SEGMENTATION DES RÉGIMES DU MARCHÉ

Rapport final PSC ECO 11

13 avril 2020

Abdelhafid SOUILMI , Farouk KHLIFI
Houssam EL CHEAIRI , Richard MALHA
Taha EL GHAZI

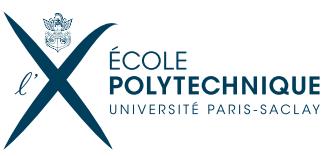


TABLE DES MATIÈRES

INTRODUCTION :

CONTEXTE

Un défi constant pour la prévision des marchés est la modification fréquente de leurs comportements, souvent de manière abrupte, en raison des périodes changeantes de la politique gouvernementale, de l'environnement réglementaire et d'autres effets macroéconomiques. Ces périodes sont communément appelées régimes de marché et la détection de tels changements est un processus courant, quoique difficile, entrepris par les acteurs quantitatifs du marché.

Ces différents régimes conduisent à des ajustements des rendements des actifs via des changements de leurs moyens, des variances / volatilités, des corrélations en série et des covariances, qui impactent l'efficacité des modèles qui reposent sur la stationnarité.

Une compréhension précise des régimes du marché facilite la création de meilleures stratégies d'allocation d'actifs et des prévisions de liquidité plus précises. Cependant, cela nécessite la capacité d'analyser des marchés et des données économiques pour découvrir les caractéristiques clés de chaque régime. Plus récemment, cette analyse peut reposer, grâce aux progrès en informatique, sur des jeux de données très volumineux impossible à traiter par un humain. A l'aide de cette nouvelle puissance de calcul, les prises de décision peuvent se faire de manière plus rapide (jusqu'au millième de seconde), et surtout avec assez d'informations pour avoir plus de recul sur les achats et les ventes.

Tous ces éléments imposent une analyse des différents outils que l'on peut utiliser pour identifier les régimes du marchés et les incorporer dans des modèles prédictifs. C'est là qu'entre en jeu le véritable centre d'intérêt de ce PSC.

Nous nous sommes proposés d'implémenter des algorithmes pour la segmentation et la prévision de l'évolution du marché, avec l'aide de Napoléon Capital. Napoléon Capital est une entreprise de conseil financier qui propose à ses clients des stratégies et des algorithmes d'investissement. Sous leur encadrement, nous avons eu pour objectif de travailler principalement sur le S&P500, un indice boursier basé sur 500 grandes sociétés cotées sur les bourses aux Etats-Unis, ainsi que sur multiples features macro-économiques. Nous avons alors eu accès aux valeurs du S&P500 sur plus de 20 ans, ainsi que d'autres indices qui permettront d'effectuer nos analyses. Notre stratégie est alors d'effectuer une segmentation des régimes du marché permettant une meilleure performance des modèles, en utilisant des méthodes comme les Hidden Markov Models (HMM) ou en utilisant des méthodes de segmentations, et de tester à la fois l'interprétabilité des différents régimes obtenus ainsi que leur capacité à améliorer les modèles prédictifs.

NAISSANCE DU PSC ET RÉPARTITION DES TÂCHES

Notre point de départ dans la recherche du sujet du PSC était notre envie de découvrir les différentes méthodes utilisées dans le monde de l'apprentissage statistique en particulier et en intelligence artificielle plus généralement. Donc, on a décidé de contacter le CEO de l'entreprise Napoléon Capital

M. Stéphane Ifrah (X92), que certains membres de notre groupe ont rencontré à travers l'association X-Finance, qui nous a proposé un sujet sur la segmentation et la prédiction du marché.

Muni de cette idée, nous sommes allés voir M. Guillaume Hollard, chercheur au CREST dans le domaine de décision publique et individuelle et des méthodes expérimentales en économie qui est aussi le coordinateur des PSC en économie afin lui présenter le sujet, pour discuter des différentes pistes que l'on peut suivre. Notre première tâche a été de nous familiariser avec les différentes méthodes utilisées pour l'apprentissage ainsi qu'avec l'aspect mathématique de certains modèles (HMM). Cela étant fait, nous avons décidé de partager le travail en deux grandes parties : la première consistait à utiliser le modèle des chaînes de Markov cachées (HMM) pour segmenter et prédire le marché tandis que la deuxième consistait à utiliser des méthodes d'apprentissage pour segmenter le marché à l'aide des méthodes de clustering ensuite adapter sur chaque segment obtenu un modèle adéquat.

La répartition du travail au début s'est faite selon ces deux grandes parties, Houssam et Farouk se sont chargés de la partie HMM : ils se sont tous les deux partagés les parties de l'implémentation informatique et mathématique, et Richard, Abdelhafid et Taha se sont chargés de la partie de l'apprentissage statistique : Abdelhafid et Taha se sont chargés de la partie de l'implémentation des modèles sur chaque segments tandis que Richard s'est chargé de la partie du clustering.

Le deuxième groupe s'est affronté à quelques résultats contre-intuitifs et à un plafonnement des performances des modèles, en particulier en combinant entre les données de nature macro-économique et les données liées directement à l'indice, les résultats trouvés divergeaient de nos attentes, donc deux nouvelles problématiques ont émergées : l'étude de la relation entre ces deux types de données afin de mieux comprendre l'effet qui se produisait d'une part, d'autre part on a décidé d'explorer des modèles de Deep learning, plus précisément les réseaux de neurones pour évaluer si la performance pourrait être améliorer. Le premier groupe s'est chargé de la partie du deep learning tandis que le deuxième s'est chargé de l'étude de la relation entre les deux types de données. Certes, le travail était bien réparti par groupe, néanmoins toutes les grandes décisions concernant le projet ont été prises en groupe, et on se consultait les uns les autres pour assurer un bon avancement du travail.

1
MOTIVATIONS DERRIÈRE LA SEGMENTATION DU MARCHÉ

Afin de mieux analyser et comprendre la dynamique des marchés financiers, plusieurs modèles probabilistes ont été adoptés. Les premières approximations historiques du mouvement du marché modélisent ce mouvement en utilisant des marches aléatoires, ou un mouvement dit Brownien. Les chaînes de Markov ont été ainsi un terrain fertile pour le développement de modèles de plus en plus complexes cherchant à prédire l'évolution du marché. Le mouvement Brownien est un des plus anciens modèles. Il modélise le prix d'une action par la sous-martingale (sur-martingale si $\nu < 0$) définie par :

$$S_{j+1} = S_j exp^{\nu\Delta t + \sigma\varepsilon_j\sqrt{\Delta t}}$$

avec : S_j le prix de fermeture au temps j et ε_j qui modélise l'erreur Gaussienne. Il s'agit donc de dire, qu'il n'existe pas un meilleur moyen pour prédire le prix de demain que le prix d'aujourd'hui. Ceci suppose donc que le prix d'aujourd'hui contient toute l'information utile sur le marché et présuppose une efficience forte du marché. Il s'en suit alors que les returns centrés

$$Z_j = log(\frac{S_{j+1}}{S_j}) - \nu\Delta t$$

sont un processus stationnaire [1].

Cependant, quelque soit le modèle de prédiction de la série temporelle , il est nécessaire de vérifier au préalable si notre série est stationnaire avant de tenter de construire un modèle. Rappelons, donc rapidement la définition de la stationnarité d'un processus stochastique : Un processus Z_t à temps discret est dit stationnaire au sens fort si et seulement si pour toute f fonction mesurable on a :

$$f(Z_1, Z_2, ..., Z_t) \quad \texttt{et} \quad f(Z_{1+k}, Z_{2+k}, ..., Z_{t+k}) \text{ sont de même lois}$$

Une telle définition peut être relâchée avec la stationnarité au sens faible définie comme suit : Un processus est dit stationnaire au sens faible (ou « de second ordre », ou « en covariance ») si

$$E[Z_i] = \mu \qquad \forall i = 1...t$$

$$Var[Z_i] = \sigma^2 \neq \infty \qquad \forall i = 1...t$$

$$Cov[Z_i, Z_{i-k}] = f(k) = \rho_k \qquad \forall i = 1...t, \quad \forall k = 1...t$$

Afin de tester l'hypothèse de stationnarité, la deuxième définition est d'une plus grande utilité, car elle permet de construire des tests statistiques et ainsi construire une approche expérimentale, qui consiste à calculer et afficher le graphe des moyennes, variances et autocorrelations glissantes des returns qui devraient être constantes si les logs returns sont stationnaires.

Une approche plus rigoureuse consiste à faire des tests statistiques de la stationnarité faible (d'ordre 2) comme le test de Dickey-Fuller ou bien le test d'Okabe Nakano [2] qui montre que les log returns ne sont très souvent pas stationnaires et rejette donc l'hypothèse du modèle Brownien. La figure ci-dessous représente les courbes de l'évolution de la moyenne glissante des returns ainsi que de la variance et des autocorrelations.

On remarque l'existence de plusieurs pics de variance ainsi que des autocorrelations instables. Les tests déjà présents dans la littérature ainsi que cette représentation simple de la figure 1 nous indique que le fait d'adopter des stratégies d'investissement adaptées à chaque régime, ou intervalle de stationnarité du marché pourrait mener à de meilleurs résultats.

2
PREMIÈRES APPROCHES DE SEGMENTATION

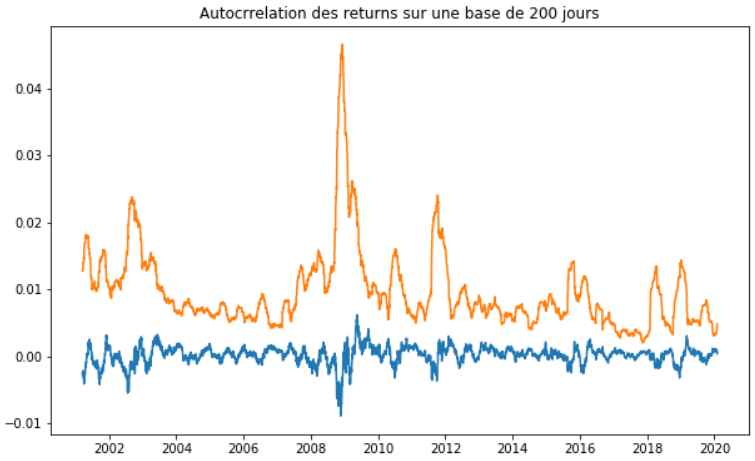


FIGURE 1 – Moyenne glissante des returns

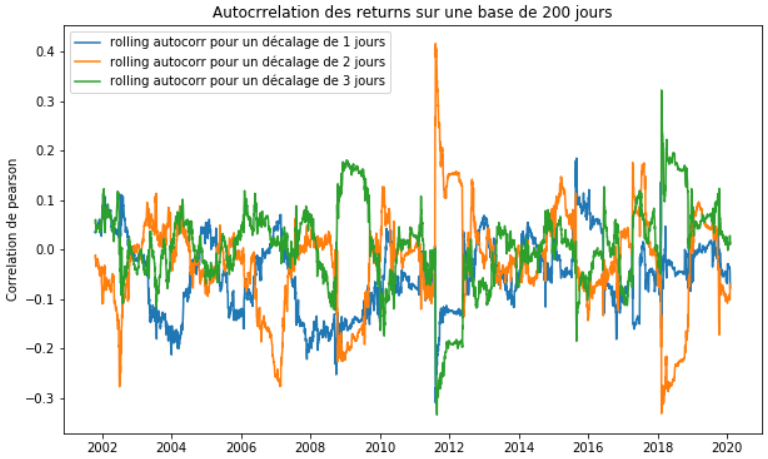


FIGURE 2 – Différentes autocorrélations

2.1 MODÈLES UTILISÉS

Commençons par rappeler que dans l'apprentissage statistique et la segmentation, il existe deux types d'apprentissage, l'un supervisé et l'autre non supervisé. L'apprentissage supervisé consiste à induire une fonction f de notre modèle à partir de $D = \{(X_i, y_i), i \in 1...n\}$ tel que $f(X_i)$ soit la prédiction de y_i tandis que l'apprentissage non supervisé construit un modèle sans lui donner les labels y_i au préalable.

Pour la segmentation nous allons nous intéresser à des méthodes d'apprentissage non supervisé, parce qu'on ne dispose pas des caractéristiques ou d'une connaissances préalables des états cachés.

Les modèles les plus utilisés dans la littérature sont représentés ci-dessous [5] :

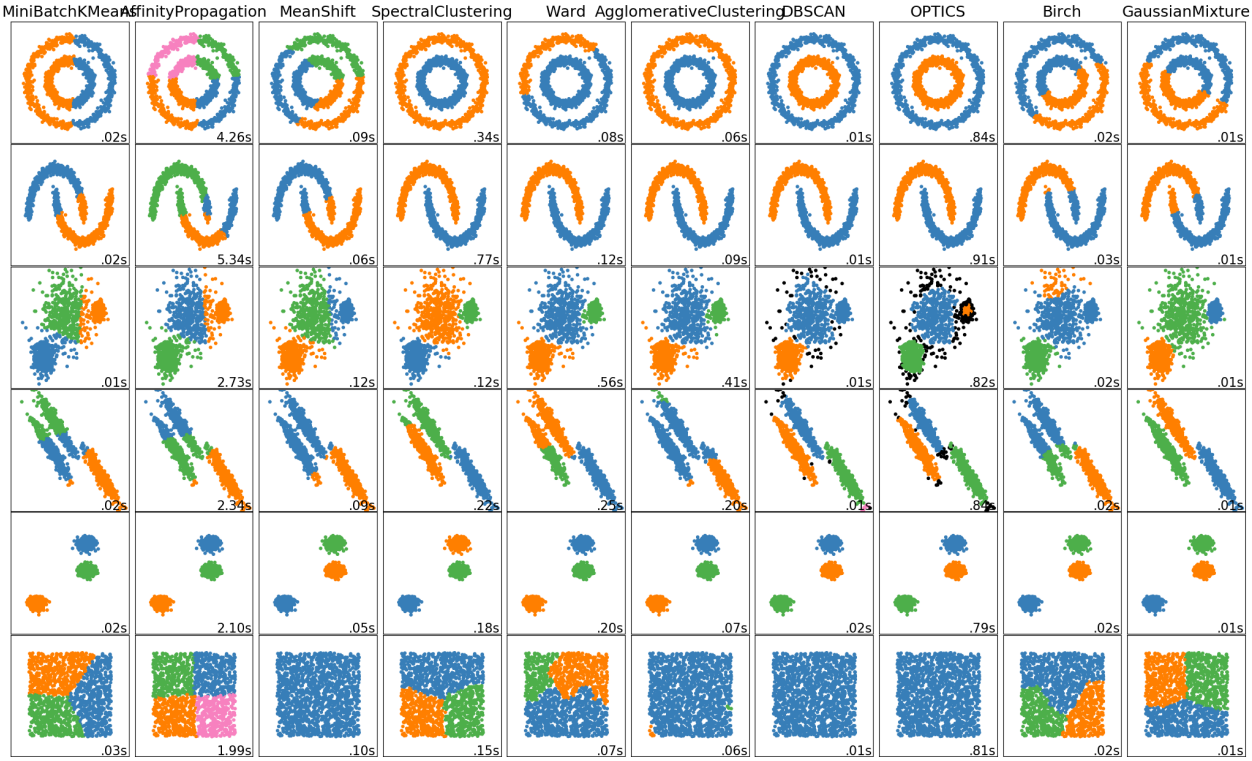


FIGURE 3 – Différentes méthodes de clustering

On peut distinguer entre les modèles d'apprentissage qui associe à chaque observation X un cluster c et ceux qui lui associe un vecteur de probabilité d'appartenance à chaque cluster $(p_1, p_2, ...p_{|C|})$.

Dans la section 4, nous évaluons la pertinence de nos segmentations en entraînant des modèles d'apprentissage supervisé dessus.

2.2 CHOIX DU MODÈLE ET RÉGLAGE DES HYPERPARAMÈTRES

La première difficulté dans notre situation est donc de choisir le nombre de clusters à utiliser [7]. On peut donc commencer à implémenter des algorithmes ne nécessitant pas une connaissance préalable du nombre de clusters afin d'effectuer notre prédiction. Citons parmi ces algorithmes là le `MeanShift` [5] qui consiste à faire translater itérativement des centroïdes initialement distribuées de la même façon que les données, dans la direction du barycentre des points qui sont à distance $< D$ du

centroïde.

