Projet de Master

Romain Mencattini

30 novembre 2017

Table des matières

1 Introduction

Le but de ce projet est d'utiliser des techniques de *machine learning* dans le cadre de la finance. Plus précisément, nous allons reprendre des techniques algorithmiques pour créer un programme de *trading* de taux de changes.

Nous allons dans un premier temps faire un état de l'art. Ce dernier sera composé de plusieurs parties.

La première traitera la marché des taux de changes afin d'obtenir les connaissances pour comprendre les enjeux et les buts recherchés. Il s'agira d'une introduction d'éléments de bases mais néanmoins essentiels à la compréhension des algorithmes.

La deuxième partie plus mathématique abordera l'aspect théorique de plusieurs algorithmes clefs. Soit :

— Les réseaux de neurones.
— Les arbres de décision.
— Les algorithmes *SVM*.
— *Logistic Regression*.
— *Naive Bayes*.
— Descente du Gradient.

Ensuite, nous verrons l'application de la théorie à notre cas concret, le marché des change ; leurs problèmes, limitations et solutions rencontrés ainsi que les résultats concernant les performances des programmes.

Pour conclure, nous justifierons le choix de l'algorithme ainsi que les éléments clefs du *machine learning*.

Après l'état de l'art, nous implémenterons l'algorithme ou une analyse plus poussée de sa partie mathématique sera effectuée, mais également de son application au domaine financier. Nous analyserons également les problèmes et solutions rencontrés.

Une fois l'implémentation terminée, plusieurs *benchmark* seront effectués afin d'estimer les améliorations les plus pertinentes et la performance générale de l'algorithme.

Finalement, nous analyserons les résultats et déduirons des conclusions. De plus une analyse de la pertinence de l'algorithme et des optimisations sera réalisée et nous proposerons des pistes d'améliorations.

2 État de l'art

2.1 Introduction

Avant la démocratisation de l'informatique, les opérations financières étaient réalisées par des humains. Ce système pouvait avoir des inconvénients :

— L'émotionnel influençait les transactions. En effet, ces dernières étant effectuées par des humains, il y avait un risque non négligeable que l'état de la personne agisse sur sa décision.
— Un problème sous-jacent était de maintenir une discipline de *trading*. Afin de minimiser les pertes et de maximiser les gains, il fallait se tenir à un plan afin de ne pas se laisser influencer par des paramètres extérieurs. Cela pouvait être très difficile.
— Le *backtesting*[1] était impossible. Tester la qualification ainsi que la qualité de *trading* d'une personne était compliquée. De même pour un *trading plan*.

Ces éléments ont, en partie, favorisé l'émergence et l'utilisation d'algorithmes dans la finance. En 2014 aux États-Unis, 84% des transactions étaient accomplies par des algorithmes [17]. Ce qui représente environ 100'000 réalisations, ou *ticks*, par secondes [17]. Durant l'évolution de l'outil informatique, le monde de la finance en a suivi les améliorations afin de perfectionner leurs algorithmes. On retrouve donc des méthodes d'optimisations poussées ainsi que les récentes découvertes de *data mining* et de *machine learning*, abrégé *ML*. Des propositions de plus en plus pointues dans les deux domaines voient le jour. L'algorithme qui sera au coeur de ce projet en fait partie. Il s'agit d'un réseau de neurones avec plusieurs couches prenant en compte des paramètres particuliers à la finance.

Afin d'approcher aux mieux ces notions, nous allons discuter des éléments nécessaires à leur compréhension. Nous allons en premier lieu traiter le domaine financier ainsi que ces outils. Puis nous parlerons de plusieurs méthodes de *ML*. Voici celles vont être développées dans cet état de l'art :

— Les réseaux de neurones.
— Les arbres de décision.
— Les algorithmes *SVM* [25].
— *Logistic Regression*.
— *Naive Bayes*.
— Descente du Gradient [21] ainsi que sa version dite stochastique [16].

Finalement, nous lierons les deux domaines en montrant comment adapter les modèles mathématiques de *ML* pour les utiliser comme techniques de *trading*, en évaluant leur performances.

2.2 Finance

2.2.1 *FOREX*

Afin d'appréhender le fonctionnement du *FOREX*, il est important de mentionner certaines décisions historiques. Ces dernières ayant façonné le marché des devises actuel.

1. Le *Backtesting* est le processus qui consiste à tester une stratégie de *trading* sur des données historiques afin de s'assurer de sa viabilité avant de risquer du capital. [6]

Jusqu'à la première guerre mondiale, le système en vigueur se basait sur l'or, que l'on nommait l'étalon-or[1]. S'en suit une période d'instabilité notamment dûe aux pertes occasionnées par la guerre, un après-guerre compliqué, la crise boursière de 1929 et la seconde guerre mondiale.

C'est au sortir de cette dernière, que la nécessité de "*mettre en place une organisation monétaire mondiale et de favoriser la reconstruction et le développement économique des pays touchés par la guerre*" [23], est apparue. Le but était également "*d'aplanir les conflits économiques, reconnaissant par là les problèmes engendrés par les disparités économiques*" [19].

Plusieurs idées furent proposées, mais ce fût celle de Harry Dexter White qui fût mise en place. Cette dernière prévoyait entre autre :

— le choix du Dollar américain comme étalon, avec rattachement à l'or[2].
— Création de la Banque internationale pour la reconstruction et le développement (BIRD) qui deviendra la banque mondiale.
— Le Fond monétaire international (FMI).
— Création de l'Organisation mondiale du commerce[3].

On remarque que ces institutions sont toujours en activités, cela démontre l'importance de ces accords pour le système financier actuel.

Le marché *FOREX* porte sur les devises. La valeur d'une devise ne peut être exprimée qu'en fonction d'une autre. Par exemple 1 franc suisse vaut 1.05 euro.[4] La transaction porte donc sur deux monnaies comme CHF/EUR. On va vendre des francs suisses pour acheter des euros ou l'inverse. Le nom du marché vient d'ailleurs de ces échanges. On échange une monnaie contre une autre, c'est un *FOreign EXchange*, ou *FOREX*.

Il y a deux variations possibles :

— La monnaie peut subir une dépréciation.
— La monnaie peut subir une appréciation.

Lorsque le prix d'une devise augmente par rapport à une monnaie étrangère, on parle d'appréciation. Ainsi dans le cas contraire, on parlera d'une dépréciation.

La mondialisation a facilité ce marché. En effet, toutes devises étant accessibles depuis n'importe où, il devient donc possible d'avoir des marchés avec des devises plus exotiques.

Les principaux acteurs financiers sont [4] :

— Les banques commerciales. Elles peuvent pratiquer des interventions directes car elles gèrent des dépôts et veulent opérer des transactions sur ces derniers. Il leur est également possible de réaliser le rôle d'intermédiaire financier.
— Les entreprises. Ces dernières vont pratiquer des transactions directes, si elles disposent d'un accès aux marchés sinon via des intermédiaires.

1. Source : [19].
2. Suspension de l'équivalence or pour le dollar américain en août 1971 puis abandon définitif en mars 1973 [23].
3. Ne verra le jour qu'en 1995 faute d'accord [23].
4. Taux fictif utilisé pour l'exemple.

— Les institutions financières non-bancaires. On peut citer les fonds de pensions, les sociétés d'assurances ou les *hedge funds*. Ce sont surtout dans un but de spéculation, d'arbitrage ou de couverture de risque qu'elles agissent.
— Les banques centrales. Il peut y avoir des interventions directes, dans le but de modifier l'appréciation de la monnaie.
— Les ménages. Surtout dans une optique de voyage, d'achat ou de spéculation.

Henry Bourguinat a énoncé "*la règle des trois unités*" qui correspondent aux unités de temps, de lieu et d'opérations et d'acteurs. Le *FOREX* répond à ces trois unités [12] :
— Ce marché fonctionne 24h/24 et les transactions s'effectuent presque en continue.
— Il fonctionne à l'échelle mondiale tout en étant décentralisé. De part l'évolution des technologies, l'information circule aisément malgré son statut.
— L'uniformité des procédés ainsi que des produits est présente. Les acteurs malgré nationalité sont de même nature.

Il existe principalement deux horizons temporels : le *spot* et le *forward*.

Le premier est également appelé "Le marché au comptant". Lorsque deux acteurs se mettent d'accord sur une transaction, cette dernière se réalise immédiatement [1].

Le second peut être nommé "Le marché à terme". L'accord est passé à un temps T mais la transaction effective ne se réalise que dans le futur. Ce futur, ou maturité, peut être de plusieurs dizaines de jours, voir des années, soit $T + X$ [2].

Il y a des opérations réalisable sur le marché à terme. :
— Les *swaps*. Ils consistent à vendre une monnaie au comptant puis à la racheter à terme [3].
— Les *futures/forwards*. La différence entre ces deux tient surtout à leur standardisation et leur mise en place. Cependant le principe reste le même : on réalise une opération (d'achat ou de vente) qui ne s'effectuera qu'à maturité.
— Les *options*. Cela représente un contrat vendu par un parti (*the option writer*) à un autre parti (*the option holder*). Ce contrat offre le droit, et non l'obligation contrairement aux *futures/forwards*, d'acheter (*call*) ou de vendre (*put*). Ici encore, il faut attendre la maturité [4].

Les options sont très versatiles. Elles peuvent être utilisées afin de spéculer ou de diminuer le risque. Voici les différents types possibles :
— **long call** → on achète le droit d'acheter le sous-jacent à un certain prix.
— **short call** → on vend le droit d'acheter le sous-jacent à un certain prix.
— **long put** → on achète le droit de vendre le sous-jacent à un certain prix.
— **short put** → on vend le droit de vendre le sous-jacent à un certain prix.

