Utilisation d'outils de machine learning en environnement Python pour le trading sur contrats futures européens (Dax, EuroStoxx)

Adrien DUBOIS
Tristan AMADEI
Rémi HURSTEL
Ranitea GOBRAIT

Janvier - Juin 2022

Encadrants : Laurent ABRIL & Sylvain LE CORFF

Table des matières

Remerciements

Avant toute analyse de notre projet Cassiopée, il apparaît opportun de commencer ce rapport par des remerciements à ceux qui nous ont beaucoup appris au cours de ce projet, mais également à l'équipe qui a fait de ce projet un moment très profitable. Ainsi, nous souhaitons remercier :

- **Laurent ABRIL**, notre tuteur du projet, pour nous avoir accueillis chez lui et transmis sa passion pour l'analyse de marché, mais aussi pour toutes les heures qu'il nous a consacrées pour répondre à toutes nos questions. Sa pédagogie et sa patience nous ont motivé tout au long du projet. Il a su nous mettre en confiance et nous pousser dans nos retranchements. Nous le remercions chaleureusement pour sa confiance et son dévouement.

- **Sylvain LE CORFF**, enseignant-chercheur en statistiques à l'Institut Polytechnique de Paris, pour son temps, très précieux du fait de ses engagements académiques, et son expertise lors de notre projet. Ses conseils nous ont grandement aidé à progresser et nous avons pu découvrir des outils mathématiques qui nous étaient encore inconnus.

- **Joséphine KOHLENBERG**, responsable des programmes Cassiopée, pour la réalisation et l'organisation du suivi des différents programmes du projet Cassiopée.

Introduction

Du fait de sa place centrale dans l'économie mondiale, les marchés financiers ont suscité un très grand engouement ces dernières années. Comme la plupart des stratégies d'investissement et de trading, les marchés financiers utilisent eux aussi l'analyse technique. Cette dernière repose sur l'étude des graphiques de cours de la bourse mais aussi sur l'étude des différents indicateurs boursiers ; à l'inverse de l'analyse fondamentale qui se base essentiellement sur des éléments factuels. Cette frénésie pour les marchés financiers souligne l'importance de la prédiction de l'évolution des prix d'un actif financier en s'appuyant sur des données du passé.

Cependant, beaucoup de traders omettent encore les volumes, et en particulier la répartition des volumes acheteurs et volumes vendeurs dans leurs analyses. Pourtant, ces éléments sont cruciaux et lorsque nous les négligeons, une grande partie de l'information fournie par les bourses est perdue. De ce fait, notre projet Cassiopée vise à prendre en compte cette répartition des volumes dans notre analyse des marchés financiers. Mais ce projet nous donne aussi l'opportunité de créer, grâce aux différents algorithmes de Machine Learning, un moyen de prédire l'évolution des prix.

1 Le monde de la finance

Avant même de rentrer dans les détails techniques de notre projet, il est primordial de comprendre les bases du monde de la finance.

1.1 Introduction

Pour avoir une idée de ce que nous étudions, nous avons d'abord cherché à comprendre ce que sont les marchés financiers et quel est leur fonctionnement mais nous avons aussi cherché à comprendre ce qu'étaient les contrats futures et les indices boursiers.

1.1.1 Les marchés financiers

Un marché financier est défini comme un lieu physique ou virtuel. Dans un marché financier, nous pouvons retrouver les acheteurs et les vendeurs qui en sont les principaux acteurs. Ces derniers se retrouvent afin de pouvoir négocier des produits financiers. C'est donc ce processus qui permet aux investisseurs de faire des placements et de financer l'économie actuelle.

Comme le montre la Figure 1 ci-dessous, un marché est dit "*primaire*" lorsque des entreprises ou bien l'État ou les collectivités publiques émettent des actions, des obligations ou des titres de créance pour satisfaire leurs besoins de financement. Pour illustrer cela, nous pouvons notamment prendre l'exemple d'une introduction en bourse d'une entreprise ou l'augmentation de capital. Par la suite, les titres obtenus seront négociés sur un marché "*secondaire*" auprès des acheteurs et vendeurs.

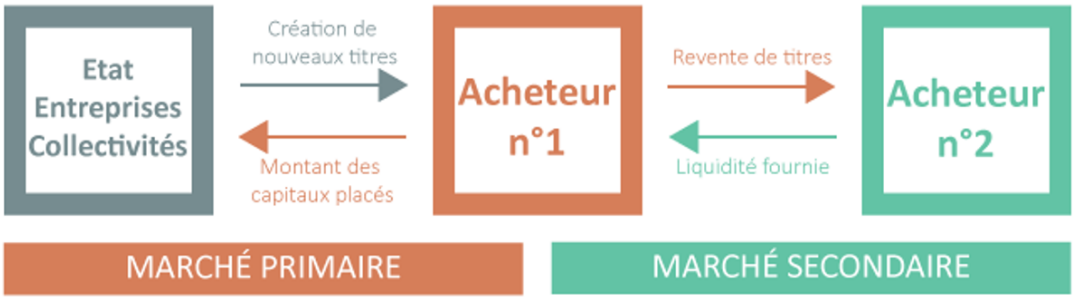


FIGURE 1 – Schéma descriptif du principe des marchés financiers

Notons également qu'il y a plusieurs types de marché. Ces derniers se distinguent et s'entremêlent suivant différents critères d'économies, d'organisations et d'engagements.

1.1.2 Les contrats futures

Les contrats *futures* sont souvent caractérisés comme les plus anciens types de contrats encore utilisés aujourd'hui. C'est aussi un marché qui est caractérisé par un échange de gros volume de produits financiers. Ce sont des instruments financiers utilisés pour anticiper les variations futures d'un actif sous-jacent. En autres mots, vous achetez ou vendez une quantité déterminée de cet actif sous-jacent, à une date d'échéance et à un prix connus à l'avance. Ainsi, nous pouvons retrouver ce produit financier sur le marché des échanges de matières premières comme les céréales ou encore bien l'or ou le pétrole. Mais, c'était avant tout un moyen pour les producteurs de

se prémunir d'une éventuelle chute des prix. En effet, les contrats futures permettaient aux producteurs de vendre leurs matières premières avant qu'elles ne soient produites. C'est pourquoi cette vente dite dans le "futur" a donné son nom à ce concept.

De nos jours, nous utilisons principalement ces contrats futures pour investir ou spéculer sur les différents indices boursiers. Par exemple, en finance de marché, le sous-jacent d'un contrat future est composé d'indices boursiers. La valeur de ce dernier se base sur le cours des principales entreprises cotées en bourse. C'est ainsi que les spéculateurs vont commencer à anticiper le sens du cours de la bourse (haussier ou baissier) afin d'acheter ou bien de vendre ces titres financiers.

1.1.3 Les indices boursiers

Comme il a été souligné précédemment, les indices boursiers occupent une place centrale au sein du marché financier et plus particulièrement des contrats futures. D'où l'importance de comprendre ce qu'est un indice boursier.

Un indice boursier est une valeur qui est obtenue grâce aux cours des actions des plus importantes entreprises cotées en bourse d'un pays. À ce titre, nous pouvons retrouver parmi ces indices boursiers le CAC-40 (Français), le Dow Jones (Américain), le FTSE-100 (Anglais) ou le DAX-30 (Allemand). Les professionnels du milieu s'accordent sur le fait que ce dernier n'a pas de valeur précise. En effet, un indice fluctue dans le temps et évolue en points afin de refléter le prix des actions de chacun des actifs sous-jacents. De plus, en allant plus dans les détails, nous pouvons remarquer que certains indices boursiers n'ont pas la même pondération pour chacune de leur action et que certaines ont plus de poids que d'autres.

1.2 Pourquoi le DAX ?

Le DAX, ou Deutscher AktienindeX, est le principal indice boursier allemand publié pour la première fois en 1988. Avant le 21 septembre 2021, ce dernier se basait uniquement sur le cours des actions des 30 principales sociétés allemandes de la Bourse de Francfort. Depuis ce jour, 10 nouvelles entreprises l'ont intégré passant des 30 plus grandes entreprises aux 40 plus grandes. Ce changement fut attendu par de nombreuses personnes. En effet, le DAX couvrait auparavant une infime partie du marché boursier national par rapport à la moyenne de l'ensemble de ce dernier.

Bien que le DAX allemand ait quelques similitudes avec le CAC-40 français, son caractère volatil le démarque de son homologue. C'est précisément ce caractère qui nous intéresse, d'où notre choix d'utiliser les données de cet indice boursier. Parmi les entreprises qui le composent, nous pouvons retrouver : Siemens, Deutsche Telekom, Allianz, Volkswagen, Adidas, BMW, Metro, Deutsche Lufthansa, Bayer AG, BASF, Henkel, Daimler etc.

1.3 L'analyse technique des marchés financiers

De nos jours, l'analyse technique est souvent considérée comme l'une des meilleures méthodes pour la prédiction des cours boursiers par les professionnels du milieu. Mais, c'est grâce à John Murphy, analyste des marchés financiers spécialisé dans l'analyse technique, que nous avons aujourd'hui une définition précise de cette notion. À travers son livre L'analyse technique des marchés financiers [1], le lecteur comprend que "*l'analyse technique est l'évolution d'un marché, principalement sur la base de graphiques, dans le but de prévoir les futures tendances*". C'est

pourquoi l'analyse technique permet de déterminer le moment opportun pour "entrer" sur le marché. D'autre part, l'analyse technique trouve ses ressources dans l'histoire. En effet, il est fréquemment dit que l'histoire se répète. De ce fait, il est possible d'identifier la tendance sur une période précise. De plus, l'analyse technique est caractérisée par le fait qu'elle s'adapte à tous types de marchés, s'appuie sur des graphiques et met en évidence les tendances futures.

Dans notre cas, nous avons décidé d'utiliser les données issues du *tape* du DAX. Le *tape* est connu comme étant un enregistrement continu de toutes les transactions sur un indice boursier. Historiquement, ce *tape* (traduction anglaise de ruban) était un long bout de papier sur lequel figuraient les prix et les volumes de transactions d'un produit financier. Nous pouvons faire un échantillonnage de cet enregistrement continu pour représenter les transactions sur différentes tranches de temps : pour ce faire il a fallu introduire les bougies. Une bougie traduit l'évolution d'un cours boursier sur un instant donné. Dans le cadre de notre projet, nous étions amenés à travailler avec des bougies de 5 minutes et des bougies journalières, ce qui nous a permis de trouver un juste milieu entre le trading haute fréquence (certes intéressant mais qui est très couteux en mémoire au vu du nombre de données) et l'investissement.

Par la suite, il est important de préciser que l'analyse technique respecte principalement 3 règles. Une de ces règles est que le marché suit des tendances. Comme il a été mentionné précédemment, les graphes sont les éléments clés de l'analyse technique. Sur ces graphiques, nous pouvons apercevoir des droites de tendances. La Figure 2 (ci-dessous) nous montre quelques exemples : les **droites de support** et les **droites de résistance**. Ce sont des seuils qui permettent d'indiquer des retournements de tendance.

FIGURE 2 – Exemple de droites de résistance et de support

C'est ainsi que nous pouvons dégager 3 tendances d'évolution montrées dans la Figure 3 (page suivante) : **la tendance baissière**, **la tendance haussière** et la **tendance latérale**.

À noter que les Figures 2 et 3 illustrent également une des principales représentations graphiques utilisées pour la représentation des données : les chandeliers japonais. Cependant, il en existe plusieurs autres : les courbes continues, les bars charts ou graphiques en bâtonnet, mais la représentation avec les chandeliers japonais est celle qui est la plus populaire.

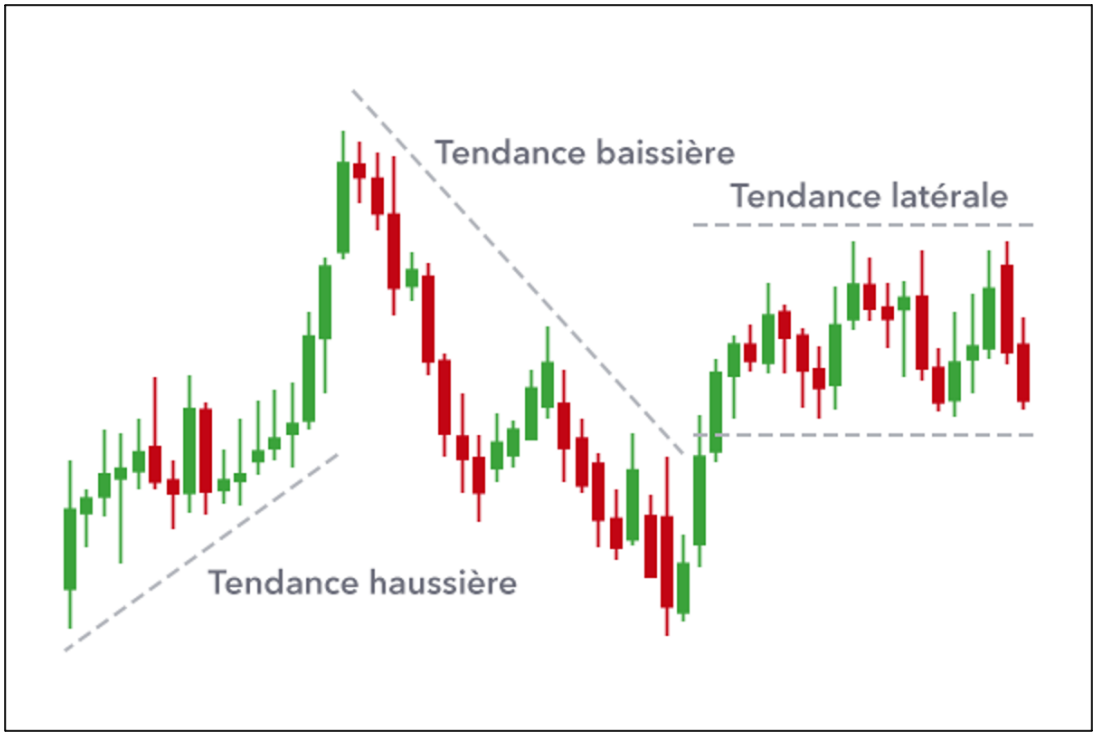


FIGURE 3 – Exemple des différentes tendances présentes dans les marchés financiers

1.4 Objectifs du projet

L'objectif principal de notre projet est d'analyser la relation entre d'un côté les flux acheteurs et vendeurs et de l'autre côté les prix associés à ces volumes afin de corréler cette relation aux tendances futures. Plus précisément, ce projet a pour objectif d'analyser les composantes du volume comme le volume total d'échanges pendant une unité de temps donnée et le rapport volume acheteur/volume vendeur durant cette même unité de temps. Cela permettra de prédire la variation du prix compte tenu des caractéristiques de l'effort, représentée par des niveaux d'engagements des acteurs acheteurs et vendeurs.

Dans ce projet, nous chercherons aussi à croiser les données de volume par unité de temps avec la distribution de ces volumes par niveau de prix. Cette vision de la distribution des volumes en fonction des prix est dérivée de la théorie des marchés d'enchères.

Cependant, pour mieux cerner les objectifs de ce projet, il faut comprendre les données, variables et indicateurs avec lesquels nous avons dû travailler.

2 Exploration et prise en main des données

Une fois que nous avons reçu un fichier avec toutes les données des contrats futures du DAX depuis 2013 échantilloné par tranche de 5 minutes, il nous a fallu découvrir et représenter l'ensemble de ces données.

2.1 Présentation de nos données

Nous avons découpé cette partie selon les types de données auxquelles nous avons fait face.

2.1.1 Variables issues des bougies

Comme mentionné précédemment, les données financières du DAX avec lesquelles nous avons travaillé contiennent de nombreuses informations prélevées toutes les 5 minutes durant les jours ouvrés de la Bourse (tous les jours du lundi au vendredi excepté les jours fériés) entre 2013 et 2022. Dans ce jeu de données, nous trouvons d'abord les différentes informations qui nous permettent de faire une représentation à l'aide de bougies (Figure 4).

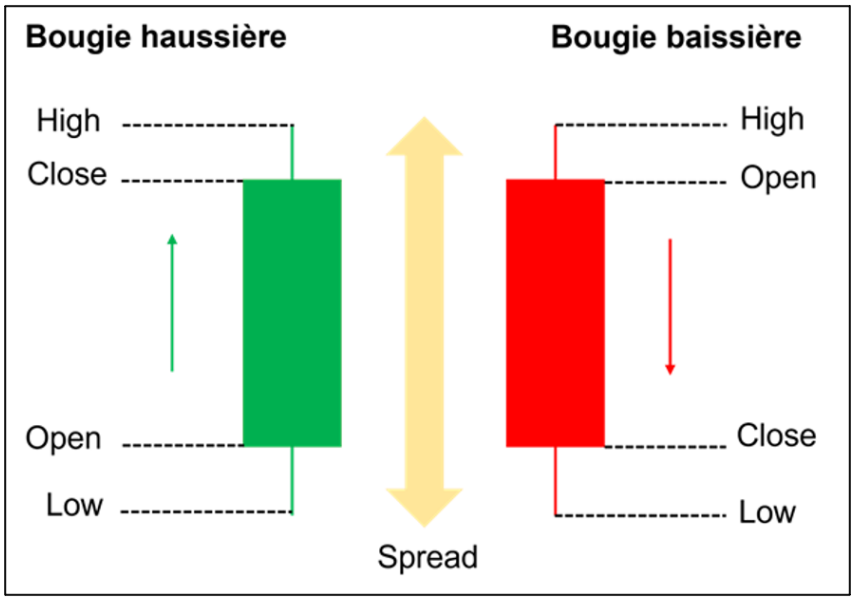


FIGURE 4 – Schéma descriptif du principe des marchés financiers

On trouve ainsi les informations suivantes :

— **High** : prix maximum durant la bougie de 5 minutes

— **Low** : prix minimum durant la bougie de 5 minutes

— **Open** : prix à l'ouverture de la bougie de 5 minutes

— **Close** (ou **Last**) : prix à la clôture la bougie de 5 minutes.

On peut ensuite définir le **Spread** comme étant la longueur de la bougie :

$$\textbf{Spread} = \textbf{High - Low}$$

2.1.2 Volumes

On trouve en plus de cela les données liées aux volumes échangés durant la bougie de 5 minutes. En effet, nous avons le **Volume** qui désigne le nombre de contrats tradés durant la bougie de 5 minutes, le **BidVolume** qui désigne le volume de contrats liés aux ventes et le **AskVolume** qui désigne le volume de contrats d'achat. Mais nous avons aussi # **of Trades** qui représente le nombre de transactions réalisées durant la bougie.

Enfin, nous avons les relations suivantes :

$$\textbf{Volume} = \textbf{AskVolume} + \textbf{BidVolume}$$

$$\textbf{Delta} = \textbf{AskVolume - BidVolume}$$

2.2 Conversion en données journalières

L'une des premières tâches que nous avons effectuée après avoir pris connaissance du jeu de données est de convertir les données des bougies de 5 minutes en données journalières.

Nous avons pour cela appliqué les formules suivantes :

— Open de la journée = Open de la bougie à 8h00 (horaire d'ouverture)

— Last de la journée = Last de la bougie à 22h00 (horaire de clôture)

— High de la journée = maximum des High de la journée

— Low de la journée = minimum des Low de la journée.