Algorithm 1: Algorithme `MeanShift`

initialization;

All data points are initialized to cluster centroids ; $m_x = \frac{\sum_{i \in N(x)} K(x-x_i)x_i}{\sum_{i \in N(x)} K(x-x_i)} \forall x \in$ `Dataset` ;

while $\|x - m_x\| > \varepsilon$ **do**

$x \leftarrow m(x)$;

$m_x = \frac{\sum_{i \in N(x)} K(x-x_i)x_i}{\sum_{i \in N(x)} K(x-x_i)} \forall x \in$ `Dataset` ;

end

Cette approche translate naturellement les points vers les zones à forte densité. Le modèle nous permet donc d'avoir des clusters de tailles différentes, chose qui semble être assez intéressante dans notre problème. Le seul hyperparamètre à déterminer dans le cas de la fonction `MeanShift` est la `bandwidth` paramètre associé à D qui peut être estimé au préalable avec d'autres moyens statistiques basés sur les *nearest neighbours*. Cette méthode nous donne 4 clusters (voir figure 8), avec un quatrième cluster quasi inexistant (sur une dizaine de jours répartis).

Nous nous sommes intéressés aussi à la méthode de segmentation k-moyennes (*kmeans*) qui étant donné un ensemble de point $x_1, x_2, ...x_n$ cherche à les partitionner en k ensembles $S_1, S_2, ...S_k$ tout en minimisant la distance entre les points de chaque partition, donc le problème peut être reformulé comme suit :

$$argmin_S \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||^2$$

Pour utiliser cette méthode, il est nécessaire de déterminer le nombre de clusters k optimale. Le choix de ce paramètre est crucial : une valeur petite va nous fournir un modèle sous-entraîné (underfit) tandis qu'une valeur élevée va nous fournir un modèle surentraîné (overfit). Une des méthodes classiques qui peut permettre de trouver la valeur de k, est la visualisation des données après une Analyse de Composantes Principales (*PCA*) et identifier le nombre de nuages dans cette visualisation. Néanmoins, cette méthode n'a pas abouti dans notre cas. Nous avons décidé d'implémenter une autre méthode dite *elbow* de la classification kmeans qui consiste à tracer l'évolution de diamètre maximale des clusters en fonction de leur nombre k. Cette méthode permet de nous donner une idée sur le meilleur nombre de clusters à entrer en paramètre en sélectionnant le point le plus anguleux du graphe.

Nous voyons que sur la figure 4 le point le plus anguleux est le point qui correspond à n=3.

2.3 CHOIX DES ATTRIBUTS POUR LA SEGMENTATION

Un attribut ou une feature est une fonction des différentes données relatives à l'actif sur une durée T. Le choix des attributs ou features est critique car c'est à travers elles qu'on cherche à établir notre modèle (dans ce cas trouver les segments). Par exemple, une feature pourrait représenter l'effet du prix du pétrole ou des variables macro-économiques de croissance, une autre feature pourrait représenter l'effet des tweets de D.Trump. Dans notre cas on a commencé par nous intéresser d'abord aux "daily

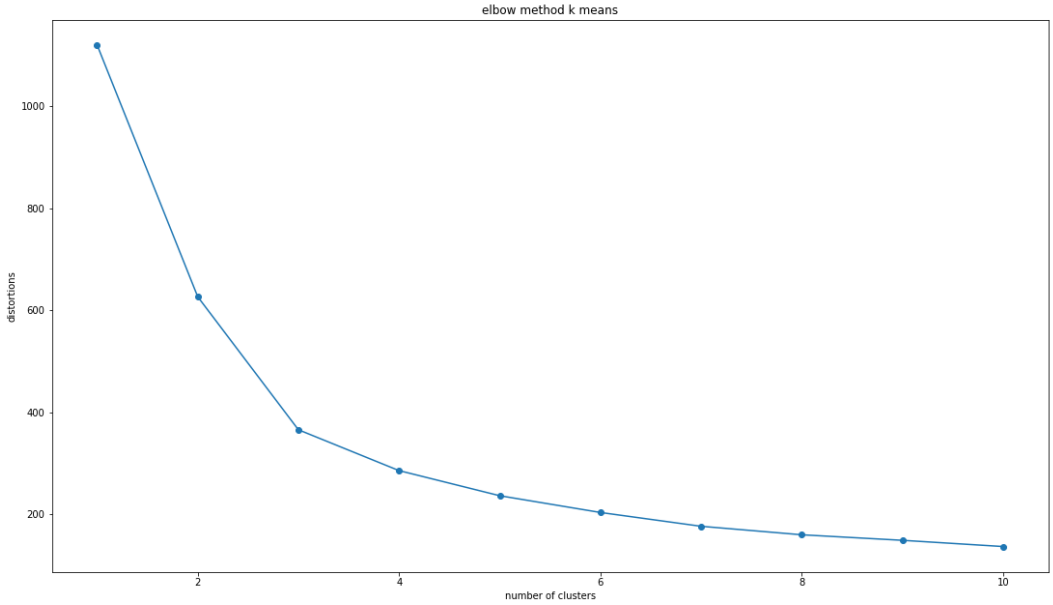


FIGURE 4 – Elbow Method

returns" du marché des jours précédents, un "daily return" est défini de la façon suivante :

$$ret = \frac{Open - Close}{Open}$$

où :

Open :La valeur boursière de l'actif à l'ouverture du marché.
Close :La valeur boursière de l'actif à la fermeture du marché.

Nous nous intéressons plus précisément aux 20 jours précédents (ce qui présuppose l'existence d'un signal dans les 20 jours -1 mois financier- non exploité). Nous nous intéressons aussi à d'autres features liées d'abord au prix comme le VIX (indice de volatilité) ou le volume des transactions effectué le jour d'avant ainsi que le rapport PUT/CALL traduisant bien le sentiment du marché. En effet, si le PUT/CALL est < 0.7 cela veut dire que les investisseurs **s'attendent** à ce que les marchés montent. Inversement, une valeur très grande du PUT/CALL traduit une appréhension à la baisse. Le fine tuning plus poussé du choix des features s'est fait plus de manière expérimentale, on se retrouve donc souvent avec des vecteurs comme dans la figure 5.

Nous avons aussi rajouté plus tard dans nos recherches des features macro-économiques et le prix de quelques commodities afin de raffiner notre segmentation et renforcer donc les modèles qu'on va mettre dessus.

Nous avons donc regroupé les attributs en deux classes : d'une part les attributs liées principalement au marché, comme les returns, le volume des transactions, le VIX ... qu'on a appelé *Price features*, d'autre part les attributs de nature *macro-économique* comme le taux de croissance, le taux

...	Return -9	Return -8	Return -7	Return -6	Return -5	Return -4	Return -3	Return -2	Return -1	VIX
...	0.000146	0.003819	0.007121	-0.003641	0.002027	0.007244	-0.015877	0.010524	0.001774	18.78
...	0.003819	0.007121	-0.003641	0.002027	0.007244	-0.015877	0.010524	0.001774	0.002389	19.85
...	0.007121	-0.003641	0.002027	0.007244	-0.015877	0.010524	0.001774	0.002389	0.002147	20.22
...	-0.003641	0.002027	0.007244	-0.015877	0.010524	0.001774	0.002389	0.002147	0.000017	20.71

FIGURE 5 – Expemple de vecteurs d'attributs

de chômage ... On a cherché des segments avec chaque type d'attribut dans un premier temps, ensuite on a essayé de fusionner les deux.

2.4 INTERPRÉTATION DES RÉSULTATS

Il s'agit dans cette sous partie, de présenter des interprétations macro-économiques des résultats de la segmentation par les méthodes développées ci-dessus.

2.4.1 • NASDAQ COMPOSITE

Avant tout, commençons par une brève description du Nasdaq : le Nasdaq composite est un indice qui regroupe 3000 actions listées sur le Nsadaq stock exchange. En général, ces actions sont celles des entreprises situées aux États-Unis et peuvent parvenir de tous les milieux (industriel, tech...) mais pour des raisons que nous allons pas approfondir, le Nasdaq composite regroupe essentiellement des entreprises américaines de tech comme Google, Apple, Amazon, Ebay, Tesla et autres.

Le Nasdaq a connu une forte croissance consistante entre 1995 et 2000, touchant en Mars 2000 son apogée -ayant augmenté de 400%. Cette forte croissance a été suivie immédiatement d'une forte décroissance, avec une diminution de 78% entre mars 2000 et octobre 2002.

Ce phénomène a été interprété par les économistes comme étant une "bulle financière". En effet, les analystes expliquent la Dotcom Bubble de la façon suivante :

Pendant les années 1990, l'industrie de tech est vue comme la nouvelle ère. Plein de startups se créent et se trouvent financées par des Venture capitalists de façon excessive. Ces startups ont pu créer une illusion de succès à travers les fortes dépenses budgétaires en marketing, ce qui s'est traduit en une forte surévaluation boursière.

Suite à cette surévaluation, quelques nouvelles négatives (par exemple l'entrée en récession du Japon) ont créé un mouvement de panique qui a induit une forte chute du Nasdaq.

Notons aussi que l'effet de la crise des subprimes était moins notable que celui de l'éclatement de la bulle Dotcom. Dans cette partie, nous allons voir si nous pouvons détecter le comportement particulier de la bulle Dotcom à travers un `3-means` clustering sur des features de prix (`VIX`) et de macros comme la croissance des US et les crédit US. Nous choisissons alors, par souci d'homogénéité (stationnarité) temporelle, de découper au préalable l'axe temporel et nous intéresser aux périodes avant et après 2008 séparément, afin de bien réussir à capturer chaque crise toute seule.

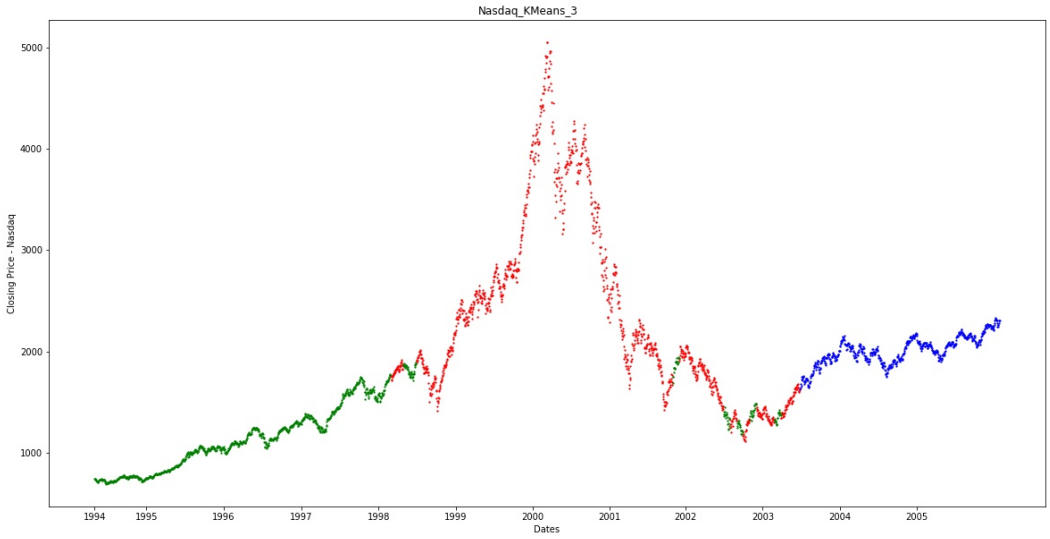


FIGURE 6 – 3 means clustering nasdaq entre 1994 et 2005

Commençons par signaler que notre algorithme a segmenté la série du Nasdaq en 3 parties évidemment distinctes temporellement. Le fait de chercher un clustering spatial sur des données temporelles devrait donner intuitivement des segments bien répartis sur la droite temporelle. Il n'y a autrement dit aucune raison évidente, comme on raisonne sur des dérivés de série temporelles, pour séparer les points si ce n'est effectivement un changement structurel dans les marchés. C'est ce qu'on cherche à mettre en évidence dans notre PSC.

Nous pouvons identifier en première vue la division suivante :

1. La partie en vert correspond à la phase en amont de la bulle
2. La partie en rouge correspond à la bulle et toutes ses phases (spéculation et panique)
3. La partie en bleu correspond à la phase en aval de la bulle.

Cette fois-ci nous segmentons postérieurement à 2008 (figure 7). Nous pouvons observer que la technique utilisée donne des résultats prometteurs aussi puisque nous pouvons identifier (comme pour la bulle DotCom) :

1. La période pré-crise en vert.
2. La période de crise en rouge.
3. La période post-crise en bleu.

Un point intéressant à relever est l'apparition d'une zone rouge après 2012 (donc un comportement de crise), la volatilité aurait donc pu atteindre des niveaux comparables à ceux en 2007. On constate un comportement quasi similaire avec un autre algorithme de segmentation et sur un autre actif dans la section qui suit.

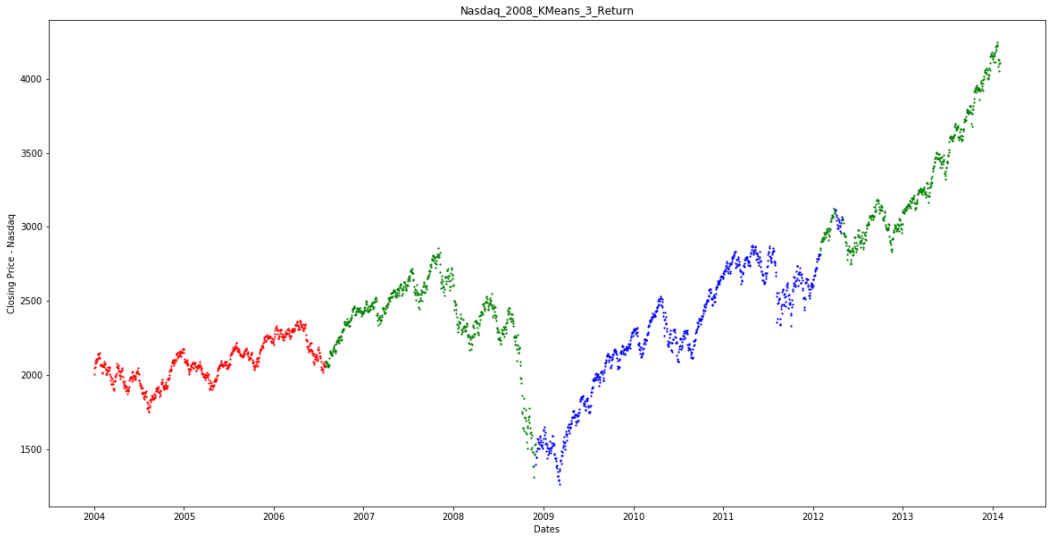


FIGURE 7 – Clustering à partir de 2008

2.4.2 • S&P500

Nous avons segmenté le S&P500 cette fois avec l'algorithme `MeanShift` avec le même jeu de features, et on trouve les résultats ci-dessous :

Le fait que l'algorithme arrive à identifier la crise des "subprimes" et la période qui la suit et les mettre dans un seul cluster en bleu est satisfaisant. Plus surprenant est le fait de classer la période de 2007 à 2008 et la longue période haussière après 2012 dans un même cluster. Comme le `US real estate` fait partie des attributs utilisés pour l'entraînement, il serait donc possible que cette classification soit dû à une structure du marché similaire, qui pourrait caractériser l'existence d'une bulle spéculative risquant de s'éclater à tout instant depuis 2012. Les points dans le quatrième cluster en violet sont des outliers à très fort returns absolus.

L'utilisation de 3 means précédée d'une réduction de la dimension via `PCA`, (toujours sur le S&P 500) avec les features de prix (sans le real estate ou la dette souveraine) donne cette fois-ci une classification tout à fait différente 9.

Afin de mieux comprendre le comportement de chaque segment, nous proposons cette fois de faire une analyse un peu plus quantitative en calculant ensuite le Sharpe ratio défini comme le rapport du return moyen sur la volatilité moyenne sur chaque état :

On constate que le régime 0 est un régime haussier stable le régime 1 est baissier tandis que le régime 2 est haussier. Une telle approche nous permet de comprendre ou d'identifier des régimes cachés sans nous permettre de prédire un changement de régime vu qu'on reste principalement dans un seul échantillon de données.

