M2 EKAP-Économétrie Appliqué et statistiques
IAE-Nantes - Université de Nantes

Dossier SVM et ANN :
Jane Street Market Prediction

Jeff ABRAHAMSON

ROMAND Kyllien
DEL'CHATEAU Jean-Baptiste

2021/2022

Sommaire

Introduction

Nous allons tout d'abord commencer par donner une définition des SVM & ANN. Les SVM sont des machines à supports vectorielles, soit une ensemble de techniques d'apprentissage supervisé detinées à résoudre des problèmes de discrimination. L'ANN, également appelée réseau de neurones artificiels, est un système dont la conception est à l'origine schématiquement inspirée du fonctionnement des neurones biologiques, et qui par la suite s'est rapproché des méthodes statistiques.

Dans ce dossier nous allons donc traiter du machine learning avec un sujet de Kaggle, le sujet traité sera les prédictions du marché de Jane Street. Ce sujet nous semble intéressant, il traite de la prédiction d'un marché. Le marché en question est celui de Jane Street, une entreprise de trading basée aux Etats-Unis. Nous allons devoir créer différents modèles nous permettant de prédire si la variable resp de l'action est validée, si l'action fait un bénéfice, un rendement positif.

Ce dossier va donc être divisé en plusieurs parties, la première partie concerne l'analyse économique du sujet ainsi que la présentation de nos données. La seconde partie sera sur l'analyse pratique sous le logiciel python, l'analyse des résultats. Ce dossier se terminera par une conclusion. Vous pourrez retrouver une table des matières à la fin de celui-ci.

Partie économique

Explication du sujet

Comme dit dans l'introduction, nous allons traiter les prédictions du marché de jane street. Jane Street est une entreprise basée à New York mais qui possède différents bureaux notamment un qui se situe à Londres. C'est une société de trading pour compte propre, c'est une des plus grandes sociétés dans ce domaine, négociant plus de 17 000 milliards de dollars de titres en 2020. Elle est même considérée comme l'entreprise réussissant à maintenir la liquidité des ETF (fonds de placements qui permettent d'investir en bourse) obligataires durant la crise de la COVID-19. Son co-fondateur Tim Reynolds estime que la société a été fondée en 2000. Jane Street a une activité commerciale très rentable a noté en juillet 2020 le S&P Global Ratings.
Cette entreprise utilise un langage de programmation peu répandu, il s'agit de Ocaml. Le langage Ocaml est un langage spécifique, dit multi-paradigme du fait qu'il permet la programmation orienté objet et la programmation modulaire.

Le fait d'acheter bas et de vendre haut paraît si simple mais est en fait beaucoup plus complexe que ce que l'on peut croire. Ce problème est de surcroît de plus en plus difficile à résoudre avec des évolutions de plus en plus rapides de nos jours, et ce de plus sur un marché financier complexe. Avec le commerce électronique, n'importe quelle personne ayant des connaissances financières ou non peut aller sur une place boursière pour acheter des parts dans une entreprise. Cela se traduit par des milliers de transactions par seconde et des opportunités presque illimités. Mais ce qui est rentable aujourd'hui, ne le sera peut être plus demain, notamment avec la volatilité du marché qui rend la prédiction impossible pour une transaction. Si nous étions sur un marché parfait, les acheteurs et les vendeurs auraient toutes les informations nécessaires pour prendre des décisions commerciales rationnelles, ne pas acheter trop et ne pas vendre trop bas. En conséquence, les produits resteraient à leur juste valeur, ils ne seraient ni

sous-évalués ni surévalués. Définir une stratégie de rentabilité est complexe, il est donc difficile de distinguer la chance d'avoir pris une bonne décision.

Pour répondre à cette problématique, nous allons construire un réseau neuronal.

Présentation des données

Les données que nous allons utiliser ont été téléchargées sur kaggle, nous n'avons pas eu de recherche de données à faire. Sur ces données, nous avons les données boursières réelles représentées par les *features* allant de 0 à 129. Chaque ligne de la base représente une opportunité de trading pour laquelle nous allons devoir prédire une action de valeur 1 si la transaction se fait et de valeur 0 si elle ne se fait pas. Sur chaque transaction, nous avons un poids (*weight*) et une variable associé (*resp*), ces deux variables représentent ensemble un retour sur la transaction. Nous avons le jour de chaque transaction par la variable *date* et la variable *ts_id* qui elle représente un ordre de temps. Cette base de données est un ensemble de fonctionnalités anonymisées. Dans notre base de données, correspondant à la base de données d'apprentissage, nous aurons des métiers avec un poids de zéro (weight=0), ceux-ci sont inclus dans la base de données par souci d'exhaustivité. Notre base de données représente environ 2 millions de lignes.