On peut alors définir le spread journalier comme la différence entre le High et le Low journalier. On définit ensuite un volume journalier comme étant la somme de tous les volumes de la journée. On fait de même pour les variables AskVolume, BidVolume et # of Trades.

L'implémentation était la suivante. Tout d'abord il a fallu récupérer l'ensemble des Open pour chaque jour, i.e. la valeur de l'Open de la bougie de 8h.

```
daily_morning = dataFrame.loc[dataFrame['Time'] == '08:00:00']
days = list(daily_morning.Date)
daily_open = list(daily_morning.Open)
```

Ensuite, il fallait prendre les Close. Or, nous avons été confronté à un premier obstacle : parfois la fin de la journée s'arrêtait un peu avant 22h (au maximum à 21h50) il a donc fallu récupérer l'ensemble des valeurs de Close pour chaque journée :

```
daily_last = []
for i in range(len(days)-1): #le dernier jour ne s'arrete pas a 22h, d'ou le -1
↪  dans le range
    df_last_minute = dataFrame.loc[(dataFrame.Time == '22:05:00') &
    ↪  (dataFrame.Date == days[i])]
    if len(df_last_minute) == 0:
        df_last_minute = dataFrame.loc[(dataFrame.Time == '22:00:00') &
        ↪  (dataFrame.Date == days[i])]
    if len(df_last_minute) == 0:
        df_last_minute = dataFrame.loc[(dataFrame.Time == '21:55:00') &
        ↪  (dataFrame.Date == days[i])]
    if len(df_last_minute) == 0:
```

```
        df_last_minute = dataFrame.loc[(dataFrame.Time == '21:50:00') &
        ↪  (dataFrame.Date == days[i])]
    cur_last = list(df_last_minute.Last)
    daily_last.append(int(cur_last[0]))

daily_last.append(Last[len(Last)-1]) #le dernier jour ne s'arrete pas a 22h
```

Puis enfin, nous avons calculé les High et Low ainsi que le Volume :

```
daily_high = []
daily_low = []
daily_volume = []

for day in days:
    df_day = dataFrame.loc[dataFrame.Date == day]

    volume_journalier = np.sum(list(df_day.Volume))
    daily_volume.append(volume_journalier)

    high_journalier = max(list(df_day.High))
    daily_high.append(high_journalier)

    low_journalier = min(list(df_day.Low))
    daily_low.append(low_journalier)
```

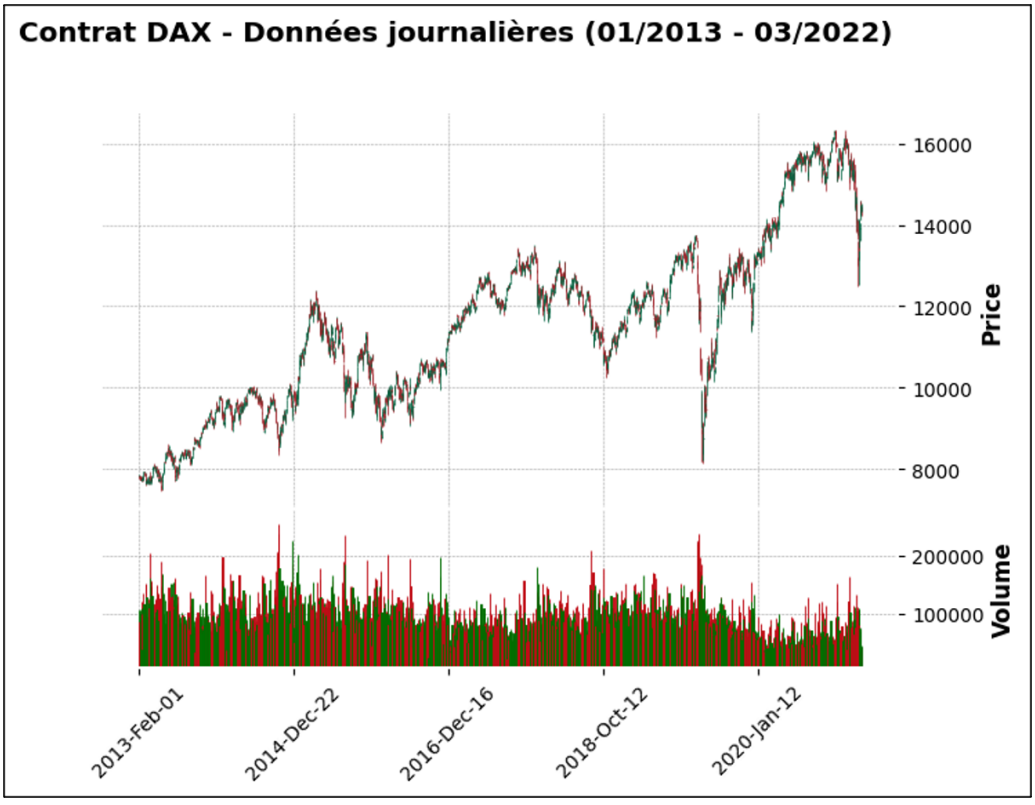


FIGURE 5 – Affichage des données journalières du DAX entre 2013 et 2022

2.3 Moyenne mobile, écart-type mobile et z-score

Outre les variables brutes obtenues directement depuis le jeu de données, il a été intéressant de calculer de nouvelles variables qui permettent de mieux comprendre l'évolution d'un cours sur la durée. De ce fait, pour prendre en compte les valeurs des jours précédents, nous avons utilisé une moyenne mobile avec nos données journalières. La moyenne mobile notée $MA_x(t)$ se calcule comme étant la moyenne d'une variable x sur une période T à l'instant t :

Définition 1 *La moyenne mobile de la variable x à l'instant t avec une profondeur de T est :*

$$MA_x(t) = \frac{x(t-T+1) + ... + x(t)}{T}$$

Dans notre cas, nous avons calculé la moyenne mobile de plusieurs variables sur différentes périodes : 7, 14 et 23 jours. De plus, nous pouvons aussi définir l'écart-type mobile :

Définition 2 *L'écart-type mobile de la variable x à l'instant t avec une profondeur de T est :*

$$\sigma_x(t) = \sqrt{\frac{x(t-T+1)^2 + ... + x(t)^2}{T} - MA_x(t)^2}$$

Enfin, le z-score, aussi appelé standard score, est une proportion du nombre d'écarts-types en-dessous ou au-dessus de la moyenne. Il se calcule de la manière suivante :

Définition 3 *Le Z-Score de la variable X est :*

$$Z_X = \frac{X - \mu}{\sigma}$$

où :
- Z_X est le z-score
- μ est la moyenne du Close des données journalières
- σ est l'écart-type du Close des données journalières

De même on peut définir un Z-Score mobile :

$$Z_X = \frac{X - MA_X(t)}{\sigma_X(t)}$$

Ces 3 variables nous ont été très utiles pour faire une classification du Spread, ce qui sera expliqué en profondeur ultérieurement.

NB : À noter que l'implémentation Python de ces calculs est triviale donc ne nécessite pas d'exemple ici.

2.4 Normalisation des données

Maintenant que nous avons pris en main les différentes variables, nous nous sommes tournés vers la normalisation.

2.4.1 Pourquoi normaliser ?

Étant donné que le taux d'inflation a progressivement augmenté durant les années 2013 à 2022, nous avons cherché à réduire son influence sur nos données en les normalisant de différentes manières afin de pouvoir comparer quelle normalisation serait la plus appropriée.

De plus, nous verrons dans les pages qui suivent que lors d'une utilisation avec des algorithmes de prédiction, avoir des données normalisées permet de faciliter les calculs et de ne plus avoir des valeurs qui croissent exponentiellement.

2.4.2 Exponential Smoothing

La première approche que nous avons utilisée est l'exponential smoothing [2]. Cette méthode permet de lisser et prévoir des données d'une série temporelle. L'avantage de cette méthode est qu'elle prend en compte les données passées pour prédire les données futures, tout en donnant des poids décroissants exponentiellement avec leur ancienneté par rapport à la donnée à calculer. En notant X_t nos données et $s(t)$ la sortie de l'algorithme exponential smoothing ($\forall t \geq 0$) :

Définition 4 *On a :*

$$\begin{cases} s_0 = X_0, t = 0 \\ s_t = \alpha X_t + (1-\alpha)s_{t-1}, \forall t > 0, \end{cases}$$

où $\alpha \in [0, 1]$ *est le facteur de lissage*

Nous avons déterminé α de manière empirique : pour chaque facteur que nous testions, nous calculons un score d'erreur, avec la formule suivante :

$$erreur_\alpha = \sum_{i=1}^{N} |s_t - X_t|$$

Nous avons alors choisi le facteur α qui minimisait ce score d'erreur. Nous avons trouvé $\alpha = 0.963$.

Comme $s(t)$ peut être considéré comme la meilleure estimation de la prochaine valeur de X, nous avons opté pour la normalisation suivante. Pour tout $t \in [1, N]$, où N est la taille du jeu de données :

— prédire une donnée $X_{predite,t}$ par exponential smoothing

— constuire la nouvelle donnée $X_{normalisee,t} = X_t - X_{predite,t}$

Nous avons donc utilisé cette méthode pour normaliser les données du High, Low, Open et Close. Le problème que nous avons rencontré avec cette approche était dû au fait que nous normalisions les données séparément. Ainsi, dans certains cas, la donnée de High normalisée peut se trouver en-dessous de la donnée normalisée du Low.

On peut voir sur le schéma (page suivante) que les données ne sont pas cohérentes : les bougies ne sont pas toutes continues.

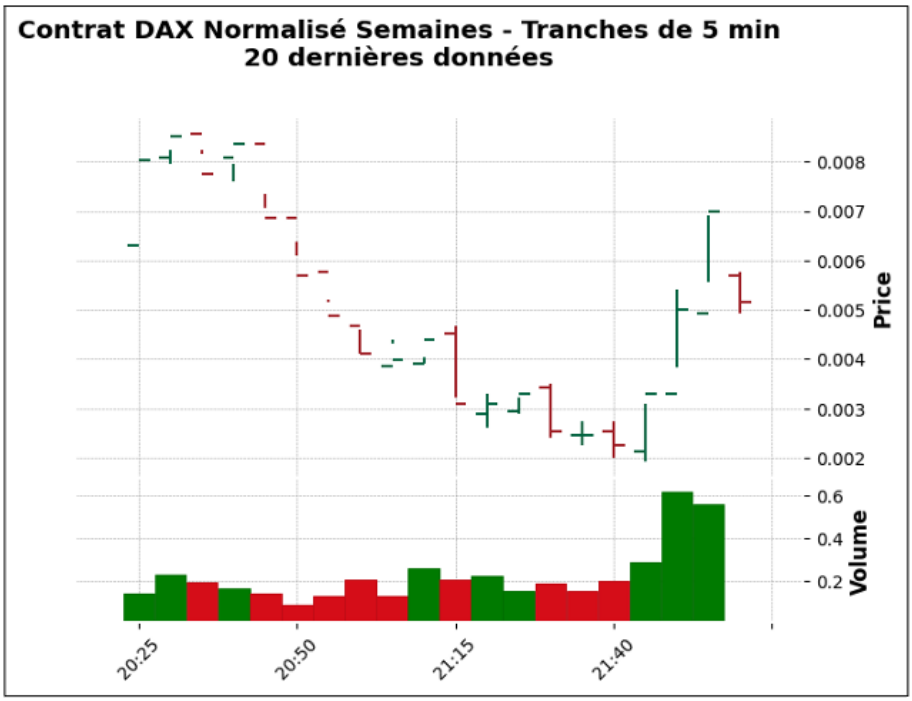


FIGURE 6 – Normalisation tranche par tranche avec exponential smoothing

2.4.3 Normalisation semaines par semaines et mois par mois

Nous avons donc tenté d'effectuer une normalisation des données d'une manière différente. Nous avons dans un premier temps calculé la moyenne du Close sur chaque mois puis nous avons divisé le High, le Low, l'Open et le Close de chaque bougie par cette moyenne. Pour les volumes, nous les avons simplement normalisés en calculant la moyenne des volumes sur toute la période puis en divisant les volumes de chaque bougie par cette moyenne.

De même, nous avons normalisé nos données en calculant la moyenne du Close sur une ou plusieurs semaines précédentes, avec une profondeur de 14 pour la moyenne mobile. Les normalisations sont les suivantes :

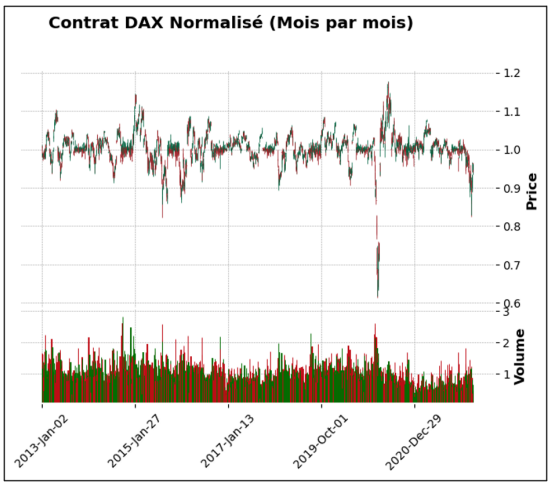


FIGURE 7 – Normalisation - mois

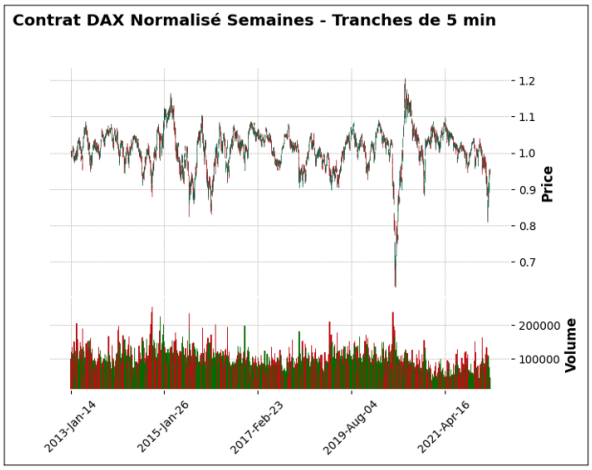


FIGURE 8 – Normalisation - semaines

On a une normalisation qui conserve une allure proche de celle obtenue par la méthode de l'exponential smoothing.

De plus, si on s'intéresse aux 20 dernières données normalisées, on peut remarquer que le problème que nous avions rencontré avec l'exponential smoothing, à savoir le manque de cohérence des données normalisées, n'apparaît plus.

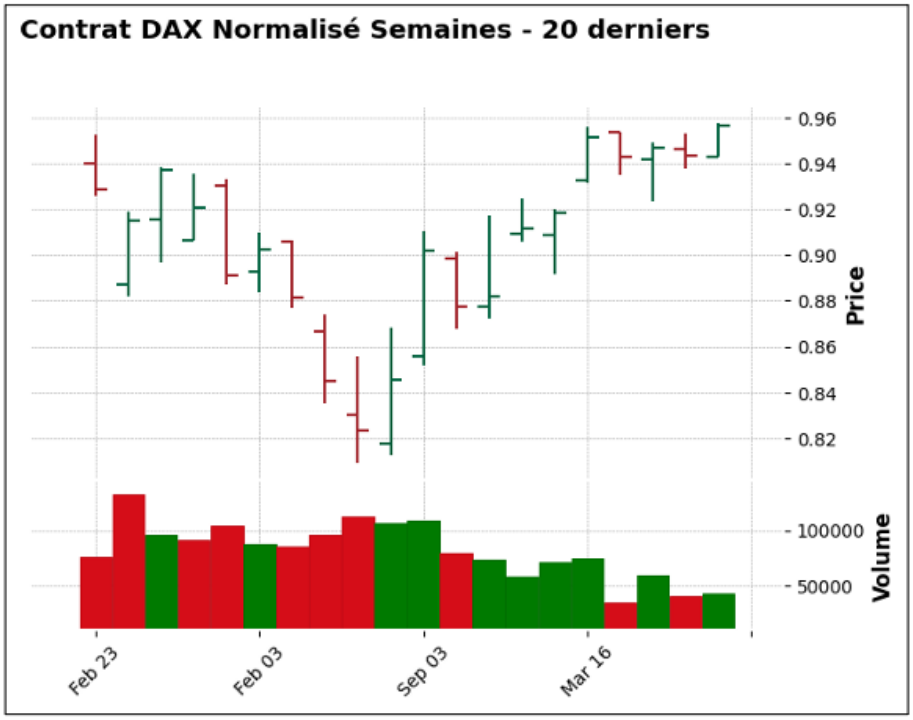


FIGURE 9 – Normalisation - semaines (20 dernières données)

2.4.4 Choix de la méthode de normalisation adaptée

Nous avons mis en place différentes méthodes de normalisation dans la partie précédente. Ainsi, il nous reste à déterminer quelle méthode est la plus adaptée à notre projet, afin de sélectionner les données normalisées qui soient optimales pour nous. Pour cela, nous avons mis en place un réseau de neurones assez simple, ayant pour but de prédire les données de High, Low, Open et Close à partir des données du passé. Nous savions que ce modèle n'aurait pas une bonne précision et ne serait pas capable de prédire précisément les données que nous lui demandions. Cependant, l'idée de notre étude était de comparer les résultats de ce modèle avec les données normalisées de différentes manières.

La première étape était de sélectionner la profondeur de la moyenne mobile à utiliser pour la normalisation semaine par semaine.

Pour cela, nous avons entraîné notre réseau de neurones sur une normalisation faite pour une profondeur variant entre 1 et 30. Puis, on s'est intéressé aux erreurs faites par la prédiction du modèle, notamment la MSE (Mean Squared Error) et la MAE (Mean Absolute Error).

