Trading algorithmique sur cryptomonnaies

Infrastructure et modélisation

TD FIN 04

Cynthia Wang & Benjamin Wu

Sommaire

I) Qu'est-ce que le trading algorithmique 3

Définition 3

Histoire 3

Utilisation 3

II) Relative Strength Index (RSI) 4

Histoire 4

Utilisation 4

III) Bot de trading 5

Fonctionnement 5

Turtle trading et modélisation 7

Résultats 9

IV) Gestion du projet 10

Déroulé du projet / Répartition des tâches 10

Améliorations et perspectives futures 10

V) Conclusion 10

VI) Annexes 11

I) Qu'est-ce que le trading algorithmique

1. Définition

Le trading algorithmique est une forme de trading utilisant un algorithme saisissant des ordres de bourses sur des plateformes électroniques. L'algorithme va décider les différents aspects de l'ordre comme le moment d'ouverture, de clôture, le prix et le volume de l'ordre sans avoir besoin d'intervention humaine (dans la plupart des cas).

2. Histoire

L'informatisation du flux d'ordres sur les marchés financiers a commencé au début des années 1970, lorsque la Bourse de New York a introduit le système de «rotation des ordres désignés» (DOT). SuperDOT a été introduit en 1984 en tant que version améliorée de DOT. Les deux systèmes permettaient l'acheminement des ordres par voie électronique vers le poste de trade approprié.
Aux États-Unis, la décimalisation a fait passer la taille minimale des ticks de 1/16 de dollar (0,0625 USD) à 0,01 USD par action en 2001, et peut avoir encouragé le trading algorithmique car elle a changé la microstructure du marché en permettant des différences plus faibles entre l'offre et l'offre prix, diminuant l'avantage commercial des teneurs de marché, augmentant ainsi la liquidité du marché. Cette liquidité accrue du marché a conduit les traders institutionnels à fractionner les ordres selon des algorithmes informatiques afin qu'ils puissent exécuter des ordres à un meilleur prix moyen. Ces prix de référence moyens sont mesurés et calculés par ordinateur en appliquant le prix moyen pondéré dans le temps ou plus généralement par le prix moyen pondéré en fonction du volume.
Au fur et à mesure du temps, d'autres stratégies de trading algorithmiques ont été introduites. Ces stratégies sont plus facilement mises en œuvre par les ordinateurs, car les machines peuvent réagir plus rapidement à des erreurs temporaires de prix et examiner les prix de plusieurs marchés simultanément. L'utilisation des robots de trading algorithmique a ainsi augmenté en conséquence (proportion d'ordres passés sur les marchés boursiers via des robots de trading algorithmique 30 à 40% en 2006 contre 70 à 90 % depuis 2010).

3. Utilisation

Aujourd'hui le trading algorithmique se compose de deux activités : les opérations de bourse assistées par des algorithmes qui anticipent et favorisent les opportunités de bénéfices, et le trading automatisé qui utilise des automates comme agents autonomes effectuant des transactions selon des algorithmes et des stratégies paramétrées.

On retrouve, parmi les différentes catégories de trading algorithmique, le trading haute fréquence (HFT) dans lequel le rythme de création d'ordre est du nanoseconde et celui de passation du microseconde. Le HFT remplace les market makers traditionnels (personnes physiques) en apportant plus de liquidité au marché avec un prix moyen des transactions à la baisse.

II) Relative Strength Index (RSI)

1. Histoire

La stratégie choisie lors de la conception de notre bot de trading algorithmique est une stratégie basée sur l'indicateur RSI. Le RSI ou relative strength index est un indicateur avancé d'analyse technique proposé par Welles Wilder en 1978. Celui-ci reflète la force relative des mouvements haussiers en comparaison aux mouvements baissiers. Cet indicateur technique est utilisé par les traders pour jauger la force d'une tendance et repérer des signes de fin de tendance. L'indicateur RSI mesure le rapport des mouvements ascendants et descendants et normalise le calcul afin que son indice soit exprimé dans une plage de 0 à 100. Il permet d'indiquer à un investisseur le meilleur moment pour acheter lorsqu'il considère un marché en survente et le meilleur moment pour vendre lorsqu'il considère un marché en surachat.