On constate que les analyses qu'on peut faire qualitativement ou via des critères simples pour essayer de comprendre le sens derrière chaque segment ou état caché sont parfois difficiles à mener jusqu'au bout. Néanmoins, nous avons vu à travers cette partie que les algorithmes non supervisés

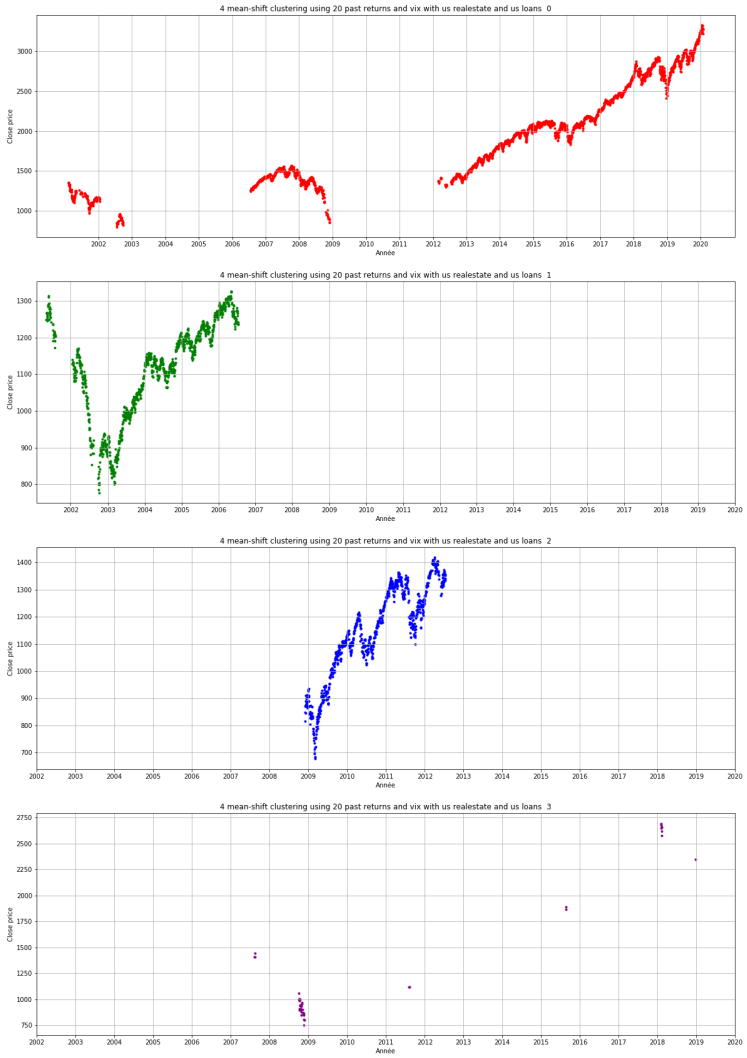


FIGURE 8 – MeanShift sur la période

donnent des résultats qui semblent suivre une logique économique. Avant d'évaluer l'utilité de ces segments, nous nous intéresserons d'abord à une méthode plus classique dans l'analyse des séries temporelles, spécialement en finance qu'est la méthode des chaînes de Markov cachées.

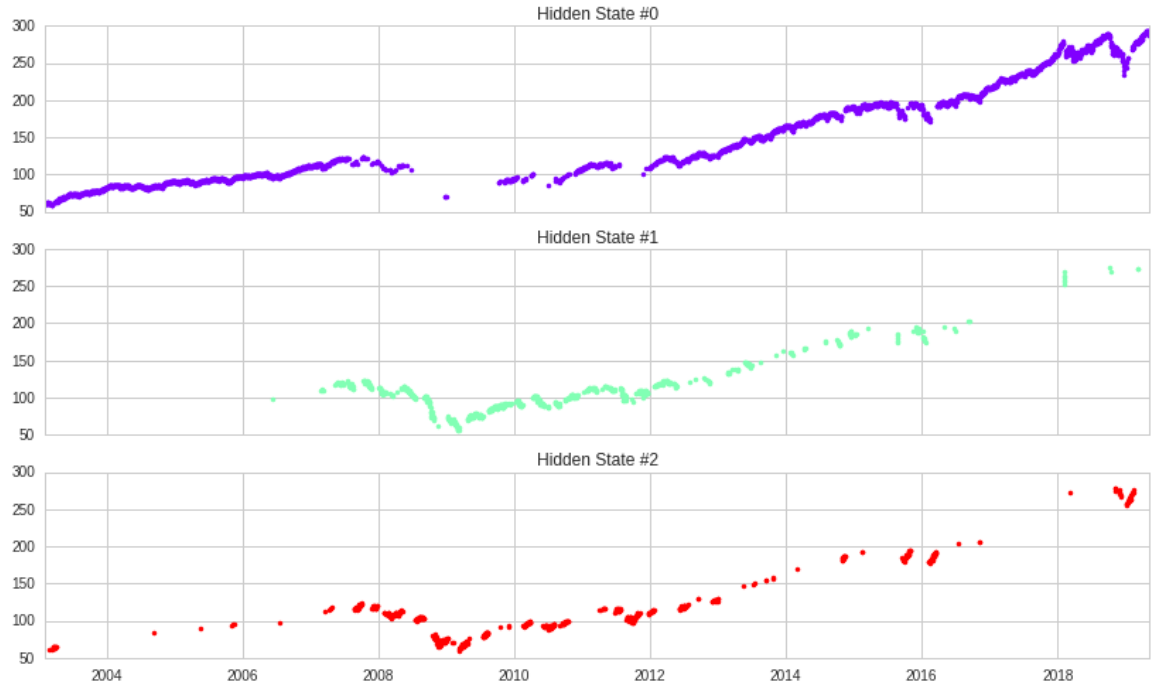


FIGURE 9 – Clustering qui correspond au segments trouvés précédemment

3
MÉTHODES HMM DANS LA CLASSIFICATION

Les chaînes de Markov sont souvent utilisées pour reconnaître des données séquentielles. Dans une chaîne de Markov, une réalisation dépend des réalisations précédentes. Considérons un système avec un ensemble d'états distincts $S = \{1, 2, ..., N\}$. A instant t, le système va passer à un nouvel état dans S. Ces transitions sont régies par une matrice de transition P, qui exprime la dépendance des états précédents.

On peut modéliser le mouvement d'un indice financier par une chaîne de Markov. Par exemple, considérons un ensemble S à cinq états :

$$S = \{\text{grande hausse , petite hausse, pas de changement, grande baisse, petite baisse}\}$$

En prenant une matrice de transition $P = (a_{i,j})_{1 \leq i \leq 5, 1 \leq j \leq 5}$, on peut répondre à un spectre de questions intéressantes sur le comportement de l'indice modélisé. En particulier on peut répondre à la question suivante : connaissant les états des 5 derniers jours, par exemple $X = 1, 3, 2, 2, 4$, quel est l'état le plus probable à l'étape suivante ? Il suffit en fait d'évaluer les probabilités d'apparition de chaque état de S au 6ème jour sachant X et prendre l'état qui maximise cette probabilité.

3.1 MODÈLES DE MARKOV CACHÉS

Les modèles de Markov cachés sont une généralisation des chaînes de Markov qui utilise deux suites de variables aléatoires, la première observable et la deuxième cachée [8].

Chaque état de la variable cachée est associé à une distribution de probabilité. Au temps t une observation o_t est générée par une fonction probabiliste $b_j(o_t)$, qui est associée à la distribution j (la distribution de l'état j) et qui vérifie :

$$b_j(o_t) = \mathbf{P}(o_t|X_t = j)$$

En d'autres termes, les états de la chaîne de Markov considérée sont des distributions entre lesquelles on transite. À l'instant t on obtient une réalisation qui provient de la distribution sous-jacente au même instant.

3.2 POURQUOI HMM ?

Il y a intrinsèquement une bonne correspondance entre analyse de série temporelle et HMM, notamment car une série temporelle peut être modélisée par un processus stochastique et le modèle d'HMM a une grande capacité à s'adapter aux données.

En effet, l'algorithme Maximisation-Espérance (EM) donne accès à un ensemble de méthodes d'entraînement efficaces qui permettent de calibrer un HMM sur les données, et ainsi trouver les meilleurs paramètres correspondant aux données observées.

Plus généralement, les modèles HMM sont un framework d'apprentissage non supervisé. On utilisera ce framework pour deux buts distincts : segmentation, puis prédiction des variations du marché.

3.3 FORME GÉNÉRALE D'UN HMM

Un HMM se compose d'un quintuplé (S, K, Π, A, B).

1. $S = \{1, ..., N\}$ est l'ensemble des états. L'état à l'instant t est noté s_t.
2. $K = \{k_1, ..., k_M\}$ est l'alphabet de sortie. Dans le cas d'observations de données discrètes K correspond à toutes les valeurs observables, dans le cas où il s'agit d'une série temporelle à valeurs continues (température par exemple) K va plutôt désigner une discrétisation de l'intervalle de variation de la série en question.
3. Distribution initiale $\Pi = \{\pi_i\}_{i \in S}$. avec

$$\pi_i = \mathbf{P}(s_1 = i)$$

4. Matrice de transition $A = \{a_{i,j}\}_{i,j \in S}$.

$$a_{i,j} = \mathbf{P}(s_{t+1} = j | s_t = i), 1 \leq i, j \leq N$$

5. Distribution probabiliste des observations $B = \{b_j(o_t)\}_{j \in N}$:

$$b_j(o_t) = \mathbf{P}(\text{observation de la série} = o_t | s_t = j)$$

On va de plus s'intéresser à une modélisation continue des observations de la série temporelle. Plus précisément on modélise les $b_j(o_t)$ par des mixtures de gaussiennes comme suit :

$$b_j(o_t) = \sum_{k=1}^{M} w_{j,k} \beta_{j,k}(o_t), pour j = 1, ..., N$$

M étant le nombre de mixtures et les $w_{j,k}$ les poids associés tel que :

$$\sum_{k=1}^{M} w_{j,k} = 1, j = 1, ..., N; w_{j,k} \geq 0, pour j = 1, ..., N et k = 1, ..., M$$

$$\beta_{j,k}(o_t) = \mathcal{N}(o_t, \mu_{j,k}, \Sigma_{j,k})$$

Pour des distributions gaussiennes de dimension 1 :

$$\beta_{i,j}(o_t) = \frac{1}{\sqrt{2\pi}|\Sigma_{j,k}|^{1/2}} \exp(-\frac{(o_t - \mu_{j,k})^2}{2{\Sigma_{j,k}}^2})$$

Pour des distributions multivariées de gaussiennes de dimension D :

$$\beta_{i,j}(o_t) = \frac{1}{(2\pi)^{D/2}|\Sigma_{j,k}|^{1/2}} \exp(-\frac{1}{2}(o_t - \mu j,k)^T {\Sigma_{j,k}}^{-1}(o_t - \mu_{i,j}))$$

3.4 PROBLÉMATIQUES FONDAMENTALES DES MODÈLES HMM

Il existe trois problèmes fondamentaux auxquels on s'intéresse lors de l'implémentation et de la calibration des HMM :

1. Etant donné un modèle avec des paramètres $\mu = (A, B, \Pi)$, et une suite d'observations $O = (o_1, ..., o_T)$, comment peut-on calculer efficacement la probabilité d'observer cette suite avec notre modèle ? i.e $\mathbf{P}(O|\mu)$.
2. Etant donné un modèle μ, et des observations O, quelle est la suite d'états $\{state_1, ..., state_N\}$ qui "explique" le mieux ces observations ?
3. Etant donné une suite d'observations O, et un espace de modèles possibles, comment ajuster les paramètres pour trouver le modèle qui maximise $\mathbf{P}(O|\mu)$.

Le premier problème permet de déterminer quel modèle parmi ceux entraînés est le plus pertinent pour la suite donné des observations. Dans le cas d'une série temporelle représentant un marché financier, cela permet de trouver le meilleur modèle du mouvement de ce marché étant donné les observations au passé.

Le second problème consiste à retrouver les états cachés du modèle, tandis que le troisième problème est le plus intéressant car c'est là que l'entraînement du modèle entre en jeu et que l'optimisation des paramètres est nécessaire.

3.4.1 • PROBLÈME 1 : CALCULER LES PROBABILITÉS DES OBSERVATIONS

Considérons une suite d'observations $O = (o_1, ..., o_T)$ et un modèle HMM donné par $\mu = (A, B, \Pi)$, on veut calculer la probabilité $\mathbf{P}(O|\mu)$. Puisque les observations o_t sont indépendantes entre elles pour une suite d'états sous-jacents fixée $S = (s_1, ..., s_T)$:

$$\mathbf{P}(O|\mu) = \sum_{S \text{ possibles}} \mathbf{P}(O|S, \mu)\mathbf{P}(S|\mu)$$

On peut facilement montrer que :

$$\mathbf{P}(O|\mu) = \sum_{S \text{ possibles}} \pi_{s_1} b_{s_1}(o_1) a_{s_1,s_2} ... b_{s_T}(o_T) a_{s_{T-1},s_T}$$

Le calcul ci-dessus est certes assez direct, or il pose des problèmes d'efficacité étant donné que la complexité correspondante est exponentielle en T dès que $N \geq 2$.

Pour remédier à ce problème on utilise l'algorithme forward-backward qui permet d'évaluer ces expressions en temps linéaire.

3.4.2 • PROBLÈME 2 : TROUVER LA SUITE D'ÉTATS OPTIMALE

La difficulté principale ici est qu'il est possible d'avoir plusieurs critères d'optimalité à vérifier. Un de ces critères consiste à prendre les états qui sont individuellement plus probables à chaque instant t. Ainsi, pour $1 \leq t \leq T$ on cherche à calculer :

$$\gamma_i(t) = \mathbf{P}(s_t = i|O, \mu) = \frac{\mathbf{P}(s_t = i, O|\mu)}{\mathbf{P}(O|\mu)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$$

L'ensemble d'état S' optimal est alors donné par :

$$S' = (argmax_{1 \leq i \leq N} \gamma_i(t), 1 \leq t \leq N)$$

3.5 ALGORITHME DE VITERBI

L'algorithme de Viterbi permet de calculer l'ensemble des états S' décrit ci-dessus. Commençant par remarquer qu'il suffit en fait de calculer :

$$argmax_{S'} \mathbf{P}(S', O|\mu)$$

Introduisons alors :

$$\delta_j(t) = \max_{s_1,...,s_{t-1}} \mathbf{P}(s_1, o_1, ..., s_{t-1}, o_{t_1}, s_t = j|\mu)$$

Ces variables stockent la probabilité d'observer $o_1,...,o_t$ en empruntant le chemin le plus probable qui s'arrête à l'état j au temps t sachant le modèle μ.

Algorithm 2: Algorithme de Viterbi

initialization;

$$\delta_i(1) = \pi_i b_i(o_1), 1 \leq i \leq N;$$

$$\psi_i(1) = 0, 1 \leq i \leq N;$$

$t = 0$;

while $t \leq T$ **do**

$$\delta_j(t) = b_j(o_t) \max_{1 \leq i \leq N} \delta_i(t-1) a_{i,j};$$

$$\psi_j(t) = argmax_{1 \leq i \leq N}[\delta_i(t-1)a_{i,j}];$$

$t = t + 1$;

end

$P^* = \max_{1 \leq i \leq N}[\delta_i(T)]$; ${s_T}^* = argmax_{1 \leq i \leq N}[\delta_i(T)]$;

La suite d'état la plus probable peut être calculée itérativement comme suit : ${s_t}^* = \psi_{t+1}({s_{t+1}}^*)$.