Définition 5 *Soit X notre vecteur représentant les données de test, on a :*

$$MSE = \mathbb{E}\left[(X - X_{prediction})^2\right]$$

$$MAE = \mathbb{E}\left[|X - X_{prediction}|\right]$$

Ainsi, on cherche à minimiser les valeurs de MAE et MSE. Voici les résultats que nous obtenons :

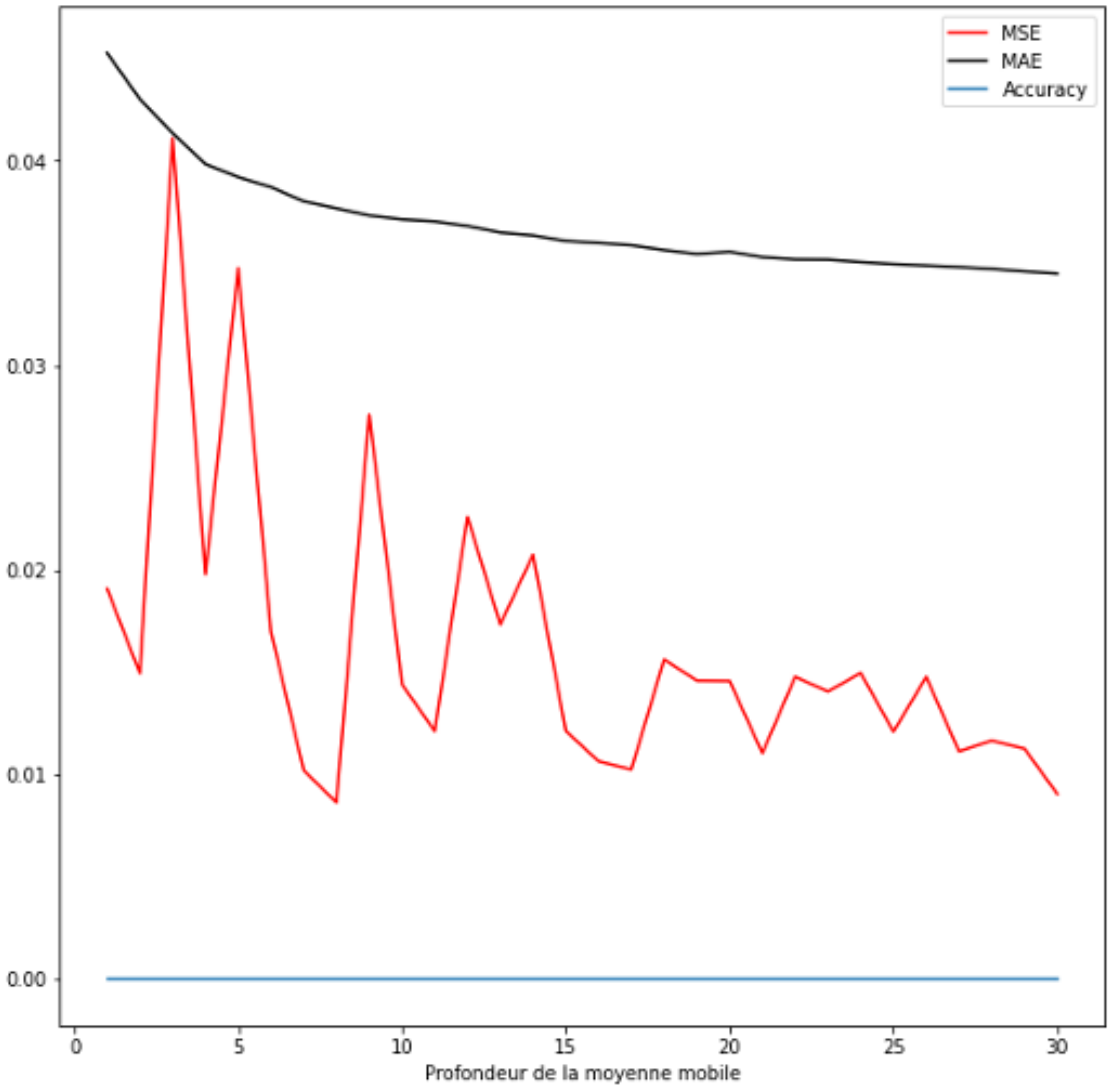


FIGURE 10 – Évolution des métriques en fonction de la profondeur de la moyenne mobile de la normalisation

On remarque effectivement que plus la profondeur de la moyenne mobile augmente, plus les erreurs diminuent. Cependant, on peut remarquer que les erreurs ne diminuent plus beaucoup à partir d'une profondeur égale à 14. Ainsi, c'est celle que nous choisirons dans la suite de notre étude de comparaison des normalisations.

On s'intéresse maintenant à comparer les performances des différents types de normalisation. On compare ainsi la normalisation semaine par semaine, avec une profondeur de 14, la

normalisation par mois et une normalisation standard appelée “normalisation min-max”.Nous choisissons d’ajouter cette dernière à notre étude afin de prouver que la normalisation que nous avons effectuée améliore les performances par rapport à une normalisation plus standard.

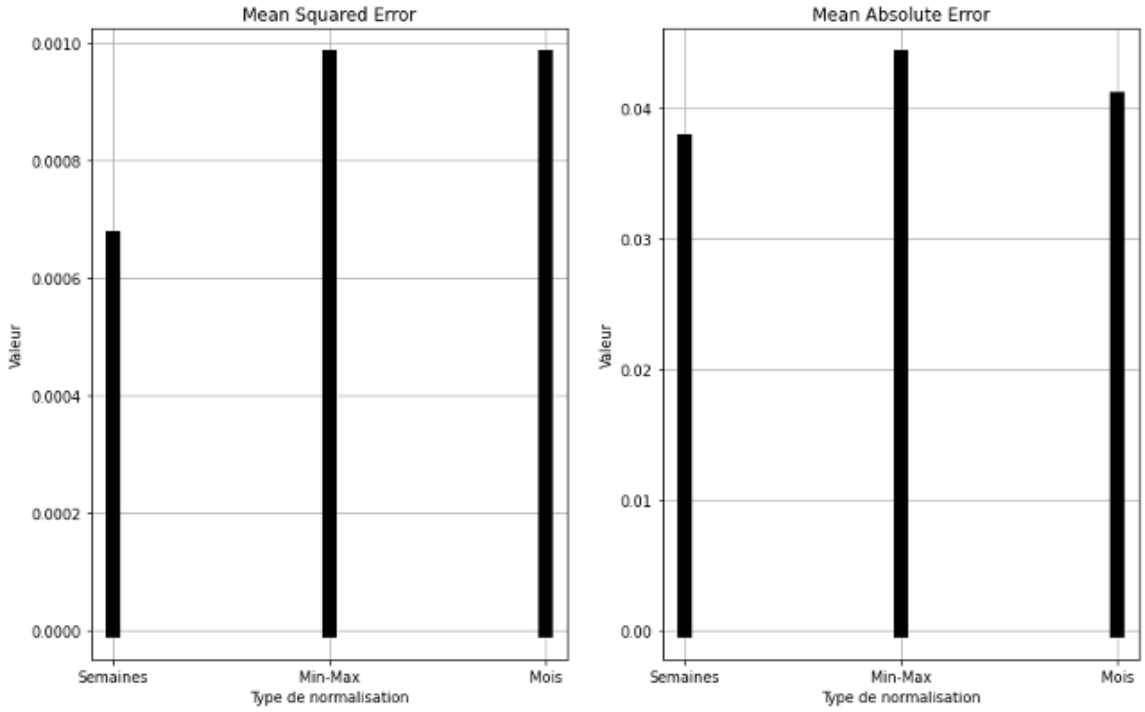


FIGURE 11 – Comparaison des indices métriques

On remarque bien que nos normalisations, que ce soit celle par semaine ou celle par mois, améliorent les performances du réseau de neurones que nous avons implémenté. De plus, si on les compare entre elles, on voit clairement que la normalisation par semaine parvient à obtenir des scores d’erreurs plus faibles que la normalisation par mois. Ainsi, dans la suite de notre projet, nous choisirons la normalisation par semaine, avec une profondeur de 14 pour la moyenne mobile, lorsque nous aurons besoin de travailler sur des données normalisées.

2.5 Recherches statistiques autour de différentes variables

Une fois la normalisation effectuée, comme nous pouvions comparer les données sans fausser nos résultats, nous avons cherché à comprendre les liens entre elles et leur évolution. Pour ce faire, nous avons fait quelques recherches statistiques autour de nos données.

2.5.1 Recherche des gaps

On parle de gap lorsque le cours d’ouverture de la séance est situé en dehors de la bougie du jour précédent. On distingue alors deux types de gap :

— les gaps vers le haut pour lequel l’Open de la journée se situe au dessus du High de la journée précédente

— les gaps vers le bas pour lequel l’Open de la journée se situe en dessous du Low de la journée précédente.

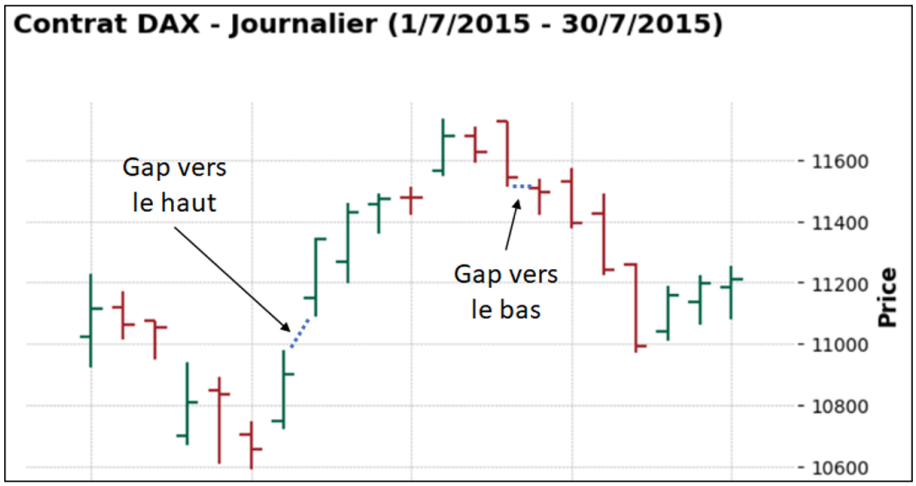


FIGURE 12 – Affichage de Gap vers le haut et Gap vers le bas sur le cours du DAX

Ainsi, sur la période entre janvier 2013 et mars 2022 (2333 jours ouvrés), nous trouvons :

— 346 gaps vers le haut

— 208 gaps vers le bas

De plus , nous pouvons aussi nous intéresser à quel jour de la semaine les gaps sont les plus fréquents.

Nous remarquons ainsi que les gaps sont un peu plus fréquents le lundi que les autres jours de la semaine. Cela est plutôt logique étant donné que les prix sont plus susceptibles d'évoluer entre le vendredi et le lundi.

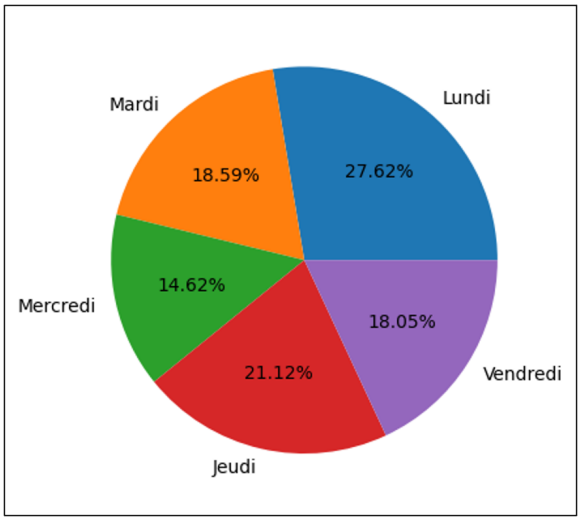


FIGURE 13 – Répartition des gaps selon les jours de la semaine

2.5.2 Relations entre le High de l'Initial Balance et celui de la journée

On appelle Initial Balance ce qu'il se passe sur le marché entre son ouverture à 8h du matin jusqu'à 9h du matin. Pendant ce moment, le marché peut beaucoup varier et peut, dans certains cas, donner des indications sur la journée à venir. Nous nous sommes notamment intéressés à la relation entre le maximum atteint lors de l'Initial Balance (High de l'Initial Balance) et le High de la journée, ainsi que celle entre le Low de l'Initial Balance et le Low de la journée. Nous voulions déterminer, avec une certaine tolérance, combien de fois nous pouvions observer que le High (respectivement Low) de l'Initial Balance était égal au High (respectivement Low) de la journée.

Cependant, quelle tolérance choisir ? Par exemple, si on prend 10 points de tolérance, cela n'a pas le même impact sur les données de 2013 et celles de 2021-2022 à cause de l'inflation. Ainsi, nous avons travaillé avec les données normalisées afin de contourner ce problème.

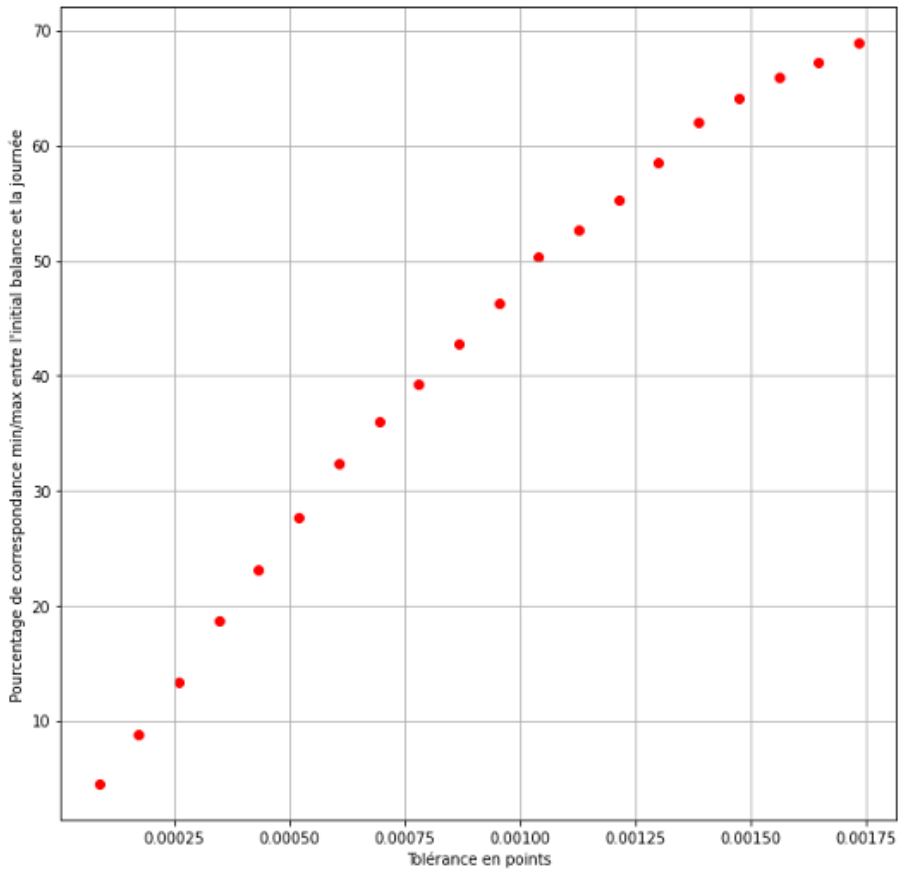


FIGURE 14 – Correspondance - High/Low - Initial Balance / Journée

Sans grande surprise, plus la tolérance est grande, plus les correspondances sont nombreuses. Cependant, on sait que les marchés se comportent différemment en fonction du jour de la semaine ; ainsi, on va faire la même étude mais en différenciant les jours de la semaine.

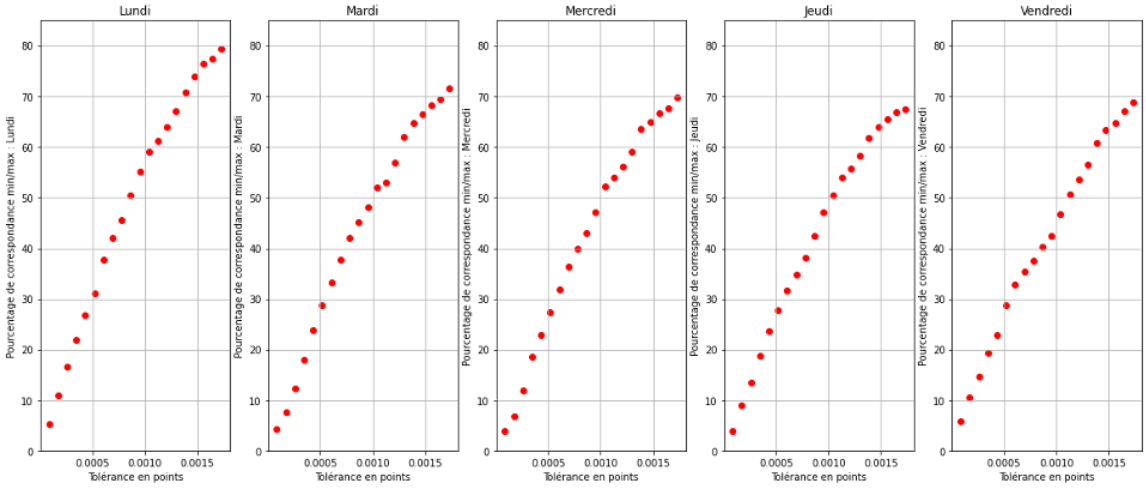


FIGURE 15 – Correspondance - High/Low - Initial Balance / Journée - par jour

On remarque effectivement que les résultats sont plus prometteurs le lundi. En effet, le marché enregistre de fortes actions tôt le lundi en réponse au week-end, puis se stabilise assez vite. Cette étude montre un biais statistique qui pourrait donner un avantage intéressant sur le marché. Une étude plus poussée de ce biais pourrait être intéressante à effectuer, et pourrait révéler de nouveaux biais.

Pour étudier la correspondance nous avons créé deux fonctions qui permettent pour chaque jour de comparer le High et le Low de l'Initial Balance avec celui de la journée (en disjonction de cas), de les trier puis de décompter sans doublons. En voici l'implémentation :

```python
def correspondances_opening_day_spread(dataF, dataFMorning, tolerance):
    jours_max_correspondants = []
    jours_min_correspondants = []

    for day in days:
        dataFrame_day = dataF.loc[dataF.Date == day]
        dataFrame_morning_this_day = dataFMorning.loc[dataFMorning.Date == day]

        if len(dataFrame_day) > 0 and len(dataFrame_morning_this_day) > 0:
            if abs(float(dataFrame_day.High) -
            ↪  float(dataFrame_morning_this_day['High'])) <= tolerance:
                jours_max_correspondants.append(day)

            if abs(float(dataFrame_day.Low) -
            ↪  float(dataFrame_morning_this_day['Low'])) <= tolerance:
                jours_min_correspondants.append(day)

    return jours_max_correspondants, jours_min_correspondants
```

```python
def compter_jours_correspondance(dataF, dataFMorning, tolerance):
    jours_max, jours_min = correspondances_opening_day_spread(dataF,
    ↪  dataFMorning, tolerance)
    nb__correspondances = len(jours_max)
    for jour in jours_min:
        if not jour in jours_max:
            nb__correspondances += 1
    return nb__correspondances
```

3 Classification

Une fois que nous avions fait nos explorations statistiques, nous sommes rentrés dans le vif du sujet. Pour cerner notre objectif principal, il faut introduire la notion de classification (ou état variable).

3.1 Classification Z-Score

La classification z-score d'une variable x donnée (ou l'état variable) est le résultat de la discrétisation de cette dernière. Dans la suite, à chaque fois que nous parlerons de classification, nous faisons référence à la classification z-score. Dans notre projet, nous avons étudié la classification du spread journalier.

La première classification utilisée lors de ce projet nous a été fournie par Laurent Abril. Elle comporte 5 modalités : -2, -1, 0, 1 et 2. Ces modalités sont basées sur deux valeurs α_1 et α_2 qui ont été trouvées empiriquement dans la première classification ($\alpha_1 = 0.4$ et $\alpha_2 = 0.7$). La classification se fait de la manière suivante :

Classes	Conditions
-2	$spread < moy\_mobile$ - $\alpha_2 \times var\_mobile$
-1	$spread < moy\_mobile$ - $\alpha_1 \times var\_mobile$
0	$spread > moy\_mobile$ - $\alpha_1 \times var\_mobile$ & $spread < moy\_mobile + \alpha_1 \times var\_mobile$
1	$spread > moy\_mobile + \alpha_1 \times var\_mobile$
2	$spread > moy\_mobile + \alpha_2 \times var\_mobile$

3.2 Intérêt

Cette classification du spread a 2 intérêts majeurs. D'une part elle permet de comprendre l'état de la journée par rapport aux journées précédentes et donc nous donne une indication quant à sa tendance. Nous sommes à présent en mesure de quantifier cette tendance, lorsqu'elle est haussière ou baissière. D'autre part, avoir cette information à l'avance permettrait de se positionner plus tôt sur le marché. Un algorithme de trading va se reposer sur des indicateurs qui, une fois leurs valeurs seuils dépassées, attendent une confirmation de l'état du marché avant de se positionner. Donc si l'on est en mesure de prédire à l'avance cette classification, l'algorithme peut se positionner plus tôt et donc faire plus de bénéfice.

Se pose alors une question : comment faire pour prédire cette classification le plus tôt possible ?

4 Prédiction

Il est donc clair que nous avons des données avec de nombreux paramètres tout aussi importants les uns que les autres au premier abord. Notre objectif était de prédire la classification du Spread journalier vue précédemment. Nous commencerons par expliquer ce qu'est l'apprentissage supervisé, avant de mettre en avant les obstacles liés au contexte et comment nous avons fait pour les contourner pour enfin exposer les résultats de nos réseaux de neurones sur le jeu de données.

4.1 Apprentissage supervisé

L'apprentissage supervisé est une méthode particulière de Machine Learning. Le principe est simple : déterminer une fonction de prédiction à partir d'exemples annotés.

Définition 6 *L'objectif de l'apprentissage supervisé est d'apprendre une fonction f qui minimise l'écart entre $f(X) = Y$ où X et Y sont deux variables aléatoires*

Dans notre cas, X serait constituée de données sélectionnées parmi celles du tape et parmi les variables créées et Y serait la classification du spread.

On trouve deux types de sous méthodes d'apprentissage supervisé : la régression et la classification. Dans notre cas, comme nous voulons prédire une classification, il est évident, comme son nom l'indique, que nous nous sommes dirigés vers la deuxième méthode.

En reprenant l'exemple précédent, l'apprentissage supervisé pour la classification permet de trouver la fonction f où la sortie Y est un ensemble de classe. Dans notre cas, le vecteur Y a une dimension de 5 qui correspondent aux 5 modalités possibles de classification. Par exemple, pour chaque entrée qui serait fournie, la sortie serait la classe associée (et non une variable mathématique).

Pour faire de la classification nous nous sommes dirigés vers les réseaux de neurones car c'est une technique sophistiquée d'apprentissage supervisé capable de modéliser des relations extrêmement complexes.

4.2 Réseau de neurones artificiel

L'architecture d'un tel système est inspirée des systèmes de neurones humains. Pour comprendre le fonctionnement d'une telle structure, il faut d'abord comprendre le fonctionnement d'un neurone i.e d'un perceptron.

4.2.1 Perceptron

Un perceptron [3] est l'architecture de réseau de neurones la plus simple car c'est un modèle à 1 neurone. Ce modèle prend en entrée un certain nombre de variables qui sont pondérées par des poids et en sortie on trouve soit 0 soit 1 (sortie booléenne).

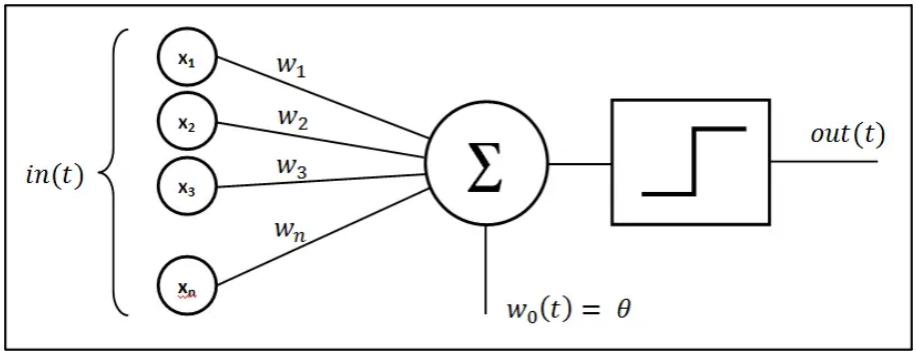


FIGURE 16 – Schéma descriptif d'un perceptron

Définition 7 *Un perceptron à n entrées $z = (x_1, ..., x_n)$ et à une seule sortie o est défini par la donnée de n poids $(w_1, ...w_n)$ et un biais (ou seuil) θ par :*

$$o = \sigma(z) = \begin{cases} 1 & si \quad \sum_{i=1}^{n} w_i x_i > \theta \\ 0 & sinon \end{cases}$$

σ est dite fonction d'activation

Ici σ est une fonction assez simple mais on retrouve parfois de nombreuses fonctions d'activation dans lesquelles une non-linéarité est introduite. Dans les réseaux de neurones, on retrouve souvent les suivantes :

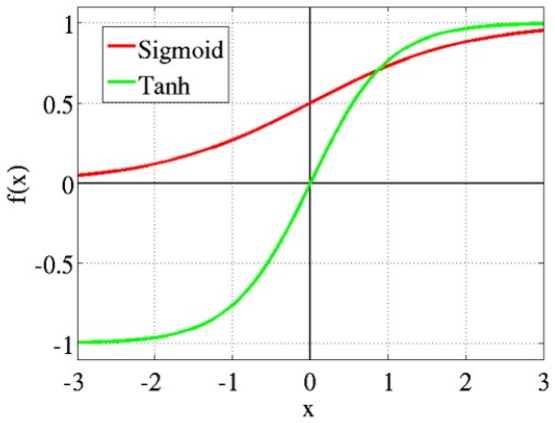
Fonction sigmoïde et tangente hyperbolique :

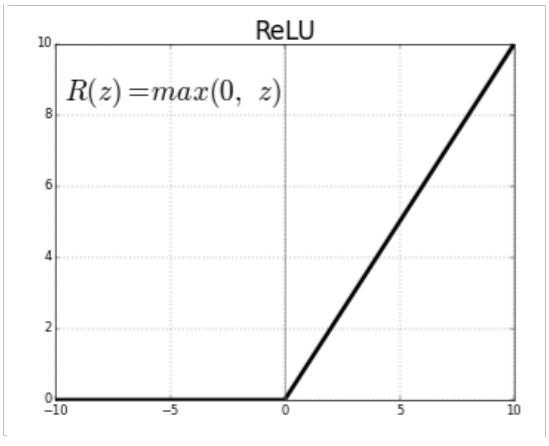
$$x \mapsto \frac{1}{1 + e^{-x}}$$

$$x \mapsto tanh(x)$$

Fonction ReLU :

$$x \mapsto \begin{cases} 0 & si \quad x < 0 \\ x & si \quad x \geq 0 \end{cases}$$





Un seul neurone est cependant trop simple pour un modèle qui doit être capable de prédire une structure de données aussi complexe. De ce fait, nous avons été amenés à considérer des perceptrons multi-couches.

4.2.2 Multi-layer Perceptron

C'est avec cette architecture ici que nous retrouvons les similitudes avec un réseau de neurones biologiques. On peut observer plusieurs couches (les lignes verticales de neurones). Et chaque neurone d'une couche est connecté à tous les neurones de la couche suivante.

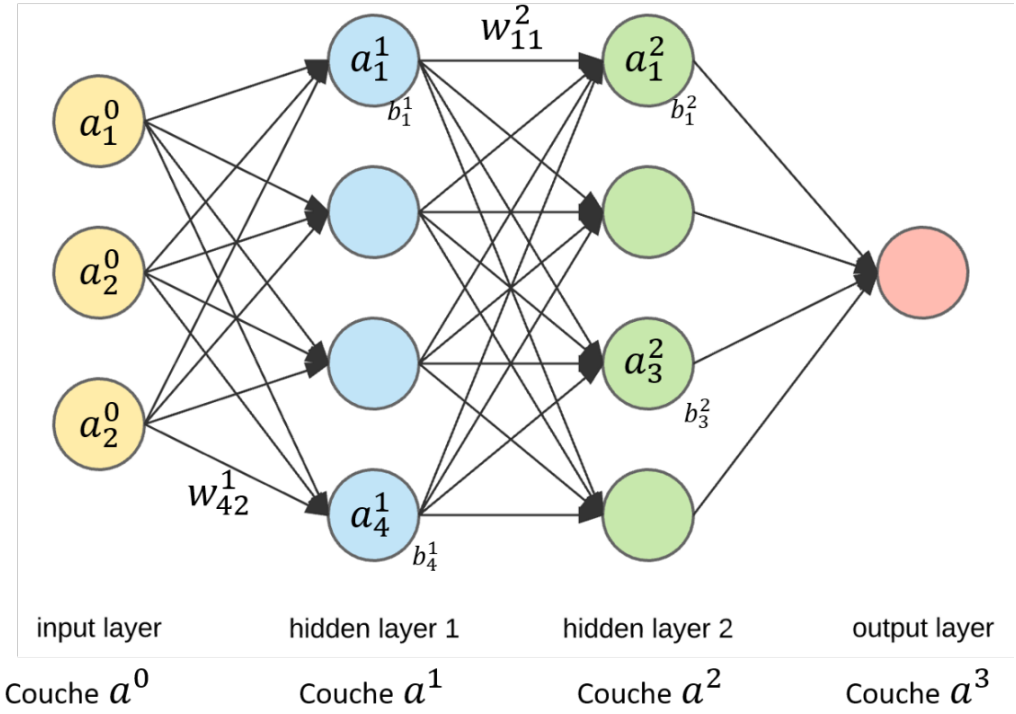


FIGURE 17 – Schéma descriptif d'un perceptron multi-couches

Nous avions vu le cas pour un neurone, ici on se ramène à un cas où la dimension augmente. De ce fait, la sortie o s'exprime de manière similaire :

$$o = \sigma(z) = WX + B$$

avec

$$z = (x_1, ..., x_n), \quad X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad W = \begin{bmatrix} w_{11} & \dots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \dots & w_{nn} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} \\ \vdots \\ b_{n1} \end{bmatrix}$$

où W est la matrice des poids et B la matrice des biais.

L'objectif est à nouveau d'apprendre la fonction σ : l'algorithme qui fait cela s'appelle l'algorithme de propagation (car l'on passe d'une couche de neurones à une autre). L'apprentissage de cette focntion repose sur 2 notions : la fonction de coût et la descente du gradient.

La fonction de coût (ou fonction de perte) est une fonction qui permet de quantifier l'écart entre des valeurs prédites et les valeurs réelles. Des fonctions connues sont le MSE et MAE (que nous avons introduits précédemment). Dans le cas d'une classification multi-classes, une fonction de coût qui est utilisée est la fonction d'entropie croisée (cross-entropy loss function [4]).

Définition 8 *La fonction de cross-entropy est définie par :*

$$(x_1, ..., x_n) \mapsto \sum_{i=1}^{n} x_i log(x_i)$$

Nous reviendrons plus tard sur cette fonction.

Une fois que l'erreur entre $o = \sigma(z)$ et y est calculée, il faut effectuer des modifications pour minimiser cette dernière. C'est là que l'algorithme de descente du gradient intervient. Le principe est assez simple :

Définition 9 *Soit θ un paramètre de la fonction σ, alors la modification à l'instant t de θ est donnée par :*

$$\theta_{t+1} = \theta_t - \alpha \nabla C$$

avec α le taux d'apprentissage et C la fonction de coût

Rappel : $\nabla C = \frac{\partial C}{\partial \theta_i}$

On va chercher à minimiser chacun des paramètres de la fonction σ pour que l'on trouve tous les θ_i tels que :

$$C_{\theta_1,...,\theta_n}(\sigma(z), y) = \min_{\lambda_1,...,\lambda_n} C_{\lambda_1,...,\lambda_n}(\sigma(z), y)$$

En réitérant les modifications des θ_i, $C_{\theta_1,...,\theta_n}$ converge vers le résultat ci-dessus.

Tout cela nous donne donc le fonctionnement de base d'un réseau de neurones à plusieurs couches à propagation avant. Or, comme nous devons prédire une classification à partir de nombreuses données d'entrées et non pas un unique point, nous avons dû introduire le LSTM.

4.2.3 LSTM

Un LSTM (Long-Short Term Memory) est un type de réseau de neurones avec rétropropagation du gradient. Sans rentrer dans trop de détails, un neurone d'un LSTM peut traiter les données de manière séquentielle et conserver un état caché dans le temps. En gardant en mémoire des informations sur les données, ce type de réseau est efficace pour faire des prédictions en se basant sur des séries temporelles.

En Python 3, nous n'avons pas eu besoin de programmer entièrement ce système car il existe une API de la bibliothèque *Tensorflow* nommée *keras* qui permet d'implémenter de nombreux algorithmes d'apprentissage supervisé. Nous verrons cette implémentation ultérieurement.

Maintenant que nous avons vu en quoi consistait un réseau de neurones et plus précisément comment fonctionnait un LSTM, il a fallu réfléchir sur les variables à mettre en argument de notre réseau de neurones et l'architecture à choisir.

4.2.4 ACP et Entraînements

Pour faciliter l'utilisation des données et pour réduire la quantité de calcul avec des algorithmes tels que ceux utilisés en Machine Learning, nous avons effectué des normalisations. Cependant, en sachant que notre base de données comporte plus de 20 variables, la question de

la dimensionnalité des données que nous allons utiliser pour la prédiction de la classification se pose. Pour ce faire nous avons réalisé une ACP.

Définition 10 *L'analyse en composantes principales est une technique multivariée qui permet d'analyser la structure statistique d'observations dépendantes de grande dimension en représentant les données à l'aide de variables orthogonales appelées composantes principales.*

Soit (X_i) $1 \leq i \leq n$, n variables aléatoires dans $\mathbb{R}^d$. On pose X la matrice dans $\mathbb{R}^{n \times d}$ telle que la ligne i corresponde à la transposée de X_i. On suppose de plus, quitte à les soustraire par leur espérance, que les variables X_i sont centrées. On cherche à réduire la dimension des variables X_i, à l'aide d'une matrice de compression qu'on note W. On cherche alors à calculer :

$$(U_*, W_*) \in \underset{(U,W) \in \mathbb{R}^{d \times p} \times \mathbb{R}^{p \times d}}{\arg\max} \sum_{i=1}^{n} ||X_i - UWX_i||^2$$

On pose la matrice de covariance empirique :

$$\Sigma_n = \frac{1}{n} \sum_{i=1}^{n} X_i X_i^T$$

Cette dernière nous permet de simplifier le problème à la recherche de :

$$U_* \in \underset{U \in \mathbb{R}^{d \times p}, U^T U = I}{\arg\max} trace(U^T \Sigma_n U)$$

On a alors $W_* = U_*^T$. Ainsi, pour tout X_i, $1 \leq i \leq n$, la donnée réduite nous est donnée par : $W_* X_i$.

En utilisant l'implémentation donnée par *keras*, nous avons :

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

df2 = pd.read_table(r"./Data/DailyData_VolumeProfile.csv", sep = ";", engine =
↪  'python')

target_cols = ['Open', 'High', 'Low', 'Last', 'Volume', '# of Trades', 'OHLC
↪  Avg', 'HLC Avg', 'HL Avg', 'Bid Volume', 'Ask Volume', 'priceChange',
↪  'delta', 'spread', 'IBH', 'IBL', 'IB_Spread', 'VAH', 'VAL']

scaling=StandardScaler()
scaling.fit(df2[target_cols])
Scaled_data=scaling.transform(df2[target_cols])

principal=PCA(n_components=19)

principal.fit(Scaled_data)
x=principal.transform(Scaled_data)
```

En effectuant une ACP sur les 19 variables que l'on voit dans l'implémentation, on s'aperçoit qu'il n'y a plus de progression de l'inertie cumulée à partir de 5 dimensions et qu'à partir de 3 dimensions on dépasse déjà les 90%. De ce fait, on peut penser qu'avec une dimension de taille 3 nous aurons assez d'éléments pour faire notre prédiction.

Maintenant que nous avons notre dimension, nous avons choisi quelles variables utiliser pour effectuer cette prédiction. Nous avons opté pour : spread journalier, spread de l'initial balance et le delta. Ces variables donnent des indications sur l'évolution du spread et des volumes d'où leur intéret pour notre prédiction.

Maintenant que nous avons nos données à mettre en entrée, il a fallu choisir la bonne architecture. Nous avons donc entrainé plusieurs réseaux sur l'ensemble des données avec différentes architectures. Le prermier problème auquel nous avons été confronté a été la sur-représentation de la classe 0. En effet, nous avions les effectifs suivants :

— Classe -2 : 338
— Classe -1 : 295
— Classe 0 : 1176
— Classe 1 : 123
— Classe 2 : 405

On remarque très facilement que la classe 0 est donc dominante : la majorité des journées depuis 2013 est donc classée dans la classe 0. Pour permettre un bon apprentissage, nous avons procédé à un échantillonnage : nous avons considéré 100 éléments que nous avons pris aléatoirement dans chacune des classes. On a donc un ensemble de données d'entrainement contenant 500 entrées.

Enfin, pour un entrainement plus performant nous fournissons en entrée de notre modèle un vecteur avec des données sur 7 jours consécutifs pour les 3 variables sélectionnées : Une entrée i de notre échantillon d'entrainement est donc de la forme 3 x 7 : le spread journalier, le spread de l'initial balance et le delta pour les jours i-7 jusqu'à i-1 (on regarde les 7 journées qui précèdent la journée i). La classification associée à l'entrée i est la classification du spread de la journée i (on veut donc prédire la classification d'une journée avec uniquement les données des jours précédents).

Nous avons commencé par comparer les nombres de neurones sur un réseau assez complexe. L'implémentation était la suivante :

```
model = tf.keras.models.Sequential()
model.add(Embedding(5741, 21))
model.add(Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2,
↪  activation='tanh')))
model.add(Dense(15, activation='relu'))
model.add(Dense(5, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
↪  metrics=['accuracy'])
```

On remarque que nous avons une couche LSTM avec un nombre de neurones indiqué et la dimension des données en entrée. La dernière couche *Dense* permet de nous ramener à notre classification à 5 modalités. De plus, les fonctions d'activation choisies sont tanh, ReLU et softmax ($(z_1, ..., z_n) \mapsto \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}}$ pour tout $j \in [1 : n]$), qui est utilisé pour la classification à n-classes). Enfin, nous n'avons pas utilisé une descente du gradient classique mais l'algorithme Adam [5] qui est plus performant. Les résultats obtenus sont les suivants :

On voit donc que pour une couche le modèle à 64 neurones nous donne de meilleurs résultats. C'est d'ailleurs corroboré par les matrices de confusion :

Neurones	Accuracy	Loss	MSE
16	0.9716	0.0189	0.0202
32	0.9770	0.0165	0.249
64	0.9824	0.016	0.004
128	0.9788	0.0143	0.44

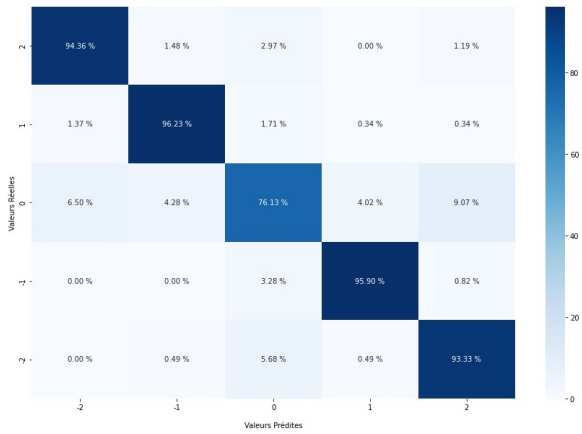


FIGURE 18 – 16 neurones

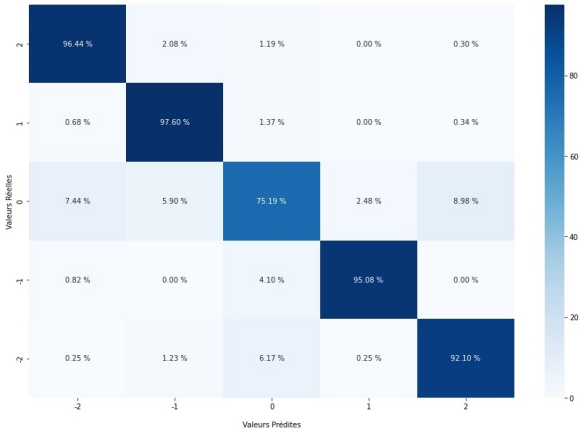


FIGURE 19 – 32 neurones

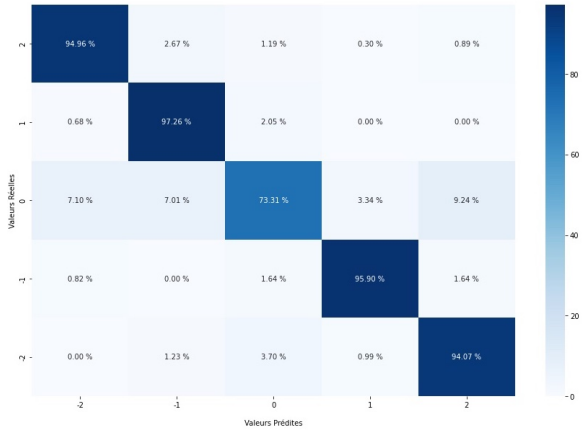


FIGURE 20 – 64 neurones

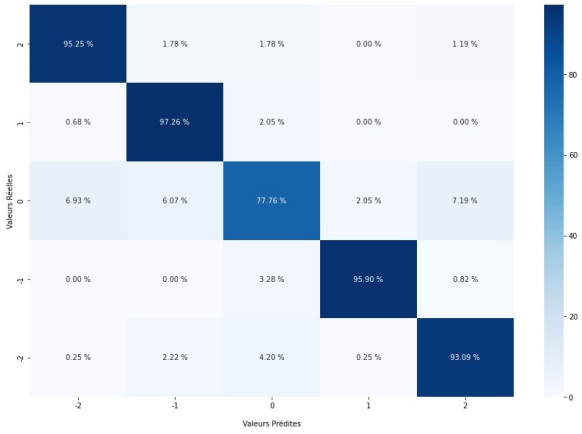


FIGURE 21 – 128 neurones

Les matrices de confusion nous permettent d'éliminer le modèle à 16 neurones qui n'est pas à la hauteur des autres modèles. Par la suite, nous avons voulu évaluer le nombre de couches nécessaire, donc nous avons testé des modèles avec 2 et 3 couches en regardant donc que les modèles à 32, 64 et 128 neurones.

Couches	Nombre de neurones	Accuracy	Loss	MSE
2	32	0.9374	0.038	0.009
	64	0.9598	0.027	0.003
	128	0.9786	0.015	0.027
3	32	0.8840	0.066	0.05
	64	0.9361	0.042	0.028
	128	0.9574	0.276	0.08

Les résultats confirment bien que le meilleur modèle est celui à 64 neurones (peu importe le nombre de couche). Par ailleurs les 2 meilleurs modèles semblent être les plus simples i.e. ceux avec 1 ou 2 couches. Il était donc logique de comparer les matrices de confusion des modèles : 1 et 2 couches à 64 neurones.

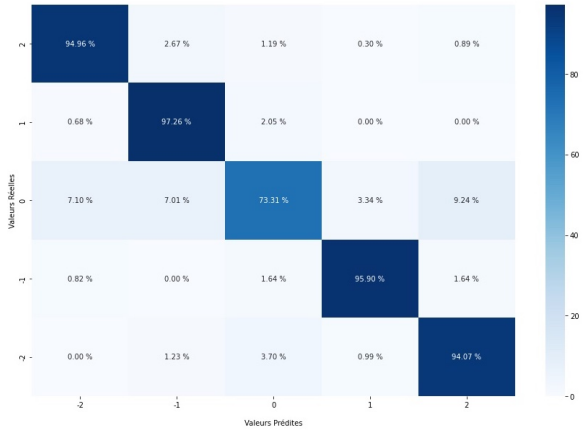


FIGURE 22 – Modèle 1 couche - 64 neurones

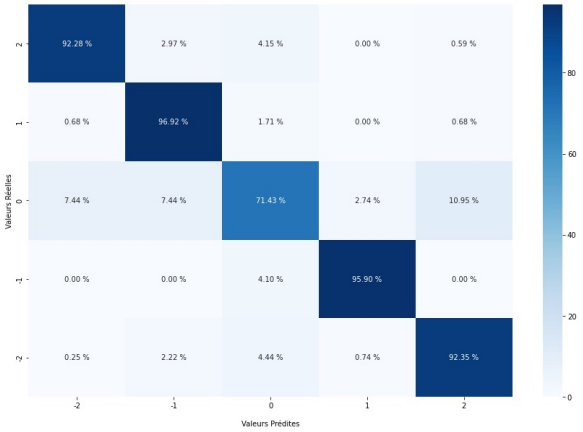


FIGURE 23 – Modèle 2 couches - 64 neurones

On voit bien que le modèle 1 couche à 64 neurones est le plus performant. Nous garderons donc cette architecture.

4.2.5 Résultats

Avant de rentrer dans le détail de nos résultats, il faut se rappeler quel était notre objectif principal. Comme on veut prédire la classification au mieux 1 jour à l'avance, on ne peut entrainer notre modèle sur toutes les données. Nous avons donc entrainé différents modèles sur les données de 2013 à 2021 sur notre modèle complexe. Cependant, même si l'algorithme arrivait à faire décroitre la courbe de la fonction de coût sur les données d'entrainement, la courbe de la fonction de coût sur les données de 2022 ne faisait que croître. Nous avons même essayé d'introduire une fonction de coût qui permettait d'améliorer les résultats dans le cas d'une sur-représentation d'une classe en affectant des coûts d'erreur plus élevés aux autres classes (focal loss cross-entropy) : le résultats restait le même, la courbe de la fonction de coût ne décroissait pas pour les données de 2022.

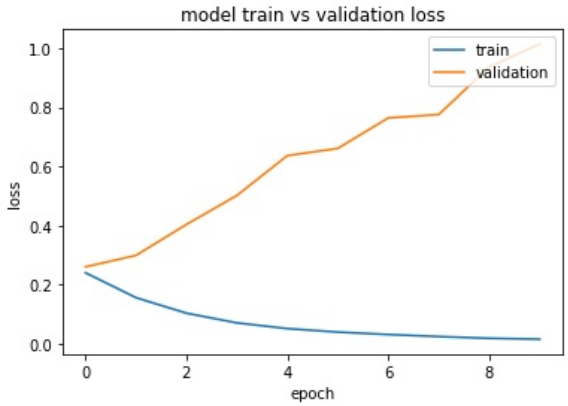


FIGURE 24 – Fonction de coût avec cross entropy

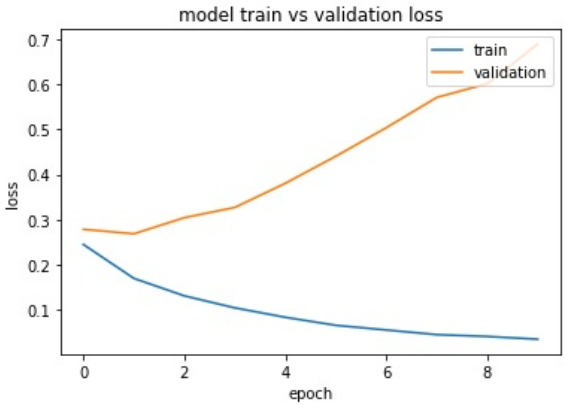


FIGURE 25 – Fonction de coût avec focal loss

Nous avons donc ensuite essayé de nous ramener à des modèles plus simples c'est à dire avec des architectures minimalistes telles que :

```
sfc = tfa.losses.SigmoidFocalCrossEntropy(gamma=2)
```

```
model = tf.keras.models.Sequential()
model.add(LSTM(5, input_shape=(21,1), activation='softmax'))
model.compile(loss=sfc, optimizer='adam', metrics=['accuracy'])
```

