Projet 9

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

Auteurs : Renaud LEURQUIN & Saïf FARES
Tuteurs : Laurent ABRIL, Sylvain LE CORFF & Bruno DEFUDE

Remerciements

Nous tenons à remercier tout particulièrement et à témoigner notre reconnaissance aux personnes suivantes pour leur implication et leur dévouement ayant contribués au bon déroulement de notre projet :

M. Laurent ABRIL, tuteur du projet, pour sa grande disponibilité, sa passion pour le sujet et la confiance qu'il nous a accordé pour mener à bien un projet très demandé, ce qui nous a grandement motivé et poussé à nous impliquer très rigoureusement dans la tâche qui nous était confiée. Nous témoignons notre reconnaissance pour ses conseils toujours pertinents, sa patience et le temps qu'il nous a accordé.

M. Sylvain LE CORFF pour son expertise mathématique en matière de Machine Learning qui nous a permis de comprendre nos erreurs, d'aller plus loin dans la compréhension des notions apprises en cours, et d'interpréter au mieux les résultats et d'avoir toujours des pistes d'amélioration à suivre. Un grand merci pour le temps qu'il nous a accordé lors de ces réunions hebdomadaires.

M. Bruno DEFUDE, directeur de recherche et de formations doctorale, pour avoir proposé ce projet conjointement avec M. Abril.

Mme Josephine KOHLENBERG, responsable des programmes Cassiopée et **Mme Nesma HOUMANI**, maître de conférence à Télécom SudParis, pour avoir mis en place et réalisé le suivi des programmes Cassiopée cette année encore.

Table des matières

I- Présentation du projet

1. Le contexte

Le **quantitative trading**, bien développé de nos jours grâce à l'augmentation des capacités de calcul, est une méthode qui repose sur l'utilisation d'outils mathématiques et statistiques assez sophistiqués. Le Machine Learning est une science qui exploite les données massives dont nous disposons aujourd'hui pour prédire certains résultats et améliorer le fonctionnement de certains systèmes. Cette science a fait son entrée dans le monde du quantitative trading et plusieurs travaux de recherches contemporains cherchent à exploiter ses résultats pour optimiser les niveaux de performance et de risque des fonds d'investissement tout en automatisant leurs gestions.

Mais la majorité des stratégies d'investissement **ignorent les volumes** et plus précisément leurs répartitions entre volume acheteurs et volume vendeurs. Ceci signifie que ces stratégies négligent la relation entre le volume de l'offre et de la demande qui influent sur le prix, pour ne s'intéresser qu'au prix lui-même. Cela conduit à une perte d'une partie de l'information fournie par les bourses.

2. L'objectif

La première étape cruciale de notre projet est **d'analyser la microstructure des marchés financiers** dans le but de comprendre comment y agissent les acteurs. En effet, il existe de grands comptes qui possèdent des fonds importants et qui ont une grande influence sur le mouvement des prix. L'objectif est d'essayer de prédire leurs prises de positions et de suivre leurs tendances.

Le but principal est donc d'exploiter nos connaissances en Machine Learning pour **concevoir des modèles de prédiction des prix** du DAX. Ces modèles prédictifs, basés sur des résultats mathématiques et probabilistes, auront comme but de prédire les tendances des bougies de prix (haussières ou baissières) et représenteront ainsi un moyen qui peut aider les traders dans leurs prises de positions.

Ces modèles ne vont toutefois pas être appliqués de manière à ne considérer que le prix, mais de manière à prendre en compte, tout de même, les volumes et leurs répartitions. La prise en compte de ces composantes dans les variables d'entrée permettra de déceler l'influence des grands fonds sur les prix et de tirer profit de ces informations.

L'apprentissage se fera sur une base de données fournie par le tuteur et contenant des données issues de la bourse et des indicateurs conçus par le tuteur lui-même. Cet apprentissage sera fait de manière progressive : il s'agira tout d'abord de construire un modèle prédictif simple qui nous servira de benchmark et l'erreur commise par le modèle sera un premier objectif à dépasser. Nous commencerons donc par construire des modèles dont les seules données d'apprentissage seront : le prix de clôture (Last), d'ouverture (Open), le prix maximum à l'intérieur d'une bougie, appelé High (bougie = intervalle de temps de 5min lors duquel les prix varient entre deux extremum), et le minimum appelé Low. Puis l'ajout progressif de variables additionnelles apportant de l'information sur les marchés sera considéré et les modèles ainsi construits seront censé logiquement avoir de meilleures performances.

II- Exploration statistique

1. Le tape

Les données telles que fournies par la Bourse, quel que soit l'actif considéré, contiennent des informations telles que :

- L'horodatage précis de la transaction : il s'agit de la date et de l'heure (précision jusqu'à la milliseconde) ;
- Le volume de la transaction, c'est-à-dire le nombre d'unités d'actifs financiers échangés ;
- L'origine de la transaction appelé Bid s'il s'agit d'une vente et Ask s'il s'agit d'un achat, tel que le volume total soit la somme des volumes acheteur et vendeur.

	Date	Time	Open	High	Low	Last	Volume	Bid Volume	Ask Volume	# of Trades
0	04/01/2013	08:00:00.0	7842.0	7844.5	7839.5	7840.5	484	78	68	89
1	04/01/2013	08:05:00.0	7840.0	7842.0	7837.0	7838.0	463	299	164	214
2	04/01/2013	08:10:00.0	7837.5	7837.5	7833.0	7836.5	477	297	180	177
3	04/01/2013	08:15:00.0	7836.5	7840.0	7835.5	7839.5	191	74	117	80
4	04/01/2013	08:20:00.0	7840.0	7840.0	7837.0	7837.0	81	68	13	49
5	04/01/2013	08:25:00.0	7837.0	7842.5	7836.5	7840.5	164	57	107	87
6	04/01/2013	08:30:00.0	7840.5	7841.5	7839.0	7840.0	112	68	44	43
7	04/01/2013	08:35:00.0	7840.0	7841.0	7839.0	7840.0	117	56	61	64
8	04/01/2013	08:40:00.0	7840.0	7841.0	7838.0	7840.0	119	72	47	55
9	04/01/2013	08:45:00.0	7840.0	7842.5	7839.0	7842.5	92	20	72	49

Figure 2.1 : Exemple du tape fournis par notre tuteur sur l'actif financier du DAX

Ici, nous voyons que les valeurs sont regroupées par tranche de 5min, ce qui permet de définir les variables Open, High, Low et Last. Le fichier CSV fournis par notre tuteur contient l'ensemble du tape du DAX depuis le 4 janvier 2013 jusqu'à fin 2019, ainsi que de nombreux autres indicateurs financiers créés par notre tuteur et ayant un sens concernant la dynamique du marché.
L'analyse exploratoire a permis de se familiariser avec les données, de déceler les corrélations entre les variables, de **mettre en avant certains aspects du marché** comme l'heure de la journée détenant en moyenne le plus grand nombre de trades, la distribution heure par heure des volumes, la corrélation entre les indicateurs, l'influence de l'initial balance, etc.

