Credit Scoring par régression logistique

Le credit scoring, ou évaluation du risque-client, désigne l’ensemble des outils statistiques permettant d’évaluer la solvabilité d’un individu (et donc sa probabilité de défaut). Ce type de méthodes statistiques est très largement répandu dans les établissements de crédit pour contourner l’asymétrie d’information inhérente aux contrats de prêts. En effet, le prêteur ne connaissant pas la probabilité de défaut de l’emprunteur, il va pouvoir évaluer sa capacité à rembourser en fonction des caractéristiques de ce dernier. Il existe une grande diversité de méthodes pour construire des modèles économétriques de credit scoring. On peut notamment citer la méthode du « random forest » ou bien de régression logistique. Nous avons décidé d’implémenter cette dernière méthode en utilisant le langage C++.

Nous verrons que le credit scoring est un outil précieux dans l’aide à la décision (I), puis nous aborderons les aspects techniques de notre modèle et de notre code (II).

I – L’intérêt et l’utilisation du Credit Scoring dans l’aide à la prise de décision

L’objectif du credit scoring est de déterminer si un emprunteur fera défaut ou non sur son crédit. Le credit scoring est devenu un outil crucial pour la prise de décision dans les établissements de crédits (A). On remarquera, toutefois, que son implémentation n’est pas sans difficultés (B).

A) Objectifs du credit scoring

L’enjeu des méthodes d’évaluations est de déterminer si une entreprise ou un particulier va faire défaut sur son prêt ou non. Le risque de défaut s’évalue grâce à un score ou une note. Dès lors, le crédit scoring devient un véritable outil d'aide à la décision. Un prêt sera accordé si l’on est sur que l’emprunteur pourra rembourser, ou bien si le score se situe au-dessus du seuil préalablement fixé. En effet, si l’on utilise par exemple, les probabilités de défaut comme critère d’évaluation final (comme cela est le cas dans notre modèle), on peut se fixer un seuil et non une réponse affirmative et binaire de type oui/non. On peut, par exemple, envisager de prêter à des individus ayant une probabilité de faire défaut inférieur à 40 % sur deux ans.

L’enjeu pour le prêteur est de surmonter le problème d’asymétrie d’information auquel il fait face lorsqu’il accorde un prêt : il ne connaît pas la capacité de sa contrepartie à rembourser. Un modèle de credit scoring peut lui permettre de discriminer les « mauvais payeurs » potentiels et ainsi éviter de subir des pertes financières. D’autant plus que les règlementations financières et prudentielles sont de plus en plus exigeantes pour les établissements de credit avec l’implémentation de Bâle II[[1]](#footnote-1). L’évaluation financière des clients est devenue indispensable pour répondre à ces exigences règlementaires, mais également pour améliorer la rentabilité de ces établissements. En effet, en pouvant discriminer les profils les plus à risqués, les établissements de crédit peuvent éviter des pertes sèches pouvant s’avérer importante, notamment sur des crédits de court terme plus risqués comme les crédits revolving[[2]](#footnote-2).

B) L’utilisation du credit scoring

Afin de construire le modèle économétrique d’évaluation, on doit déterminer les variables explicatives que l’on utilisera, et on doit disposer d’une base de données fiable. Cette base de données servira à construire le modèle. Elle doit contenir deux informations : les caractéristiques pour chaque individu, et le remboursement ou non de leur prêt.

Pour que la technique d’évaluation choisie soit performante, il faut réunir plusieurs conditions et disposer des informations suivantes :

- avoir un produit de crédit (ex : l'ouverture de crédit de 2.500 €)

- L’échantillon doit être homogène. Idéalement, le prêteur mettra au point un credit scoring par type de crédit proposé. En effet, selon les caractéristiques propres à chacun d'eux (montants minima et maxima, taux d'intérêt, durées de remboursement, mode de vente - agences, web, grandes surfaces,...), le credit scoring différera

- Disposer d’un échantillon de clients ayant eu accès à ce crédit, pour lesquelles les informations personnelles ont été conservées. Ces données sont principalement collectées grâce à un formulaire de demande de crédit, imprimé ou « en ligne ». Dans la phase de construction, les décisions d'octroi sont mises en œuvre par un Comité de crédit qui travaille sans le support du score (de façon traditionnelle)

- Disposer de l'historique de remboursement de ces clients : pour réaliser un bon credit scoring, il est important de disposer dans la base de données d'un nombre suffisant de clients défaillants (n'ayant pas honorés leurs engagements), idéalement de plusieurs centaines d'unités.

Sur base de ces informations, une analyse statistique des données personnelles des clients peut être réalisée. Le travail peut se révéler complexe, puisque cette analyse doit permettre d'identifier les combinaisons de données qui sont les plus fréquentes lors de défaillances, et à partir de là, construire une grille de score qui permettra de prédire la probabilité de défaillance (versus de remboursement) de chacun des clients. On doit, ainsi, identifier les variables explicatives qui impactent le remboursement de la dette. Voici quelques exemples de variables fréquemment utilisées :

- une femme sera plus rarement défaillante qu'un homme

- une femme célibataire sera plus rarement défaillante qu'un homme célibataire

- une femme célibataire sera plus souvent défaillante qu'une femme mariée

- une mère célibataire sera plus souvent défaillante qu'un père célibataire

- la profession est déterminante : un cadre aura un risque de défaut plus faible

On remarquera que le choix de ces variables s’avère déterminant pour construire un modèle précis minimisant l’erreur de prédiction. Le credit scoring est donc un outil prédictif d'autant plus efficace que les demandeurs ont des profils proches de ceux qui composent l'échantillon d'origine. Il se révèle dès lors peu adéquat pour estimer le risque de défaillance de profils qu'il n'a pas intégrés, qu'il n'a pas « appris à reconnaître » dans sa phase de construction. Des refus peuvent naître d'une « non reconnaissance » plutôt que d'un lien effectif avec une probabilité de défaillance.