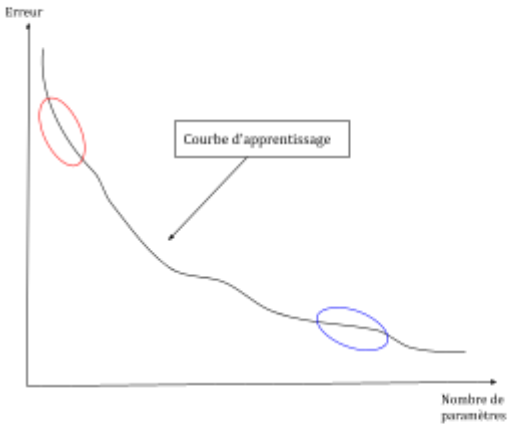
Méthodologie

Avant de commencer la partie pratique, il est important d'expliquer les différentes méthodes que nous allons utiliser. Dans ce dossier, nous avons utilisé un modèle de perceptron multiples, un gradient boosting ainsi qu'un stochastic gradient boosting.

Descente de gradient :

Une descente de gradient est un algorithme d'optimisation qui permet de trouver le minimum de n'importe quelle fonction convexe. Une fonction convexe est une fonction dont l'allure ressemble à celle d'une vallée avec au centre le minimum global (le point le plus bas de la vallée). Pour définir cette descente de gradient, on prend souvent l'exemple d'être en haut d'une montagne et descendre dans la vallée mais il y a du brouillard, donc on y va petit à petit en cherchant le chemin le plus court (Exemple vu en cours). Le but de cette descente de gradient est de trouver le paramètre Beta de notre équation. Pour mieux comprendre cette descente de gradient, nous allons l'expliquer par un graphique. Si notre premier résultat se trouve au niveau du rond rouge, on peut continuer la descente de gradient pour trouver un meilleur résultat. Si le résultat en sortie se trouve dans le bleu, dans ce cas, on sortira un bon résultat.

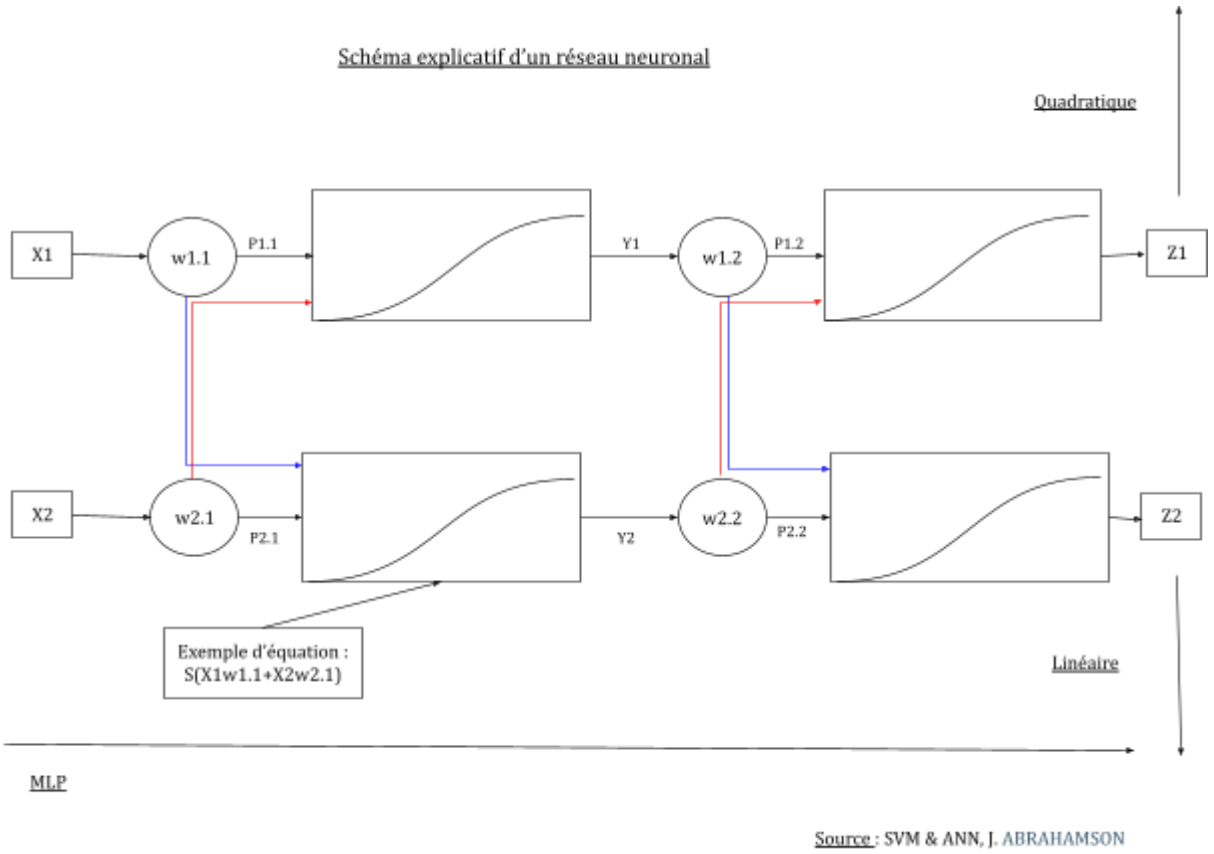
Graphique d'explication de la descente de gradient