2. Initial Balance

L'initial balance est la gamme des prix de la première heure de cotation du marché des Futures. Cette première heure est assez importante car elle représente une période tensions pendant laquelle les investisseurs assurent la liquidité en faisant plusieurs trades de petites fluctuations mais dans de gros volumes afin que les acteurs en

opposition prennent position dans cette gamme (sans pour autant donner une stricte direction du marché).

Il est donc intéressant d'essayer de comprendre l'influence de l'IB sur l'évolution des prix d'une journée donnée. Pour cela, nous avons calculé le pourcentage de jours où le maximum (ou le minimum) du prix pendant l'initial balance (IB haute, IB basse) est exactement égal à celui de la journée entière. Ce pourcentage était sur les 7 dernières années égal à **44,33%**.

Comme le DAX est un indice volatile, nous avons fait le même calcul en nous permettant une marge de p points d'erreur. Nous avons ensuite tracé le graphe qui représente le pourcentage des jours où le maximum (ou le minimum) du prix pendant l'IB est, à p points près, égal à celui de la journée entière en fonction du nombre de points p. Le résultat était le suivant :

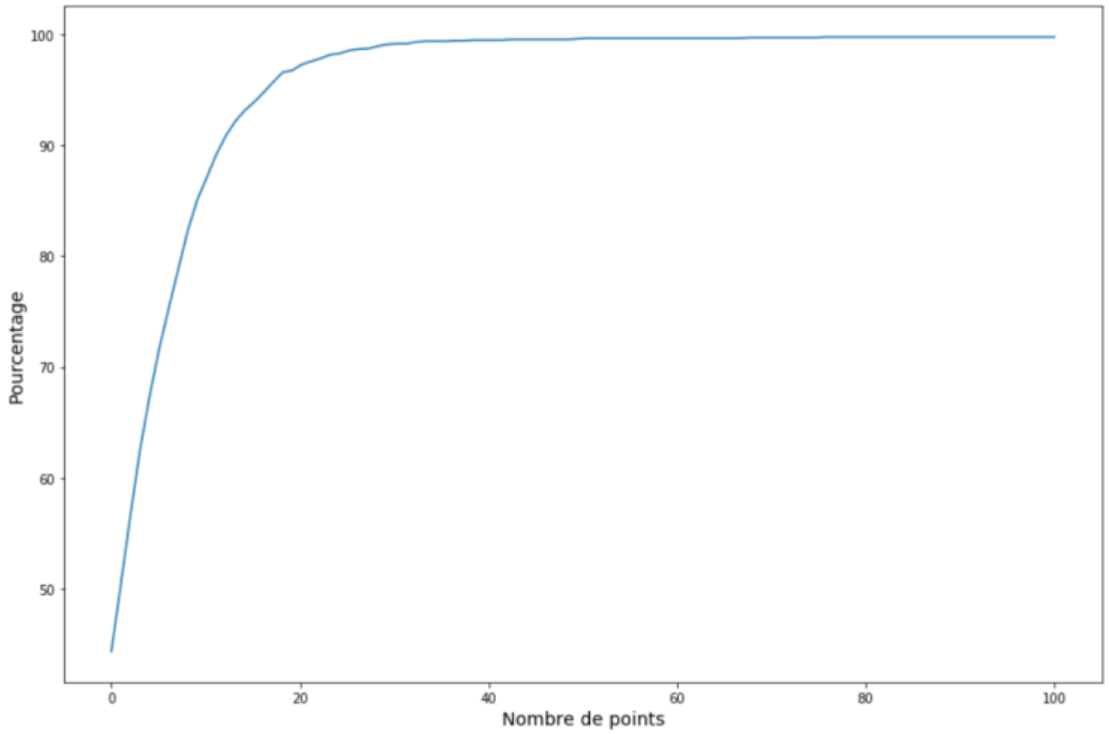

Figure 2.2 : Pourcentage des jours où le maximum (ou le minimum) du prix du DAX de l'IB est, à p points près, égal à celui de la journée entière.

On voit bien que, si on se permet 3 points d'erreur, ce pourcentage atteint quasiment 60%. On en déduit donc l'importance de de cette gamme initiale sur l'évolution des prix de la journée entière.

III- Mise en place de modèles de Machine Learning prédictifs en environnement Python

1. Classification des bougies à l'aide d'un réseau de neurones FFNN

L'objectif étant de prédire la nature de la bougie à l'instant t, en connaissant les informations du *tape* aux instants précédents, nous avons de manière naturelle mis en place un modèle de classification simple, le FFNN (Feed Forward Neural Network).

Explications théoriques :

Nous avons choisi de façon arbitraire les classes suivantes pour les bougies :

- Label $Y = 1$ si la bougie est haussière (prix de clôture à l'instant t supérieur à celui à l'instant t-1) ;
- Label $Y = -1$ si la bougie est baissière (prix de clôture à l'instant t inférieur à celui à l'instant t-1) ;
- Label $Y = 0$ si la bougie est constante (prix de clôture à l'instant t égal à celui à l'instant t-1).