3.5.1 • PROBLÈME 3 : ESTIMATION DES PARAMÈTRES

On s'intéresse ici à l'estimation des paramètres $\mu = (A, B, \Pi)$ qui décrivent de la meilleure façon les observations de la série temporelle. Le problème peut être reformuler comme suit :

$$argmax_\mu \mathbf{P}(O|\mu)$$

Le moyen le plus classique d'y procéder est l'algorithme de Baum-Welch qui converge vers l'optimum en temps linéaire. Pour retrouver l'optimum $\mu = (A, B, \Pi)$ itérativement, on définit d'abord les variables $p_t(i,j), 1 \leq i, j \leq N$ comme suit :

$$p_t(i,j) = \mathbf{P}(s_t = i, s_{t+1} = j|O, \mu) = \frac{\mathbf{P}(s_t = i, s_{t+1} = j, O|\mu)}{\mathbf{P}(O|\mu)}$$

On a ainsi :

$$p_t(i,j) = \frac{\alpha_i(t) a_{i,j} b_j(o_t) \beta_j(t+1)}{\sum_{m=1}^N \alpha_m(t)\beta_m(t)} = \frac{\alpha_i(t) a_{i,j} b_j(o_t) \beta_j(t+1)}{\sum_{m=1}^N \sum_{n=1}^N \alpha_m(t) a_{m,n} b_n(o_t) \beta_n(t+1)}$$

Formellement, $p_t(i,j)$ est la probabilité d'être à l'état i au temps t et à l'état j au temps $t+1$ sachant μ et les observations O. On peut alors en déduire que :

$$\gamma_i(t) = \mathbf{P}(s_t = i|O, \mu) = \sum_{j=1}^N \mathbf{P}(s_t = i, s_{t+1} = j|O, \mu) = \sum_{j=1}^N p_t(i,j)$$

L'équation ci-dessus est vraie étant donné que $\gamma_i(t)$ est l'espérance du nombre de transitions depuis l'état i, et $p_t(i,j)$ l'espérance du nombre de transitions depuis l'état i à l'état j.

Ayant posé ces définitions, on commence par un modèle initialisé μ , puis on estime les espérances de chaque paramètre grâce aux données d'entraînement O. On peut alors changer le modèle pour maximiser la vraisemblance des chemins empruntés. En répétant cette approche, on espère converger vers les valeurs optimales des paramètres du modèle.

$$\pi'_i = \text{probabilité d'être à l'état i au temps t=1} = \gamma_i(t)$$

$$a'_{i,j} = \frac{\sum_{t=1}^{T} p_t(i,j)}{\sum_{t=1}^{T} \gamma_i(t)}$$

$$b'_{i,j,k} = \frac{\sum_{1\leq t\leq T, o_t=k} p_t(i,j)}{\sum_{t=1}^{T} p_t(i,j)} = \frac{\text{espérance du nombre de transition de i à j avec k observé}}{\text{espérance du nombre de transitions de i à j}}$$

3.6 APPLICATION À LA SEGMENTATION

L'une des applications des HMM est la segmentation des séries temporelles en plusieurs catégories caractérisées par des comportements similaires. La bibliothèque Python `hmmlearn` permet d'implémenter très simplement des modèles HMM et les entraîner sur des observations données, ainsi le modèle entraîné final permet de classifier les régimes du marché sachant le nombre d'états imposé au départ [9].

Plus particulièrement, on a entraîné des modèles HMM à deux états sur deux indices boursiers : le premier étant S&P500 et le deuxième est SHCOMP l'indice boursier le plus représentatif du marché financier Chinois.

D'autre part on prend le soin de considérer des données sur une large durée afin que la segmentation ait un sens d'un point de vue macro-économique, pour identifier plusieurs périodes historiques.

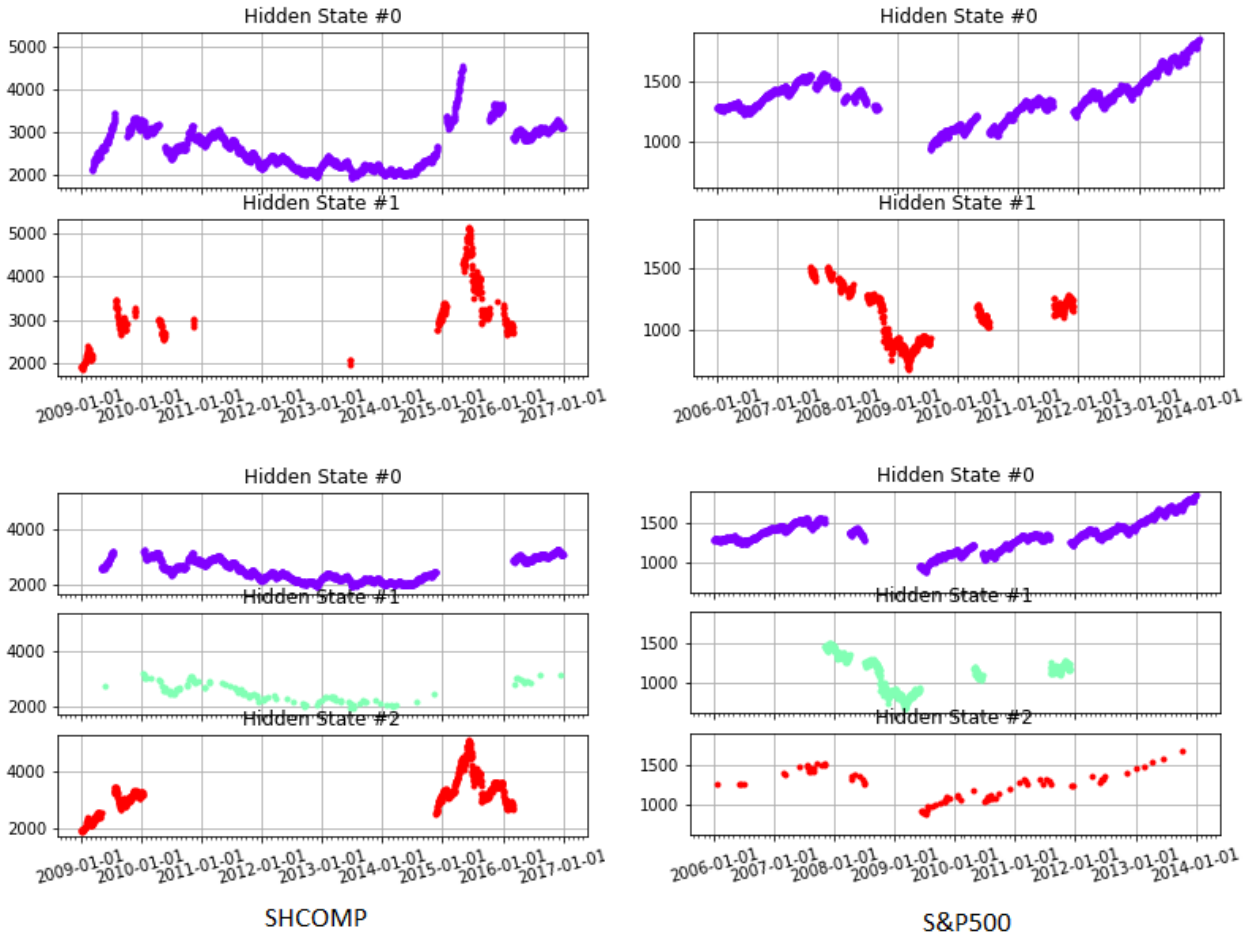


FIGURE 10 – Segmentation avec 2 et 3 mixtures SHCOMP Chine

Il est assez remarquable de voir 10 que la segmentation générée par chacun des deux modèles HMM entraînés (l'un sur S& P500, l'autre sur SHCOMP) a classé les périodes de stress financier et de grande volatilité dans un état (représenté par les points rouges) et les périodes plus stables dans un second état (points bleus). Le fait de passer à 3 états, rend la segmentation plus fine, et identifie mieux le segment stable dans le cas chinois.

4

EVALUATION DES SEGMENTS PAR DES ALGORITHMES SUPERVISÉS

Afin de mesurer la qualité des clusters, nous avons décidé d'examiner leur impact sur la performance de différents modèles que nous avons choisi. Pour cela, on entraîne nos modèles sans clusters, puis on les entraîne sur chaque cluster séparément en choisissant les meilleurs hyperparamètres adaptés à chaque cluster. Une bonne segmentation devrait intuitivement améliorer la performance.

On considère les clusters fournis par kmeans en prenant en premier comme attributs les données liées aux prix, ensuite les données macro-économiques et finalement en prenant les deux.

On a rajouté une quatrième façon de segmenter qui est basée sur une forme simplifiée des données liées aux prix, où on ne considère plus les valeurs exactes des attributs mais plutôt on prend des variables binaires comme indicatrices des quartiles. Par exemple, pour le volume on définit quatre variables binaires, si la valeur du volume appartient au premier quartile la première variable prendra la valeur 1 alors que les trois autres prendront la valeur 0. On appelle les résultats de cette segmentation, Bins clustering.

On a utilisé des modèles de classification supervisée qui ont pour but de prédire le signe du return journalier. On prend la classe (1) comme la classe des returns positifs et la classe (-1) comme la classe des returns négatifs.

4.1 PRÉLIMINAIRE

4.1.1 • SUR LES PÉRIODES D'ENTRAÎNEMENT, VALIDATION ET TEST :

On a choisi comme période de validation la période entre 01/08/2014 et 31/01/2017. On a laissé une période de 6 mois entre la fin de la période d'entraînement et le début de la période de validation pour éviter des prédictions translatées . On a pris comme période d'entraînement la période entre 04/01/2001 et 31/01/2014.

Le fait de baser nos choix d'hyperparamètres sur la période de validation peut engendrer un overfit des modèles sur cette période. C'est pour cette raison que nous avons gardé les données des deux dernières années en tant qu'Out-of-sample pour estimer l'erreur indépendamment de la période de validation.

4.1.2 • EVALUATION DE LA PERFORMANCE

Pour évaluer les performances des modèles, on ne peut pas simplement se contenter de calculer le pourcentage de bonne prédiction sur l'ensemble de validation [4]. Évaluer un modèle de cette façon ne fournit aucune information sur son comportement pour les différents résultats de classification. Il est important de faire cette distinction surtout dans les cas où on a une dissymétrie entre les résultats . Dans notre cas, une mauvaise prédiction sur un retour positif est plus risquée qu'une mauvaise prédiction sur un retour négatif. Pour raffiner notre analyse, on s'est basé principalement sur la matrice de confusion. Le coefficient $c_{i,j}$ de cette matrice indique le nombre d'observations qui appartiennent à la classe i et qui ont été classifié comme appartenant à la classe j. Pour les classifications binaires, on peut alors facilement distinguer entre les type d'erreurs (I et II) pour chaque résultat possible. On s'intéresse particulièrement à deux métriques à savoir la précision (qu'on notera p) et le recall (noté r) pour chaque classe qui sont définis de la façon suivante :

		Classe réelle	
		-	+
Classe prédite	-	True Negatives *(vrais négatifs)*	False Negatives *(faux négatifs)*
	+	False Positives *(faux positifs)*	True Positives *(vrais positifs)*

$$p_i = \frac{TP_i}{TP_i + FP_i} = \frac{c_{i,i}}{c_{i,1} + c_{i,2}}$$
$$r_i = \frac{TP_i}{TP_i + FN_i} = \frac{c_{i,i}}{c_{1,i} + c_{2,i}}$$

Une grande valeur de la précision pour la classe k veut dire que le modèle donne très peu de faux positives pour cette classe, et donc qu'on peut faire confiance au modèle s'il prédit la classe k parce qu'elle a une grande chance d'être la valeur réelle. Une grande valeur du recall de la classe k veut dire que le modèle est capable d'identifier une grosse partie des observations qui appartiennent à la classe k.

En ce qui concerne l'implémentation, la matrice de confusion ainsi que les deux métriques qu'on vient de définir sont fournies par la bibliothèque `scikit-learn` de `Python` (`confusion_matrix` et `classification_report`).

4.1.3 • CHOIX DES FEATURES

Pour entraîner les modèles, nous avons choisi les données suivantes :

— Le volume des transactions
— Le ratio des options Put/Call
— L'index du VIX
— Le Momentum de l'indice, qui représente l'élan de celui-ci. Il est calculé à partir de la moyenne sur les 5 derniers jours des returns

En plus de ces features, on a rajouté d'autres données qui ne sont pas directement liées au prix mais qui sont de nature plus macro-économiques, comme :

— Le taux de croissance
— Le taux de chômage
— Deux autres indices concernant les prêts et l'immobilier.

Dû à une faute de manipulation des données, on a shifté les données du Volume, et on s'est aperçu que la connaissance du Volume du jour k en début de journée a amélioré significativement la performance du modèle. Donc on a décidé de rajouter le Volume journalier du marché chinois que l'on a shifté. On peut considérer cette donnée comme une estimation du volume du marché américain qui peut être connu avant l'ouverture des marchés aux Etats-unis.

4.2 MODÈLES LINÉAIRES

4.2.1 • RÉGRESSION LOGISTIQUE

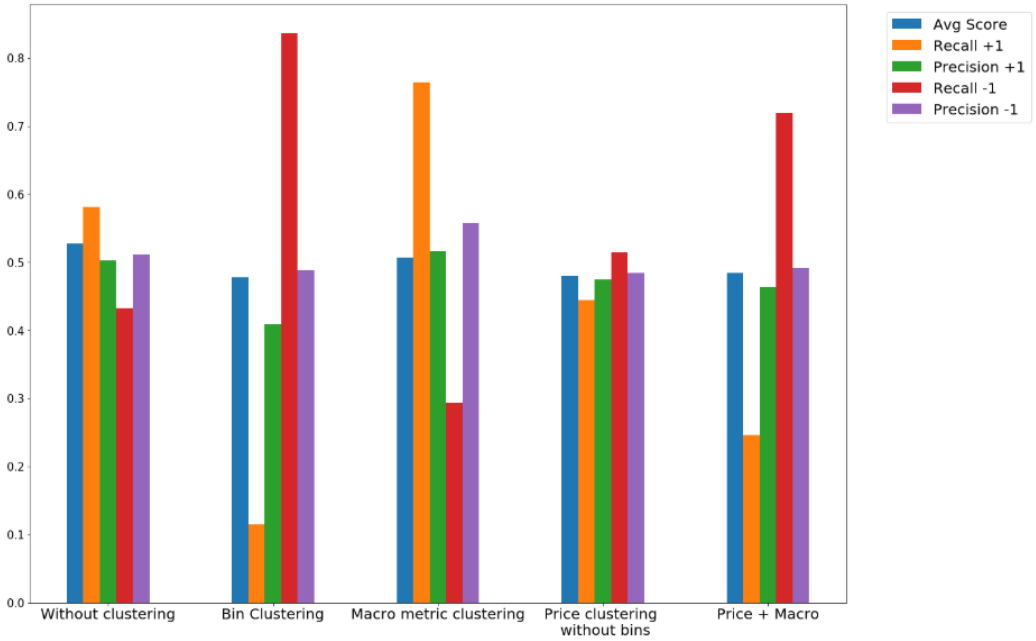
Dans le cas de la régression logistique, on veut déterminer p, la probabilité que l'observation appartient à la classe 1, en fonction des observations. On essaie de trouver une relation affine entre les attributs et $f(p) = log(\frac{p}{1-p})$ pendant la phase d'apprentissage, ie

$$f(p) = w^T x + b$$

où w et b sont déterminés grâce à une regression linéaire.
Avec de simples manipulations algébriques, on aboutit à la relation

$$p = \frac{1}{1 + e^{-(w^T x + b)}}$$

Afin de classifier les données, on calcule alors p à partir de la relation obtenue précédemment. Si $p > \frac{1}{2}$ (ie le point a plus de chances d'après notre modèle d'être dans la classe (1) que (-1)), on affecte notre point à la classe (1), sinon il est dans la classe (-1). Ce modèle nous permet de régler la sensibilité aux outliers (rares pics) grâce à un hyperparamètre souvent noté c.
Pour un nombre de clusters $k = 3$ avec *kmeans*, on trouve les résultats suivants :



On s'aperçoit d'emblée que certaines segmentations ont modifié la performance du modèle. La segmentation qui utilise les métriques macroéconomiques, a donné une valeur élevée pour le recall de la classe (1) ainsi qu'une bonne précision pour la classe (-1), on peut dire que notre modèle est capable de localiser les éléments de la classe (1), tout en étant capable de signaler les éléments de la classe (-1). Le quatrième modèle n'est pas très intéressant parce que les résultats fournis sont plus proches de ceux d'une variable de Bernoulli de paramètre 0.5. Cela veut dire que le clustering semble interférer avec la méthode de classification, de sorte que le processus de décision est proche de l'aléatoire. En effet, les returns sont distribués de façon symétrique, et en considérant une variable $\hat{y}$ de Bernoulli on a :

$$P(\hat{y} = k | y = k) = \frac{1}{N_k} \sum_{i, y_i = k} 1_{\hat{y}_i = k} = recall_k = 0.5$$

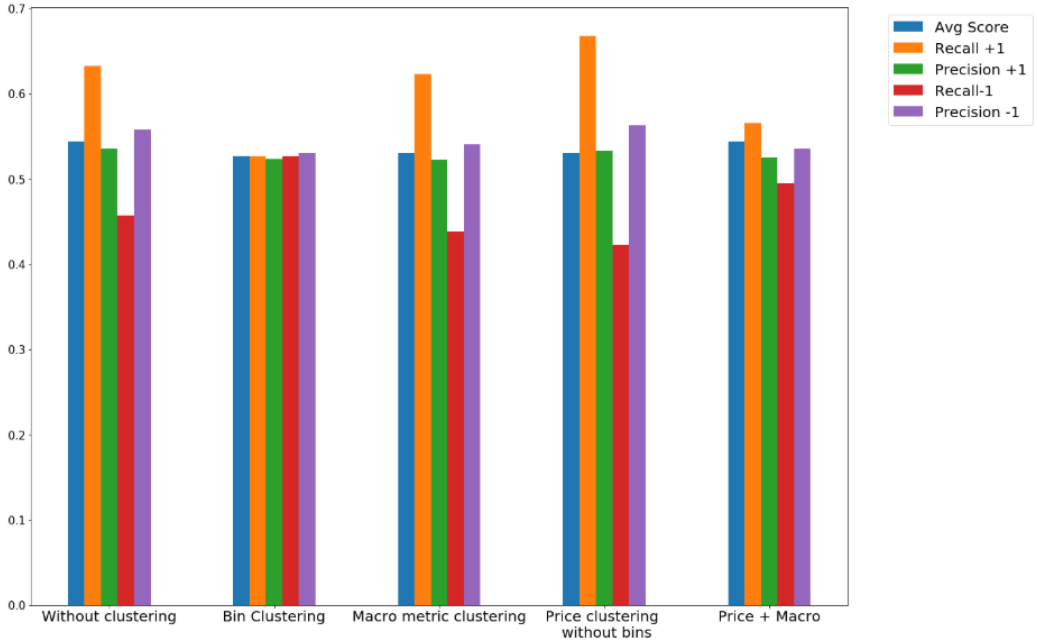
$$P(y = k | \hat{y} = k) = \frac{1}{N_k} \sum_{i, \hat{y}_i = k} 1_{y_i = k} = \texttt{precision}_k = 0.5$$

avec $\hat{y}$ est la prédiction et y est la classe réelle.

