the variance explained by each one of them.

Usually, PCA is not the best method for categorical data, as the covariance matrix is not meaningful then. We still perform it to illustrate how it works. Also, PCA is needed when dimensions are too big compared to the size of the dataset, but in our case, we had a large enough dataset, with not too many features, thus it is not necessary to reduce dimensions for the building of our models.

3.4 Logistic Regression

Logistic regression is an algorithm used to predict a categorical variable Y given independent variables X. To predict the dependent variable, the model uses the Sigmoid Function to return a probability value between 0 and 1, combined with a threshold that classifiers the variable according to the given probability.

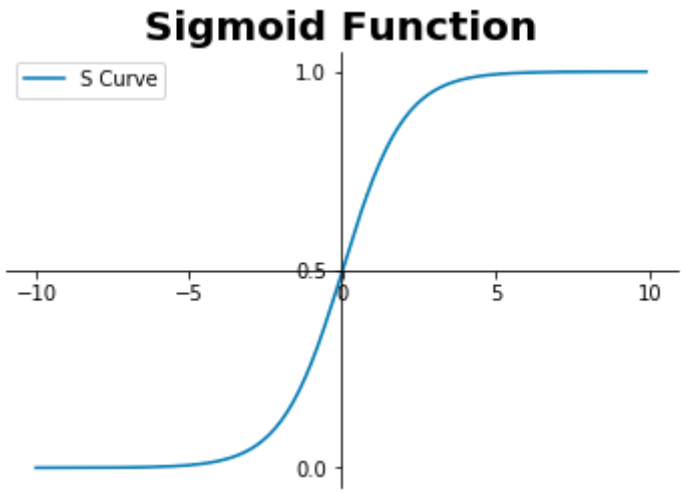


Figure 7: Illustration of a sigmoid function

In order to minimize error and find the optimal weights, the model tries to minimize the following cost function using gradient descent.

$$J(\theta) = -\frac{1}{m}\sum\left[y^{(i)}\log(h\theta(x(i))) + \left(1-y^{(i)}\right)\log(1-h\theta(x(i)))\right]$$

Figure 8 Cost function with gradient descent optimization

3.5 Ridge regression

Ridge regression is an extension of the linear model, which consists in adding a penalty to the cost function, the goal of the penalty will be to reduce the variance, by shrinking the value of the

coefficients towards zero. Behind those models, there is an idea of a tradeoff between the bias and the variance. Indeed, regularization increases the bias to reduce the variance of the model.

Logistic Regression with L2 norm implies that the loss function takes in account a penalty term, which gives us the following loss function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

With m the number of samples, n number of features.

The hyperparameter λ needs to be chosen, it is the regularization penalty. The closest it is to 0, the lowest the impact of the penalization, and then the regression will tend to a simple OLS regression (or Logistic one in our case). On the contrary, the bigger lambda is, the stronger the coefficient's size is penalized.

3.6 Lasso regression

For Lasso, we also have a penalization term, but it takes a different form than for the ridge regression. Indeed, we consider the absolute value of the coefficients in the penalization term.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) + \frac{\lambda}{m}\sum_{j=1}^{n}|\theta_j|$$

Again, the hyperparameter λ needs to be chosen, it is the regularization penalty. The closest it is to 0, the lowest is the impact of the penalization, and then the regression will tend to a simple OLS regression. On the contrary, the bigger λ is, the stronger the coefficient's size is penalized. The main difference with Ridge is that here the coefficient can be equal to zero (while it cannot be the case for Ridge).

3.7 Elastic-Net

Elastic-Net is simply a model which mixes Lasso and Ridge penalization terms. It is using a combination of L1 norm from Lasso and L2 norm from Ridge, as shown in this diagram:

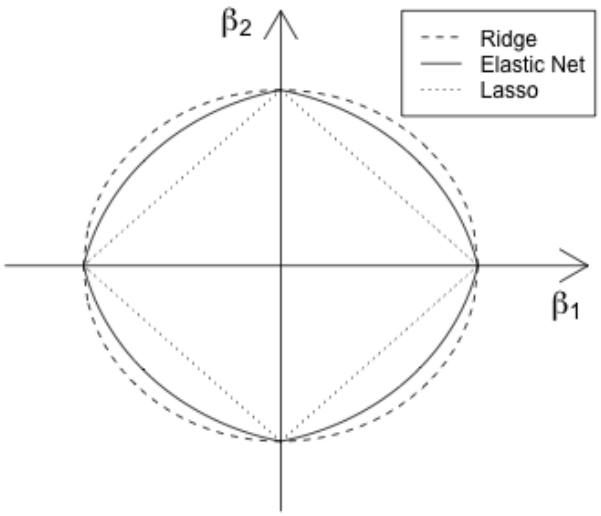


Figure 9: Elastic Net

There is the cost function with the two penalizations, λ_2 for Ridge and λ_1 for Lasso:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) + \frac{\lambda_2}{2m}\sum_{j=1}^{n}\theta_j^2 + \frac{\lambda_1}{m}\sum_{j=1}^{n}|\theta_j|$$

3.8 Support Vector Classifier (SVC)

The Support Vector Classifier, or SVC is a classification method which is quite easy in concept, the basic idea is to try and separate data into two groups by cutting through it by the means of a hyperplane.

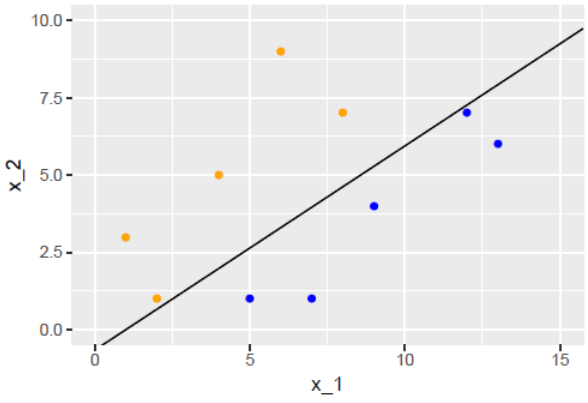


Figure 10: Splitting data with a hyperplane

In the above image we can see what a hyperplane cutting the data into two classes would look like in two dimensions. In three dimensions, it becomes a plane, and in higher dimensions it is more generally called a hyperplane. To find this hyperplane, we have to solve an optimization problem. In simple cases like the one above, there exists an infinite number of hyperplanes separating the data perfectly, what

we want is to find the one which maximizes the margin (represented by M in the formula below), i.e. the distance from each point to the hyperplane.

$$\max_{\beta_0,\beta_1,\ldots,\beta_p,M} M$$
$$\text{s.t. } \sum_{j=1}^{p} \beta_j^2 = 1$$
$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M, \forall i = 1, \ldots, n)$$

The constraints of the optimization problem ensure that the observations stay on the right side of the hyperplane, and that the distance is rightly measured.

In classification problems, like ours, it is sometimes not possible to perfectly separate the data. That is why there can be a tolerance of misclassifications allowed for the hyperplane, which slightly change the optimization problem.

Another possibility of separating the data best is to change the shape of the hyperplane, most commonly it is a linear hyperplane, but it is possible to change its shape by changing the associated kernel function.

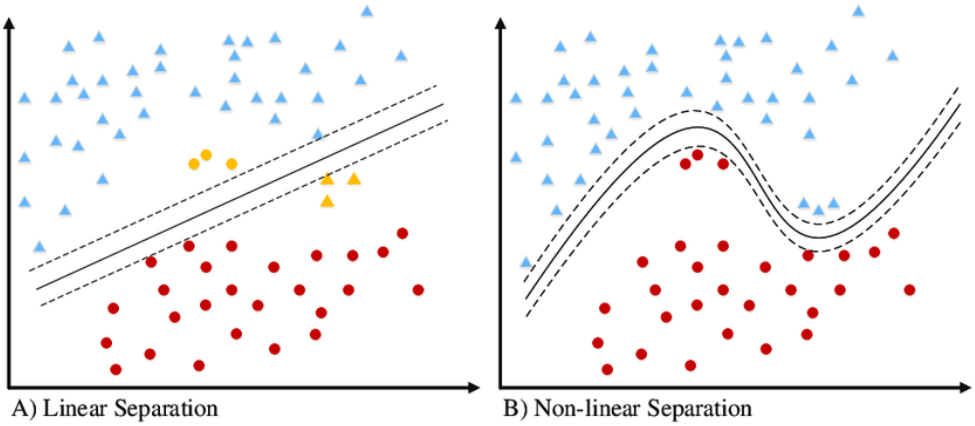


Figure 11: Different types of kernel

The nonlinear separation can allow for a better separation of the data, but it is costly in computation.

The Support Vector Classifier works well for binary classification problems, especially in higher dimensions, even with more features than observations. It becomes very time costly with larger datasets. This is an issue in our case, as with 300 000 observations it is not possible to test other types of kernel functions than the basic linear.

As in our case, only the SVC with linear hyperplane is feasible, we can only tune the hyperparameter C, which is a penalization term, that penalizes misclassifications depending on its value.

3.9 Stochastic Gradient Descent Classifier (SGD)

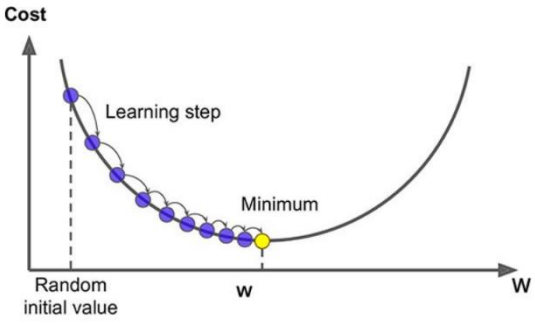
The Stochastic Gradient Descent is a classifier that adapts linear classification methods with gradient descent training.

The gradient is computed on the loss on each sample and is updated along the way. This classifier is a function that can combine different methods depending on the chosen loss function. It can be used as a logistic function, a linear support vector classifier.

Gradient descent is an optimization method which lets us find the minimum or the maximum of a function, in this case it is used to find the minimum of the loss function.

$$w := w - \eta \nabla Q(w) = w - \frac{\eta}{n} \sum_{i=1}^{n} \nabla Q_i(w),$$

The above is the way to find the parameter which minimizes the objective function. The main idea is to jump from point to point of the function and to compute the derivative in order to know where to go to find the minimum.



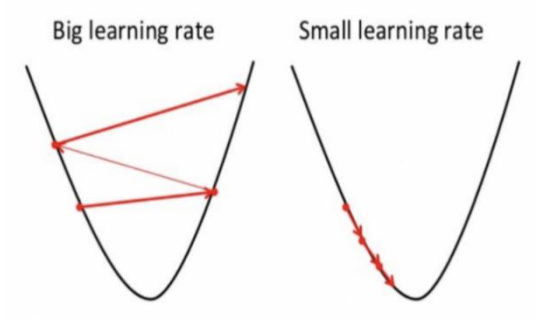


Figure 12: Gradient Descent

Figure 13: Learning rate

The illustration above shows how the gradient descent works, the jumps made between points are based on the learning rate parameter, which is the important hyperparameter for this model. If it is too small, it takes longer and is less efficient, but if it is too big it will jump too far and miss the minimum too often before finding it.

We use this model as an additional method, to try and compare it with the others, this is a fast model, and thus the tuning of the hyperparameters is quite easy and lets us choose a model fast.

3.10 Decision Tree Classifier

Decision Tree classification is a machine learning method for classification based on the idea of consecutively splitting the data into multiple sets on different consecutive conditions to rightly classify inputs into the right category. The simple concept and thus easy interpretability are the reason Decision Trees and their extensions into Random forests are some of the most used machine learning algorithm.

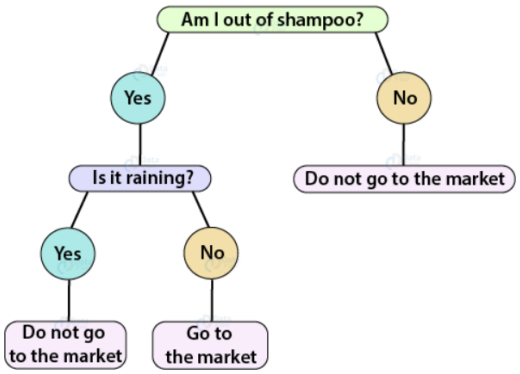


Figure 14: Simple decision tree

The above illustration is a simple yet valid example of a decision tree classifier, with multiple possible outputs ("Do not go to the market", "Go to the market"), and with splits at each node based on a condition ("Is it raining?"). This shows the easy interpretability of a Decision tree classifier.