```
model = tf.keras.models.Sequential()
model.add(LSTM(64, input_shape=(21,1), activation='softmax'))
model.add(Dense(5, activation="softmax"))
model.compile(loss=sfc, optimizer='adam', metrics=['accuracy'])
```

Initialement, nous n'avions pas de changement dans les résultats. Nous avons donc changé notre méthode d'entrainement, nous avons fait un entrainement uniquement sur 2021 pour prédire 2022 (online machine learning). Cette modification nous a permis (avec ces modèles simplifiés) de nous délaisser du problème de décroissance de la courbe de la fonction de coût, mais le pourcentage de correspondance de prédiction (l'accuracy) ne dépassait jamais les 50%. Généralement, l'accuracy trouvée stagnait autour de la proportion d'élement dans la classe 0 en 2022.

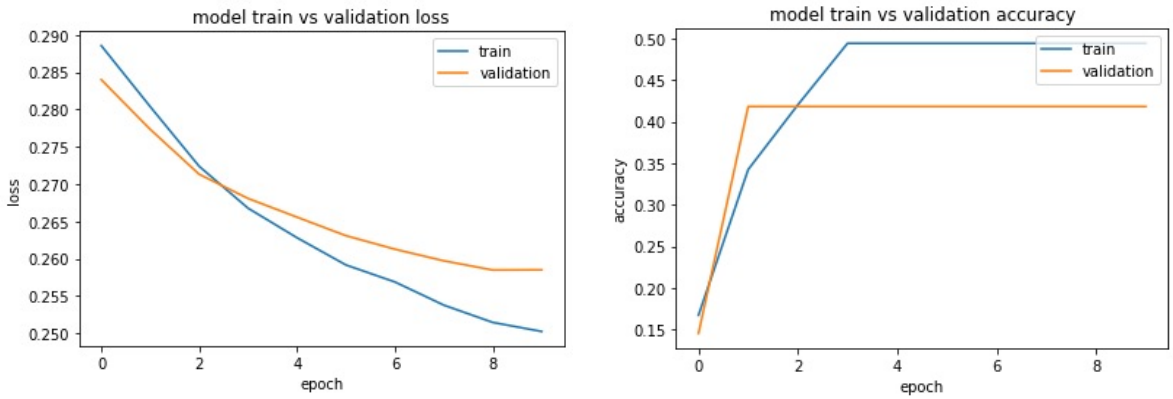


FIGURE 26 – Loss et Accuracy - modèle 2 couches avec 5 neurones LSTM

4.3 Forêts aléatoires

Au vu des résultats de notre modèle LSTM, nous avons décidé d'étudier nos données et les relations qui existent entre elles. Pour cela, nous avons décidé de mettre en place un modèle de forêt aléatoire (Random Forest), qui est un modèle de régression non paramétrique [6].

Définition 11 *Une forêt aléatoire est un modèle qui se base sur les arbres de décision. Soit $X \in [0,1]^d$, soit $Y \in \mathbb{R}^m$. On cherche à estimer une fonction $f : \mathbb{R}^d \to \mathbb{R}^m$ telle que :*

$$Y = f(X) + \xi, \quad \xi \sim \mathcal{N}(0, \sigma^2 I)$$

On ne suppose pas que f appartient à un modèle paramétrique, ainsi on cherche à estimer f seulement à partir des données.

Soit $n \in \mathcal{N}$. On va construire l'estimateur $\widehat{f_n} : \mathbb{R}^d \to \mathbb{R}^m$ de f à partir de $(X_i, Y_i)_{1 \leq i \leq n}$ qui sera la moyenne de M estimateurs que l'on aura construits :

$$\widehat{f_n} : x \to \frac{1}{M} \sum_{j=1}^{M} \widehat{f_{n,j}}(x, \eta_j)$$

où η_j est l'ensemble des variables aléatoires nécessaires pour construire $\widehat{f_{n,j}}$. On s'intéresse à présent à la construction des estimateurs $\widehat{f_{n,j}}$. On pose :

— $\forall x \in \mathbb{R}, \forall j \in [1, M], A_n(x, \eta_j)$

— $\forall x \in \mathbb{R}, \forall j \in [1, M], N_n(x, \eta_j) = \sum_{i=1}^{n} \mathbb{1}_{X_i \in A_n(x, \eta_j)}$

Avec ces éléments, on peut définir :

$$\widehat{f_{n,j}} : x \to \sum_{i=1}^{M} \frac{1}{N_n(x, \eta_j)} . \mathbb{1}_{X_i \in A_n(x, \eta_j)} Y_i$$

Il nous reste alors à définir comment construire les partitions $A_n(x, \eta_j)$.

1. On initialise la partition $P = [0,1]^d$
2. Pour chaque $A_i \in P$:
   (a) on tire η_i éléments de $1, ..., d$ sans remise, qu'on stocke dans M_d
   (b) on optimise le critère CART (Classification And Regression Tree) :

   Soient $x \in \mathbb{R}, j \in M_d$:

   $$L_n(j, x) = \frac{1}{\sum_{i=1}^{n} \mathbb{1}_{X_i \in A}} \left[\sum_{i=1}^{n} (Y_i - \overline{Y_a})^2 \mathbb{1}_{X_i \in A} - \sum_{i=1}^{n} (Y_i - \overline{Y_{A_L}} \mathbb{1}_{X_i(j) < x} - \overline{Y_{A_R}} \mathbb{1}_{X_i(j) > x})^2 \mathbb{1}_{X_i \in A} \right]$$

   où $A_L = u \in A; u(j) < x; \quad A_R = u \in A; u(j) > x \quad ; \overline{Y_A} = \frac{1}{\sum_{i=1}^{m} \mathbb{1}_{X_i \in A}} \sum_{i=1}^{M} \mathbb{1}_{X_i \in A} Y_i.$

   On partitionne alors en choisissant $(j_{optimal}, x_{optimal}) = \arg\max_{j,x} L_n(j, x)$
3. On remplace la partition P par celle obtenue après tous les découpages.

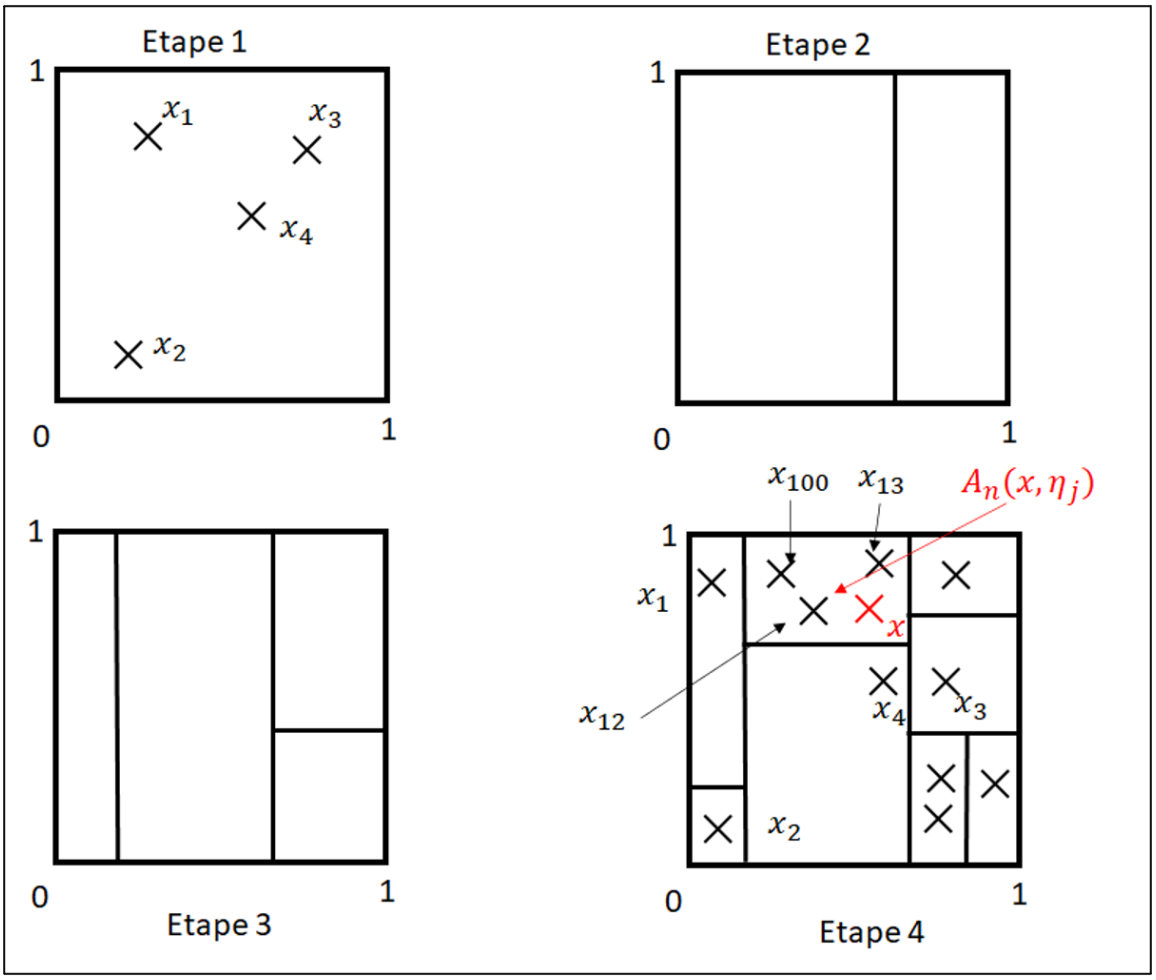


FIGURE 27 – Illustration partionnement

4.3.1 Motivation et Implémentation des forêts aléatoires

Nous avons choisi d'implémenter une forêt aléatoire car l'estimateur qu'on établit n'est pas supposé correspondre à un modèle paramétrique, ainsi il n'est le résultat que des données et permet de mettre en avant les relations que les données ont les unes avec les autres.

Nous avons utilisé la librairie *scikit-learn* pour coder notre modèle de Random Forest. Cette librairie nous offre la possibilité d'accéder aux importance features : cette fonctionnalité nous permet de déterminer le degré d'importance et d'influence de chaque variable utilisée pour entraîner le modèle.

L'idée était d'entraîner un modèle de forêt aléatoire afin qu'il puisse prédire les classes de spread des données de test. Il nous restait à déterminer quelles données passer en paramètres d'apprentissage à la forêt.

Notre première approche a été de donner en paramètre d'entrée le spread journalier, le spread de l'Initial Balance et le delta des sept journées précédentes (on gardait la même approche que pour le LSTM).

Nous avons décidé d'entraîner deux modèles différents, l'un qui ne soit entraîné que sur les

données de 2021, et l'autre sur toutes les données disponibles jusqu'à la fin de l'année 2021, en gardant pour les deux modèles les données du début de l'année 2022 comme jeu de test. L'idée était de voir si donner plus de données d'entraînement au modèle serait bénéfique ou non. Nous demandions au modèle de classifier les spreads dans 3 classes, en regroupant les classes -2 et -1 dans la classe qu'on a nommée "classe -" et les classes +1 et +2 dans la classe "classe +". Les résultats sont similaires à ceux obtenus avec le LSTM : le modèle prédit très souvent le spread comme étant dans la classe 0.

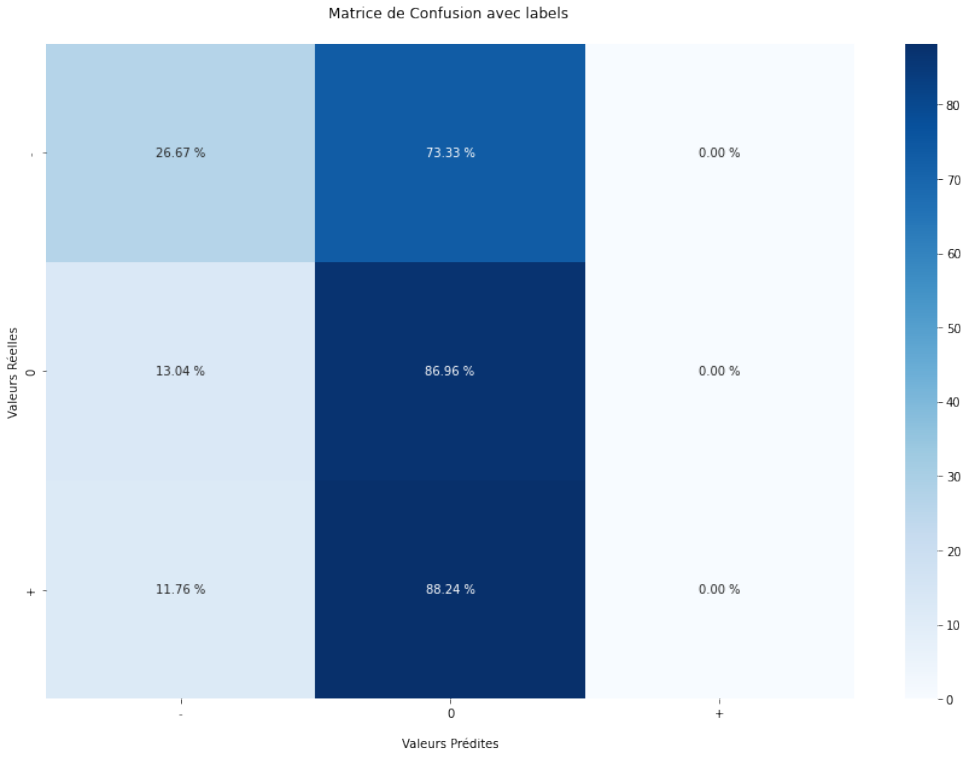


FIGURE 28 – Matrice de confusion de la forêt entraînée sur les données de 2013 à 2021, sur les 7 jours précédents

La précision finale, qui donne le pourcentage de classes qui ont été prédites correctement sur les données de test, est ici égale à 43.6%. On remarque effectivement que le modèle a prédit majoritairement les spreads comme étant dans la classe 0. L'autre modèle est arrivé à des conclusions très similaires. Les résultats de précision sont donc en dessous de nos attentes. En effet, par construction, la forêt aléatoire devrait donner des résultats plus satisfaisants.

On s'intéresse alors aux importance features pour comprendre quelles données ont été importantes pour l'entraînement du modèle.

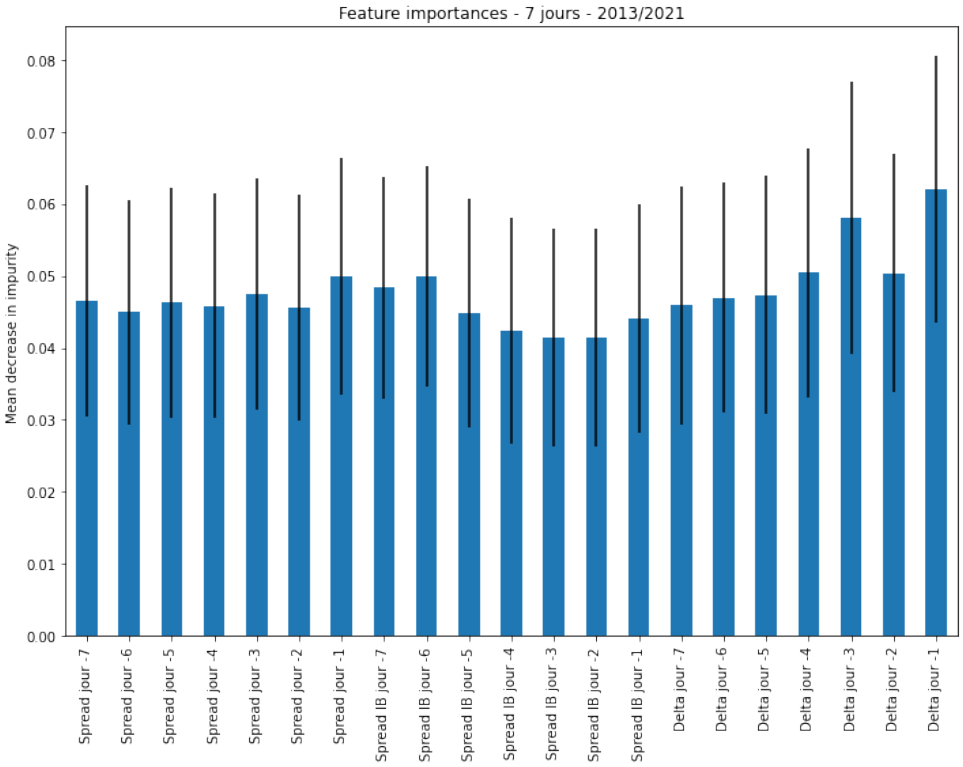


FIGURE 29 – Graphe des Features Importance

Ce graphique donne le degré d'importance de chaque donnée qui a été utilisée lors de l'entraînement. Plus la barre bleue est haute, plus ce degré d'importance est élevé. Ce qui est intéressant ici, c'est que les données semblent avoir été toutes plus ou moins autant importantes. Ainsi, on serait en droit de se demander si les données sont liées entre elles.

Dans une tentative d'amélioration des résultats de classification de la forêt aléatoire, nous avons décidé de créer de nouveaux modèles avec des données d'entraînement différentes passées en entrée. Nous avons gardé la même idée que précédemment en créant pour chaque cas deux modèles, l'un entraîné sur l'ensemble des données de 2013 à 2021, et l'autre entraîné sur les données de 2021.

Dans notre deuxième modèle, nous avons donné à nos modèles le spread de l'Initial Balance, le spread des données en 5 minutes et les volumes des données en 5 minutes. L'idée était de voir si les connaissances des spreads et volumes passés, additionnées à celle des spreads de la matinée pouvaient améliorer nos résultats. Notre troisième approche a été de ne donner que les spreads et les volumes des données en 5 minutes en paramètres d'entrée. Les précisions de classification n'ont cependant pas été réellement impactées. Nous résumons les résultats de classification ci-dessous.

Précision des classifications des données de test	Apprentissage sur 2013 - 2021	Apprentissage sur 2021
Première approche	45%	43.6%
Deuxième approche	40%	45%
Troisième approche	47%	45%

On remarque effectivement que les valeurs de précision ne varient que peu, et restent toujours

décevantes. Si on s'intéresse aux importance features, de même, ces dernières ne varient pas beaucoup.

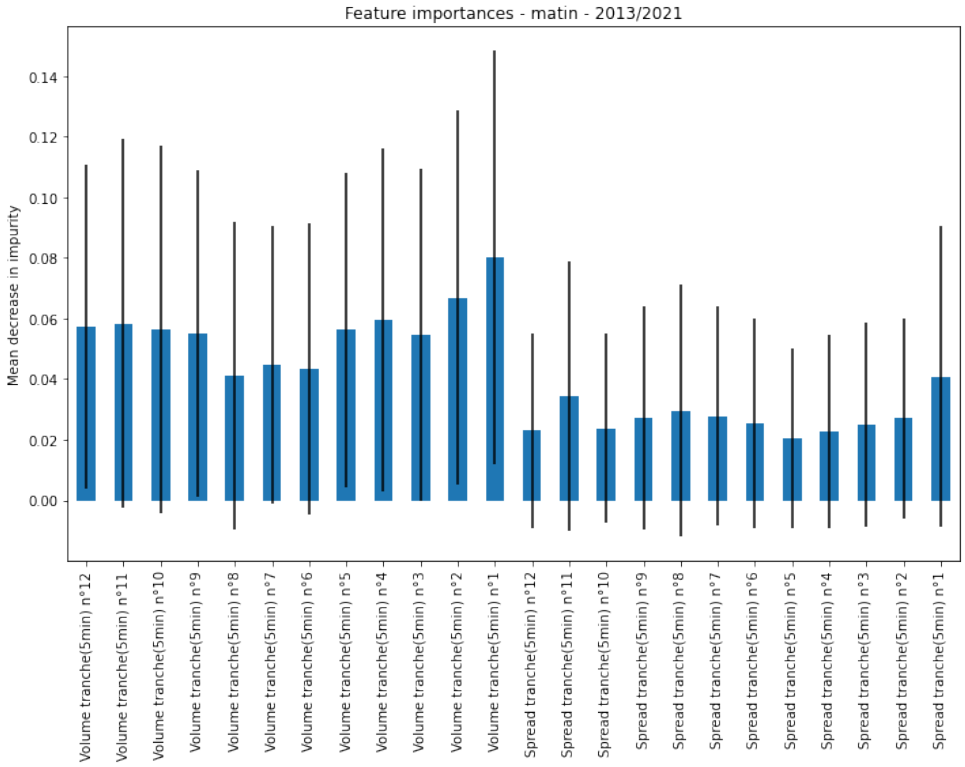


FIGURE 30 – Graphe des Features Importance

Cependant, on remarque sur l'exemple ci-dessus, qui correspond aux importance features du troisième modèle entraîné sur les données de 2013 à 2021, que le volume semble avoir été plus important dans la prise de décision de la classification que ne l'ont été les spreads.

Ces résultats de nos modèles de Random Forest nous poussent à nous demander si les données sont suffisamment liées entre elles pour être prédites par un LSTM.

5 Estimation de Densité

Au vu des résultats obtenus avec notre LSTM, il semblerait que notre modèle ne soit pas optimal pour classifier les données. Nous avons donc voulu apporter des modifications non pas au modèle, mais à la manière dont étaient classées les données. En effet, les données de spread avec lesquelles nous travaillions avaient déjà été classées, en s'appuyant sur le calcul de leur z-score.

Pour rappel, nous avions choisi une période égale à 14 pour le calcul des moyennes et écart-types mobiles. Voici comment étaient classées les données de spread (voir ci-dessous). Cependant, si on s'intéresse à la distribution des données de spread, il semble y avoir des différences.

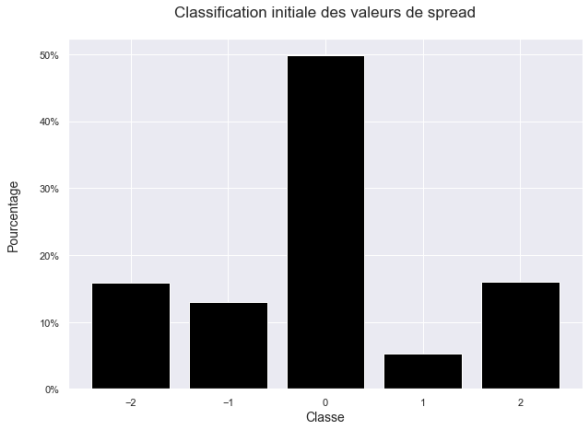


FIGURE 31 – Classification initiale

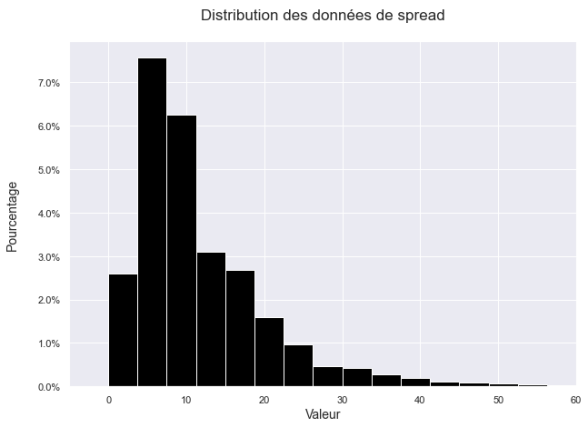


FIGURE 32 – Distribution Spread

