Credit Team, *credit@chalearn.org*
Ghiles Sidi Said (*ghiles.sidi-said@u-psud.fr*), Taycir Yahmed (*taycir.yahmed@telecom-paristech.fr*),
Eden Belouadah (*eden.belouadah@yahoo.fr*, Adrien Pavao (*adrien.pavao@gmail.com*).
`https://github.com/didayolo/credit`
Challenge website : `https://codalab.lri.fr/competitions/83`
Video URL : `https://www.youtube.com/watch?v=u6fz_gtHgvU`

Team Credit Project Report : Give Me Some Credit

January 2018

Abstract

In this work, we propose a method to solve *"Give me some credit"* Kaggle challenge. The first section gives a little introduction and the background describing the application domain and some motivations for the work. The second section is related to materials and method in which we present the task, we describe the data by showing the results of so many significant statistics and we explain how did we change the original data for cheating prevention purposes and how this helped improving our model's performance. In the third section, we present our contribution in terms of up-sampling the rare class, and we give more details about the models used. We Finally show and discuss the results obtained for each model and we resume our accomplishments with a conclusion.

1 Introduction and Background

In an economic and financial world full of risks and challenges, the task of detecting potential threats has become an urgency, especially in the banking sector. Machine learning is an awesome tool to analyze customer's data and make accurate decisions. This is why it is increasingly used in this field.
People in the United States are more and more interested in taking bank loans. Figure 1 [1] shows the total mortgage debt outstanding in the United States from 2001 to 2016. We see that the total mortgage debt outstanding in the U.S. amounted to approximately 14.29 trillion U.S. dollars in 2016.

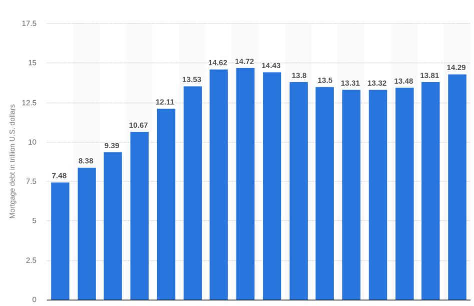

FIGURE *1 – Total mortgage debt outstanding in the US.*

The goal of this kaggle challenge (*Give me some credit - Sep 19, 2011*)[2] is to be able to decide if a loan should be granted or not by taking into account some information about the borrower. This project treats a serious real-world problem of risk detection. So, resolving such problems helps improve the security in the financial domain, and leads to more confidence between customers and companies.

2 Material and methods

The main task is to exploit information about borrowers expressed as 56 numerical features to predict the class which can be *true* or *false*.
In this section, we give some plots describing the data to help us understand the task particularities and challenges.

2.1 Exploratory analysis

We used *Stratified Shuffle Split* to get the same classes distribution in train, test and validation sets.
Here is the classes distribution :

FIGURE 2 – *Marginal distribution of target value*

The two classes are clearly not balanced. Most (93%) of customers will normally not face a financial distress in the next two years. Indeed most of customers have normal living conditions : They are aged, they have children (average=3), they have a good monthly income (average=6900$) and they weren't more than 90 past due. The loan for these people is granted.
Exact number of examples in Train/test/valid for each class is given in the next table :

Set	Train	Test	Valid
# Granted loans	112995	12629	13950
# Non Granted Loans	8505	971	1050
Total # of examples	121500	13500	15000

TABLE 1 – *Information about the dataset*

Dataset is not sparse and there's no missing data. Also, all features are numerical.

2.2 Univariate distributions

Since there are 56 variables, we're not going to show the empirical distribution for each one of them. Instead, we've selected a few and in the next figures, their empirical distributions (histograms) are shown.