1. Valable en théorie, dans la réalité cela peut prendre du temps [4]
2. Où T est le moment présent, et X une durée de temps.
3. Soit à $T + X$
4. Cela est vrai pour les options dites européennes [8]. Dans le cas des options américaines [5], le droit peut s'exercer à n'importe quel moment, offrant ainsi une plus grande flexibilité.

Le *bid* est le prix maximum qu'un acheteur est d'accord de payer pour un sous-jacent. De la même manière, le *ask* est le prix minimum qu'un vendeur accepte pour vendre un sous-jacent [7].

La différence entre le *bid* et le *ask*, appelée le *spread*, représente la liquidité d'un actif. Il est également utilisé comme marge par les *broker* [22] et autres plateformes.

2.3 Cadre théorique des algorithmes de *Machine Learning*

2.3.1 Introduction

T. Mitchell a donné une définition formelle [14] :

"*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P is its performance at tasks in T, as measured by P, improves with experience E*"

On a donc une tâche T à accomplir, où T peut consister à trier des images ou à reconnaître des motifs. La mesure de la réussite de cette tâche T est nommée P. C'est-à-dire la qualité du résultat du programme pour la tâche donnée, T. Si le programme améliore son résultat P pour la tâche T grâce à l'expérience E. Il s'agit d'un programme de *machine learning*. L'expérience peut être vue comme une phase d'entraînement ou comme le fait de retenir les réponses après avoir accompli la tâche.

Il existe deux catégories d'apprentissage :
— L'apprentissage supervisé.
— L'apprentissage non-supervisé.

Dans le cas du premier, on fournit au programme, un ensemble d'entraînement [1], qui contient des réalisations ainsi que le résultat de la classification. Le programme va donc pouvoir utiliser ce savoir afin d'améliorer sa performance P. Nous disposons donc de nombreux couples (x_i, y_i) et le but est de trouver une fonction $f \in F$ telle que : $f(x) = y$.

Pour l'apprentissage non-supervisé, on fournit des données, mais sans le résultat voulu. C'est uniquement après avoir décidé d'une valeur qu'on va signifier au programme si cette dernière est correcte. On ne lui donnera jamais la valeur attendue. Il va donc utiliser uniquement les résultats précédents pour améliorer son P.

Par exemple, on désire reconnaître un certain type de voiture à partir d'images. Dans le cas de l'apprentissage supervisé, nous allons fournir au programme un ensemble d'entraînement qui contient de nombreuses photos de voitures, ainsi que la marque des dites voitures. L'algorithme va donc travailler avec ces données.

Par contre dans le cas de l'apprentissage non-supervisé, le programme ne pourra utiliser que les photos, et après avoir retourné le résultat, nous lui dirons si c'est juste ou faux. Il mémorisera le résultat optimisera en conséquence ses réponses.

1. Ou d'expérience, E

Concernant, l'ensemble d'entraînement, il y a des points à prendre en compte afin de minimiser les risques de sur-apprentissage [1], et de maximiser la qualité de nos données. Pour ce faire il faut :

- Représenter la population générale. Donc si le but est du traitement de la langue, il faut que la propension et la répartition des mots soient les mêmes que ceux de la langue.
- Contenir des membres de chacune des classes. Pour reconnaître des chiffres, il est important de disposer de chacun des chiffres dans l'ensemble d'entraînement.
- Contenir de grandes variations ainsi que du bruit. Afin d'éviter le sur-apprentissage, il faut de nombreux exemples différents, voir très différents, les uns des autres ainsi que du bruit [2].

Il est important de saisir comment fonctionne les algorithmes de *machine learning*. Le but est d'utiliser des données, souvent de très hautes dimensionnalités [3], dans des équations dont on pourra faire varier les paramètres afin de classifier au mieux. Le cœur des algorithmes de *machine learning* consiste à optimiser les dits paramètres.

2.3.2 *Logistic Regression*

Une régression en statistique consiste à analyser la relation entre une variable par rapport à un ensemble d'autres [24]. On veut estimer la probabilité conditionnelle, en se basant sur des variables et en utilisant une distribution logistique cumulative. Cette dernière a une forme semblable à une distribution Gaussienne, mais avec des queues épaisses et donc une *kurtosis* plus élevée. La *kurtosis* étant définie comme suit :

$Kurt(X) = \frac{\mu_4}{\sigma^4}$, où μ_4 est le quatrième moment centré et σ^4 la variance au carré.

Le but est de modéliser : $P(Y = 1|X = x)$ comme fonction de x. Nous voulons donc savoir quelle est la probabilité que la classe [4] Y vaille 1 sachant que X vaut x.

Le modèle de régression est le suivant [11] :

$$log(\frac{p(x)}{1-p(x)}) = \beta_0 + x \cdot \beta$$

, où β_0 représente l'ordonnée à l'origine de la régression linéaire , β est le coefficient de régression, de même dimension que x, et x la donnée dont on veut obtenir la classe.

En résolvant cette équation pour p, cela donne [11] :

$$p(x|y) = \frac{1}{1+e^{-(\beta_0+x\cdot\beta)}}$$

1. Le sur-apprentissage consiste à apprendre par coeur la tâche, plutôt que d'apprendre les principes pour réaliser la tâche.
2. Comme des faux exemples.
3. Cela signifie qu'elles sont représentées par un grand nombre d'attributs.
4. Dans ce cas, la classe désigne une réalisation de la variable aléatoire Y.

Dans le cadre de l'article : *A Machine Learning Approach to Automated Trading* [11], l'auteur a implémenté deux variations de cet algorithme :

— *Logistic regression with a ridge penalty* :

$$\sum_{i=1}^{N}(y_i - \sum_j \beta_j x_{ij})^2 + \lambda \sum_j \beta_j^2$$

, où y_i est la classe de notre observation, β_j sont les coefficients de la régression logistique originale [11] et x_{ij} sont les j éléments de l'observation i.
L'objectif est de minimiser le carré de la différence entre la classe, ou résultat, de y_i et le résultat calculé : $\sum_j \beta_j x_{ij}$. Donc en fonction de l'observation x_i et des coefficient de β. En ajoutant une pénalité, sur β, on va tenter d'éviter le sur-apprentissage.

— *Lasso logistic regression/ Lasso regularization* :

$$\sum_{i=1}^{N}(y_i - \sum_j \beta_j x_{ij})^2 + \lambda \sum_j |\beta|$$

Le fonctionnement est similaire au précédent, seul change la pénalité.

Comme mentionné plus haut, l'optimisation porte sur les paramètres de l'équation. Dans le cas de la régression logistique, il s'agît du β. Il conviendra donc de trouver la valeur optimale pour cette variable afin de maximiser ou minimiser les équations ci-dessus. De manière similaire pour les autres techniques de *ML*, les équations et les paramètres vont changer, mais le but sera toujours d'optimiser ces éléments.

2.3.3 Les arbres de décision

Un arbre de décision est un arbre, dont chaque nœud représente un test sur un attribut. Les branches qui suivent directement le nœud sont les valeurs possibles de l'attribut. Les feuilles de l'arbre, quant à elles, sont la classification d'élément donné en entrée.

Il est important de disposer des attributs avant de commencer la construction de l'arbre. Lors de l'implémentation, nous pouvons représenter l'arbre comme une suite de *if-then-else* afin d'améliorer la lisibilité. Dans ce cas très précis, disposer d'un langage permettant le *pattern matching* est fort utile.

Cet algorithme a tendance à très facilement sur-apprendre, il convient donc de bien choisir la manière de construire l'arbre ainsi que l'ensemble d'entraînement pour minimiser cet effet.

Voici un exemple d'arbre de décision [1] :

1. http://cloudmark.github.io/images/kotlin/ID3.png

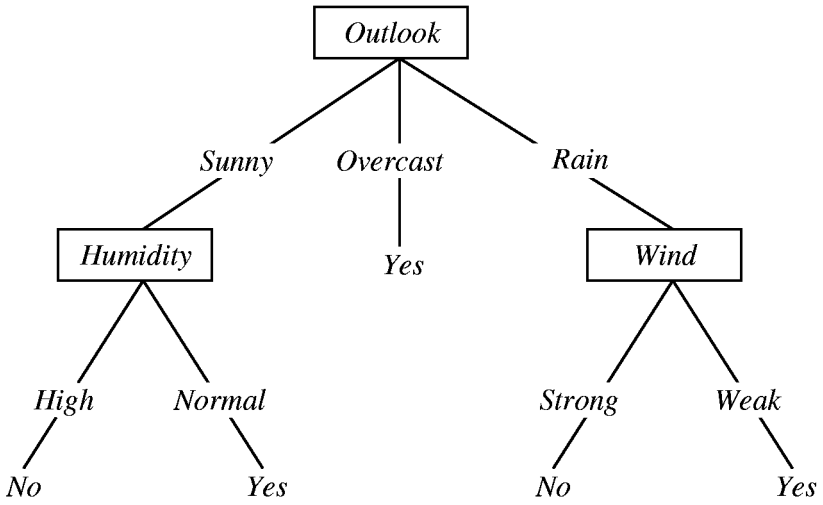

FIGURE 1 – Exemple d'arbre de décision : Permet de décider si nous pouvons aller jouer au tennis ou non.

Afin de construire l'arbre à partir de l'ensemble d'entraînement, il existe plusieurs algorithmes. Un des plus connus est le *ID3* [13]. Il s'agit d'une méthode de type *greedy*.

À chaque itération, il faut :

— effectuer un test statistique [1] afin de trouver l'attribut le plus discriminant.

— utiliser cet attribut comme nœud.

— retourner au premier point, tant que l'ensemble des attributs n'est pas vide.

Nous allons construire l'arbre de manière *top-down* en utilisant ,à chaque pas, le meilleur attribut, selon notre test statistique.

2.3.4 *Naive Bayes*

À l'instar de la régression logistique (voir 2.3.2), il s'agit d'un classifieur probabiliste. Ce dernier se base sur le théorème de Bayes [2] :

$$P(Y|X) = \frac{P(X|Y)}{P(X)}P(Y)$$

À partir de cela, nous obtenons l'équation de *machine learning* suivante [11] :

$$V_{NB} = arg\ max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j)$$

Où V_{NB} est la classe obtenue, $P(v_j)$ la probabilité à *priori* donc sans informations et $\prod_i P(a_i|v_j)$ la probabilité de vraisemblance. Il convient donc de trouver la classe qui maximise ce calcul.

1. Il vise à vérifier la quantité d'information gagnée pour la classification [13].
2. https://brilliant.org/wiki/bayes-theorem/

Afin d'avoir une certaine sécurité dans les résultats, il est possible d'ajouter un seuil. Les réponses du classifieur étant comprise entre 0 et 1, le seuil permettra de décider si la réponse sera prise en compte.

Par exemple, avec un seuil de 0.6 si $V_{NB} = 0.58$, alors la réponse n'est pas validé et le classifieur ne renvoie aucun résultat. Si $V_{NB} = 0.7$ alors la réponse est jugée sûre et la classe de V_{NB} est retournée.

La *ROC*[1] *Curve analysis* permet d'améliorer le classifieur. En effet cette dernière peut détecter les *true positive rate* par rapport aux *false positive rate* pour différents seuils de classification de l'algorithme *Naive Bayes* [11]. À partir de cela, nous pouvons déterminer le meilleur seuil de sortie et donc perfectionner notre algorithme. De plus cette courbe peut aider à comparer des classifieurs entre eux en comparant la surface sous la courbe [11].

La méthode utilisée est la suivante [11] : il faut faire s'intersecter la pente S avec la courbe ROC et ainsi obtenir une valeur optimale pour le seuil. Cette pente S est définie comme suit [11] :

$$S = \frac{Cost(P|N) - Cost(N|N)}{Cost(N|P) - Cost(P|P)} \cdot \frac{N}{P}$$

Sachant que $Cost(P|N)$ est le coût pour avoir mal classé une classe négative comme positive. P est la somme des vrais positifs et des faux négatifs. N, quant à lui, vaut la somme des vrais négatifs ainsi que des faux positifs.

Voici une illustration[2] :

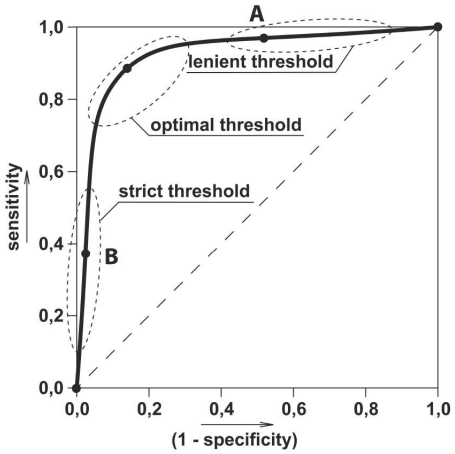

FIGURE 2 – Exemple de *ROC curve* : L'axe des x correspond au taux de faux positifs et l'axe des y au taux de vrais positifs. On veut donc trendre vers $y = 1$ et $x = 0$. Lors de l'optimisation par la pente S, le but sera d'obtenir une intersection avec la *ROC curve* dans la zone *optimal threshold* afin d'avoir le meilleur seuil possible.

1. Receiver Operating Characteristic
2. Source : http://www.prolekare.cz/dbpic/jp_5403_f_20-x1000_1600

2.3.5 *SVM*

Le but de l'algorithme *SVM*[1] est de séparer les données grâce à un hyper-plan. Cela permet de différencier les classes des observations suivantes en déterminant s'ils se trouvent d'un côté ou l'autre du plan. Ce dernier n'est pas unique[2] :

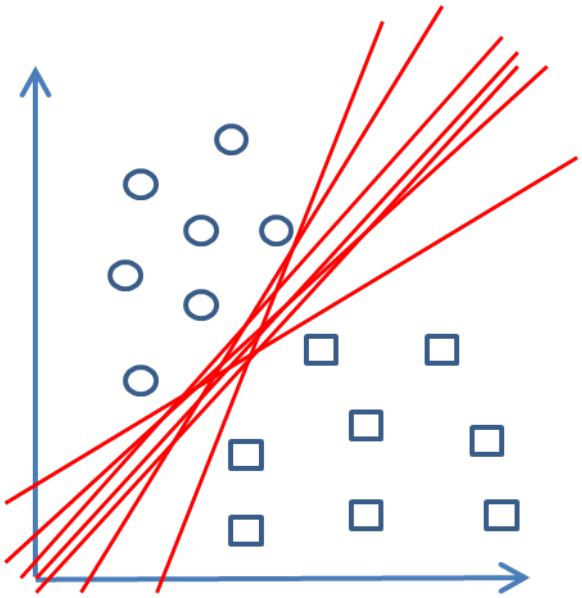

FIGURE 3 – Exemple d'hyper-plans séparant les données. Si nous voulons classifier un nouvel élément, il suffit de calculer s'il se trouve à gauche ou à droite de l'hyper-plan. Dans le premier cas, il s'agira, pour notre algorithme, d'un rond et dans l'autre d'un carré

Notre fonction est :

$$f(x) = (w \cdot x) + b$$

Le but est de maximiser la distance entre les points les plus proches de l'hyper-plan, tout en pénalisant les points mal classés. Il n'est pas toujours possible de séparer les données de dimensions n, il conviendra donc d'augmenter la dimension afin d'obtenir une dimension $m > n$ plus discriminante. La fonction $\phi(x)$ est utilisée dans ce but. Un exemple de fonction est :

$$\phi : R^2 \to R^3 : (x, y) \to (x, y, z) := (x^2, \sqrt{2}xy, y^2)$$

Il est possible d'imager cette opération comme cela[3] :

1. *Support Vector Machine*
2. Source : https://computersciencesource.files.wordpress.com/2010/01/svmafter.png
3. https://www.dtreg.com/uploaded/pageimg/SvmDimensionMap.jpg

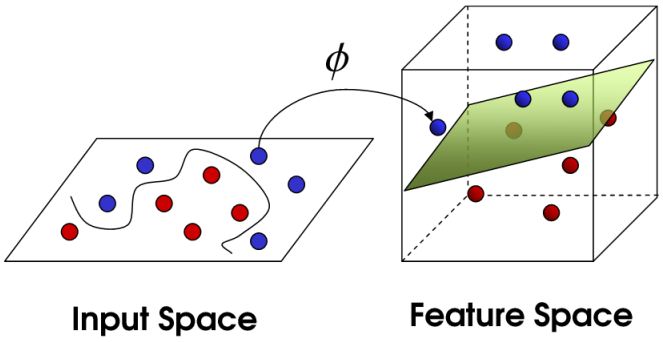

FIGURE 4 – Exemple d'utilisation de la fonction ϕ pour passer d'un espace R^2 à R^3 afin de faciliter la séparation.

En terme d'équation, nous voulons minimiser w, dans l'équation de l'hyper-plan : $(w \cdot x) + b$. Il faut également prendre en compte $y_i \in \{-1, +1\}$:

$$(w \cdot x) + b = \begin{cases} \geq +1 & \text{si } y_i = +1 \\ \leq -1 & \text{si } y_i = -1 \end{cases}$$

Ce qui donne [18] :

$$y_i(w \cdot x_i + b) \geq 1$$

Si malgré l'augmentation de la dimension, les données ne sont pas séparables, il faut tenter de minimiser le nombre d'éléments mal placés. Pour ce faire [18] :

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \text{ avec } \xi_i > 0$$

Afin de diminuer l'erreur et optimiser au mieux notre classifieur.

2.3.6 Réseaux de neurones

Tout comme les algorithmes génétiques s'inspirent de la sélection naturelle dans un but d'optimisation, les réseaux de neurones se basent sur un modèle formel de neurones [1] afin de copier la capacité d'apprentissage des êtres vivants.

Il s'agit d'opérer à partir de données en entrée, des *inputs*, une ou plusieurs multiplications matricielles en utilisant des vecteurs de poids, des *weights*. L'optimisation s'applique sur les *weights*, afin de maximiser la classification.

Un réseau de neurones peut avoir plusieurs couches, *layers*. Dans ce cas, la première couche est appelée *input layer*, la dernière *output layer* et toutes celles entre ces deux sont les *hidden layers*. De plus, les neurones peuvent être pleinement connectés avec ceux de la

1. Neurones formels : http://www.peoi.org/Courses/Coursesfr/neural/neural3.html

couche suivante, *feed-forward* ; ce qui signifie que les neurones de la couche $n-1$ influencent ceux de la couche n. Il est également possible que les éléments de n agissent sur les neurones de $n-1$, ce phénomène est appelé *feedback networks*.

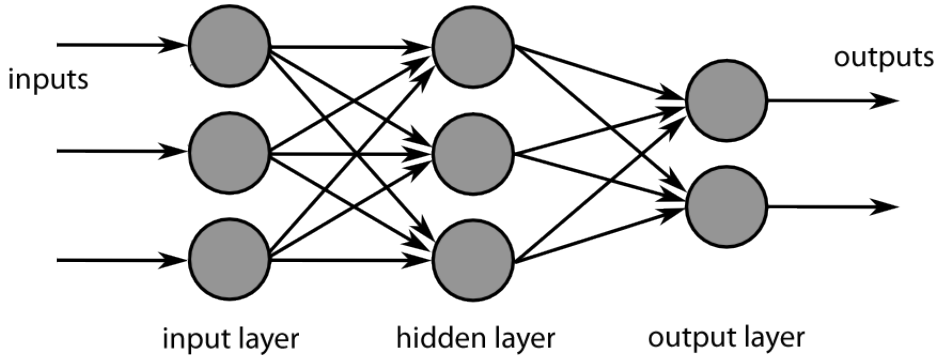

FIGURE 5 – Exemple d'un réseau de neurones avec plusieurs couches. Illustre également le *feed-forward* : l'*ouput* de l'*input layer* est propagé dans chaque neurones de l'*hidden layer*. Même chose pour les deux dernières couches. [1]

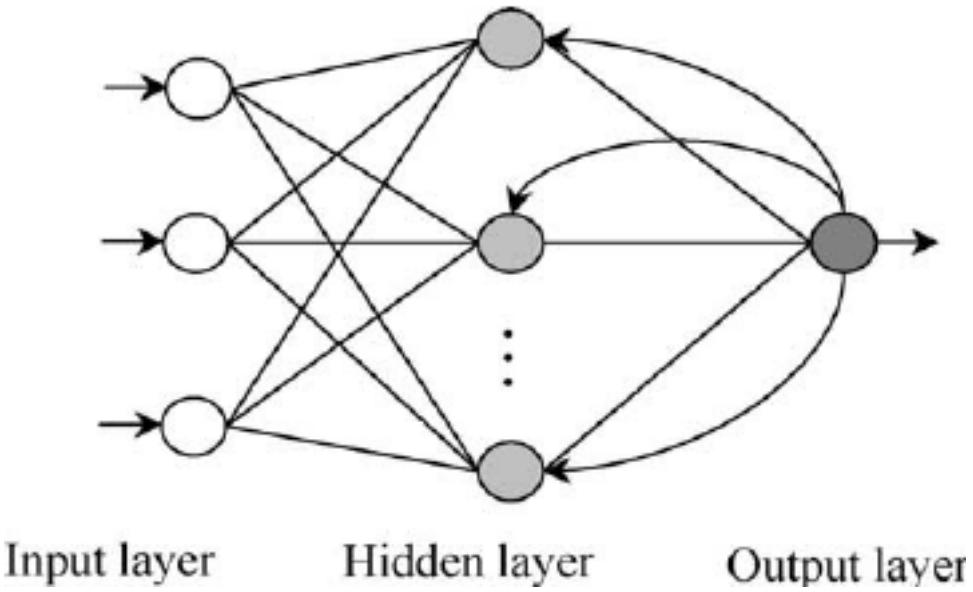

FIGURE 6 – Exemple d'un réseau de neurones avec plusieurs couches. Dans ce cas là, les couches font du *feed-forward* mais de plus, le *feedback* est utilisé afin d'influencer les couches précédant l'*output layer*. [2]

1. Source : http://web.utk.edu/~wfeng1/spark/_images/fnn.png
2. Source :https://jcrisch.files.wordpress.com/2015/04/reseau_de_neurones.png

Concernant la valeur de sortie, elle peut être simple, comme le résultat d'une classification binaire,*i.e.* 0 ou 1 en sortie. Mais elle peut également être d'une dimensionnalité plus élevée comme un vecteur, citons l'exemple d'un point dans un espace R^2.

Afin de borner les valeurs en sortie, la plupart des réseaux utilisent une fonction. Nous pouvons citer :

— La fonction sigmoïde : $S(t) = \frac{1}{1+e^{-t}}$.

— La fonction tangente hyperbolique : $f(x) = tanh(x)$.

Bien souvent, l'utilisation d'un réseau de neurones à une couche est suffisante. Cela est valable pour les fonctions continues, dans le cas de fonctions discontinues, il est intéressant de passer à un réseau disposant de plusieurs couches. Attention toutefois, si le nombre de neurones est trop important, l'algorithme va avoir tendance à sur-apprendre, et à l'inverse, à sous-apprendre si le nombre est trop faible. Il est donc important de bien doser cette quantité afin d'éviter ces problèmes.

Un exemple concret de réseau de neurones est celui présenté dans l'article qui se trouve au cœur du projet [2]. Avant de montrer les équations, en voici un schéma [2] :

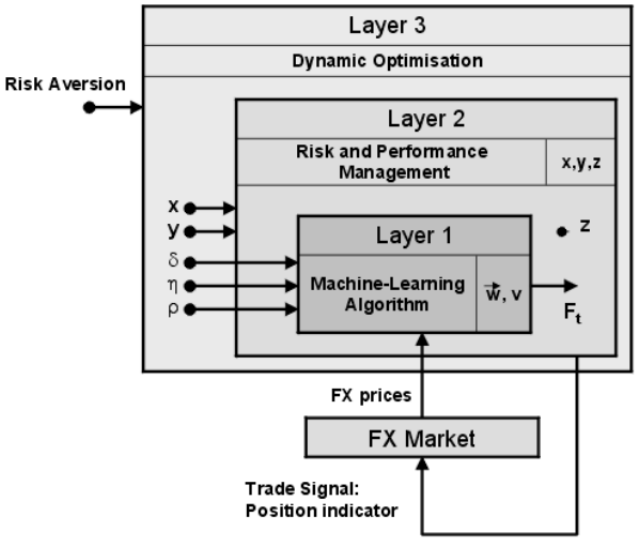

FIGURE 7 – Représentation sous forme de schéma du réseau de neurones. Avec w le vecteur de poids, v le seuil, δ le coût de *trading*, η un paramètre d'adaptation, ρ le taux d'apprentissage, x le seuil d'arrêt lors de pertes, y le seuil de *trading*, z la condition d'arrêt automatique lors de pertes critiques et l'aversion au risque notée v.

Il s'agit de trois couches qui doivent optimiser chacune une série de paramètres. La première va maximiser le vecteur de poids w ainsi que v le seuil. À partir de ces paramètres,

nous pouvons utiliser la formule suivante :

$$F_t = sign(\sum_{i=0}^{M} w_{i,t} r_{t-i} + w_{M+1,t} F_{t-1} + v_t)$$

où $r := p_t - p_{t-1}$ est le retour d'une position. F_t nous permet de calculer la position à prendre au temps t en tenant compte de l'historique des prix.

L'optimisation de w se fait par un algorithme de descente du gradient (voir 2.3.7).

La deuxième couche va travailler à partir des paramètres x, y, z. Il est possible que le marché soit irrationnel durant une longue durée, dans ce cas la psychologie peut pousser à garder une position en espérant un changement. C'est ce genre de comportement qu'un algorithme permet d'éviter. Pour ce faire, nous définissons un excédant des pertes et nous veillons qu'il soit toujours à une distance x du meilleur prix atteint ; afin d'éviter de tenir trop longtemps une position défavorable.

Il est également intéressant de définir un seuil. Si la réponse du réseau est supérieure à ce seuil, nous la prenons en compte et dans le cas contraire, nous ne faisons rien. Cela permet d'évaluer la "force" du signal renvoyé par la première couche. Ce seuil est y.

La dernière variable utilisée dans cette couche est z. Il y a un consensus dans la communauté de *trading* concernant le fait que les algorithmes fonctionnent bien durant un temps puis cesse d'être profitable. À ce moment, il convient d'arrêter le programme et d'y apporter des modifications [1] avant de le relancer. La tâche du paramètre z est de donner un seuil pour les pertes du profit cumulé qui, lorsqu'il est dépassé, lance une procédure d'arrêt. Contrairement à x, y qui seront optimisés par la troisième couche, le paramètre z est fixé au début du programme et ne change plus.

La couche d'optimisation dynamique est la troisième couche. Cette dernière va, à chaque itération, optimiser les paramètres suivants : x, y, η, δ, ρ.

Pour ce faire elle va utiliser [2] :

$$\Sigma := \frac{\sum_{i=0}^{N}(R_i)^2 I(R_i < 0)}{\sum_{i=0}^{N}(R_i)^2 I(R_i > 0)}$$

$$U(\overline{R}, \Sigma, \upsilon) := a \cdot (1 - \upsilon) \cdot \overline{R} - \upsilon \cdot \Sigma$$

,où $R_i := W_i - Wi - 1$ est le retour au temps i avec W_i le profit cumulé, $\overline{R} := \frac{W_N}{N}$ est le profit moyen avec N qui est le nombre d'intervalle, a une constante et υ l'aversion au risque.

Ces équations ont été construites de manière à posséder ces propriétés [2] :

— Une stratégie négative implique un risque très élevé afin d'éviter de soudaines et importantes pertes.

— Avec la définition de Σ, un large total de *trading* défavorable [2] par rapport à l'impact total des opérations favorable [3] conduit à un risque important même si le profit reste

1. Au niveau des équations, des données ou des paramètres.
2. Calculé par : $\sum_{i=0}^{N}(R_i)^2 I(R_i < 0)$.
3. Calculé par : $\sum_{i=0}^{N}(R_i)^2 I(R_i > 0)$.

le même. Cela favorise un système qui augmente de manière monotone ses profits plutôt qu'un autre plus irrégulier.

— La mesure Σ pénalise uniquement les stratégies négatives et pas les stratégies positives.

La fonction à optimiser est donc :

$$\max_{\delta,\eta,\rho,x,y} U(\overline{R}; \Sigma : \delta, \eta, \rho, x, y; \upsilon)$$

Car x, y, η, δ, ρ interviennent tous dans le calcul de Σ. Ces derniers peuvent être optimisés de la manière voulue. L'article ne creuse pas cette fois et utilise une recherche aléatoire composante par composante.

2.3.7 Descente du gradient

L'algorithme de descente du gradient fonctionne sur des fonctions réelles différentiables sur un espace tel que $\mathbb{R}^n$. Il est itératif et fonctionne donc en améliorant l'itération précédente, jusqu'à atteindre une condition d'arrêt.

Avant d'en expliquer la teneur mathématique, voici un exemple de l'algorithme dans $\mathbb{R}^2$:

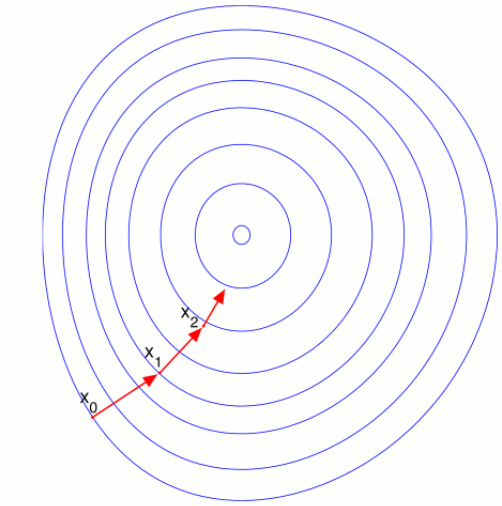

FIGURE 8 – Exemple de l'algorithme de descente du gradient en deux dimensions. À chaque itération, il faut prendre la direction opposée au gradient [1], cela permet d'arriver à un nouveau point. En itérant, on se rapproche de l'optimum.

Afin de comprendre, les divers algorithmes, il est important d'avoir les connaissances mathématiques sur ce sujet. L'algorithme se définit comme suit [2] :

1. Dans cet exemple, il est possible de dire, qu'il faut prendre la normal de la courbe de niveau.
2. Inspiré de [21].

Soit un point initial $x_0 \in \mathbb{R}$. Soit $\epsilon > 0$ un seuil de tolérance. L'algorithme définit une suite d'itération $x_1, x_2, .. \in \mathbb{R}^n$, jusqu'à ce qu'un test d'arrêt soit satisfait. Pour passer de x_i à x_{i+1}, il faut :

- Calculer $\bigtriangledown f(x_k)$
- Si $|| \bigtriangledown f(x_k)|| \leq \epsilon$ alors arrêt.
- Sinon il faut calculer α_k par recherche linéaire [1] sur f en x_k. Cette recherche se fait dans la direction opposée au gradient, soit $- \bigtriangledown f(x_k)$. Une fois α_k calculé, il faut mettre à jour le point itéré :

$$x_{k+1} = x_k - \alpha_k \bigtriangledown f(x_k)$$

La preuve que la recherche dans la direction opposée au gradient induit une décroissance est la suivante. Si la dérivée est non nulle [2] au point x, $f'(x) \neq 0$. Soit

$$d = - \bigtriangledown f(x)$$

Puisque :

$$f'(x) \cdot d = \bigtriangledown f(x) \cdot - \bigtriangledown f(x) = -|| \bigtriangledown f(x)||^2 < 0$$

L'égalité est strictement plus petite car la dérivée est non nulle par hypothèse. Cela implique que :

$$f(x - \alpha \bigtriangledown f(x)) < f(x), \forall \alpha > 0$$

Nous avons donc que pour chaque itération, la valeur obtenue va décroître jusqu'à attendre un maximum ou le seuil ϵ.

La descente du gradient d'un point de vue mathématique peut également être utilisée conjointement à un réseau de neurones. En effet dans cet article [2], les poids w de la première couche, sont optimisés suivant cette formule :

$$w_{i,t} = w_{i-1,t} + \rho \bigtriangleup w_{i,t}$$

Où t correspond au temps, $i - 1$ à l'itération courante et ρ le taux d'apprentissage.

De part la convergence de w_i vers son optimum, cela permet, une fois cette valeur obtenue, de l'injecter comme étant les poids d'un réseau de neurones.

Il existe deux types d'algorithme de descente du gradient :

- Le *Batch Algorithm* a pour but de minimiser la fonction de coût qui aura été définie au préalable en utilisant toutes les données. Ce dernier peut prendre énormément de temps, de par son côté itératif, si l'ensemble d'entraînement croît.

1. La recherche linéaire consiste à choisir une direction de descente afin de minimiser une fonction donnée jusqu'à atteindre l'optimum ou un seuil fixé [20].
2. i.e : Si nous ne sommes pas déjà sur un maximum

— Le *Online Algorithm* va également minimiser une fonction de coût. Contrairement au *Batch Algorithm*, il prendra les exemples un à un. L'optimisation ne se fera que pour l'exemple courant puis recommencera sur une autre donnée. Cette manière de procéder s'applique aisément sur les gros volumes de données.

Provenant de cet article [16], voici des équations pouvant être minimisées dans le cadre d'un programme de *machine learning*. Soit (x_i, y_i), $i = 1..N$, un ensemble de données. Notre fonction de coût est $l(y, y')$. Cela représente le coût de prédire y' quand la réponse est y. Moyennons cela sur tous les exemples :

$$E_N(f) = \frac{1}{N} \sum_{i=1}^{n} l(f_w(x_i), y_i)$$

Où f_w est une fonction pondérée par un vecteur de poids w que l'on cherche à optimiser. Pour optimiser les poids, nous allons utiliser la descente du gradient :

$$w_{k+1} = w_k - \alpha_k \bigtriangledown f_{w_k}(x) \rightarrow w_{k+1} = w_k - \alpha_k \sum_{i=1}^{n} \bigtriangledown Q((x_i, y_i), w_k)$$

Où $Q(x, y) = l(f_w(x), y)$.

Comme mentionné plus haut, la technique de *Batch Algorithm* devient lente lorsque le nombre de données augmente. Pour pallier ce problème, il existe une technique : *stochastic gradient descent*. C'est une méthode d'approximation statistique de la descente du gradient.

Il convient de ne prendre qu'un seul couple au lieu de l'ensemble complet d'entraînement. La somme disparaît donc dans l'équation à minimiser :

$$w_{k+1} = w_k - \alpha_k \bigtriangledown Q((x_i, y_i), w_k)$$

C'est donc une technique de *Online Algorithm*. L'algorithme peut traiter des données à la volée car il n'a pas besoin de se souvenir des exemples précédant.

Il est possible pour pénaliser la complexité de w de rajouter un élément. Cela permet de borner quelque peu les valeurs des poids :

$$w_{k+1} = w_k - \alpha_k \bigtriangledown Q((x_i, y_i), w_k) + \sigma P(w_k)$$

Où $\sigma > 0$ est un hyper-paramètre et P peut valoir :

— **L1 norm** = $P(w) := \sum_{i=1}^{n} |w_i|$

— **L2 norm** = $P(w) := \frac{1}{2} \sum_{i=1}^{n} w_i^2$

— **Elastic Net** = $P(w) := \rho \frac{1}{2} \sum_{i=1}^{n} w_i^2 + (1 - \rho) \sum_{i=1}^{n} |w_i|$

2.3.8 *Majority vote*

Le *majority vote* n'est pas strictement un algorithme de *ML*. Il s'agît plutôt d'améliorer les résultats en modifiant la manière d'utiliser certains des algorithmes pré-mentionnés. Il est donc possible d'employer le *majority vote* avec la régression logistique, les réseaux de neurones, etc.

Le principe est le suivant. Soit E notre ensemble d'entraînement, soit $M = \{f_1, f_2, ...\}$ notre ensemble de méthodes f_i de *machine learning*. Notons $M_E = \{f_{1_E}, f_{2_E}, ...\}$ notre ensemble de méthodes f_{i_E} entraînées.

$$f_{i_E}(x) = \begin{cases} 1 & \text{si la classe calculée de } x \text{ est 1.} \\ 0 & \text{si la classe calculée de } x \text{ est 0.} \\ -1 & \text{si aucune classe n'est trouvée ou si le seuil est insuffisant.} \end{cases}$$

L'algorithme se comporte comme suit pour une observation x :

$\forall f_{i_E} \in M_E$, il faut calculer $f_{i_E}(x)$ et garder la classe résultante dans notre liste de résultat R. Si $\exists\ classe_i \in R$, telle que

$$\sum(classe_i \in R) > \frac{|M_E|}{2}$$

Alors cela signifie que la $classe_i$ est notre résultat, dans le cas contraire, l'algorithme ne donne aucune réponse.

L'algorithme va donc, à partir d'un ensemble de méthodes et d'une observation, calculer les classes. Si une de ces dernières est représentée de manière majoritaire [1], le programme retournera ce résultat. Si la majorité n'est pas atteinte, aucune classe n'est jugée valable et donc aucune réponse ne sera rendue.

Cette technique réduit la composante individuelle des méthodes de *ML* ainsi que le risque d'erreur. Si une observation est mal classifiée, cela signifie que plus de la moitié des algorithmes ont fait une erreur. Dans ce cas, il convient de changer les attributs utilisés ou l'ensemble d'entraînement car l'erreur ne proviendra vraisemblablement pas d'un problème d'implémentation ou de la faiblesse de classification d'un algorithme particulier.

2.3.9 *Random Subset*

À l'instar du *majority vote*, le *random subset* est une technique pour employer des méthodes de *ML*.

1. *i.e.* 50% des voix + 1 voix

Le principe est le suivant :

Il faut prendre N différents ensembles de données [1] du domaine concerné. Ils permettront d'entraîner un algorithme sur chacun d'entre eux [2]. Nous disposerons donc de N instances de l'algorithme de *ML* initial, mais avec un entraînement différent pour chacun, puis pour chaque observation, nous allons utiliser ces N instances afin d'obtenir un résultat de classification.

Comme pour le *majority vote*, si une même classe est représentée une majorité de fois, le *random subset* retournera ce résultat, et aucun le cas contraire.

Il est nécessaire d'avoir un grand jeu de données afin de fournir suffisamment d'échantillons aux instances pour qu'elles apprennent correctement. Une fois ce problème résolu, cette technique nous donne la possibilité d'utiliser au mieux l'ensemble d'entraînement. En le fractionnant, cela permet d'entraîner les algorithmes avec des données différentes et donc d'augmenter l'horizon de connaissance du programme.

De plus, le système de majorité promeut une qualité et une confiance accrue dans les résultats obtenus.

2.4 *Machine Learning* dans le cadre de la finance

2.4.1 Introduction

Avant de parler des performances des algorithmes mentionnés, il convient de préciser que ces derniers proviennent de plusieurs articles différents. Ils ont donc été testés avec des paramètres ayant des valeurs disparates ainsi que sur des données distinctes. Il est donc très compliqué de comparer les résultats des algorithmes entre deux articles différents et d'affirmer qu'une méthode est meilleure qu'une autre.

Pour un même article, il sera possible de comparer les performances cependant dans le cas contraire, ces résultats serviront plutôt à illustrer la qualité intrinsèque des procédés. Avec un résultat de 80%, nous pourrons estimer que la performance est bonne, et l'inverse pour une valeur de 20%.

Pour tous les algorithmes qui ont été mentionnés dans la section précédente, nous allons expliquer les changements appliqués à ces derniers en vue de les étendre au domaine financier. De plus nous étudierons les résultats obtenus et analyserons quelles améliorations apportent un gain significatif dans la classification.

La séparation sera quelque peu différente. Certains articles cumulant plusieurs algorithmes, afin d'éviter les répétitions d'explications, il convient de les séparer par article afin de regrouper les améliorations.

De plus, tous les algorithmes ne sont pas forcément évalués dans les articles. Il est donc possible que certains comme les réseaux de neurones n'aient pas de valeurs.

1. Il est possible de découper notre ensemble initial afin d'atteindre ce critère.
2. Comme *Naive Bayes* ou SVM.

2.4.2 *A Machine Learning Approach to Automated Trading*

2.4.2.1 Introduction

L'auteur a appliqué l'algorithme sur le marché des actions [11] [1]. Après avoir essayé deux approches :

— L'approche individuelle.
— L'approche par secteur.

La première partait de l'hypothèse que l'historique du prix d'une action contenait des motifs permettant la prédiction du prix futur. Pour prédire P_N l'auteur utilisait les N précédant prix, soit : $[P_{N-1}, P_{N-2}, ..., P_1]$. Il s'agissait d'un modèle simple qui ne prenait pas en compte les actions des entreprises concurrentes. Après les premiers résultats, l'auteur a conclu que cette approche était trop simple pour être utilisée, car les valeurs étaient significativement moins bonnes que celle de la seconde approche [11].

La seconde approche, quant à elle, repose sur la supposition que le prix d'une action dépend des autres actions, souvent concurrentes. Il a donc pris en compte l'historique des prix du sous-jacent évalué, mais également celui de ses concurrents en termes de marché. Les données possédées par l'auteur portent sur divers secteurs. Notamment celui de l'*utility*, de l'*energy* et de l'*information technology*. Le procédé est le suivant, pour déterminer si une action précise A_1 va augmenter ou diminuer, il va analyser le prix de l'action durant les N jours précédant mais également celui des M autres actions du secteur donné.

$$f(A_{(1,t_1)}, \ldots, A_{(1,t_N)}, A_{(2,t_1)}, \ldots, A_{(2,t_N)}, \ldots, A_{(M,t_1)}, \ldots, A_{(M,t_N)}) = A_{(1,t_0)}$$

Un autre point qu'il convient d'aborder porte sur l'ensemble d'entraînement. Afin d'entraîner et d'évaluer son programme, il a fallu partager le *set* de données.

Dans le cas contraire, il aurait été impossible de tester les algorithmes entraînés. Ou bien, ils auraient été évalués sur les mêmes données que leur entraînement. Ce qui n'est pas possible.

Le choix a donc été fait de partager l'ensemble de données en deux sous-parties. Une première contenant 80% des données qui sera dédiée à l'entraînement et une seconde avec 20% pour l'évaluation.

L'auteur a utilisé le S&P [2] pour sélectionner des actions. Les données du secteur *utility* en contiennent 29, le secteur energy 39 et le secteur *technology* 61. Tous les *ticks* entre le 02.01.2014 et le 01.02.2016 sont présents dans les données. La taille de l'historique est de quatre jours afin que les vecteurs d'attributs ne soient pas trop grand.

Le but est donc à partir des quatre premiers jours de la semaine, de prédire si l'action sera en hausse ou non le cinquième jour.

1. Ou *Stock market*.
2. Le *Standard & Poor's* est un indice calculé à partir de 500 grandes sociétés capitalisées dans les bourses américaines.

Les trois métriques utilisées sont :

— Le *True Positive Rate* est définit comme suit :

$$TPR = \frac{TP}{TP + FN}$$

Où TP sont les positif détectés positifs et FN les négatifs détectés positif.

— Le *True Negative Rate* est définit comme :

$$TNR = \frac{TN}{TN + FP}$$

— Le *True Rate* :

$$TR = \frac{TP + TN}{TP + TN + FP + FN}$$

2.4.2.2 Résultats

Les résultats proviennent de l'article suivant [11] :

	TPR	*TNR*	*TR*
Utility	0.5595	0.4507	0.5235
Energy	0.4653	0.5369	0.5047
Information Technology	0.5244	0.5031	0.5102

TABLE 1 – Tableau de résultats pour l'algorithme *Lasso Logistic Regression* (voir 2.3.2).

	TPR	*TNR*	*TR*
Utility	0.5699	0.4624	0.5179
Energy	0.4524	0.5320	0.5042
Information Technology	0.5075	0.54966	0.5052

TABLE 2 – Tableau de résultats pour l'algorithme *Decision Tree* (voir 2.3.3).

	TPR	*TNR*	*TR*
Utility	0.5949	0.4957	0.5495
Energy	0.4797	0.5812	0.5193
Information Technology	0.5115	0.5048	0.5091

TABLE 3 – Tableau de résultats pour l'algorithme *Naive Bayes* (voir 2.3.4).

	TPR	*TNR*	*TR*
Utility	0.5804	0.5081	0.5562
Energy	0.4818	0.6049	0.5201
Information Technology	0.5142	0.5040	0.5149

TABLE 4 – Tableau de résultats pour l'algorithme *SVM* (voir 2.3.5).

À ce stade, nous remarquons que les performances de la régression logistique et des arbres de décisions sont du même ordre de grandeur. Ces derniers étant battus par les algorithmes *Naive Bayes* et *SVM*.

Un autre point important concernant le fait que les algorithmes détectent mieux les *TPR* que les *TNR*[1]. Cela est dû au cours des actions des différents secteur globalement en hausse dans les données prises en compte. Du coup, de par le manque de données à la baisse dans l'ensemble d'entraînement, l'apprentissage est limité. Concernant le secteur de l'énergie, vu qu'il était en baisse, l'inverse se produit.

Afin d'améliorer cet aspect l'auteur a utilisé le *majority vote* avec les quatre algorithmes[2] :

	TPR	*TNR*	*TR*
Utility	0.5807	0.4921	0.5573
Energy	0.4753	0.6003	0.5192
Information Technology	0.5133	0.5055	0.5233

TABLE 5 – Tableau de résultats pour le *majority vote* (voir 2.3.8).

Les résultats sont similaires à ceux de *Naive Bayes* et de *SVM*, cela n'est donc pas très concluant comme amélioration. Il est possible que les deux algorithmes ayant les meilleurs résultats sont souvent d'accord sur la classe, cassant ainsi l'intérêt du vote.

L'auteur va utiliser l'algorithme *Naive Bayes* afin de l'améliorer. En se concentrant sur un seul algorithme, il lui est plus facile d'en voir les effets[3].

	TPR	*TNR*	*TR*
Utility	0.5949	0.4957	0.5495
Energy	0.4797	0.5812	0.5193
Information Technology	0.5115	0.5048	0.5091

TABLE 6 – Tableau de résultats pour l'algorithme *Naive Bayes* (voir 2.3.4) avec une optimisation par *ROC curve analysis* (voir 2.3.4).

Le fait d'utiliser la *ROC curve analysis* afin d'obtenir le seuil optimal permet principalement d'augmenter la performance dans la détection des négatifs. Il est important de mentionner que cela n'influence que peu la détection des positifs car ils sont bien représentés dans l'ensemble d'entraînement et donc mieux reconnus. L'amélioration est donc intéressante car sans augmenter le risque de sur-apprentissage, elle augmente la qualité de classification.

1. Exception faite du secteur énergie.
2. Source des résultats : [11]
3. Source des résultats : [11]

	TPR	*TNR*	*TR*
Utility	0.6021	0.5187	0.5779
Energy	0.4574	0.5871	0.5108
Information Technology	0.5348	0.5317	0.5382

TABLE 7 – Tableau de résultats pour le *Random Subset* (voir 2.3.9) avec l'algorithme *Naive Bayes* (voir 2.3.4) optimisé par *ROC curve analysis* (voir 2.3.4).

L'amélioration est assez importante par rapport à l'algorithme *Naive Bayes Roc curve analysis*. Le *Random Subset* obtient 58% et 54% pour l'*utility* et l'information technology, pour seulement 55% et 51% au *Naive Bayes ROC curve analysis*. Même s'il y a une légère perte pour le secteur *energy* d'environ 1%. L'ensemble procure de meilleurs résultats.

2.4.3 *Online Machine Learning Algorithms For Currency Exchange Prediction*

2.4.3.1 Introduction

Cet article a principalement exploré la technique de la descente du gradient (voir 2.3.7) et plus précisément de sa version probabiliste ou *stochastic gradient descent* [16].

Trois variations ont été implémentées et testées :

— La descente du gradient stochastique simple ou *Plain Stochastic Gradient Descent*[1].

— La descente du gradient stochastique avec choix aléatoire ou *Stochastic Gradient Descent with random sample picking*[2].

— La descente du gradient stochastique avec choix aléatoire et "départ chaud" ou *Stochastic Gradient Descent with random sample picking and warm start*[3].

Le *random sample picking* est une technique qui consiste à choisir aléatoirement un élément d'un ensemble de manière non uniforme. Dans ce cadre précis, il est intéressant d'accorder plus de valeurs à une donnée récente qu'à une plus ancienne. On considère que les éléments proches de nous d'un point de vue temporel, ont une plus grande influence.

La meilleure manière d'appliquer cela est d'utiliser une sélection aléatoire exponentielle.

1. Noté : *plainSGD*.
2. Noté : *approxSGD*.
3. Noté : *approxWarmSGD*.

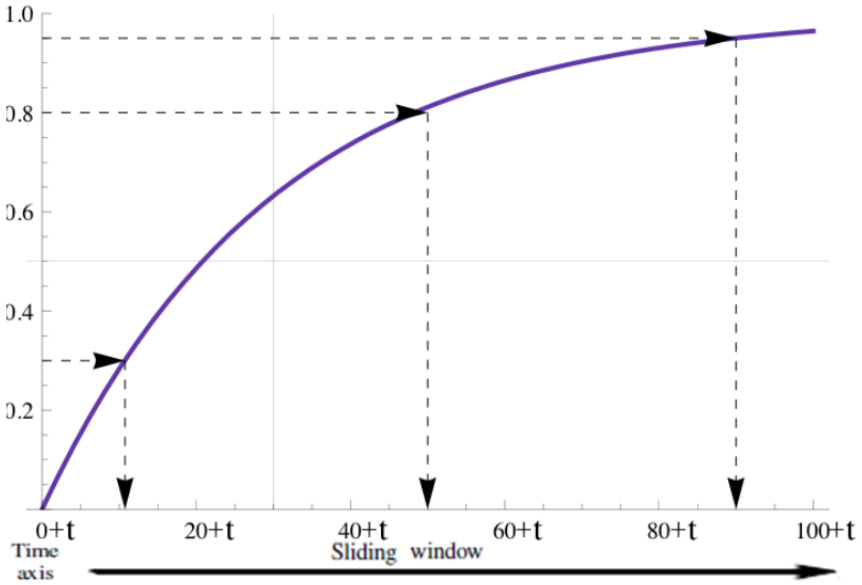

FIGURE 9 – Exemple de distribution exponentielle [1]. Cela va donc permettre de choisir les points les plus proches avec une probabilité plus grande.

L'autre point important est le *warm start*. C'est une pré-optimisation. Il s'agît d'initialiser les paramètres du gradient d'une certaine manière afin d'augmenter la vitesse et les chances de convergence. Pour ce faire, la meilleure option trouvée dans l'article consiste à utiliser les valeurs obtenues lors de la précédente itération [16]. Par conséquent, l'algorithme convergera plus vite pour un coût plus faible en termes d'erreur de calcul.

La métrique d'évaluation est différente des notions de TPR, TNR et TR. Dans cet article, le but est de prédire le prix d'un cours, comme EUR/USD à partir de plusieurs autres, comme EUR/CAD, EUR/AUD, EUR/GBP. Il est donc extrêmement compliqué de calculer avec plusieurs décimales le résultat exacte, impliquant donc des taux nuls pour chacune des métriques. Il fallu donc trouver une méthode basée sur l'erreur relative :

$$relative_{error} = \frac{\Delta x}{x}$$

Cela quantifie la différence entre le résultat calculé et le résultat réel tout en le pondérant.

2.4.3.2 Résultats

Le but est de calculer un taux de change à partir de données. Pour obtenir ces résultats précis, l'auteur a pris le *Singapore Hedge Fund FOREX Data Set* ainsi que le *Capital K*

1. Source : [16]

FOREX Data Set. Les dates ainsi que la fréquence ne sont pas précisées dans l'article.

L'algorithme retourne deux valeurs : le *bid* et le *ask* à partir des données fournies. Ci-dessous, nous allons voir la précision de ces résultats. Les tableaux représentent la comparaison entre les valeurs calculées et les valeurs réelles, soit l'erreur relative.

Données	*plainSGD*	*approxSGD*	*approxWarmSGD*
	Bid/Ask	*Bid/Ask*	*Bid/Ask*
Singapoore with SW-1 [1]	0.1366274551%	0.133887583%	0.2294997927%
Singapoore with SW/2 [2]	0.1366274551%	0.09281447603%	0.1551851906%

TABLE 8 – Tableau de résultats [3] pour les différentes versions de *SGD* sur le *FOREX* de Singapore. Les différences entre le *bid* et le *ask* étant minimes, l'auteur ne les a pas consignées.

Les erreurs relatives sont très faibles. Les estimations sont donc très proches des valeurs réelles. De plus, on remarque que le changement de la taille de la SW peut avoir une grande influence. Sur le *plainSGD* cela ne change rien, on peut donc en conclure que les $N/2-1$ données supplémentaires ne sont pas significatives dans le calcul. Pour le *approxSGD* et le *approxWarmSGD*, le taux d'erreur diminue. Nous pouvons donc dire que la fenêtre était trop grande et, pire encore, ajoutait du bruit rendant le calcul moins précis.

D'un point de vue calculatoire c'est très intéressant car il est possible de diminuer la taille de la fenêtre et donc gagner en vitesse tout en gardant, voir en augmentant, la qualité des résultats.

Données	*plainSGD*	*approxSGD*	*approxWarmSGD*
	Bid/Ask	*Bid/Ask*	*Bid/Ask*
Capital K SW-1	0.01167%/0.0117%	0.0116%/0.0117%	1.502e-03%/1.554e-03%
Capital K SW/2	0.01167%/0.0117%	0.0314%/0.0313%	1.701e-03%/1.751e-03%

TABLE 9 – Tableau de résultats [4] pour les différentes versions de *SGD*.

Nous constatons qu'il n'y a pas de différence pour le *plainSGD* entre les deux tailles de fenêtres. On remarque cependant que pour les deux autres variantes, la diminution de la taille les pénalise. Cependant, l'ordre de grandeur des erreurs demeure faible. Il est donc important de savoir si nous voulons un chiffre précis ou obtenir le résultat de manière rapide. L'objectif aiguillera le choix pour l'une ou l'autre des fenêtres.

À noter, que le *approxWarmSGD* obtient de meilleures valeurs que ces concurrents et subit beaucoup moins les effets de la diminution de SW que le *approxSGD*.

1. $SW-1$ signifie une fenêtre de choix de taille $N-1$.
2. $SW/2$ signifie une fenêtre de choix de taille $N/2$.
3. Source des résultats : [16].
4. Source des résultats : [16]

2.5 Conclusion

Afin d'élaborer des algorithmes de *ML*, il est important de saisir les considérations théoriques. À partir de ces connaissances, le choix d'un algorithme est plus aisé. En effet, si vous voulez apprendre une fonction continue, mieux vaut utiliser des techniques de descente de gradient et dans le cas de données faiblement différentiables, l'utilisation d'un programme *SVM* avec un *kernel trick* est recommandée.

Cela est également valable pour sa mise en place. Les optimisations mathématiques possibles sont nombreuses et complexes, comme nous l'avons vu dans la partie (2.4), il conviendra donc de les comprendre afin de mieux cerner les contraintes et les gains lors de l'optimisation.

Ces éléments peuvent être vus comme une boîte à outils algorithmiques, le choix et l'utilisation des différents outils reviendra à la personne qui programmera. Ce sera cette dernière qui devra construire l'algorithme le plus adapté avec les données, les contraintes et les techniques à sa disposition.

Les considérations mathématiques ne sont pas les seuls éléments importants. Appliquer un algorithme sans chercher à comprendre les cas concrets peut mener à des programmes peu efficaces.

Dans chacun des articles, les auteurs avaient une connaissance et une compréhension du monde de la finance suffisante, pour améliorer les techniques en dehors du carde mathématique. L'approche par secteur [11] ou l'utilisation de méta-paramètres propre au domaine financier [2] en sont des exemples concrets. Chacun de ces éléments a sensiblement amélioré les performances des algorithmes auxquels ils ont été appliqués.

Une technique de *machine learning* n'est qu'une solution à un problème mathématique précis. L'ajout d'éléments, comme ceux cités auparavant, améliore la quantité d'informations disponible et précise l'équation mathématique à optimiser.

Il est important de lier ces deux parties pour obtenir des résultats optimaux.

Les valeurs peuvent être encourageantes même s'il convient de relativiser les résultats. Ces derniers ayant pu être obtenus sur des ensembles de données "faciles". Ce mot qualifie des périodes avec peu de changements ou peu de variations, ce qui facilite grandement la classification. Néanmoins, les taux d'erreurs sont faibles et la classification efficace. Cela démontre que les bons algorithmes appliqués avec une bonne connaissance du domaine fournissent des résultats satisfaisants.

Le choix d'implémenter l'algorithme de réseau de neurones provient des justifications suivantes :

— Les *Neural Nets* peuvent, avec assez de ressources [1], approximer n'importe quel fonction. De plus ils sont capables d'opérer des séparations non linéaires sans recourir au *Kernel trick*.

