COLLECTIVE SCIENTIFIC PROJECT

Estimation of the probability of default

Michel ALHAWA, Reda BELHAJ-SOULLAMI, Pierre MORDANT,
Mohamed Amine RABHI, Céline RATON

ÉCOLE
POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

Abstract

The recent financial regulations established following the 2008 crisis have led banks to better control credit risk. In particular, it has become necessary for them to be able to identify debtors who will default, i.e. those who will not be able to repay their loans. The development of new statistical learning techniques, based on the exploitation of data, provides new directions for research to estimate the probability of default.

It is in this context that we carried out this project in collaboration with BNP Paribas on data provided by BNPP Leasing Solutions. We started by conducting simple statistical analyses, observing correlations between variables and using a dimension reduction algorithm. It soon became apparent that studying the linear correlations between the variables would not be sufficient to obtain a relevant prediction algorithm.

We then implemented two algorithms for predicting the probability of default: one based on a *Gradient Boosting* technique which was to serve as a reference, and the other on a neural network. We then optimized these algorithms and their hyper-parameters on our data to arrive at high-performance models that met the requirements imposed by the credit risk context.

We finally conducted a comparative analysis of the results obtained with our models and the one currently used by the BNPP. Our rather encouraging results, particularly with neural network models, suggest that these models could play a key role in client risk assessment.

CONTENTS

1 INTRODUCTION

The control of risks related to economic and financial hazards is one of the major tasks of any banking institution, especially when it comes to granting credit to individuals. Indeed, while it is necessary to protect against loans that are too risky, one should not put in place overly severe constraints that would harm bank profits or the productivity of economic agents. In fact, credit institutions have gradually developed increasingly complex models in order to discriminate as best they can between loan applications deemed risky.

Thus, numerous stochastic models have emerged since the beginning of the last century in order to anticipate the behavior of the various players involved in these transactions. Although relevant, they are now competing with *machine learning* methods whose effectiveness has been demonstrated over the last twenty years in other data processing problems. Consequently, these techniques have enabled banks to determine default risks, i.e. the fact that an agent is unable to repay its loan, via the calculation of the probability of default, the cornerstone of the credit score. It is in this trend that our project is situated, in which we have tried to go further into the field of techniques by applying *deep learning* methods to the problem of estimating the probability of default, because of the growing reputation of these methods and the interest that this represents for the BNP Paribas bank (BNPP).

In order to conduct our task, we established a schedule since September 2019. First, we needed to have access to confidential data from BNPP, then we had to clean the data. After this task, we had to study the data via various descriptive techniques, and establish benchmark results with *machine learning* algorithms. Finally, we focused on neural networks.

However, the time required to obtain the secure equipment necessary to access the data has forced us to revise our schedule. We then immediately studied and cleaned the data (Section 3) in order to run some description algorithms that we had already studied on fictitious data from the Kaggle platform. We then chose the XGBoost (Section 4.3) as the only reference algorithm whose results allowed us to measure the efficiency of neural networks (Section 4.4).

2 CONTEXTUALIZATION

2.1 After the 2008 crisis

The 2008 financial crisis has largely challenged the prediction models used by banks. In fact, as early as 2004, the Basel II agreements [1] allowed them to modulate their equity requirements according to the risk of their assets, which amounts to allowing banking institutions to set up their own credit risk assessment models. As a result, multiple credit scoring models have emerged. In the aftermath of the financial crisis, the new Basel III accords and the resulting financial regulations prompted banks to select and implement credit scoring techniques more appropriate to their loan portfolios.

2.2 A summary of methods used to estimate the probability of default

Several classification techniques are used to estimate the probability of default. These include statistical methods, such as Linear Discrimination Analysis (LDA) and Logistic Regression (LOG), unsupervised learning models, such as the k-nearest neighbor algorithm, or learning models widely recognized for their interpretability such as the decision trees [2]. We looked at several models like LDA and LOG. The latter is in fact the model used by BNPP in order to estimate the probability of default.

Comparative studies of statistical learning algorithms for credit risk can be found in the literature. Brown and Mues [3] have thus compared five different methods, including a method using Random Forests, a neural network, and logistic regression. They were particularly interested in their performance in the case of an unbalanced data set with a strong majority class (the non-defaults in our case). This paper was used as a reference in order to obtain a first measure of the performance of the possible models and to select the most relevant models.

We first studied each of the methods independently from a theoretical point of view by testing them on Kaggle data. We quickly retained two types of models: *gradient boosting* and neural networks. We were then able to compare our results with the results of the model used by the BNPP.

3
PRESENTATION OF THE DATA AND DETAILED DATA ANALYSIS

3.1 Primary description of the variables

3.1.1 • Qualitative description of the data

In december 2019, we had access to the data, which was from both BNPP Leasing Solutions and external sources. There was 91 variables and nearly 3 million entries. Every entry was associated to a 0-1 label (1 if there was a default, 0 otherwise) that was named "target". There was approximately 2% of defaults. Then, we started by conducting a qualitative description of the data, and we tried to bring out the most correlated variables with the target (the default variable).

The data consists of variables of many types : financial data, legal data and social data. Some of the variables are : line of business of the company, that gives an idea of the struggles that the company of a specific line of business may face. The data can be devided into to categories :

Legal data These variables give a precise idea of the history of the company, via its financial history. These variables feature for example the date of the last payment incident. We consider that these variables are very important because they have, *a priori*, a strong correlation with the default of the company.

Financial and social data These data could give an indication of the solvability of the company, for example with the following variables : age of the company, capital of the company, revenues.

We have access to more precise financial data. These variables give information about the state of the finances of the company, like the net cashflow, or the debt/cashflow ratio. This data gives an indication of the solvability of the company and its capacity to pay back the loan, hence they are very important.

One of the most important indicators of the debt reimbursement is the solvability ratio. It is defined as the ratio of the company equity over its total assets.

We also have acess to social indicators. For example, we know the type of client of the companies (identified by a label) that allows to have information on the market in which the company operates. By taking into account the APE code that indicates the business line of the company, this data allows us to position the company in terms of profitability [4]..

3.1.2 • Statistical description of some variables

In this section, we will show correlations between some variables and the "target" variable that tells if the company defaulted or not. It is only a primary analysis on data that we did not clean yet. (Section 3.2) to get a first grasp on the data. It is impossible to look at every single variable, as some of them are not numerical (hence inoperable for now). We will focus on a few examples to give an interpratation of the correlations.

- **Age of the company** : intuitively, we feel that an older company would be less likely to default. Graphically, it seems indeed that there is a link between the age of the company and the probability of default (Figure 1). Indeed, we note that the median and the first and third quartiles are a bit higher when the value of the target is 0 (i.e. when the client did not default) than when the the target is 1, i.e. when the client defaulted. Besides, the oldest companies did not default.

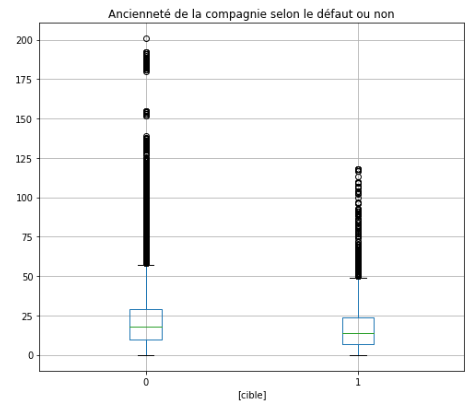


Figure 1: Boxplot of the age of the company depending on the target

- **Amount of the granted credit** : similarly, we could think that a company with a higher granted credit has a smaller chance of defaulting.. Although figure 2 seems to confirm this idea, as the amounts seem higher when the company did not default, figure 3 indicates the contrary. Without taking into account the maxima on the plot, we still note that the third quartile of the companies that defaulted is almost twice the one of the companies that did not default. We could argue that small loans are easy to pay back ; it is loans of a medium amount that give rise to defaults. However, we note with figure 2 that loans of a high amount of money (more that 500,000 €) have all been paid back for, which proves that these loans are granted with caution. This idea should be tempered though, as the small amount of such loans prevents to have statistically significative results.

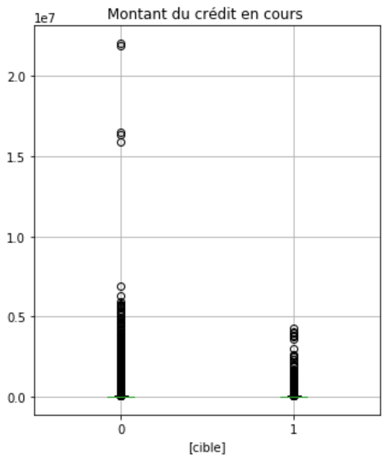


Figure 2: Repartition of the amounts loaned by the bank (in millions of euros)

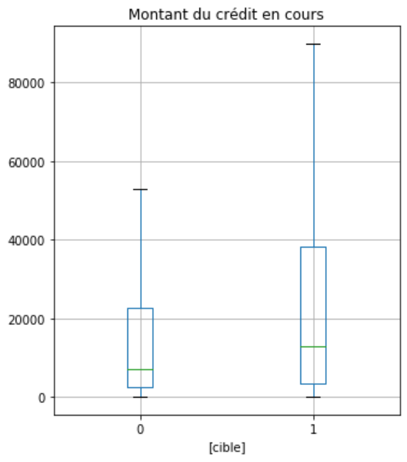


Figure 3: Boxplot of the amount of the loans, in euros

- **Last overdue payment** : Indeed, if a company has already had an overdue payment, it seems probable that it defaults again, compared to a company that has always paid back its loans. Figure 4 confirms this idea : close to 20% of the companies that have already had an overdue payment effectively default, compared to 1% in the case of companies that never defaulted. Therefore, we could argue that if the date of the last overdue payment is very old, the probability of default decreases. This is indeed illustrated in figure 5.

	Non default	Default
No overdue payment	0.99	0.01
Overdue payment	0.83	0.17

Figure 4: Default as the function of the existence of an overdue payment

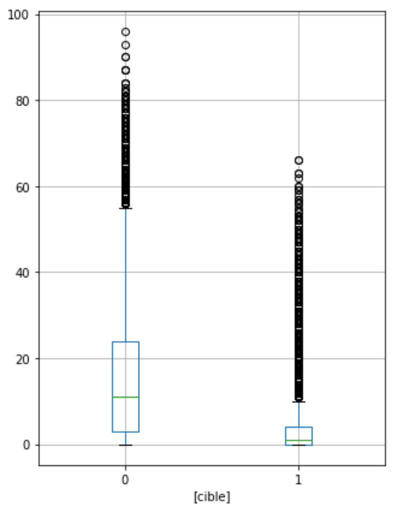


Figure 5: Boxplot of the age of the last overdue payment (in months)

- **Loan denial** : similary as before, we could argue that if a company has been denied a loan, this was for valid reasons, that could still be applicable at the time of the new loan application, which could increase the probability of default.

	Non default	Default
No loan denial	0.98	0.02
Loan denial	0.95	0.05

Figure 6: Default as a function of the existence of a loan denial

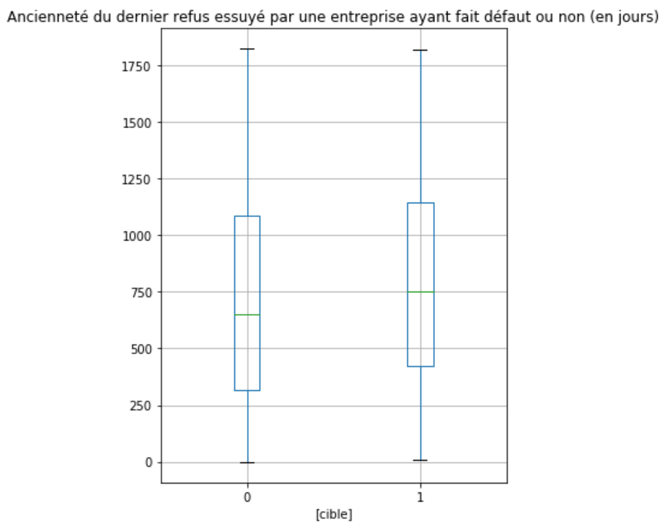


Figure 7: Boxplot of the date of the last loan denial (in days)

In this section we had a first look at the dataset, and we illustrated correlations between some variables with the target, giving an interpretation of these correlations. To conduct in-depth analysis of the data, the dataset must be prepared.

3.2 Preparation of the dataset for the model

Before applying our algorithms to the data, it is necessary to make them usable. Mainly, the cleanup phase consisted in solving type problems and transforming all values into numerical data, then studying the cases of missing data (which the neural network does not support). Finally, normalizing data that can be very spread out significantly increases the performance of the algorithms used.