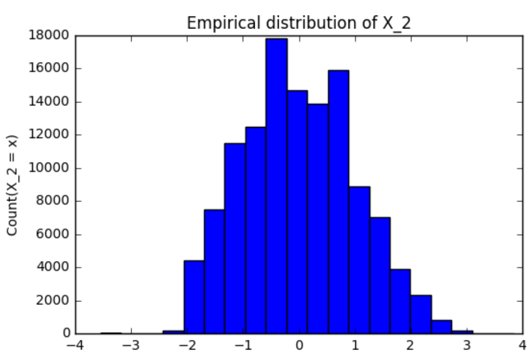

FIGURE 3 – *Empirical distribution of the variable X_2.*

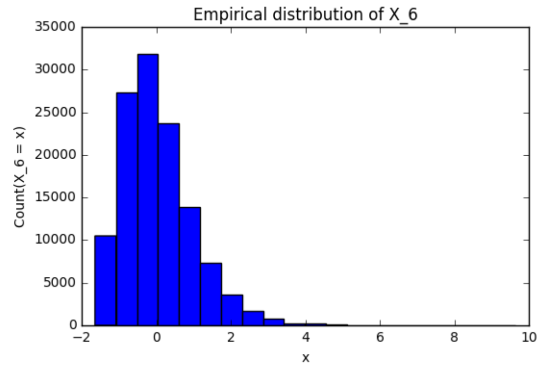

FIGURE 4 – *Empirical distribution of the variable X_6.*

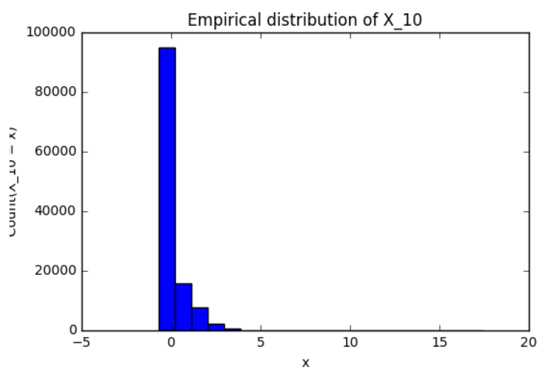

FIGURE 5 – *Empirical distribution of the variable X_{10}.*

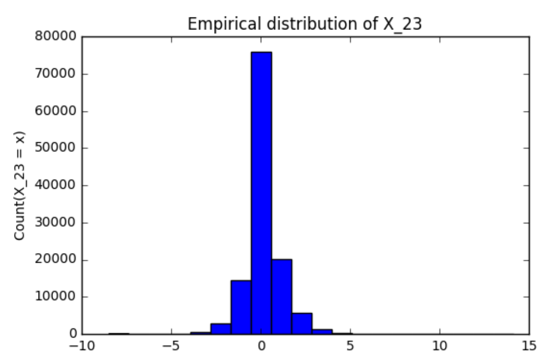

FIGURE 6 – *Empirical distribution of the variable X_{23}.*

We see that most variables follow approximately a normal distribution but there are some exceptions, X_{10} for example follows a distribution that looks more like a chi-square, the lower values being highly represented.

2.3 Scatter plots

The next figures show scatter plots of selected couples of variables. In the plots, the points corresponding to a target value of 1 (the loan should not be granted) are in red whereas the ones corresponding to a target value of 0 are in green (the loan should be granted).