The two key points to decision trees are how to split the data? And how to avoid overfitting?

To choose how to split the data, the algorithm looks at the level of impurity in each subset after a split at a node, based on one of possible indicators (Gini coefficient is the most used). This measures the part of different classes still present in each split and chooses the split which maximizes the loss of impurity. Thus, splits are made until each branch of the tree is free of impurities. This unfortunately can lead to some trees with too many branches, which equals to overfitting issues, where the model is too well fitted to the training data and will perform poorly on the out of sample data.

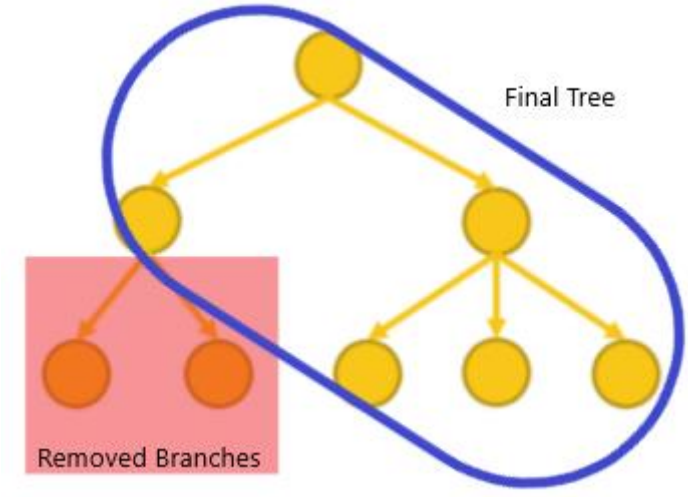


Figure 15: Pruning of a decision tree

The pruning process consists of removing branches which add little information to the final model. The choice of these branches is done by looking at how much they diminish the loss function, if the loss diminishes less than a given threshold value, then the concerned branch is considered redundant and is removed. The pruning process is a key part in getting a good model for out of sample predictions.

3.10.1 Cross validation

In machine learning models, it is a common way of find the accuracy of a model by using K-fold cross validation, which consists in splitting the data into K subsamples, and estimating the model K times, by using each time a different subsample as test sample and thus K-1 samples as training samples, this reduces overfitting issues.

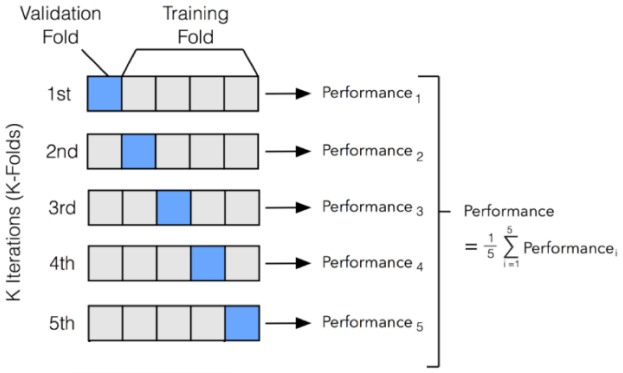


Figure 16: K Fold cross validation

The performance of the model is given by the mean of the subsamples' performances.

While this method gives good results, we are not able to use it on our models, we are just able to split our data into training and tests datasets at the beginning.

3.11 Random Forest

Random forests classification is a learning method that is based on classification trees: indeed, it builds an ensemble of decision trees that outputs the predicted class (as here, we are in a classification problem), using the mode of the class for each tree. As we have 2 classes to predict, if a majority of tree is predicting a given class, then our final prediction will be this class.

Each tree will firstly partition the feature space in a set of regions, that are built with a recursive binary splitting approach. For a given feature x_j, we select a cutting point that will be used to split our feature space into 2 regions.

To choose the assignment to a region, we use the classification error rate as follows:

$$\epsilon_j = 1 - \max_k(\hat{p}_{jk})$$

With, p^$_{jk}$: the proportion of training observations in the jth region that are from the kth class.

In practice for our algorithm, the measure used is the Gini Index. We prefer it to another measure which is the cross-entropy because the Gini index is more efficient in terms of computing power.

$$G = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

The Gini Index measures the total variance across the K classes. It is a measure of purity; we aim at minimizing the value. The more we have a predominant class in our region/node, the purest is this region/mode (Fig 17).

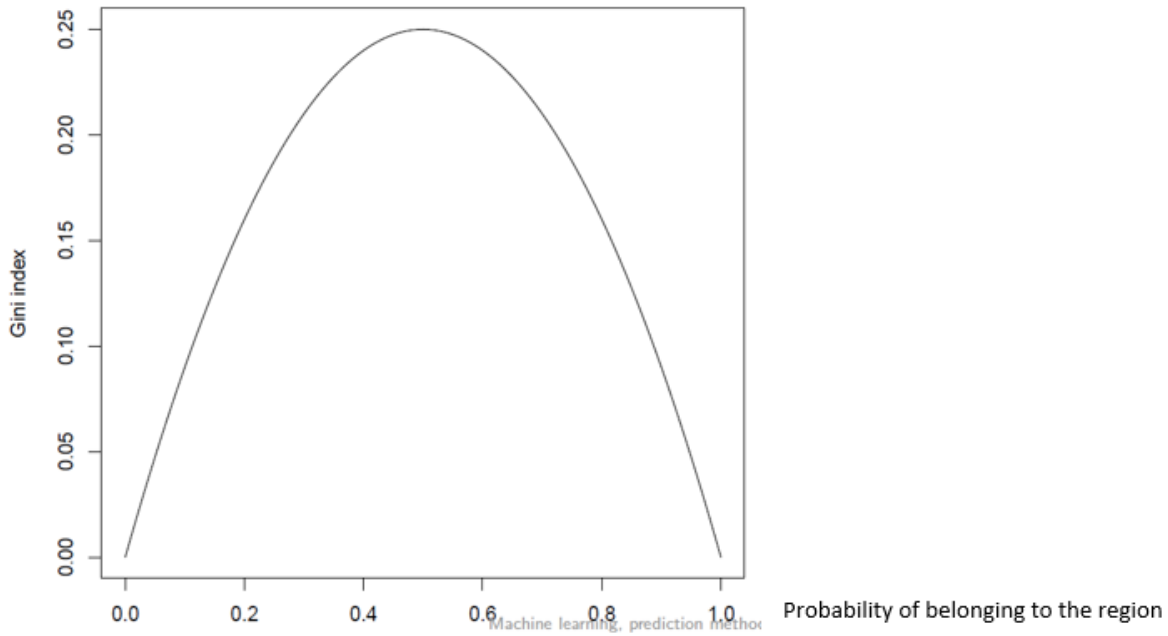


Figure 17 Gini Curve

3.12 K Nearest Neighbors (KNN)

KNN or K nearest neighbors' classification is a supervised machine learning method used for classification consisting in assigning a label to an individual with given features. The classification is done by taking all input features and finding the K nearest neighbors of the individual to classify. The nearest neighbors are commonly found by computing the Euclidean distance to the other training individuals in the feature space.

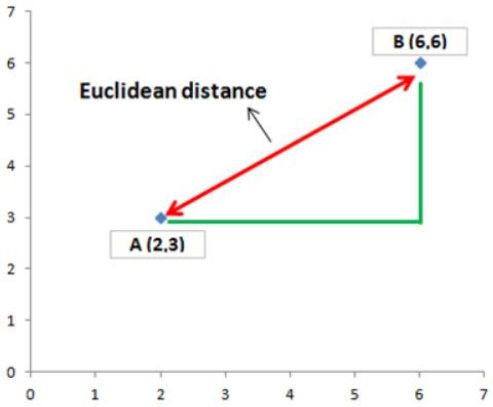


Figure 18: Euclidian Distance measure

Euclidian distance formula : $\sqrt{\sum(x-y)^2}$

The above formula is used to compute the Euclidean distance. The K closest individuals are then found by taking the K smallest values for the distance.

Once the neighbors are found, the target class is found by looking at the predominant class among the K neighbors.

In a KNN model, the most important parameter, which can influence the model's results greatly, is the choice of the K value, which will be the numbers of neighbors considered.

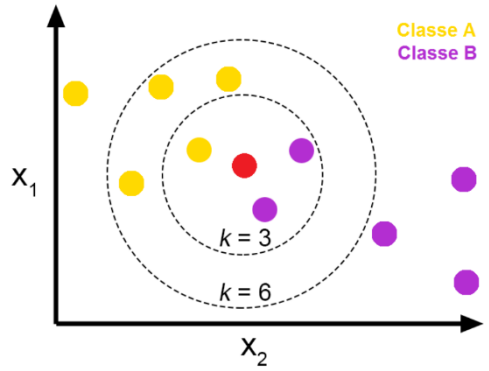


Figure 19: K neighbors with different K values

Indeed, in the above image we can see this illustrated, as with K = 3 or K = 6, the predicted result is not the same. Thus, this parameter must be chosen through trial of multiple values.

The other parameter, which is closely linked to the K parameter, is the choice of the distance. As this is how the K neighbors are found, the most known distance is the Euclidian distance, which was illustrated above. There exist other ways of computing this distance, which can be better suited to some types of data. When working with categorical data for example it is difficult to compute a significant Euclidian distance between groups if the values are not referring to ordinal categories.
In our dataset, which is a mix of non-ordinal categorical and continuous data, we try combinations of different K values and distance measures to get the best results. The distance measure which gives the best result for us is the Bray Curtis measure, which is most used in biology to quantify dissimilarities between sites. It is computed with the formula:

$$BC_d = \frac{\sum |x_i - x_j|}{\sum (x_i + x_j)}$$

The KNN method, like any other method has its pros and cons. The main advantages are that it is easy to implement because of no necessary strong assumptions, and it is also easy to interpret, while still giving good results on both classification and regression. The main disadvantage is that it becomes costly in computational power with bigger datasets, as it computes each distance, this is quite an issue in our case. The other problem can be that it is sensitive to outliers as all observations have the same impact.

3.13 Artificial neural networks (ANN)

The artificial neural networks (ANN) are a computing system inspired by the neural network from the brain, it is based on a collection of nodes. The connections between the nodes are called edges, and all those edges make the link between the layers, which are an aggregation of neurons. A neural network is then characterized by the number of layers, and how many neurons and nodes they are made up of.

In a neuron, we multiply each input by a weight (w), and we add a constant (b). Then, we apply a transformation to this number, the most often not linear, this transformation corresponding to the activation function of the neuron. We can choose the activation function we use, based on the type of problem we face.

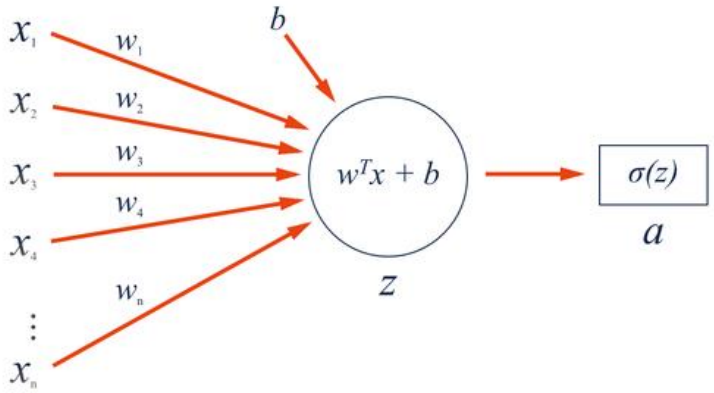


Figure 20: Simplified representation of the process of an input through neurons in a neural network

A layer can be fully connected when all neurons are connected to all the other inputs of the other layer, otherwise it is called convolutional. To sum things up, a classical neural network can be defined with an input layer (corresponding to data we feed it with), hidden layers that apply complex mathematical computations and transformations, and an output layer (Fig 21).

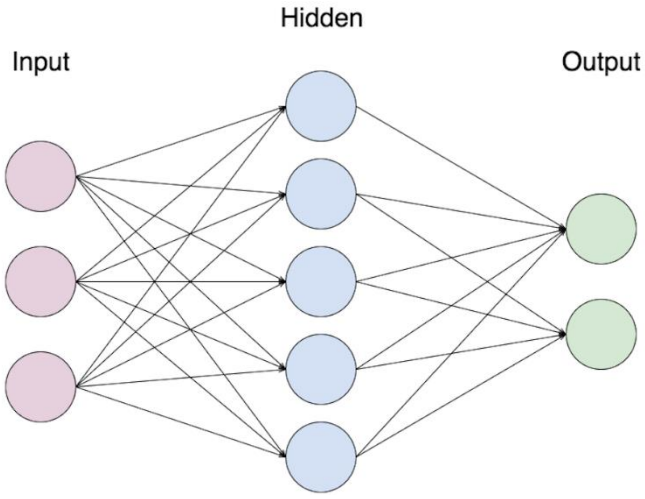


Figure 21: Simple representation of a neural network

We built different sequential neural networks before choosing one, which is the following.