L'objectif de la classification est de prédire le label $Y \in \{-1, 0, 1\}$ des données à partir des observations $X \in \mathbb{R}^d$, d étant le nombre de features considérées. En utilisant le dataset d'apprentissage $D_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$, on doit construire un classifieur minimisant la fonction de perte moyenne (le risque). Le meilleur classifieur au sens de cette fonction de perte à minimiser (**classifieur de Bayes**) est donc :

$$f^* = \underset{f}{\text{Argmin}}\{\mathbb{E}[\mathbb{1}_{Y \neq f(X)}]\} = \underset{m \in \{-1,0,1\}}{\text{Argmax}}\{\mathbb{P}(Y = m \mid X)\}$$

La solution explicite nécessite la connaissance de la loi conditionnelle de Y sachant X. C'est là qu'on fait le **choix du modèle semi-paramétrique de la régression softmax** (extension multi-classe du modèle de régression logistique) qui modélise la loi de Y sachant X de la façon suivante (M est le nombre de classes, ici M=3) :

$$\mathbb{P}(Y = m|X) = softmax(z)_m$$

$$softmax(z)_m = \frac{\exp(z_m)}{\sum_{j=1}^{M} \exp(z_j)}$$

$$z_m = < w_m, X > + b_m$$

Cette modélisation permet de **différencier les données non-linéairement séparables** à la différence du modèle paramétrique d'analyse discriminante dans lequel les frontières sont des hyperplans.

Chaque nœud du FFNN est un neurone. Il convient donc d'expliquer de manière non-exhaustive le fonctionnement d'un neurone à l'aide de la Figure 3.1.1 suivante :

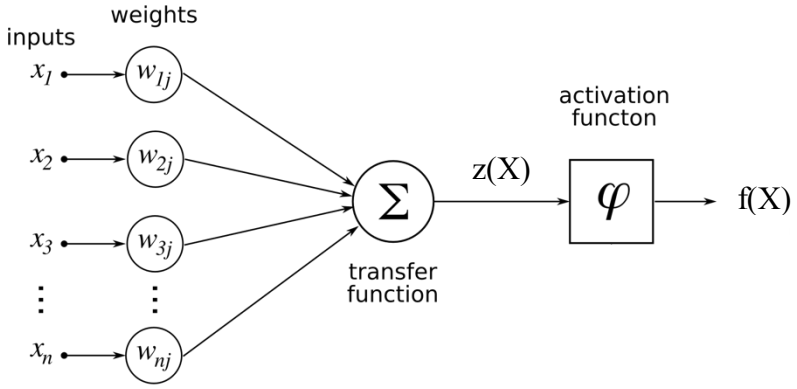

Figure 3.1.1 : Représentation schématique d'un neurone

On a donc en sortie une **transformation non-linéaire** (car fonction d'activation non-linéaire) d'une combinaison linéaire des valeurs d'entrée.

$$z(X) = W^T X + b \quad \text{et} \quad f(X) = \varphi(W^T X + b)$$

Si φ est la fonction softmax, le résultat renvoyé en sortie correspond au vecteur des probabilités de chaque classe. La classe prédite par le modèle correspondra à celle pour laquelle la probabilité sera la plus grande.

La structure du FFNN est la suivante (Figure 3.1.2) dans laquelle chaque couche est composée de plusieurs neurones :

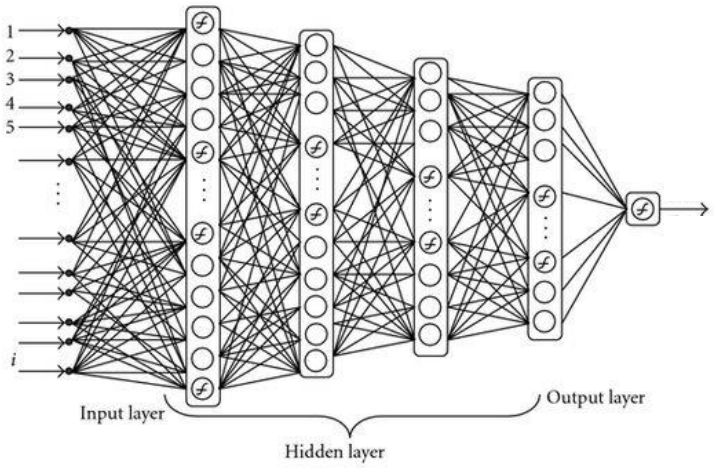

Figure 3.1.2 : Représentation schématique d'un réseau FFNN

La dernière fonction d'activation sera la fonction softmax et la sortie du réseau sera le vecteur des probabilités de chacune des 3 classes. Cette structure permet d'apprendre les paramètres grâce à une estimation par back-propagation.

Implémentation en Python :

Feed-Forward Neural Network

```
model_ffnn = Sequential()
model_ffnn.add(Dense(128, input_dim = 3, activation = 'relu', name = 'first_hidden_layer'))
model_ffnn.add(Dense(3,input_dim = 128, activation = 'softmax', name = 'dense_softmax'))

model_ffnn.compile(
                    loss=keras.losses.categorical_crossentropy,
                    optimizer=keras.optimizers.Adagrad(),
                    metrics=['accuracy']
)

model_ffnn.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
first_hidden_layer (Dense)   (None, 128)               512
_________________________________________________________________
dense_softmax (Dense)        (None, 3)                 387
=================================================================
Total params: 899
Trainable params: 899
Non-trainable params: 0
_________________________________________________________________
```

Figure3.1.3 : Mise en place d'un modèle FFNN avec 1 unique couche cachée

Le travail consiste à optimiser les hyper-paramètres du modèle, c'est-à-dire le nombre de couches cachées, les fonctions d'activation des couches cachées, le nombre de neurones, la taille du batch, nombre d'epochs, etc. Nous réalisons une méthode de recherche de la meilleure combinaison de paramètres appelée GridearchCV() qui permet de calculer l'erreur de l'apprentissage de chacun des modèles construits par l'ensemble des combinaisons de paramètres possible parmi :

- Batch_size : { 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024} ;
- Nb_neurones : { 64, 128, 256} ;
- Nb_couches_cachées : { 1, 2, 3, 4} ;

Nous obtenons les résultats suivants : le meilleur modèle au sens de la metrics 'accuracy' à maximiser est le modèle ayant pour hyper-paramètres : batch_size = 256, nb_neurones = 64 et nb_couches_cachées = 2

Résultats et interprétation :

```
err_class_ffnn = 0
for i in range(len(pred_ffnn)):
    if test_ffnn[i] != pred_ffnn[i]:
        err_class_ffnn += 1
print((err_class_ffnn/(len(pred_ffnn)))*100, '%')
```

```
29.819924004625804 %
```

Figure 3.1.4 : Erreur de classification du modèle FFNN

Nous obtenons une erreur de **29,82%**. Notons que l'erreur maximale d'un problème à 2 classes est 50%, mais que dans notre cas (3 classes), l'erreur aurait pu largement dépasser les 50% : on en déduit que le modèle n'est pas non plus un échec. Ce résultat constitue un benchmark à dépasser en mettant en place d'autres modélisations des données avec d'autres modèles. En effet, ce modèle très simple suppose que **les couples de données**

$(\boldsymbol{X_i, Y_i})$ **sont indépendants**, ce qui est évidemment faux dans le cas des données financières que nous avons. On ne peut pas affirmer que la valeur du prix de clôture à l'instant t est indépendante du prix de clôture à l'instant d'après.

Il est donc nécessaire de mettre en place un modèle dans lequel les données sont dépendantes les unes avec les autres. On met donc en place des **modèles récurrents**, ce qui nous amène à tester tout d'abord la mise en place d'un RNN simple toujours dans une optique de classification de la nature des bougies.

2. Classification avec un RNN :

Explications théoriques :

Ces réseaux sont récurrents, c'est-à-dire que la variable prédite à chaque instant t dépend non seulement des inputs à cet instant t mais aussi du passé, dont l'état caché conserve l'information.
A l'instant t, l'état caché du réseau est calculé de la manière suivante :

$$h_t = \sigma_h(W_x x_t + W_h h_{t-1} + b_h)$$

Où σ_h est une fonction d'activation non-linéaire, x_t est l'entrée à l'instant t et h_{t-1} est l'état caché à l'instant t-1. W_x, b_h *et* W_h sont les paramètres inconnus à estimer.
La prédiction s'obtient donc de la façon suivante :

$$\hat{y}_t = \sigma_y(W_y h_t + b_y)$$

Où σ_y est la fonction d'activation de sortie et W_y, b_y sont les paramètres inconnus de l'étape de prédiction.

Le schéma suivant explique ce processus récursif :

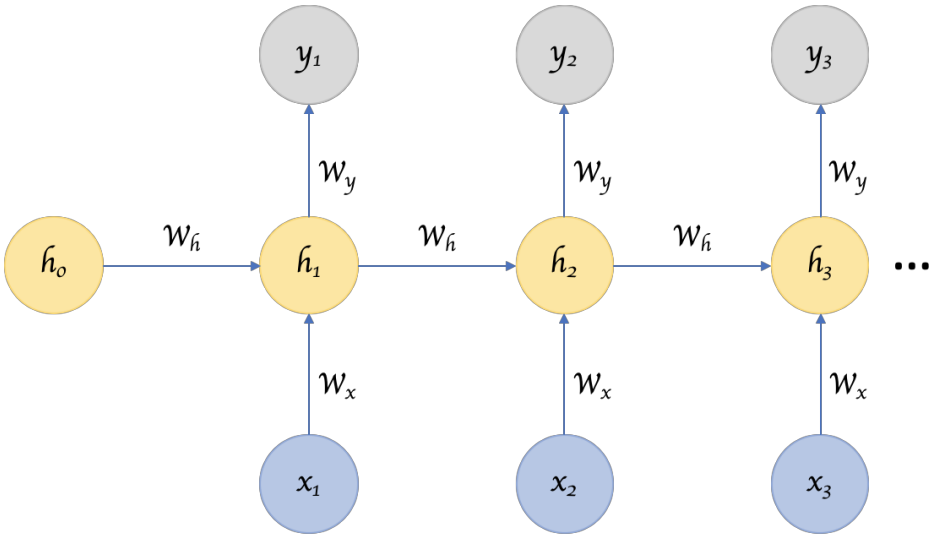

Figure 3.2.1: schéma explicatif du principe d'un RNN

Implémentation d'un RNN pour la classification :

De même que pour le réseau Feed Forward, nous allons ici chercher à optimiser le modèle que nous allons utiliser pour faire la classification. Dans le cas présent, nous allons chercher à optimiser le nombre de neurones et le nombre de pas de temps à considérer dans la variable d'entrée (que nous appellerons time_lag).

Pour mieux expliquer la notion de time_lag, nous allons exposer les exemples suivants :

- Time_lag = 0 : les données d'entrée sont constituées des features à l'instant t (privés de la variable à prédire qui est le label de la bougie).
- Time_lag = 1 : les données d'entrée sont constituées des features à l'instant t (privés de la variable à prédire) et de l'ensemble des features de l'instant t-1.
- Time_lag = 2 : les données d'entrée sont constituées des features à l'instant t (privés de la variable à prédire) et de l'ensemble des features de l'instant t-1 et t-2.

Une validation croisée, en utilisant le module GridsearchCV de Scikit Learn, dans laquelle nous faisons varier le nombre de neurones par couche et le time_lag nous indique que le meilleur modèle à considérer sera celui avec time_lag = 2 et rnn_dim = 256.

Nous avons donc implémenté notre réseau RNN avec deux couches RNN de dimension 256 et des variables d'entrée de time_lag=2.

```
model_rnn = Sequential()
model_rnn.add(SimpleRNN(256, input_shape = input_shape, return_sequences=True))
model_rnn.add(SimpleRNN(256, return_sequences=False))
model_rnn.add(Dense(3, activation='softmax'))
model_rnn.compile(
                  loss=keras.losses.categorical_crossentropy,
                  optimizer='adam',
                  metrics = ['accuracy'])
```

Figure 3.2.2 : Le modèle du réseau RNN implémenté

Après apprentissage du modèle, nous aboutissons à une accuracy de 0.79 :

```
score_rnn = model_rnn.evaluate(X_train_rnn, Y_train_rnn, verbose = 1)
print('Test loss :', score_rnn[0])
print('Test accuracy :', score_rnn[1])
```

```
24211/24211 [==============================] - 1s 34us/step
Test loss : 0.530904457779958
Test accuracy : 0.793275773525238
```

Figure 3.2.3 : la valeur de l'accuracy obtenue en utilisant une classification par un RNN

Nous appliquons donc la prédiction sur les données de test, puis nous créons deux listes *test_rnn* (contenant les labels réels) et *pred_rnn* (contenant les labels prédits).Le taux d'erreur de classification est donc calculé en comparant ces deux listes. Nous obtenons un taux d'erreur égal à **23,53 %**.

```
err_class_rnn = 0
for i in range(len(pred_rnn)):
    if test_rnn[i] != pred_rnn[i]:
        err_class_rnn += 1
print("Le teux d'erreur est égal à ",(err_class_rnn/(len(pred_rnn)))*100, '%')
```

```
Le teux d'erreur est égal à  23.533300280945298 %
```

Figure 3.2.4 : Taux d'erreur obtenu en utilisant une classification par un RNN

3. Modèle de classification postérieure à la prédiction d'une variable continue à l'aide d'un modèle de régression par un réseau récurrent : le LSTM

Nous avons eu l'idée de prédire une variable continue, le prix de clôture (le Last), et à partir de cette prédiction, de construire une fonction de classification simple comme suit (hat signifie chapeau, c'est ce qu'on met ordinairement pour signifier que la variable associée est la valeur prédite de cette variable) :
Si $Last\_hat(t) > Last\_hat(t-1)$, alors la classe prédite vaut $Y_{hat} = 1$ (bougie haussière) ;
Si $Last_{hat(t)} < Last\_hat(t-1)$, alors la classe prédite vaut $Y_{hat} = -1$ (bougie baissière) ;
Si $Last_{hat(t)} = Last\_hat(t-1)$, alors la classe prédite vaut $Y_{hat} = 0$ (bougie constante) ;

a) Prédiction à l'aide d'un modèle LSTM

Le LSTM (Long Short-Term Memory) est un réseau récurrent particulièrement bien adapté au traitement de données séquentielles comme les séries temporelles en finance. En effet, ce sont des réseaux de neurones à mémoire interne qui se souviennent des entrées aux instants précédents ? Selon Lex Fridman, professeur au MIT, il faut utiliser un LSTM « chaque fois qu'il y a une séquence de données et que la dynamique temporelle qui relie les données est plus importante que le contenu spatial de chaque trame individuelle ». Ici on cherche en effet à capter la tendance, la dynamique de l'évolution des prix de clôture, et nous voulons capter la dynamique des prix car c'est à partir de ces évolutions à dérivées positives ou négatives que la classification se fera. Ce modèle parait donc parfaitement adapté à nos données. Voici sans trop rentrer dans le détail la structure d'un LSTM :

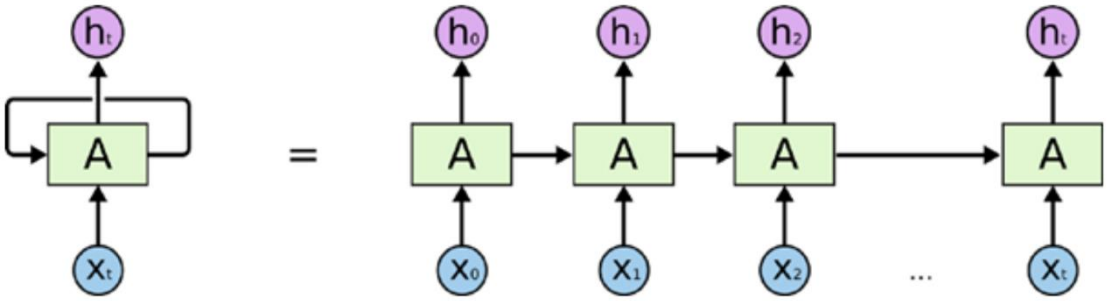

Figure 3.3.1 : Structure explicative d'un modèle LSTM

Recherche des meilleurs paramètres du modèle :

Comme nous pouvons considérer avec le LSTM que la prédiction à chaque instant t résulte des inputs à l'instant t, mais aussi des valeurs en mémoire de quelques instants précédents, il convient de faire la recherche des paramètres optimaux du modèle pour chaque pas de temps considéré (time_lag). Voici les résultats de la méthode GridSearchCV pour chercher à optimiser la dimension de la couche cachée :

```
For time_lag = 2 :
Best parameters set found on development set for a time lag equal to 2:

{'lstm_dim': 64}

Grid scores on development set:

-0.003 (+/-0.002) for {'lstm_dim': 4}
-0.003 (+/-0.001) for {'lstm_dim': 8}
-0.002 (+/-0.001) for {'lstm_dim': 16}
-0.003 (+/-0.003) for {'lstm_dim': 32}
-0.002 (+/-0.001) for {'lstm_dim': 64}
-0.003 (+/-0.003) for {'lstm_dim': 128}

==========================================

For time_lag = 3 :
Best parameters set found on development set for a time lag equal to 3:

{'lstm_dim': 128}

Grid scores on development set:

-0.003 (+/-0.001) for {'lstm_dim': 4}
-0.003 (+/-0.001) for {'lstm_dim': 8}
-0.003 (+/-0.000) for {'lstm_dim': 16}
-0.002 (+/-0.001) for {'lstm_dim': 32}
-0.003 (+/-0.002) for {'lstm_dim': 64}
-0.002 (+/-0.001) for {'lstm_dim': 128}

==========================================
```

Figure 3.3.2 : Dimension optimale de la couche cachée pour différents time_lag

Cette méthode a également permis d'optimiser les autres paramètres pour finalement obtenir un LSTM définit comme suit : 2 couches cachées, un time lag de 2 unités de temps, une dimension de 64 pour les couches cachées, une taille du batch de 32. Ce modèle nous donne les résultats de régression suivants, ce qui donne une **erreur de classification de 57,4%** :

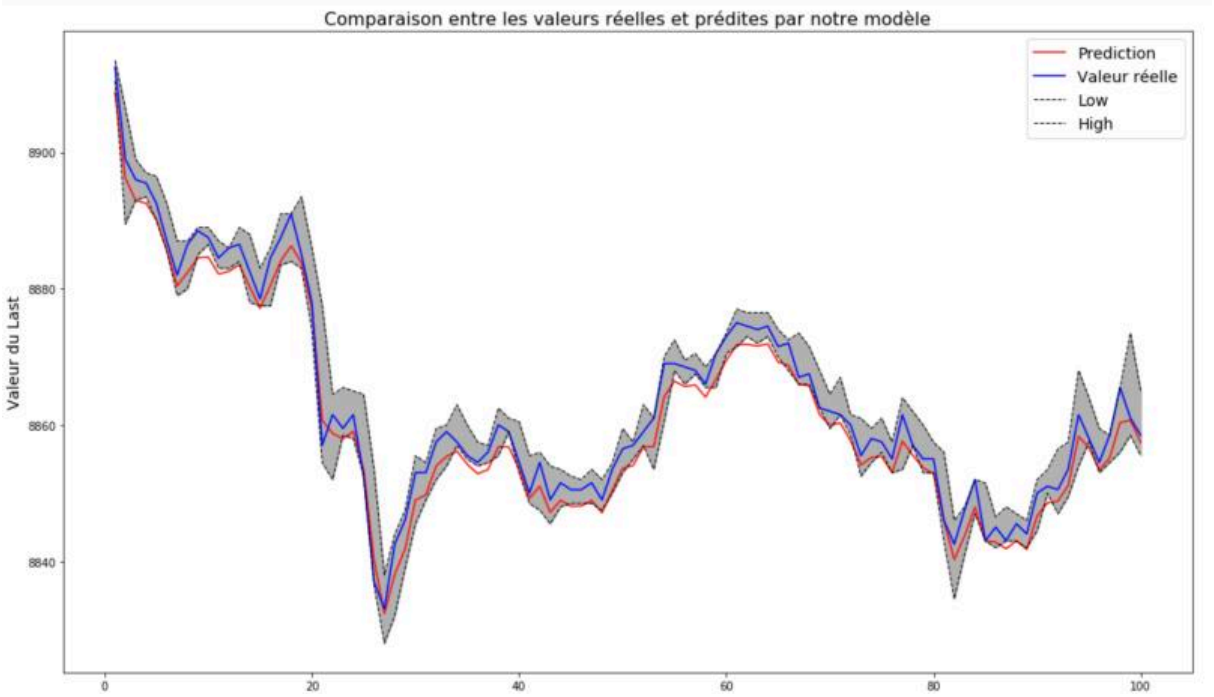

Figure 3.3.3 : Résultat du modèle LSTM

b) Correction de l'erreur de régression

Principe de la méthode :
Nous avons vu à l'étape précédente que la prédiction de la valeur du Last ne se fait pas de manière parfaite et induit donc des erreurs de classifications. Nous pouvons donc écrire :

$$\boldsymbol{Last(t) = Last_{hat}(t) + \varepsilon(t)}$$

Il s'agit dans cette partie d'apprendre l'erreur $\varepsilon(t)$ pour essayer d'améliorer notre modèle de prédiction en vue d'effectuer une meilleure classification.
Nous avons donc considéré la fonction d'erreur $\boldsymbol{\varepsilon(t) = Last(t) - Last_{hat}(t)}$ qui sera calculée sur les données de test de la première prédiction (car c'est pour ces données-ci que nous avons fait le calcul de $Last_{hat}(t)$).
Nous avons subdivisé notre échantillon, comme dans l'étape précédente, en deux parties : une pour l'apprentissage et une pour le test, tout en prenant cette fois-ci la fonction d'erreur $\boldsymbol{\varepsilon(t)}$ comme variable de sortie du LSTM.

Résultat :
Après apprentissage de $\boldsymbol{\varepsilon}$, l'erreur de classification passe à $\mathbf{20\%}$, elle a donc diminué.
Voici le résultat de prédiction que nous avons obtenu :

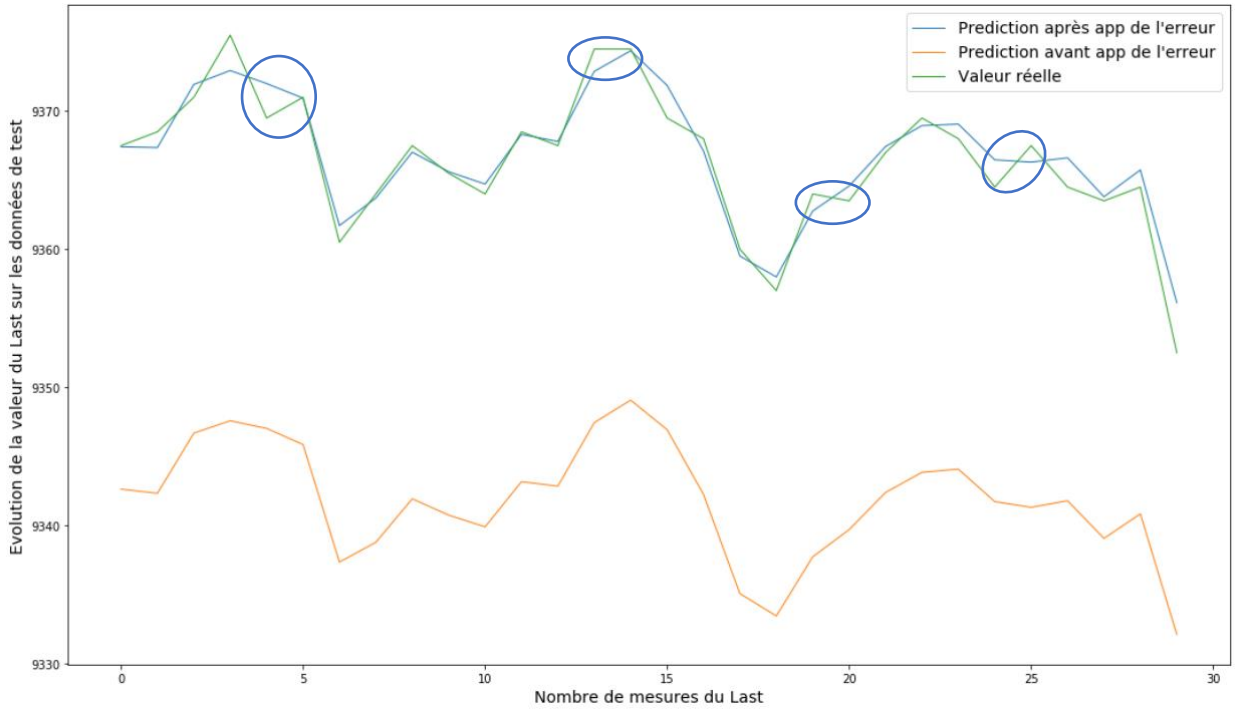

Figure 3.3.4: les courbes des valeurs réelles et prédites (avant et après apprentissage de l'erreur).

N.B : Les parties encerclées de la figure correspondent aux endroits où se produit une erreur de classification.
Ce résultat reste perfectible. Nous avons, donc, refait l'apprentissage en rajoutant des variables explicatives additionnelles au vecteur d'entrée, le but étant de rajouter de l'information sur les marchés et sur les transactions effectuées par les acteurs financiers.

c) Ajout d'une variable explicative

Puis, nous avons réalisé le même modèle en ajoutant 1 variable à ces 4 variables déjà utilisées que sont l'Open, le High, le Low et le Last pour l'apprentissage et la correction de l'erreur. Ainsi, nous pouvons **analyser l'impact de chaque variable/indicateur un à un et détecter ceux qui améliorent la classification** (ie : qui entrainent une diminution de l'erreur de classification). Cette logique combinatoire nous a amené à entraîner de nombreux modèles, dont voici un exemple : l'ajout de la variable Volume.
Ici, le pré-processing de la variable volume est réalisé avec un z-score dont la formule est la suivante : $z_{score(Volume)}(t) = \frac{Volume(t)-\mu_{23}(t)}{\sigma_{23}(t)}$
Avec $\mu_{23}(t)$ la moyenne mobile des volumes sur les 23 instants précédant t et $\sigma_{23}(t)$ l'écart-type des volumes sur les 23 instants précédant t. Nous obtenons alors la prédiction suivante de la valeur du Last (régression) :

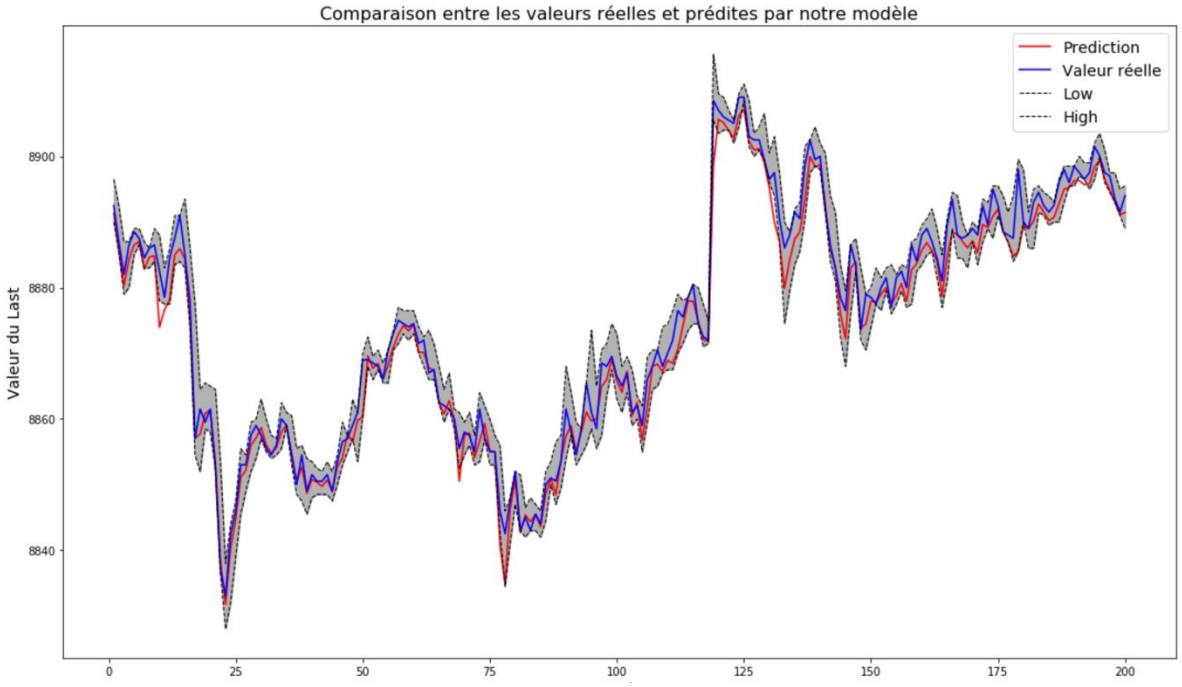

Figure 3.3.5 : Prédiction de la valeur du Last à l'aide du modèle

La classification réalisée à partir des valeurs prédites du Last nous donne un taux d'erreur de **26,65%**. Ensuite, nous réalisons l'apprentissage de l'erreur commise à l'aide des mêmes features d'apprentissage, c'est-à-dire l'Open, le High, le Low et le Volume et nous réalisons un modèle prédictif sur d'autres valeurs non apprises par le modèle, ce qui nous donne le résultat de régression suivant :