Source : SVM & ANN, J. ABRAHAMSON

Perceptrons multicouche :

Ce modèle est un réseau neuronal, système dont la conception est à l'origine schématiquement inspirée du fonctionnement des neurones biologiques, et qui par la suite s'est rapproché des méthodes statistiques. Le schéma ci-dessous, nous présente le fonctionnement d'un réseau neuronal. Deux neurones sont entrés et vont circuler dans le réseau, prendre des poids pour au final prendre une valeur de sortie qui sera en fonction des différents poids pris.

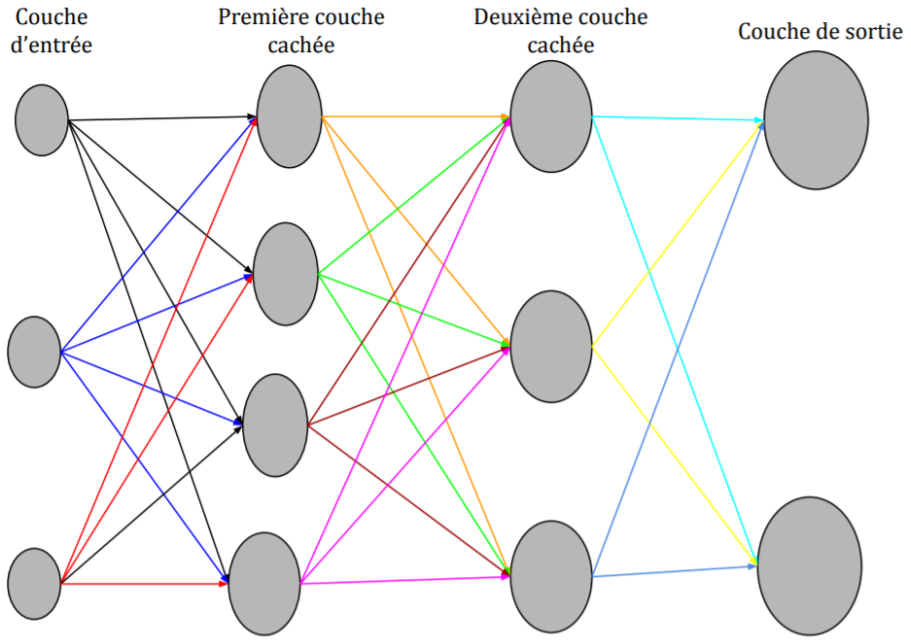


Source : SVM & ANN, J. ABRAHAMSON

Le perceptron multicouche (MLP) est un type de réseau neuronal artificiel organisé en plusieurs couches. Dans ce réseau, l'information circule de la couche d'entrée à la couche de sortie et uniquement dans ce sens, ce qui en fait un réseau à propagation directe. Chaque couche de ce réseau est constituée de neurones d'un nombre variables, les neurones se trouvant sur la dernière couche sont les neurones de sorties. Pour mieux comprendre le concept, nous allons nous aider d'un schéma. Nous avons trois neurones sur la couche d'entrée, puis nous avons deux couches cachées composées respectivement de quatre et trois neurones chacune. La couche de sortie est composée

de deux neurones. Ce schéma pourrait correspondre à une variable binaire avec une couche de sortie de 0 ou 1. Il faut savoir que chaque neurones de chaque couche est relié à tous les neurones de la couche suivante. Dans ce schéma, l'information va de neurones en neurones afin de donner le résultat en couche de sortie. La source de ce schéma est le cours de SVM & ANN de J. ABRAHAMSON.

Schéma d'explication d'un perceptron multicouche



Gradient boosting :

Le boosting est une technique ensembliste qui consiste à agréger des modèles élaborés séquentiellement sur un échantillon d'apprentissage pour lequel le poids des individus est modifié au fur et à mesure. Les modèles sont ensuite pondérés selon leurs performances et agréger. L'apprentissage itératif, à chaque étape, agit essentiellement sur le biais. La combinaison finale des modèles permet de réduire la variance. Ne nécessite pas de grands arbres. C'est une stratégie adaptative.