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 97)	18915
dropout_4 (Dropout)	(None, 97)	0
dense_7 (Dense)	(None, 97)	9506
dropout_5 (Dropout)	(None, 97)	0
dense_8 (Dense)	(None, 1)	98

Total params: 28,519
Trainable params: 28,519
Non-trainable params: 0

Figure 22: Representation of the neural network from Python

Artificial Neural Network:

- Input layer: Fully connected layer using a ReLU activation function

- Dropout layer 1: This layer adds some randomness, because it activates only a part of the neurons, and those neurons are chosen randomly. The part of non-activated neurons is defined by the value of p (here 10% of the neurons are not activated), and those neurons are chosen randomly. It is used to reduce overfitting issues.

- Hidden layer: Fully connected layer using a ReLu activation function.

- Output layer: This output layer will return an output of size 1 (with a value of 0 or 1 corresponding to the prediction), using a sigmoid activation function.

The ReLu activation function:

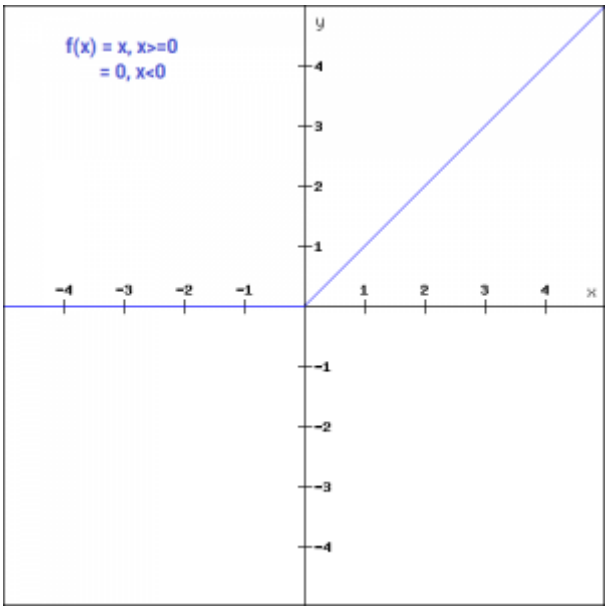


Figure 23: Graphic representation of the Rectified Linear Unit function (ReLu)

The ReLu activation function is a quite popular function, as it allows non-linear transformations and it prevents an exponential growth of computations required: indeed, all neurons are not activated at the same time when using this function. The disadvantage of this function is that as the weight of some neurons is not updated, this can lead to dead neurons that are never activated.

3.14 K Means

K-means is a clustering algorithm, its purpose is to discover patterns and group the data according to them into clusters. To do so, the algorithm follows these steps:

- Pick the number of clusters k
- Create k random centroids (representing the center of the cluster)

Then it repeats these steps until the centroids converge:

- Assign the data to its closest centroid based on Euclidean distance between them
- Calculate the mean for each cluster
- Update the centroid to be the center of the cluster

K-means only work for numeric data due to the method used (calculation of Euclidean distances). To choose the number of clusters, the elbow method can be use, but in our case, we know that we want 2 clusters (success and failed), so we will not use it.

K-modes algorithm was created to solve this issue, it is a clustering algorithm working with categorical data. Instead of working with Euclidean distance, the distance between 2 observations is based on dissimilarities (how many categorical variables they do not have in common), a low number of dissimilarities means that the observations are similar, so separated by a short distance.

In our case, we have almost as many numerical variables as categorical ones. K-Prototypes consist in a combination of the 2 algorithms, so it is good in our case and that is what we will use.

4 Results of the study

4.1 Exploratory Data Analysis

4.1.1 Visual representations and basic statistics

We measure some basic statistics that we display to get a better idea of how the data is built and distributed.

Firstly, there are 15 main categories for our projects.

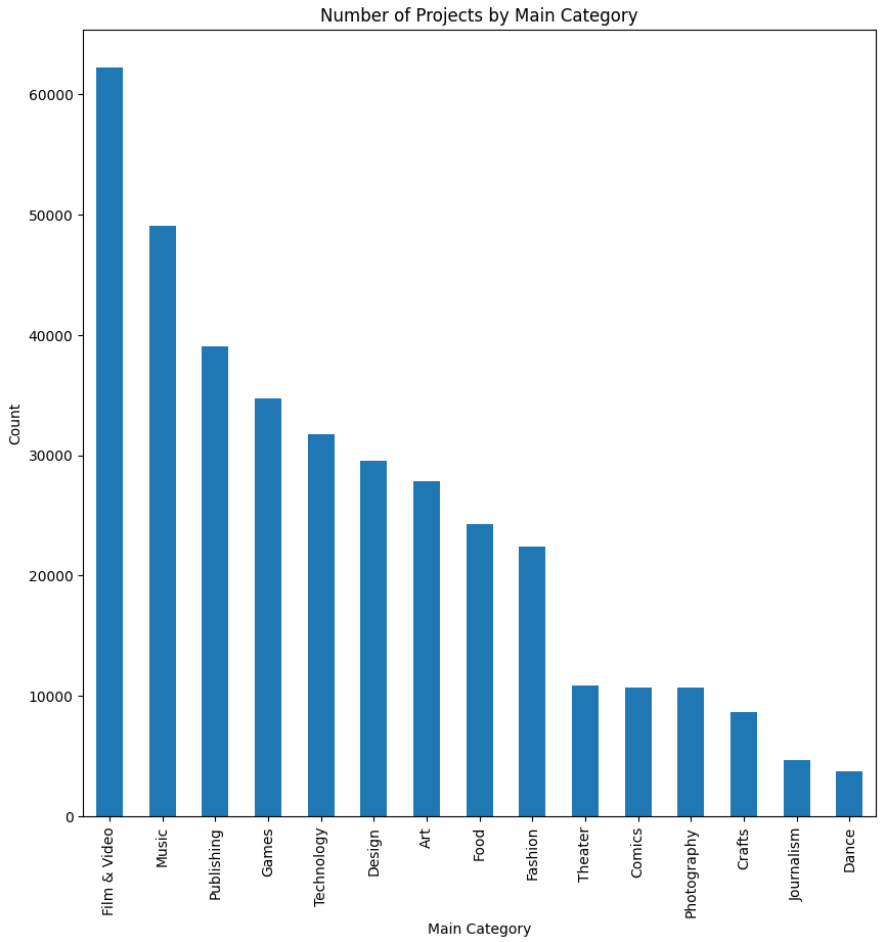


Figure 24: Number of projects by main category

We can clearly see that some categories are better represented than other, with much more projects launched, in artistic categories like Film&Video and Music.

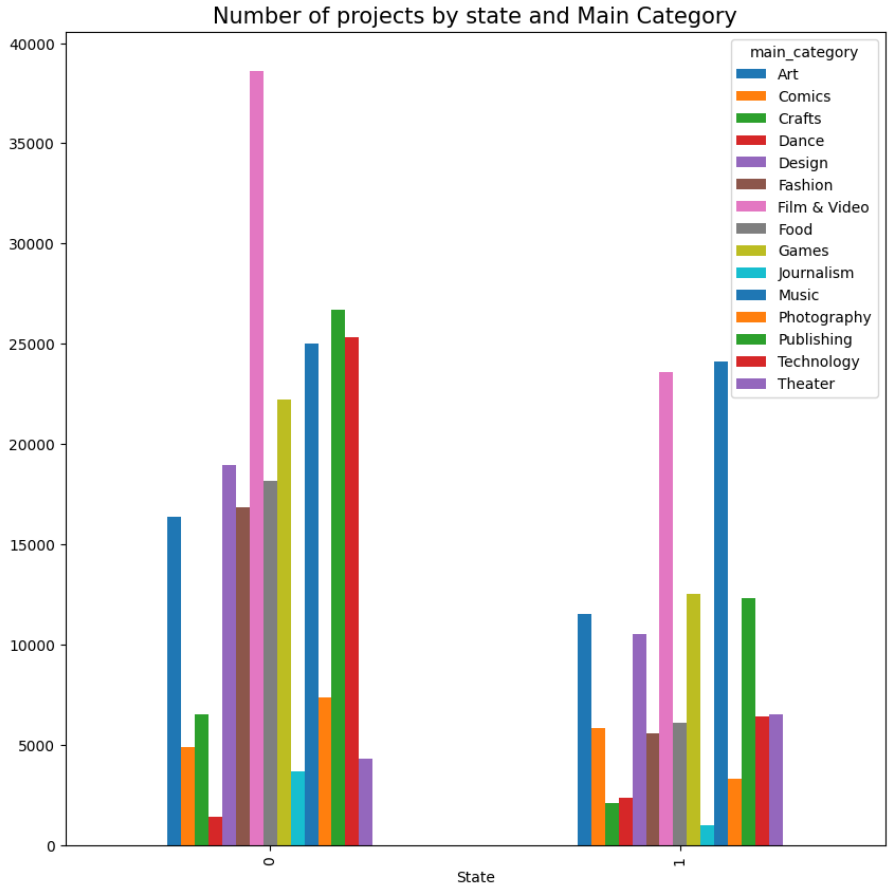


Figure 25: Number of projects by state and main category with 0=fail, 1=success

Also, some main categories seem to succeed better than other like “Crafts”, while “Art” seems to fail more than it succeeds.

Then, as we have 82 categories, we will only represent the 10 largest ones:

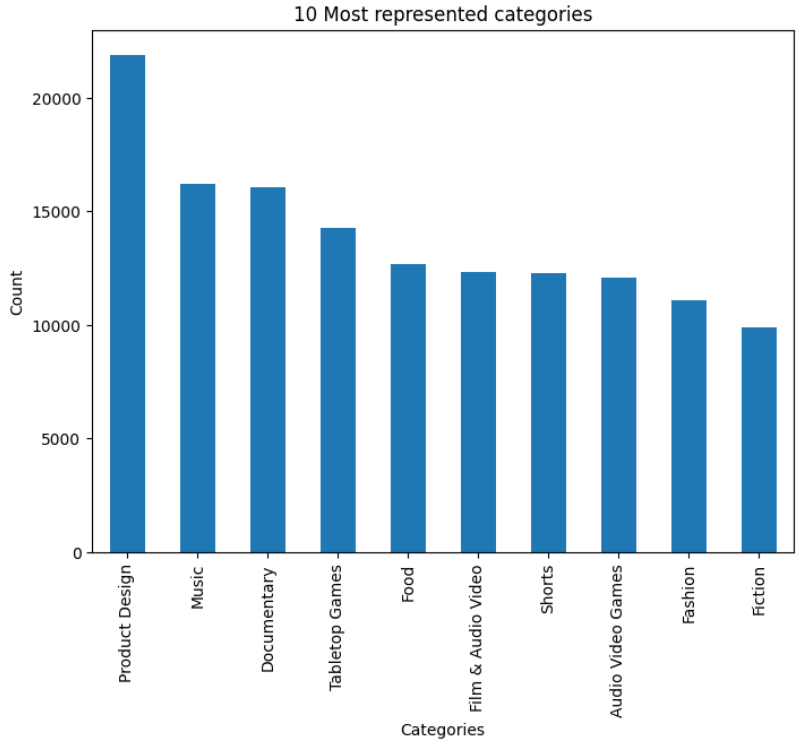


Figure 26: 10 most represented categories

We can see that the most represented categories represent more than 15 000 projects, while the smallest, after merging categories, are at least 1000 observations strong. To have a better insight of the link between the success of project and the amount of money needed to succeed it, depending on the category, we can rely on the following boxplots. To be able to clearly visualize the boxplots, we have removed the outliers from the graph.