4.2.2 • LINEAR SVM

La régression logistique cherche un hyperplan qui sépare les deux classes en utilisant une approche statistique, le Support Vector Machine (linéaire) utilise une approche géométrique. L'algo-

rithme dispose les données en un nuage de points et cherchent un hyperplan qui sépare les données en deux classes. Sous l'hypothèse de séparabilité des données -ie on peut mettre tous les éléments d'une classe d'un côté de l'hyperplan- on a donc une infinité d'hyperplans qui peuvent satisfaire cette condition, toutefois cet algorithme nous fournit un hyperplan optimal, dans le sens où il est à une distance égale des deux classes. On utilise par la suite cet hyperplan pour prédire. Plus formellement, si l'hyperplan est décrit à l'aide de son vecteur normal de la façon suivante : $H = \{x \in \mathbb{R}^n | w^T.x + b = 0\}$. Les prédictions sont données directement par la fonction h : $h(x) = signe(w^T.x + b)$ Encore une fois, on peut déterminer la sensibilité du modèles aux outliers (rares pics) à l'aide de l'hyperparamètre C. Pour un nombre de clusters k=3 avec *kmeans*, on trouve les résultats suivants :

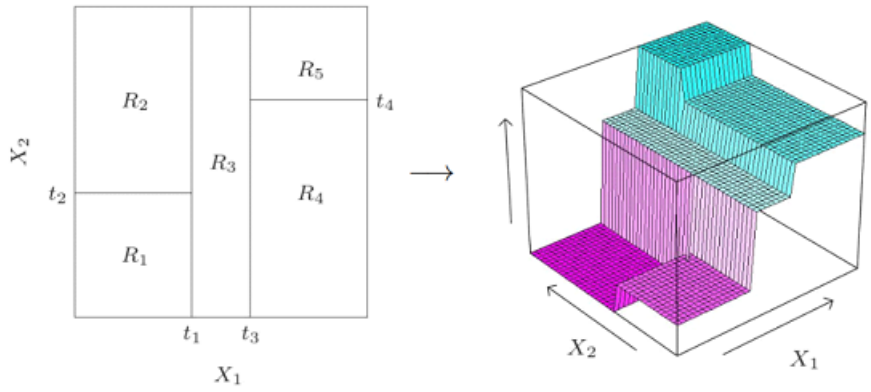


On remarque que les segmentations ont encore une fois amélioré la performance du modèle. Les modèles avec une segmentation ont un grand recall pour la classe (1) et une bonne valeur pour la précision (-1) (à l'exception de la segmentation avec des bins). Dans ce cas on peut dire que le modèle est capable d'identifier les returns positifs tout en ayant un bon signal sur les returns négatifs. La première segmentation donne des valeurs proches de 0.5 ce qui suggère qu'elle a un comportement similaire à une variable de Bernoulli (voir commentaire précédent). Après avoir combiné les données du prix avec les données macro pour segmenter, on remarque que la performance du modèle a baissé, en particulier, on a une baisse dans les précisions des deux classes.

4.3 MODÈLES BASÉS SUR LES ARBRES DE DÉCISIONS

Un arbre de décision est un organigramme qui nous aide à prendre des décisions [11]. Pour ce type de modèles, on construit des organigrammes et on les exploite pour prendre les meilleurs décisions. Ces modèles partitionnent l'espace en plusieurs plages de valeurs où chacune contient des observations homogènes entre elles et associent à chacune une valeur ou une classe (voir figure ci-dessous). La différence majeure entre ces modèles et les modèles linéaires est que les premiers essaient de mo-

déliser la relation entre les attributs et le résultat à l'aide d'une fonction en escalier tandis que les derniers utilisent exclusivement des fonctions affines. [3] Voici une illustration d'arbre de décisions sur un plan : il a été séparé en 5 régions homogènes entre elles, et à chaque région on associe une valeur. On obtient une fonction en escalier.



Un unique arbre de décision a empiriquement une très mauvaise performance [3]. Afin de dépasser cette difficulté, plusieurs algorithmes essaient de construire plusieurs arbres. Ces arbres sont construits de façon indépendante ou successivement où chaque arbre est créé à partir des résultats du dernier. Etant donnée la complexité des données financières, et le caractère assez réducteur des modèles linéaires (basés uniquement sur une régression linéaire), il est envisageable que les arbres de décisions, qui peuvent modéliser des situations plus complexes, aient de meilleures performances.

4.3.1 • RANDOM FOREST

C'est l'exemple le plus connu où les arbres de décisions sont construits de façon indépendante. On fixe d'abord le nombre d'arbres qu'on veut construire au total. Le but de ce modèle est d'avoir un grand nombre d'arbres relativement non corrélés. Si la majorité des arbres ont pris une décision, vu la grande décorrélation entre eux, cette décision aura plus de chances d'être la bonne. Pour chaque arbre, on crée une nouvelle base de données à partir de la base de données initiale en tirant avec remise les observations de cette dernière. On choisit aussi aléatoirement un sous ensemble d'attributs pour chaque arbre. Ce processus est connu sous le nom de booststap aggregation ou encore bagging qui sert à assurer la décorrélation. On choisit une condition sur un attribut et on obtient deux groupes : les observations pour lesquels la condition est vérifiée et les autres pour lesquelles elle ne l'est pas. On peut ensuite calculer l'hétérogénéité dans les deux groupes obtenus en utilisant une des deux mesures d'impureté défini en dessous. On choisit la condition pour laquelle l'impureté est minimal et on recommence le même processus pour les deux groupes obtenus jusqu'à ce qu'on n'ait plus d'observations ou on atteint une condition d'arrêt.

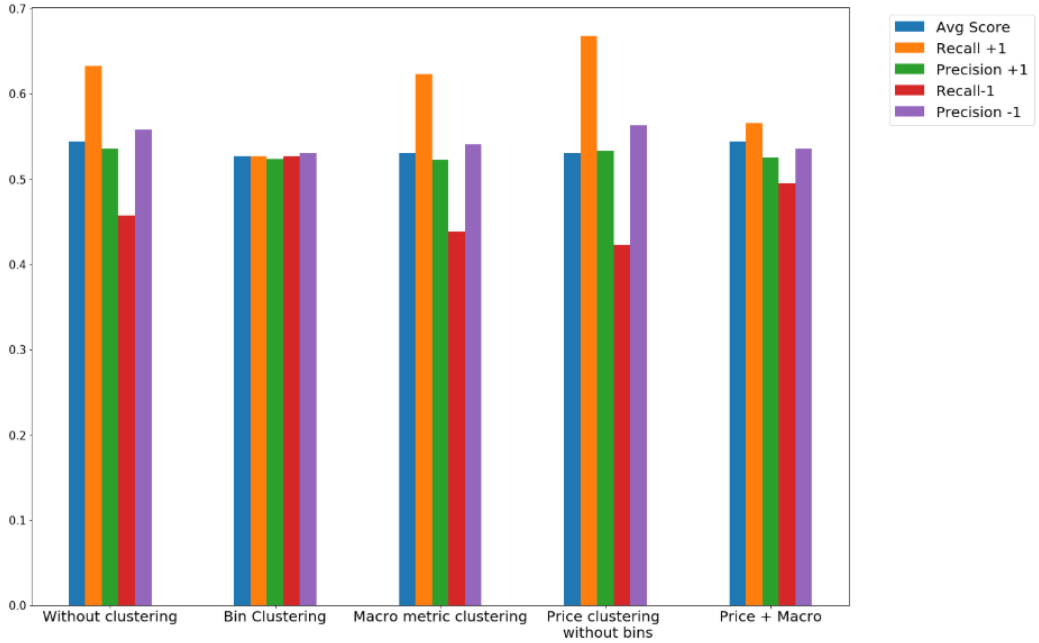
L'hétorgéniéité peut se calculer aussi avec l'entropie :

$$S = P(\text{True})(-\sum_k p_{k|True} log(p_{k|True})) + P(\text{False})(-\sum_k p_{k|False} log(p_{k|False}))$$

Une autre méthode est avec le coefficient de Gini :

$$Gini = P(\text{True})(1 - \sum_k p^2_{k|True}) + P(\text{False})(1 - \sum_k p^2_{k|False})$$

Pour un nombre de clusters k=4 avec *kmeans*, on trouve les résultats suivants :



Le résultat qu'on trouve pour la segmentation avec les bins a une très grande variabilité qui dépend des segments fournies par l'algorithme qui peuvent largement varier entre deux simulations. On remarque que deux (macro et prix) des trois segmentations ont amélioré la précision du modèle pour la classe (-1) ainsi que le recall de la classe (1). On peut dire que la segmentation a amélioré la capacité du modèle à détecter les returns postifs tout en améliorant le signal des returns négatifs. La segmentation avec bins a baissé la performance du modèle, avec des signaux qui ressemblent plus aux signaux d'un Bernoulli de paramètre $\frac{1}{2}$ On s'aperçoit aussi que lorsqu'on combine entre les données du prix ainsi que les données macro-économiques, on a une légère baisse de la performance.

4.3.2 • XGBOOST

XGBoost est l'une des méthodes de Gradient Boosting les plus connues [13]. Ces algorithmes construisent des arbres itérativement. On commence avec un arbre décision, et à chaque itération, on construit un arbre basé sur les erreurs commises par l'arbre précédent afin de compenser sa mauvaise performance.
Idéalement, à chaque itération on doit avoir un meilleur classifieur, jusqu'à ce qu'on converge vers le meilleur classifieur. Cette démarche ressemble beaucoup à la méthode du descente du gradient en optimisation, d'où la nomination Gradient Boosting. Ces algorithmes ont été créé pour gérer des données complexes et dont il est difficile d'extraire des tendances, ce qui convient bien à notre situation. XGBoost (e**X**treme **G**radient **B**oost) suit le même processus. Chaque arbre à chaque étape est créé récursivement :

— **Etape 0 :** On crée une racine qui contient toutes les entrées.
— **Etape n+1 :** Etant donné un noeud qui contient un ensemble d'entrées, on évalue le gain de leur division en deux groupes grâce au score de similarité défini ci-dessous. Comme précédemment, on sépare les données en 2 à partir d'une condition sur un attribut du type $x > a$, avec a fixé.
On choisit alors la séparation qui correspond au gain maximal, et on applique l'algorithme sur chacun des noeuds obtenus.
— **Condition d'arrêt :** On arrête l'algortihme si une des conditions est verifiée :
  — Toutes les feuilles ne contiennent qu'une seule observation chacune
  — Toutes les séparations donnent des gains négatifs
  — Une autre condition d'arrêt supplémentaire est atteinte.

Comme pour le Random Forest, on choisit un sous ensemble de données à classifier aléatoirement, mais contrairement à l'algorithme précédent, on ne tire pas les données de manière uniforme, mais en privilégiant les données mal classifiées par l'arbre construit à l'étape d'avant. En effet, une fois que la construction de l'arbre est terminée, on associe à tous les éléments un poids qui sera d'autant plus grand que l'observation a été mal classifiés par l'arbre, et la probabilité qu'une observation soit tirée est proportionnelle à son poids. Ainsi, les observations mal classifies auront plus de chances d'apparaître plusieurs fois (car le tirage est sans remise, comme précisé précédemment) dans le nouvel échantillon, et l'arbre suivant va mieux les classifier.

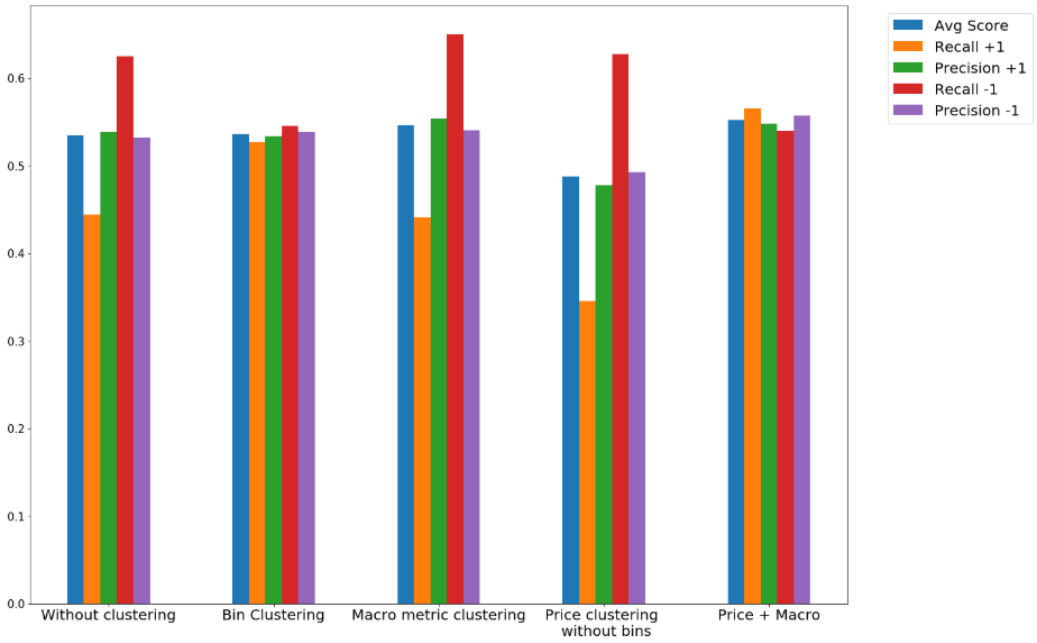
$$\text{Similarity Score} = \frac{(\text{Sum of residuals})^2}{\sum_i \text{Previous Proba}_i(1 - \text{Previous Proba}_i) + \lambda}$$

$$\text{Gain} = \text{Similarity Score}_{Group1} + \text{Similarity Score}_{Group2} - \text{Similarity Score}_{All} - \gamma$$

A la fin pour classifier une observation on combine entre les différentes prédictions des arbres obtenus.

$$Probability(X_i) = sigmoid(\varepsilon \sum f_k(x_i))$$

où chaque f_k représente le résultat d'un arbre et ε est un hyperparamètre du modèle (`learning_rate`). Pour un nombre de clusters $k = 4$ avec *kmeans*, on trouve les résultats suivants :