5.1 Normalisation des données

Ce qui nous intéresse dans les données de spread, ce n'est pas leur valeur à un instant j, mais leur valeur à cet instant comparée aux valeurs précédentes. En effet, ce raisonnement nous permet de distinguer des spreads importants ou faibles à l'échelle d'une moyenne mobile de profondeur 14. Nous avons donc normalisé nos données de spread, en les divisant par la valeur de la moyenne mobile de profondeur 14, comme suit :

$$spread_{normalise,i} = \frac{spread_i}{\frac{1}{14}\sum_{j=i-14}^{i-1} spread_j}$$

5.2 Théorie de l'estimation de densités

Pour classifier nos données, nous avons d'abord cherché à estimer une densité de probabilité qui se rapproche de la distribution de nos variables. Pour cela, nous avons mis en place une estimation de densité par noyau [7]. "*L'estimation par noyau [...] est une méthode non-paramétrique d'estimation de la densité de probabilité d'une variable aléatoire. Elle se base sur un échantillon d'une population statistique et permet d'estimer la densité en tout point du support.*" (Wikipédia).

Définition 12 *On note x_i nos données de spread, pour $i \in 1, ..., N$, N la taille de notre jeu de données. L'estimateur non-paramétrique de densité par notre méthode est alors :*

$$\forall x \in \mathbb{R}, \widehat{f}_h(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right)$$

où h est un réel appelé fenêtre et K est une fonction réelle appelée noyau. K vérifie :

— $\int_{-\infty}^{+\infty} K(u)du = 1$

— $\forall u \in \mathbb{R}, K(u) = K(-u)$

Il existe de nombreux noyaux, mais les plus courants sont les suivants :

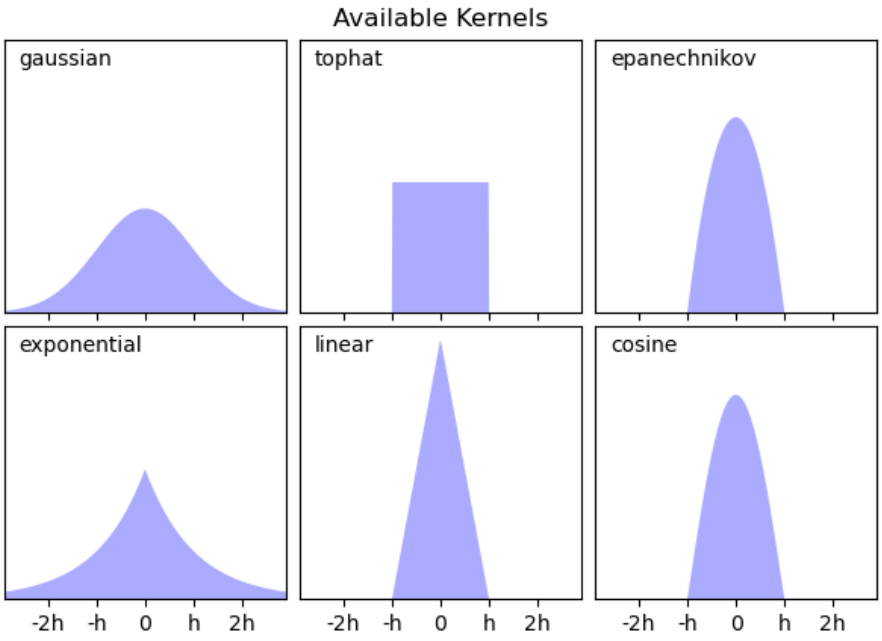


FIGURE 33 – Noyaux courants

La fenêtre h agit comme un paramètre de lissage, contrôlant le compromis entre le biais et la variance du résultat. Une grande valeur de h conduit à une distribution de densité très lisse (c'est-à-dire avec un biais élevé), alors qu'une petite valeur conduit à une distribution de densité non lisse (c'est-à-dire à forte variance).

5.3 Mise en pratique

Pour mettre en place cette estimation de densités, il nous fallait déterminer les hyperparamètres K et h. Un hyperparamètre est un paramètre qui n'est pas directement appris par un estimateur, et dont la valeur est utilisée pour contrôler le modèle d'apprentissage.

Nous les avons déterminés de manière empirique. Pour cela, nous avons utilisé la fonction GridSearchCV de scikit-learn afin d'effectuer une cross-validation sur les différents hyperparamètres et sélectionner la paire optimale.

Définition 13 *La cross-validation [8], ou validation croisée, est "une méthode d'estimation de fiabilité d'un modèle fondée sur une technique d'échantillonnage.". (Wikipédia) Nous avons utilisé la cross-validation appelée k-fold. L'approche est la suivante : le jeu de données est divisé en un jeu d'entrainement (training set), qui regroupe entre 75% et 80% des données, et un jeu de test (test set). Ensuite, le jeu d'entraînement est de nouveau divisé en k jeu de données plus petits, appelés blocs ou "folds" en anglais*

Pour chaque bloc de données :

— un modèle est entraîné sur les k-1 autres blocs de données ;

— le modèle résultant est évalué et validé sur le bloc sélectionné ; en d'autres termes, ce modèle est utilisé pour calculer une mesure de performance globale, comme la précision.

La mesure de performance déterminée par la cross-validation est alors la moyenne des valeurs calculées sur chaque bloc k.

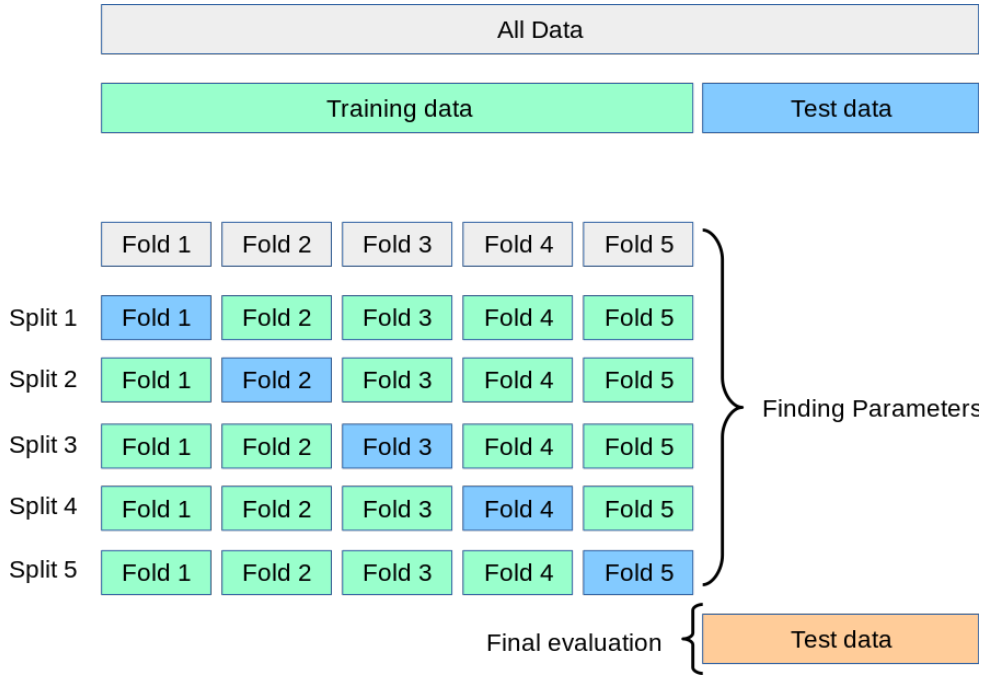


FIGURE 34 – Blocs - Crossvalidation

Finalement, on s'intéresse aux hyperparamètres qui ont donné la mesure de performance maximale, que l'on considère alors comme hyperparamètres optimaux pour notre modèle. Ainsi, la cross-validation permet de sélectionner les meilleurs hyperparamètres en évitant le sur-apprentissage ("overfitting" en anglais) : "*une analyse statistique qui correspond trop précisément à une collection particulière d'un ensemble de données. Ainsi, cette analyse peut ne pas correspondre à des données supplémentaires ou ne pas prévoir de manière fiable les observations futures.*" (Wikipédia).

Nous avons effectué cette procédure pour le spread des valeurs de 5 minutes d'intervalle, le spread journalier et le spread des données entre 8h et 9h du matin (le spread de l'Initial Balance). Voici les résultats que nous avons obtenus :

Le noyau exponentiel est de la forme : $\forall u \in \mathbb{R}, K(u) = \frac{1}{2}e^{-|u|}$

Données	Noyau K	Fenêtre h
Spread - 5 minutes	exponentiel	h = 0.059
Spread - Initial Balance	exponentiel	h = 0.059
Spread - Journalier	exponentiel	h = 0.045

Une fois les hyperparamètres déterminés, nous pouvions effectuer l'estimation de densité. Pour cela, nous nous sommes servi de la fonction KernelDensity présente dans la libraire de scikit-learn. Voici les résultats que nous avons obtenus pour l'estimation de densité des valeurs du spread normalisé.

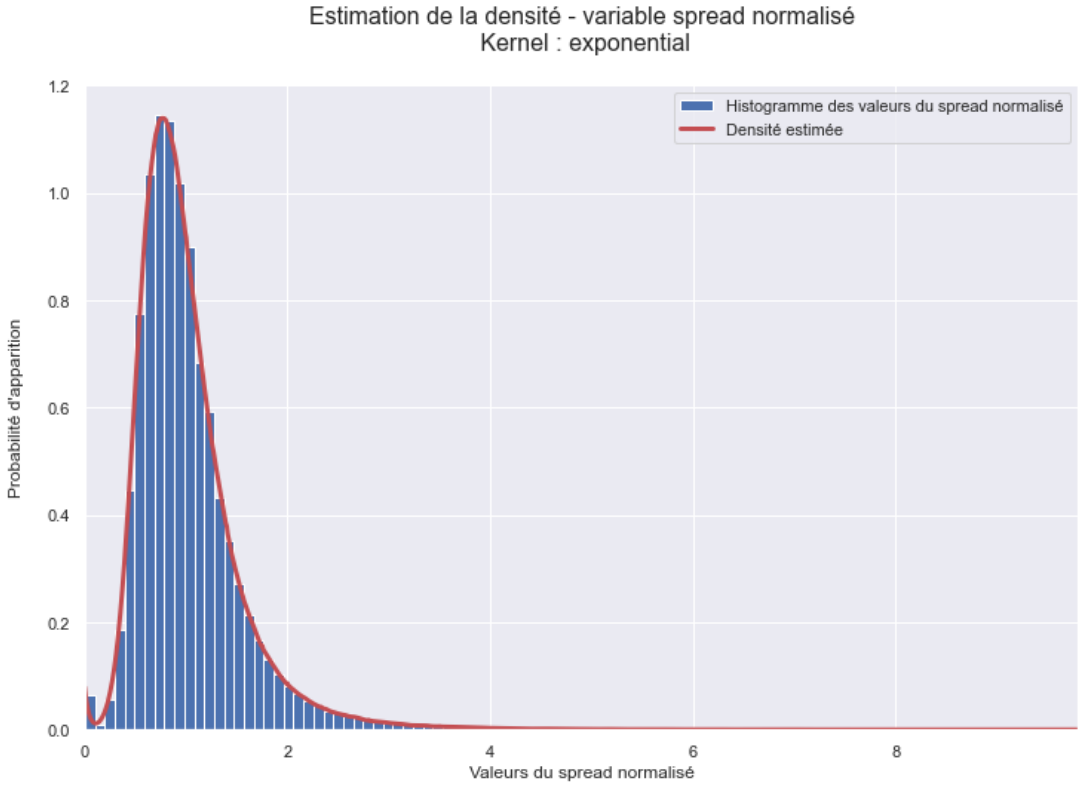


FIGURE 35 – Estimation de densité

5.4 Recherche de quantiles

A l'aide des fonctions de densité que nous avons estimées précédemment, nous pouvons rechercher leurs quantiles. Notre approche a été d'écrire un script Python qui calcule l'intégrale $\int_X^{+\infty} f(x)dx$ où f est la fonction de densité estimée. Nous faisons alors diminuer X jusqu'à ce que la valeur de l'intégrale atteigne le quantile recherché. Ainsi, ce script nous permet de calculer tous les quantiles des fonctions de densité que l'on estime.

5.4.1 Classification des données du spread

Dans notre étude, nous nous intéressons principalement aux 5^e et 95^e centiles. En effet, pour que les classes extrêmes (classes -2 et 2) ne soient pas trop conséquentes ou trop faiblement

représentées, nous les avons déterminées à l'aide des centiles calculés. Ainsi, si une valeur est inférieure au 5e centile (donc inférieure à 95% des valeurs du jeu de données), nous la plaçons dans la classe -2 ; au contraire, si cette valeur est supérieure au 95e centile (soit supérieure à 95% des valeurs du jeu de données), nous la plaçons dans la classe +2.

Pour les classes restantes, soient les classes -1, 0, 1, nous réalisons une classification avec le z-score. Voici comment nous avons procédé pour classer les données du spread dans 5 classes.

Classe	**Condition**
-2	X <5e centile
-1	5e centile <X ET Z <σ_1
0	σ_1 <Z <σ_2
1	σ_2 <Z ET X <95e centile
2	X >95e centile

où X est une valeur et Z est son z-score.

Les paramètres σ_1 et σ_2 sont des entiers, qu'il nous faut déterminer. Le but auquel nous cherchions à aboutir avec notre classification était d'avoir des classes représentant le mieux possible nos données, et donc collant au maximum la fonction de densité. Comment calculer les σ_1 et σ_2 optimaux ? Nous avons calculé les quantiles de notre fonction de densité à 25%, 50% et 75%, que l'on note q25, q50 et q75. Pour chaque σ_1 et σ_2 fixés, on calcule un score :

$$\begin{aligned} score(\alpha_1, \alpha_2) = &\left| \frac{Card(classe_{-1})}{Card(X)} - \frac{f(q_{25})}{f(q_{25}) + f(q_{50}) + f(q_{75})} \right| \\ &+ \left| \frac{Card(classe_0)}{Card(X)} - \frac{f(q_{50})}{f(q_{25}) + f(q_{50}) + f(q_{75})} \right| \\ &+ \left| \frac{Card(classe_{+1})}{Card(X)} - \frac{f(q_{75})}{f(q_{25}) + f(q_{50}) + f(q_{75})} \right| \end{aligned}$$

où $Card(classe_i)$ pour $-1 \leq i \leq 1$ est égal au nombre d'éléments classés en i, $Card(X)$ est égal au nombre d'éléments total dans notre jeu de données et f est la fonction de densité estimée.

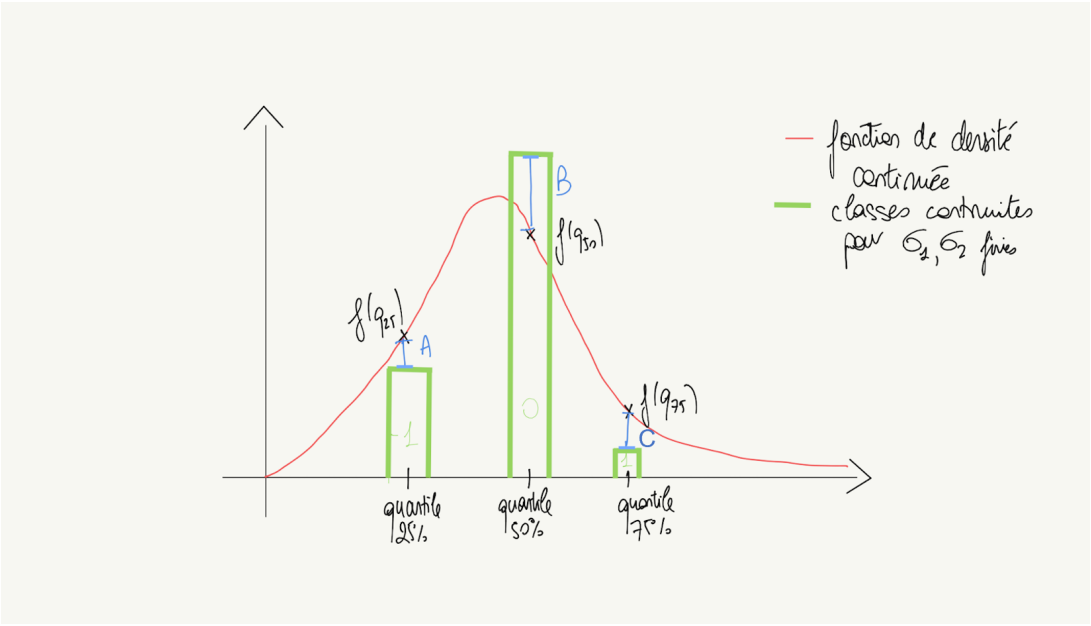


FIGURE 36 – Représentation pour une mauvaise classification

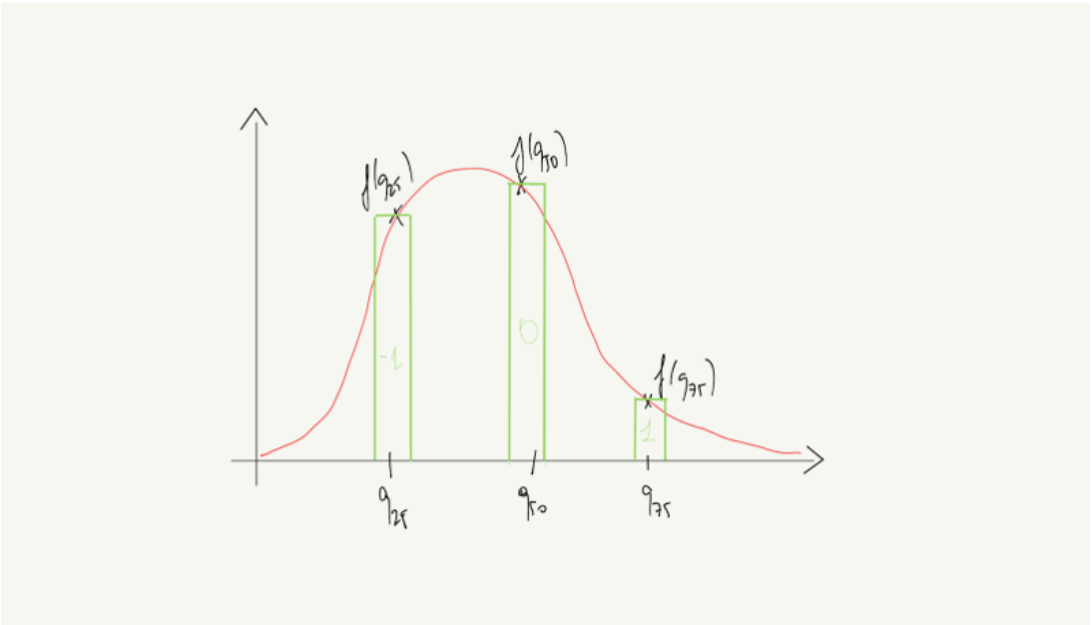


FIGURE 37 – Représentation pour une bonne classification

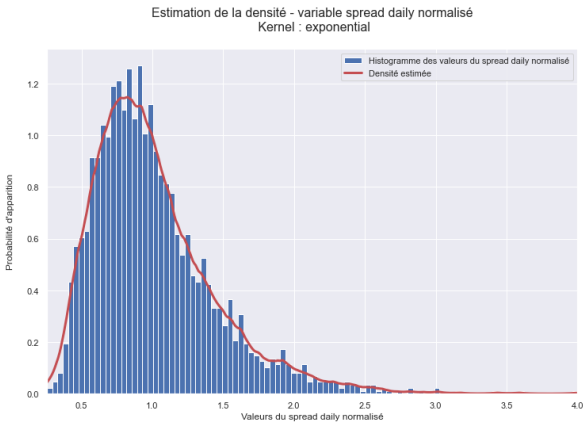
On peut voir, sur le 1er schéma, qu'on a : $score(\sigma_1, \sigma_2) = A + B + C$, ce qui représente la différence entre la classification effectuée pour σ_1, σ_2 fixés et la courbe de la fonction de densité estimée.

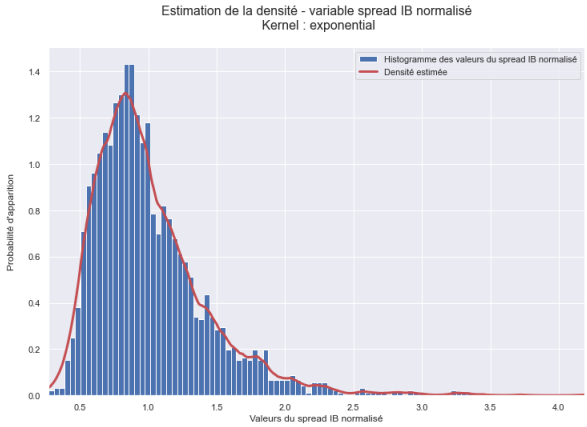
On comprend aussi, avec le second schéma, que la classification optimale, celle qui épouse parfaitement la courbe de la fonction de densité estimée, est atteinte lorsque $score(\sigma_1, \sigma_2) = 0$. Minimiser ce score nous permet de rapprocher notre classification de la fonction de densité

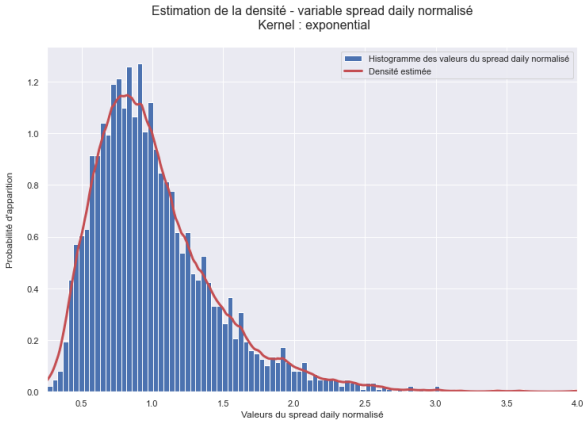
estimée, on pose donc : $\alpha_{1,optimal}, \alpha_{2,optimal} = \underset{\alpha_1,\alpha_2}{\arg\min}\, score(\alpha_1, \alpha_2)$.

	5e centile	σ_1	σ_2	95e centile
Spread - 5 minutes	0.44	-0.36	0.86	1.97
Spread - Initial Balance	0.50	-0.49	0.6	1.82
Spread - Journalier	0.46	-0.42	0.56	1.88

5.4.2 Affichage des résultats de classification du spread







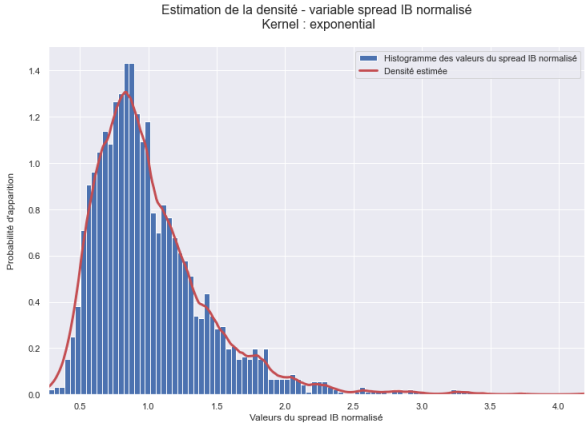


FIGURE 38 – Spread journalier

FIGURE 39 – Spread de l'Initial Balance

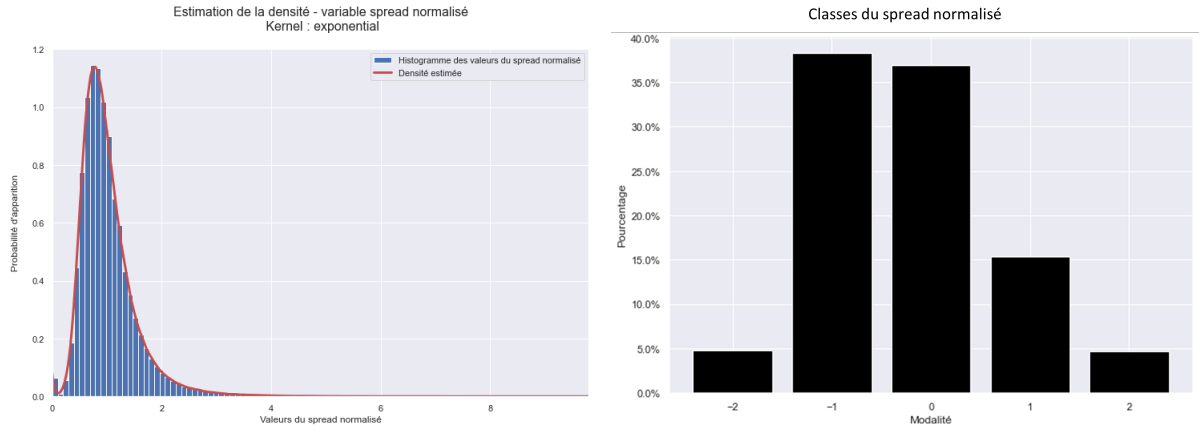


FIGURE 40 – Estimation de densité et classification du spread de 5 minutes

5.4.3 Classification des données de volume

De la même manière que pour les données de spread, on va chercher à classifier les données de volume (avec les mêmes intervalles : 5 minutes, Initial Balance et journalier).