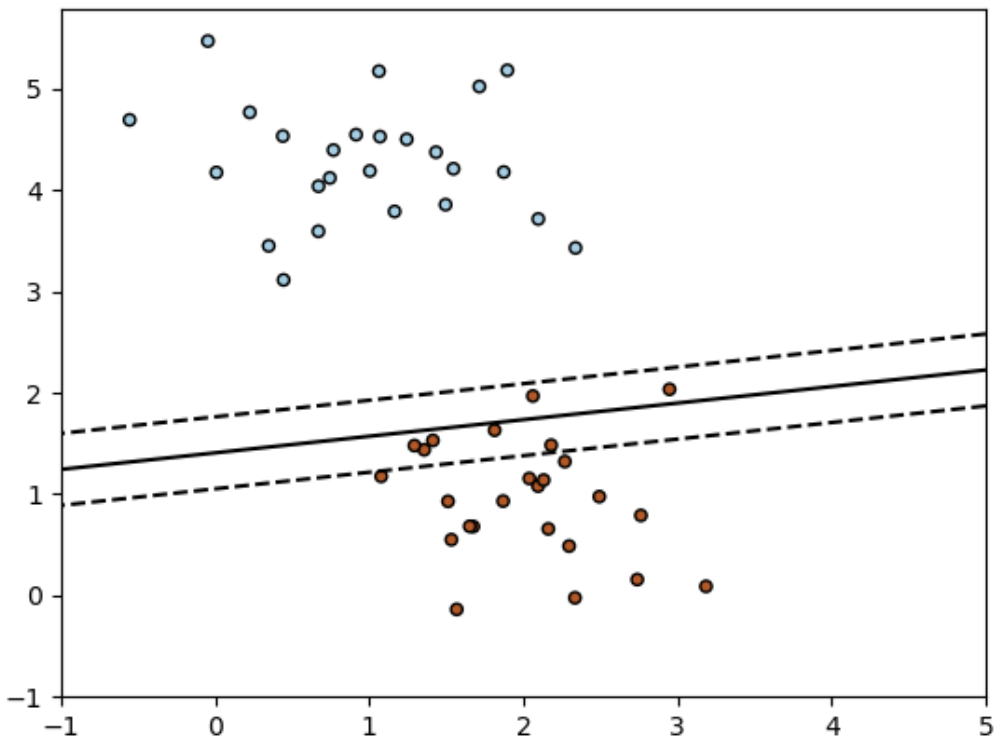
Dans le gradient boosting, son résultat change en fonction de plusieurs paramètres, le premier est la taille des arbres. Si d=1 (2 nœuds terminaux, « stumps »), on ne peut pas introduire d'éventuelles interactions entre les prédicteurs. Si d=2 (3 nœuds terminaux), le modèle peut prendre en compte des interactions d'ordre deux entre les prédicteurs. Certains auteurs suggèrent de choisir entre 3 et 9 . Le second paramètre est le learning

rate, empiriquement, lorsque le paramètre de lissage est faible, on constate de meilleures performances prédictives, mais au prix d'une convergence plus lente (nombre d'itérations, B, plus élevé). Plus ce learning rate est bas et plus le nombre d'estimateurs doit être élevé.

Stochastic gradient boosting

A chaque étape, seule une fraction f de l'échantillon est utilisée pour la construction des « base learners ». Sous python, il existe une fonction qui fait ce genre de modèle, ce qui va nous permettre de ne pas prendre les lignes une par une mais aléatoirement, il fait le surapprentissage tout seul. Ce type de modèle est un SVM, dont le but est d'avoir une frontière la plus large possible. La séparation sur le graphique ci-dessous sert de séparateur. On voit que les points bleus sont en haut du graphe au-dessus du séparateur. Les points rouges sont quant à eux en bas du graphe mais sont dans le séparateur, cela s'explique par le fait que les points sont répartis plus au milieu du graphe que les bleus.

Exemple de Classification



Partie pratique

Analyse préliminaires

Avant de commencer la partie modèle, nous avons effectué une analyse préliminaire sur notre base de données. Nous avons tout d'abord partitionner la base de données, elle est trop lourde pour nos machines. Nous avons donc travaillé sur un dixième de la base soit un peu plus de 200 000 observations. Pour nous permettre de faire nos modèles, nous avons créé la variable action, qui est la variable à expliquer. Cette variable prend 1 quand la variable resp est positive. La variable resp correspond aux rendements de l'actif pour chaque ligne de la base de données. On aurait pu travailler sur des échelles de temps différentes pour perfectionner le modèle mais nous avons été pris par le temps, nous avons vu la limite de nos ordinateurs, pas du tout adapté à faire du calcul sur ce genre de base (Big Data). Nous avons effectué un describe de nos variables mais les résultats ne sont pas affichables et pas compréhensible car les données sont anonymisées. Nous avions des valeurs manquantes dans les features que nous avons changées par zéro. Ce changement permet que toutes les observations soient prises en compte dans nos modèles.