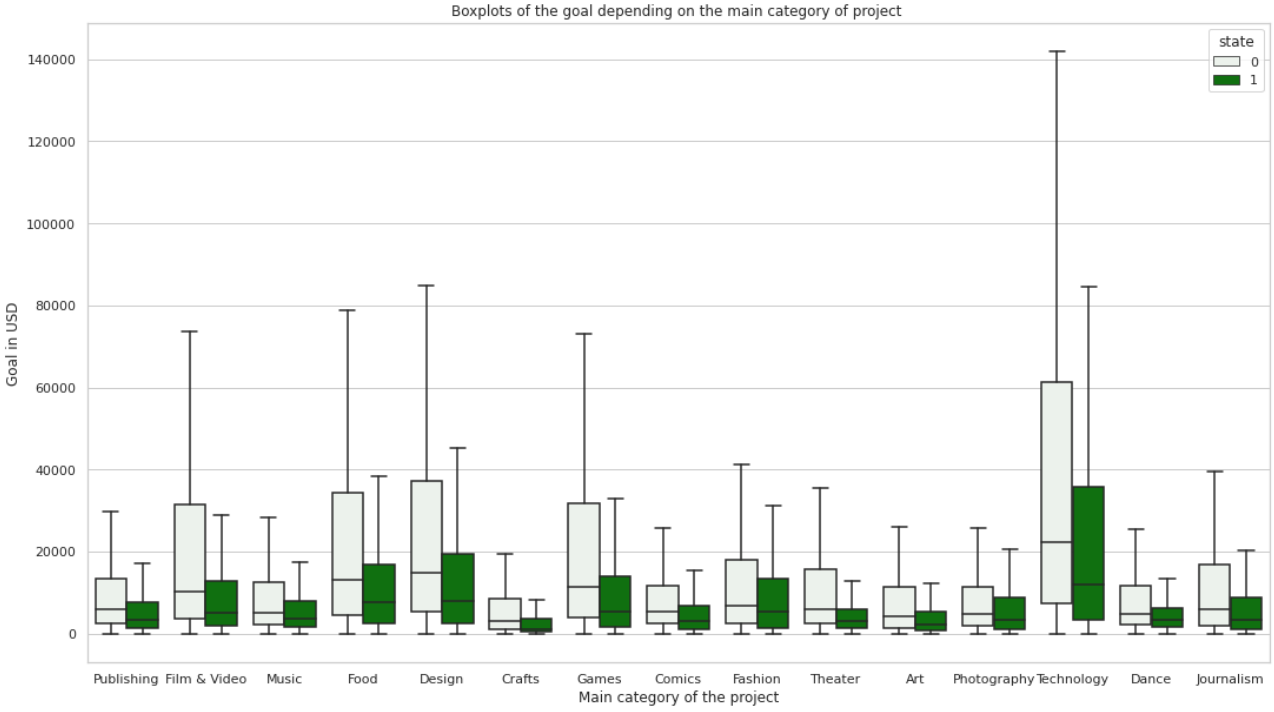


Figure 27 Boxplot of the goal and category of the project

There we can see that the amount of money asked is clearly dependent of the type of project: technological projects, for instance, require more money than the other type of projects. There is another very important information to get from this graph: in general, the failed projects were asking for more money than the successful projects. This statement is true for any category of project.

We can also look at some basic descriptive statistics on the continuous variables. We see that the amount of money needed (usd_goal_adj) is quite dispersed between the projects.

	usd_goal_adj	money_per_day_adj	len_name	count_7_days	time_last_proj
count	3.702130e+05	3.702130e+05	370213.000000	370213.000000	370213.000000
mean	5.004175e+04	1.608402e+03	34.839430	1097.111625	14.441616
std	1.287227e+06	8.651300e+04	15.942851	463.456023	122.010662
min	9.156439e-03	2.543455e-04	1.000000	0.000000	0.000000
25%	2.165666e+03	6.968687e+01	21.000000	857.000000	1.000000
50%	6.378794e+03	1.970090e+02	34.000000	1033.000000	4.000000
75%	1.795041e+04	5.478694e+02	49.000000	1310.000000	13.000000
max	1.699421e+08	4.329899e+07	96.000000	3980.000000	35376.000000

Figure 28: Descriptive stats on continuous variables

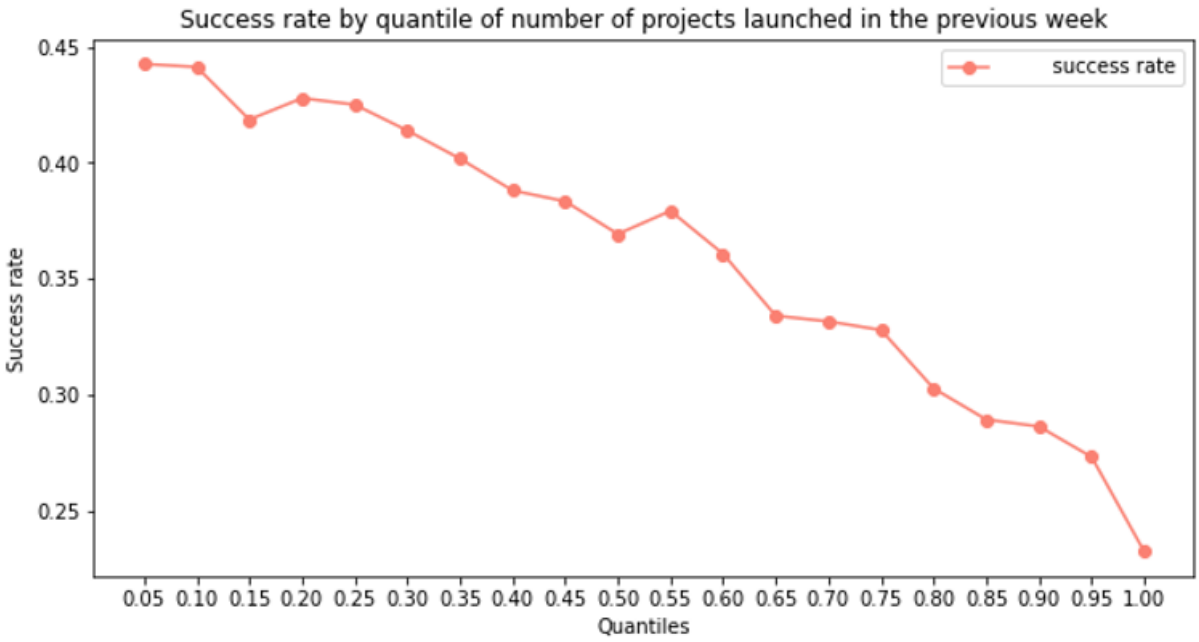


Figure 29 Relation between the the succes rate of a project and the number of projects launched in previous week

On the graph above, there seems to be a negative correlation between the success rate and the number of projects launched in the week before. This relationship sounds logical as the presence of too many projects will reduce the visibility of each project on the website.

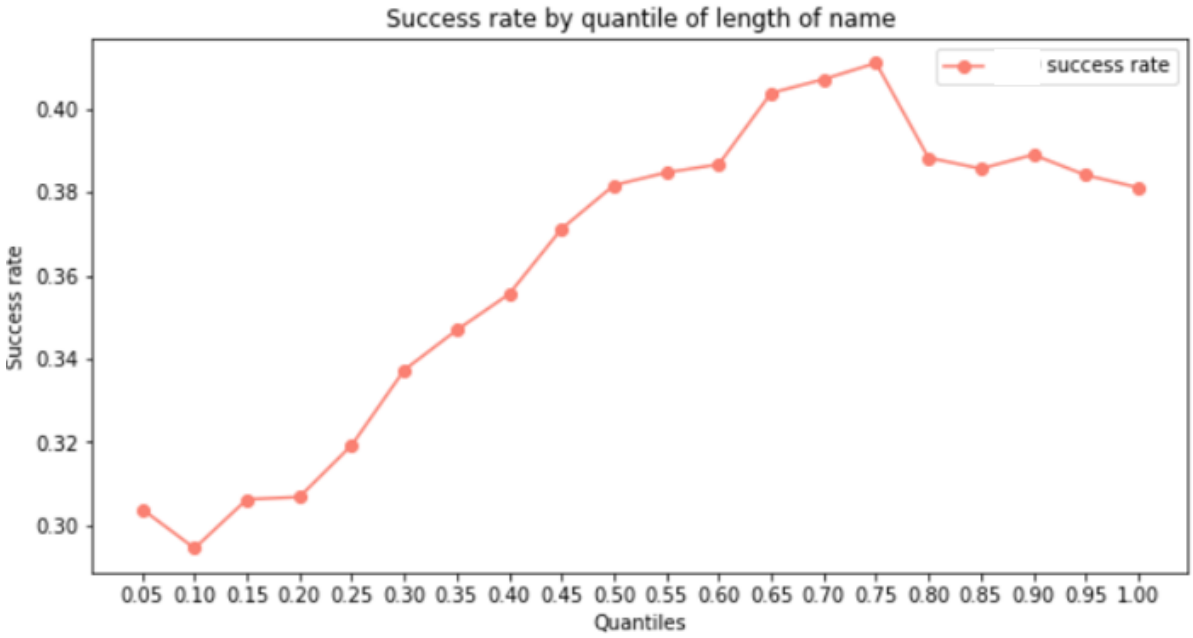


Figure 30 Relation between the success rate of a project and the length of the name

count	17910.000000
mean	47.997878
std	0.817269
min	47.000000
25%	47.000000
50%	48.000000
75%	49.000000
max	49.000000

Figure 31 Descriptive statistics of the length of the name of the projects

From the graph above, we clearly see a positive correlation between the length of the title of the project and the success rate: the maximum of the success seems to be reached for project with a length of a name around 50 characters.

4.1.2 Detecting multicollinearity with VIF

The Variance Inflation Factor helps to quantify the correlation between predictors in a model. The variance inflation for a variable is computed the following way:

$$VIF = \frac{1}{1 - R^2}$$

We want to measure the collinearity between the different predictors as it can be helpful to choose the model to use. Indeed, in the case of high collinearity, a simple logistic regression is not advised as the variance of the regression coefficient is inflated. The higher the value of the VIF, the bigger the correlation of the variable in question with the other variables.

	feature	VIF
0	launched_hour	1.018548
1	launched_year	0.000046
2	launched_month	1.006852
3	launched_day	1.017444
4	category_cat	1.038064
5	main_category_cat	1.039904
6	country_cat	35.976894
7	currency_cat	36.040404
8	usd_goal_adj	7.887604
9	money_per_day_adj	1.268296
10	number_of_day	1.021268
11	count_7_days	1.055962
12	len_name	1.009880
13	usd_goal_sq	7.310763
14	time_last_proj	1.000057

Figure 32: VIF results after running on our dataset

After running the VIF, we find some results:

- The features “country_cat” and ”currency_cat” have a very high value of VIF (around 35). This comes from the fact that the currency depends on the country where the project is launched, they must be correlated between themselves.
- There is also an identified correlation for the features “usd_goal_adj” and “usd_goal_sq”, as they have a VIF value of 7. The fact that we find correlation is completely normal, as one is just a transformation of the other.
- For the rest of the features, we can consider that they are not concerned by a collinearity issue.

To conclude, we detect presence of collinearity between some features in our dataset. This can be remediated by removing some variables, which gives a loss of information, or by implementing methods used in case of multicollinearity. The ridge regression, for instance, can be considered for this goal. A simple logistic regression without any penalty would not be appropriated.

4.2 Unsupervised methods result

4.2.1 K Means

We run K-Prototypes on a sample of our dataset, the reason behind that is that the dimensionality reduction technique we used (Uniform Manifold Approximation and Projection (UMAP)) to visualize our clusters in 2D needed more RAM than we have access to.

We obtain the following visualization:

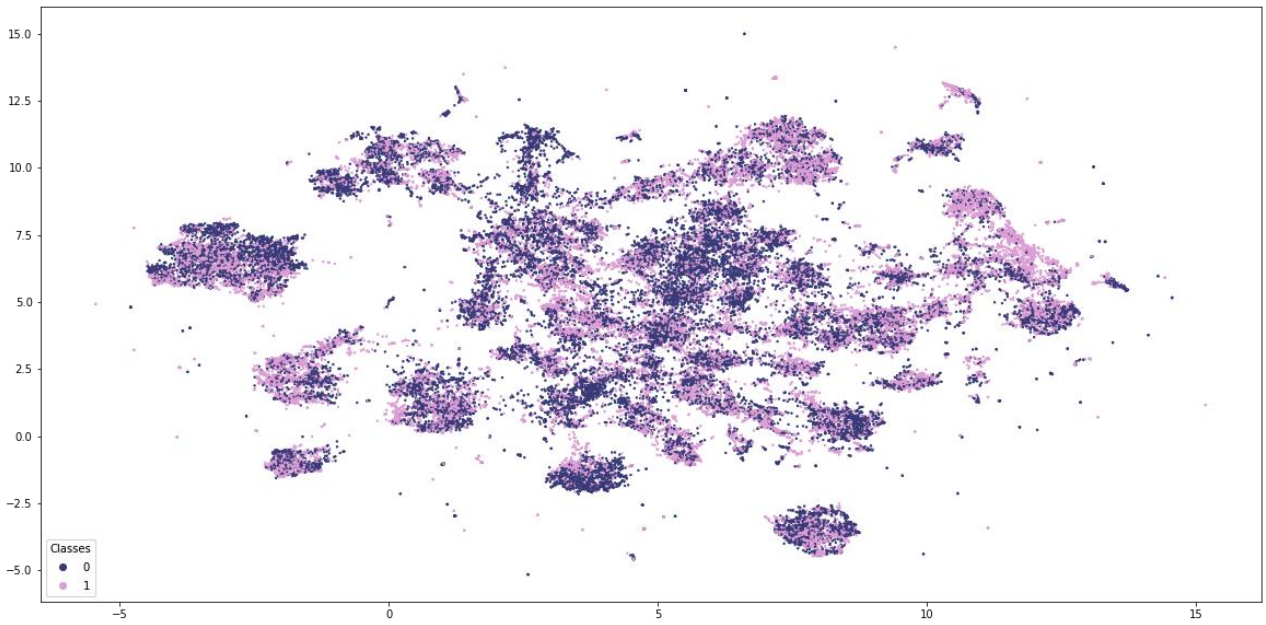


Figure 33: Clusters visualization using UMAP

Unfortunately, we cannot observe distinct clusters on it. This unsupervised technic does not give new insights on the dataset.