Type transformations One of the necessary steps for the exploitation of data by algorithms is the management of the type difference between the variables in the data set: out of the 91 variables we have, 41 are of type `float`, 19 are of type `int`, and finally 31 are of type `object`. While floats and integers do not pose any difficulty to our algorithms, the management of the remaining 31 variables is more tricky. Among them, there are two main types :

- Dates;
- Categorical variables.

To handle the dates, we decided to only keep the order relation. Indeed, the date type was not supported by the algorithms. We replaced each date by the elapsed time from a fixed reference date. Categorical data correspond to parameters that take values in a finite set (usually of type `string`), for example, labels corresponding to a sector or a type of client. Most literal variables do not have an intrinsic order between elements, they are nominal variables. The difficulty is to convert them into numerical values without creating artificial correlations.

We used two methods to transform categorical variables into numerical variables :

- **Encoding with integers** : Each label of the class is associated with an integer according to a manually defined dictionary.
- ***one-hot* encoding** : for each of the elements of the class, we add a column taking the values "0" or "1" depending on whether the occurrence corresponds to this label or not.

By associating each value with an integer, we create false correlations since by calculating the distances between the inputs, we induce arbitrary proximities. This can bias the results of the chosen model. However, by encoding in *one-hot*, we increase the number of columns by as many variables as there are values in the class. This increases the complexity of the model, and thus the computation time of the algorithms we will use. Nevertheless, we took the decision to privilege this last method when the number of values taken by the variable was sufficiently small (less than ten values).

Handling missing data This type of data is not allowed by neural networks (although XG-Boost can handle it). The difficulty is that when some data is missing because it does not exist (for example the date of the last default for companies that have never defaulted), it is difficult to assign a value to it. The BNPP's *Leasing Solutions* department, which has worked on other leasing data, took the decision to divide their data into a "risky" and a "non risky" set according to financial criteria. In particular, a company was risky when it had already defaulted. Thus, they did not encounter this difficulty .

We propose here to quickly explain the processing that has been applied to the variables of the Italian dataset to manage the absence of some data (a missing value being associated to the type `NaN`). In fact, almost all the variables are missing for a few tens or even hundreds of thousands of entries. Replacing these missing data has a definite impact on the correlations between the variables that we have tried to minimize by interpreting the presence of these `NaN`.

In the case of **categorical data** for a *one-hot* encoding, the value `NaN` has been treated as a possible value of the variable and thus has been associated to a new column. Indeed, the absence of information can be interpreted as a piece of information in its own right, as in the above example of the time of the last default. For the only column where we have used a dictionary, we have associated `NaN` to an arbitrarily large value, which is very far from the others so as not to create a correlation. In the case of dates, the absence of data could correspond to data that was either not filled or inexistant. In the first case, we generally assigned to the `NaN` a mean or median value, especially for the dates of creation of the companies. Indeed, this simple method makes it possible to find a good estimator of the expectation of the variable, i.e. the empirical mean. In the other case, we replaced it by an extreme value very far from the other dates.

In the case of **floating variables**, in the same way as for dates, depending on the interpretation of the absence of values, the `NaN` are replaced by a suitable value . In the critical case where the rate of missing values in a column was too high, we simply deleted the variable. This is the method applied by the BNPP in its model [4]. Indeed, if too much data is missing, the variable becomes unusable.

At the end of this data cleaning, we obtain a dataset of 121 variables, all numerical and with all values filled in. It is therefore usable for the rest of our project. Nevertheless, this dataset is the result of a compromise between complexity and precision of the statistical study, without forgetting that a certain number of choices were made.

Data normalization In many statistical learning models, the quality of model optimization on the dataset is improved when the variables are normalized. We have therefore normalized the floating variables with the `StandardScaler` method of the Scikit-learn (Python) library which recenters and reduces the variables. We thus handle the problem of extreme values that are not representative of the dataset.

Preparation of the training and validation set In order to train the learning algorithms, we first divided the dataset into three subsets. The training set corresponding to 70 % of the dataset is used to train the model by supervised learning (default/non-default targets are known in advance) and to determine the final parameters. The validation set (20 % of dataset) is used to validate the selected model and check for overfitting problems (4.3.1). Finally, the test set is the one on which the model is applied to obtain predictions. This separation is necessary to ensure that the inputs on which we want to obtain a prediction have not been used during our training phase

3.3 Analysis of the correlations between variables

We will now use this cleaned up dataset to proceed to a more detailed description of the correlations between the variables. For this, we will use **correlation matrices**.

We define a linear correlation coefficient of two random variables X and Y, each having a finite variance, as $\frac{\text{Cov}(X.), Y)}{\sigma(X)\sigma(Y)}$ where $\sigma(X)$ and $\sigma(Y)$ denote the standard deviations of X and Y, respectively, and $\text{Cov}(X, Y)$ the covariance of the random variables X and Y.

We then define a correlation matrix of a vector of p random variables $\vec{X} = \begin{pmatrix} X_1 \\ \vdots \\ X_p \end{pmatrix}$ each of which has a (finite) variance, by the matrix of size $p \times p$ whose general term is given by :

$$r_{i,j} = \mathrm{Cor}\left(X_i, X_j\right) = \frac{\mathrm{Cov}(X_i, X_j)}{\sigma(X_i)\sigma(X_j)}$$

From our dataset with p variables and N entries $\{(x_k^1, \dots, x_k^p) \mid 1 \leq k \leq N\}$, we can construct an estimate of this correlation matrix, which we will also call correlation matrix, the r square matrix of size $p \times p$ and coefficients

$$r_{i,j} = \frac{\frac{1}{N}\sum_{k=1}^{N} x_k^i x_k^j - \frac{1}{N^2}\left(\sum_{k=1}^{N} x_k^i\right)\left(\sum_{k=1}^{N} x_k^j\right)}{\frac{1}{N}\sqrt{\sum_{k=1}^{N}(x_k^i)^2 - \left(\sum_{k=1}^{N} x_k^i\right)^2}\sqrt{\sum_{k=1}^{N}(x_k^j)^2 - \left(\sum_{k=1}^{N} x_k^j\right)^2}}$$

This correlation coefficient is always between 1 and -1. If it is equal to 1, it means that the two variables are proportional with a positive proportionality coefficient. If it is equal to -1, the two variables are proportional with a negative proportionality coefficient. The higher the absolute value of this coefficient, the more linearly correlated the variables are. This does not mean, however, that if the coefficient is zero, then the two variables are independent, since they may depend on each other in a non-linear way. It may therefore help to identify strong correlations between certain variables.

We can then try to establish the correlation matrix of our dataset (Figure 8) :

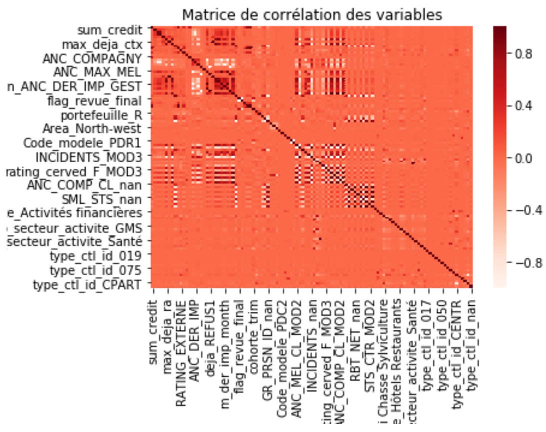


Figure 8: Correlation matrix of all variables

Apart from the very high computation time, it is complicated to extract any information from this matrix. To look at the influence of the variables, we will therefore try two methods: first, we have selected some variables that seemed relevant, and we have looked at the correlation matrix of these variables (Figure 9).

Several remarks can be made from this matrix: first of all, it would seem that the probability of default is fairly strongly correlated with the existence and seniority of a contract in amicable

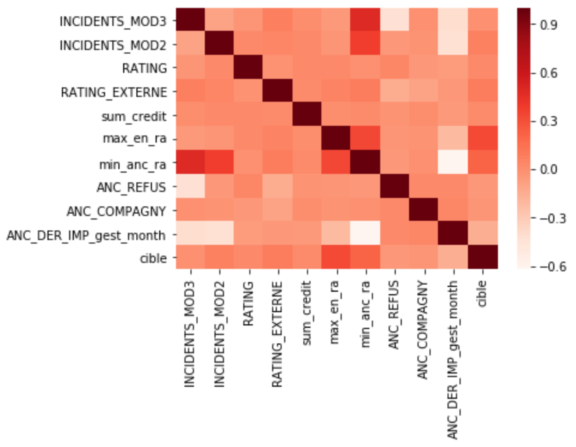


Figure 9: Correlation matrix of some arbitrarily selected variables

debt recovery (respectively "max_en_ra" and "min_anc_ra"). Similarly, it also seems to be correlated with the date of the last unpaid debt (ANC_DER_IMP_gest_month) but with a negative coefficient. This is in line with the hypothesis formulated during the qualitative analysis: the longer the last payment problem has occurred, the lower the probability of default. There are also fairly strong correlations, particularly between the length of time a contract has been in amicable debt collection and the length of time the last default has occurred. This correlation, like the others, should be considered with caution, since we have decided in cases of absence of refusal or default to replace the value `NaN` respectively by an arbitrarily small value and by an arbitrarily large value. Since a majority of firms had neither unpaid debts nor amicable collections, the presence of these large common values may have created arbitrary correlations. For the rest of the variables, the correlation coefficient with the target is too low in absolute value to be able to put forward hypotheses, except that they do not apparently correlate linearly.

The presence of a very large number of variables with a very low correlation coefficient with the target encouraged us to go directly to the variables with a high correlation coefficient with the target (Figure 10 and 11).
the length of time since the last unpaid payment was made, der_imp1, the existence of an unpaid payment, among others) but also to the existence of a contract for amicable collecting (max_en_ra, the existence of a contract for amicable collecting and min_anc_ra, the seniority of the last contract in amicable debt collection), of a contract in default (max_DEF_PAST), or of an incident in debtor (max_en_deb_gest, the existence of such an incident, and min_anc_deb_gest, the age of the last incident). Thus, the most important variables seem to be those that indicate whether the client has experienced serious financial difficulties in the past, and if so, how long ago. This can be seen in Figure 11 since, while correlated with the target, they are also highly correlated with each other.
With this information on correlations between variables, it seems reasonable to look at other methods of identifying and classifying categories of clients.

Name	Coefficient
target	1.0
ANC_DER_IMP	0.338
max_en_ra	0.333
max_en_deb_gest	0.331
ANC_DER_IMP_GEST	0.258
der_imp1	0.258
min_ANC_DER_IMP_GEST	0.258
max_der_imp	0.258
min_anc_ra	0.222
max_DEF_PAST	0.22
min_anc_deb_gest	0.203

Figure 10: Ten variables with the strongest correlation coefficients with the target

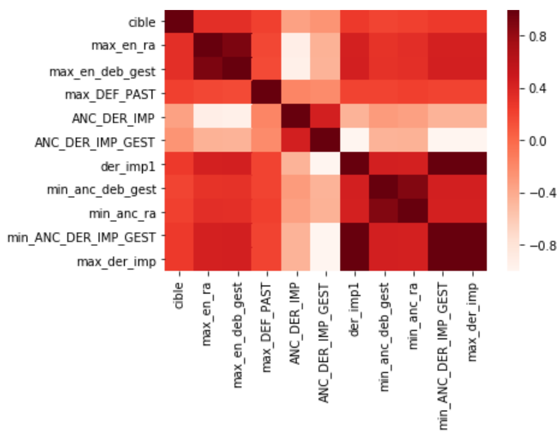


Figure 11: Correlation matrix with the 10 variables most correlated to the target

3.4 Principal Component Analysis

The dataset we are using has 121 variables for a total of about three million lines. The matrix representing the data is therefore of the order of one hundred million coefficients. Our goal is to implement predictive algorithms on these data to estimate the probability of default. These algorithms (decision trees, regressions, neural networks...) are often costly in computation time. One of the major issues is therefore to reduce the size of our data set to make the calculations feasible. This reduction of dimensions has other advantages: reducing the number of variables makes the model more easily interpretable, and the "dimension curse" (the fact that it is very difficult to infer conclusions from data when the data lives in a very large space) is avoided.

The following problem then arises: how to reduce the dimension while losing as little information as possible? This formulation leads us to try to quantify the information carried by a certain number of variables, then to look for algorithms to reduce the number of variables while preserving this information as well as possible. We have chosen Principal Component Analysis (PCA).

The general idea PCA is a data description tool, it is used to remove redundant information by looking for correlations between variables.

PCA has two interpretations (which are equivalent): either one seeks to reduce the number of variables, or one seeks to reduce the number of individuals. Depending on how one wants to use PCA, one of the two interpretations is chosen. In our case, since we want to do a PCA to reduce the dimension and then apply predictive algorithms, we will try to reduce the number of variables.

Our variables are therefore vectors of $\mathbb{R}^n$ (where n is the number of individuals). The principle of PCA is to look for axes on which to project these variables, so that the variance of the scatterplot around this axis is maximal (the goal is to capture as much diversity as possible from the original scatterplot), which is equivalent to diagonalizing the covariance matrix of the data. The eigenvectors correspond to the projection axes (and thus to the new variables), and are called principal components, and the eigenvalues to the amount of variance explained by each axis. The $q < p$ first principal components can then be retained if they are believed to summarize enough information (for example, 95 % of the information). This will reduce the number of variables used. Note also that the principal components are linear combinations of the original variables, so they can be interpreted.

Application to our data Before proceeding with PCA, it is a good idea to consider what should we expect.

During the pre-processing phase of the data, we were confronted to a large amount of missing data, which we sometimes treated in a uniform manner (by replacing it with the average for example). This mode of processing then artificially increases the level of correlation between the variables, which can also make the PCA results difficult to interpret: if there were too much missing data processed in this way, we would have too much correlation and therefore we will have artificially very few principal components.

Figures 12 and 13 contain the results obtained after running a PCA on the normalized data. We present here the graph giving the proportion of variance explained as a function of the number of principal components retained (Figure 13). We can see that if only about 60 variables are retained, only 5% of the information is lost. The figures 12 and 13 show the eigenvalues associated with the different main components and the part of variance explained by each main component.

In addition, one can be interested in the correlations of the initial variables with the factor variables - those inferred from the PCA. This will give us an indication of the representativeness of the factorial variables. We choose to draw the **correlation circle** on the first factorial plane - the one generated by the first two factorial variables - in Figure 14. This circle represents the scalar product $(X, Y) \mapsto \text{Cov}(X, Y)$ between the initial variables and the first two factorial variables . There are two types of variables:

- Those that are fairly well described by the first two factor variables. These are the ones that have a high norm in the circle. One can therefore deduce a strong proximity between these variables.
- Those that, on the other hand, are poorly represented in the first factorial plane. These are more numerous, as predicted by the high number of variables needed to reach 50% of the explained variable.

In fact, it is reasonable to work with a transformed dataset containing only about 40 variables, which explains 80% of the variance of the data.

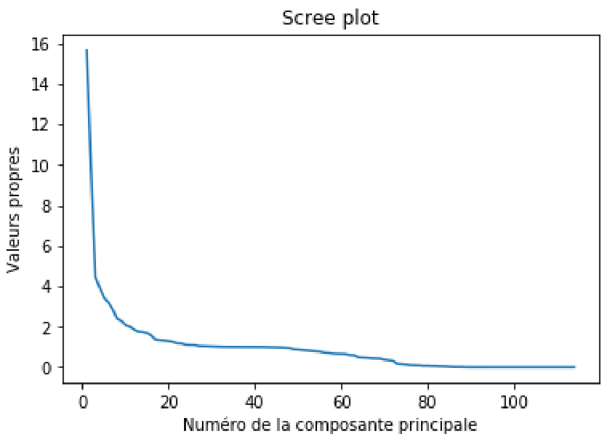


Figure 12: Eigenvalues associated with the principal components

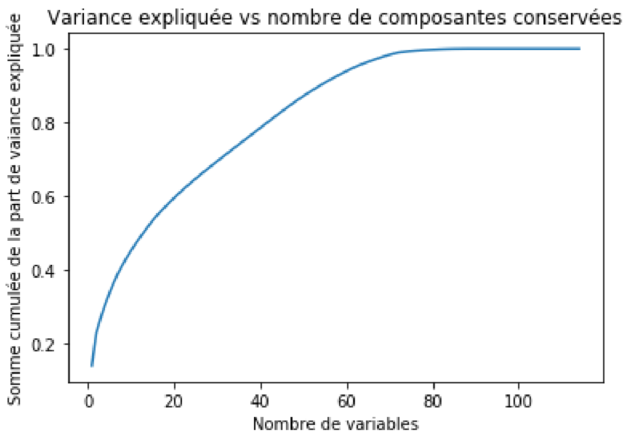


Figure 13: Proportion of explained variance

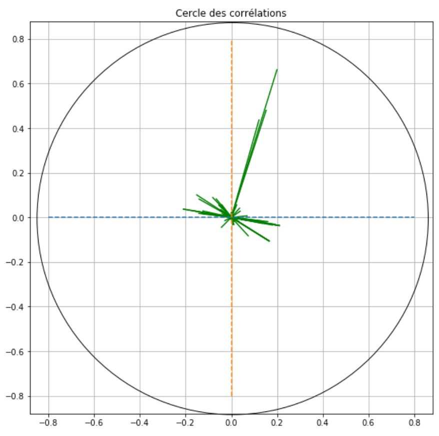


Figure 14: Circle of correlations of the variables on the first factorial plane

In addition to this aspect of saving time and computational complexity, it is necessary to look at the new data transformed by the PCA. This will give us an indication of the type of algorithm we should use. Indeed, if we manage to identify discriminant groups between the points of the two classes (cited above), then a linear separation algorithm, such as logistic regression, is sufficient. [2].

In fact, one can look at the projection of the points of the dataset on the first factorial plane. This will give a fairly good indication of the distribution and the possible separability of the points. Nevertheless, as shown in figure 15, the variables are not discriminated according to the target. Indeed, although we see three clusters of points for which we have developed a Gaussian estimation, it is clear that the red points (the "default" targets) are not distinguishable from the yellow points (the "non-default" targets) within each cluster. Therefore, we know that it will be necessary to use non-linear methods. This further supports our choice of XGBoost and neural networks as classification techniques.

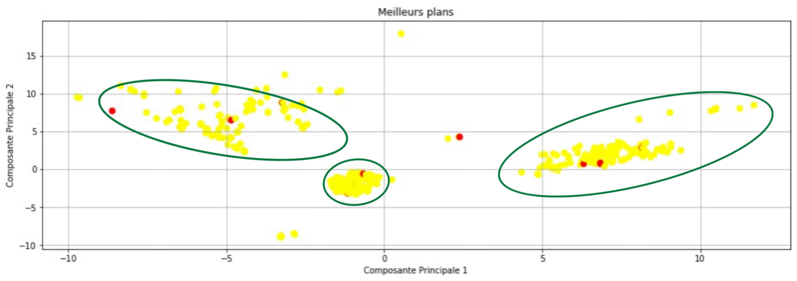


Figure 15: Projection of the points of the dataset on the first factorial plane

4 APPLICATION OF THE LEARNING MODELS

4.1 Choice of models

Many types of models have been used for predicting the probability of default, as detailed in chapter 2. Our project, carried out in collaboration with BNP Paribas, is part of an exploratory logic for the company. Our objective was therefore to test new models for the bank. The Leasing subsidiary of BNPP, which provided us with the data, uses a model based on logistic regression [4]. In addition, models based on boosting techniques had already been studied and successfully tested by the BNPP [5].

In this project, we looked at two models in particular.

- **A gradient boosting model**, which we use as a reference, since this technique has already been tested before [5].
- **A model based on a neural network**. Indeed, neural networks can perform two-class classification tasks, and are particularly efficient when the amount of data is high. This is the case for our dataset since it contains nearly three million lines.

4.2 Evaluation of model performance

The model that we want to set up aims to estimate the probability of default of a company. In order to select a good model and to be able to compare it to other existing models, we first need to determine a performance evaluation metric.

4.2.1 • The problem of asymmetric losses and unbalanced classes

Unbalanced classes In a classification problem, many metrics are possible. The most common and simplest metric is *accuracy* It is defined very simply as the proportion of correct predictions.

However, let's remember that our two classes (defect and non-defect) are unbalanced: there are only 2% of defaults in the dataset. Thus, a classifier whose prediction would be systematically " non-default " would already have 98% of correct predictions. Other metrics that take into account the imbalance between classes exist. These are based on a matrix called *confusion matrix*. Let's first give some useful definitions for the rest of the report:

- True positive (TP): individuals of the default class, having been classified as defaults.

- True Negative (TN): individuals in the non-default class, having been classified as non-default.
- False positives (FP): individuals in the non-default class, having been classified as defaults.
- False negatives (FN): individuals of the default class, having been classified as non-defaults.

The confusion matrix is

$$M = \begin{pmatrix} \text{TP} & \text{FN} \\ \text{FP} & \text{TN} \end{pmatrix}$$

Several metrics are defined from these values.

- Precision : $P = \frac{\text{TP}}{\text{TP+FP}}$. It indicates the proportion of estimated defaults that are actually defaults.
- Recall $R = \frac{\text{TP}}{\text{TP+FN}}$. It indicates the proportion of true defaults detected by the model.
- Score F1 : $\text{F1} = 2\frac{P \times R}{P+R}$. It combines precision and recall.

A good model must have both good precision and good recall, since it must both detect a fairly large number of defaults, without unfairly classifying too many individuals in the default class.

Note that these metrics are applicable to classification algorithms that return labels 0 or 1, for example the classifier implemented in XGBoost (4.3.2) . However, in the case of the neural network or the low-level implementation of XGBoost, the algorithm returns a probability. Therefore, it is necessary to specify a decision threshold. Once the threshold t is specified, an individual with a predicted probability of default p will be classified as a default if and only if $p > t$.

Thus, the above metrics depend on the decision threshold one chooses. However, there is a tool that can be used to overcome this difficulty: it is the *ROC* curve, or *receiver operating characteristic*. It is a curve giving the rate of true positives as a function of the rate of false positives, when the decision threshold varies. The curve is plotted from the $(0, 0)$ point to the $(1, 1)$ point. The ideal classifier corresponds to the (0.1) point. A measure of the quality of a classifier is given by the area under the ROC curve, which is called *AUC*. This type of curve and the AUC metric are widely used in the case of unbalanced data because their results relate to the positive class (in our case the defect class) that we want to highlight. The higher the AUC score, the better the model's ability to discriminate defaults.

For the neural network-based model, when we refer to precision or recall, we will refer to the metrics associated with the classifier corresponding to the $t = 0.5$ threshold. We will also use AUC (which does not require a threshold to be defined).

Asymmetric losses The metrics described above are relevant because they allow performance to be measured on each class, which is useful when classes are unbalanced. However, our problem presents another difficulty. Intuitively, we understand that it is more problematic for a bank to grant credit to an individual who is going to default than not to grant credit to an individual who is not going to default. It is this phenomenon that is referred to here by the expression "asymmetric losses".

The metrics defined above do not allow this phenomenon to be taken into account. Indeed, a classifier (defined by a threshold of 0.5), detecting very well the defects but less well the non-defects can have the same AUC or F1 score as a classifier detecting very well the non-defects but less well the defects. In practice, a compromise must be made between recall (detecting a good part of future defects) and accuracy (not detecting too many false defects). As the previous metrics take into account recall and precision in a symmetrical way, they are not perfectly relevant.

The F_β score allows combining recall and precision to keep only one evaluation metric and differentiating the importance given to each score. It is defined by

$$F_\beta = (1+\beta^2)\frac{P \times R}{\beta^2 P + R}$$

It allows to take into account the recall β times more than the precision. The difficulty is then to choose the β parameter. Thus, we should ideally quantify how important it is to properly detect defects.

In [6] an approach based on the costs associated with the different predictions is proposed. These costs, corresponding to the cost for the bank to classify a defect as non-default or a non-default as default, make it possible to set a decision threshold (different from 0.5), making it possible to classify an instance in the class with a minimal expected cost. This approach seems very relevant since it provides a definitive answer to the problem of asymmetrical losses. However, in order to implement this approach, the costs associated with the different predictions must be precisely estimated. In reality, the estimation of these costs is complex and requires the consideration of several other phenomena. Moreover, such approaches are not common in the literature and would therefore prevent the comparison of our model with other existing models. We discuss this problem again in the case of the boosting model (4.3.2).

4.2.2 • Approach used

We then decided not to pursue a quantitative approach based on costs, as it is too complex to estimate them efficiently. Nevertheless, we need to keep the possibility of measuring the discriminant character of our model, in particular its capacity to detect defaults.

We use the following approach: we train the models by optimizing the classical metrics (AUC, F1). As these metrics do not allow us to quantify the difference in performance of our model between precision and recall, we must also ensure that from this point of view, the model performs well. To do this, we therefore observe precision and recall, as well as other metrics such as the mean probability of default for the default class, and the mean probability of default for the non-default class.

We will use this approach to train the models (and in particular to adjust their various parameters), but also to compare our models with each other, even if the criteria used to choose the model are partly qualitative.

4.3 Gradient boosting

XGBoost (eXtreme Gradient Boosting) is a well-known and efficient open source implementation of the *gradient boosting* algorithm. The *gradient boosting* is a supervised learning algorithm whose principle is to combine the results of a set of simpler and weaker models to provide better prediction. XGBoost has won numerous learning competitions because of its ability to perform with a wide variety of data types. In addition, its large number of hyperparameters makes it adaptable to the problem at hand. This flexibility makes XGBoost a solid choice for regression and classification problems. It was first introduced in Chen and Guestrin's 2016 article [7].

4.3.1 • Presentation of the model

The theoretical principle of the *gradient boosting* algorithm is presented here. The algorithm is based on the boosting method according to which learning is done by successively training sub-algorithms and improving at each step the response of the previous algorithm. The XGBoost algorithm works on this principle by using shallow decision trees as sub-algorithms as shown in Figure 16. The parameters of these trees can be modified by the user. Decision trees are built successively so as to minimize a cost function by a second order gradient method.

Position of the problem in the case of classification and expression of the cost function. Let's consider the following classification problem in which we try to assign each entry a value between 0 and 1:

Let n entries $x_1, ...x_n$ associated with the values $y_1, \dots, y_n \in \{0, 1\}$ and m variables. We associate to the t-th decision tree a function f_t which contains information on its structure (nodes and leaves). In order to learn these functions, the algorithm will minimize a cost function noted $\mathcal{L}$ at each iteration. Formally, let $\hat{y}_i^{(t)}$ be the prediction of the i-th instance at the t-th iteration, then we build the t tree, so as to minimize the cost at the t-iteration, given by

$$\mathcal{L}^{(t)} = \sum_i \left(l(y_i, \hat{y}_i^{(t-1)}) + f_t(x_i) \right) + \Omega(f_t) \quad \text{et} \quad \Omega(f) = \gamma|f| + \frac{1}{2}\lambda \left\|\omega\right\|^2$$

where $|f|$ is the number of leaves of the tree modeled by f and ω is the vector of weights associated with each leaf. We do not detail how to build the tree at each iteration.

Here, l is a convex loss function and Ω is the "target" which is used to penalize the complexity

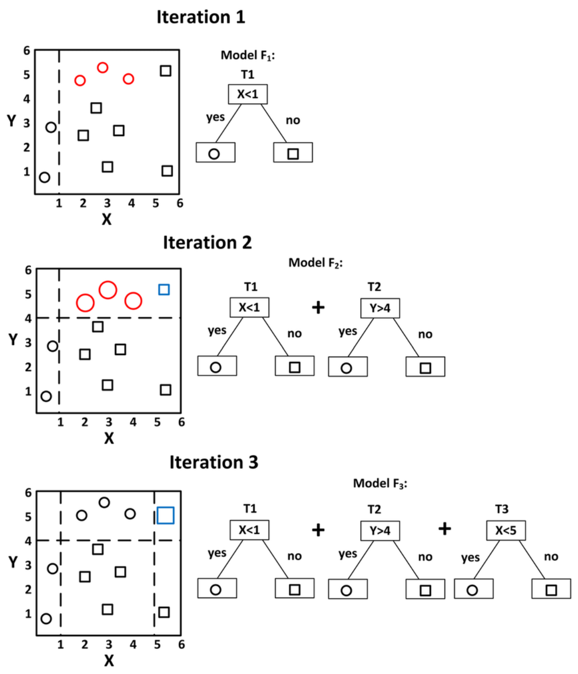


Figure 16: How the algorithm works

of the model in order to avoid overfitting [1]. In the case of a classifier, the loss function generally used is called cross entropy described below. The parameters γ and λ are modifiable in the model [7].

The study of this mathematical formalism allows us to set up a strategy to parameterize the model. It will first be a matter of choosing the best loss function for unbalanced data and then selecting the parameters relating to the decision trees, their size, their number, etc.

Choice of the loss function The loss function is essential when learning the algorithm. There are several possible functions for classification available in the XGBoost library. However, in our case they do not seem optimal because they do not penalize enough the prediction of non-defaults on real defaults (paragraph 4.2.1). XGBoost allows to implement a custom loss function.

It is enough to provide him with the expression of the first and second order derivatives of the function to apply the gradient method. In the case of unbalanced data, the loss must penalize more when "non-default" is wrongly predicted. This problem has been discussed in Chen Wang's article [8] which introduces `imbalance-xgboost`, a Python package dealing with the issue of unbalanced data. He proposes a classifier based on the XGBoost classifier whose

[1]Overfitting refers to a situation where the model fits the training data perfectly but fails to generalize to other data. Overfitting can be detected by monitoring the model's performance during training using the validation set. When model performance on the validation set deteriorates, overfitting occurs

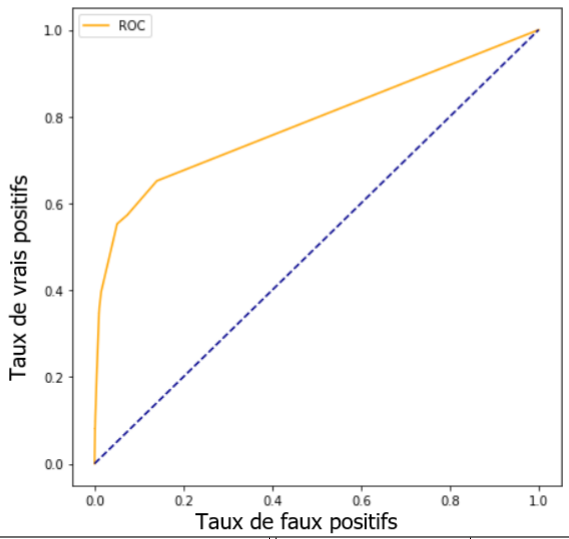
loss function has been modified and now corresponds to a weighted cross-entropy :

$$\mathcal{L}_p = -\sum_{i=1}^{n} p y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \quad \text{où } p \text{ is a penalty coefficient}$$

The parameter $p \in]0, +\infty]$ is easily interpreted: the larger p is, the more the network is encouraged to better detect defects. The symmetrical case is obtained for $p = 1$, where our loss function matches the classical loss *Binary Cross-Entropy*. On the contrary, when $p < 1$, we rather favor the detection of non-defects. Intuitively, we will therefore use a value of p greater than 1 because the model will tend to better predict defects (class 1) to minimize the loss. As the package was still being designed, we could not apply it directly on our data, so we implemented weighted cross entropy in the case of our model.

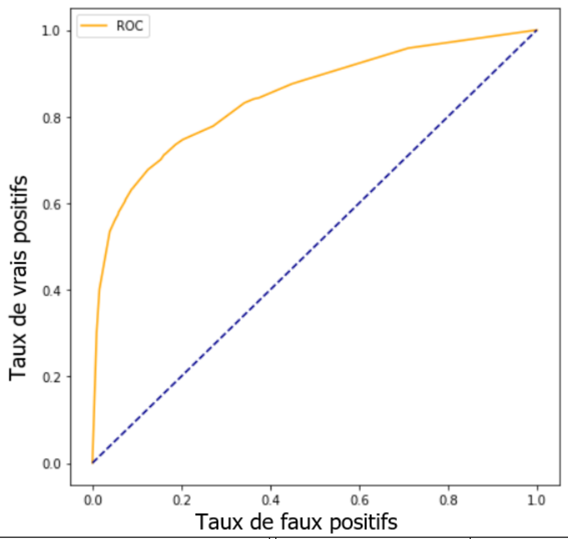
After testing different values of the penalty, we observed that the relevant values of the p parameter had the same order of magnitude as the non-defect rate on the defect rate (which is 45 for our data).

Indeed, to choose the penalty value that should be used, we studied the performance of the algorithm on the validation set for several values of p. When the penalty coefficient p is very small (0.01) then the model only predicts non-defects, on the contrary the greater the penalty, the more defects the model predicts. We present (Figures 17 and 18) the ROC curves for $p = 1$ and $p = 45$.



True / Prediction	Non default	Default
Non default	815 954	43 659
Default	8 833	10 918

Figure 17: Results for $p = 1$



True / Prediction	Non default	Default
Non default	809 493	50 120
Default	8 352	11 399

Figure 18: Results for $p = 45$

Note that in the case p =45, the model predicts more defects (61,000 for 53,000 in the case $p = 1$). This is indeed in line with our intuition. Moreover, the AUC score increases with the penalty. On the other hand, the accuracy of the model is lower. However, it can be seen that the model is more capable of identifying defaults. Indeed, the recall increases from 0.55 for $p = 1$ to 0.57 for $p = 45$. Although this difference is small, it is not insignificant, especially since these performances were obtained on a model whose hyper-parameters had not yet been adjusted to the data. This is why we chose to use the new metric with $p = 45$.

Note on XGBoost's `scale pos weight` parameter Changing the loss function is intended to balance the classes in the case of unbalanced data. As this problem is recurrent in the literature, the XGBoost classifier has a parameter called `scale pos weight` which would allow to deal with it. The literature does not specify how this parameter intervenes in the model, but it would allow to compensate for the imbalance by setting its value to the rate of non-defects. An attempt was made to estimate the performance of the model as a function of this parameter (Figure 19).

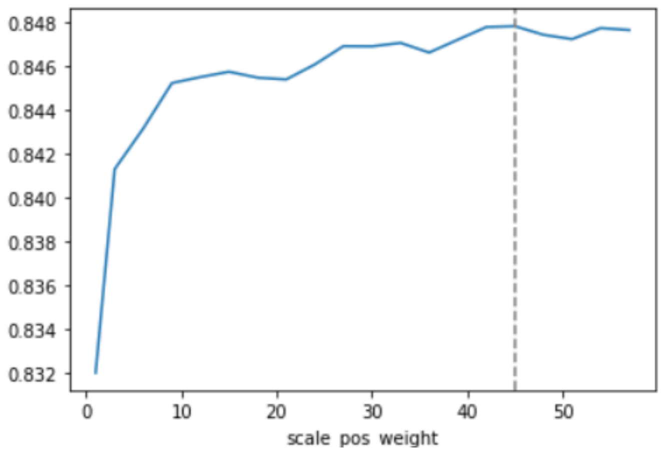


Figure 19: AUC score as a function of the `scale pos weight` parameter

Performance was evaluated with the model over the validation set. The area under the curve is larger for the value of the parameter equal to 45 (the non-defect to defect rate of our data). Nevertheless, it can be observed that increasing this parameter quickly has little effect and that the score seems to converge. Moreover, the scale of the figure proves that the variations are minimal. In the following, we set the value of the parameter to 45.

4.3.2 • Training and choice of hyper-parameters

Complexity The algorithm has a large number of adjustable parameters, so the selection of the optimal parameter set is crucial. The method consists in testing several dictionaries of parameters and selecting the best one. At each parameter set test, the algorithm has to be re-trained, which takes a very long time (about five minutes at each training) on the two million entries. We were therefore confronted with computation time problems related to the implementation of the XGBoost classifier.

In particular, it was necessary to find a more efficient implementation that would allow us to define the number of iterations i.e. the number of trees successively driven in the model. We used the AUC score and the error rate during the iterations, which are presented in Figures 20 and 21. We can notice that after a certain number of iterations, the model converges (especially on the error rate curve) and the marginal contribution of each iteration decreases. In addition, one can see on the AUC curve that the model begins to overfit. This behaviour is visible starting from the 100th iterations on the Figure 20.

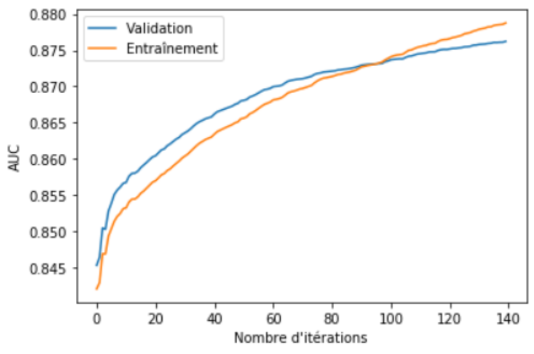


Figure 20: Evolution of the area under the curve in function of the number of iterations

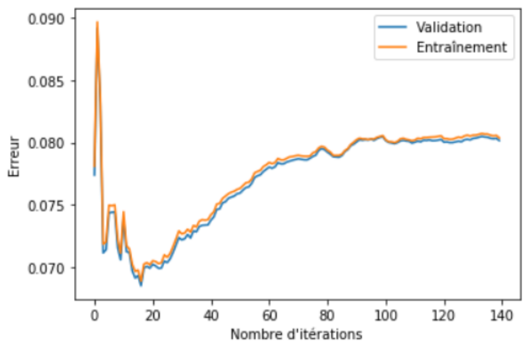


Figure 21: Evolution of the error rate in function of the number of iterations

Choice of the XGBoost architecture Adjusting the number of iterations is therefore essential in order to maintain reasonable computation times (especially when training the algorithm with many parameter dictionaries) and avoid overfitting. The XGBoost library is divided into two parts:

- a high-level implementation with a number of predefined parameters, including XGBClassifier, the classifier we initially used. This model has the advantage of directly returning a 0 or 1 label. (4.2.1),
- a low-level implementation that allows the user to redefine all the parameters and build his own model. This makes it more adaptable and parameters such as the number of iterations can be set. In addition, these models are much faster in practice. We have chosen to use this implementation for the following.

The training, which lasted about two minutes, was reduced to a few seconds. This enabled us to test more parameter sets quickly. However, low-level models do not return a class 0 or 1 but a probability depending on the position of each entry in the decision trees. This is why, in the same way as for neural networks (Section 4.4), it was necessary to define a threshold beyond which the input would be predicted as a "default". (cf. 4.2.1). The value of this threshold has an influence on the classification performance of the model. To adjust it, the value of the quantiles of the probabilities returned by the algorithm on the training set was used. This gives a good estimation of the threshold to be used. The interest of this method is not to set the threshold at 0.5 and thus to discriminate classes more (since we want to identify the defaults). For example, if the quantile at 90

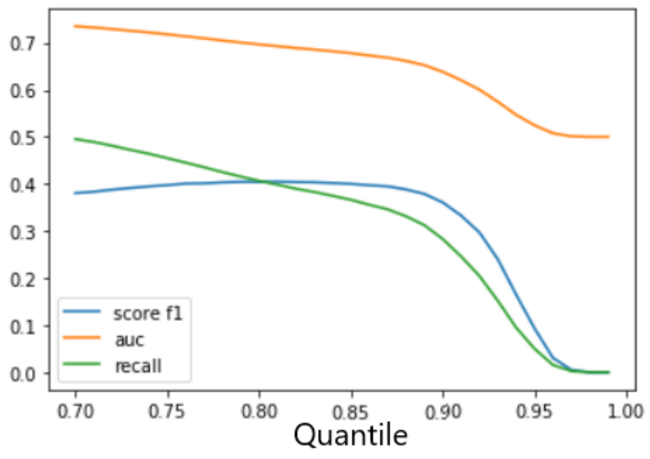


Figure 22: Score according to the quantile used

We ploted (Figure 22) the AUC score on the validation set for different values of the quantile. When it's equal to 1, then all the probabilities returned will be lower and therefore no defects will be predicted. Empirically, a good threshold value is the 0.9 quantile. However, as we will see later on, the choice of this threshold will strongly depend on our priorities regarding the performance of the model.

Definition of the model parameters In this section, we present the most significant parameters on which we have worked

The first parameters mentioned below are used to control the decision trees. Well adjusted, they avoid the overfitting presented in 4.3.1. In the case of the boosting model, this occurs when the trees become too branched; in the extreme case, the leaves of the trees are associated with a single input of the drive assembly.

- The learning rate `eta`. This is the step in the gradient method. It indicates the participation of each new tree in the result. The smaller the step, the smaller the influence of each tree.
- `max depth`, the maximum size of a tree;
- `min child weight`, the minimum weight of a leaf, i.e. the minimum number of entries in this leaf. A large value of the parameter avoids overfitting;
- `subsample`, the rate of inputs on which the algorithm works at each iteration. Set to 0.5, the algorithm would start by randomly selecting half of the inputs at each iteration before building the decision sub-tree;
- `colsample tree`, defines the number of variables used in the construction of each tree (this is the equivalent of `subsample` on the columns). The subsampling is done once for each constructed tree;

- λ and α, the parameters of the objective function Ω;
- `scale pos weight`, controls the balance of negative and positive weights;
- `num parallel tree`, the number of trees built at each iteration, we then average the result of the trees. This improves reliability and accuracy.

Learning parameters are also available:

- `num boost round` is equivalent to the number of iterations;
- `objective`, the loss function passed as argument (4.3.1);
- `eval metrics`, the metric to evaluate the model on the validation set;
- `cv`, the number of subsets to apply the cross-validation [2].

Strategy to improve the hyper-parameters These parameters are not learned by the model, so they must be set before the training. This step is known as *fine-tuning*.

To adjust the parameters, we used the `GridSearch` method of the *scikit-learn* library which consists in training the algorithm for several values of the parameters and which evaluates the performance of the model for the score passed as an argument. However, it is impossible to use this method on all possible values of the parameters because of the computation time. Thus, we have developed a strategy to adjust the parameters successively. We start by setting a step `eta` large enough (0.3). We then adjust the parameters specific to the sub-trees, and then the parameters that control overfitting. The final step is to decrease the step `eta`. Although this strategy does not seem optimal because it does not confront many parameters at the same time, it allows us to obtain a performing model. Nevertheless, we decided to apply the `GridSearch` function one last time for parameter values close to those previously established to improve the reliability of the method. Figures 23 and 24 show the evolution of the AUC score as a function of the parameters `max-depth` and `subsample`. We therefore chose a maximum depth of 3 and a value of `subsample` of 0.8.

As shown in the curve on the left, which represents the AUC score as a function of the maximum depth of the decision trees, this method is not optimal. Indeed, here it seems obvious that the chosen parameter will be a depth of 3, and one could legitimately believe that using deeper trees would increase the score. However, it seems that trees of height 5 are equivalent to trees of height 3. In fact, perhaps this maximum depth is rarely reached because of other parameters such as `min child weight` which limit the growth of each tree. This example proves that an analysis which covers several parameters is necessary and that not studying all the parameters simultaneously will certainly result in poorer performance in the end.

[2]When training, in order to better estimate the model performance, we chose to use the cross-validation method. For a given k integer, the training set will be divided into k data sets and the algorithm will be trained successively on $k-1$ of these sets and tested on the last one. This allows to obtain an average performance and to increase the reliability of the predictions.

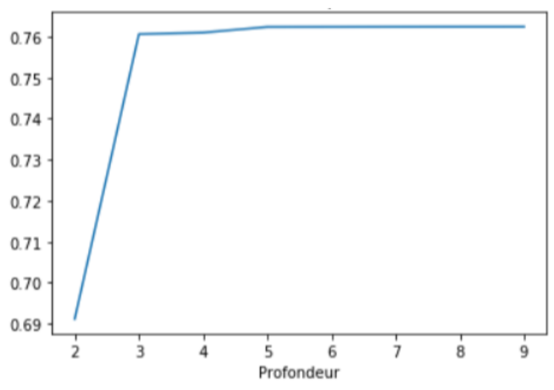


Figure 23: AUC score as a function of the maximal depth

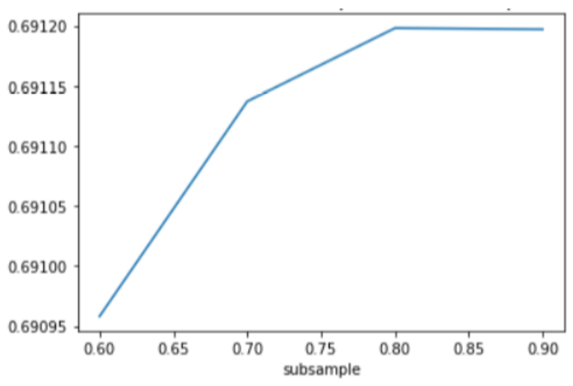


Figure 24: AUC score as a function of the `subsample` parameter

The final parameters of the model are therefore as follows:
{p = 45, `max_depth`= 4, `eval_metric` = {`'auc', 'error'` }, `learning_rate` = 0.1, `scale_pos_weight` = 45, `subsample` = 0.8, `nthreads` = -1, `colsample_bytree` = 0.8, `gamma` = 4, `nb_iterations` = 90, `objective` = `weighted_loss`, `cv` = 5}

4.3.3 • Presentation of the results

Now the selected model can be applied to the test set and its performance can be evaluated. The performance can already be presented with the ROC curve (Figure 25). We obtain an AUC score of 0.793 which gives a Gini score of 0.59 where the Gini score is given by the formula Gini = 2AUC − 1. The BNPP model obtained a Gini score of 0.72 on fairly close data. More precisely, they obtained a Gini index of 0.65 on the set qualified as "risky" and 0.42 on the set qualified as "non risky" (Paragraph 3.2).

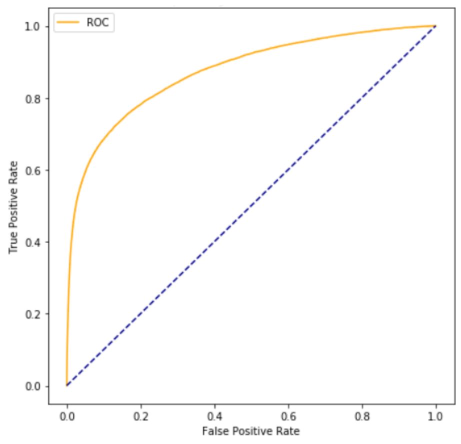


Figure 25: ROC curve of the selected model

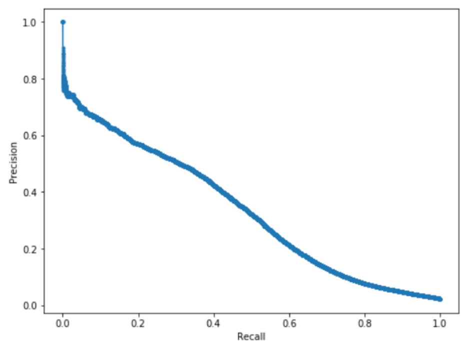
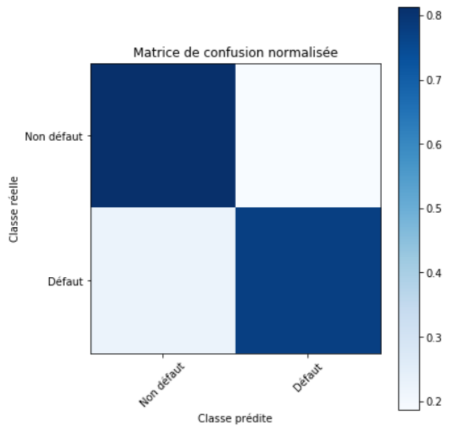


Figure 26: Precision and recall curve

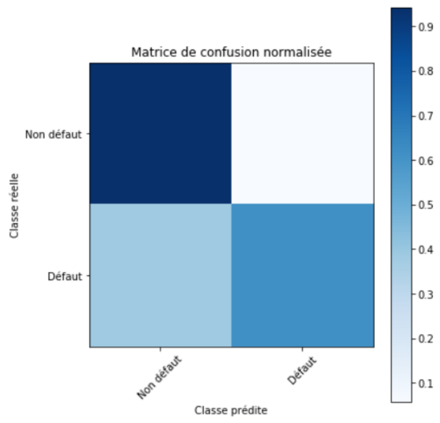
Figure 26 is the precision-recall curve. It is a representation of the couples (precision, recall) obtained for different values of the threshold. Indeed, let us recall that the algorithm returns a probability and that it is then necessary to set a threshold to obtain a classification which results from the value of the probability obtained (paragraph 4.2.1). Ideally, we would like a value of precision and recall close to 1, in which case the model differentiates well the classes and identifies all the defects.

Choice of the threshold and balance between precision and recall We present (Figures 27 and 28) the results for two different thresholds using standardized confusion matrices and the values of the associated scores. The matrix on the left is associated with the quantile of order 0.8, and the one on the right with the quantile of order 0.9 (according to the paragraph 4.3.2).



Truth / Prediction	Non default	Default
Non default	699 032	160581
Default	4459	15292

Figure 27: Normalized confounding matrix for a threshold associated with quantile 0.80



Truth / Prediction	Non default	Default
Non default	810 149	49 464
Default	7659	12092

Figure 28: Normalized confounding matrix for a threshold associated with quantile 0.90

When comparing the confusion matrices, we notice that a choice has to be made between the precision that quantifies the number of over-predicted defects and the recall that characterizes the number of undetected defects. Indeed, if the recall is too low then many companies likely to fail will not be identified. On the other hand, if the precision is too low, then many cases will be predicted to fail and the organization will need to set up an additional control to discriminate the remaining cases. The performance of these two models is shown in Figure 29.

Score	Recall	Precision	Accuracy	F1 Score
Threshold 0.80	0.78	0.09	0.81	0.16
Threshold 0.90	0.61	0.20	0.93	0.30

Figure 29: Results of the two models

Thus, it can be noted that the second model (threshold 0.90) only detects 61% of defaults but is relatively selective with a precision of 0.20. Its F1 score is therefore higher. Of course, we see here that these scores cannot be considered as absolute indicators and that the efficiency of a model depends on our objective. By choosing the first model (threshold 0.80) and in the case where the bank would rely only on the probability of default, it would have refused about one sixth of healthy loans, which represents a significant loss of earnings.

Interpretability of variables One of the advantages of XGBoost is its ability to produce interpretable results. Indeed, one can measure the importance of each of the variables in the construction of the final probability based on the structure of the decision trees. Two criteria can be used to quantify the importance of the variable:

- Weight criterion: this is the frequency of appearance of the variable in the structure of a sub-tree. The results are presented in Figure 31.
- Gain criterion: it quantifies the contribution of the variable in minimizing the loss function. When the variable is chosen at the level of a node, the loss function is evaluated before and after the construction of this node. The gain of the node corresponds to the difference between these two evaluations. The total gain is the sum of the gains of each node associated with the variable. The results are presented in Figure 30.

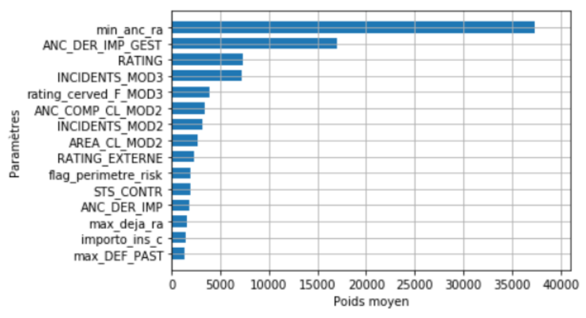


Figure 30: Importance of variables on the gain criterion

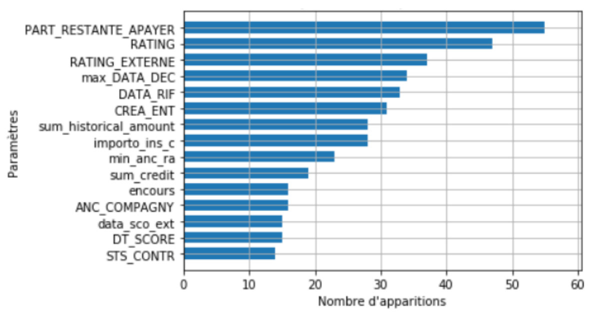


Figure 31: Importance of variables on the weight criterion

We have chosen to show the fifteen most important variables according to each of the criteria. It is interesting to note that they do not completely coincide. In particular, the variable "min_anc_ra", which represents the minimum seniority of the last contract in amicable debt recovery, is a determining factor. Intuitively, this is logical since the default of a firm occurs following a certain number of malfunctions of the firm and warnings from the bank. In the same way, it is understandable that the seniority of the last unpaid debt (the second most important variable in earnings) and the "rating" which corresponds to a mark attributed by the Italian financial authorities allow for class discrimination. We note that, according to the gain criterion, only about ten variables are important.

On the contrary, the frequency of appearance of the variables is more evenly distributed among the variables. We chose 90 iterations and a maximum depth of 4 per tree. This means that there are about 1500 possible nodes to which a variable is associated. We can notice that according to the weight criterion, the important variables are of a more global nature and concern the company but are not linked to the default, for example the date of creation of the company, the sum of the credit, or the initial sum provided. On the other hand, according to the gain criterion, the important variables are more related to credit risks, incidents and delinquencies. It should be noted that for some of the variables important in the sense of gain, we did not have values for healthy firms and therefore chose arbitrary values depending on the context. Thus, the choices we made during data cleaning) have a significant influence on the model since it uses these degraded variables. One improvement we would have liked to make is to modify the set of variables following the responses of this first model to see if we could select more relevant variables. This would allow us to reduce the computation time by targeting only the most useful variables, without degrading the performance of the model.

4.3.4 • Conclusion

In this section we have put in place a strategy to establish a model that would address the imbalance in the data and achieve good scores, in accordance with the approach described in 4.2.2. The few strategies we used (changing the loss function, adjusting the "scale pos weight" or using adequate scores) were internal to the XGBoost model. Although the final performance was correct, we looked for external methods to compensate for this imbalance. To do this, it is natural to think of algorithms to rebalance the data that act directly on the training data, either by removing entries or, on the contrary, by adding new entries from the minority class. This is exactly the principle of the SMOTE algorithm that we wanted to implement.

The SMOTE algorithm (for "Synthetic Minority Over-sampling Technique") [9] solves the problem of imbalance when a class is strongly dominant. It uses the nearest neighbor method to artificially create new entries in the minority class. The model takes as argument an integer k, the number of neighbors on which the algorithm is based and the expected class ratio. It is therefore more efficient than other naive algorithms that one could spontaneously think of and that would simply copy entries or make averages of them. It is important to specify that the SMOTE method is an external method to XGBoost which is applied directly on the data and therefore can also be used before using a neural network or another model.

The results of the XGBoost model are nevertheless satisfactory and constitute a control performance against which we were able to compare the performance of neural networks. The XGBoost algorithm, although already studied in the field of credit scoring, remains in itself a very good model that could eventually replace the models based on logistic regression currently in use. [4].

4.4 Neural networks

4.4.1 • Presentation of the model

We will use neural networks of the "Multi-Layer Perceptron" type. We present in this part the principle of such a model and its various parameters.

General principle A neural network consists of several layers (*layers*) of neurons. The first layer is called the (*input layer*), and has as many neurons as there are explanatory variables in the model (in our case, about 120 or 40 after a PCA as explained in 3.4). The next layers are the (*hidden layers*), and the last layer is called the (*output layer*).

Between two consecutive layers, each neuron i of the anterior layer is connected to each neuron j of the posterior layer. This connection is characterized by a weight $\omega_{i,j} \in \mathbb{R}$. From the activation values of the neurons of the previous layer x_i, the activation of the neuron j of the next layer is calculated as follows

$$x_j = f\left(b + \sum_i x_i \omega_{i,j}\right)$$

where f is a function (usually non-linear), called an activation function. It is common to all neurons of the j layer. b is called the bias of the j neuron. This formula is used to calculate activations from the anterior layer to the posterior layer. Thus, only the activations of the input layer are needed to calculate those of the output layer (Figure 32). The input layer activations are the values of the explanatory variables of the individual whose output is to be computed.

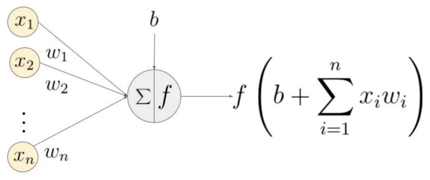


Figure 32: Functioning of a neuron.

The most common activation functions are the sigmoid and ReLU functions, defined by

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \text{et} \quad \mathrm{ReLU}(x) = \max(x, 0)$$

Training Thus, a neural network with a known number of layers, number of neurons per layer and activation functions is characterized by the weights and biases associated with the different connections. The particularity of neural networks is that these weights and biases are learned by the model with the help of examples during the training phase. The principle of learning weights and biases is iterative and is based on a loss function $\mathcal{L}(y, \hat{y})$ between the $\hat{y}$ outputs of the network and the real y classes. The easiest way to learn the weights is *the gradient descent*: the weights and biases are updated at each iteration by:

$$w \leftarrow w - \eta \nabla_\omega \mathcal{L} \quad \text{et} \quad b \leftarrow b - \eta \nabla_b \mathcal{L},$$

where η is a parameter called *learning rate*, and where the gradients are computed by an algorithm called backpropagation that we do not detail here.

There are many variants of this gradient descent algorithm that can speed up the learning of weights. We present two of them, which we have used.

- The stochastic gradient descent (SGD): this consists in not using the whole learning set during each iteration but only a *mini-batch*. The size of the mini-batch must be set, as well as the learning rate.

- Adam optimizer [10]. This is a variant of the SGD that is often more efficient. It has many parameters for which the original article advises to keep the default value. We will only set the size of the mini-batch and the learning rate.

These techniques are all iterative ones (and are not guaranteed to converge), so a number of iterations must be specified before. This number of iterations is called *epochs*. Specifically, it refers to the number of times the training has been iterated over the totality of the training data.

Regularization When training a neural network, the phenomenon of overfitting (4.3.1) can occur quickly. There are several techniques to control it.

- *Dropout* [11]: This technique consists in ignoring neurons randomly: at each iteration of training, we ignore each neuron with a probability d. d is called *dropout rate*. This disabling of neurons is only used in the training phase.

- *Batch-normalization* [12] : It consists of controlling the means and variances of neuron activation. This method makes the network more robust to data and allows the network to generalize more easily.

- *Early stopping* : This technique simply consists of stopping training when the model's performance on the validation set no longer increases.

Hyper parameters Thus, a neural network has several parameters:

- Network architecture parameters: number of layers, number of neurons in each layer, activation functions.
- Learning parameters: loss function, optimization mode, learning rate, mini-batch size, epochs.
- Parameters related to regularization techniques: dropout rate, batch-normalization.

These parameters are not learned by the model. You must therefore adjust them by trying several combinations and choosing the ones that give the best results. This is the *fine-tuning* step presented previously in 4.3.2, in the case of the XGBoost algorithm.

4.4.2 • Training and choice of hyper-parameters

An initial architecture To begin with, we use a simple architecture (Figure 33), aiming to making it more complex later on. For this, we use an architecture typical of two-class classification problems. Here is its description :

- The input layer contains 40 neurons (the number of explanatory variables of the model after PCA).
- A hidden layer that contains 120 neurons ("hidden layer"). We use the sigmoid activation function.
- An output layer. We use again the sigmoid activation function, so that the output is a real number in the interval]0.1[(corresponding to the probability of failure).

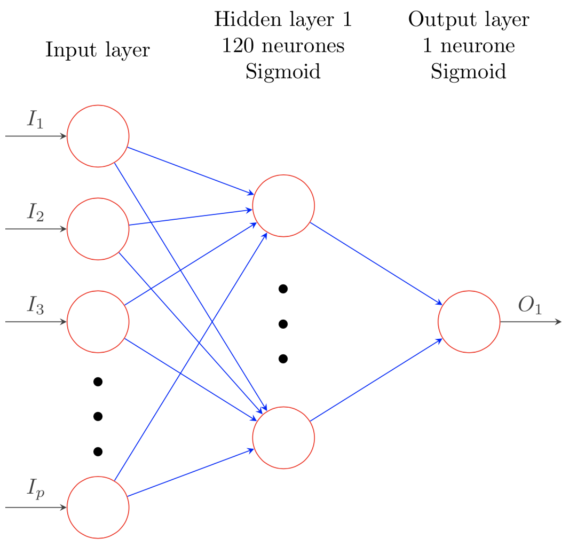


Figure 33: Initial network

We use as loss function the binary cross entropy, which is adapted to two-class classification problems. We recall (4.3.1) that it is defined by :

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i).$$

AUC	F1	Precision	Recall	Average probability of default (default)	Average probability of default (non default)
0.87	0.29	0.58	0.19	0.23	0.02

Figure 34: Results of the initial network

The results (Figure 34) show that this network does not allow to discriminate sufficiently the defects: for example it detects only 19% of them. From the point of view of the 4.2.2 part, if our first criterion, based on the AUC and F1 scores, is more or less satisfied (0.87 being a very good AUC score), our second criterion, which consisted in maximizing the recall and precision scores simultaneously, is much less so. This is logical, since the loss function, which is symmetrical with respect to the two classes, does not induce him to perform well according to this second criterion. We then decide to change the loss function.

Loss function The interest of changing the loss function is to improve the performance of the model from the point of view of the second criterion of the approach presented in part 4.2.2. Of course, we hope that these improvements do not deteriorate too much the performance on the first criteria (AUC and F1 measurements). We will see later that this is indeed the case. We could actually predict this by noting that the AUC and F1 scores are not sensitive to the difference in performance between accuracy and recall.

Let us therefore introduce an asymmetrical loss, in the same way as in paragraph 4.3.1, as well as the p parameter indicating the detection preference of the defect class (class 1)..

$$\mathcal{L}_p(y, \hat{y}) = -\sum_{i=1}^{n} p y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Our loss function is defined. Now we need to set the value of the p parameter as we did in the case of the *boosting* model. Note that p plays a slightly different role from other neural network hyper-parameters, since it directly influences the loss function. We will therefore set it first. To do so, we compare the different metrics defined in 4.2.2 for different values of p. Although this loss is identical to the one used in XGBoost, it certainly does not have exactly the same effect when learning with the neural network. This is why we need to study the influence of p on neural network performance.

Let's start with the classic metrics: F1 score, AUC score and ROC curve. As stated at the beginning of the paragraph, we do not expect any significant influence of p on these metrics.

We observe (Figures 35 and 36) that the classical metrics do not allow us to choose the p parameter. The p parameter has nevertheless the expected behavior, which can be seen in Figure 36. In this figure, for each value of p, we have plotted the maximum F1 score that

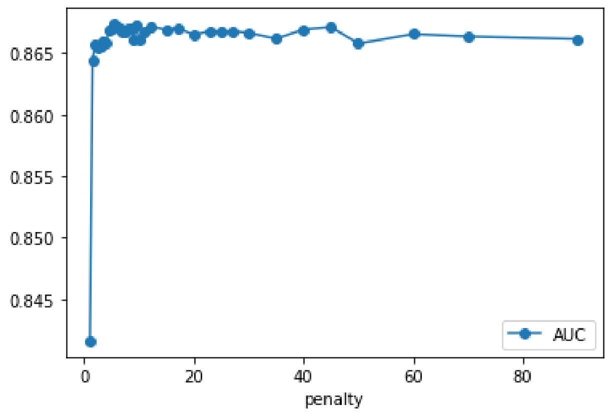


Figure 35: AUC score as a function of p

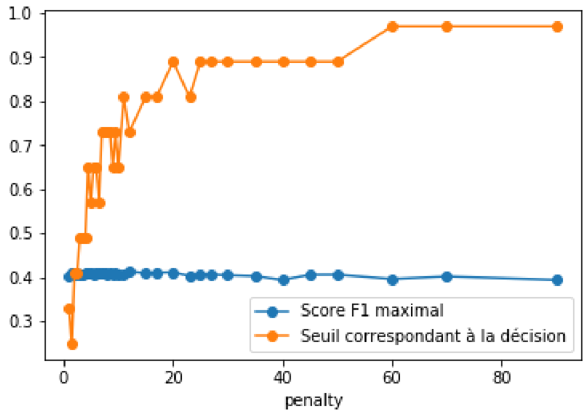


Figure 36: F1 score as a function of p

can be obtained by varying the decision threshold as well as the threshold corresponding to this maximum. We can see that the decision threshold corresponding to the maximum F1 score increases when p increases, which means that when p increases, "defaults" have a higher predicted probability of default.

Let's move on to precision, recall and average default probabilities on each class.

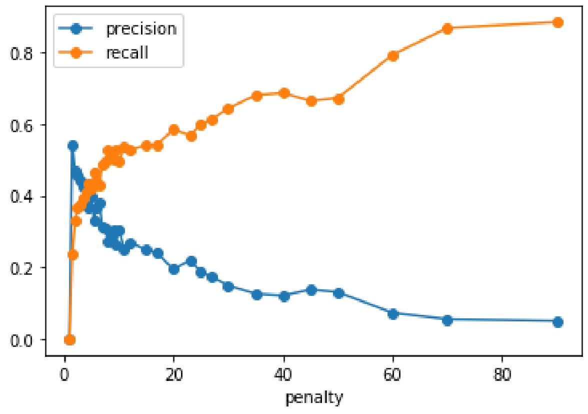


Figure 37: Precison and recall as function of p

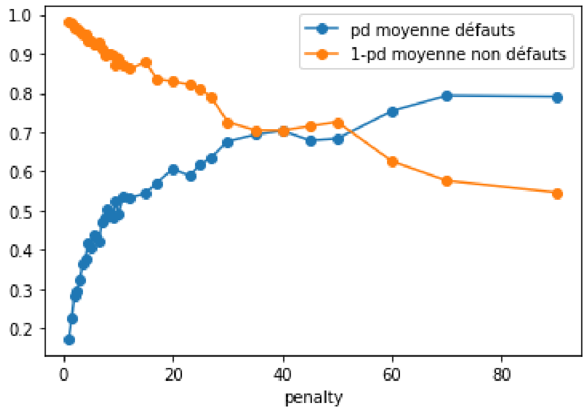


Figure 38: Average probability of default as a function of p

We observe on the figures 37 and 38 the expected behavior: when p increases, the recall increases (we observe more defects), the precision decreases (we classify more non-defects as defects), the probability of defects increases, and that of non-defects as well.

In order to choose a value for the p parameter, let's take a closer look at the figures 37 and 38. We will try to keep the discrimination between the two classes, so the value we are looking for is between 5 and 20.

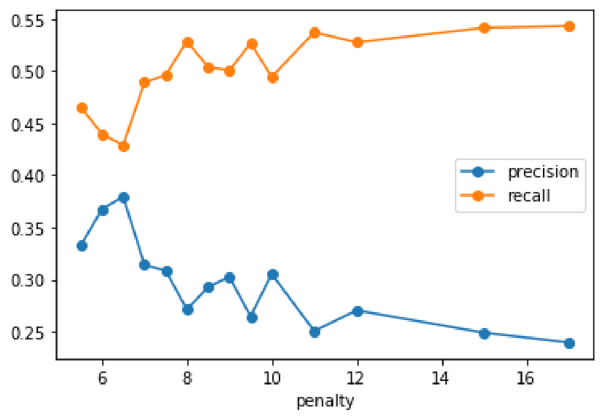


Figure 39: Precision and recall as function of p

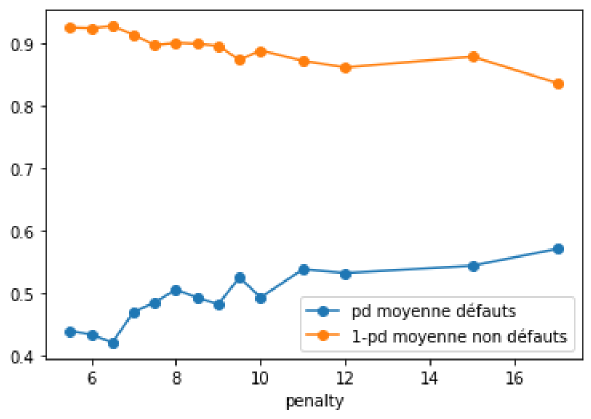


Figure 40: Average probability of default as a function of p

In the light of the figures 39 and 40, we take $p = 9$ for the rest. Indeed, we get a recall of about 50% which is quite satisfactory, while keeping a probability of defect below 10% on average for non-defects.

Network refinement, choice of hyper-parameters and fine tuning First of all, we notice that we have quite a large amount of data. We can therefore afford to use a more complex network, without risking overfitting. In [13], Jeff Heaton gives guidelines for choosing the number of layers and the number of neurons per layer. There is no theoretical reason to prefer deeper networks (with more layers) than shallower networks. That said, for the same number of parameters, the methods of the Deep Learning libraries seem to work faster on deeper networks with less dense layers. We therefore use such networks. We use a **3 hidden layers** network, with respectively **150,100 and 50 neurons**.

For **activation functions**, we use the sigmoid for the output layer. For the hidden layers, we have the choice between sigmoid and ReLU. The advantage of ReLU is that it generates sparse, because it cancels the activations of some neurons. A deficient network will be simpler and therefore probably more generalizable. Moreover, ReLU is easier to calculate than sigmoid, which speeds up training considerably. We will therefore use **ReLU for the hidden layers and sigmoid for the last layer**.

Let's go to the **batch size**. Using mini-batches makes training much faster. However, for the training to be effective, the mini-batches must be large enough to contain defaults. Otherwise, there is a risk of calculating gradients on a defect-free subset, making it impossible to train the network to recognize defaults. We will use sizes larger than 500, since there are only 2% of defects. The figure 42 allows us to choose a size of **1000 entries per batch**.

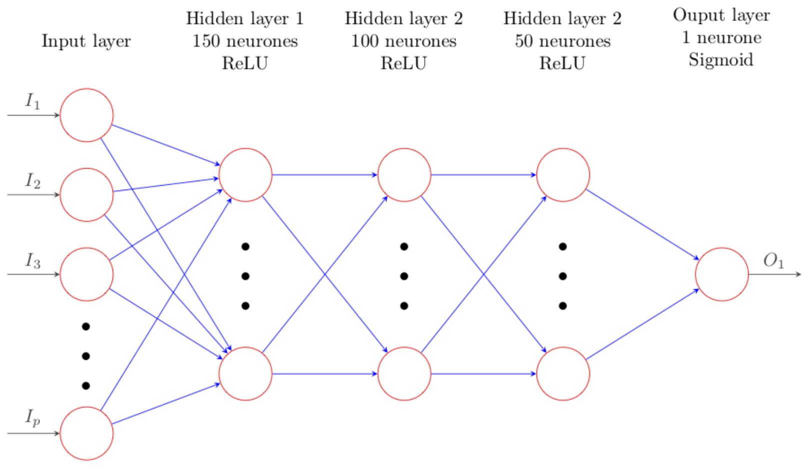


Figure 41: Chosen neural network

The **epochs** should be chosen so as to minimize loss as much as possible, while avoiding overfitting. It should also be taken into account that the greater the number of iterations, the slower the training. The figures 43 and 44 show that it is useless to go beyond 20 epochs. So we will use **20 iterations**.

Concerning the **learning rate**, it must be small enough to allow training and convergence, but too small a value slows down learning. We then plotted the learning curves (loss calculated on the training set) for a few values of the learning rate, in the cases where we use the Adam method (figure 45) and in the case of the stochastic gradient descent noted SGD (figure 46). Thus, we use a rate of 10^{-2} for Adam and 1 for SGD.

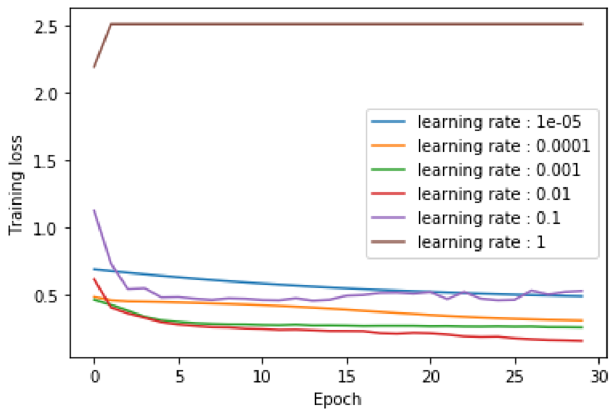


Figure 45: Loss through training(Adam)

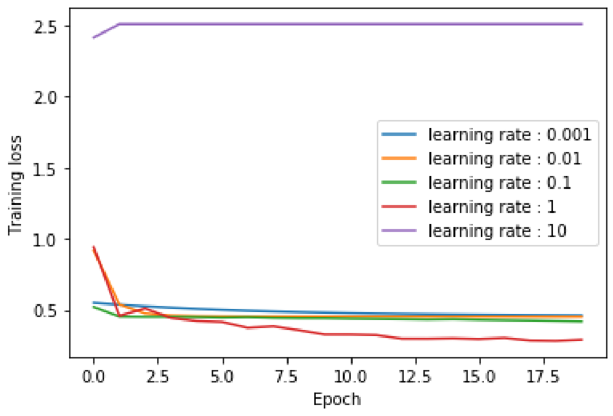


Figure 46: Loss through training (SGD)

Regularization As we can see on the figures 43 and 44, overfitting only occurs from about 50 to 60 epochs. Thus, we don't need to use *dropout* to reduce overfitting. Nevertheless, let's

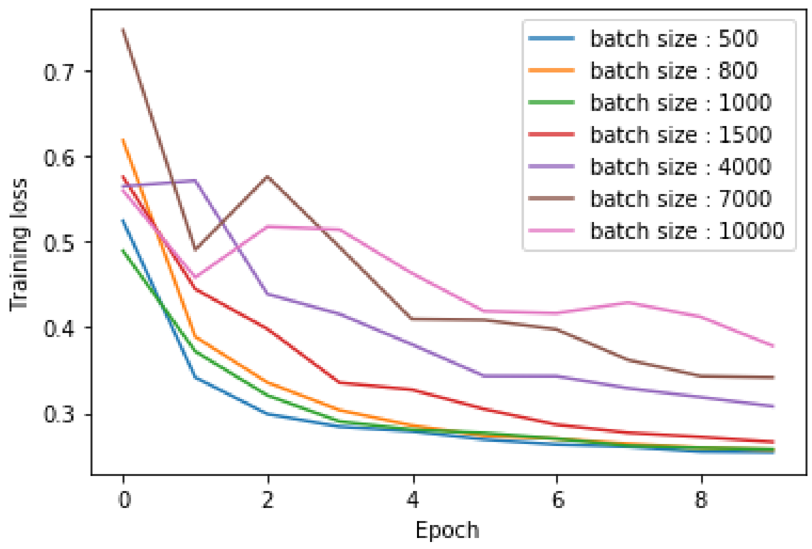


Figure 42: Evolution of the loss through the training for many batch sizes

study the **effects of dropout** on training.