1. Cela comprend le temps et les données.

— Il est possible d'éviter les problèmes de sur-apprentissage, en modifiant les méta-paramètres. Cela nous permet de profiter de n'importe quel ensemble d'entraînement sans craindre que ce dernier pose problème.

Les algorithmes d'arbres de décision sont des classifieurs linéaires. Ce qui signifie qu'ils sont limités lorsque les données ne sont pas linéaires, réduisant de fait, la qualité de classification. On retrouve ce même problème pour la régression logistique. De plus les arbres de décision (voir 2.3.3) ont de gros risques de sur-apprentissage même avec un ensemble d'entraînement adapté, sans compter qu'ils gèrent mal les exemples en continues. En effet, lors d'un entraînement *online*[1] chacun des nouveaux éléments peut introduire des exceptions. Ce qui implique de reconstruire l'arbre, ce cas pouvant être très lourd en termes de calculs, il convient donc de l'éviter.

Le classifieur *Naive Bayes* aurait pu être une solution car ce dernier peut classifier les données non linéaires, ne présente pas de problème de sur-apprentissage et est adapté pour un apprentissage *online*. Il suffit de rajouter le dernier exemple dans l'équation pour mettre à jour l'apprentissage. La raison qui nous a poussé à choisir les réseaux de neurones plutôt que l'algorithme de *Naive Bayes* est que ce dernier a des performances limitées sur un grand ensemble de données. Si l'on prend des petits jeux de données, *Naive Bayes* est très efficace[2] [3]. Cependant si l'ensemble grandit, ces performances plafonnent, ce qui n'est pas le cas du *Neural Nets* dont les résultats augmentent à mesure que le nombre de données disponibles croît.

Au vu des raisons susmentionnées, un algorithme de réseau de neurones nous semblait le meilleur choix, car le *FOREX* nous fournit suffisamment de données pour avoir d'excellent résultats, de plus, nous ne sommes pas limités par une classification linéaire.

1. Quand les exemples arrivent de manière continue.
2. Dans ce cadre précis, il est même meilleur que tous les autres algorithmes vus dans ce projet.

3 Projet

3.1 Introduction

3.2 Algorithme

Dans cette section, pour chacune des parties de l'algorithme, nous allons dans un premier temps expliquer son fonctionnement théorique et ses bases mathématiques, puis exposer la manière dont nous avons implémenter ces éléments. Une fois l'algorithme détaillé, nous présenterons les résultats pour les différentes variantes, en analysant ce qui à conduit aux valeurs obtenues. En guise de conclusion, les points problématiques seront abordés et détaillés afin que nous puissions dans la section suivante proposer des solutions à ces derniers.

3.2.1 *Layer 1*

Le rôle du *Layer 1* est de fournir un signal. Dans l'article, ce signal peut avoir comme ensemble de valeurs : $\{-1, +1\}$. Le calcul du dit signal suit la formule suivante [2] :

$$F_t = sign(\sum_{i=0}^{M} w_{i,t} r_{t-i} + w_{M+1,t} F_{t-1} + v_t$$

où r_t est le *return* au temps t obtenu par $r_t = p_t - p_{t-1}$, $w_{i,t}$ est le ième poids de l'itération t, v_t est un seuil et F_{t-1} le résultat de l'itération $t-1$.

Cette formule mathématique implémente une réseau de neurones. Il s'agît d'une multiplication vectorielle entre les poids (w) et les *returns* (r_i). Les poids ont pour but de donner une pondération à chacun des éléments afin de donner le résultat le plus "juste" possible, suivant certains critères [1] Dans notre cas, le réseau de neurones disposent d'une seule couche mais cette couche est récurrente. Cela signifie que l'*output*[2] à l'itération $t-1$ est utilisée dans le calcul du signal de t. Cette construction permet d'augmenter la quantité d'informations disponibles par le réseau afin d'améliorer les résultats. Un seuil, v_t est ajouté dans la formule afin de lisser les résultats et d'éviter des variations trop fréquente.

Afin d'obtenir de bons résultats, ils convient de trouver les valeurs de w_t et v_t qui vont maximiser une fonction de coût. L'optimisation choisie est une descente du gradient (2.3.7). La fonction de coût est définie comme cela :

$$D_t := \frac{d\widehat{S}(t)}{d\eta}|_{\eta=0} = \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{\frac{3}{2}}}$$

où $\Delta A_t := (R_t - A_{t-1})$ et $\Delta B_t := (R_t^2 - B_{t-1})$.Il faut appliquer un développement de Taylor à $\widehat{S}(t)$ en $\eta = 0$ La définition de $\widehat{S}(t)$ est la suivante :

$$\widehat{S}(t) := \frac{A_t}{B_t}$$

1. Nous développerons ces critères plus loin.
2. Notre signal $\in \{-1, +1\}$

où $A_t := A_{t-1} + \eta(R_t - A_{t-1})$ et $B_t := B_{t-1} + \eta(R_t^2 - B_{t-1})$. De manière intuitive, A_t représente l'espérance et B_t la volatilité. Le but est donc d'obtenir une grand espérance avec peut de risque. La valeur théorique maximale de la fonction $\widehat{S}(t)$ doit tendre vers l'infini.

Il reste encore à définir R_t, qui quantifie le gain au temps t :

$$R_t := F_{t-1}r_t - \delta|F_t - F_{t-1}|$$

où δ est le coût de transaction. La première partie $F_{t-1}r_t$ estime le gain en fonction de la position et du *return*. Plusieurs cas de figures se présentent :

— Si $sign(F_{t-1}) = sign(r_t)$, alors la multiplication de ces deux éléments donnera un nombre supérieur ou égal à 0.
— Dans le cas contraire, le résultat sera négatif.

Si nous avons deviné correctement la direction du cours, alors nous faisons un profit et dans le cas contraire une perte.

La deuxième partie $\delta|F_t - F_{t-1}|$ ajoute les coûts de transactions dans le calcul du profit. Cela découle du fait que prendre une position à un coût. Les différents cas sont :

— Si $F_t = F_{t-1}$ alors la différence est nulle et de même pour le coût de transaction.
— Dans le cas contraire, cela signifie que nous avons dans un premier temps fermé une position, puis ouvert une autre. Il y a deux transactions effectuées et donc 2δ

Le profit n'est donc pas uniquement dépendant du signal et du *return* mais également du signal de l'itération précédente.

Le profit cumulé est définit comme la somme des profits individuels :

$$P_T = \sum_{t=0}^{T} R_t$$

3.2.2 *Layer 2*

3.2.3 *Layer 3*

3.2.4 Résultats

3.2.5 Points problématiques

3.3 Solution

3.3.1 Points problématiques

3.3.2 Remédiations

3.3.3 Résultats

3.4 Conclusion

Références

[1] Vincent Cheung and Kevin Cannons. An introduction to neural networks. http://www2.econ.iastate.edu/tesfatsi/NeuralNetworks.CheungCannonNotes.pdf.

[2] M.A.H Dempster and V. Leemans. An automated fx trading system using adaptive reinforcement learning, 2004.

[3] George Forman and Ira Cohen. Learning from little : Comparison of classifiers given little training. www.ifp.illinois.edu/~iracohen/publications/precision-ecml04-ColorTR-final.pdf.

[4] David Guerreiro. Chapitre 1 : Le marche des changes monnaie et finance internationales. https://economix.fr/docs/1045/chap_1_2015-16.pdf.

[5] Investopedia. American option. http://www.investopedia.com/terms/a/americanoption.asp.

[6] Investopedia. Backtesting. http://www.investopedia.com/terms/b/backtesting.asp.

[7] Investopedia. Bid and asked. http://www.investopedia.com/terms/b/bid-and-asked.asp.

[8] Investopedia. European option. http://www.investopedia.com/terms/e/europeanoption.asp.

[9] Investopedia. Option. http://www.investopedia.com/terms/o/option.asp.

[10] Anil K. Jain, Jianchang Mao, and K.M. Mohiuddin. Artificial neural networks : A tutorial. http://csc.lsu.edu/~jianhua/nn.pdf.

[11] Ning Lu. A machine learning approach to automated trading, 05 2016.

[12] Estelle Mermet. La règle des trois unités du marché des changes. http://www.forex.fr/newslist/8696-la-regle-des-trois-unites-du-marche-des-changes.

[13] Tom Mitchell. Descision tree learning.

[14] Tom Mitchell. Machine learning, 1997.

[15] Ni, Jiarui, and Chegqi Zhang. An efficient implementation of the backtesting of trading strategies, 2005.

[16] Eleftherios Soulas and Dennis Shasha. Online machine learning algorithms for currency exchange prediction.

[17] Financial Times. Real investors eclipsed by fast trading. https://www.ft.com/content/da5d033c-8e1c-11e1-bf8f-00144feab49a?mhq5j=e1, 2012.

[18] Jason Weston. Support vector machine (and statistical learning theory) tutorial. http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf.

[19] Addison Wiggin. Comment on est passé de l'étalon-or à l'étalon-dollar. http://la-chronique-agora.com/etalon-or-etalon-dollar/.

[20] Wikipedia. Line search. https://en.wikipedia.org/wiki/Line_search.

[21] Wikipédia. Algorithme du gradient. https://fr.wikipedia.org/wiki/Algorithme_du_gradient.

[22] Wikipédia. Broker. https://en.wikipedia.org/wiki/Broker.

[23] Wikipédia. Les accords de bretten woods. https://fr.wikipedia.org/wiki/Accords_de_Bretton_Woods.

[24] Wikipédia. Régression statistique. https://en.wikipedia.org/wiki/Regression_analysis.

[25] Wikipédia. Support vector machine. https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support.

[26] John Wiley and Sons. Algorithmic trading : Winning strategies and their rationale (wiley trading series), 2013.