Nous savons que les rendements financiers sont des séries temporelles, nous devons donc les considérer de sorte, prendre en compte l'histoire. Mais nous n'avons pas pu travailler de cette sorte, la base nous donnait une variable jour qui est comprise entre 0 et 499, nous ne connaissions donc pas la date du trade. On considère donc dans les variables de la base, l'histoire de l'actif était comprise.

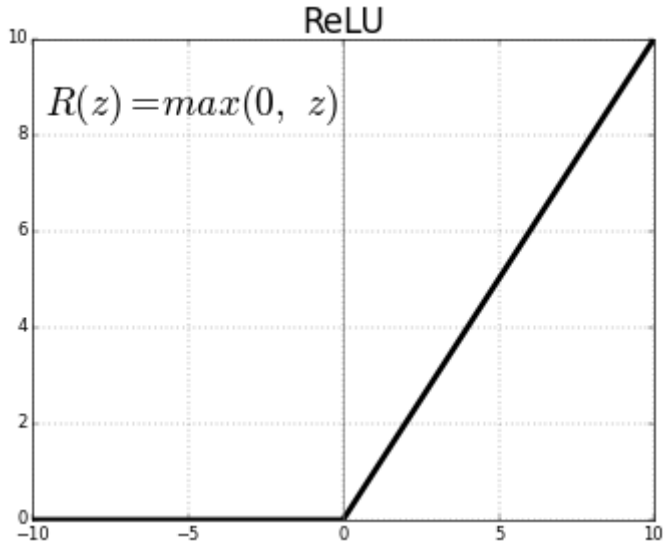
Nous avons exécuté une cross validation avant de faire la partie sur la descente de gradient. La base train prend 70% de la base entière et donc la base test prend donc 30%.

Nous avons considéré que la base était bonne, pas de nettoyage de base, nous aurions été dans l'incapacité de faire ce nettoyage.

Les modèles

MLP

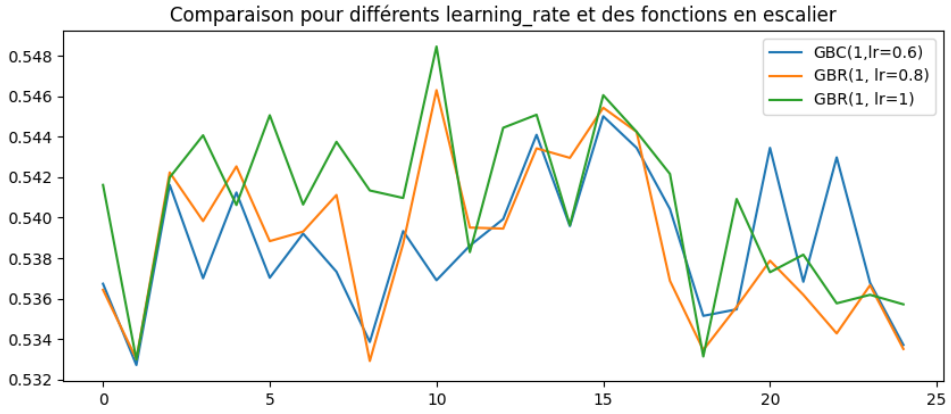
Le premier modèle que nous avons généré est le perceptron multiple. Nous avons effectué un seul modèle de perceptron multiple, trop long dans le calcul (8 heures). Le nombre d'epochs (les neurones) est de 100. Le nombre de splits de ce modèle est de 10, il a été partitionné en 10 segments. Nous trouvons donc un résultat de prédiction de 57,4% et un écart-type de 0,24%. Ce résultat de 0,24% borne notre prédiction. Notre modèle MLP prédit à 57% si nous devons prendre le trade ou pas. Ce qui est un bon résultat dans la finance.Le type de neurones que nous avons utilisé dans ce modèle est ReLu. Le principe de ce type de neurones est de mettre 0 pour les valeurs négatives, il garde que les valeurs positives. Le graphique ci-dessous, nous montre le neurone Relu.



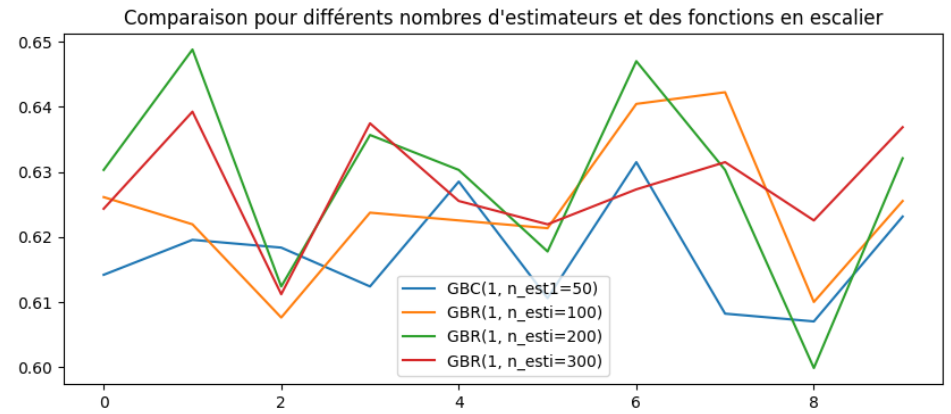
Descente de gradient booster

Le second modèle est la descente de gradient boosting. Nous en avons effectué 8 en appliquant différents paramètres afin de pouvoir comparer, et de déterminer le meilleur modèle. Pour choisir ces paramètres, nous l'avons fait graphiquement.

Sur le graphique ci-dessous, nous pouvons voir une comparaison pour différents learning rate. Nous voyons que le meilleur paramètre est lorsque le learning rate est égal à 1, la courbe verte est au-dessus des autres. Dans nos modèles, nous allons intégrer les trois valeurs pour vérifier si notre choix est le bon.



Ce second graphique nous permet de choisir le deuxième paramètre de notre modèle. il s'agit du nombre d'estimateurs à prendre. Nous remarquons que plus la valeur augmente et meilleure est la prévision. Nous avons donc décidé de mettre dans notre modèle un nombre d'estimateurs de 500 ou 1000.



Le dernier paramètre a été également choisi graphiquement, il s'agit de la profondeur de l'arbre. Nous observons que graphiquement, que plus la profondeur est grande et plus la prévision sera précise. Pour nos différents modèles, nous avons décidé de prendre des profondeurs de 1,3,5,7.

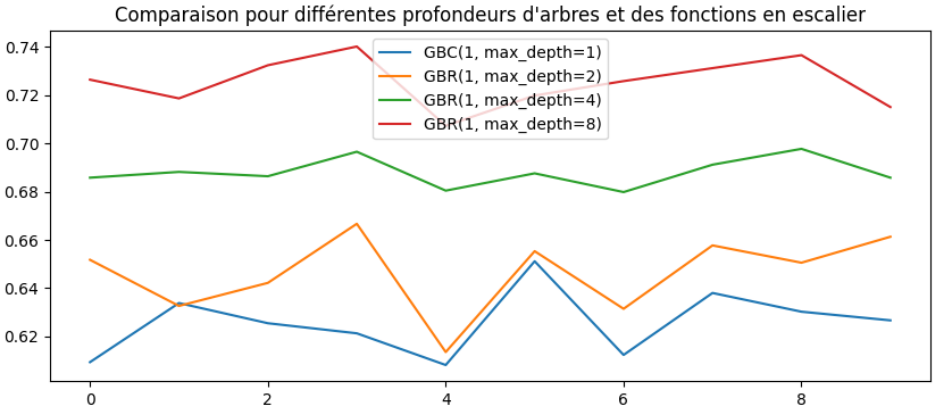


Tableau 1 : Récapitulatif des changements de paramètres sur nos modèles

	Modèle 1	Modèle 2	Modèle 3	Modèle 4	Modèle 5	Modèle 6	Modèle 7	Modèle 8
n estimator	500	1000	500	500	500	500	500	500
learning rate	1	1	1	3	0.8	0.5	1	1
Max depth	1	1	3	1	1	1	7	5

Afin de mieux visionner nos résultats, ils sont présents dans le tableau ci-dessous. Nous voyons que les modèles avec les profondeurs les plus faibles sont les moins bons modèles. Nous trouvons que notre meilleur modèle est le numéro 8, qui prédit à 64.7% sur le jeu de validation.

Nous avons sûrement un problème de surapprentissage quand on regarde le résultat sur le jeu de calibration du modèle 7, ce résultat est trop fort.

Nous décidons donc de choisir le modèle 8 et pas le modèle 7 qui retourne de meilleur résultat du fait du sur apprentissage expliqué ci-dessus. Nous observons que le learning rate de ce modèle est de 1, notre raisonnement au regard du graphique était le bon.