4.3 Supervised methods result

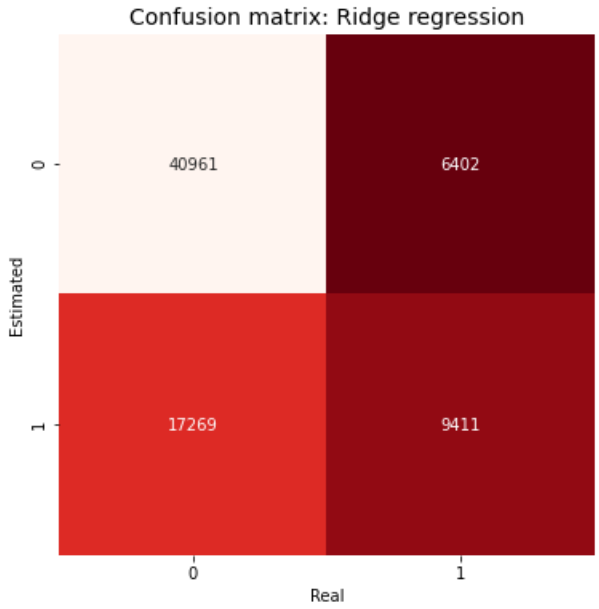
4.3.1 Logistic Regression

Logistic Regression from the Scikit Learn package uses L2 norm by default, so we decide not to include it without a penalty since it would not make much sense. Also, we detect multicollinearity between the features with VIF, in this specific case we know that Ridge and Lasso will give better results than a simple Logistic Regression.

4.3.2 Ridge Regression

Ridge Regression from the Scikit Learn Package corresponds to the default Logistic Regression, the other parameter being alpha, a high alpha implies a strong regularization, giving a high coefficient shrinkage, we test values from 0.01 to 1000 for it, but it does not translate to an impact on the final accuracy, which is 68%.

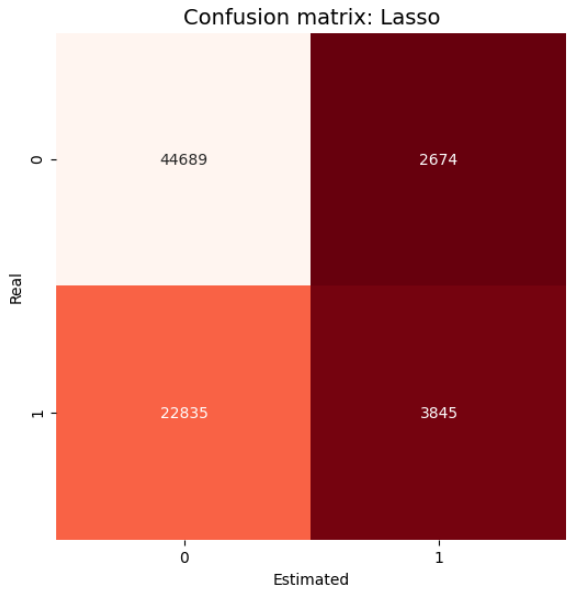
The confusion matrix we get is:



4.3.3 Lasso Regression

Lasso Regression from the Scikit Learn Package corresponds to the Logistic Regression with L1 norm, the parameters are the penalty, set to l1, and C (with λ = 1/C) a low C implies strong regularization, we test values from 0.001 to 10 for it. This time we get different accuracies based on lambda, indeed for C= 0.001, to many coefficients are shrunk to 0 which would result in a loss in accuracy of 4 percentage points. The best accuracy being 69%.

The confusion matrix for the optimal model is:

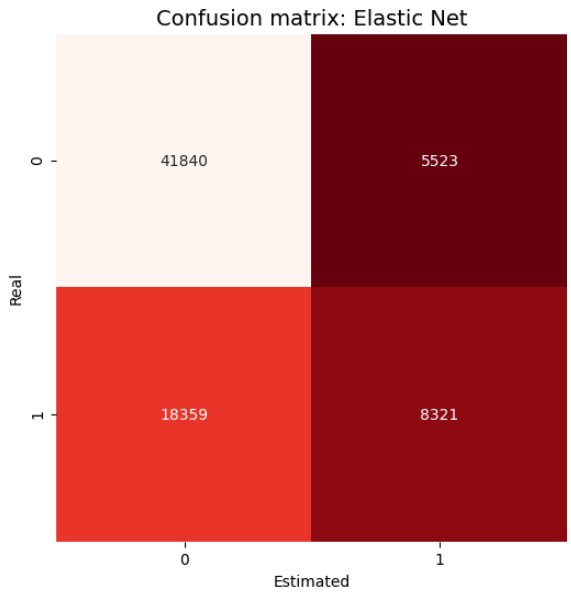


4.3.4 Elastic-Net

Elastic-Net from the Scikit Learn Package corresponds to the Logistic Regression with a mix of L1 and L2 norms, the parameters are the l1_ratio, setting l1_ratio=0 being equivalent to using penalty = L2 while l1_ratio = 1 is equivalent to using penalty = L1. For l1_ratio between 0 and 1 the penalty will be a combination of L1 and L2 norms. We test a range of values from 0.2 to 0.8 for this parameter.

At the end, the accuracy is very similar no matter what the l1_ratio value is (.01% of variation), with a value of 68%.

The confusion matrix we get is:



4.3.5 Support Vector Classifier (SVC)

For the Support Vector Classifier, we only use the Linear version from the Scikit Learn package. This way the only parameter which we can try to optimize is the C parameter, which is the part of misclassification allowed, a high value of C penalizes more the misclassified points. Thus, we try to see a large scope for its value, testing for values from 0.1, to 100 even.

The accuracy results we get:

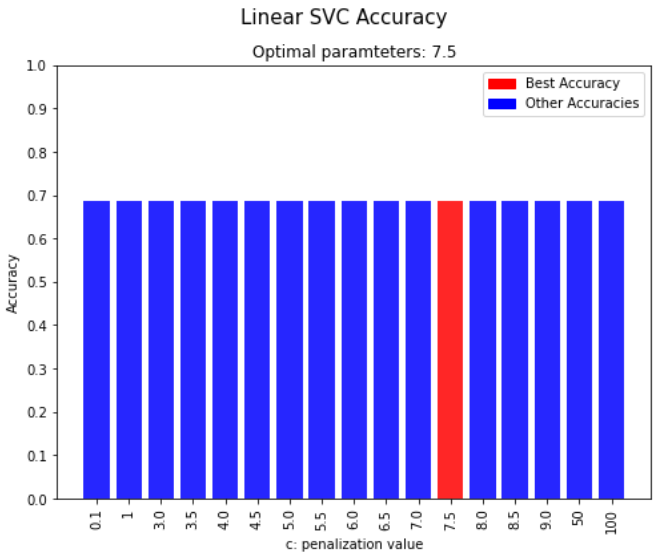


Figure 34: Linear SVC Accuracy

We get very similar results for different values of C, but the best is for C = 7.5. This has an accuracy score of 0.687.

We base the results score on the accuracy test gotten from the prediction done on the "X_train" part of the dataset. We also measure the AUC score, which is the area under the ROC curve, but as the results are in similar proportions to the accuracy results (it does not change which hyperparameters work best), we focus on accuracy.

The confusion matrix for the optimal model is:

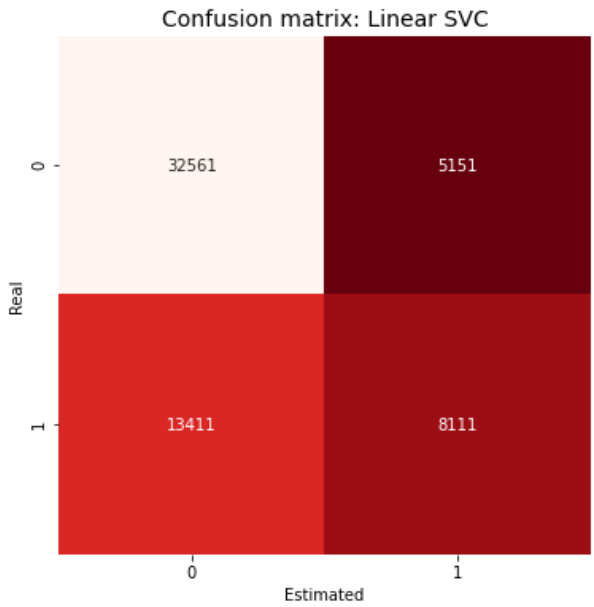


Figure 35: Confusion matrix: Linear SVC

We are not able to test SVC with other kernel functions, even on just one model, the runtime is too long.

4.3.6 Stochastic Gradient Descent Classifier (SGD)

The stochastic gradient descent classifier lets us tune several parameters, the type of loss function which would impact what kind of model it becomes (logistic regression, linear SVC, …), the alpha value and the type of learning rate which both have an impact on the latter. The accuracy results we get are:

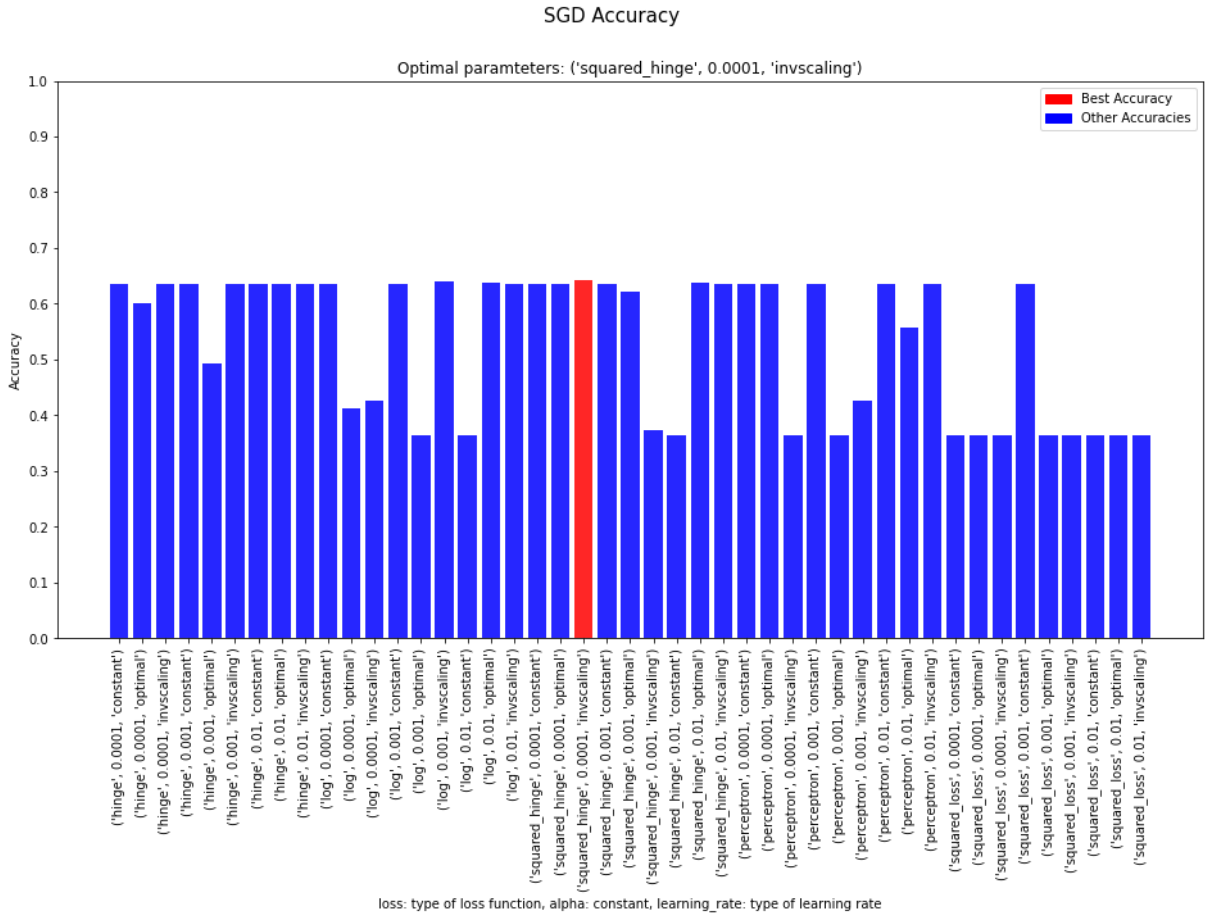


Figure 36: SGD Accuracy

The above plot shows the different accuracies, the best model is for a squared hinge loss function, which is like the loss function of a linear SVC but squared, with an alpha parameter of 0.0001 which is a constant multiplying the regularization term, and finally an inverse scaling learning rate which is given by the formula:

eta = eta0 / pow(t, power_t)

The best accuracy we get is 0.637, which is not the best.