Si le credit scoring est un outil devenu indispensable, son implémentation informatique peut être effectuée de multiples manières. Nous avons décidé d’utiliser une régression logistique afin de construire notre modèle.

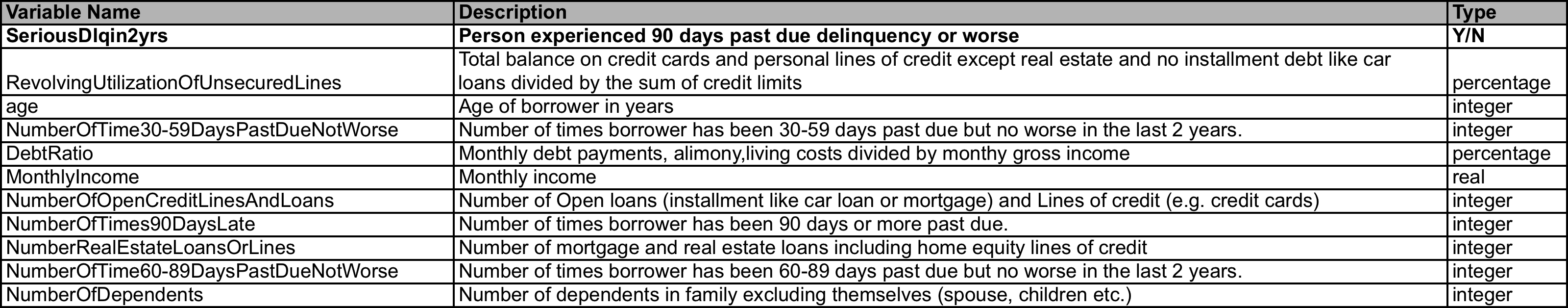
II – Utilisation de la régression logistique sous C++

Nous nous attacherons d’abord à décrire les étapes de notre algorithme (A), puis à développer la partie codage de notre projet (B). Enfin, nous tenterons d’analyser les résultats que nous obtenons ainsi que les difficultés que nous avons pu rencontrer (C).

A) Les grandes étapes de notre algorithme

Nous avons décidé d’implémenter une méthode de régression logistique pour effectuer notre analyse de risque. La régression logistique ou modèle logit est un modèle de [régression](https://fr.wikipedia.org/wiki/R%C3%A9gression) binomiale. Comme pour tous les modèles de régression binomiale, il s'agit de modéliser l'effet d'un vecteur de variables aléatoires  sur une variable aléatoire binomiale génériquement notée. C’est un cas particulier du [modèle linéaire généralisé](https://fr.wikipedia.org/wiki/Mod%C3%A8le_lin%C3%A9aire_g%C3%A9n%C3%A9ralis%C3%A9)[[3]](#footnote-3). Dans notre cas, la matrice X contient pour chaque individu (un individu par ligne) ses caractéristiques (variables explicatives). La matrice Y, contient les réponses (défaut ou non) correspondantes pour chaque individu (variable expliquée).

Nous avons utilisé une base de données déjà existante pour effectuer notre régression.[[4]](#footnote-4) Les variables explicatives, dans notre cas, sont :



https://www.financite.be/fr/reference/credit-scoring-une-approche-objective-dans-loctroi-de-credit

La première ligne de ce tableau (en gras) correspond à la réponse (oui ou non) sur la défaillance de l’emprunteur. La réponse « oui » étant codée par un 1, et la réponse « non » par un 0.

Il est à noter que nous allons incorporer une constante au modèle afin d’en améliorer la précision. Nous ajouterons donc à notre matrice X, une colonne ayant pour unique valeur : 1.

Nous utilisons une interface graphique QT afin de rendre notre outil plus simple à utiliser pour un utilisateur final. Dans cette interface, l’utilisateur entre le fichier CSV contenant les caractéristiques pour chaque individu (variables explicatives : matrice X), un autre contenant les réponses concernant la défaillance pour chaque individu de l’échantillon (réponses : matrice Y), et enfin, un dernier fichier contenant une matrice d’individus à tester (celle-ci contenant les mêmes variables explicatives que notre fichier d’entraînement : X). Les deux premiers fichiers vont servir à construire le modèle économétrique qui sera ensuite appliqué au troisième. Enfin, nous allons remplir une matrice contenant la probabilité de défaut pour chaque individu qui sera exportée sous format csv.

La probabilité de défaut se calcule ainsi :



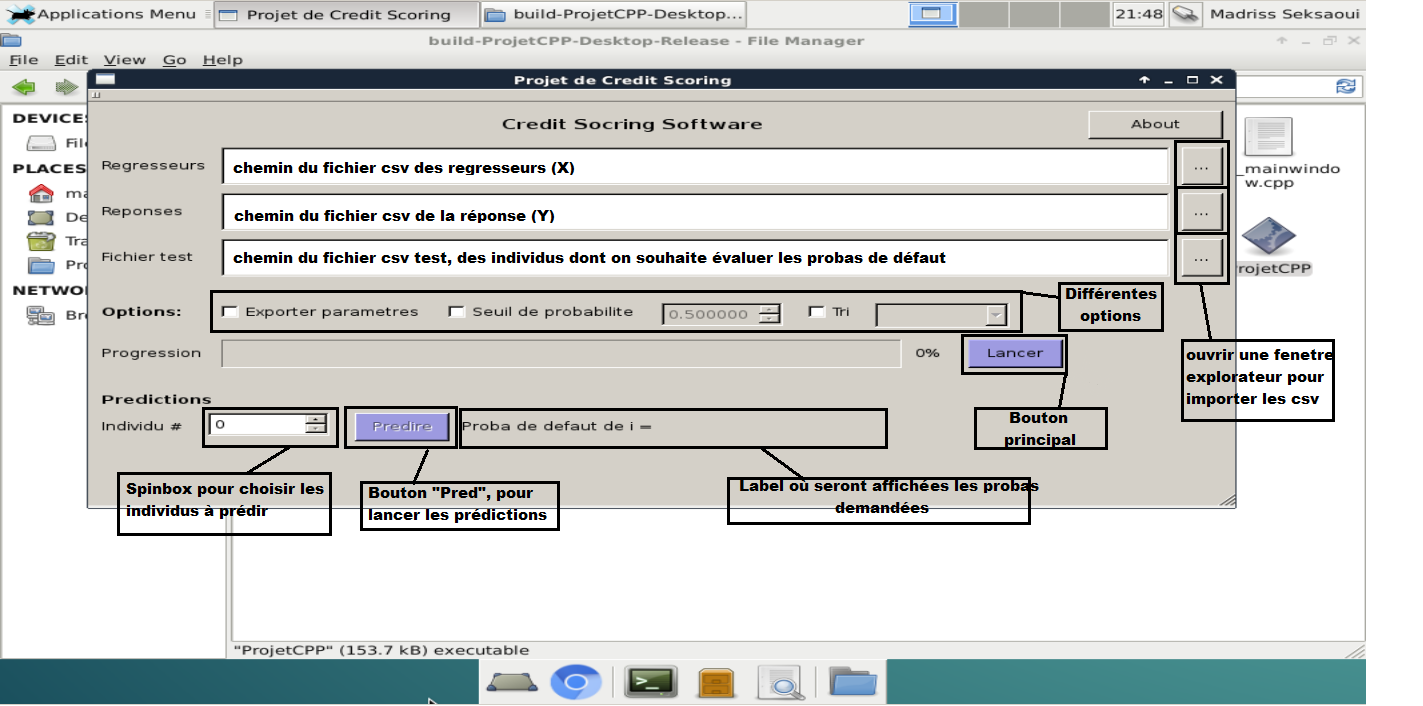
Ou sous forme matricielle :

Les répresseurs sont contenus dans le vecteur. Pour obtenir celui-ci, le calcul matriciel que nous effectuons est le suivant :

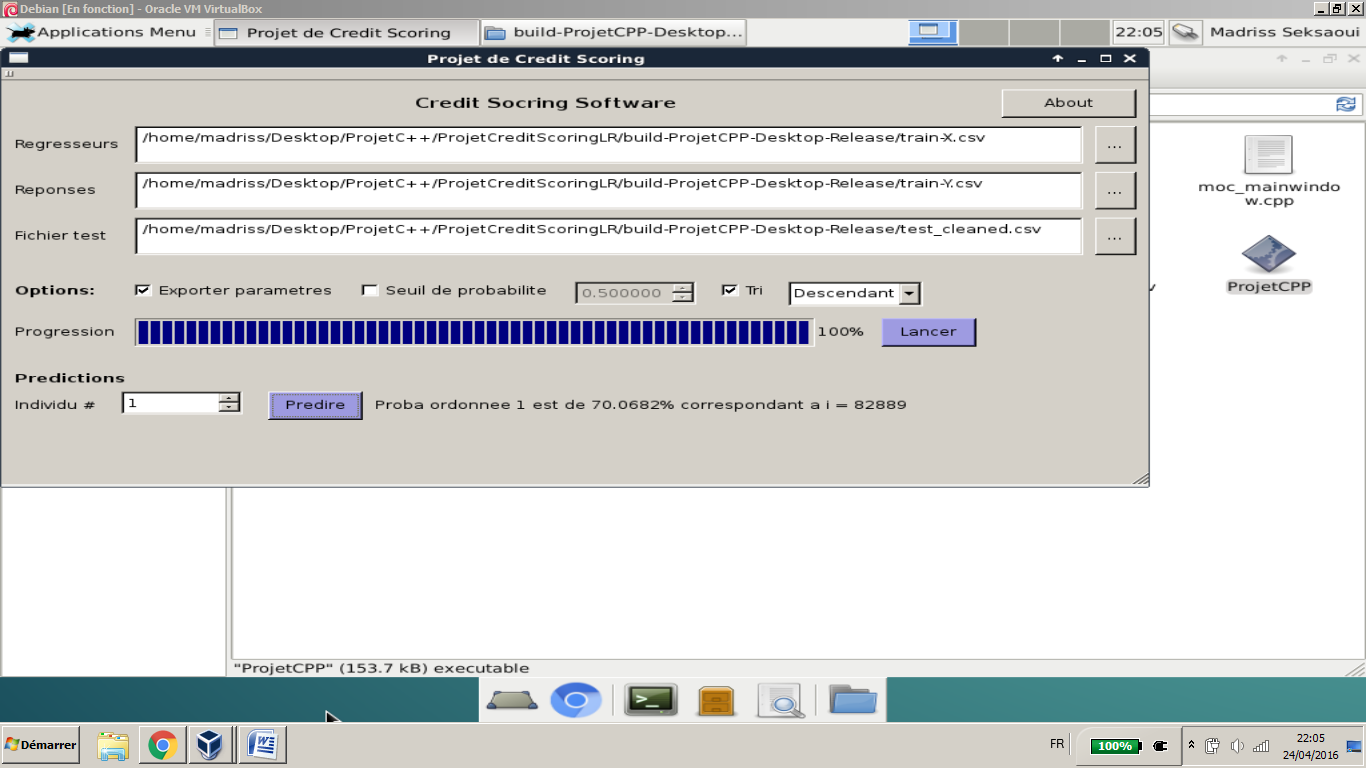
= (XTX)-1XTY

Il est de dimension (11,1) puisqu’il y a dix variables explicatives plus une constante.

Une fois le modèle construit et les probabilités calculées. L’utilisateur peut enregistrer ces probabilités dans un nouveau fichier CSV (probas.csv ou un autre nom de son choix), il a aussi le choix d’exporter les paramètres du modèle (matrice béta) sous format csv (param.csv ou un autre nom de son choix) s’il le souhaite en cochant une case. Il peut également sélectionner un individu à tester par son numéro de ligne, qu’il sélectionnera dans une spinbox dans l’interface graphique, afin de connaître sa probabilité de défaut.

****

Exemple d’utilisation :



B) Les détails technique du code de notre projet

Afin de pouvoir utiliser QT (interface graphique) nous avons inclus les librairies nécessaires à son utilisation. Il en est de même pour la librairie « mlpack » contenant une sous librairie « armadillo » qui va nous permettre d’effectuer les calculs matriciels décrits plus haut.

#include "mainwindow.h" // ligne générée automatiquement par QT pour afficher l'interface graphique

#include "ui\_mainwindow.h" // ligne générée automatiquement par QT pour afficher l'interface graphique

#include <mlpack/core.hpp> // mlpack : librairie de machine learning qui contient armadillo (librarie de calcul matricielle)

#include <QMessageBox> // librairie de QT pour afficher des msgbox

#include <QInputDialog> // librairie QT pour les inputbox

#include <QFileDialog> // librairie de QT pour ouvrir une fenêtre explorateur de fichier

#include <math.h> // pour utiliser la fonction exponentielle

#include <string> // on manipule des chaînes de caractère

#include <fstream> // pour la lecture et l’écriture de csv

#include <vector> // pour tableau dynamique

Armadillo est une librairie de calcul matricielle avancée capable d’utiliser des matrices dynamiques. Nous avons donc définit des matrices en début de code (en variables globales) afin de pouvoir les utiliser par la suite.

//Déclaration des variables globales

arma::mat parameters; // paramètres estimés par le modèle de regression linéaire (correspond au )

arma::mat P; // matrice des individus à tester dont on cherche à déterminer la probabilité de défaut

int individu; // variable globale contenant le numero de l'individu à tester

arma::mat Probas; // matrice avec les probas de chaque individus à tester

double seuil\_proba; //contient la valeur du spinner

Nous avons, ensuite, écris plusieurs fonctions auxiliaires nous permettant de faire nos calculs. Le fait d’utiliser des matrices dynamiques et la librairie Armadillo permettent de facilement effectuer ces calculs.

La fonction « probas\_logistique\_regression () » permet d’appliquer la formule de calcul de la probabilité de défaut décrite plus haut.

Liste des fonctions auxiliaires utilisées :

* **double string\_to\_double(std::string texte);** // pour la conversion de strings en double, cette fonction est utilisée pour vérifier la présence d’entêtes dans les fichiers csv
* **void Nettoyer\_csv(std::string nom\_fichier);** // fonction pour nettoyer les fichiers csv : en supprimant les entêtes si elles existent, afin d’avoir des csv contenant uniquement des nombres
* **void Verif\_csv(std::string nom\_fichier);** // fonction qui vérifie la présence d’entête, en essayant de convertir le premier lexème du fichier csv (chaine de caractère entre deux virgules), s’il y’a une erreur, on nettoie le fichier avec Nettoyer\_csv();
* **mat arrondi\_seuil(mat A, int colonne, double seuil);** // fonction qui arrondi tous les éléments d’une colonne de matrice à 1 s’ils sont au dessus d’un seuil et à 0 s’ils sont en dessous
* **mat Ajout\_col\_num\_lignes(mat A);** // fonction qui va ajouter une colonne à la matrice des probabilités afin de leur attribuer un numéro de ligne
* **mat Ordonner\_matrice(mat A,bool ordre);** // fonction qui ordonne les éléments d’une matrice par ordre croissant ou décroissant. On l’utilise car il est intéressant dans la prise de décision d’avoir une matrice de probabilités ordonnée.
* **mat Multiplier\_Matrices(mat A, mat B);** // fonction pour multiplier deux matrices entre elles
* **mat Inverse\_Mat(mat A);** // fonction pour inverser une matrice
* **mat probas\_logistique\_regression(mat X, mat Betas);** // fonction qui va calculer les probabilités par la formule