The figure 47 shows that the *dropout* has a useful effect only from 60 *epochs*, since it allows the validation loss not to increase. However, for a rather low number of *epochs* (< 20), the figure 48 shows that training is better done without *dropout*. So we won't use an *dropout* for the moment.

However, it can be useful to use **batch-normalization**. Indeed, even if its role against overfitting is not going to be used, this process allows the network to be better optimized and more robust.

Selected models In view of the experiences described above, three models have been selected.

- **Model 1**: This is the model of the figure 41, with the following parameters.
  - Optimizer: Adam with a 10^{-2} learning rate
  - Batch size: 1000
  - Number of epochs : 20
- **Model 2**: This is model 1, to which is added the *batch-normalization* after each layer, trained on 20 *epochs*.
- **Model 3**: This is model 1, to which we add the *batch-normalization* and the *dropout* (with a parameter of 0.2) after each layer, trained on 30 *epochs*.

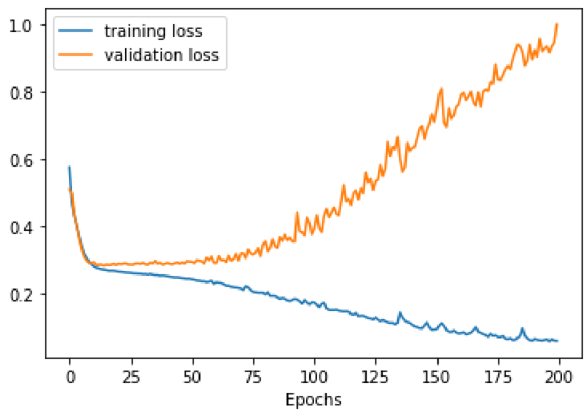


Figure 43: Loss as a function of the number of iterations

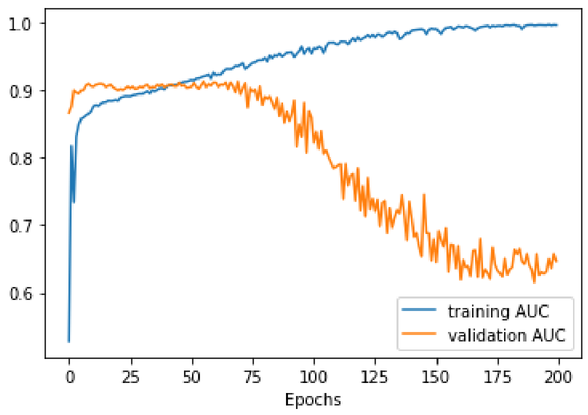


Figure 44: AUC as a function of the number of iterations

4.4.3 • Presentation of results

Let's give without further ado the results provided by the three selected models. Note that the models were trained and tested using the k-cross-validation method with $k = 5$ for more reliability on the results (4.3.2). The results obtained are summarized in Figure 49..

We did not include the class 0 non-defaults scores (AUC, recall and accuracy) since the models have about the same performance at this level (recall of 0.97, accuracy of 0.99, F1 score of 0.98).

The selected models are satisfactory from the point of view of metrics that do not take into account the difference between recall and precision (AUC, F1). Thus, in comparison with the results of the basic neural network (Figure 34), the AUC scores are similar, while the F1 score is significantly better for models 1 to 3. At the level of the AUC score, we observe an interesting phenomenon: the different neural networks have about the same score. This score is also approximately the same as the one obtained in [4]. Nevertheless, these models have radically different performances when we look more closely at the two classes. We can therefore conclude, in accordance with the remarks in 4.2.1, that the AUC score is clearly not sufficient as a performance metric for this problem. Models 1 to 3 have, contrary to the basic model, good performance in terms of fault detection, with a recall (rate of true defaults detected) of about 51%. Precision is only 30 percent, which means that when the model classifies an individual as a default, he will be right only one out of three times.

Let's move on to the comparison of models 1,2 and 3. The third model is the most complex (and requires a bit more computing time because of the *dropout*, the *batch-normalization*, and the higher number of epochs). Its performance is slightly better in terms of recall. However, it is the worst in terms of precision. Models 1 and 2 have better precision but less recall. Note that model 1 has similar performance to the other two, and it has the advantage of being the simplest (and therefore the fastest in terms of learning).

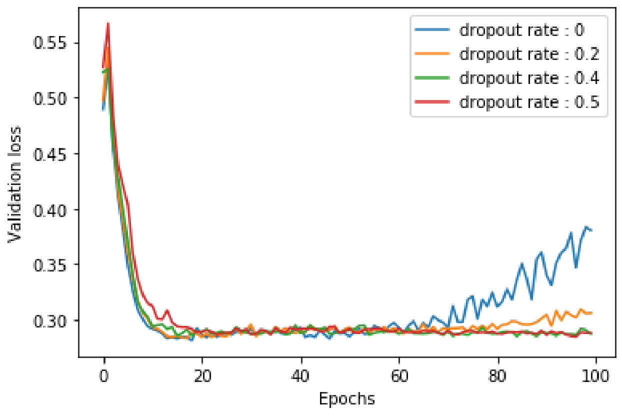


Figure 47: Validation losses for several dropout rates

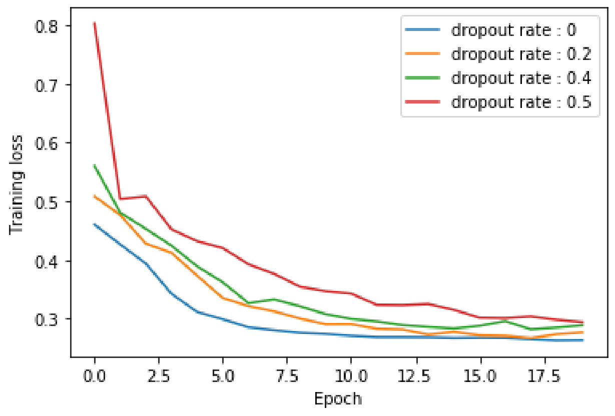


Figure 48: Training losses for several dropout rates

	AUC	F1	Precision	Recall	Average probability of default (default)	Average probability of default (non-default)
Model 1	0.87	0.38	0.30	0.51	0.50	0.10
Model 2	0.88	0.37	0.29	0.51	0.50	0.10
Model 3	0.88	0.37	0.29	0.52	0.50	0.09

Figure 49: Results of the three models

4.4.4 • Conclusion

Finally, models based on neural networks have been able to discriminate effectively between the two classes. One of the keys to this approach was the introduction of an asymmetric loss function, allowing to direct the learning of the network towards a more thorough discrimination of defects. As indicated in paragraph 4.3.4, other methods to overcome the asymmetry of the classes are possible, which could perhaps improve the performance of the model.

The models could be improved by using more data, but perhaps the main limitation is the quality of the data collected, since, as explained in 3.2, the data processing performed has made the dataset less accurate.

Let us note finally that if the neural network has shown a certain capacity to discriminate between defects and non-defects, it has the defect of being difficult to interpret, contrary to other models such as logistic regression and decision trees. Indeed, our network has about $40,000$ parameters (weight and bias), intervening between neurons that do not necessarily belong to the output or input layer. It is therefore more difficult to show the variables that were determinant in the training of the model. Interpretation methods do exist, however, but the interpretability of neural networks is still a field of research in development [14]. We have not implemented such methods in our work.

Score	AUC	Recall	Precision	F1 score
Model 1 XGB	0.79	0.78	0.09	0.16
Model 2 XGB	0.79	0.61	0.20	0.30
Model 1 NN	0.87	0.51	0.30	0.38
Model 2 NN	0.88	0.51	0.29	0.37
Model 3 NN	0.88	0.52	0.29	0.37
Model BNPP	0.86	/	/	/

Figure 50: Comparative results of the five models studied

4.5 Comparative analysis of the two approaches

Comparison of gross results We recall the main results of the models studied in the case of *gradient boosting* (XGB) and neural networks (NN) in the Table 50.

We observe that the neural network is more efficient than the XGB model in terms of AUC score. This difference is all the more relevant since it is the only score among those usually presented in a classification problem for which it is not necessary to choose a decision threshold. The neural networks that we have trained also have an AUC score that is slightly higher than the score of the BNPP logistic regression model.

It can be noticed that the modification of the loss function (weighted cross entropy) that was applied in the case of XGB and in the case of the neural network did not have the same influence in one case as in the other. It greatly improved the performance of the neural network while the improvements observed in the case of the XGB model were limited, and the application of asymmetric loss deteriorated some scores. A posteriori, we can affirm that the XGB model rather requires a re-sampling approach, such as the SMOTE algorithm that we presented in 4.3.4.

It should be remembered that in these models we did not choose exactly the same decision threshold values (although in the case of XGB the method used finally came down to taking a threshold around 0.5). This difference partly explains the differences in recall and precision scores. However, using a threshold other than 0.5 to calculate the metrics gives a probability of defect that is more difficult to read. For example, if the decision threshold is 0.1, a probability of default at 0.4 is actually very high and will be classified as a default.

Interpretability Note that the two models differ on one important point, however. Neural networks have the peculiarity of being difficult to interpret nowadays [14]. In particular, it is difficult to know the influence of each of the explanatory variables on the calculation of the final probability. The *boosting* model has the advantage of being easily interpretable. This is a very important feature of a model, especially in the context of default probability prediction. In order to be integrated into a credit scoring model, a model must be transparent and explainable. Indeed, as we explained in the introduction, the Basel regulations to which banks are subject impose transparency requirements. Logistic regression as well as *boosting* make this possible, as we saw in the 4.3.3 section.

5 GENERAL CONCLUSION

Through this PSC, we were interested in the problem of estimating the probability of default. After having qualitatively studied the data provided by the Leasing Solutions subsidiary of the BNP Paribas bank, we applied different statistical learning methods. We first used the XGBoost algorithm to obtain reference results and then we considered neural networks. The latter, more rarely studied in the context of credit risk, allowed us to achieve encouraging performances in comparison with the models used by the BNPP as well as the other models we had studied.

During this work, we were confronted with many common problems in statistical learning applied to credit risk. These were the lack of data, the excessive number of variables, the management of the asymmetry of the different classes, but also the problem of the interpretability of the results obtained. The solutions that we brought allowed the implementation of neural networks, relatively more efficient than the XGBoost method or logistic regression.

The evaluation of model performance has allowed us to question practical problems in the application of algorithms. Indeed, it is obvious that the theoretical performances of the models must be compared with the objectives set by the bank. It seems that, in the context of credit risk, a good model must allow a significant proportion of defaults to be identified. However, although from this point of view, XGBoost models are more efficient than neural networks, these latter are globally more precise. Thus, it is up to the users of these models to prioritize their requirements in practice.

Finally, as we have already mentioned, the *boosting* method is more interpretable than models using neural networks. Given this difference in behavior between the two types of models and their respective performances, we can assume that it would be interesting to use them together in a more complex model.

This group project was an enriching work experience. We were able to invest ourselves in a significant research project for which several avenues of investigation were possible. We were confronted with practical constraints that we do not encounter in the school setting. In the end, this project was an unprecedented experience of group work, in collaboration with professionals in the field.

REFERENCES

[1] Maud Aubier : Impact de bâle ii sur l'offre de crédit aux pme. *Économie & prévision*, n° 178-179:141, 01 2007.

[2] Trevor Hastie, Robert Tibshirani et Jerome Friedman : *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[3] Iain Brown et Christophe Mues : An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Syst. Appl.*, 39:3446–3453, 02 2012.

[4] Bérénice Le Perff : Documentation of italy statistical pd model retail & corporate v1.0. Rapport interne, 2016.

[5] Abdollah Rida : Machine and deep learning for credit scoring: A compliant approach. Rapport de stage, 2019.

[6] Charles Ling et Victor Sheng : Cost-sensitive learning and the class imbalance problem. *Encyclopedia of Machine Learning*, 01 2010.

[7] Tianqi Chen et Carlos Guestrin : Xgboost: A scalable tree boosting system. pages 785–794, 08 2016.

[8] Chen Wang, Chengyuan Deng et Suzhen Wang : Imbalance-xgboost: Leveraging weighted and focal losses for binary label-imbalanced classification with xgboost, 2019.

[9] Nitesh Chawla, Kevin Bowyer, Lawrence Hall et W. Kegelmeyer : Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 16:321–357, 01 2002.

[10] Diederik P. Kingma et Jimmy Ba : Adam: A method for stochastic optimization, 2014.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever et Ruslan Salakhutdinov : Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[12] Sergey Ioffe et Christian Szegedy : Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[13] Jeff Heaton : *Introduction to Neural Networks for Java, 2nd Edition*. Heaton Research, Inc., 2nd édition, 2008.

[14] Fenglei Fan, Jinjun Xiong et Ge Wang : On interpretability of artificial neural networks, 2020.