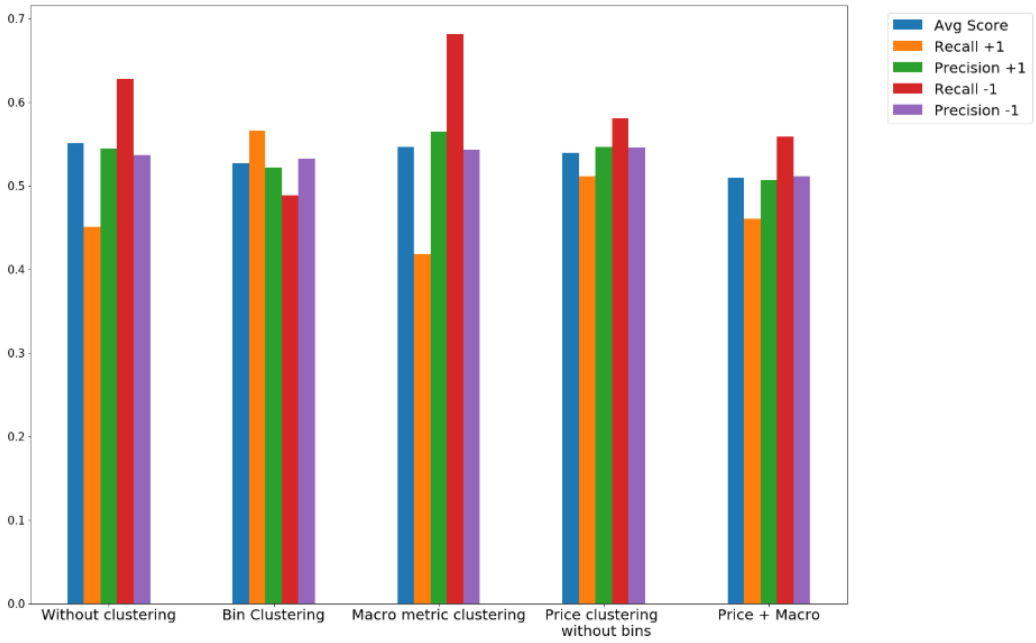


Contrairement aux cas précédents, les modèles trouvés dans ce cas ont une meilleur capacité à identifier les returns négatifs, vu la valeur du recall pour la classe (-1). La segmentation a légèrement amélioré la performance moyenne des modèles. Toutefois, elle a diminué la précision de la classe (1) notamment pour la segmentation avec les features du prix. Les modèles avec les grandes valeurs du recall (-1) avec une précision de la classe (1) $< \frac{1}{2}$ sont risqués. En effet, ils ont tendance à prédire beaucoup de faux positifs. Le modèle de la segmentation avec les deux types de données (liées au prix et les données macro-économiques) est différent des modèles obtenu lorsqu'on utilise un seul type de données, le nouveau modèle est mieux calibré pour les deux classes - toutes les valeurs sont autour de 55%.

4.3.3 • LIGHTGBM

Cet algorithme est une amélioration récente de *XGBoost* [10]. Il est basé sur une méthode dite GOSS – *Gradient-based One-Side Sampling*– pour le choix des datasets des arbres suivants. Cette méthode garde les observations avec un grand gradient par rapport à la fonction de perte (et qui correspondent donc aux données sous entrainées) et effectue un tirage aléatoire pour les instances avec un petit gradient [10]. Quelques optimisations mineures ont également été implémentées en comparaison avec la librairie XGBoost.
Cet algorithme offre une grande liberté pour la régularisation (ce qui permet d'éviter un overfit), ainsi que dans la façon dont on peut l'implémenter.

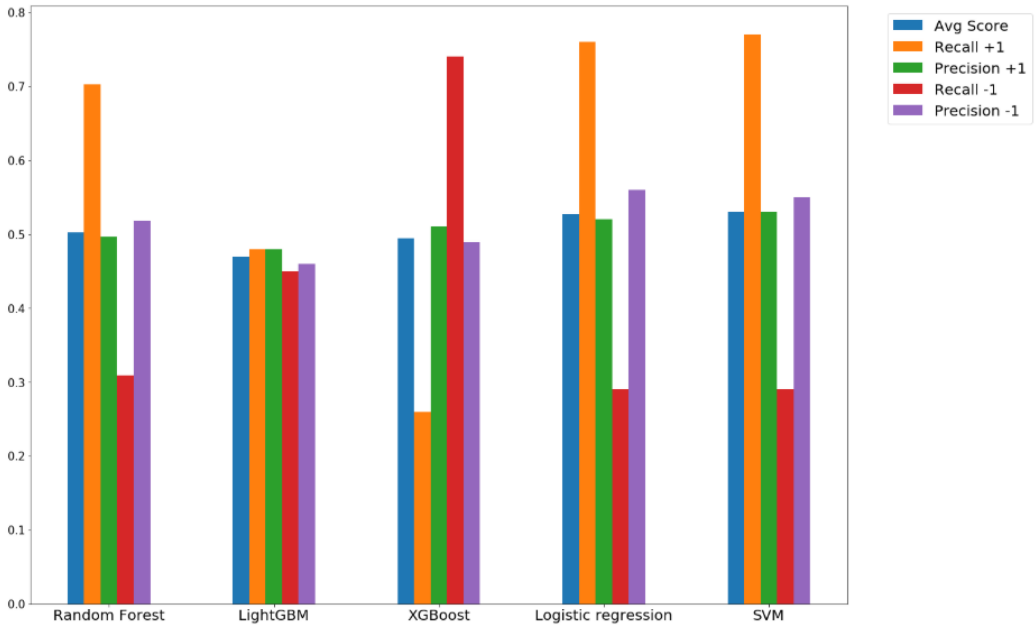
Enfin, nous avons exploité la méthode de GridSearch pour la sélection des hyperparamètres optimaux pour tous les modèles de la bibliothèque `scikit-learn`. Pour un nombre de clusters $k = 4$ avec *kmeans*, on trouve les résultats suivants :



On retrouve encore une fois un comportement similaire à celui des modèles d'XGBoost avec des modèles sont capables d'identifier les returns négatifs. La segmentation à l'aide des données macro-économiques semble donner la meilleure amélioration, en particulier en améliorant le recall (-1) ainsi que la précision (-1), ce qui veut dire que le modèle arrive à mieux identifier les returns négatifs et il est plus sûr de sa prédiction. En revanche, on s'aperçoit encore une fois que la segmentation avec les données macro-économiques et les données liées au prix ont baissé la performance, ce qui est contre-intuitif.

4.4 ESTIMATION DE L'ERREUR

On estime les performances du modèle en utilisant les données Out-of-sample qu'on a mis à part qui correspondent à la période entre 01/02/2017 et 31/10/2019.



Quelques remarques :

— La segmentation a eu un impact significatif sur les modèles linéaires, en éliminant les outliers qui empêchaient leur convergence.
— Le modèle de Random Forest a eu une performance similaire à celle trouvée pour l'ensemble de validation.
— Le modèle de LightGBM a sous-performé ce qui veut dire qu'il a été surentraîné sur l'ensemble de validation.
— Le modèle d'XGBoost a eu une performance similaire avec une légère baisse de la précision de la classe (-1).
— Les modèles linéaires ont mieux performés sur la période de test malgré leur simplicité.
— Le mode de segmentation qui améliore le plus la performance est la segmentation via HMM.

4.5 CONCLUSION PARTIELLE

A partir de l'implémentation de ces différents algorithmes, nous sommes arrivés à plusieurs conclusions, et plusieurs manières d'affiner notre analyse :

— Le fait de combiner entre les données liées au prix et les données de type macro-économique a eu un effet contre-intuitif sur la performance : les modèles fournis par cette segmentation ont généralement donné des résultats moins concluants que les autres modèles.
— Il faut faire un choix : privilégier une des deux classes. Telle qu'elle a été définie précédemment, il n'y a pas de lien clair entre les precision des 2 classes. Cependant, il est clair que dans tous nos résultats, les classifieurs qui ont une grande capacité prédictive pour une classe sont moins bon pour prédire l'autre. Ainsi, selon le type de stratégie qu'on cherche, on peut faire une sélection des classifieurs, voire même combiner entre plusieurs parmi eux.
— La meilleure segmentation dépend beaucoup du modèle de classification utilisée, et leurs per-

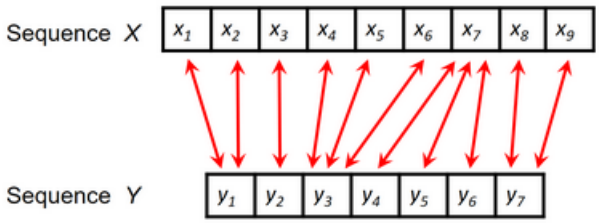
formances varient significativement. Cependant, on remarque une amélioration des performances.
— La segmentation qui utilise les bins a sous-performé les autres clusters, dans la majorité des cas, les résultats ressemblaient à ceux issus d'une Bernoulli de paramètre 0.5 (grosse p-valeur) . On peut conclure que ce choix donc n'était pas pertinent.
— Il y a 2 pistes que nous avions pensé à explorer :
  — Etudier la relation entre les données macro-économiques et les données liées aux prix afin de mieux comprendre la baisse de performance dans les différents cas, et utiliser cette relation pour améliorer ces modèles. Les test d'autocorrélations n'ayant pas abouti, les résultats les plus intéressant que nous avons trouvé dans ce sens sont à la section 5.
  — Explorer à l'aide d'autres modèles la pertinence d'utiliser des anciennes données pour faire des prédictions et en particulier, chercher une durée entraînement/prédiction qui sera optimale. Le meilleur qu'on avait pour étudier ce rapport et via un modèle de réseaux de neurones (section 6) qui s'entraine progressivement.

5
SYNCHRONICITÉ DES DONNÉES DE PRIX AVEC LES DONNÉES MACRO

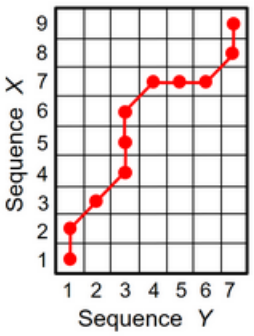
Comme on a précisé dans la partie précédente, on va essayer dans cette partie, d'examiner le lien entre les données macro-économiques et les données du prix, en particulier on va s'intéresser à la synchronicité des deux types de données. Après avoir extrait les features pertinentes pour notre étude, nous avons examiné la synchronicité des deux séries temporelles, idéalement dans le but de pouvoir dégager une tendance dans le temps que mettent les prix à s'adapter aux features macros. Commençons par une présentation des différentes méthodes que nous avons explorée pour exploiter nos données.

5.1 DYNAMIC TIME WARPING (DTW)

Une première méthode que nous avons explorée est le Dynamic Time Warping [6]. Etant donné $X = (x_1, \ldots, x_N)$ et $Y = (y_1, \ldots, y_N)$, on souhaite créer une correspondance entre les 2 jeux de données. Formellement, on cherche un chemin $(p_1, \ldots, p_L), p_l \in [1,N] \times [1,M]$ qui vérifient certaines contraintes détaillées plus loin.



Source : Muller, fundamentals of image processing, Springer

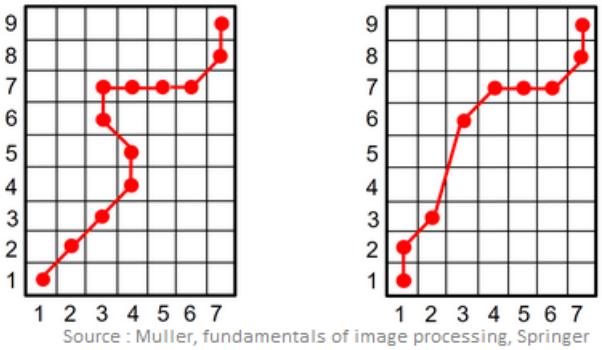


Le principe de l'algorithme de construction du chemin est le suivant :

— On définit une fonction de coût C qui prend en entrée un point du chemin. Dans notre cas, en dimension 1, le plus naturel est de prendre la distance entre les deux points de la courbe : $C(p_l) = |p_l[0] - p_l[1]|$

— En supposant p_l construit, p_{l+1} sera le point qui minimise la fonction C parmi les points adjacents à p_l qui ont des coordonnées plus grandes que p_l. Moralement, cela veut dire qu'on avance petit à petit (on ne s'autorise pas de sauter des valeurs) et qu'on ne s'autorise pas de créer un chemin qui retourne en arrière

Voici par exemple 2 chemins qui ne vérifient pas les contraintes imposées :



Source : Muller, fundamentals of image processing, Springer

C'est un algorithme hérité du domaine de la reconnaissance vocale, et nouvellement utilisé pour l'analyse des marchés. Notre idée a donc été d'adapter cet algortihme à nos données. Ce que nous nous attendions à trouver est globalement un chemin qui ressemble plutôt à une droite avec une pente proche de 1, (ce qui s'interprète comme le fait que généralement, les features et les prix "évoluent au même rythme")

5.2 PRÉPARATION DES DONNÉES ET DIFFICULTÉS RENCONTRÉES

Après quelques tests et des résultats qui sont clairement trop éloignés de la réalité, nous avons compris les spécifités de l'algorithme qui pouvaient être en cause et nous avons préparé les données afin de minimiser l'impact de ces problèmes. il y a deux difficultés majeures que nous avons recontrés :

5.2.1 • L'ORDRE DE GRANDEUR DES DONNÉES

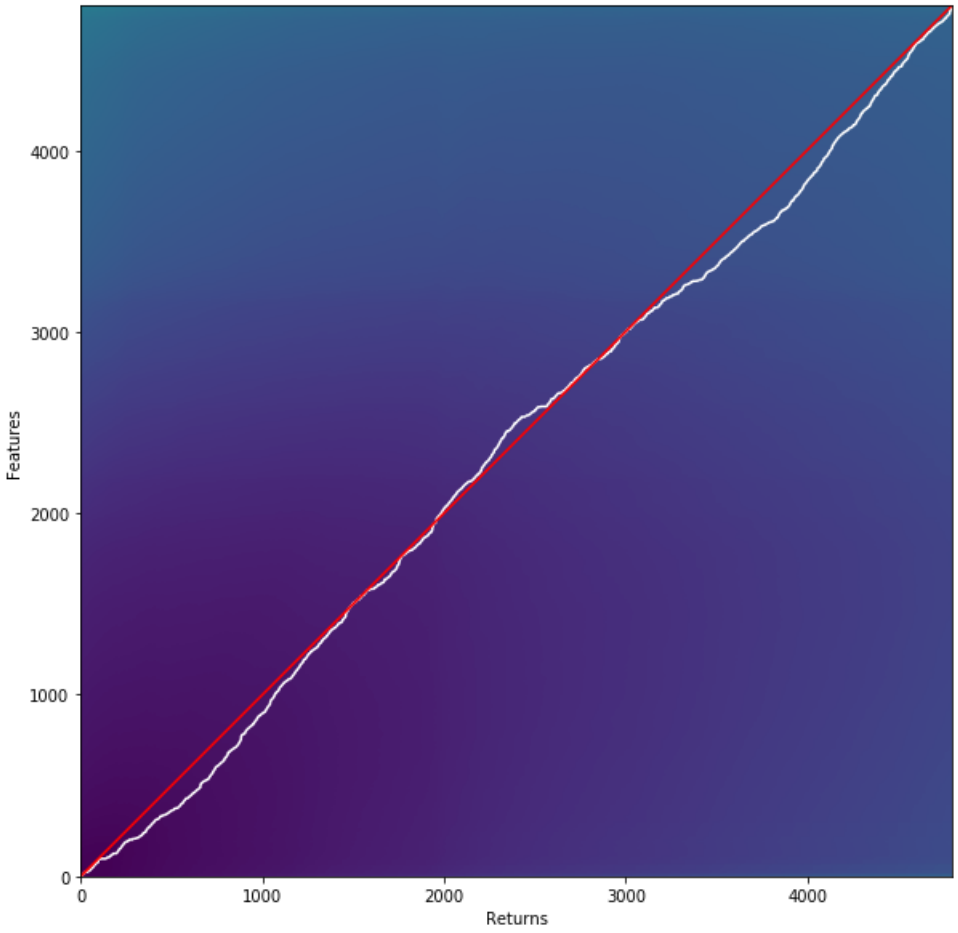
Il est très important d'avoir pour cette méthode des données qui parcourent à peu près la même plage de valeur. Comme le chemin est construit en minimisant la distance entre les points des courbes, avec des valeurs très éloignées, la méthode n'est pas fiable (nos première tentatives matchaient le maximum du vecteur avec des petites valeurs avec le minimum de l'autre, et donnait donc un chemin totalement horizontal).
Pour palier cette difficulté, nous nous sommes efforcés de choisir des données qui occupent le même jeu de valeur. nous avons donc normalisé et centré les valeurs pour avoir une analyse un peu plus pertinente.
De même, nous avons choisi d'étudier la **dérivée discrète** du vecteur de features, comme nous travaillons avec les returns (qui sont au final sensiblement similaires à la dérivée discrète du closing time), cela donne des données assez similaires et plus faciles à analyser.