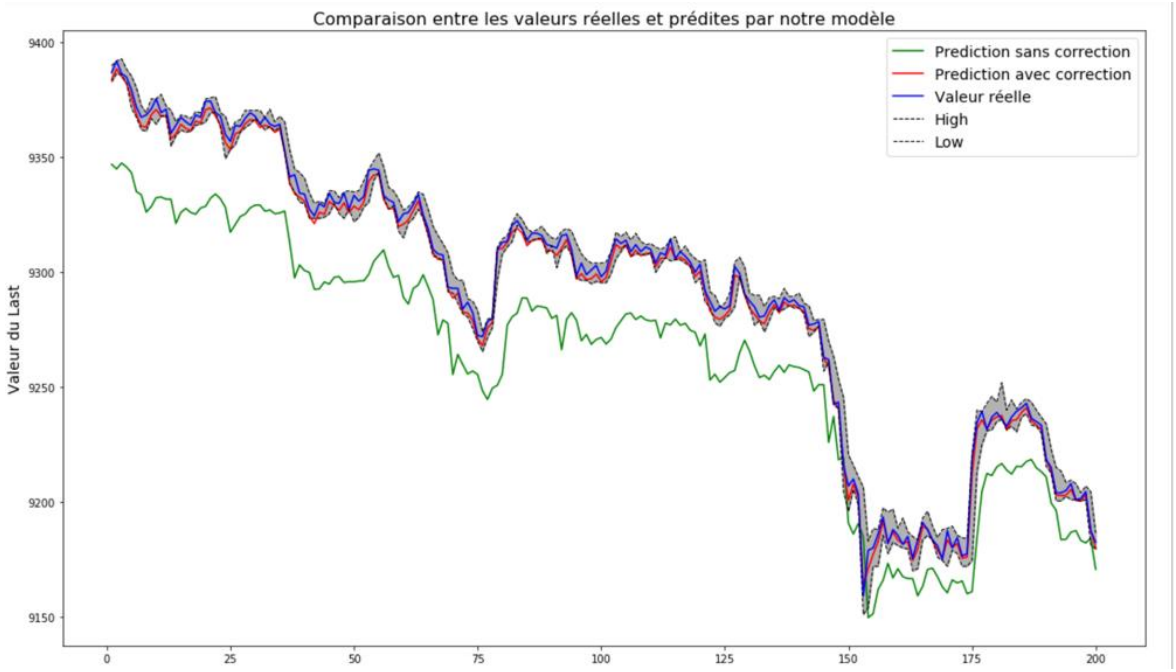

Figure 3.3.6 : Prédiction de la valeur du Last corrigée

L'erreur de classification post-régression nous donne alors **13,86%**. On peut donc noter une amélioration notable de l'erreur de classification après apprentissage de l'erreur. Si de plus, nous ne comptabilisons pas les erreurs de classification sur les bougies constantes, l'erreur tombe à **8,37%**. Effectivement, on peut se le permettre car prendre une position short ou long (à la vente ou à l'achat), alors que le prix futur n'aura ni augmenté ni baissé, n'a aucun impact sur le portefeuille du trader.

d) Ajout de couples de variables additionnelles

Dans cette étape nous avons testé l'ajout des couples de variables et indicateurs explicatifs. Le meilleur taux d'erreur de classification obtenu dans ce cas est **15,63%**. Voici un exemple des résultats de prédiction que nous avons obtenus :

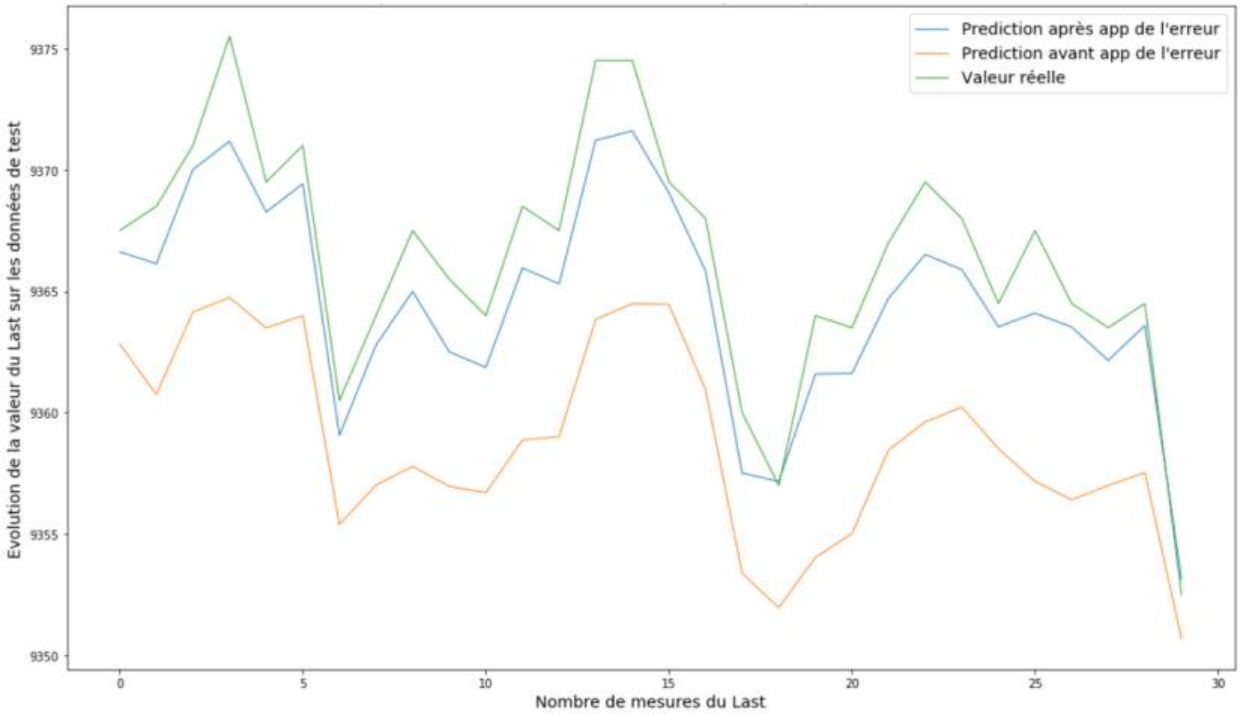

Figure 3.3.7 : les courbes des valeurs réelles et prédites (avant et après apprentissage de l'erreur) en rajoutant les deux variables 'Volume' et 'priceChange'.

Nous remarquons bien que la courbe de prédiction après apprentissage de l'erreur en rajoutant un couple de variables (en bleu) suit mieux les variations de la courbe des valeurs réelles (en vert) que dans le cas de l'apprentissage sans ajout de variables explicatives, d'où une erreur de classification plus faible (elle est passée de 20%, à **15,63%**).

Toutefois, il ne faut pas perdre de vue que l'ajout d'un couple de ces variables/indicateurs donne des résultats moins satisfaisants que l'ajout d'une seule variable (l'erreur de classification passe de 13% à 15,63%).

IV- Conclusion :

A travers notre projet Cassiopée, nous avons implémenté plusieurs modèles de classification des tendances du prix du DAX. Nous avons pu comparer les performances de chacun de ces modèles et comprendre le rôle important que joue la prise en compte des volumes des transactions.

Voici, en conclusion, un tableau récapitulatif des résultats que nous avons obtenus :

Modèle	Classification à l'aide d'un FFNN	Classification à l'aide d'un RNN	Classification après prédiction du Last (sans ajout de variables explicatives)	Classification après prédiction du Last (en ajoutant une seule variable explicative)	Classification après prédiction du Last (en ajoutant un couple de variables explicatives)
Taux d'erreur de classification	29,82%	23,53%	20%	13,86%	15,63%