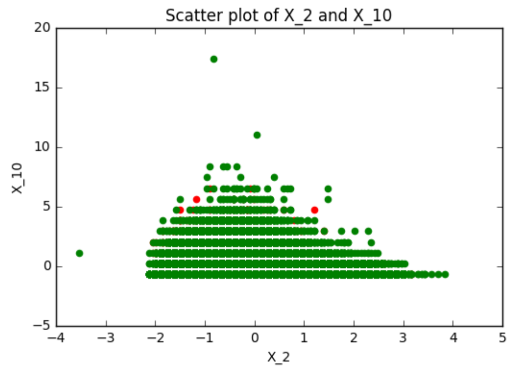

FIGURE *7 – Scatter plot of X_2 and X_{10}.*

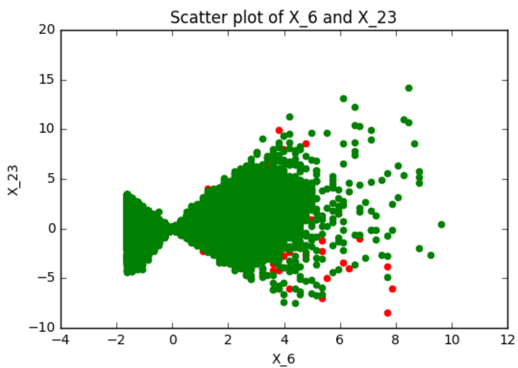

FIGURE *8 – Scatter plot of X_6 and X_{23}.*

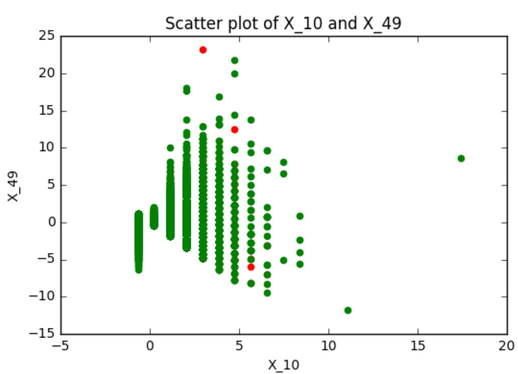

FIGURE *9 – Scatter plot of X_{10} and X_{49}.*

2.4 Univariate analysis

The variable that is the most correlated to the target is X_2, the correlation is however very week (0.12). We trained a logistic regression classifier using only the variable X_2. The classifier learns to classify everything as 0 (i.e. we can always grant a loan). This result is somewhat expected since we only used one feature, a very basic classifier, and the classes are very imbalanced.

2.5 Features correlation

The correlation between different features is as following :

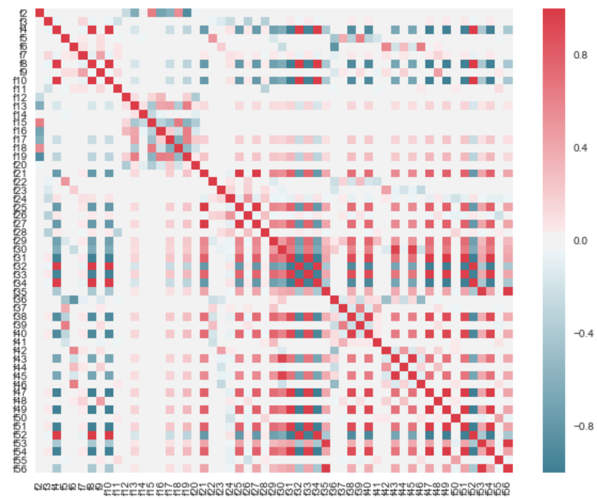

FIGURE *10 – Features correlation*

A statistic study has been done and the results of Top 5 positive and negative correlations between features are shown in the next table :

Positive correlation			**Negative correlation**		
feat1	feat2	corr	feat1	feat2	corr
f34	f52	0.99	f32	f47	-0.99
f32	f52	0.99	f47	f52	-0.99
f32	f34	0.99	f34	f49	-0.99
f25	f27	0.99	f32	f49	-0.99
f47	f49	0.99	f49	f52	-0.99

TABLE *2 – TOP 5 positive and negative correlations between features*

2.6 Principal Components Analysis

We performed PCA algorithm to see if it's feasible to reduce less important features.
We can't represent all features variance here, so we do it only for the four first features.

	PC_1	PC_2	PC_3	PC_4
Proportion of variance	0.9758	0.004497	0.003147	0.002058
Cumulative proportion	0.9758	0.980297	0.983444	0.985502

FIGURE *11 – Features correlation*

We can see that the first 10 features are enough to describe data so a dimensionality reduction is indeed possible!
Also, we see that variables have generally a very low variance (the values around the mean being highly represented) comparing to the first principal component whose variance is remarkably high.

2.7 Cheating prevention

The original dataset was composed of 10 features only. We used them to produce 46 other features using polynomial combination of the original features. Other techniques that we used to make it difficult to find the original data set is to shuffle both features and examples.
Adding features to the dataset helped also in improving the prediction quality of our model. More details are given in the next section.

3 Models and results

3.1 Evaluation metric

We choose the area under the ROC as our evaluation metric, for two reasons. First, we have an imbalanced dataset, so we need an evaluation metric that can capture the performance on both classes. Second, from a business use case perspective, it is equally important not to give credit to people who wouldn't repay, and give it to good clients.

3.2 Feature engineering

As stated above, the original dataset was retrieved from a Kaggle challenge. However, we applied some initial preprocessing on the features, to make the task easier for the participants. The below manipulations also helped us obfuscate the observations, to make **cheating** impossible.

- To handle **missing values**, we imputed the dataset using the median value.
- Shuffled the dataset along the rows.
- Scaled the dataset, by centering to the mean and component wise scaling to unit variance.
- Added polynomial features to generate a new feature matrix consisting of all polynomial combinations of the features (only interactions) with degree less than or equal to 2.