On va donc estimer les fonctions de densités sur ces nouvelles valeurs, chercher leurs quantiles puis les classer. La différence cependant, dans ce cas, est que nous allons classer les données non pas en 5 classes, mais en 7 classes. Ainsi, la classification sera effectuée de la sorte.

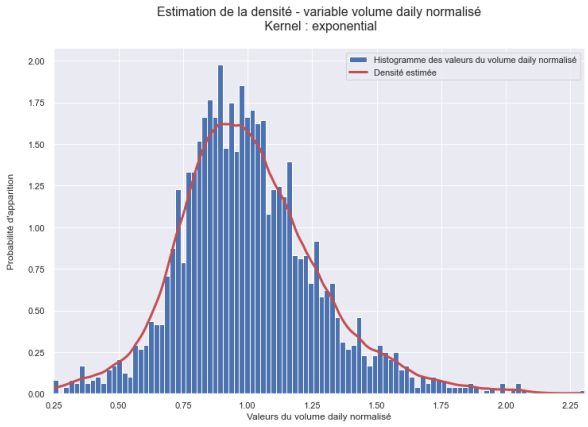
Classe	Condition
-3	X <5e centile
-2	5e centile <X ET Z <σ_2
-1	σ_2 <Z <σ_1
0	σ_1 <Z <β_1
1	β_1 <Z <β_2
2	β_2 <Z ET X <95e centile
3	X >95e centile

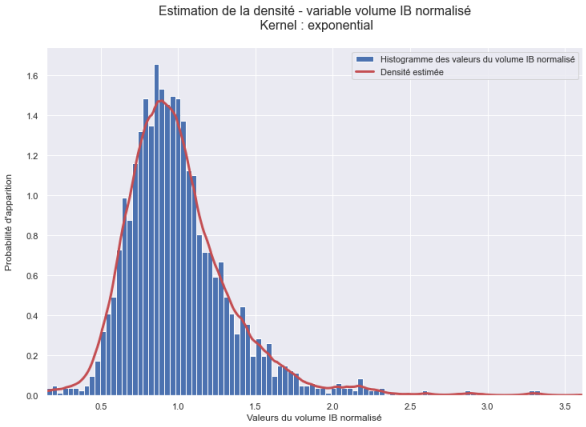
où X est une valeur et Z est son z-score.

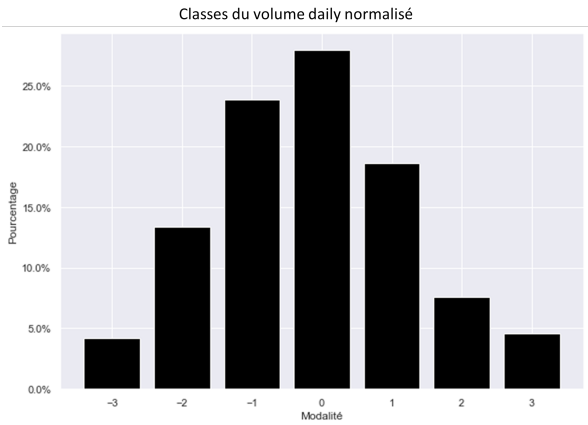
De nouveau, nous avons déterminé les paramètres σ et β à l'aide d'une cross-validation pour minimiser la fonction de score.

	5e centile	σ_2	σ_1	β_1	β_2	**95e centile**
Volume - 5 minutes	0.59	-1	-0.3	0.5	1.4	1.51
Volume - Initial Balance	0.55	-0.9	-0.3	0.5	1.4	1.64
Volume - Journalier	0.30	-0.8	-0.2	0.8	2.1	2.36

5.4.4 Affichage des résultats de classification du volume







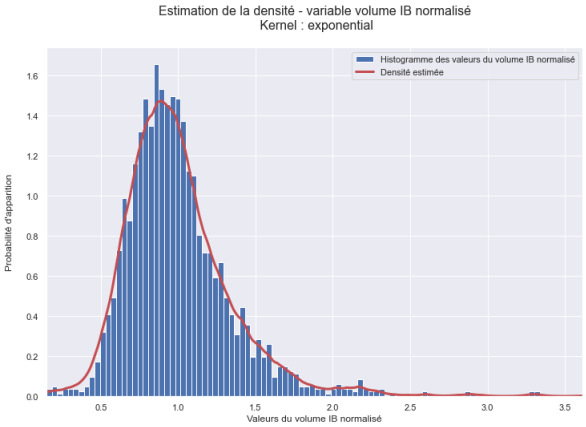
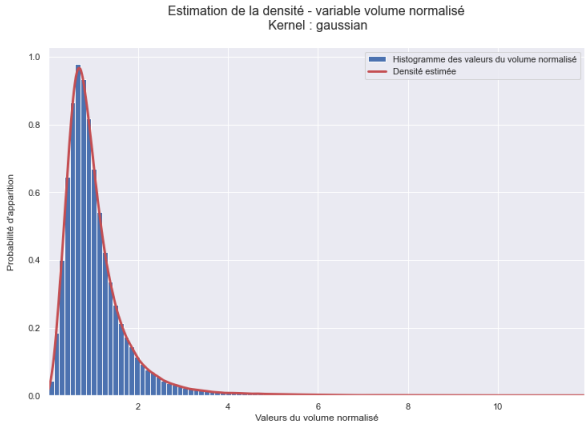


FIGURE 41 – Volume journalier

FIGURE 42 – Volume de l'Initial Balance



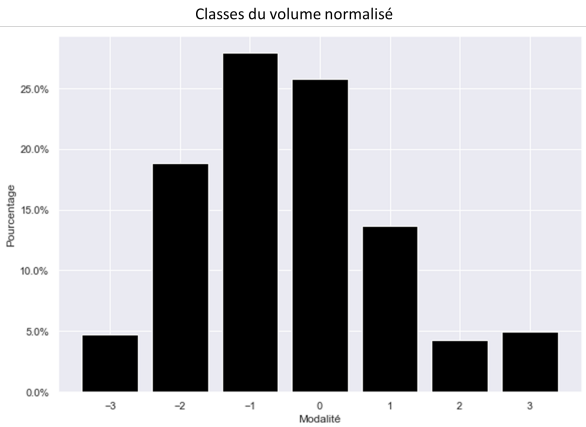


FIGURE 43 – Estimation de densité et classification du volume de 5 minutes

5.4.5 Impact sur les trades

Les études réalisées par Laurent Abril montrent que les spreads qui se trouvent dans les classes -2 et -1 (pour la classification initiale, celle qui nous était donnée) sont majoritairement perdants.

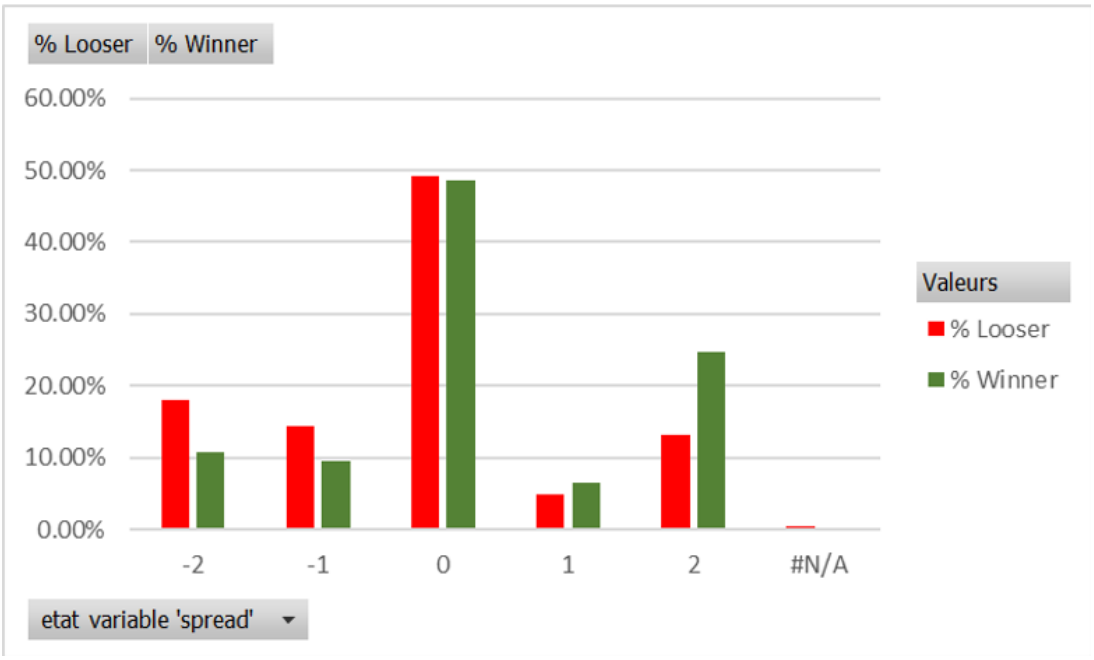


FIGURE 44 – Répartition trades selon la classe

Ainsi, l'idée du projet était de pouvoir prédire les classes de spread de la journée du lendemain, afin de pouvoir aviser et notamment prendre moins de trades sur le journées dont le spread avait été prédit dans la classe -2 ou -1, permettant d'assurer la baisse globale du nombre de trades perdus.

Notre classification permettrait d'améliorer ces résultats. En effet, les classes que nous avons réussi à mettre en place se rapprochent de la distribution des spreads, et permettent ainsi de bien décrire ces variables.

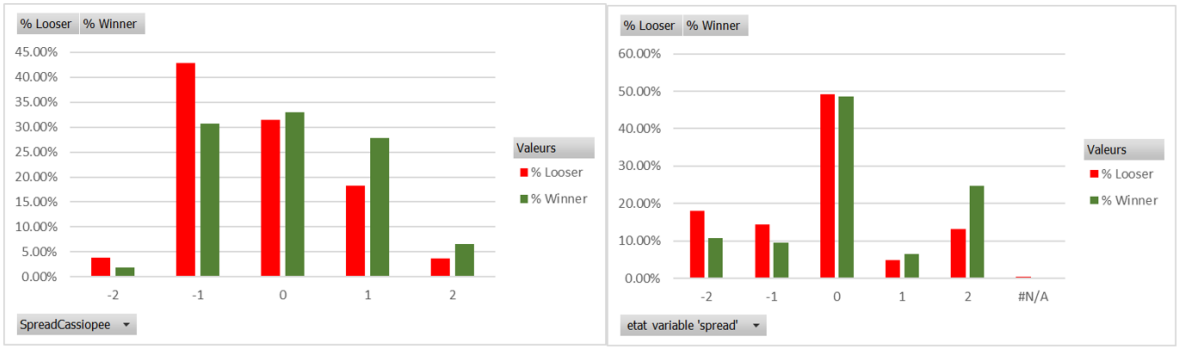


FIGURE 45 – Comparaison Trades selon la classification

Ces deux graphes montrent les pourcentages de trades gagnés ou perdus par classe. Le graphe

de gauche présente les valeurs obtenues avec les classifications que nous avons déterminées, et celui de droite présente celles obtenues avec les classifications initiales. Ici, on s'intéresse à la classification du spread journalier.

Si l'on compare les pourcentages de trades gagnés et perdus par classe, on se rend compte que, dans le cas de notre normalisation, les classes -2 et -1 contiennent majoritairement des trades perdus, mais les autres classes sont toutes dominées par des trades gagnants.

Pour mettre en perspective ce que représentent ces résultats, nous introduisons l'indicateur P&L (Profit & Loss). Cet indicateur, dont la valeur est égale à la somme des profits moins la somme des pertes, permet de mesurer la performance économique de nos trades. Ainsi, on cherche à maximiser cet indicateur.

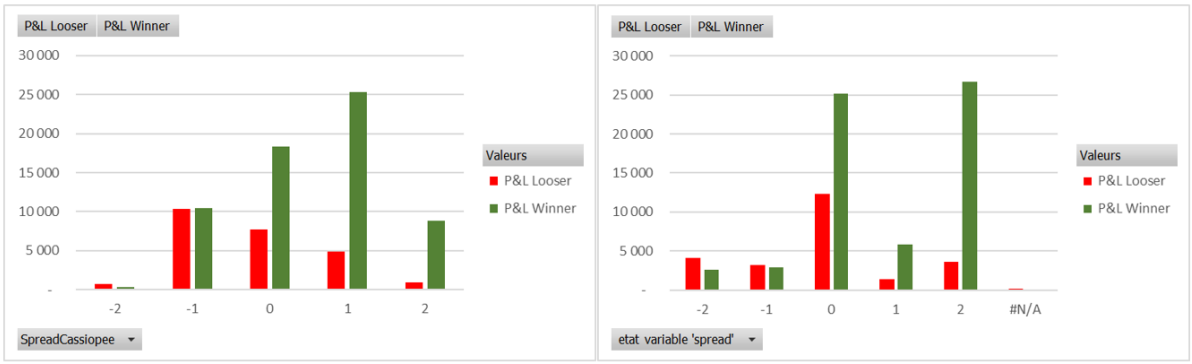


FIGURE 46 – Comparaison P&L selon la classification

Sur la figure ci-dessus, sont présentées les valeurs de l'indicateur P&L pour chaque classe. De nouveau, le graphique de gauche présente les valeurs que l'on obtient avec la classification que nous avons calculée, et celui de droite présente ce que l'on obtient avec la classification initiale. D'après l'étude réalisée par Laurent Abril, dans le cas où notre LSTM puisse prédire avec précision la classe du spread d'une journée à venir, notre nouvelle classification permettrait de faire passer le P&L moyen par trade de 20 points de DAX par trade, à 33 points de DAX par trade. On peut conclure que notre classification permettrait de grandement améliorer nos résultats.

5.4.6 Étude sur les valeurs brutes

Dans notre étude, nous nous sommes intéressés aux données normalisées par leur moyenne mobile. L'idée était de pouvoir comparer chaque donnée aux quelques données précédentes, afin de pouvoir détecter des tendances haussières ou baissières. Mais on peut également estimer la densité et classer les données brutes, non normalisées. Nous avons ainsi refait toutes les étapes précédentes, mais cette fois-ci dans le but de déterminer une classification des données du spread journalier, sans les normaliser.

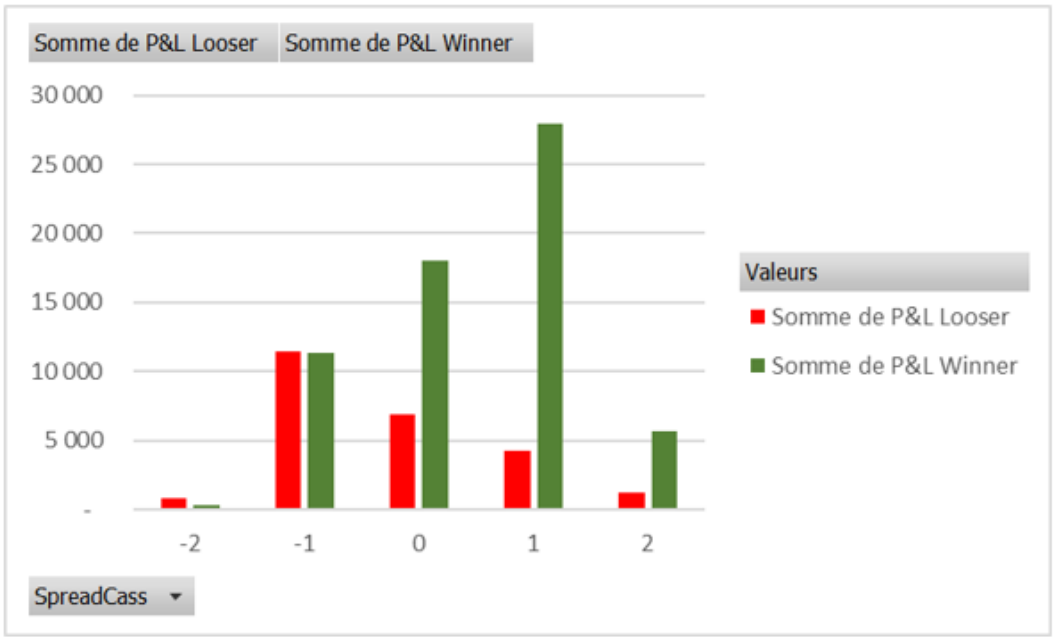


FIGURE 47 – P&L sur données non-normalisées

Il serait alors intéressant d'étudier la raison pour laquelle la classification des données brutes semble meilleure que celle des données normalisées.

Conclusion

Tout au long du projet, nous avons rencontré diverses difficultés notamment dans la compréhension et l'analyse de nos résultats mais aussi face aux différents problèmes d'ordre mathématiques ou techniques rencontrés. Au vu de la grande taille de notre jeu de données, nous nous sommes rapidement familiarisés avec ces dernières et leurs indicateurs dans un premier temps. Puis, nous nous sommes fixé un objectif, à savoir prédire à partir des données passées les variations du cours de la bourse en associant des algorithmes de machine learning aux connaissances techniques apportées par Monsieur Abril.

Dès lors, nous avons eu recours à plusieurs outils. Nous avons d'abord travaillé sur la normalisation des données. En effet, ce processus est nécessaire avant de coder les différents algorithmes de machine learning qui sont plus complexes. L'objectif principal de la normalisation est de réduire l'influence de l'inflation sur notre jeu de données sans les fausser. Nous avons alors abouti à différentes normalisations que nous avons ensuite comparées.

Ensuite, nous avons mis en place des algorithmes de LSTM qui ont été au cœur de notre sujet. En outre, nous avons passé énormément de temps à étudier quelle configuration était la plus optimale tant au niveau du nombre de couches, du nombre de neurones par couches, de la dimension du vecteur d'entrée ainsi que de la fonction de coût. Après avoir mis en place et testé les différents algorithmes de LSTM, nous avons cherché le modèle le plus performant. Pour cela, nous avons sélectionné une multitude de jeux de données pour les phases d'entraînement, de test et de validation. Encore une fois, il fallait considérer le critère de "temps" et "d'actualité".

Etant donné que les résultats obtenus n'étaient pas toujours ceux espérés, nous avons décidé, sous les conseils de monsieur Le Corff, d'effectuer une ACP avec nos données en entrée avant de les incorporer dans nos modèles de LSTM. Un autre problème s'est imposé à nous. En effet, nous avions du mal à obtenir des résultats même en ayant pris le meilleur modèle de LSTM et la configuration la plus optimale. À la suite de plusieurs semaines d'étude des résultats, nous avons conclu que nos résultats médiocres venaient soit de la trop grande complexité de nos algorithmes, soit tout simplement des données utilisées. Encore une fois, nos encadrants nous ont conseillé de nous pencher sur les forêts aléatoires et d'étudier les résultats obtenus. En outre, si ces dernières montraient des résultats convenables le problème venait de notre modèle alors que si les résultats restaient les mêmes, le problème viendrait de nos données.

Dans le même temps, nous avons travaillé sur l'estimation de densité à savoir comment améliorer la classification des différentes données, en particulier la densité des Spreads et la densité des Volumes. Cette partie du projet était grandement liée à notre travail en lien avec les LSTM puisqu'elle avait pour but que la nouvelle classification obtenue puisse améliorer les performances de notre réseau de neurones. Les résultats de cette partie du projet se sont révélés plutôt satisfaisants, les nouvelles classifications se rapprochant beaucoup plus de la réalité de nos données.

Références

[1] John Murphy. *Technical Analysis of the Financial Markets*. Valor, 2003.

[2] Charles C. Holt. "Forecasting seasonals and trends by exponentially weighted moving averages". In : *International Journal of Forecasting* 20.1 (2004), p. 5-10. issn : 0169-2070. doi : https://doi.org/10.1016/j.ijforecast.2003.09.015. url : https://www.sciencedirect.com/science/article/pii/S0169207003001134.

[3] Frank Rosenblatt. "The perceptron : a probabilistic model for information storage and organization in the brain." In : *Psychological review* 65 6 (1958), p. 386-408.

[4] G. Cybenko, D.P. O'Leary et J. Rissanen. *The Mathematics of Information Coding, Extraction and Distribution*. The IMA Volumes in Mathematics and its Applications. Springer New York, 1998. isbn : 9780387986654. url : https://books.google.fr/books?id=jDrp4QEGioMC.

[5] Diederik P. Kingma et Jimmy Ba. "Adam : A Method for Stochastic Optimization". In : (2014). url : https://arxiv.org/abs/1412.6980.

[6] Tin Kam Ho. "Random decision forests". In : 1 (1995), 278-282 vol.1. doi : 10.1109/ICDAR.1995.598994.

[7] Stanislaw Weglarczyk. "Kernel density estimation and its application". In : *ITM Web of Conferences* 23 (jan. 2018), p. 00037. doi : 10.1051/itmconf/20182300037.

[8] Andrew W. Moore. "Cross-validation for detecting and preventing overfitting". In : (2001).