5.2.2 • LES PROBLÈMES DE RÉGULARITÉ

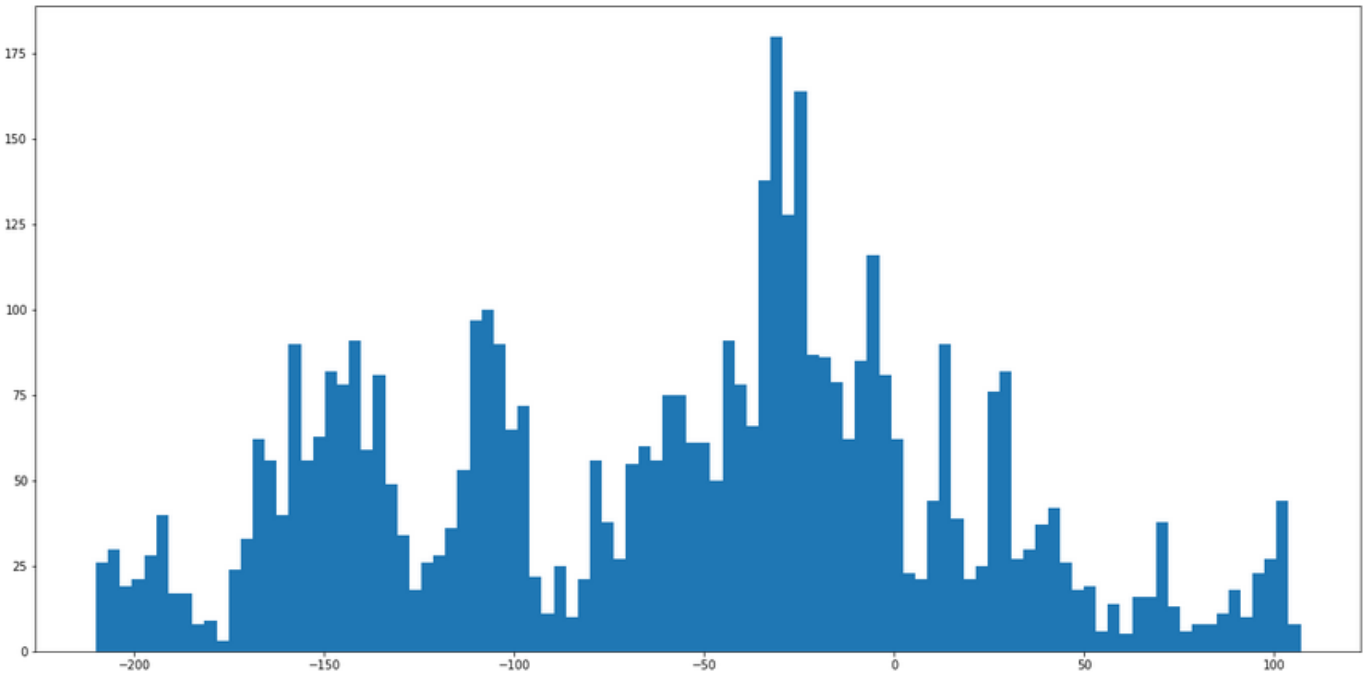
En ayant choisi de travailler avec des dérivées discrètes, les fonctions que nous avons perdent un peu en régularité, l'échelle de variation de nos valeurs est donc très petite, et cela ne convient pas forcément à la méthode DTW. Nous avons choisi de ne pas modifier les données encore plus pour résoudre ce problème, quitte à perdre en fiabilité, car cela dénaturerait trop nos entrées.

5.3 IMPLÉMENTATION ET EXPLOITATION DES RÉSULTATS

Nous avons utilisé le module dtw de python, qui crée le chemin, et qui montre une heatmap de la matrice de coût : Voici le résultat obtenu, nous avons rajouté la première bissectrice, qui correspond donc à aucun retard entre les 2 données.



On voit alors que globalement, les returns sont en retard par rapport aux features (les features expliqueraient donc l'évolution des returns futurs), mais à certains moment, la tendance s'inverse et les returns sont en avance. Aucun des 2 résultats n'est surprenant, car on a en fait une **interaction** entre les 2 : l'état des returns peut influencer les features, et vice versa.
Par ailleurs, on trouve bien une droite assez proche de la 1ère bissectrice, ce qui est bien à quoi on s'attendait.
Un peu plus quantitativement, On veut approcher le chemin obtenu par une droite, ce qui pourrait alors nous donner une idée sur le retard global de features par rapport aux returns. Grâce au module scikit, nous avons obtenu une pente de 1.002, une ordonnée à l'origine de -53 et un R^2 de 0.997. Le chemin est donc très bien approximé par la droite, et le fait que la pente soit très proche de 1 nous conforte dans nos observations. Cependant, l'ordonnée à l'origine est plus problématique. Cela voudrait dire que les returns dépendent des features d'il y a à peu près 2 mois, ce qui n'est pas du tout le cas dans la réalité. Nous nous attendions à trouver des résultats de l'ordre de quelques jours, mais on voit que le chemin met en relation des returns avec des features qui remontent parfois à 100 jours. Voici l'histogramme qui montre le retard ou l'avance des returns, en jour.



La plus grosse concentration est autour de -20, ce qui est déjà plus raisonnable mais cette valeur est quand même élevée. Néanmoins, il ne faut pas ignorer que les valeurs vont jusqu'à -200. Ce problème vient surement en partie des remarques que l'on a émises plus haut et qui sont liées à la méthode : l'algorithme s'adapte trop lentement au changement. A chaque étape le chemin avance vers un point adjacent, ce qui fait que si on matche un return avec une feature 40 jours en arrière (Ce qui pourrait arriver à cause d'un événement marquant, en cas de crise par exemple) alors le return suivant sera lui aussi matché avec une feature de cet ordre de grandeur.

Mais le problème majeur dont nous nous sommes rendus compte est que les échelles de temps ne sont pas adaptées pour ce genre d'analyse. Quand on essaie de trouver un retard de l'ordre de quelques jours au plus (Le développement des moyens de communication et d'autres techniques de trading à haute fréquence feraient encore diminuer cette estimation), on aurait besoin de données un peu plus fréquentes, c'est-à-dire plusieurs par jour plutôt qu'un par jour.

Il n'empêche que si les données quantitatives sont peut-être pas assez précises pour en tirer une conclusion claire, une conclusion partielle suite à l'implémentation de cette méthode est que globalement les returns tendent à être en retard vis à vis des features, et qu'il n'est pas rare que cette tendance s'inverse.(A peu près 20% des returns sont en fait en avance par rapport aux features)

6
OUVERTURE SUR L'UTILISATION DE RÉSEAUX DE NEURONES

On s'est intéressé aux réseaux de neurones particulièrement à cause du plafonnement des performances pour les méthodes d'apprentissage supervisée, et afin de tester la durabilité des modèles dans le temps. Les réseaux de neurones artificiels sont une classe de systèmes d'apprentissage statistique conçus sur la base du fonctionnement biologique des neurones.

Ils mettent en œuvre le principe de l'induction, c'est-à-dire l'apprentissage par l'expérience. Par confrontation avec des situations ponctuelles, ils infèrent un système de décision intégré dont le caractère générique est fonction du nombre de cas d'apprentissages rencontrés et de leur complexité par rapport à la complexité du problème à résoudre. Par opposition, les systèmes symboliques capables d'apprentissage, s'ils implémentent également l'induction, le font sur base de la logique algorithmique, par complexification d'un ensemble de règles déductives. La force des réseaux de neurones est donc qu'ils peuvent apporter des solutions très éloignées de celles qu'on aurait pu développer nous-mêmes.

La puissance de cette classe d'algorithmes s'est avérée de plus en plus palpable ces dernières années, et ce malgré le fait qu'il s'agit d'une approche connue déjà en 1958, car la récente amélioration de la puissance de calcul des processeurs a enfin permis de concrétiser ces objets théoriques jusqu'à complètement dominer les approches utilisées avec certains problèmes d'apprentissage et prédiction tels la reconnaissance visuelle d'objets.

Ce type d'architecture a été aussi incorporée dans la large littérature de la prédiction des marchés financiers, et nous nous en sommes inspirés pour construire et évaluer de tels modèles de prédiction, et plus précisément pour résoudre un problème de classification.

6.1 MODÈLE

Le modèle implémenté est formé de 3 couches denses, dont les paramètres sont :

— **Couche 1** : 256 neurones , activation : linéaire
— **Couche 2** : 128 neurones , activation : sigmoïde
— **Couche 3** : 16 neurones , activation : linéaire
— **Couche 4** : 1 neurone , activation : linéaire

La théorie des réseaux de neurones est assez large, cependant il n'existe pas de théorie qui nous décrit l'architecture du réseau optimal, cette partie relève plus d'une combinaison d'analyse empirique (en testant plusieurs architectures) et d'intuition mathématique. Le réseau utilisé ici se compose de 4 couches denses avec un certain nombre de neurones dans chacune d'eux. Le choix du nombre de couches est donc assez arbitraire et empirique. Nous avons fait quelques essais en changeant le nombre de couches, ou de neurones par couche, sans obtenir des résultats significativement différents. Intuitivement, augmenter le nombre de couches/ neurones augmente la complexité du réseau, et

théoriquement sa capacité à reconnaitre des motifs plus complexes, en contre-partie cela augmente la durée d'apprentissage du réseau tout en l'exposant davantage à des problèmes qui le rendent incapable d'apprendre (On peut citer le problème de vanishing/exploding gradient). L'amélioration de la performance d'un réseau est souvent assez incrémentale, soit en utilisant des techniques de régularisation (normalisation de batch, couche dropout, data augmentation) soit en utilisant des sous-architectures prédéfinies grâce à la littérature existante, cela est notamment le cas pour les problèmes de vision ou de reconnaissance vocale, on n'a cependant pas trouvé d'équivalent pour la problématique sous-jacente à notre projet.

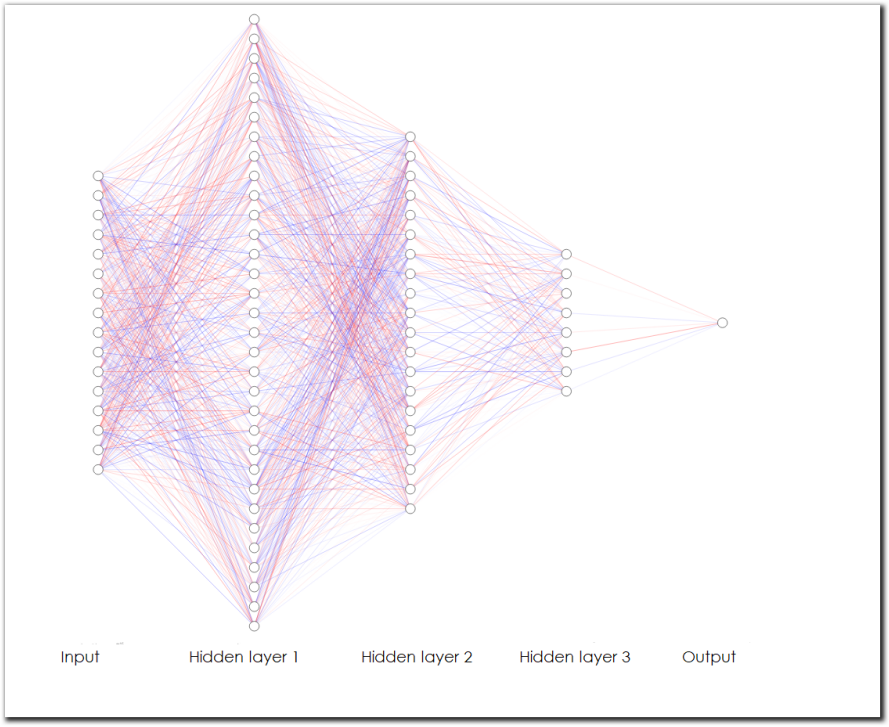


FIGURE 11 – Schéma du modèle

6.2 ASPECTS THÉORIQUES DU MODÈLE

Le fonctionnement d'un réseau de neurones peut être découpé en deux parties :

6.2.1 • ENTRAÎNEMENT

Le réseau peut être vu comme une fonction $F(\Theta, X)$ qui va approximer la valeur associée à la donnée d'entrée X sachant un certain nombre de paramètres Θ, ces paramètres dépendent des choix qu'on a fait pour construire notre réseau de neurones, à savoir :

— Les tailles des couches du réseau

— Les fonctions d'activation de chaque réseau
— Les poids régissant chaque paire de neurones

Plus formellement, supposons que l'on dispose des données $D = \{(X_i, y_i) | 1 \leq i \leq N\}$, on cherche à minimiser une fonction de coût f, par exemple la somme des distances au carré entre les prédictions et les valeurs réelles :

$$f(\Theta, D) = \frac{1}{N} \sum_{i=1}^{N} ||F(\Theta, X_i) - y_i||^2 = \frac{1}{N} \sum_{i=1}^{N} ||y_{pred_i} - y_i||^2$$

Le réseau va alors jouer sur Θ pour minimiser f. En pratique, il s'agit de réaliser une descente de gradient sur chaque paramètre unitaire θ de Θ :

$$\forall \theta_n \in \Theta_n : \theta_{n+1} = \theta_n - \eta \frac{\partial}{\partial \theta} f(\Theta_n, D)$$

Avec η un paramètre d'apprentissage à fixer.
Ainsi, le réseau va itérativement mettre à jour Θ dans l'optique de minimiser $f(\Theta, D)$.

6.2.2 • ÉVALUATION

Il s'agit ici d'évaluer les prédictions du modèle sur un ensemble test $T = \{({X^{test}}_i, {y^{test}}_i) | 1 \leq i \leq L\}$, et puisqu'il est question de classifier les données, l'évaluation se fait sur le taux de prédictions correctes et fausses. Cependant comme remarqué précédemment, la pertinence du modèle n'est pas évaluée selon la même métrique, et on utilise des mesures plus fines.

6.3 APPLICATION AUX DONNÉES

La théorie qui décrit le fonctionnement des réseaux de neurones est universelle mais il n'en demeure pas moins qu'elle ne définit pas l'approche pratique à implémenter, car découper les données en deux (entraînement et test) puis exécuter l'algorithme n'est pas forcément l'approche la plus pertinente notamment à cause des problèmes récurrents de sur-apprentissage [12].

Nous nous sommes donc intéressés à trois implémentations pratiques :

— **Approche naïve** : On entraîne le modèle sur les données de 1995 jusqu'à 2012 puis on le teste de 2013 à 2020.
— **Backtesting classique** : Il s'agit ici d'entraîner progressivement le modèle. Typiquement, si à l'étape k de l'entraînement on l'a entraîné sur la base de données entre $date_{k-1}$ et $date_k$, on va alors le tester de $date_k$ jusqu'à $date_k + \Delta$ (Δ est un paramètre fixé), puis pour l'étape $k+1$ on va l'entraîner entre $date_k$ et $date_{k+1}$...L'objectif étant de suivre l'entraînement du modèle et de mieux analyser sa performance, et aussi d'éviter le surentraînement, car les périodes d'entraînement sont assez courtes.
— **Backtesting à courte mémoire** : le principe est le même que pour le backtesting classique, sauf que le modèle ne garde en mémoire que les paramètres appris sur les λ dernières étapes (où λ est un paramètre fixé).

Remarque : Les deux dernières approches introduisent des nouveaux paramètres qui peuvent être optimisés à leur tour.

6.3.1 • APPROCHE NAÏVE

Un moyen assez répandu pour visualiser l'apprentissage d'un réseau de neurones est de tracer les valeurs de la fonction de coût en fonction du temps (ou encore les itérations sur les données).