**Note :** Le détail du code de chaque fonction se trouve en annexes.

Nous avons ensuite attribué pour chaque bouton de l’interface QT un code permettant à l’utilisateur d’utiliser notre outil de prédiction :

Le bouton « **Lancer » :** permet de lancer la régression logistique après avoir chargé les fichiers d’entrainements (respectivement dans les matrices X et Y), et le fichier à tester (dans la matrice P). La librairie armadillo permet d’importer directement des fichiers csv et de créer des objets arma::mat. Cependant il faut préalablement nettoyer les fichiers si ceux-ci possèdent des entêtes. Les fichiers à importer doivent être impérativement au format csv et ne contenir que des nombres. D’où l’utilisation de la fonction : Verif\_csv et nettoyer\_csv.

La seconde partie du code consiste à calculer les coefficients des régresseurs que nous allons stocker dans la matrice « parameters ». Pour améliorer l’expérience utilisateur, nous avons inséré une barre de progression qui évolue avec l’avancement des calculs. Celle-ci avance de 25% à chaque fois qu’un des calculs est fini. Ce rythme de progression a été défini arbitrairement.

Une fois les probabilités calculées, il est demandé à l’utilisateur s’il veut exporter la matrice des probabilités dans un fichier CSV. On utilise un objet Qmessagebox (noté msgbox) afin d’utiliser une boîte de dialogue, ainsi qu’une inputbox afin de demander à l’utilisateur de choisir un nom de fichier. On notera que le logiciel vérifie que le nom de fichier choisi comporte une extension csv, si tel n’est pas le cas l’extension est ajoutée automatiquement.

**Remarque :** l’utilisateur a de nombreuses options à sa disposition :

* trier la matrice finale des probabilités pour l’aider dans son processus de décision, en effet il pourra ainsi agir en priorité sur les individus avec les probas les plus faibles (ou fortes).
* fixer un seuil à partir du quel les probabilités sont arrondies à 1 si le seuil est dépassé et 0 sinon.
* Exporter les paramètres du modèle : et exporter les probabilités estimées sous format csv.

Toutes ces options sont simples d’accès et d’utilisation, il suffit de cocher les cases correspondantes dans l’ui et de modifier les spinboxes correspondantes.

**Code commenté du bouton « Lancer » :**

void MainWindow::on\_Lancer\_clicked() // code du bouton "Lancer"

{

bool ok; // booléen à utiliser pour l'inputbox

QMessageBox msgbox; // Creation d'un objet QMessageBox pour afficher des msgbox

// Les variables contenant les chemins des fichiers nécéssitent la conversion du QString en String, car l’utilisateur choisis les chemins dans l’ui

std::string chemin\_X = ui->Regresseurs->toPlainText().toStdString();

std::string chemin\_Y = ui->Reponses->toPlainText().toStdString();

std::string chemin\_Test = ui->Fichier\_test->toPlainText().toStdString();

arma::mat X; // Matrice des regresseurs

arma::mat y; // Matrice des réponses

// Programmation défensive : obliger l'utilisateur à choisir des fichiers à importer

if(chemin\_X==""|| chemin\_Y=="" || chemin\_Test==""){

msgbox.setText("Veuillez renseigner les champs demandes, operation impossible!");

msgbox.setWindowTitle("Impossible de continuer, champs vides");

msgbox.setIcon(QMessageBox::Critical);

msgbox.exec();

exit(0); // le programme ne peut continuer avec des fichiers vide

}

//nettoyage des entêtes s'il y'en a

Verif\_csv(chemin\_X);

Verif\_csv(chemin\_Y);

Verif\_csv(chemin\_Test);

X.load(ui->Regresseurs->toPlainText().toStdString()); // On importe la matrice qui se situe dans la textEdit puis on convertit le format QString en String

y.load(ui->Reponses->toPlainText().toStdString()); // On importe la matrice qui se situe dans la textEdit puis on convertit le format QString en String

P.load(ui->Fichier\_test->toPlainText().toStdString()); // On importe la matrice qui se situe dans la textEdit puis on convertit le format QString en String

// Ajout de la constante dans le modèle

vec Vecteur\_Unitaire = ones <vec>(X.n\_rows);

X.insert\_cols(0,Vecteur\_Unitaire);

// Obtenir les paramètres, ou coefficients **:**

parameters = Multiplier\_Matrices(X.t(),X); // opération

ui->progressBar->setValue(25); // 25 car on a fait 1/4 des opérations

parameters = Multiplier\_Matrices(Inverse\_Mat(parameters),X.t());//

ui->progressBar->setValue(50); // 50 car on a fait 1/2 des opérations

parameters = Multiplier\_Matrices(parameters,y); //

ui->progressBar->setValue(75); // 75 car on a fait 1/2 des opérations

Probas = probas\_logistique\_regression(X,parameters); //

ui->progressBar->setValue(100); // 100 car fin des opérations

// Ajout du numéro de lignes avant d'ordonner les matrices

Probas = Ajout\_col\_num\_lignes(Probas);

// Ordonner matrice proba si case cochée

if (ui->Trier->isChecked()){ // Si la case est cochée

if(ui->Ordre->currentText().toStdString()=="Ascendant"){ // Si l’utilisateur veut un classement par ordre croissant, .toStdString est utilisé pour pouvoir comparer le QString à du String standard

Probas = Ordonner\_matrice(Probas,1); // Fonction pour ordonner avec 1 qui correspond à ascendant (croissant)

}

if(ui->Ordre->currentText().toStdString()=="Descendant"){//Si ordre décroissant choisi, .toStdString pour pouvoir comparer le QString à du String standard

Probas = Ordonner\_matrice(Probas,0); // fonction pour ordonner avec 0 qui correspond à descendant (décroissant)

}

}

// Arrondi si case seuil coché

seuil\_proba = ui->seuil->value(); // on récupère la valeur seuil du spinner

if (ui->Probas\_Seuil->isChecked()==true){ // Si case arrondi cochée

Probas = arrondi\_seuil(Probas,0,seuil\_proba); // fonction pour arrondir la colonne 0 de la matrice, la colonne 1 contient les numéros de ligne

}

msgbox.setText("Les probas ont ete correctement estimees, exporter sous format csv?");

msgbox.setStandardButtons(QMessageBox::Yes|QMessageBox::No); // la msgbox aura deux boutons : Yes et No

msgbox.setDefaultButton(QMessageBox::Yes); // bouton par défaut : Yes

msgbox.setIcon(QMessageBox::Question); // on choisit une icone pour la msgbox

msgbox.setWindowTitle("Estimation des probas reussie !");

int ret = msgbox.exec(); // appuyer sur un bouton va renvoyer un entier qui sera stocké dans ret

switch(ret){ // Selon le bouton clické (dont la valeur sera dans ret) on effectue une action

case QMessageBox::Yes:{ // accolade pour déclarer des variables dans ce case

// On enregistre la matrice des probas

// code pour créer un QInputDialog

QString nom\_fichier\_probas = QInputDialog::getText(this, tr("Nom fichier matrice probas"),tr("Veuillez choisir un nom de ficheir a exporter pour la matrice des probabilites"),QLineEdit::Normal, QDir::home().dirName(), &ok);

// programmation défensive pour eviter les noms de fichiers vide ou sans extensions csv

if (ok && nom\_fichier\_probas.isEmpty()){

nom\_fichier\_probas = "probas.csv";

}

if (nom\_fichier\_probas.right(4) != ".csv"){

nom\_fichier\_probas.append(".csv");

}

Probas.save(nom\_fichier\_probas.toStdString(),csv\_ascii); // on enregistre les probas sous un fichier csv, on précise le format avec csv\_ascii

msgbox.setText("Exportation effectuee sous" +nom\_fichier\_probas);

msgbox.setStandardButtons(QMessageBox::Ok);

msgbox.setWindowTitle("Exportation reussie !");

msgbox.setIcon(QMessageBox::Information);

msgbox.exec();// pour faire sortir le msgbox

break;

}

case QMessageBox::No:

msgbox.setText("Les probas ont ete estimees mais pas exportees");

msgbox.setStandardButtons(QMessageBox::Ok);

msgbox.setWindowTitle("Probas non exportees");

msgbox.setIcon(QMessageBox::Information);

msgbox.exec();// pour faire sortir le msgbox

break;

}

// exportation des paramèetres si la checkbock est cochée

if (ui->Export\_Params->isChecked()){

// code pour créer une QInputDialog

QString nom\_fichier\_params = QInputDialog::getText(this, tr("Nom fichier parametres"), tr("Veuillez choisir un nom de ficheir a exporter pour la matrice des parametres"),QLineEdit::Normal, QDir::home().dirName(), &ok);

// programmation défensive pour éviter les noms de fichiers vides ou sans extension csv

if (ok && nom\_fichier\_params.isEmpty()){

nom\_fichier\_params = "probas.csv";

}

if (nom\_fichier\_params.right(4) != ".csv"){

nom\_fichier\_params.append(".csv");

}

parameters.save(nom\_fichier\_params.toStdString(),csv\_ascii);

msgbox.setText("Les parametres ont aussi ete sauvegardes dans le repertoire courant sous parametres.csv");

msgbox.setWindowTitle("Parametres enregistres");

msgbox.exec();

}

ui->Pred->setEnabled(1); // on active le bouton permettant de faire des prédictions car les probabilités ont été estimées

ui->individu\_i->setMaximum(Probas.n\_rows); // on fixe le maximum de la spin box

}

Après toutes ces opérations, le bouton permettant de faire une prédiction sur individus : le bouton « Pred » est activé. Il permet de renvoyer la probabilité de défaut de l’individu i qui est stocké dans la matrice « Probas ». On remarquera qu’il faut toujours tenir compte des conversions en QString lors de l’utilisation de QT et de l’affichage dans l’ui.

En effet, le type QString, est un type de variable spécial utilisé par QT pour l’affichage de caractères sur l’interface graphique.