Tableau 2 : Résultat de nos différents modèles

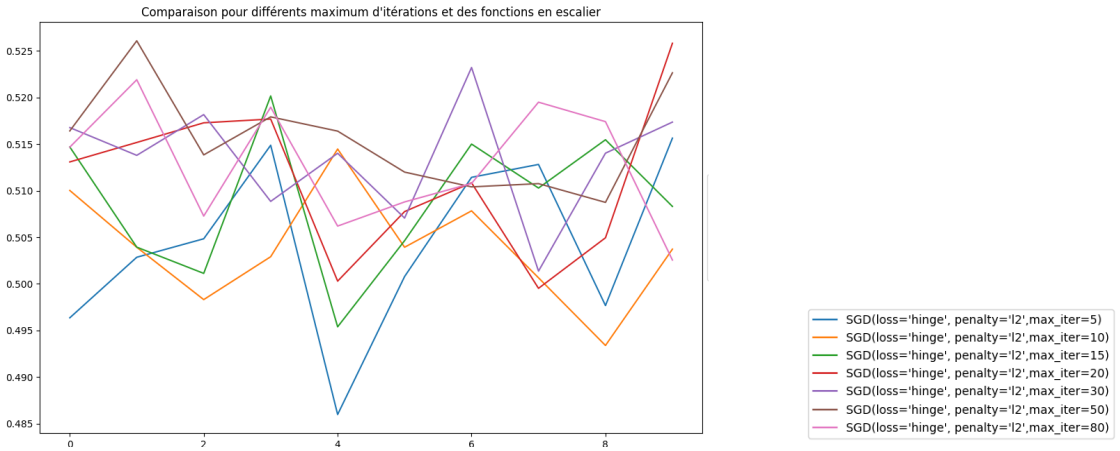
	Modèle 1	Modèle 2	Modèle 3	Modèle 4	Modèle 5	Modèle 6	Modèle 7	Modèle 8
Calibration	0.569	0.577	0.70	0.471	0.556	0.561	0.985	0.855
Validation	0.554	0.556	0.62	0.475	0.553	0.551	0.663	0.647

Nous pouvons conclure que sur ce modèle, le meilleur que nous avons est le modèle 8, avec un nombre d'estimateurs de 500, un learning rate de 1 et une profondeur d'arbre de 5. Ce modèle permet de prédire 64,7% des actions à faire et donc des trades à prendre.

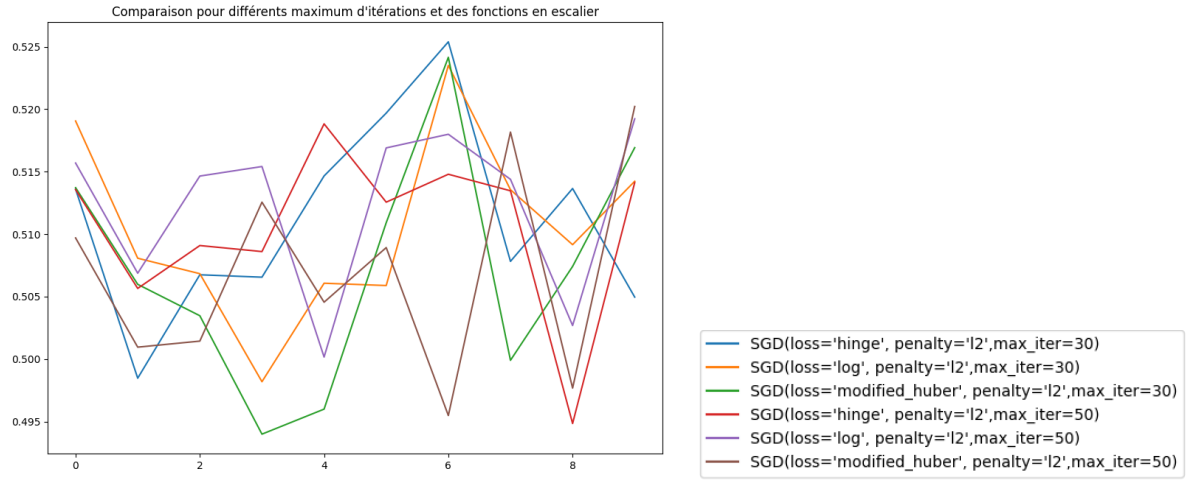
Stochastic gradient boosting

Le dernier modèle que nous avons exploré est le Stochastic gradient boosting. A notre grand étonnement, nous avons pu effectuer ce modèle sur toute la base. Cela s'explique par la simplicité algorithmique de la fonction utilisée sous Python.

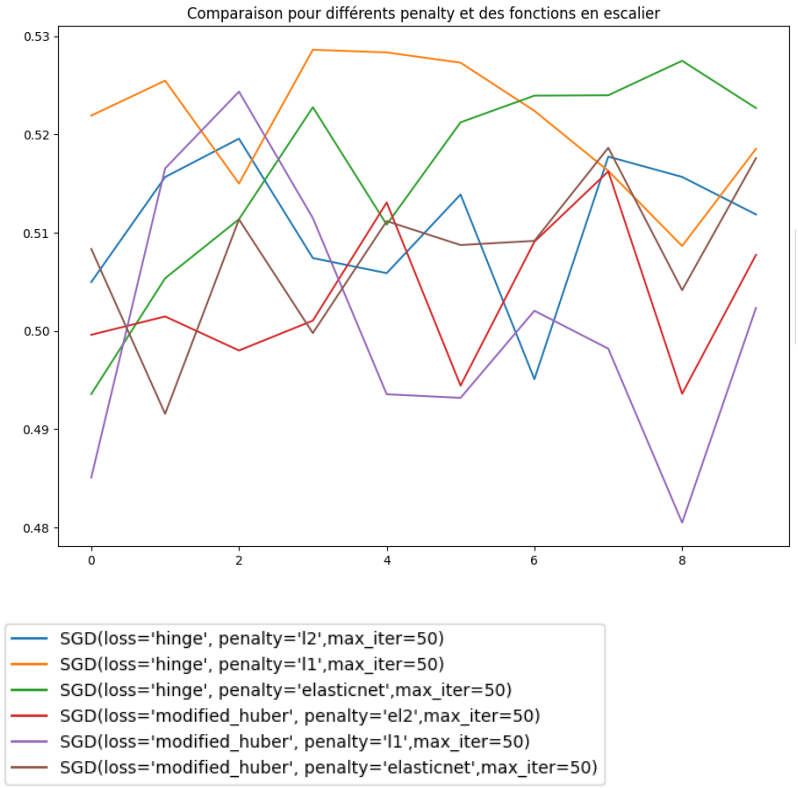
Pour ce modèle, le paramètre du nombre d'itérations à mettre à été choisi graphiquement, avec celui ci-dessous. Graphiquement nous voyons que les meilleurs paramètres sont 50 ou 30, ils donnent les meilleures prévisions. Pour nos modèles, nous avons décidé de faire tourner des modèles avec les deux paramètres afin de voir lequel est le meilleur.



Le graphique suivant permet d'essayer de déterminer la meilleure fonction loss adaptée à la base de données. On remarque que le nombre d'itérations 30 serait donc la meilleure possibilité, et les deux paramètres de fonction loss *hinge* et *modified_huber* paraissent les meilleures options. Le paramètre *hinge* correspond à la fonction de coût linéaire des SVM , alors que *modified_huber* est une fonction *smooth* de la fonction de coût précédente.



Pour le graphe qui va suivre, on modifie le paramètre qui définit le type de pénalité choisis appliqué au modèle. Pour cela, trois choix sont possibles *l2*, *l1* et *elasticnet* : le premier correspond à la pénalité L1 qui conduit à des solutions éparses, conduisant la plupart des coefficients à zéro. L'Elastic Net résout certaines lacunes de la pénalité L1 en présence d'attributs fortement corrélés. Le paramètre L2, correspond également à une pénalité,sur un coefficient. Le dernier paramètre elactic net fait une combinaison des pénalités L1 et L2, d'où le fait qu'il puisse résoudre certaines lacunes de L1.



Pour conclure sur ce stochastic gradient boosting, le meilleur modèle que nous trouvons pour 30 itérations est avec la pénalité L2. Concernant le modèle avec 50 itérations, il y a deux meilleurs modèles, un pour le début du graphique, le modèle avec la pénalité L1 en *Hinge* et le second pour la deuxième moitié du graphique qui est avec la pénalité.

Pour les prévisions, nous décidons de prendre le modèle hing avec 50 itérations. Nous l'avons fait tourner sur toutes les bases de données et nous trouvons 52.2% pour la validation et 52.3% pour la calibration.

Ce modèle va donc nous prédire à 52.2% la bonne action à choisir, c'est-à-dire prendre le trade ou non .

Conclusion

Pour conclure, malgré des machines pas assez puissantes, nous avons réussi à sortir 3 modèles au-dessus de 50% de prédiction. De par la loi des grands nombres, un modèle prédisant à 50% est un bon modèle. Nos trois modèles sont donc bons. Le meilleur modèle est la descente de gradient booster, qui nous donne le meilleur taux de prédiction.

Ce dossier est le premier ou nous avons vu les limites de nos ordinateurs pour faire tourner des grosses bases de données, par exemple, le modèle du réseau neuronal a pris 8 h pour tourner. Alors que nous avions pris un dixième de la base de données.

Si nous voulions améliorer nos prévisions, nous aurions pu travailler sur la base entière, nous aurions également pu effectuer un nettoyage sur base en ayant accès aux variables qui sont anonymes.

Ce projet nous a permis d'approfondir nos connaissances dans les réseaux neuronaux et les SVM, il nous a également permis de mieux connaître le logiciel Python.

Tables des matières