Rapport de projet

Data Challenge ENS
Qube Research & Technology
Reconstruction of Liquid Asset Performance

Elyas Benyamina
Alexis Ibrahim
Antoine Mathis

Professeur : Bertrand Michel
Cours : Apprentissage statistique

2021-2022

Table des matières

1 Introduction

1.1 Présentation du problème

On s'intéresse ici au data challenge 2021 disponible sur la plateforme de data challenge de l'ENS proposé par l'entreprise QRT (Qube Research & Technologies).
Il s'agit d'un challenge de finance quantitative et plus particulierement de la prévision de la valeur d'actifs liquides.
En finance, les actifs peuvent être séparés en deux catégories : d'une part les actifs **illiquides** et d'autre part les actifs **liquides**.

1. **Actif liquide** *(définition de l'Autorité des Marchés Financiers)* : Un actif financier (actions, obligations, etc.) est dit liquide lorsqu'il peut être acheté ou vendu rapidement sans que cela n'ait d'impacts majeurs sur son prix. La liquidité reflète donc la facilité avec laquelle cet actif peut être échangé. Ainsi, plus un marché est liquide, plus il est aisé, rapide et peu coûteux d'y réaliser des transactions.
2. **Actif illiquide** : A l'inverse, un actif illiquide est un actif difficilement échangeable en grande quantité, et où une transaction peut faire varier fortement la valeur de l'actif. Les ordres concernant ces actifs peuvent mettre du temps à s'éxecuter ou ne pas s'executer du tout.

L'objectif est le suivant : à partir de données portant sur des actifs illiquides, fournir des prédictions sur l'évolution et la valeur de différents actifs liquides.

On dispose en effet d'un dataset contenant des données portant sur des actifs illiquides : il s'agit de fournir des prédictions concernant des actifs à partir de données colléctées sur d'autres actifs.

L'objectif de ce data challenge, est de fournir un ensemble de fonctions de mapping permettant d'établir la relation entre **100 actifs illiquides** et **100 actifs liquides** d'autre part.

1.2 Présentation des données

On dispose en effet d'un dataset X_{train} contenant les données qui permettront d'entrainer notre modèle, d'un dataset Y_{train} qui regroupe les performances des 100 actifs liquides associés. Enfin, le dataset X_{test} est celui sur lequel nous allons faire nos prédictions. A l'issu de ce processus, un fichier csv sera produit, contenant les prédictions pour chaque actifs liquides de X_{test}

Dans le dataset X_{train}, chaque colonne représente un actif illiquide noté **'RET_' + id_de_l'actif**. Chaque ligne correspond à un jour. Par ailleurs, on retrouve une colonne **ID_TARGET** qui désigne l'actif liquide à prédire avec les données présentes sur la ligne. Les données à notre disposition sont donc les suivantes : on dispose jour après jour de la performance des actifs illiquides d'une part, et d'autre part la performance d'un actif liquide au même moment. Il s'agit donc d'effectuer un "mapping" des données issues d'actifs illiquides pour en extraire de l'information portant sur les actifs liquides.

Afin de manipuler les données plus facilement, et de pouvoir travailler dessus de manière plus efficace (praticité et gains de mémoire car les données étaient répétées dans le dataset initial), il est préférable de créer un data frame contenant jours après jours les performances de nos 200 actifs que ceux-ci soient liquides ou illiquides.

Dans nos données supplémentaires, on retrouve aussi les classes de nos actifs, autrement dit la spécificité de chacun de nos actifs dans un domaine de l'industrie particulier. Ces données pourront nous servir au moment de réaliser une sélection des features les plus adaptés pour effectuer nos prédictions.

Nous avons considéreé que l'on travaillait dans un espace où les entrées et les sorties se trouvaient dans R^{100}. À partir des données dont nous disposions, nous avons entrainé plusieurs modèles sur les données d'apprentissage (contenu de X_{train} et Y_{train}) afin d'effectuer des prédictions sur les données présentes dans X_{test}
Afin de pouvoir quantifier la performance de nos prédictions, la métrique proposée dans l'énoncé est la suivante :

$$f(y, y') = \frac{1}{\|y\|_1} \sum_{i=1}^{n} |y| * 1_{y_i = sign(y_i)}$$

Il s'agit au final d'un problème de classification : ce qui nous intéresse avant tout, c'est de fournir le bon signe pour chacune des valeurs que nous devons prédire, ou autrement dit, le bon signe pour chacune de nos prédictions sur nos actifs liquides.