2. Utilisation

Comme nous l'avons évoqué un peu plus tôt, l'indice RSI est exprimé sur une plage de 0 à 100. Notons également que lorsque le marché est très régulièrement en hausse ou en forte hausse, le RSI tend vers 100 et inversement lorsque le marché est en forte de baisse, le RSI tend vers 0. L'utilisation du RSI se base sur la décision qui découle de cet indice. En effet, on dit que le marché est en surachat lorsque le RSI est supérieur à environ 70, en d'autres mots il est sujet à une correction baissière. Au contraire, le marché est dit survente lorsque le RSI est inférieur à 30, il est ainsi candidat à une correction à la hausse. De plus, il est important de prendre en compte que par défaut la longueur du RSI est définie à 14, et son amplitude dépend alors de l'unité de temps du graphique choisie. En d'autres termes, sur un graphique de 1 minute, la dernière valeur du RSI prendra en compte les 14 dernières minutes. De même que sur un graphique d'un jour, il prendra en compte les 14 derniers jours.

III) Bot de trading

1. Fonctionnement

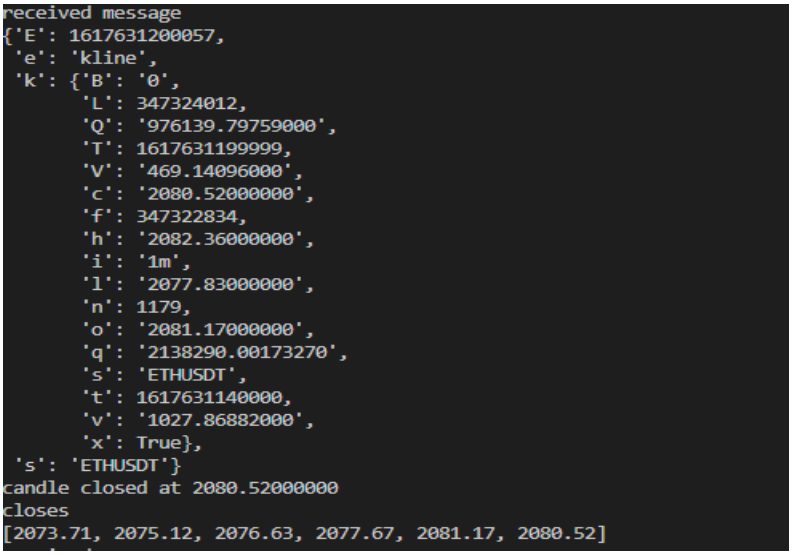
Nous avons fait le choix de coder notre bot en langage python. Afin que notre bot puisse fonctionner, nous avons dans un premier temps eu besoin d'installer plusieurs bibliothèques telles que la bibliothèque Binance et websocket pour permettre la connexion à l'API binance. La bibliothèque ta-lib afin d'obtenir les différents indicateurs financiers comme le RSI, l'indice sur lequel se base la stratégie de notre bot, et enfin des bibliothèques classiques comme numpy et json afin de traiter les données. Une fois ces installations faites, la première étape est la connexion sur le site binance afin de pouvoir récupérer les données du site comme le prix du cours ou le prix à l'ouverture, pour cela nous avons besoin du protocole de communications utilisé par binance et créer un websocket qui permettra la communication.

```
SOCKET = "wss://stream.binance.com:9443/ws/ethusdt@kline_1m"
ws = websocket.WebSocketApp(SOCKET, on_open=on_open, on_close=on_close, on_message=on_message)
ws.run_forever()
```

Lorsque la connexion avec l'API binance a été établie, la prochaine étape de notre programme a été de traduire le "stream" de données reçues et de l'afficher afin de pouvoir procéder à une analyse de celle-ci. En cherchant sur le github de binance nous avons réussi à avoir le stream de donnée ci-dessous :

```
{
  "e": "kline",     // Event type
  "E": 123456789,   // Event time
  "s": "BNBBTC",    // Symbol
  "k": {
    "t": 123400000, // Kline start time
    "T": 123460000, // Kline close time
    "s": "BNBBTC",  // Symbol
    "i": "1m",      // Interval
    "f": 100,       // First trade ID
    "L": 200,       // Last trade ID
    "o": "0.0010",  // Open price
    "c": "0.0020",  // Close price
    "h": "0.0025",  // High price
    "l": "0.0015",  // Low price
    "v": "1000",    // Base asset volume
    "n": 100,       // Number of trades
    "x": false,     // Is this kline closed?
    "q": "1.0000",  // Quote asset volume
    "V": "500",     // Taker buy base asset volume
    "Q": "0.500",   // Taker buy quote asset volume
    "B": "123456"   // Ignore
  }
}
```