These different approaches helped us prevent against cheating, since it is almost impossible to map the observations in our dataset to the original ones.

3.3 Majority class

Since we are using the AUC ROC score as an evaluation metric, the majority class classifier gives 0.5 in performance.

3.4 Classical models

Below we present the results of most commonly used classifiers, **without tuning and data preprocessing**, besides up-sampling the minority class and shuffling the dataset. This step serves as a quick evaluation of various classifiers/methods. Note that we used *sklearn*[3] and *xgboost*[4] pre-implemented models. The scores stated below were calculated on the validation and test sets (not with cross validation).

classifier	Dev AUC	Test AUC	Avg
MLP	75.31%	74.01%	74.66%
QDA	75.23%	73.29%	74.26%.
KNN	65.31%	64.11%	64.71%
DT	76.11%	76.94%	76.52%
RF	76.57%	75.42%	75.99%
ADB	77.98%	77.47%	77.72%
GNB	52.33%	51.50%	51.92%
LR	74.64%	73.08%	73.86%
GB	78.33%	78.51%	78.42%

TABLE *3 – Scores of common classifiers*

Hyper-parameters of the classifiers :

- Multi-layer Perceptron classifier (MLP) : one hidden layer with 100 neurons, a constant learning rate of an initial value 0.001, a maximum number of iteration of 200 and a tolerance equal to 1e-4. The performance stated above was achieved using the sklearn implementation of the algorithm.
- Quadratic Discriminant Analysis (QDA) : with a tolerance rate, which is the threshold used for rank estimation, equal to 1e-4. The performance stated above was achieved using the sklearn implementation of the algorithm.

- K-nearest neighbors (KNN) : neighbors = 5. The performance stated above was achieved using the sklearn implementation of the algorithm.
- Decision tree classifier : max depth = 5. The performance stated above was achieved using the sklearn implementation of the algorithm.
- Random forest classifier (RF) : max depth = 5, estimators = 10, max features = 1. The performance stated above was achieved using the sklearn implementation of the algorithm.
- Gaussian Naive Bayes (GNB) : no hyper-parameters to report. The performance stated above was achieved using the sklearn implementation of the algorithm.
- AdaBoost classifier (ADB) : with 50 decision trees and a learning rate of 1. The performance stated above was achieved using the sklearn implementation of the algorithm.
- Logistic Regression classifier (LR) : with L2 as penalty, a tolerance of 0.0001, a C value of 1, the lib-linear solver and a maximum number of iteration equal to 100. The performance stated above was achieved using the sklearn implementation of the algorithm.
- Gradient Boosting (GB) : max depth = 3, learning rate = 0.1, number of estimators = 100. The performance stated above was achieved using the xgboost implementation of the algorithm.

Please note that the **performance** (score) shouldn't be the only criteria to select a model, **training time** is equally important when it comes to deploying the solution in production, particularly for professional use cases. Models based on support vector machines and multiple layer perceptron take significantly an enormous amount of time to be trained. Furthermore, rationale behind the decision made (model's **explainability**) is important to take into account, especially in this type of problem where the decision can influence one's life.

3.5 The Baseline classifier

In this paragraph, we evaluate the classifier proposed in our starting kit. We used quadratic discriminant analysis. It gives 74.16% performance on the validation set and 72.83% on the test set, with an average of 73.50%.

3.5.1 Hyper-parameter tuning

When setting the hyper-parameter *priors* to the classes percentages (6.68% and 93.32% in the training set, the same as the validation and test set by construction), we get a performance of 74.86% on the validation set and 74.34% on the test set, with an average of 74.60%. Besides the fact that the scores are higher, we can notice that the validation and test scores are close, compared to the results obtained without hyper-parameters tuning ; which is an indicator of the model's stability.

3.5.2 Up-sampling the minority class

Since we have an imbalanced classification problem, we considered up-sampling the minority class so that the algorithm learns to distinguish between both classes. Up-sampling is the process of randomly duplicating observations from the minority class in order to reinforce its signal. There are several heuristics for doing so, but the most common way is to simply resample with replacement. Thus, we applied the following steps :

- First, we separate observations from each class into different DataFrames.
- Next, we resample the minority class with replacement, setting the number of samples to match that of the majority class.
- Finally, we combine the up-sampled minority class DataFrame with the original majority class DataFrame.

When doing so, the ratio of the two classes is 1 :1 and we get the following results : 75.23% on the validation set and 73.29% on the test set with an average of 74.26%.

Remark : We also considered **down-sampling** the majority class. Down-sampling involves randomly removing observations from the majority class to prevent its signal from dominating the learning algorithm. The most common heuristic for doing so is resampling without replacement. The process is similar to that of up-sampling. Actually, down-sampling reduced significantly the training set size. Since we have 121500 training examples with only 6.684% of the observations corresponding to the minority class, down-sampling leaves us with only 16242 samples in the training set, whereas the validation and test sets (used for measuring the performance) have 15000 and 13500, respectively.

3.5.3 Up-sampling and tuning

When we combined the previous two methods, we get 74.90% as the AUC score of the validation set, and 74.18% as of the test set and an average of 74.54%.

3.5.4 Summary

We can see that our baseline model *Quadratic Discriminant Analysis* gave the best results after hyper-parameter tuning. You can find below the learning curve of this classifier.

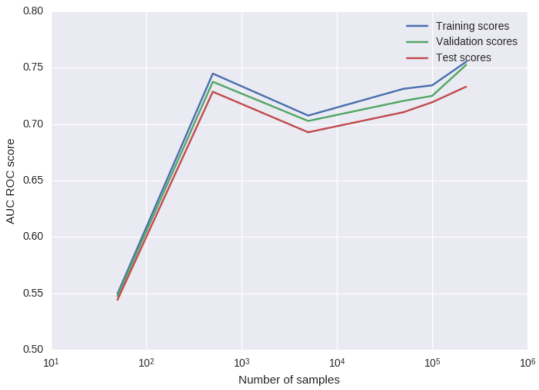

FIGURE *12 – Learning curve of the baseline*

3.6 Ensembling models : soft voting with adaptive weights

Generally, ensembles of classifiers perform better than single classifiers, by allowing for more granularity of choice in the bias-variance trade-off. That's why, we tried to ensemble the top performing classifiers, using soft voting, meaning that we predict the class label based on the arg-max of the sums of the **weighted** predicted probabilities, which is recommended for an ensemble of well-calibrated classifiers.

classifier	Dev AUC	Test AUC	Avg
DT	76.11%	76.94%	76.52%
RF	76.57%	75.42%	75.99%
ADB	77.98%	77.47%	77.72%
GB	78.33%	78.51%	78.42%
ENS	78.00%	77.75%	77.88%

TABLE *4 – Ensembling top classifiers*

Although, we weighted the contribution of each classifier by its performance on the validation set, we found that the results were lower than the best classifier : Gradient Boosting. That's why we decide to rather tune this classifier instead of ensembling it with other models. This decision is based on the benchmark analysis we performed in the section 3.4 and the fact that ensembling didn't outperform the Gradient Boosting model.

3.7 Tuning Gradient Boosting

First of all, we notice that we get better results using xgboost implementation compared to sklearn implementation of Gradient Boosting classifier.

3.7.1 Tuning the learning rate

The learning rate parameter can be set to control the weighting of new trees added to the model. We use Grid Search to, exhaustively, select the best learning rate for our model, among a list of values.

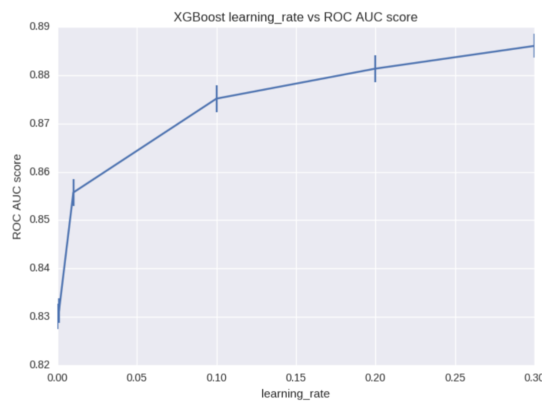

FIGURE *13 – Learning rate vs ROC AUC score*

We can see that the best learning rate was 0.3. We think this is a high learning rate and it suggests that perhaps the default number of trees of 100 is too low and needs to be increased. Next, we will look at varying the number of trees along with varying the learning rate.

3.7.2 Tuning the learning rate and the number of trees

Smaller learning rates generally require more trees to be added to the model. In this section, we will explore this relationship by evaluating a grid of parameter pairs. The number of decision trees will be varied from 100 to 500 and the learning rate varied from 0.0001 to 0.1. We expect that for a given learning rate, performance will improve and then plateau as the number of trees is increased. Below is a plot of each learning rate as a series showing ROC AUC performance as the number of trees is varied.

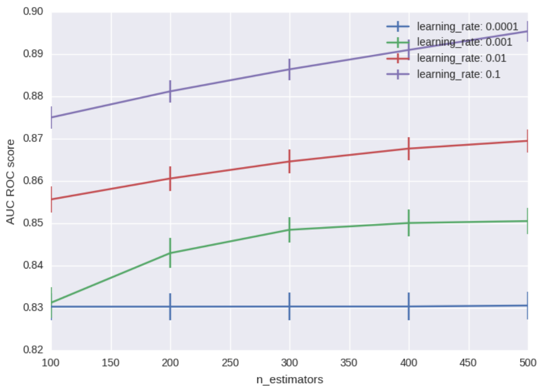

FIGURE *14 – Number of estimators vs learning rate*

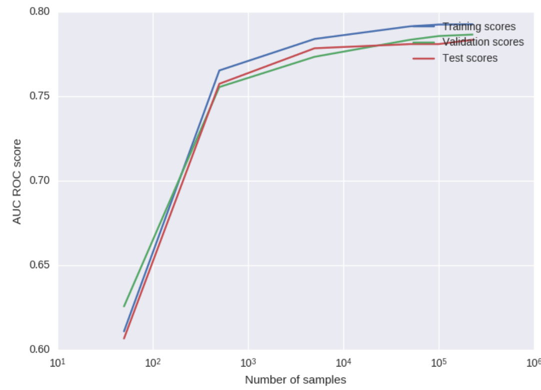

FIGURE *16 – Learning curve of Gradient Boosting*

We can see that the expected general trend holds, where the performance improves as the number of trees is increased. Performance is generally poor for the smaller learning rates, suggesting that a much larger number of trees may be required. We may need to increase the number of trees to many thousands which may be quite computationally expensive.

3.7.3 Dimensionality reduction

We also studied the performance as a function of the number of principal components after performing Principal Component Analysis.

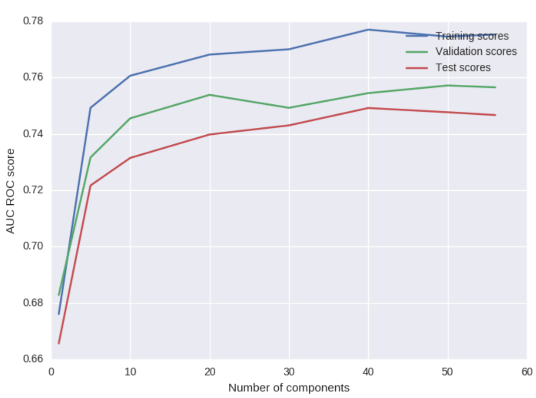

FIGURE *15 – Learning curve of Gradient Boosting as a function of the number of principal components.*

We can see that the best performance is achieved using 40-50 components.

3.7.4 Performance of the best model

Below we can see the learning curve of Gradient Boosting classifier.

Important remark : The results shown above for hyper-parameters tuning (figures 14 and 13) are calculated using cross-validation (10 folds of the training set). However, below, we present the best hyper-parameters tuned (giving the best performance) on the validation set.

Final results : The best performance is obtained with a number of estimators equal to 90 and a learning rate of 0.1. The training score is equal to 79.26%, the validation score is equal to 78.48%, the test score is equal to 78.53% with an average (on validation and test) of **78.50%**.

Remark : We also tried to plot error bars on the learning curve using bootstrapping, but since we have an imbalanced dataset, bootstrapping messed with the classes' percentages, leading to a drop in the performance.

4 Conclusion

Deciding to give credit to customers or not is a complex task since there is a combinatorial explosion in the number of possible examples. Machine learning algorithms are a powerful tool to accomplish such a task. In this work we prove that we are able to successfully generalize for any possible customer only by having a certain number of labeled examples.

Références

[1] T.Chen, C.Guestrin
Xgboost : a scalable tree boosting system

[2] https://www.statista.com/statistics/274636/combined-sum-of-all-holders-of-mortgage-debt-outstanding-in-the-us/

[3] `https://www.kaggle.com/c/GiveMeSomeCredit`