Implémentez un modèle de scoring

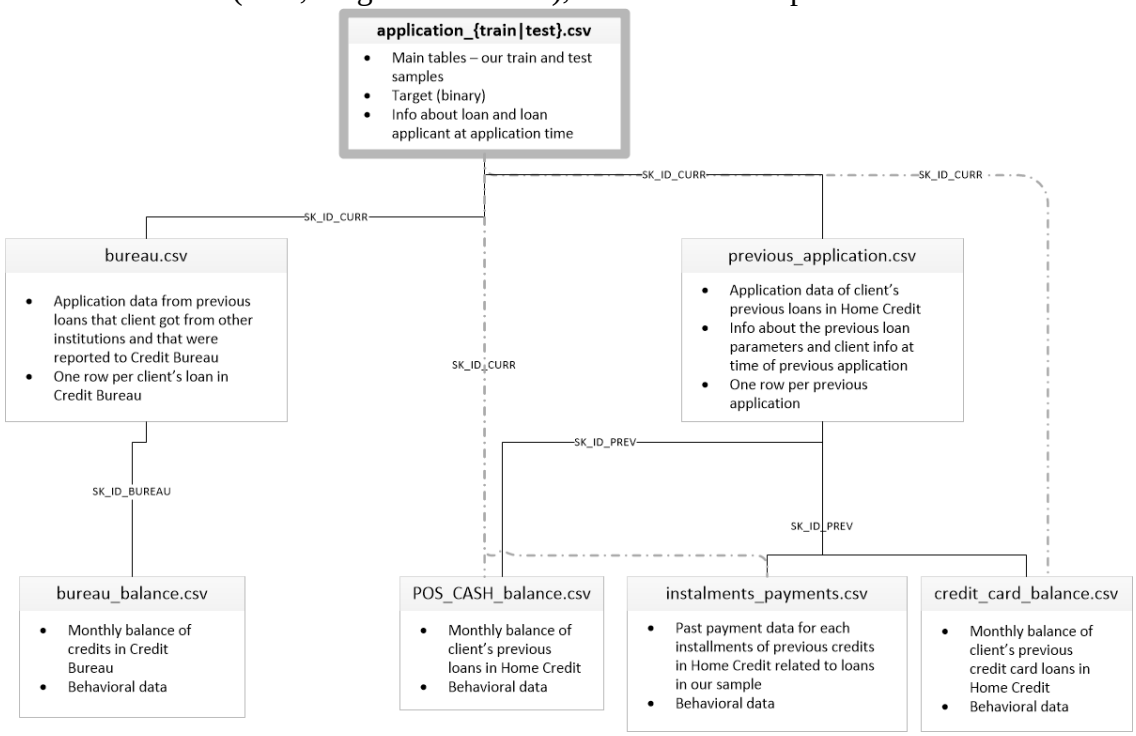
github : https://github.com/iamnotarobotbutitsok/projet_07_model_scoring
dashboard : https://p7ocrscore.herokuapp.com/

Introduction

L'entreprise Prêt à dépenser s'est spécialisée dans les **prêts** pour la consommation destinée aux clients avec un faible historique de prêt.
Aujourd'hui, l'entreprise souhaite pouvoir associer à ses clients un score qui détermine la probabilité de défaut de paiement de ce dernier. D'autre part, l'entreprise souhaite fournir aux chargés de la clientèle un dashboard pour leur permettre de motiver leur décision auprès des clients.

Le score

La probabilité de défaut de paiement d'un client sera établi à partir d'un algorithme de classification binaire. À partir d'un certain nombre de faits économiques du client (revenus, ancienneté de travail etc) et des données indirectes (sexe, rang de la ville etc), il décidera de la potentialité de défaut ou de non



défaut de paiement du client.Le jeu de données

Pour la réalisation de notre modèle, nous utiliserons pour l'essentiel les données du fichier principal Application_train.csv

Modélisation

Matrice de confusion

Le problème de classification doit répondre à la question si un client va faire un défaut de paiement ou non. Dans cette perspective soit la réponse est juste soit elle est fausse. Ceci implique 4 possibilités (2 types de bonne prédiction et 2 types d'erreur) que nous allons traduire dans un tableau que l'on nomme communément Matrice confusion :

		Prédiction	
		0	1
Réel	0	VRAI NÉGATIF (TN)	FAUX POSITIF (FP)
	1	FAUX NÉGATIF (FN)	VRAI POSITIF (TP)

Un modèle parfait ne prédit que des vrais négatifs et des vrais positifs, et dans une moindre mesure, nous souhaitons minimiser un type d'erreur et maximiser un type de bonne prédiction.

Les prêts

Reprenons la matrice de confusion, mais cette fois en renommant les entrées de la matrice selon des points de vue financiers.

		Prédiction	
		0	1
Réel	0	GAINS ($$)	OPPORTUNITÉS MANQUÉES ($$)
	1	PERTES ($$)	PERTES ÉVITÉES ($$)

En partant du principe qu'une entreprise quelle qu'elle soit souhaite augmenter ses bénéfices en premier lieu, l'enjeu ici est de maximiser les gains et de minimiser les pertes. Dit autrement, la capacité prédictive du futur modèle devra être grande concernant les vrais négatifs et devra produire le moins possible de faux négatifs car ceux-ci coûtent chers. Toujours dans la perspective d'augmenter les bénéfices, le modèle devra être capable de prédire les vrais positifs afin d'éviter les pertes ou réciproquement, réduire les opportunités manqués.

D'un point de vue mathématique, cela se traduit par un Recall élevé. Soit :

Recall = TP/(TP+FN) équivalent = (PERTES ÉVITÉES)/(PERTES ÉVITÉES + PERTES)

Cette équation est comprise entre 0 et 1, et elle est d'autant plus élevée que les PERTES sont faibles. Ce que nous souhaitons exactement.

Dans un second temps, voire dans un troisième, nous porterons un regard tout de même sur la PRÉCISION qui équivaut à l'équation

Précision = TP/(TP+FP) = (PERTES ÉVITÉES)/(PERTES ÉVITÉES + OPPORTUNITÉS MANQUÉES)

Certes les opportunités manquées coûtent nettement moins que les PERTES, mais il faut prendre garde qu'il n'y en ait pas beaucoup. Sinon c'est la banqueroute également.

Ces deux équations sont rendues en quelque sorte par le score ROC AUC

Choix de l'algorithme

Light gradient boosting machine (LGBM)

Les tests pour comparer différent modèle ont mis en évidence la performance du LGBM pour la résolution de notre problématique dont voici le rapport de classification des prédictions du modèle dont nous lui fournissons que les données nettoyées et amputées de Application_train.csv

	precision	recall	f1-score	support
Non-Default	0.96	0.69	0.80	56534
Default	0.16	0.68	0.26	4965
accuracy			0.69	61499
macro avg	0.56	0.69	0.53	61499
weighted avg	0.90	0.69	0.76	61499

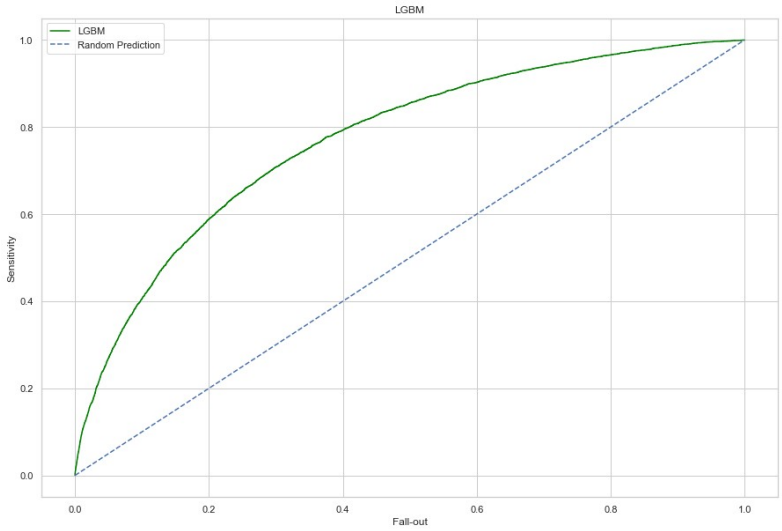
Et ici, nous présentons l'agrégation de l'ensemble des jeux de données aux données précédentes.

	precision	recall	f1-score	support
Non-Default	0.96	0.71	0.82	56534
Default	0.17	0.70	0.28	4965
accuracy			0.71	61499
macro avg	0.57	0.70	0.55	61499
weighted avg	0.90	0.71	0.77	61499

NB : le modèle a été entraîné avec un jeu de données équilibré, soit 19860 profils en non défaut et 19860

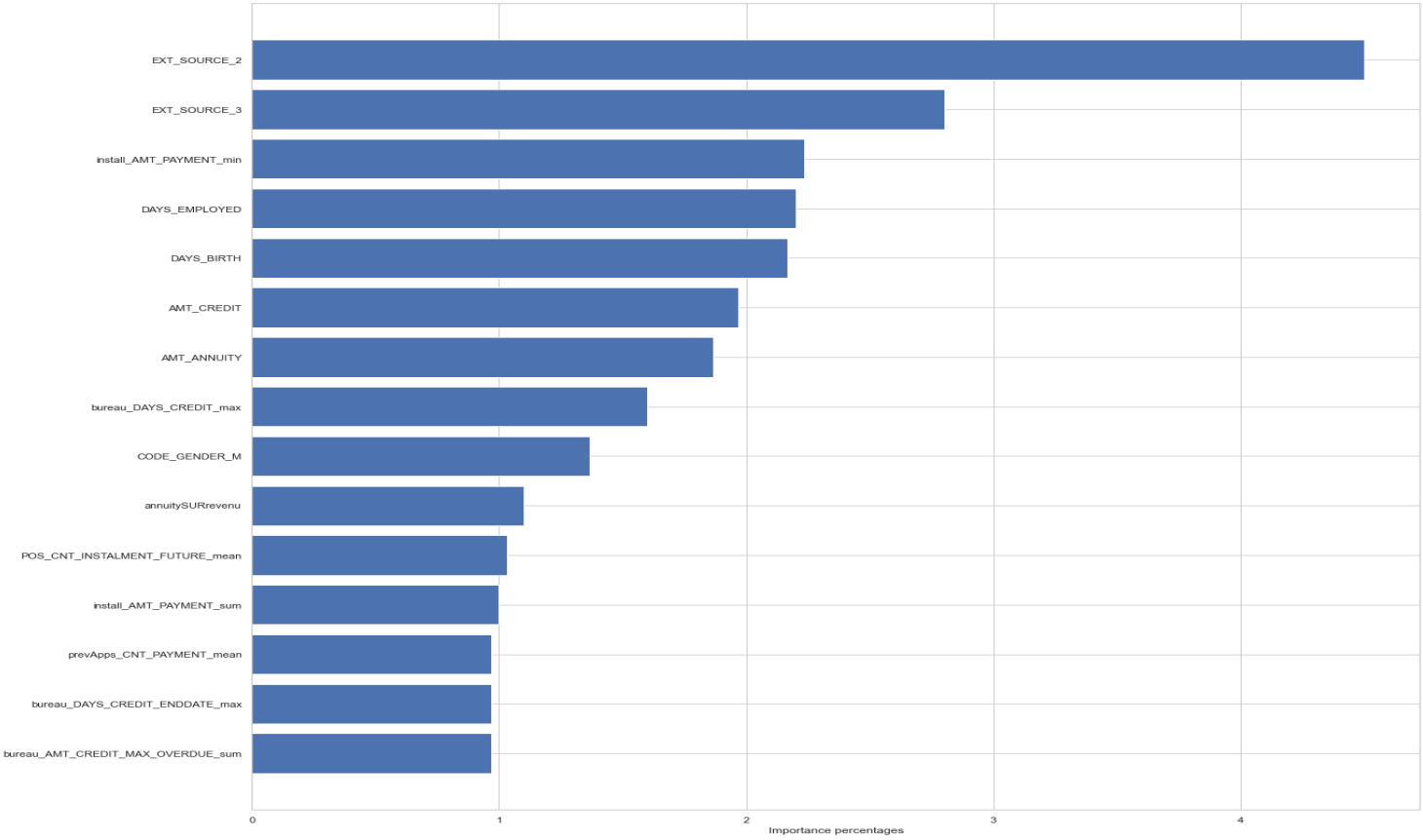
profils en défaut

Nous pouvons y lire un Recall de 0.7 démontrant ainsi sa capacité à trouver des faux négatifs sur un jeu de données test de 61499 profils contenant 8 % de profils en défaut.



Le ROC AUC score est de 0.7, bien mieux que le modèle aléatoire.

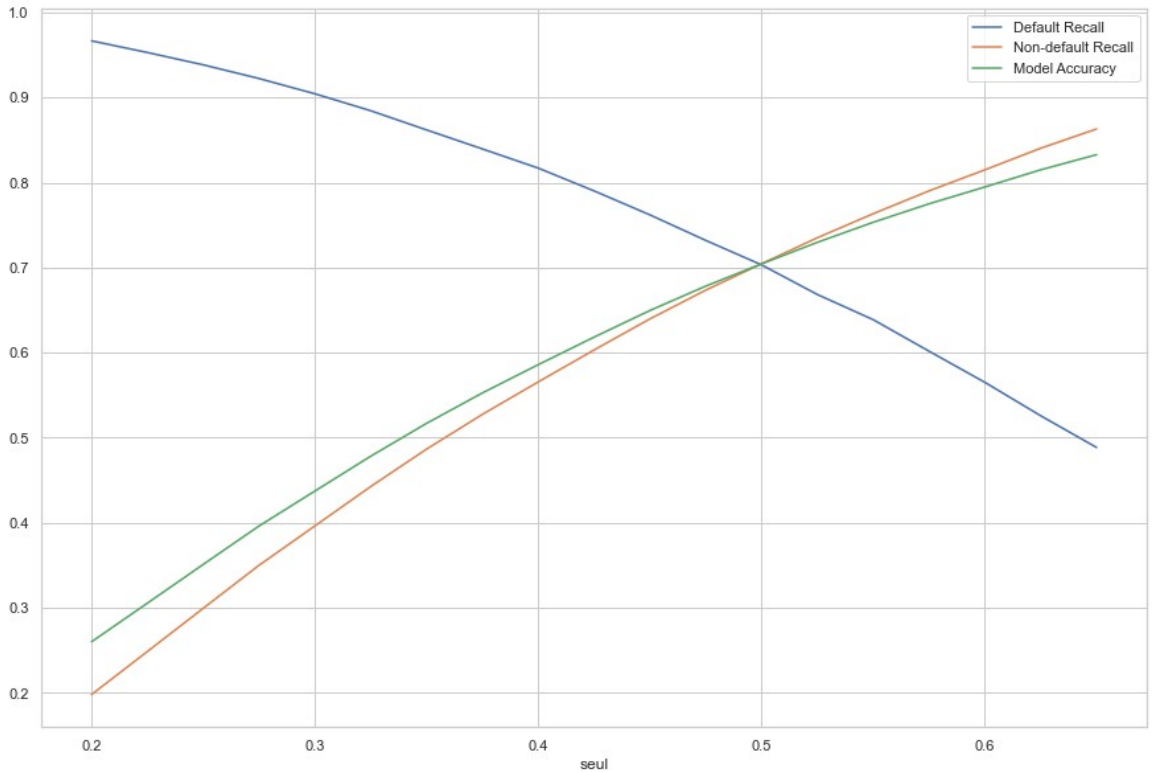
Interprétation



Dans la mesure où l'agrégation des données n'améliore les différents score du modèle que de 0.01 à 0.03 points, nous pouvions prédire que les features du dataset principal auraient plus de poids dans la décision de l'algorithme.
D'autre part nous pouvons constater l'importhttps://vimeo.com/276614526ance des feature EXT_SOURCE2 et EXT_SOURCE3 montrée par notre analyse univariée que vous pouvez observer dans le Notebook. Nous avons une disparité importante (en moyenne) entre les profils en non défaut de paiement et les profils en défaut, et à la fois l'homogénéité des valeurs du fait d'une faible variance.
Il en est de même pour la variable DAYS_EMPLOYED, CODE_GENDER_M et annuitySURrevenu (ratio de l'annuity sur le revenu).

Spécificités du modèle

Le modèle par défaut décide qu'un client fera un défaut de paiement si son score est supérieur au seuil de 0.5 sinon il décidera que non. Mais cette valeur par défaut, est-elle optimale si l'enjeu est d'augmenter les bénéfices de l'entreprise ?
Voici un graphique qui montre l'évolution des Recall et de la précision du modèle.



Nous avons dit précédemment que nous souhaitions avoir le recall le plus élevé possible. Ce n'est pas exactement cela. Dans le graphique ci-dessus, nous avons un recall maximal pour les défauts quand nous fixons le seuil à 0.2, c'est à dire que le modèle trouve tous les profils en défaut de paiement. Toutefois nous voyons bien que fixer un seuil aussi bas n'est pas réaliste. Certes, nous manquerons

aucun client en défaut, mais nous perdrions en même temps beaucoup d'opportunités. Ce n'est pas réaliste.

Faut-il préférer revenir à un seuil par défaut (0.5) ?

Voyons-voir cela en évaluant les profits pour deux seuils différents.

Seuil 0.5 :

pred_TARGET	0	1
TARGET		
$0.00	$23,868,480,866.11	$10,013,422,847.16
$1.00	$883,396,293.79	$2,092,222,836.93

Seuil 0.8 :

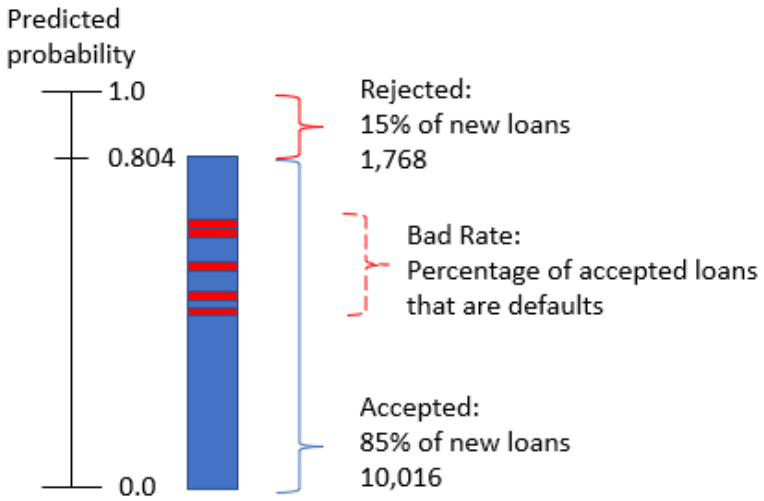
pred_TARGET	0	1
TARGET		
$0.00	$29,717,834,887.44	$4,164,068,825.84
$1.00	$1,610,969,632.10	$1,364,649,498.62

Pour des seuils différents, nous obtenons des bénéfices différents et tout l'enjeu de notre discussion à venir va être de trouver le seuil pour lequel le bénéfice va être maximal

Un seuil pour un profit maximal

Bad rate

Quelque soit le seuil, en dessous duquel le score du profil sera prédit en non-défaut, nous auront malgré tout des faux négatifs.
Leur proportion est donnée par une mesure que nous appellerons le Bad Rate = #(Faux Négatif)/#(Clients prédits en Non-défaut)



Métrique d’optimisation

Considérons la « métrique » Estimation qui va nous permettre d’estimer les bénéfices pour un pourcentage donné de clients à qui nous accorderons un prêt.

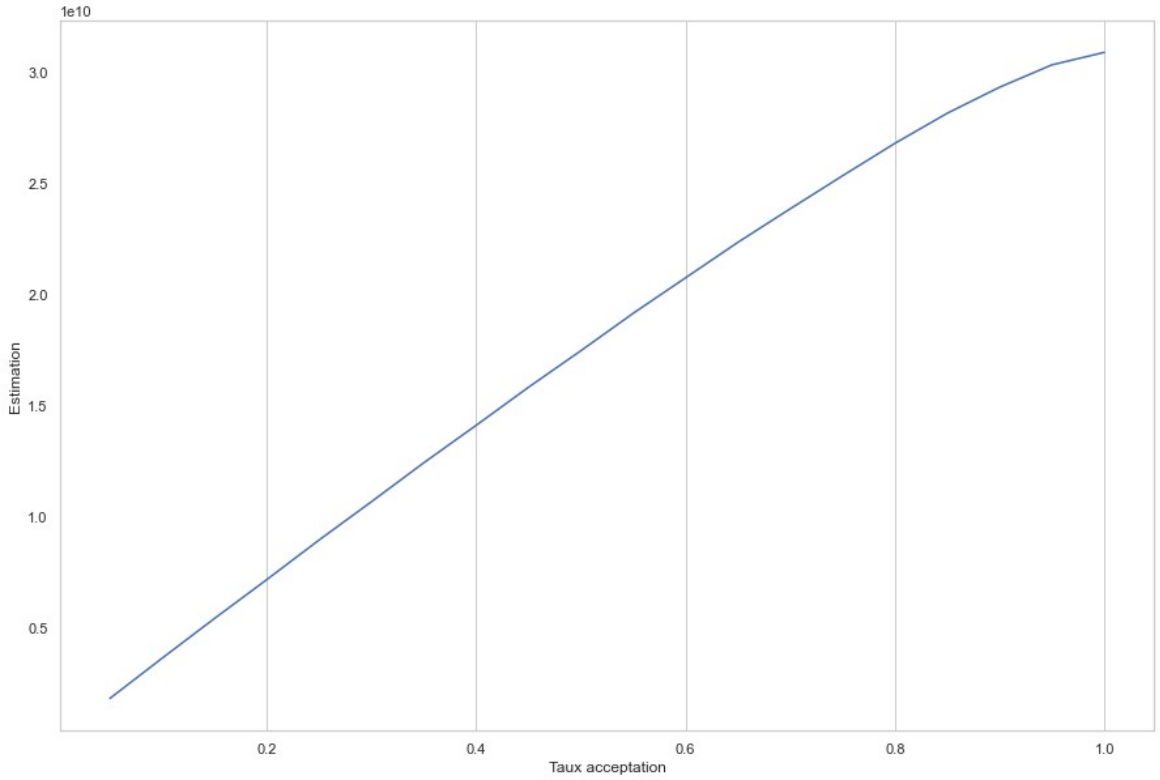
'**Estimation**' = ['nb_accept']*(1-['Bad Rate'])*['Avg'] - ['nb_accept']*['Bad Rate']*['Avg']

	Taux acceptation	Threshold	Bad Rate	Avg	nb_accept	estimation
0	1.00	0.969	0.081	599319.059562	61499	3.088660e+10
1	0.95	0.795	0.067	599319.059562	58427	3.032422e+10
2	0.90	0.723	0.058	599319.059562	55337	2.931743e+10
3	0.85	0.666	0.051	599319.059562	52297	2.814564e+10
4	0.80	0.615	0.046	599319.059562	49231	2.679061e+10
5	0.75	0.567	0.042	599319.059562	46141	2.533031e+10
6	0.70	0.523	0.038	599319.059562	43058	2.384426e+10
7	0.65	0.483	0.034	599319.059562	39986	2.233479e+10
8	0.60	0.445	0.031	599319.059562	36887	2.073644e+10
9	0.55	0.410	0.028	599319.059562	33830	1.913957e+10
10	0.50	0.376	0.026	599319.059562	30721	1.745427e+10
11	0.45	0.345	0.024	599319.059562	27711	1.581056e+10
12	0.40	0.316	0.022	599319.059562	24603	1.409626e+10
13	0.35	0.288	0.020	599319.059562	21573	1.241195e+10
14	0.30	0.260	0.018	599319.059562	18449	1.065879e+10
15	0.25	0.235	0.017	599319.059562	15434	8.935394e+09
16	0.20	0.208	0.015	599319.059562	12302	7.151638e+09
17	0.15	0.182	0.013	599319.059562	9252	5.400733e+09
18	0.10	0.154	0.011	599319.059562	6164	3.612930e+09
19	0.05	0.121	0.007	599319.059562	3041	1.797014e+09

À gros trait, voilà ce que nous faisons :

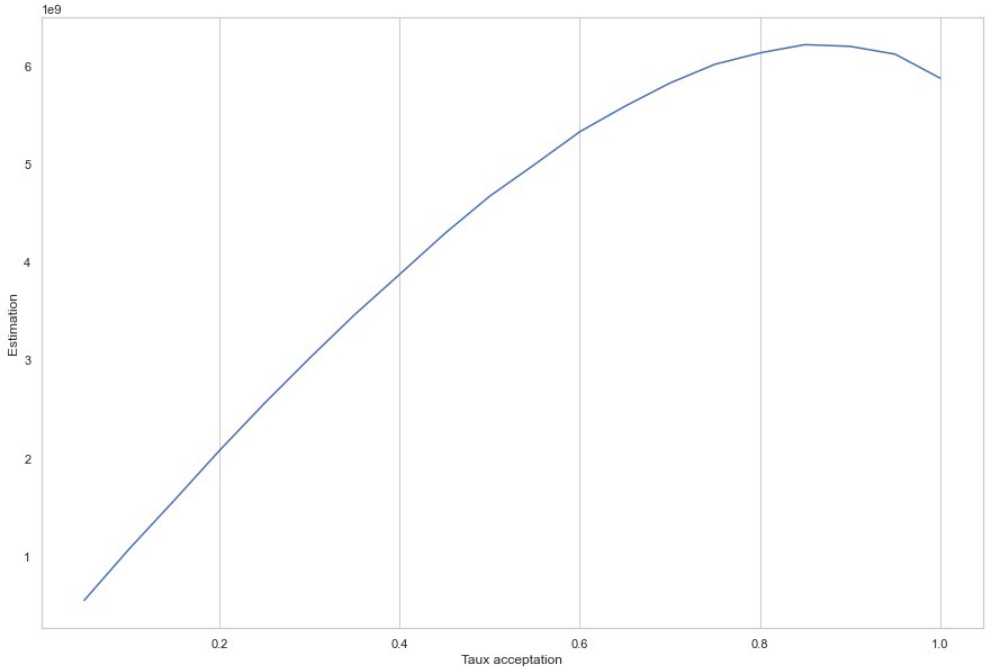
1. Nous prétendons accepter N % des clients
2. N % implique un nombre de clients acceptés (nb_accept)
3. N quantile implique un certain seuil (Threshold)
4. Le Thresholt nous permet de mesure le Bad rate.
5. Calcul des bénéfices à l’aide de la métrique Estimation

Évolution de Estimation par Taux acceptation



Cette visualisation nous dit que si le dataset (qui par ailleurs n'est pas réaliste par rapport au métier du fait qu'il est tiré d'un tournois de Kaggle dont l'enjeu était tout autre) contient seulement 8 % de défauts alors il faut accorder un prêt à tous les clients pour maximiser les bénéfices.

Toutefois, soyons moins optimiste et extrayons du dataset test un échantillon qui contient 25 % de profils en défaut.



Dans ce cas-là, notre métrique atteint un maximum lorsque nous acceptons 85 % de clients soit que nous fixons un seuil à 0.721.

Taux acceptation	Threshold	Bad Rate	Avg	nb_accept	estimation
0.85	0.721	0.189	591335.4375	16893	6.213425e+09

Conclusion

Nous avons esquissé une métrique qui nous permet d'estimer les bénéfices « nets » à partir de la connaissance du taux de faux négatifs qu'un jeu de données contient.
Cette méthode est simple à mettre en œuvre et rapide, permettant ainsi de mettre à jour régulièrement le seuil qui sépare les profils prédits en non défaut de ceux en défaut.