Lorsque nous exécutons le programme, nos données ressemble alors à :



Avec ces données, notre but à présent a été de vouloir placer des ordres d'achat et de vente sur le site binance. Pour cela, nous avions besoin d'une stratégie afin d'indiquer les conditions permettant à l'algorithme de savoir quand est ce qu'il faut acheter et quand est ce qu'il faut vendre une position. Nous l'avons évoqué un peu plus tôt dans le rapport, la stratégie sur laquelle se base notre bot utilise l'indicateur RSI. Notons que cet indicateur est présent en python dans la bibliothèque TA-lib (Thechnical Analysis librairy) et s'utilise en prenant un tableau de prix de clôtures ainsi qu'une période.

```
rsi = talib.RSI(np_closes, RSI_PERIOD)
```

Une fois l'indice stocké, nous la comparons simplement avec la limite que nous avons définie au début qui représente les conditions d'un marché en surachat ou en survente. Ici, nous avons choisi 70 comme valeur indiquant un marché en surachat et 30 pour un marché en survente. Lorsque les conditions sont vérifiées, notre fonction "order" préparera l'ordre et l'enverra vers l'API du site binance.

```
def order(side, quantity, symbol,order_type=ORDER_TYPE_MARKET):
    try:
        print("sending order")
        order = client.create_order(symbol=symbol, side=side, type=order_type, quantity=quantity)
        print(order)
    except Exception as e:
        print("an exception occured - {}".format(e))
        return False

    return True
```

Malheureusement nous n'avons pas réussi à faire passer des ordres en fictif, il fallait alors se connecter à un compte binance afin de poursuivre nos opérations. C'est pourquoi nous avons décidé de pivoter notre thématique de base qui visait la connexion et le passage d'ordre sur binance à une nouvelle thématique où cette fois ci nous visons à l'analyse des données que nous récupérons du site afin de la modéliser sur un graphique qui indique les moments les différents moments de vente et d'achat.

2. Turtle trading et modélisation

Afin de pouvoir modéliser les moments d'achat et de vente, nous avons choisi d'utiliser la méthode du turtle trading.

a. Turtle trading

La méthode du turtle trading provient d'une expérience menée au début des années 1980 par Richard Dennis et William Eckhardt. Cette expérience provient de l'idée de Richard Dennis qui voulait produire rapidement et efficacement des traders de la même manière qu'il avait vu les éleveurs de tortues à Singapour efficacement et rapidement créer des tortues.

Son expérience consistait à prendre un groupe aléatoire de personnes, à leur enseigner un ensemble de règles à suivre et à voir dans quelle mesure ils avaient négocié avec succès.
L'issue de cette expérience est un énorme succès car avec un investissement initial de 23 millions de dollars, ils obtiennent un bénéfice combiné de 175 millions de dollars, soit le montant multiplié par plus de 7. C'est donc devenu l'une des plus célèbres expériences de l'histoire du Trading.

b. Modélisation moment d'achat et de vente

Après avoir récupéré les données du site binance, pour pouvoir les traiter, nous les avons d'abord entrées dans un fichier csv en choisissant la plage des dates et l'intervalle entre chaque donnée.

```
client = Client(config.API_KEY, config.API_SECRET)

csvfile = open('coinview/2020_1DAY.csv', 'w', newline='')
fieldname = ['X', 'Open','High','Low','Close','X1','X2','X3','X4','X5','X6','X7']
candlestick_writer = csv.writer(csvfile,delimiter=',')

candlesticks = client.get_historical_klines("ETHUSDT", Client.KLINE_INTERVAL_1DAY, "1 Jan, 2021", "5 Avr, 2021")

candlestick_writer.writerow(fieldname)
for candlestick in  candlesticks:
    candlestick[0] = candlestick[0] / 1000
    candlestick_writer.writerow(candlestick)
csvfile.close()


sns.set()
df = pd.read_csv('2020_1DAY.csv')
df.head()
count = int(np.ceil(len(df) * 0.1)) #26
signals = pd.DataFrame(index=df.index)
signals['signal'] = 0.0
signals['trend'] = df['Close']
signals['RollingMax'] = (signals.trend.shift(1).rolling(count).max())
signals['RollingMin'] = (signals.trend.shift(1).rolling(count).min())
signals.loc[signals['RollingMax'] < signals.trend, 'signal'] = -1
signals.loc[signals['RollingMin'] > signals.trend, 'signal'] = 1
signals
```

Nous ouvrons ensuite le fichier csv et créons une dataframe "signals" avec les données souhaitées (dans notre cas, nous prenons le prix de clôture pour la simulation).

Explication des différents index :

- ‘signal’ correspond au potentiel moment d’achat ou de vente mis à 0 par défaut.
- ‘trend’ correspond au prix de clôture
- ‘RollingMax’ le prix maximum sur les 26 jours passés
- ‘RollingMin’ le prix minimum sur 26 jours passés

Le ‘signal’ change ensuite de valeur en fonction de la comparaison entre ‘RollingMax’ (respectivement ’RollingMin’) et ‘trend’. ‘signal’ pourra donc avoir comme valeur -1 , 0 , 1.

```
def buy(i, initial_money, current_inventory):
    shares = initial_money // real_movement[i]
    if shares < 1:
        print(
            'day %d: total balances %f, not enough money to buy a unit price %f'
            % (i, initial_money, real_movement[i])
        )
    else:
        if shares > max_buy:
            buy_units = max_buy
        else:
            buy_units = shares
        initial_money -= buy_units * real_movement[i]
        current_inventory += buy_units
        print(
            'day %d: buy %d units at price %f, total balance %f'
            % (i, buy_units, buy_units * real_movement[i], initial_money)
        )
        states_buy.append(0)
    return initial_money, current_inventory
```

Ensuite, nous avons une fonction buy permettant “d’acheter” une unité, à l’issue de cette fonction, nous récupérons le stock et la trésorerie actuelle une fois l’achat effectué.

Nous prenons aussi en compte le cas où la trésorerie est insuffisante pour acheter une unité.

Lorsque ‘state’ (qui prend les valeurs de ‘signal’) vaut 1 on a une position d’achat et -1 de vente. Dans la position de vente nous prenons en compte le cas où nous n’avons pas d’unité dans l’inventaire mais une position de vente est quand même émise. Sinon la vente peut se passer et nous décrémentons l’inventaire, nous ajoutons donc la somme perçue dans la trésorerie et récupérons le jour de la transaction dans ‘states_sell’. Pour la position d’achat, nous appelons la fonction buy et récupérons le jour de la transaction dans ‘states_buy’.

```
for i in range(real_movement.shape[0] - int(0.025 * len(df))):
    state = signal[i]
    if state == 1:
        initial_money, current_inventory = buy(i, initial_money, current_inventory)
        states_buy.append(i)
    elif state == -1:
        if current_inventory == 0:
                print('day %d: cannot sell anything, inventory 0' % (i))
        else:
            if current_inventory > max_sell:
                sell_units = max_sell
            else:
                sell_units = current_inventory
            current_inventory -= sell_units
            total_sell = sell_units * real_movement[i]
            initial_money += total_sell
            try:
                invest = ((real_movement[i] - real_movement[states_buy[-1]]) / real_movement[states_buy[-1]]) *100
            except:
                invest = 0
            print('day %d, sell %d units at price %f, investment %f %%, total balance %f,'
                % (i, sell_units, total_sell, invest, initial_money) )
        states_sell.append(i)

invest = ((initial_money - starting_money) / starting_money) * 100
total_gains = initial_money - starting_money
return states_buy, states_sell, total_gains, invest
```

3. Résultats

En affichant le code nous obtenons le graphique suivant :

La courbe rouge représente celle de l'ETHUSDT avec un intervalle d'un jour entre chaque donnée récupérée du 1er janvier 2021 au 5 avril 2021.

Nous pouvons remarquer que dans le graphique il y a plus de vente que d'achat, cet aspect est pris en compte dans notre modélisation comme expliqué précédemment, nous ne pouvons pas vendre lorsque nous n'avons pas d'unité.

```
day 80: buy 1 units at price 1680.970000, total balance 9620.590000
day 81: buy 1 units at price 1668.080000, total balance 7952.510000
day 82: buy 1 units at price 1581.840000, total balance 6370.670000
day 87, sell 1 units at price 1816.740000, investment 14.849795 %, total balance 8187.410000,
day 88, sell 1 units at price 1840.460000, investment 16.349315 %, total balance 10027.870000,
day 89, sell 1 units at price 1919.370000, investment 21.337809 %, total balance 11947.240000,
day 90: cannot sell anything, inventory 0
day 91: cannot sell anything, inventory 0
```

IV) Gestion du projet

1. Déroulé du projet / Répartition des tâches

Avec la situation sanitaire actuelle, nous n'avons malheureusement pas pu nous réunir afin de travailler en groupe sur ce projet mais seulement à distance. C'est pourquoi nous avons décidé de procéder dans un premier temps à des recherches chacun de notre côté afin de pouvoir gagner le plus de temps et par le biais de plusieurs réunions par semaine, pouvoir mettre en commun les connaissances acquises au fil de nos recherches. Une fois bien informé sur notre sujet, l'étape suivante a été d'implémenter le code afin de créer notre bot de trading. Comme vous avez pu le remarquer un peu plus tôt dans ce rapport, nous avons fait le choix de séparer en deux parties ce projet. La première partie étant l'implémentation du bot afin que celui-ci puisse se connecter à un site de trading et d'y passer des ordres tout en récupérant des flux de données. Dans un deuxième temps, nous avons décidé de saisir les données récupérées du flux afin de les analyser et de les reporter sur un graphique explicite qui met en évidence les moments où le marché est en survente ou en surachat et ainsi montrer les moments où il était judicieux de passer des ordres d'achat ou de vente.
J'ai (Benjamin) décidé de commencer en prenant en main l'implémentation du bot de trading, puis Cynthia a repris le projet afin d'y implémenter la partie modélisation des courbes. Bien que cela soit un projet assez complexe, nous avons malencontreusement consacré que les deux dernières semaines sur ce projet, ceci étant dû au fait qu'il était en parallèle de nos examens de fin d'année ainsi que des rendus et soutenance PPE. Ce qui a constitué l'une des plus grandes difficultés de ce projet en plus des difficultés de compréhension des articles et codes fournis sur le github.

2. Améliorations et perspectives futures

Ce projet étant très complexe mais très ouvert à la fois, de nombreuses possibilités d'amélioration sont envisageables. A travers ce projet, nous nous sommes seulement intéressés à la connexion du bot vers un site de trading de crypto monnaie en faisant appel à une stratégie de base et assez simpliste. Or il serait trop simple de considérer pouvoir trader et à terme gagner de l'argent avec seulement une stratégie implémentée dans notre algorithme. Il serait alors intéressant de voir si nous pouvions additionner plusieurs indicateurs ou mixer plusieurs stratégies et rendre notre bot plus "intelligent" ou plus performant. De plus, si nous avions eu plus de temps, nous aurions pu améliorer le bot de la première partie et réussir à placer des ordres sur la plateforme binance.
Concernant la deuxième partie du projet, où nous faisons une modélisation des moments de placement d'ordre, nous pouvons envisager une amélioration au niveau des transactions en augmentant la quantité d'unité par transaction. Dans notre cas, l'agent n'est capable d'acheter ou vendre maximum qu'une seule unité par transaction.

V) Conclusion

En conclusion, nous avons trouvé ce projet vraiment très intéressant et enrichissant, il nous a énormément appris concernant la connexion à une API mais également à l'implémentation en code de stratégie de trading tout en renforçant nos connaissances en python. De plus, les connaissances développer au cours de ce projet pourront dans le future nous aider à créer un bot plus performant que celui-ci, sur lequel nous pourrons peut-être gagner de l'argent. Ce projet nous a permis de développer le travail en équipe à distance dû à la situation sanitaire actuelle et met de nouveau en évidence les difficultés d'une telle situation. Enfin, nous souhaitons à terme reprendre ce sujet et repartir de ce bot afin d'apporter d'éventuelles améliorations quant à sa stratégie lorsque nous aurons acquis de nouvelles compétences dans ce domaine.

VI) Annexes

- https://en.wikipedia.org/wiki/Algorithmic_trading
- https://actufinance.fr/trading/robots-trading/
- https://en.wikipedia.org/wiki/Automated_trading_system
- GitHub - huseinzol05/Stock-Prediction-Models
- https://www.youtube.com/watch?v=GdlFhF6gjKo
- https://www.investopedia.com/articles/trading/08/turtle-trading.asp