On note par ailleurs que cette métrique donne plus d'importance aux bonnes classifications avec des hautes valeurs du "return", ou autrement dit pour des actifs dont la performance est élévée. Il est ainsi plus important (vis à vis de la métrique) d'avoir une bonne prédiction pour un actif représentant 7 % de notre mise de départ plutôt que 0.5 %.

Nous avons donc catégorisé nos données en deux classes représentées respectivement par les nombres 1 et -1. 1 désignera une valeur prédite positive tandis que -1 désignera une valeur négative.

1.3 Espace de travail

Afin de simplifier la mise en commun du code source, nous avons utilisé les plateformes **Jupyterlab** ainsi que **Google Colab** pour permettre un suivi centralisé de l'avancée du projet.
Par manque de puissance (temps d'exécution très longs pour certaines opérations, même après réduction de la taille du dataset), nous avons décidé de monter un serveur sur la plateforme **Google Cloud Project**. Nous avons créé un serveur web et un environnement de travail Python 3 (installation d'Anaconda et des librairies dont nous avions besoin) auquel nous pouvions accéder à distance via un navigateur web à travers l'IDE Jupyter Lab.
Nous avons connecté cette plateforme à un **GitLab** installé sur le serveur afin de pouvoir suivre nos modifications.

2 Pré-traitement des données

Avant de pouvoir passer à la phase d'apprentissage, il est nécéssaire de pré-traiter nos données. En effet, certaines valeurs des actifs sont manquantes (NaN). Il faut donc trouver des solutions afin d'obtenir un jeu de données complet que l'on peut exploiter.
Il est également intéressant de normaliser les données afin d'obtenir des performances potentiellement meilleures.

2.1 Données manquantes

Nos données présentent de base un certain nombre de "Nan" : "Not a Number". Il s'agit de cellules dans lesquelles les valeurs ne sont pas renseignées. Pour un actif donné, on peut avoir jusqu'à 13% de valeurs manquantes.
Il s'agit donc de trouver une méthode afin de remplacer ces valeurs manquantes.

Approche 1 : Supprimer les lignes où l'on a des données manquantes

Le problème de cette solution est le suivant : la suppression des lignes entraine une diminution du nombre de données à tester assez importante et les performances des algorithmes sont assez faibles. Une alternative est de prédire la valeur de chaque actif liquide séparément et de supprimer les lignes uniquement pour un actif donné mais le temps d'exécution est extrêmement lent pour des scores qui restent encore faibles.

Approche 2 : Interpolation

Une autre possibilité consiste à utiliser la fonction interpolate() de pandas. Là encore, on se retrouve limité étant donné que interpolate consiste à effectuer une régression linéaire sur l'index, ou une colonne définit arbitraiement. Ainsi les Nan sont remplacés par l'interpolation linéaire éffectués à partir du contenu des autres lignes pour le même feature.
Etant donné que les jours ont été "anonymisés", cette approche est limitée dans la mesure où l'on utilise des valeurs qui n'ont potentiellement aucun lien avec le feature que l'on souhaite évalué.

Approche 3 : Remplacer par colonne (moyennage par actif)

Une autre solution consiste à remplacer une valeurs manquante d'un actif donné par la valeur moyenne que prend l'actif.
Cette solution donne d'assez bons résultats.

Approche 4 : Remplacement par ligne (utilisation des classes d'actifs)

On utilise le jeu de données supplémentaires fournis pour pouvoir dégager une tendance globale de l'évolution d'une catégorie d'actif (il en existe quatre).
On dispose d'un tableau que l'on a créé nommé *assets_coefs* qui pour chaque actif donne son "ratio d'appartenance" à chaque classe (la somme des ratio d'appartenance fait 1). Ensuite, pour un jour donné, on parcourt les actifs pour lesquels la valeur est présente et l'on obtient une moyenne pour chaque catégorie d'actif pondérée selon la valeur des actifs et leur pourcentage d'appartenance à la classe.
On obtient donc une "valeur" pour chacune des quatre catégories d'actifs et on remplace une valeur manquante en effectuant une moyenne de ces quatre valeurs pondérée par les "ratios d'appartenances" de l'actif. Cette méthode nous a semblé pertinente car elle essaie de remplacer les valeurs vides en se servant de la tendance générale en bourse d'un secteur industriel. Cette méthode donne également d'assez bons résultats.

Nous avons retenu les deux dernières approches qui donnaient des meilleurs résultats lors de nos tests.

2.2 Normalisation et Standardisation

Nos datasets présentent initialement des ordres de grandeurs différents. Cette différence d'échelle peut conduire à des performances moindres. Pour palier à cela, des traitements préparatoires sur les données existent. No-

tamment le Feature Scaling qui comprend la Standardisation et la Normalisation.

La normalisation consiste à redimensionner les variables numériques pour qu'elles soient comparables sur une échelle commune. Dans la plupart des cas nous avons utilisé Standard Scaler.

L'idée derrière StandardScaler est qu'il transforme les données de telle sorte que sa distribution aura une valeur moyenne de 0 et un écart type de 1.
En cas de données multivariées, cela se fait par features (en d'autres termes indépendamment pour chaque colonne des données).

3 Feature Engineering & Traitement par lots

3.1 Selection des Features

Nos données contiennent un nombre important de features. Aussi, il nous a semblé pertinent de nous poser la question de la pertinence et de l'utilité de chacun des features pour nos prévisions. En effet,nos actifs peuvent représenter différents types de sociétes. Par exemple, des actifs répresentant des "grosses sociétés", ayant des secteurs d'activité divers pourraient être plus à même de refléter des tendances plus globales.

Il semble donc intéressant d'effectuer une séléction de nos features afin de ne garder que les plus pertinents. Pour ce faire, nous pouvons utiliser par exemple la fonction SelectKBest de scikit-learn. Lors de nos tests, la sélection des features donne des résultats inférieurs à ceux obtenus sans sélection.

3.2 Batch Processing des données

Nous nous sommes ensuite intéressés à d'autres approches pour essayer d'augmenter les performances obtenues.
Au départ, nous utilisions un unique modèle pour prédire l'ensemble des valeurs des actifs liquides.
Nous avons pensé à utiliser différents modèles et à traiter les actifs séparément. Plusieurs approches étaient possibles : En effet plusieurs choix là aussi sont possibles : procéder à un traitement des données en un bloc. Autrement dit, on considère que le lien entre nos 200 actifs illiquides et liquides est le même quelque soit l'actif liquide considéré. Les actifs ne sont pas alors séparés ce qui peut entrainer que la performance d'un actif liquide puisse influer sur celle d'un autre actif liquide.

Procéder de la sorte revient à sous-évaluer l'independance des actifs liquides entre eux. Aussi nous avons essayer d'utiliser nos données en regroupant de différentes manières :

1. Pas de regroupement (traitement de chaque actif liquide spérament, 100 modèles utilisés)
2. Regroupement total (traitement de tous les actifs liquides en même temps, 1 seul modèle utilisé)
3. Regroupement des actifs par des méthodes de clustering, par exemple plus proches voisins (traitement de groupes d'actifs, n modèles utilisés où n est le nombre de groupes d'actifs obtenu après clustering)

Un manque de puissance nous a freinés dans la première approche. Nous ne pouvions obtenir des résultats, l'entrainement et l'utilisation de 100 modèles étant trop lourd pour la machine. La dernière approche, bien

qu'assez lente (nous avons essayé de former entre 3 et 10 groupes des groupes par la méthodes des plus proches voisins) pouvait s'exécuter sur notre environnement.

Ces façons de procéder bien qu'elles puissent s'avérer intéréssantes ne nous ont pas permis d'obtenir des résultats significativement meilleurs.
Le clustering permet dans certain cas de grapiller quelques centièmes. Cela permet également une approche hybride entre la création et l'entraînement d'un unique modèle pour 100 actifs et de 100 modèles différents pour 100 actifs (il nous était parfois impossible de travailler avec 100 modèles étant donné les ressources matérielles que nous avions à notre disposition qui ne supportaient pas une telle charge).
Néanmoins,il est important de noter qu'au moment de la mise en oeuvre de la méthode de regression partielle, la combinaison d'un traitement par lot et d'une selection des features les plus adaptés se sont avérés plutôt concluants.

4 Choix des modèles et résultats

4.1 Premiers modèles

Dans un premier temps nous nous sommes intéréssé aux modèles vus en cours parmi lesquel Naive Bayes, K-Neighbors, Random Forest. Nous présentons ici les résultats que nous avons obtenus. À titre de comparaison, nous rappelons que le modèle proposé dans le benchmark permettait d'atteindre un score de 69% au moment de la soumission.

4.2 Naive-Bayes

Pour rappel la méthode Naive Bayes consiste à calculer la probabilité d'appartenance à la classe 1 ou 2, à partir des probabilités conditionnelles portant sur les features.
La formule de Bayes nous donne :

$$p(C|F_1, ..., F_n) = \frac{p(C)*P(F_1,...,F_n|C)}{P(F_1,...,F_n)}$$

où C désigne la variable de classe et $F_1, .., F_n$ les variables explicatives (nos features). L'utilisation de cette méthode implique de calculer le maximum de vraisemblance pour nos classes. Par ailleurs, ce modèle présuppose que nos features sont indépendants ce qui est une hypothèse forte étant donnés que nos actifs illiquides peuvent être fortement corrélés les uns aux autres.

Avec cette méthode, en appliquant les transformations mentionnées précédemment (pre-processing, normalisation etc..) nous n'avons obtenu qu'un score de 68% soit en deça de celui du BenchMark.

4.3 K-Neighbors Classifier

La méthode des K-Neighbors consiste à réaliser une prédiction à partir des classes obtenues pour des échantillons proches de celui à prédire (on utilise ici la norme euclidienne). Afin de mieux estimer la valeurs des hyper paramètres nous avons associé cetté approche avec l'utilisation d'une GridSearch permettant d'estimer au

mieux la valeur de nos hyperparamètres en particulier celui du nombre de voisins optimal et de la norme à utiliser.
On joint en annexe les scores obtenus en testant pour différentes valeurs de k, le nombre de plus proches voisins. Comme on peut le voir cette approche côuteuse en terme de compléxité, ne permet pas de dépasser les 65% de bonnes prédictions ce qui reste en deça des performances du Benchmark.

4.4 Random Forests

Le random forest est composé de plusieurs arbres de décision, travaillant de manière indépendante sur une vision d'un problème. Chacun produit une estimation, et c'est l'assemblage des arbres de décision et de leurs analyses, qui va donner une estimation globale. Nous avons utilisé du tuning en utilisant Randomized Grid Search afin de nous guider sur le choix de paramètres offrant de bonnes performances. Cette méthode nous a permis d'obtenir des résultats corrects mais nous avons été freinés (notamment dans l'utilisation de Grid Search) par un manque de puissance rendant les temps de calculs extrêmement longs ou bloquant la machine.

4.5 Ada Boost Classifier

Nous avons aussi testé l'algorithem Ada Boost disponible dans Scikit learn.
Le principe de base consiste à entrainer des modèle relativement simplistes, légèrement plus performant que des prédictions aux hasard (comme des petits arbres de décisions) de manière répétée sur des versions modifiées des données de bases. On combine ensuite les predictions en utilisant le principe de majorité des votes afin de fournir la prédiction finale.
Les modifications successives consiste à appliquer des poids à nos échantillons pour séléctionner les plus "probants". On réentraine notre modèle ainsi en changeant successivement la valeur des poids.

Cette approche bien qu'elle ait pu nous sembler intéréssant s'est avérée limité et ne nous a pas permis de dépasser un score de 70%.

5 Autres approches

En complément des méthodes mentionnées ci-dessus, nous avons aussi mis en place un certains nombre de modèles qui nous ont permis d'obtenir des résultats significativement meilleurs.

5.1 Méthode LASSO

Nous avons mis en oeuvre la méthode LASSO (Least Absolute Shrinkage and Selection Operator). Cette méthode permet de selectionner les meilleures variables explicatives, LASSO s'est avérée être relativement efficace c'est pourquoi nous en présentons ici le fonctionnement.

LASSO est une méthode liée à la régression. LASSO vise à estimer les coefficients $\beta_1, .., \beta_100$ permettant de pondérer nos variables explicatives.

On cherche $\min_{\beta_1,\dots\beta_n} \frac{1}{2}(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_i x_{ij}))^2$
avec comme contrainte

$$\sum_{j=1}^{n} |\beta_i| < t$$

5.2 Régression de processus gaussien

En observant la distribution d'un actif quelconque, on remarque une répartition de ses valeurs qui semble se rapprocher d'une distribution gaussienne. Nous avons donc voulu essayer une régression de processus gaussien. Un processus gaussien est un ensemble de variables aléatoires telle que la distribution conjointe de chaque sous-ensemble fini de variables aléatoires est gaussienne.