**Code commenté du bouton « Pred » :**

void MainWindow::on\_Pred\_clicked() // code du bouton Pred

{

individu = ui->individu\_i->value(); // récupérer le contenu de la spinbox

double proba\_i = Probas(individu,0); //on récupère la probabilité correspondant au numéro de la spinbox

QString doubleAsString = QString::number(proba\_i\*100, 'd',4); // Conversion d'un double en QString affin de l'afficher dans QT (label)

QString intAsString = QString::number(Probas(individu,1));// Obtenir le num de l'individu avant que la matrice soit ordonnée pour l'afficher dans le label

QString individu\_i\_QString = QString::number(individu);// conversion du contenu de la spinbox en QString pour l'afficher dans le label

if ((ui->Trier->isChecked()== false)|| (ui->Probas\_Seuil->isChecked()== true)){ // Si le bouton trier n’est pas coché on n’affiche pas les anciens numerous de lignes car ils corresnpodent à la spinbox

ui->label\_7->setText("Proba de defaut de l'individu "+individu\_i\_QString+" est de "+doubleAsString+"%"); // affichage de la variable de type QString dans l'interface

}

else if(ui->Trier->isChecked()== true){// le bouton trier est coché on affiche les numéros des individus après tri

ui->label\_7->setText("Proba ordonnee "+individu\_i\_QString+" est de "+doubleAsString+"%"+" correspondant a i = " +intAsString); // affichage de la variable de type QString dans l'interface

}

}

C) Les difficultés rencontrées et les résultats obtenus

Les principaux obstacles que nous avons dû surmonter sont l’utilisation de l’interface QT et les calculs matriciels.

Tout d’abord, l’utilisation de QT a requit de comprendre toutes les subtilités de cette librairie. Il a notamment fallu comprendre l’impossibilité pour QT de lire directement des variables string et donc la nécessité de toujours passer par des conversions en QString. De même, pour afficher des messages (de type boîte de dialogue) il a fallu créer un objet spécifique QMessageBox.

Ensuite, le calcul matriciel indispensable à la régression logistique a demandé beaucoup de réflexion, notamment pour l’inversion d’une matrice. Nous avons surmonté ce problème en utilisant des matrices dynamiques et la librairie armadillo. Ainsi, l’utilisateur pourra plus facilement tester des échantillons d’emprunteur de tailles différentes (en gardant évidemment la même structure des caractéristiques). L’intérêt du dynamisme est également qu’il permet de réduire le nombre de fonctions auxiliaires. Ainsi, une seule fonction « multiplier » est nécessaire même si les opérations matricielles ne portent pas à chaque fois sur des matrices de mêmes dimensions.

En ce qui concerne nos résultats, nous arrivons à des probabilités de défaut pour des emprunts sur deux ans, proche de 50 % dans une majorité de cas. Cette estimation est relativement proche de celle obtenu avec les meilleures méthodes économétriques (notamment celle de random forest). On notera que la précision de ces méthodes reste loin de la perfection. En effet, les prédictions ne dépassent pas un degré d’exactitude de 70 %.

Conclusion

Nous avons implémenté un outil de credit scoring dont le but est l’aide à la décision. La question que se pose l’utilisateur est la suivante : dois-je oui ou non accorder un crédit à cet individu ? Notre outil va l’aider dans sa décision en lui renvoyant une probabilité de défaut en fonction des caractéristiques de sa contrepartie. Nous avons choisie d’implémenter une méthode économétrique appelé régression logistique. Cette méthode est relativement efficace au vu des résultats obtenus, même si les prédictions restent imparfaites. Enfin, on notera que la régression logistique (et notre logiciel) peuvent s’utiliser pour d’autres types de problèmes tels que : la détection de fraude, la détection de blanchiment de fonds, ou tout autre tâche demandant une classification des individus ou demandant l’estimation de probabilités …

**Annexes :**

Code des fonctions auxiliaires :

double string\_to\_double(std::string texte){ // conversion string en double

return std::stod(texte.c\_str(),NULL);

}

void Nettoyer\_csv(std::string nom\_fichier){// on nettoie les entêtes

std::fstream fichier;

std::vector <std::string> tableau; //vecteur dynamique de string

std::string ligne;

QMessageBox msgbox;

// on avertit l'utilisateur qu'une entête a été détectée et est en cours de nettoyage

msgbox.setText("Entete detectee dans :\n\n"+ QString::fromStdString(nom\_fichier)+"\n\nNettoyage en cours ..."); // string en QString pour afficher dans la msgbox

msgbox.setWindowTitle("Nettoyage des entetes en cours");

msgbox.setIcon(QMessageBox::Warning);

msgbox.exec();

int i = 0;

fichier.open(nom\_fichier,ios::in);

while(getline(fichier,ligne)){ // on enregistre tout le fichier dans un tableau

tableau.push\_back(ligne);

i++;

}

fichier.close();

fichier.open(nom\_fichier,ios::out);

for (int k = 1; k < i;k++){ // i contient le nombre total de lignes

fichier << tableau[k] << std::endl;

}

fichier.close();

msgbox.setText("Nettoyage termine avec succes !");

msgbox.setWindowTitle("Nettoyage reussi");

msgbox.setIcon(QMessageBox::Information);

msgbox.exec();

}

void Verif\_csv(std::string nom\_fichier){//on verifie la présence de texte dans la première ligne pour la nettoyer

std::fstream fichier;

std::string lexeme;

fichier.open(nom\_fichier,ios::in);

while(getline(fichier,lexeme,',')){ // ajout de ',' pour lire entre les virgules

try{

string\_to\_double(lexeme); // on tente de convertir le string en nombre

}

catch(const std::invalid\_argument& e){// si une erreur « invalid\_argument » est rencontrée dans try (impossibilité de convertir le string en double), on ferme le fichier et on le nettoie en supprimant la première ligne, car cela signifie qu’il y’a un header

fichier.close();

Nettoyer\_csv(nom\_fichier);

return;

}

}

fichier.close();

return;

}

mat arrondi\_seuil(mat A, int colonne, double seuil){ // arrondi à 1 si élement de la matrice > seuil, arrondi à 0 sinon

int nb\_lignes = A.n\_rows;

for (int i = 0; i < nb\_lignes; i++){

if(A(i,colonne) < seuil){

A(i,colonne) = 0;

}

else{

A(i,colonne) = 1;

}

}

return A;

}

mat Ajout\_col\_num\_lignes(mat A){ // ajout d’une colonne pour y mettre les numéros de lignes si on trie

vec Vecteur\_nul = zeros(A.n\_rows); // création d'un vecteur de zéros, qui sera rempli avec les numéros de lignes

A.insert\_cols(1,Vecteur\_nul);

return A;

}

mat Ordonner\_matrice(mat A,bool ordre){ // fonction pour ordonner une matrice

uvec indices; // on crée une variable de type uvec, obligatoire pour utiliser la fonciton sort\_index

vec colonne\_a\_ordonner = A.col(0); // création d'une variable vecteur, contenant la colonne à ordonner

int nb\_lignes = A.n\_rows;

if(ordre == 1){

indices = sort\_index(colonne\_a\_ordonner,"ascend");//fonction qui donne les anciens indices de lignes après avoir été ordonné

A = sort(A,"ascend");

for (int i = 0; i < nb\_lignes; i++){ // boucle pour attribuer les bons numéros de lignes à la colonne de zéros

A(i,1)=indices(i);

}

return A;

}

else{ // pareil qu'avant mais ordre décroissant

indices = sort\_index(colonne\_a\_ordonner, "descend");//fonction qui donne les anciens indices de lignes après avoir été ordonné

A = sort(A,"descend");

for (int i = 0; i < nb\_lignes;i++){ // boucle pour attribuer les bons numéros de lignes à la colonne de zéros

A(i,1)=indices(i);

}

return A;

}

}

mat Multiplier\_Matrices(mat A, mat B){ // fonction pour multiplier deux matrices entre elles

int nb\_colsA = A.n\_cols;

int nb\_lignesB = B.n\_rows;

QMessageBox msgbox;

if(nb\_colsA != nb\_lignesB) // programmation défensive

{

msgbox.setText("Multiplication matricielle impossible, mauvaises dimensions");

msgbox.setWindowTitle("Impossible de multiplier les matrices");

msgbox.setIcon(QMessageBox::Critical);

msgbox.exec();

exit(0); // ferme tout le programme pour éviter d'avoir des bugs après

}

return A\*B;

}

mat Inverse\_Mat(mat A){ // fonction pour inverser une matrice

QMessageBox msgbox;

double determinant = det(A);

if(determinant == 0){ // programmation défensive

msgbox.setText("Inversion impossible, determinant nul");

msgbox.setWindowTitle("Inversion impossible");

msgbox.setIcon(QMessageBox::Critical);

msgbox.exec();

exit(0); // quitter le programme pour éviter les erreurs

}

return inv(A);

}

mat probas\_logistique\_regression(mat X, mat Betas){ // permet d’avoir la matrice des probabilités

return 1/(1+exp(-Multiplier\_Matrices(X,Betas)));

}

1. Les [normes](https://fr.wikipedia.org/wiki/Norme) Bâle II (le second accord de [Bâle](https://fr.wikipedia.org/wiki/B%C3%A2le)) constituent un dispositif prudentiel destiné à mieux appréhender les [risques](https://fr.wikipedia.org/wiki/Risque) [bancaires](https://fr.wikipedia.org/wiki/Banque) et principalement le [risque de crédit](https://fr.wikipedia.org/wiki/Risque_de_cr%C3%A9dit) ou de contrepartie et les exigences, pour [garantir un niveau minimum de capitaux propres, afin d'assurer la solidité financière](https://fr.wikipedia.org/wiki/Histoire_des_bourses_de_valeurs#Les_entreprises_cot.C3.A9es_r.C3.A9duisent_leurs_capitaux_propres) des établissements bancaires. [↑](#footnote-ref-1)
2. Le crédit renouvelable (autrefois appelé crédit permanent ou en anglais *revolving* credit) est une forme de [crédit](https://fr.wikipedia.org/wiki/Cr%C3%A9dit) consistant à mettre à disposition d’un emprunteur une somme d’argent réutilisable au fur et à mesure de son remboursement pour financer des achats non prédéfinis. [↑](#footnote-ref-2)
3. Source : Ricco Rakotomalala, Pratique de la régression logistique [↑](#footnote-ref-3)
4. Kaggle : give me some credit [↑](#footnote-ref-4)