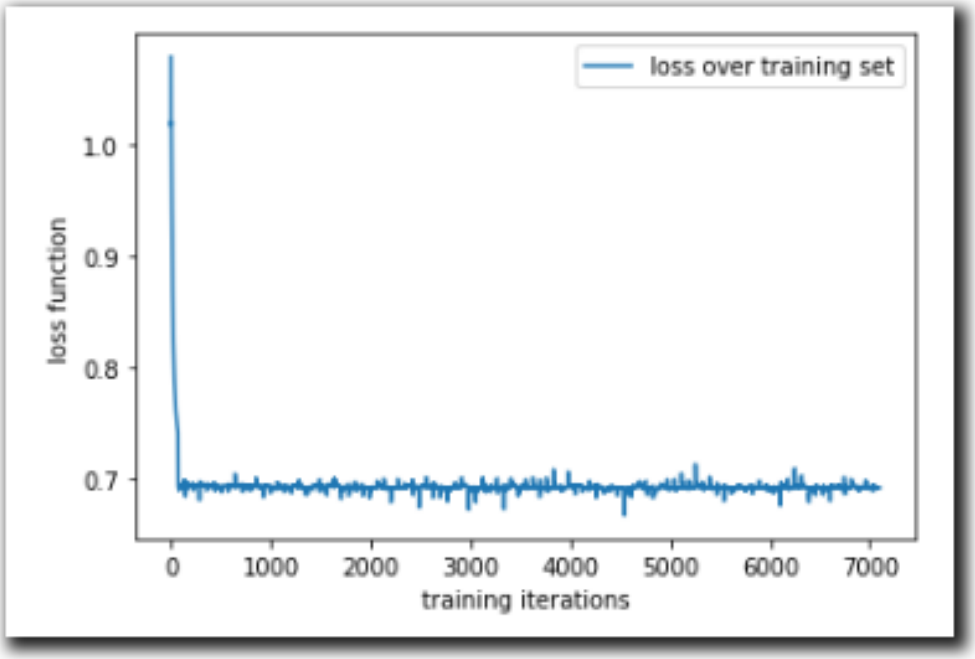


FIGURE 12 – Erreur au cours de l'apprentissage

Ce graphe montre que la fonction coût est décroissante au début puis va rapidement se concentrer sur une valeur constante et fluctuer autour d'elle. Les fluctuations s'expliquent par le caractère stochastique des algorithmes de descente du gradient, d'autre part le fait de converger rapidement sans pourvoir minimiser d'avantage la valeur limite traduit la difficulté d'apprentissage du réseau. Autrement dit le réseaux a du mal à détecter des motifs permettant de mieux associer à chaque attribut sa valeur prédite, il apprend rapidement le peu d'informations utiles au bout d'environ 100 itérations puis stagne. Ce qui était envisageable, vu la complexité des données, et le caractère stochastique de la bourse.

Néanmoins, l'évaluation sur les données test a donné lieu à une prédiction avec un taux de réussite de 54.49%.

6.3.2 • BACKTESTING

Comme décrit précédemment, le modèle a été entraîné itérativement en découpant l'ensemble des données en plusieurs périodes et on a calculé progressivement les performances du modèle entraîné sur les tests itératifs.

Une première problématique qu'on a rencontré est la définition d'une métrique capable de refléter la performance du modèle.

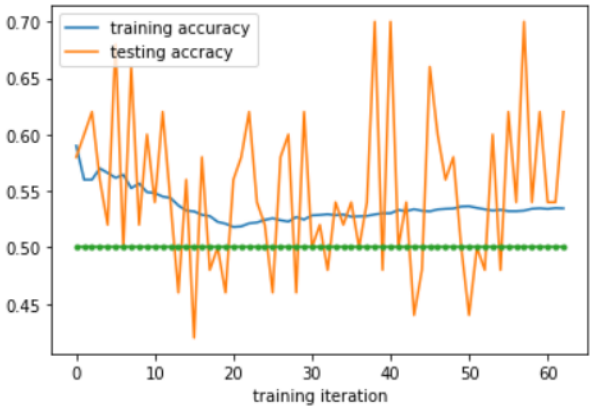


FIGURE 13 – Erreur au cours de l'apprentissage avec une autre approche

Le graphe 15 représente l'évolution des taux de prédiction correctes du modèle sur son ensemble d'entraînement d'une part et sur les ensembles de tests itératifs d'autre part, la ligne verte correspond à la valeur constante 50%. On a utilisé ici une incrémentation de l'entraînement égale à 100 jours, car on a vu qu'au bout d'une centaine de jours (avec l'approche naïve) le réseau était exposé au surentraînement et une durée de test égale 50 jours. Ainsi à chaque itération on entraîne le modèle sur les 100 jours suivants et on le teste sur les 50 jours qui suivent directement ces 100 jours.

Un moyen assez représentatif pour décrypter ce signal est de regarder la moyenne des performances sur les itérations. Le signal précédent donne lieu à une moyenne de taux de prédiction réussie égale à 54.98%.

Une fois qu'on a décidé de cette métrique comme moyen d'évaluer la performance du modèle, on peut optimiser les paramètres propres au backtesting, i.e la longueur des incrémentations pendant l'entraînement et les durée de tests.

On a ainsi mené des tests avec différentes valeurs d'incrémentation pour le backtesting, les résultats sont résumés par les trois figures ci-dessus.

Pour mener une analyse rigoureuse de ces courbes il est nécessaire de prendre en compte les variations des performances selon chacun des paramètres. On remarque d'abord que même si on observe des performances assez élevées pour certaines valeurs du couple (incrémentation de l'entraînement, durée du test)

Cela dit, on observe que l'évolution globale des performances est positivement corrélée à Δ_{train} jusqu'à $\Delta_{train} = 700$ jours. D'autre part l'évolution selon Δ_{test} est globalement croissante en moyenne puis décroissante, avec une variance décroissante, ce qui est normal car on augmente le nombre de valeurs sur lesquelles on test, et on est un peu plus surs de nos résultats. C'est pour cela que même si des performances élevées sont à signaler pour des petites valeurs de Δ_{test}, cela n'a pas beaucoup de valeur pratique, car tester sur peu de points est assez peu significatif.

On peut alors en déduire que la valeur de $(\Delta_{train}, \Delta_{test})$ qui optimise les performance du modèle tout en donnant des résultats consistants en terme de variance est :

$$\Delta_{train} = 700, \Delta_{test} = 45$$

Avec une performance moyenne $\approx 65\%$ Cette performance d'apparence élevée est à revoir à la

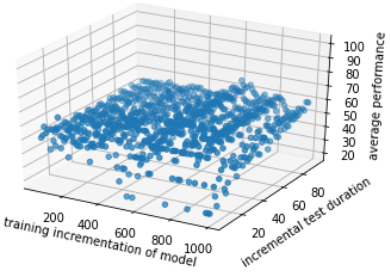


FIGURE 14

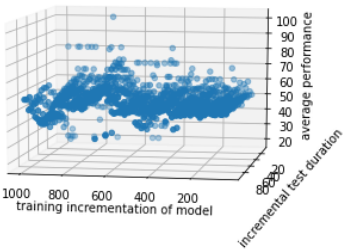


FIGURE 15

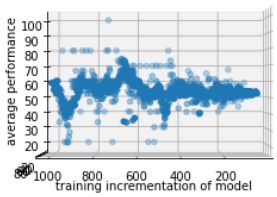


FIGURE 16

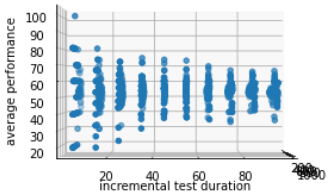


FIGURE 17

baisse du fait du biais de sélection : La fonction de performance moyenne est à vrai dire un estimateur de nature aléatoire. L'intervalle de confiance de l'estimateur maximale est plus grand que l'intervalle de confiance d'un estimateur normal. Le fait de sélectionner le maximum introduit une variance supplémentaire. De telles analyses ont souvent conduit au choix d'hyper paramètres donnant des performances extradordinaires mais qui finissent ultimement par sous-performer une fois testé sur des données futures.

6.4 RÉSEAUX DE NEURONES À LONGUE-COURTE MÉMOIRE (LSTM)

Long sort-term memory (LSTM) est une classe d'algorithme de prédiction du type réseau de neurones qui a la faculté de mieux gérer le traitement des séries temporelles et qui pourrait donc mieux s'adapter à notre problème. En effet les LSTM se basent sur un principe de tri des motifs stockés en mémoire par le modèle, autrement dit au fur et à mesure que le modèle s'entraîne sur un même ensemble de données un mécanisme probabiliste va progressivement ignorer certaines parties de la série temporelle car jugées peu informatives, en tronquant simplement la série temporelle sur laquelle on s'entraine. Mathématiquement celle-ci ce traduit par une variation du gradient quasi nulle en cette troncature.

La problématique fondamentale que les LSTM permettent de résoudre est en fait le phénomène de Vanishing/Exploding gradient rencontré sur des structure moins sophistiqué de réseaux de neurones,

qui se traduit en une incapacité du modèle à minimiser considérablement la fonction coût lors de la descente du gradient. Plusieurs facteurs peuvent expliquer ce comportement aberrant du modèle rencontré durant l'application de l'approche naïve.

Il est donc naturel d'implémenter un réseau de neurones sur l'archétecture LSTM. On a ainsi construit un modèle similaire au précèdent avec cette fois ci la structure suivante :

— **1ère couche LSTM** : 256 neurones ,
— **2ème couche LSTM** : 128 neurones ,
— **3ème couche LSTM** : 16 neurones ,
— **Couche dense finale** : 1 neurone , activation : Sigmoid.

L'utilisation de la fonction sigmoïde dans la dernière couche dense est nécessaire pour garantir une valeur de sortie dans l'intervalle $[0,1]$ afin de pouvoir procéder à la classification.

On applique alors le même algorithme de Backtesting utilisé pour le modèle classique des réseau de neurones implémenté précedement, de plus, on effectue une étude similaire pour optimiser les paramètres $(\Delta_{train}, \Delta_{test})$ du modèle LSTM. Les résultats sont représentés sur les graphs suivants :

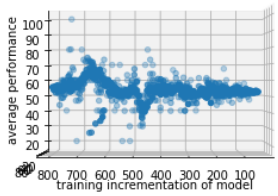


FIGURE 18

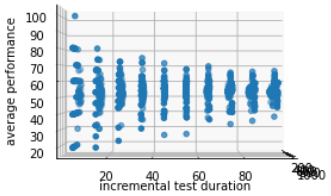


FIGURE 19

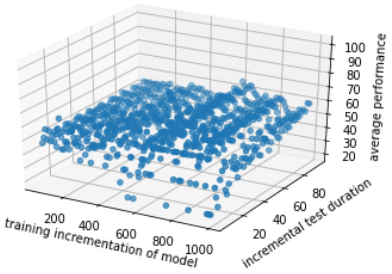


FIGURE 20

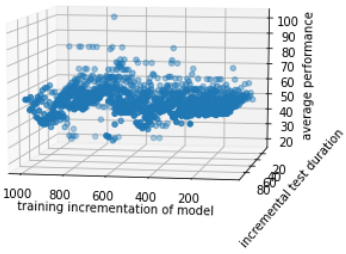


FIGURE 21

Les variations observés des performances moyennes en fonction de Δ_{train} sont quasi identiques à ceux du premier réseaux de neurones implémenter, en revanche on peut remarquer que les nuages de points sont plus compacts (figure 8), ce qui traduit une variance plus faible par rapport à celle observé en (figure 5). La structure LSTM a donc permis de renforcer la consistance du modèle précèdent, elle n'a sert pas donné lieu à une amélioration considérable au niveau des performances moyennes (que l'on a déjà trouvé un peu trop élevées) mais a en contrepartie contribué à réduire la variance des performances moyennes selon Δ_{train}, ce dont souffrait notre modèle précédent.

D'autre part, on retrouve aussi des variations selon Δ_{test} quasi identiques à ceux du modèle initial avec une baisse significative de la variance quand Δ_{test} augmente.

Les performances moyennes sont optimales (maximisation de leur valeur avec une bonne consistance géométrique (faible variance)) pour :

$$(\Delta_{train}, \Delta_{test}) = (650, 45)$$

avec une performance moyenne $\approx 66\%$, les remarques précédent sur le biais de sélection entrepris dans cette démarche d'optimisation reste cependant valable.

Cela veut dire que le comportement du LSTM donne à peu près les même résultats d'optimisation que le réseau de neurone normal. On peut donc retenir, qu'afin d'entraîner un modèle de réseau de neurones, il est bien de de commencer par entraîner sur les 700 derniers jours, et ne pas l'appliquer plus loin que sur les 2 mois qui suivent.

CONCLUSION

Ce travail de recherche nous a permis d'apprendre plusieurs méthodes de classfication et d'apprentissage statistique et essayer de les évaluer en nous confrontant aux problèmes de réglage des paramètres.

Nous nous sommes rendus compte, en appliquant ces même méthodes sur d'autres problèmes, de leur puissance de prédiction qui dépasse celle de l'humain. Leur performance à peine meilleure que le hasard dans le problème que nous traitons au PSC prouve le degré de complexité du fait de la préditcion des retours journaliers du marché boursier. On aurait pu aussi nous intéresser à des temps plus longs en espérant faire compenser le bruit, ou beaucoup plus court afin de rendre compte des inefficiences qu'il puisse y avoir à cette échelle et comparer la puissance de nos modèles.

Le PSC nous a aussi permis d'améliorer notre compréhension des différentes features décrivant le marché. Si les variables en rapport avec le S&P500 (returns, volume, VIX , PUT/CALL) traduisent directement la vision des investisseurs, d'autres features auxquels on avait accés était directement liés à l'économie réelle en suivant des taux de production ou même les inventaires en industrie. Il n'était pas sans surprise de nous rendre compte que l'ajout de ce genre de features baissait la qualité de nos modèles tout au long du PSC tant ils sont de nature différente.

Le choix des modèles, features et paramètres était compliqué et souvent difficilement explicable du fait de la nature quasi aléatoire des données. On s'était confronté au cours de nos recherches plus qu'une fois à des modèles qui avaient un pouvoir de prédiction moins bon que le hasard sans savoir donner aucune explication à leurs comportements. On a choisi de présenter dans ce rapport que les meilleurs résultats qui ont réussi les backtests et les périodes de validations. Il y a dans cette démarche d'apparence expérimentale, un biais de sélection. Ce biais est difficilement quantifiable sinon par le nombre d'échecs avant de trouver un modèle réussi. La petite taille des données auxquelles nous avons eu accès pour les tests (une centaine de jours) rend cette éventualité non négligeable.

Autrement dit, la plupart de nos résultats sont à prendre à titre hautement indicatif.

RÉFÉRENCES

[1] Myron Scholes FISHER BLACK. In : (1973).

[2] Maciej Klimek et Johan Tysk GUSTAF STRANDEL. « Testing weak stationarity of stock returns ». In : (2000).

[3] Robert Tibshirani JEROME H. FRIEDMAN et Trevor HASTIE. « The Elements of Statistical Learning : Data Mining, Inference and Prediction, Chapter 10 : Boosting and Additive Trees, p.314 ». In : (2009).

[4] Jason BROWNLEE. In : (2014). URL : https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/.

[5] In : (2015). URL : https://scikit-learn.org/stable/modules/clustering.html.

[6] Meinard MÜLLER. « Fundamentals of Music Processing ». In : (2015).

[7] Von DGEROMIC. « Machine Learning-Based Classification of Market Phases ». In : (2017). URL : http://riskdatascience.net/machine-learning-based-classification-of-market-phases.

[8] Massimo Cassia & Bruno REMILLARD. « Option pricing and hedging for discrete time autoregressive hidden markov model ». In : (juil. 2017).

[9] In : (oct. 2018). URL : https://rubikscode.net/2018/10/29/stock-price-prediction-using-hidden-markov-model/?fbclid=IwAR16rkzONuWbgPUsVPxfy4JCMPcd-d6FCKHmMBsIfa2KZ

[10] In : (jan. 2018). URL : https://mlexplained.com/2018/01/05/lightgbm-and-xgboost-explained/.

[11] Mamed CAKI. « Economic regime identification using Machine Learning Techniques ». In : (juil. 2018).

[12] Vegard FLOVIK. In : (juin 2018). URL : https://towardsdatascience.com/how-not-to-use-machine-learning-for-time-series-forecasting-avoiding-the-pitfalls-19f9d7adf424.

[13] Ramya Bhaskar SUNDARAM. In : (2018). URL : https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost.