Prise en main : le Reinforcement Learning chez Aequam Capital

Prérequis : Le Reinforcement Learning : une introduction théorique et pratique

Quoi de mieux pour comprendre une technologie que de l'utiliser ? Ce document détaille un cas d'usage de reinforcement learning que fut l'application de cette technologie à l'allocation de portefeuille chez Aequam Capital. La question est assez simple : comment paramétrer une allocation dynamique d'actions au sein d'un portefeuille pour en maximiser le rendement (corrigé de la volatilité et de la perte maximale) ?

Pourquoi le Reinforcement Learning ?

Alors que de nombreux articles prétendent battre systématiquement les marchés financiers à l'aide d'algorithmes prédictifs, on observe que la réplication des méthodes « magiques » proposées ressemble bien souvent à du surapprentissage, à du hasard ou à un choix judicieux de paramètres qui permettent un « backtrading » efficace. Souvent, ces articles emploient des méthodes d'apprentissage supervisé et cherchent à prédire le rendement de tel ou tel actif (de manière binaire : positif ou négatif ; ou de manière continue : le rendement exact). La stratégie alors employée est simple : prédire le rendement et se positionner « en avance » du marché. Ces stratégies négligent des dynamiques temporelles (coût de transaction, lissage du portefeuille) et ne sont propices qu'au day-trading (prise opportuniste de profit, trading du bruit, …). En ce qui concerne la gestion de portefeuille orientée moyen-long terme, la dynamique temporelle est clef : l'allocation de portefeuille change au cours du temps mais d'autres critères entrent en jeu, comme la stabilité du portefeuille, l'évitement de grandes pertes, la faible volatilité des rendements, bref la capacité du portefeuille à correspondre à un mandat précis vendu à des investisseurs. En d'autres termes, l'objectif est essentiellement différent et inclut une dynamique temporelle : c'est ici que le reinforcement learning prend son sens en tant qu'il optimise une trajectoire là où l'apprentissage supervisé prédit ponctuellement des labels.

Données du problème

Pour pouvoir poser le problème et l'adapter à la structure du reinforcement learning, il faut au préalable clarifier les données du problème :

- **Univers d'investissement** : 6 facteurs de risque (paniers d'actions présentant les mêmes caractéristiques) académiques : Value, Profitability, Low Vol, Quality, Carry, Momentum ;
- **Mandat de gestion** : performance défensive, alignée avec l'indice de référence (Eurostoxx 50) quand ce dernier est haussier, protecteur du capital quand ce dernier est baissier ;
- **Cible de turnover** : changement d'allocation à fréquence faible (entre 15 et 30 jours – entre 3 et 6 semaines) pour baisser les coûts de transaction et assurer une cohérence avec le modèle de gestion actuel (et vendu) ;
- **Coût de transaction** : hypothèse réaliste de 10bps/transaction (0.1%) ;

- **Référence à battre** : un portefeuille équipondéré des facteurs de risque remplit déjà le mandat de gestion. On cherche donc à mettre en place une stratégie qui soit encore meilleure qu'un portefeuille statique équipondéré.

La question est la suivante : un agent (trader) peut-il, en observant des **données qui contiennent supposément un signal** (environnement), choisir une allocation de portefeuille parmi ces six facteurs (action), afin de maximiser le rendement de long-terme du portefeuille sous des contraintes de volatilité et de perte maximale (récompense) ?

A ce stade, il faut dire un mot des données à notre disposition pour « nourrir » les algorithmes et leur permettre d'apprendre la trajectoire de portefeuille à suivre.

Données disponibles

La question des données est cruciale dans tout problème d'apprentissage statistique. Dans un problème de finance de marché, il y a rarement un problème de pénurie de données ou de données manquantes, ou erronées, ou corrompues. Au contraire, les fournisseurs de données financières comme Bloomberg ou Reuters ont une quantité phénoménale de données à proposer et la problématique est plutôt celle de la sélection. L'interconnexion est une caractéristique forte des marchés financiers, donc le nombre de variables ayant une corrélation avec la variable cible (pouvant expliquer le rendement, donc ayant un pouvoir prédictif) est immense et il est évidemment impossible de toutes les prendre en compte. Nous avons donc été contraints de faire des choix. Voici les données sélectionnées *a priori* comme utilisables par l'algorithme :

- **Les prix des facteurs** : les prix ayant une dynamique propre, il est évident - et démontré - que les observer est utile pour faire un choix d'allocation gagnant ;
- **Des indicateurs macroéconomiques** : indicateurs d'états économiques, de tendances à venir, de changements etc, ils offrent des regards exogènes sur les marchés financiers. Parmi eux :
  1. Des données de **volatilité** qui peuvent indiquer un environnement stressé ;
  2. Des données de **volumes** échangés qui renseignent sur les comportements des agents évoluant sur les marchés ;
  3. Des données de **surprises économiques** qui informent sur les chocs informationnels et leurs impacts sur les prix de marché.

Se pose bien sûr la question de la transformation des données. A ce sujet, il a fallu jongler entre la théorie statistique, les contraintes d'implémentation pratiques et le caractère non-aléatoire, autocorrélé des séries temporelles à disposition.

Prenons un exemple simple : les prix de marché, c'est de notoriété publique, sont des séries hautement non stationnaires. Elles font montre de persistance, d'autocorrélation, d'intermittence, etc ; dans le même temps, les transformer en séries stationnaires a-t-il du sens ? Conserver le bruit dans ces séries conduit à l'écueil suivant : si le modèle assimile ce bruit comme de la stationnarité, alors les actions prescrites par le modèle ont tendance à être également bruitées et donc l'allocation qui s'ensuit va être chaotique, donc non implémentable (les coûts de transaction cumulés devenant vite prohibitifs) et surtout non vendable (comment expliquer que l'on change l'allocation du tout au tout d'un jour sur l'autre ?).

Des transformations non stationnaires peuvent donc être envisagées pour assurer une continuité temporelle des décisions prises par l'algorithme et même si un point d'honneur a été mis à considérer des données débruitées et stationnarisées, des choix ont dû être faits pour tempérer ce besoin.

En outre, on observe que ce n'est pas parce qu'une transformation ne fait pas de sens mathématiquement qu'elle n'apporte pas d'information. Si de nombreux traders suivent et utilisent tous les jours dans leurs stratégies une transformation, alors cette transformation a un impact sur les prix et donc l'utiliser apporte de l'information et un pouvoir prédictif sur les rendements. Cette remarque conduit à calculer des écarts-types sur des prix (non stationnaires) plutôt que sur des rendements (potentiellement plus stationnaires) par exemple.

Voici une liste non exhaustive des transformations de données qui ont été utilisées lors de ce projet. Bien souvent, ces transformations ont été empruntées au domaine de l'analyse technique :

- **MACD** (Moving Average Convergence Difference) : différence entre deux moyennes mobiles exponentielles sur un prix (une de courte période, une de longue période). Cet indicateur est très utilisé par les analystes techniques et le signe de cette différence apporte une information sur le prix (un changement de signe indique un retournement de tendance) ;
- **Stochastique** : indicateur de positionnement de prix sur la période récente. On fixe un paramètre de période, n, on calcule le prix maximum sur la période, H_n, et le prix minimum sur la période B_n et le stochastique se calcule selon la formule *Stochastique = 100* x $\frac{\mathrm{Prix}-\mathrm{B_n}}{\mathrm{H_n}-\mathrm{B_n}}$
  Lorsque le stochastique est proche de 100, le prix est proche de son maximum ; inversement, lorsque le stochastique est proche de 0, le prix est proche de son minimum sur la période ;
- **RSI** (Relative Strength Index) : indicateur comportemental qui indique la tendance récente et son potentiel excès. A nouveau, on fixe un paramètre de période n. Sur cette période, on calcule une moyenne mobile exponentielle des hausses H_n et une moyenne mobile exponentielle des baisses, mise en valeur absolue, B_n. Le RSI se calcule comme suit : $\mathrm{RSI} = 100 \times \frac{\mathrm{H_n}}{\mathrm{H_n}+\mathrm{B_n}}$. Lorsque le RSI est proche de 100, le titre est très régulièrement en hausse, potentiellement suracheté ; lorsque le RSI est proche de 0 en revanche, le titre est très régulièrement en baisse, potentiellement survendu.

Sans aller plus dans le détail, on voit que chacune de ces transformations contient des paramètres (taille de la fenêtre, durée des moyennes mobiles, ...), on voit également le nombre de données disponibles, et donc on imagine la quantité possible de jeux de données à considérer. Il a donc fallu faire des choix, dès avant le modèle, dans la sélection des données et de leurs transformations.

Une manière de réduire le bruit et de réduire le nombre de variables à considérer a été l'**Analyse en Composantes Principales** (ACP).

L'ACP est une méthode de réduction de dimension : elle vise à réduire l'information donnée par un large nuage de points en grande dimension en trouvant un équilibre entre un faible nombre de dimensions et une forte variance expliquée. L'ACP transforme des variables liées - dans notre cas, les prix d'un facteur sont corrélés avec les prix d'autres facteurs, avec des indicateurs macroéconomiques, ... - en des variables décorrélées appelées composantes principales, ou facteurs. Utilisée dans une optique descriptive, l'ACP permet d'isoler des sources de variances indépendantes pour expliquer, au sens mathématique de la variance, mais également au sens littéral de la compréhension, des mouvements.

L'ACP diagonalise la matrice de variance-covariance d'un nuage de points donné et trouve un vecteur tel que la variance de la projection du nuage sur ce vecteur soit maximisée : ce vecteur est la première composante principale (CP1). Puis, il trouve, parmi les vecteurs orthogonaux à la CP1 (selon le produit scalaire canonique) un vecteur tel que la variance de la projection du nuage sur l'espace vectoriel

engendré par ces deux premiers vecteurs soit maximisée : c'est la CP2. Et ainsi de suite. Le gain en variance expliquée décroît, par construction, à chaque nouvelle CP et de nombreux critères existent pour savoir combien de CP retenir (moins de CP que de variables évidemment, sinon la méthode a peu de sens). C'est une méthode très utilisée en finance, notamment parce qu'elle conserve relativement le pouvoir descriptif tout en réduisant drastiquement le nombre de variables.

Nous avons donc appliqué l'ACP sur 15 variables (les 6 séries de prix stationnarisées - en rendement - et 9 indicateurs macroéconomiques cités ci-dessus également stationnarisés) standardisées (pour éviter un effet-taille caractéristique de l'ACP). L'analyse de la variance expliquée par les composantes principales conduit à ne garder que 4 composantes principales. En outre, ces 4 composantes principales suffisent à expliquer près de 75% de la variance du nuage de points, ce qui est amplement suffisant pour notre problématique.

Mise en place du projet

Une fois le problème posé, les hypothèses clarifiées, les données explicatives proposées, on peut mettre en place l'infrastructure de reinforcement learning. **Il faut encore dire un mot de l'espace des actions choisi pour l'agent de trading**. Dans le cas général, on veut que l'algorithme choisisse, à toute date, 7 poids de facteurs entre 0 et 1 dont la somme vaut 1 (6 facteurs et le reste en cash).

On cherche donc à décider à toute date du couple (p_1, p_2, p_3, p_4, p_5, p_6, p_7) tel que $\forall \boldsymbol{i} \in \mathbf{I} = \{\mathbf{1}, \dots, \mathbf{7}\}, \mathbf{0} \leq \boldsymbol{p_i} \leq \mathbf{1}, \boldsymbol{avec} \sum_{i \in I} \boldsymbol{p_i} = \mathbf{1}$. Cette distribution est non triviale et n'existe pas dans les librairies standards. On a donc décidé**, au départ, d'avoir une action binaire** : choisir d'être investi de manière équipondérée ou d'être en cash. En d'autres termes, on a restreint l'espace des actions aux couples de poids (0, 0, 0, 0, 0, 0, 1) (cash) et $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, 0)$ (équipondéré). **Dans un second temps, on a quelque peu élargi l'espace des actions. Nous avons défini des nouveaux portefeuilles hybrides (défensifs, offensifs, etc) et l'algorithme pouvait choisir à chaque date non plus un facteur, mais un portefeuille avec des poids prédéfinis de facteurs**. Cela inclut l'exemple précédent avec (1, 0) et (0, 1) (portefeuille cash ou équipondéré), mais d'autres exemples plus fins également.

A ce stade, le lecteur peut arrêter et se plonger dans la partie pratique du projet. Les éléments de ce dernier sont décrits dans le fichier **README.md** et, surtout, les fichiers **aequam_env.py** et **Aequam_env.ipynb** sont abondamment commentés. Un certain temps peut être ici consacré à la découverte du projet et au tâtonnement avec ces deux fichiers.

Deux remarques toutefois : dans le code **aequam_env.py**, on a **fixé l'espace d'actions à un espace discret** (à chaque fois, l'agent de trading choisit un seul actif/portefeuille dans lequel investir). Il est possible de changer ce paramètre pour un espace continu (selon la nouvelle stratégie que l'on veut mettre en place). Il faut simplement garder à l'esprit que chaque algorithme (DQN, A2C, TRPO, …) a des exigences en termes de types d'espaces et d'observations. A titre d'exemple, à cette adresse on voit les exigences de l'algorithme DQN. Il en va de même pour les observations (même s'il est assez standard d'avoir des observations continues). En outre, toujours dans ce fichier, on **peut modifier la fonction de récompense**. Dans la méthode « step », on a plusieurs récompenses définies selon un mot-clef passé en paramètre (*'reward_type'* dans le notebook). Il y en a trois, la plus utilisée étant 'vol' : dans cette fonction de récompense, on définit

$$reward_t = return_{t+T} - \alpha \cdot semivariance_{t+T} - \beta \cdot transactioncosts_{t+T}$$

Dans cette fonction, T est le paramètre *'reward_window'* qui donne à quel horizon regarder la récompense ; α est le paramètre '*risk_aversion'* qui pénalise une variance des rendements négatifs et β est le paramètre *'transaction_smoothing'* qui pénalise les coûts de transaction élevés (donc les transactions trop fréquentes). Il est tout à fait possible de compléter ce fichier en ajoutant d'autres options de récompenses qu'on définira comme on voudra et dont on notera la clef/le paramètre dans le notebook.

Développement

Le projet a connu de nombreux tâtonnements du fait de sa nouveauté pour toute l'équipe.
Aujourd'hui il est relativement robuste quant à ce qui a été construit, et assez flexible en termes de stratégie à mettre en place. Cela est dû notamment à Open AI et à sa librairie Gym « plug and play » très pratique à utiliser. **Mais il est loin d'être abouti** et voici la liste, par **ordre croissant de difficulté**, de ce qui peut être abordé sur le **chemin de la mise en production et de l'utilisation effective du Reinforcement Learning dans le processus d'allocation d'actifs d'Aequam Capital**.

- **Ajouter/modifier/supprimer des variables explicatives** : dans le notebook Aequam_env.ipynb, on peut par un simple changement de la variable df_obs changer l'ensemble des variables explicatives. Ces dernières sont donc un paramètre très simplement modifiable. Il faut simplement être attentif à garder la structure donnée dans les exemples ;
- **Ajouter/modifier/supprimer un portefeuille ou un actif à échanger** : exactement la même remarque, mais avec la variable 'df_prices' ;
- **Jouer avec les paramètres de l'environnement** (coût de transaction, fenêtre de regard de l'agent dans le passé, horizon de récompense de l'agent, durée de l'entraînement, …) : c'est tout le but du code flexible qui a été écrit. Tout ce qui n'est pas évident est un paramètre optimisable, et un bloc de code y est dédié dans le notebook ;
- **Optimiser avec différentes logiques et optimiseurs** : un exemple a été pris, celui d'utiliser skopt.forest_minimize. D'autre minimiseurs existent, et comportent des paramètres de minimisation qu'on peut modifier très facilement. Il faut garder à l'esprit que c'est un exemple qui a été fait et que ce dernier peut être modifié ;
- **Valider un modèle donné** : le risque du surapprentissage est fort tant la logique du reinforcement learning est basée sur la répétition d'épisodes. Mais les backtests ne font jamais apparaître un surapprentissage flagrant semblable à du backtrading. En tout cas, on peut écourter, via df_obs et df_prices, l'historique d'apprentissage et utiliser la fonction play_last_episode sur la suite de l'historique. **La validation croisée est exclue** du fait du caractère temporel du problème : une trajectoire ne saurait être optimisée avec un trou ;
- **Pour une mise en production, il faut un modèle validé** : étant donné le caractère encore instable et non discriminant de l'optimisation, il est difficile de choisir un point (un jeu de paramètres) optimal et de le mettre en production (même si cette partie est très simple en pratique). Il faut une forme de validation. En l'absence de critère évident, **on peut utiliser une forme d'agrégation, donc de vote**. On peut imaginer prendre 10, 20, 30 modèles qui fonctionnent bien et ont des paramètres bien distincts, puis les faire voter. A toute date, chacun va choisir un actif (ou un portefeuille) parmi tous et ainsi **on n'aura plus un choix binaire… mais une allocation**. Cela permettrait également **de réduire les coûts de transaction** (par des votes se compensant d'un jour sur l'autre). Une fois un modèle enregistré via pickle (module de sauvegarde Python, voir le notebook), on peut l'utiliser très simplement (aussi simplement qu'en écrivant model.predict()) ;

- **Feature engineering** : on a choisi des variables et des transformations qui paraissaient logiques. Mais le biais humain reste central et on peut essayer de le réduire en créant des variables de manière automatique. Cela s'appelle du feature engineering et se fait facilement en Python grâce à la librairie **Featuretools** ;
- **Travailler la fonction de récompense : l'Inverse Reinforcement Learning (IRL)**. On l'a vu, la question de la fonction de récompense est cruciale car elle dicte l'amélioration séquentielle de l'algorithme. Parfois, on récompense un comportement et on génère des effets pervers, indésirables ; on modifie donc la fonction de récompense de manière incrémentale. En ce qui nous concerne, nous avons opté pour une fonction paramétrique dont les paramètres sont optimisés. Un champ de recherche renverse le problème et se demande : étant donné une trajectoire optimale (dite « experte »), est-il possible de trouver la fonction de récompense qui, si elle avait été utilisée pour l'optimisation, aurait conduit à cette trajectoire ? La librairie Gym propose un algorithme de ce type, appelé GAIL (Generative Adversarial Imitation Learning), pour, à partir de trajectoires expertes, trouver la fonction de récompense adaptée et réentraîner un algorithme avec cette fonction de récompense. **A ce stade, cette librairie présente quelques défauts (bugs de format), mais il est à n'en pas douter que ces bugs seront vite corrigés. En tout cas, on peut utiliser des trajectoires expertes faites manuellement mais le plus simple est d'en générer, grâce à des modèles déjà bien entraînés (experts)**.

Le champ des possibles est donc ouvert et de nombreuses pistes sont offertes à qui veut reprendre ce chantier.