On suppose que la fonction f suit un processus gaussien : $f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k\left(\mathbf{x}, \mathbf{x}'\right)\right)$ On choisit une fonction moyenne et un noyau qui vont définir les types de fonctions potentielles. Le noyau est semi défini positif, et nous avons utilisé le noyau RBF (Radial Basis Function), où l et α sont des paramètres qui permettent de jouer sur la corrélation entre les différents points.

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \operatorname{Cov}\left(f(\mathbf{x}), f\left(\mathbf{x}'\right)\right) = \alpha \exp\left(-\frac{1}{2\ell^2}\left\|\mathbf{x} - \mathbf{x}'\right\|^2\right)$$

Ce noyau fonctionne en effet bien avec des distributions se rapprochant de distributions gaussiennes, et cela se vérifie avec les résultats obtenus en comparaison à l'utilisation d'autres noyaux. Nous pouvons ensuite, après avoir défini le noyau, utiliser les processus gaussiens pour estimer des valeurs de fonction à priori, ainsi que des valeurs de fonctions à posteriori conditionnelles aux observations précédentes (on se restreint aux fonctions à priori qui concordent avec les données d'apprentissage).
Les résultats obtenus sont assez faibles.

5.3 Réseaux de neurones

Nous avons aussi éssayé de mettre en oeuvre un modèle de réseaux de neurones afin de maximiser nos performances. Néanmoins, la question du choix de nombre de couches et de leurs paramètrages s'est avérée crucial. Aussi, la rédéfinition de la métrique a pu pu poser un certain nombre de problème. Bien que les réseaux furent une piste intéréssante, la non convergence des résultats malgré un nombre important "d'épochs" et l'utilisation de la métrique "binary cross-entropy" ne nous a pas permis d'obtenir des résultats concluants.

6 Synthèse des résultats

Le tableau suivant permet de récapituler les résultats obtenus pour nos différents modèles/ approches.
Nous avons donc entrainé notre modèle en utilisant les méthodes suivantes :

- Naive Bayes
- LASSO
- Random Forest
- Algorithme Ada Boost

- Régression de processus Gaussiens

La plupart de ces méthodes ont fourni des résultats significativement moins bon que ceux que nous avons obtenu par la suite.

Par exemple dans le cas de Naive Bayes, résultat de 0.68, ce qui est en dessous du score obtenu avec le benchmark d'origine.

De la même manière Random Forest nous a fourni des résultats nettement moins bons.

Méthodes	**Score**
Naive-Bayes	0.68
LASSO	0.7429
ADA	0.67
Benchmark	0.69
Random Forest	0.72
Régression de processus Gaussiens	0.68

Il est important de noter que les modifications réalisées sur les données avant qu'elles ne soient fournies à notre modèles, ont pu dans certains cas affecter nos résultats.

7 Conclusion

Comme nous l'avons vu, différentes approches sont possibles afin de traiter ce problème. À travers ce data challenge, nous avons pu voir l'importance du pré-traitement des données pour la performance de nos modèles. Une grande partie du temps a été passée sur la mise en forme des données et sur le pré traitement de celles-ci, avant que nous passions au choix et à l'entrainement des modèles.
La méthode LASSO s'est avérée particulièrement efficace, associée à un travail de pré-traitement en amont. Nous obtenons un taux de 74% de bonnes prédictions. A noter les résultats prometteurs obtenus en utilisant Random Forest. En effet, avec une machine plus puissante (et/ou plus de temps), on peut explorer davantage les pistes du tuning et du clustering qui pourraient améliorer les résultats obtenus.
Nous avions d'autres pistes que nous n'avons pas pu explorer. Par exemple, essayer de prédire chaque actif ou des groupes d'actifs avec des modèles (éventuellement différents) l'un après l'autre, de sorte que dès qu'un actif ou groupe d'actif est prédit, il devient une variable pour la prochaine prédiction.
Nous avons également envisagé de plus travailler les liens entre les différents actifs, en essayant de regrouper les actifs liquides et illiqudes, afin de travailler sur des groupes dont les actifs suivent des tendances communes.
Enfin, nous avons essayé d'utiliser des réseaux de neurones, avec des résultats peu concluants. Il peut être intéressant d'essayer de travailler cette approche en essayant des nouveaux types de réseaux de neurones.