The confusion matrix for the optimal model is:

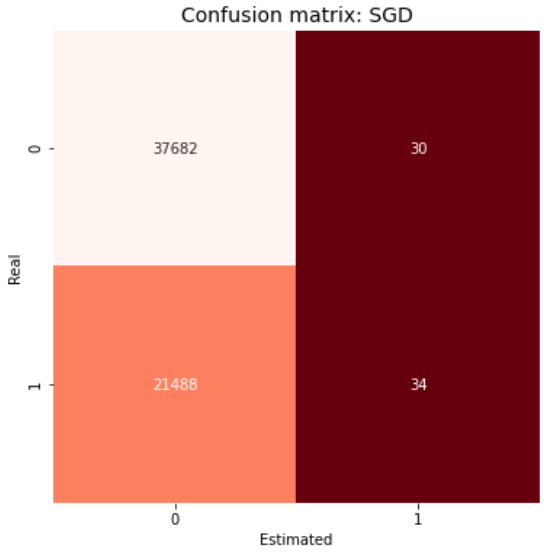


Figure 37: Confusion matrix: SGD

4.3.7 Decision Tree Classifier

For the decision tree classifier, we have 4 parameters to tune to get the best possible model:

- The criterion: either Gini or entropy, this is the criterion by which each split is chosen, if it is Gini then the split is made to reduce impurities, if it is entropy it is based on the gain of information through each split.
- The minimal samples per leaf: how many samples minimum are left on each end of branch after all the splits are done, we test values from 3 to 7 for this.
- The number of features for a split: this looks at how many features are considered at each node for a split, we test different values: the logarithm or the square root of the number of features, and an auto parameter which chose.
- The maximum number of nodes: this gives a maximum number of splits in total to limit overfitting, we test values 10, 100 and No limit.
- The minimum of impurity to decrease at each split: this sets a minimum for the criterion at each split, we test 0.2 and 0.4.

We test the 162 combinations, so it is difficult to show them all in one graph, we only plot the ones around the best combinations:

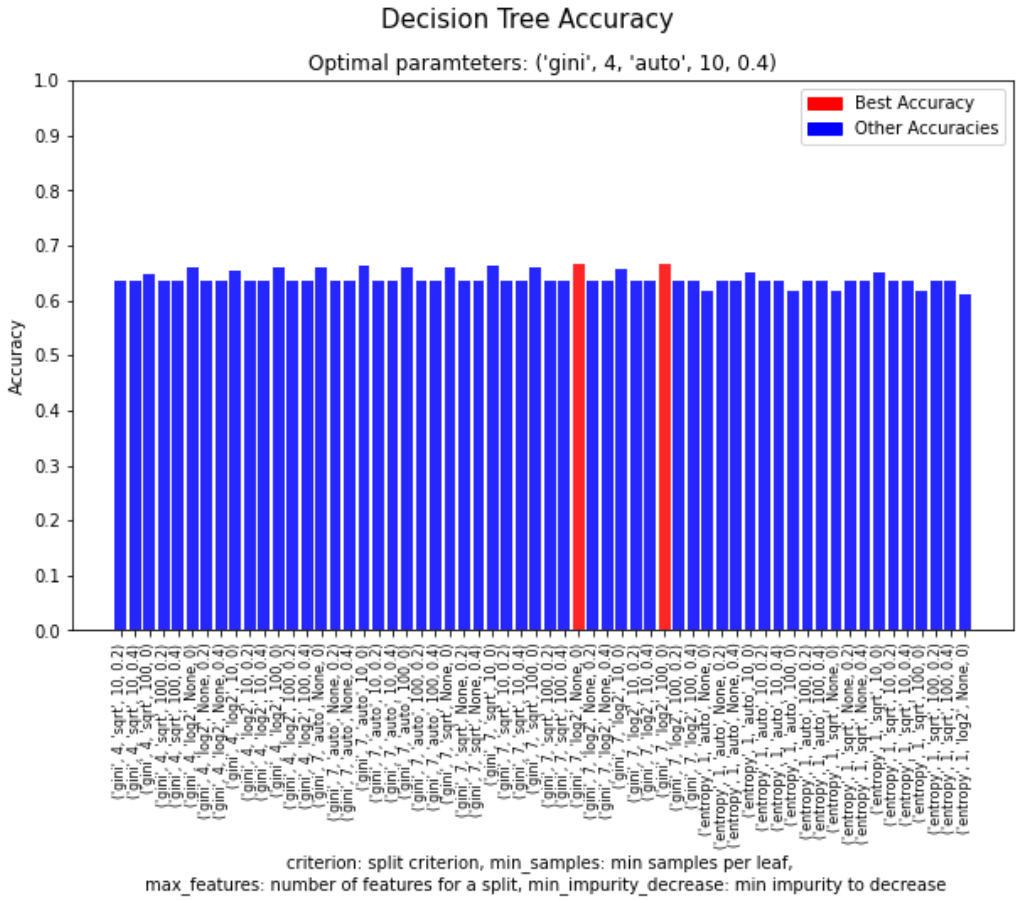


Figure 38: Decision Tree Accuracy

There are two possible combinations for the model giving the best accuracy, we choose to keep just one of them with parameters = ('gini', 4, 'auto', 10, 0.4), which gives an accuracy of 0.6572.

The confusion matrix for this model is:

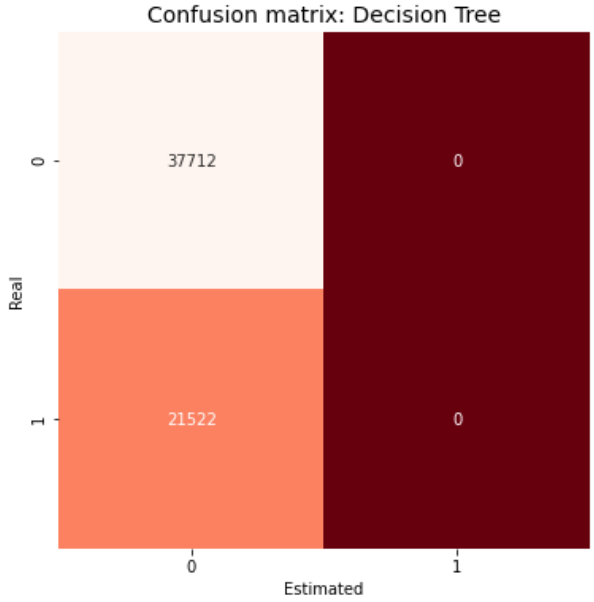


Figure 39: Confusion matrix: Decision Tree

This model is clearly not exploitable as it seems to only predict failed projects.

4.3.8 Random Forest

For the random forest, the computational cost is more important than for a simple decision tree classifier. Thus, we choose to tune just 3 parameters to get the best possible model:

- The number of trees per forest, we test the values 50, 150,250,350.
- The minimal samples per leaf: how many samples minimum are left on each end of branch after all the splits are done, we test 1,2,3,4.
- The maximum number of nodes: this gives a maximum number of splits in total to limit overfitting, we test 10,20,30,40.

The criterion we keep is the Gini criterion as it is faster for computations, and the maximum of features is fixed on automatic selection. This represents a total of 64 unique triplets of hyperparameters. To choose the best triplet, we look at the global accuracy on the test set. We find that the optimal hyperparameters are: 150 trees, a maximum number of nodes of 40 and a minimal sample per leaf of 2.

With this final optimal model, we can measure the feature importance to see which features are the most decisive in the random forest. This gives us an insight on what characteristics of a project are the most deterministic for its success:

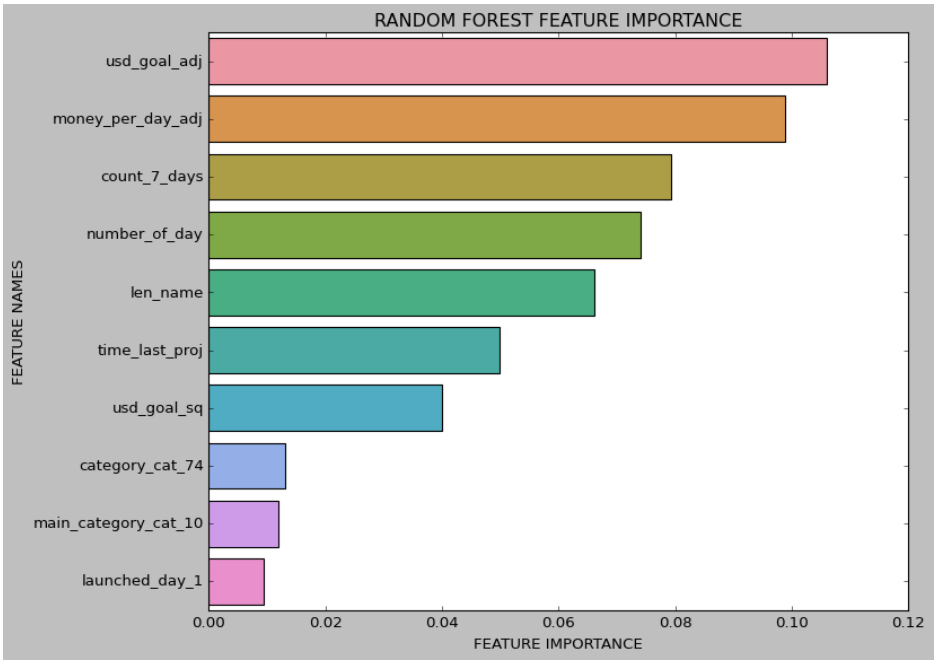


Figure 40: Random forest, feature importance

We can see that the amount of money needed in total, and per day of availability of the project are the 2 most important features. Besides, most of the variables we have created seem to be important for the classification, even the length of the name.

Finally, as we use one hot encoding, some specific categories appear in this graph: the main category number 10 (Videobooks) and the category number 74 (Tabletop games).

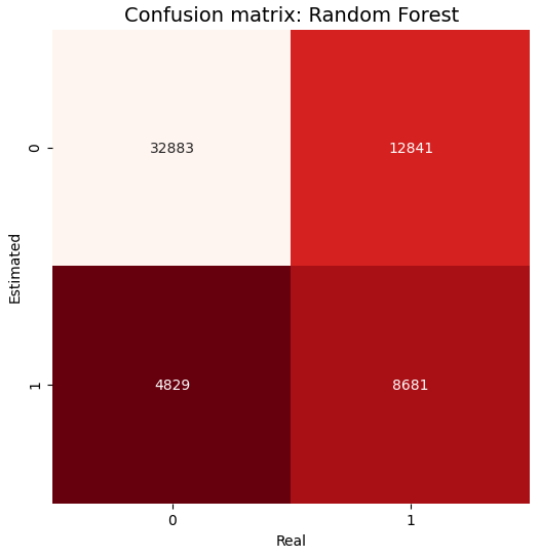


Figure 41: Confusion matrix: Random forest

4.3.9 K Nearest Neighbors (KNN)

We apply the KNN method classifier to our dataset to try and find if we will get a good model. This model is one of the most difficult to adapt, but only because of the high cost in runtime, which makes it nearly impossible to test as many combinations of hyperparameters as we would like.

We are able to optimize with only 2 hyperparameters: K and distance, which are the most important, but the SciKit Learn package we use lets us tune other ones like the “leaf_size”, which allows us to control the speed of construction and thus the accuracy, but we have to set it to a high number to get an acceptable runtime (it is comparable to the learning rate of the gradient descent).

After a few tests, we find that the optimal K is between 20 and 50. And we also use measures of distance recommended for work with both continuous and categorical data (we use “hamming”, “canberra”, and “braycurtis”).

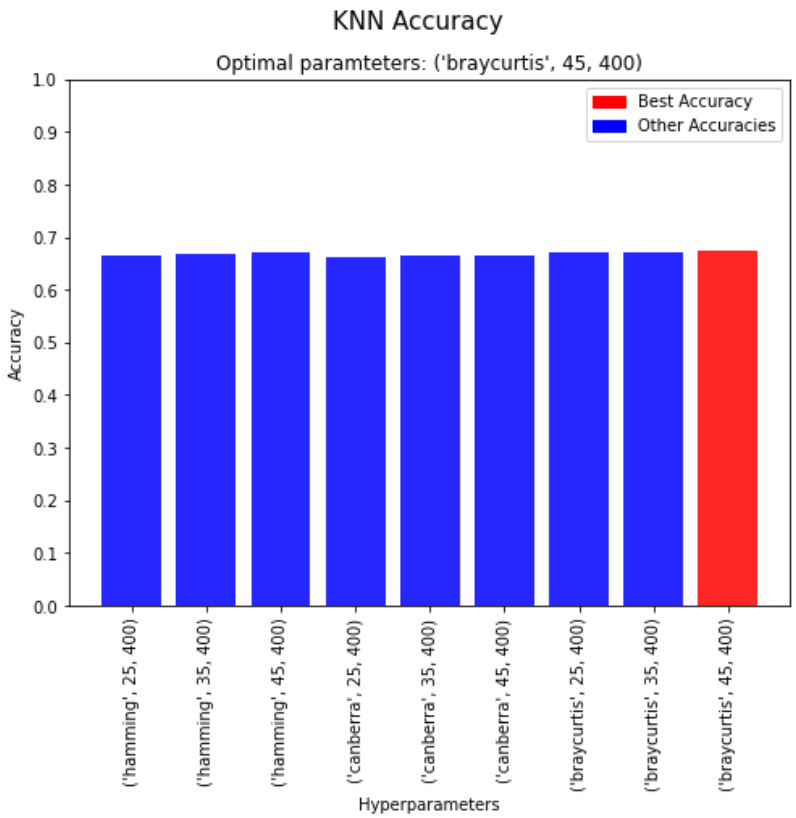


Figure 42: KNN Accuracy

We look at accuracy to compare the models, we will in the end use the confusion matrices to look at which model will be best, for different situations.

The best KNN model scores an accuracy of 0.675.

Here we can see that the best KNN model is gotten with bray Curtis distance which was detailed earlier above, and with K = 45. The high value of K can be explained by the size of the dataset, with very low values of K the model scored poorly.

We compute the confusion matrix of the optimal model:

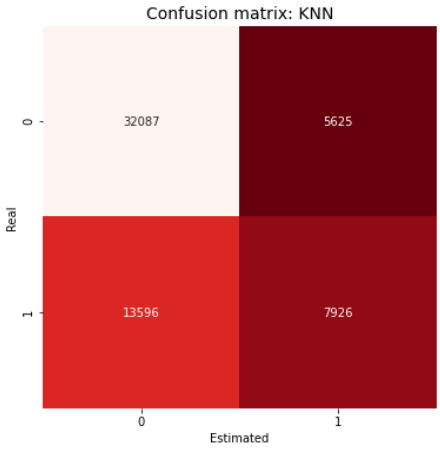


Figure 43: Confusion matrix: KNN

KNN is a very long model to run, this is one of its disadvantages.

4.3.10 Artificial neural networks (ANN)

Once we have fixed the number and the characteristics of the layers, the choice of the hyperparameters has been made on the global accuracy of the model. There are the parameters we use:

- The activation function is the ReLu activation function.
- The metrics is the accuracy: it means that the output we try to maximize through our epochs is the global accuracy of the model.
- The optimizer is rmsprop, for root mean square prop. It uses a concept close to the exponentially weighted average of gradient as gradient descent with momentum, but the difference is parameter update. The main difference with a gradient descent is the way of computation of the gradients, it is supposed to increase the learning and converge faster.
- The output activation function is the sigmoid function. Indeed, it is a good candidate to map the value to a number between 0 and 1, corresponding to a probability. We tried using the softmax function, which is also appropriated, but it was not concluding.

The epochs correspond to the number of times the algorithms work with the entire dataset. We fix the number of epochs to fit our model to 100, with callbacks. To avoid unnecessary computations and overfitting issues, we implement two safeguards from Keras:

- Early Stopping: it allows us to test a high number of epochs, and the model stops iterating at a certain epoch if its performance (here measured by the loss) stops improving.
- Model Checkpoint: we save and make a checkpoint of the current model once the condition above is satisfied, to keep the “best” model in memory.

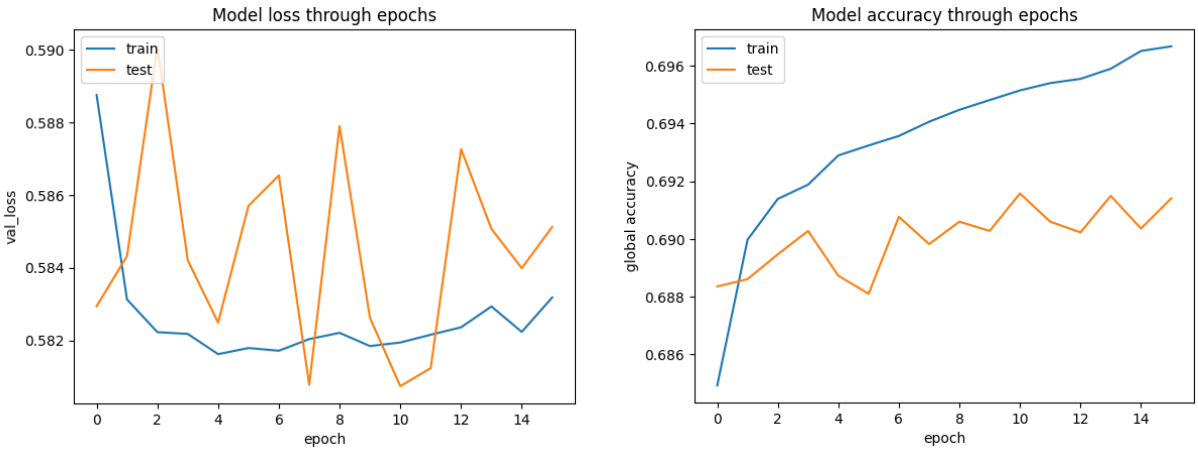


Figure 44: Model accuracy and loss evolution through epochs

We base ourselves on the val_loss (the value of the loss function in the test set) because a smaller cost function means a more precise model. Besides, we measure the loss on the test set and not on the train set to avoid overfitting. From the safeguards we implemented, the model is trained for a maximum of 16 epochs, and the optimal model is for 11 epochs (epoch number 10 on the figure above).

From the model accuracy, we clearly see the overfitting induced by the increasing number of epochs: indeed, the accuracy does not stop increasing on the train set, while it is quite stagnant on the test set.

Finally, the model chosen has respectively for the train and test dataset, an accuracy of 69,54% and 69,16%, and a loss of 58,08% and 58,07%.

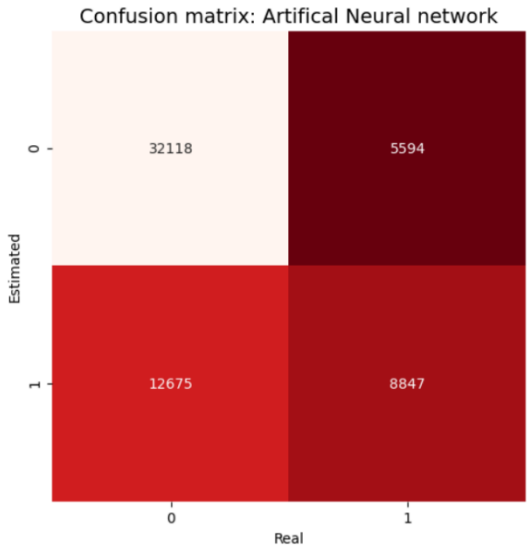


Figure 45: Confusion matrix, ANN

4.4 Model chosen

We train our models on the same dataset, which is X_train and y_train, to select the right hyperparameters we compute scores on predictions made on the test samples X_test and y_test. We base these choices of hyperparameters on the accuracy scores, as after first computing the AUC scores we find that the results are very similar to the accuracy scores. Thus, we select the best model from each type of method with the selected sets of hyperparameters.

When we first split the dataset, we keep 20% of the values representing 74043 observations in a sub sample called X_test_model and y_test_model. This is done to test each different model on a new sample and pick the very best one. To pick the best model here, we first compute each accuracy score, and each confusion matrix for each model.

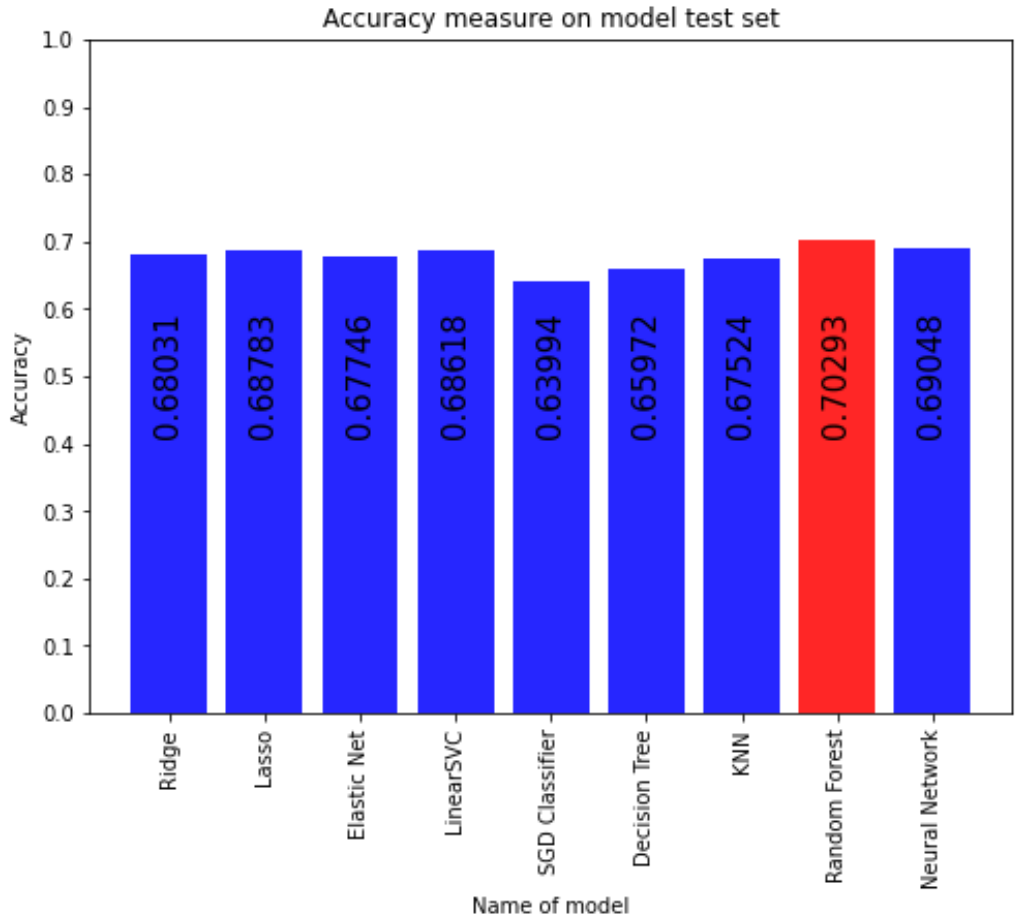


Figure 46: Accuracy measure on model test set

As we can see, one model is at 70% accuracy, the random forest model. This is in line with general results, as random forests are very frequently used models for classification problems. The second-best model from the accuracy score is the Neural Network, this is also in line with general results.

We can note that all models are in the same range of accuracy, this is because our original dataset is made up of approximately 60% of failed (=0) and 40% of success (=1), thus models that are not performing well, and only predicting 0, the accuracy will still be around 60%.

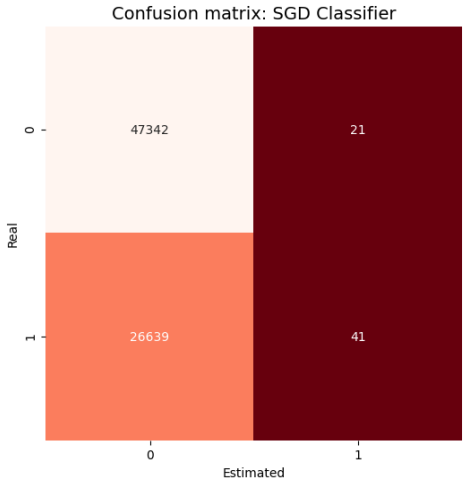


Figure 47: Confusion matrix: SGD Classifier

The Stochastic Gradient Descent classifier for example predicts almost exclusively 0 and has an accuracy score of 64%.

We will look at the confusion matrix of the two best models to see what they predict:

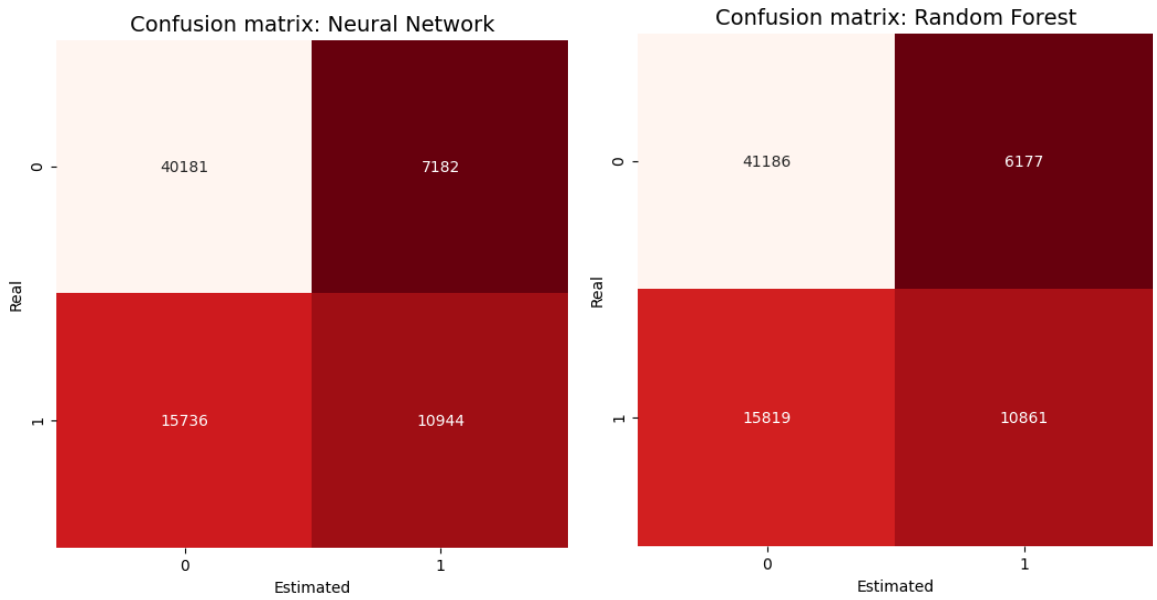


Figure 48: Confusion matrix: Neural Network

Figure 49: Confusion matrix: Random Forest

As we can see, both models predict very similar results. Both show almost the same number of True positives (10944 for ANN, and 10861 for Random Forest), but where random forests seem to be better is in predicting True negatives (40181 for ANN, 41186 for Random Forest).

Our results indicate that Random Forest is the model to use to predict the success or the failure of a Kickstarter project. Another point that can be in favor of Random Forest is its interpretability, as it is one of the most interpretable models used, as we can see with the feature importance it is directly possible to see what feature has the most impact, whereas an Artificial Neural Network is more or less a black box model, where it is difficult to understand what happens with each feature until the result.

Thus, the clearest and best model to predict the success of a project is the Random Forest, which can help someone in need of financing to better adjust his project, in terms of both money and time. It might also help to understand how to highlight one's project, maybe by trying to highlight its closeness to a successful category.

On Kickstarter, as there is no financial return on investment for the donors, and as the money is only given if the goal is reached, the interest for a donor to know if the project will be successful is not as important. But on other crowdfunding platforms, where there might be a financial return depending on the success of a project, and no all-or-nothing policy, it can become very important for a potential donor to know if he is giving to a doomed project.

Our model is based on Kickstarter data, but it is not only applicable on this website, as it does not necessarily consider features based solely on Kickstarter.

5 Conclusion

This project allowed us to build a predictive model based on Random Forest, with a global accuracy of 70% for predicting the success or not of a Kickstarter project before its launch. Other models such as Neural Networks, Ridge or Lasso regression also performed well to predict the output of a project. We introduced new pertinent features such as the length of the name of a project, or the number of Kickstarter projects launched in the same period, that have brought some more information than what was initially present in the database. From an economic point of view, such a study can be useful for a creator to look at its chances of success based on the type of project, but also to be able to adapt it (reducing the amount of money needed, not launching it when many projects are launched…) to maximize its probability to succeed.

Some extensions of this analysis could be lead, either by adding more explanatory variables by capturing the effect of the social network of the creator of the project, as it was done in the literature, or by changing the binarity of the prediction into the prediction of how much money will be pledged.

https://www.youtube.com/watch?v=dQw4w9WgXcQ

6 Sources

6.1 Bibliography:

Bramoullé Y., Ductor L. (2018): “Title length”. Journal of Economic Behavior & Organization, Volume 150, Pages 311-324.

Kindler A., Golosovsky M., Solomon S. (2019): “Early prediction of the outcome of Kickstarter campaigns: is the success due to virality?”. Palgrave Commun 5, 49.

Mollick E. (2013) “The dynamics of crowdfunding: An exploratory study”. Journal of Business Venturing.

Etter V., Grossglauser M., Thiran P. (2013) “Launch Hard or Go Home! Predicting the Success of Kickstarter Campaigns”. Proceedings of the first ACM conference on Online social networks, 177-182.

Sawhney K., Caelin T., Ramon T. (2016) “Using Language to Predict Kickstarter Success”. Paper. Stanford University.

6.2 Webography

https://builtin.com/data-science/when-and-why-standardize-your-data

https://scikit-learn.org/stable/

https://towardsdatascience.com/

https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/

https://antonsruberts.github.io/kproto-audience/

6.3 Images

Figure 1: Example of a project

https://www.kickstarter.com/projects/livestockproductions/stay-tooned-presents-btas?ref=discovery&term=toon

Figure 4: Illustration of the splitting procedure of the dataset

https://harvard-iacs.github.io/2017-CS109A/labs/lab4/notebook/

Figure 5: Example of One Hot Encoding with dummy variables

http://astuces-boulot.over-blog.com/2019/03/one-hot-encoding-pour-transformer-les-categoriels-en-python.html

Figure 10: Splitting data with a hyperplane

Machine learning and Statistical Learning, E. GALLIC

Figure 11: Different types of kernel

https://www.researchgate.net/figure/This-figure-shows-the-linear-and-non-linear-SVM-for-2D-dataset-for-text-data-we-have_fig11_332494043

Figure 12: Gradient Descent

https://morioh.com/p/b0fefc78f330

Figure 13: Learning rate

https://morioh.com/p/b0fefc78f330

Figure 14: Simple decision tree

https://data-flair.training/blogs/r-decision-trees/

Figure 16: K Fold cross validation

https://ethen8181.github.io/machine-learning/model_selection/model_selection.html

Erreur ! Source du renvoi introuvable.

Machine learning, prediction methods and applications, P.MICHEL

Figure 18: Euclidian Distance measure

https://towardsdatascience.com/k-nearest-neighbors-knn-explained-cbc31849a7e3

Figure 19: K neighbors with different K

https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d

Figure 20: Simplified representation of the process of an input through neurons in a neural network

https://www.innoarchitech.com/blog/artificial-intelligence-deep-learning-neural-networks-explained

Figure 50: Simple representation of a neural network

https://medium.com/@jamesdacombe/an-introduction-to-artificial-neural-networks-with-example-ad459bb6941b

Figure 23: Graphic representation of the Rectified Linear Unit function (ReLu)

https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

7 Appendix

7.1 Merged categories

Initial category	New category
Video Art	Digital Art
Textiles	Art
Ceramics	Sculpture
Installations	Sculpture
Events	Comics
Anthologies	Comics
Webcomics	Graphic Novels
Taxidermy	Crafts
Letterpress	DIY
Quilts	Crafts
Weaving	Crafts
Pottery	Crafts
Embroidery	Crafts
Glass	Crafts
Crochet	DIY
Knitting	DIY
Stationery	Crafts
Printing	Crafts
Candles	DIY
Residencies	Performances
Workshops	Performances
Spaces	Performances
Typography	Graphic Design
Civic Design	Design
Interactive Design	Design
Architecture	Graphic Design
Pet Fashion	Fashion
Couture	Fashion
Childrenswear	Fashion
Ready-to-wear	Fashion
Footwear	Fashion
Romance	Film & Video
Movie Theaters	Film & Video
Festivals	Film & Video
Family	Film & Video
Fantasy	Action
Experimental	Film & Video
Music Videos	Film & Video
Thrillers	Drama
Science Fiction	Action
Bacon	Food
Community Gardens	Farms

Farmer's Markets	Farms
Spaces	Farms
Cookbooks	VeganFood
Vegan	Food
Events	Restaurants
Puzzles	Tabletop Games
Gaming Hardware	Video Games
Photo	Audio Video
Audio	Audio Video
Video	Audio Video
Print	Audio Video
Comedy	Music
Chiptune	Music
Latin	Music
Blues	Music
Kids	Music
Punk	Music
R&B	Music
Metal	Music
Animals	Photography
Nature	Photography
Places	Photography
Fine Art	Photography
Letterpress	Periodicals
Literary Journals	Periodicals
Comedy	Publishing
Translations	Publishing
Literary Spaces	Periodicals
Calendars	Art Books
Anthologies	Publishing
Zines	Publishing
Young Adult	Fiction
Academic	Nonfiction
Radio & Podcasts	Periodicals
Makerspaces	Technology
Fabrication Tools	Software
Space Exploration	Technology
Camera Equipment	Technology
Flight	Technology
Robots	Software
Sound	Technology
3D Printing	Technology
DIY Electronics	Hardware
Comedy	Plays
Spaces	Theater
Immersive	Plays
Experimental	Plays
Festivals	Theater

Musical	Theater

7.2 Correspondance category and category_cat

category	category_cat
3D Crafts	0
Accessories	1
Action	2
Animation	3
Apparel	4
Apps	5
Art	6
Art Books	7
Audio	8
Audio Video	9
Videobooks	10
Videography	11
Audio Video	12
Audio Video Games	13
Children's Books	14
Classical Music	15
Comic Books	16
Comics	17
Conceptual Art	18
Country & Folk	19

Crafts	**20**
DIY	**21**
Dance	**22**
Design	**23**
Digital Art	**24**
Documentary	**25**
Drama	**26**
Drinks	**27**
Electronic Music	**28**
Faith	**29**
Farms	**30**
Fashion	**31**
Fiction	**32**
Film & Audio Video	**33**
Food	**34**
Food Trucks	**35**
Gadgets	**36**
Games	**37**
Graphic Design	**38**
Graphic Novels	**39**
Hardware	**40**
Hip-Hop	**41**
Horror	**42**

Illustration	**43**
Indie Rock	**44**
Jazz	**45**
Jewelry	**46**
Journalism	**47**
Literary Performances	**48**
Live Games	**49**
Mixed Media	**50**
Mobile Games	**51**
Music	**52**
Narrative Film	**53**
Nonfiction	**54**
Painting	**55**
People	**56**
Performance Art	**57**
Performances	**58**
Periodicals	**59**
Photography	**60**
Playing Cards	**61**
Plays	**62**
Poetry	**63**
Pop	**64**
Product Design	**65**

Public Art	**66**
Publishing	**67**
Restaurants	**68**
Rock	**69**
Sculpture	**70**
Shorts	**71**
Small Batch	**72**
Software	**73**
Tabletop Games	**74**
Technology	**75**
Television	**76**
Theater	**77**
Wearables	**78**
Web	**79**
Webseries	**80**
Woodworking	**81**
World Music	**82**

7.3 Correspondance main_category and main_category_cat

main_category	main_category_cat
Art	**0**
Comics	**1**
Crafts	**2**

Dance	**3**
Design	**4**
Fashion	**5**
Film & Video	**6**
Food	**7**
Games	**8**
Journalism	**9**
Music	**10**
Photography	**11**
Publishing	**12**
Technology	**13**
Theater	**14**