

SUJET 14 : Théorie des jeux et Finance

algorithmes de regret logarithmique pour la prédiction d'un portefeuille

PROJET 3 : Machine Learning et Théorie des jeux



LEMAIRE Paulin

TSIKA NGOUARI Kevin

Introduction

Lorsqu'on cherche à valoriser une entreprise et son potentiel, l'étude du portefeuille d'actifs d'une entreprise donne un aperçu de ses perspectives de croissance, de sa dynamique et donc de sa valeur de marché.

L'analyse des actifs réels représente un grand pas en avant par rapport aux mesures d'évaluations statiques tels que les ratios cours/bénéfices et cours/valeur comptable. Depuis lors, il était possible de comparer plusieurs entreprises à partir de ces ratios si elles avaient la même croissance attendue des bénéfices. Or c'est rarement le cas, ce qui donne une dimension réelle à l'analyse des actifs, puisqu'elle se base sur la croissance des bénéfices. On met donc ici en avant l'étude des opportunités de croissance qui s'offrent à l'entreprise sachant si elle peut les exploiter ou non. C'est donc les compétences de la direction qui deviennent intéressantes à étudier.

En théorie des jeux, les décisions prises par les acteurs (entreprises) auront des conséquences sur leur situation (et valeur) et celle des autres participants ainsi que sur le marché lui-même. Il conviendra donc d'établir la meilleure stratégie afin d'optimiser leurs intérêts.

Optimisation convexe pour la prédition d'un portefeuille d'actifs

Dans de nombreuses applications pratiques, l'environnement est si complexe qu'il est impossible d'établir un modèle théorique complet et d'utiliser la théorie algorithmique classique et l'optimisation mathématique. Il est donc nécessaire et bénéfique d'adopter une approche robuste, en appliquant une méthode d'optimisation qui apprend, en tirant des leçons de l'expérience au fur et à mesure que de nouveaux aspects du problème sont observés. Cette vision de l'optimisation en tant que processus s'est imposée dans des domaines variés et a conduit à des succès spectaculaires dans la modélisation, comme dans notre cas qui porte sur l'étude des prédictions des cours dans la valorisation d'actifs d'un portefeuille.

Cette étude se ramène à observer un problème d'optimisation convexe (online convex optimization), où un décideur prend une séquence de décisions, c'est-à-dire qu'il choisit une séquence de points dans l'espace euclidien à partir d'un ensemble faisable fixe. A partir de ces points, on pourra étudier une séquence de fonctions de coût convexes (éventuellement non liées). L'objectif sera d'optimiser ces fonctions de coût grâce à des algorithmes de regret logarithmique basé sur la méthode de Newton. Pour rappel, le regret représente ici la différence entre le bénéfice en utilisant n fois le meilleur choix d'actifs et l'espérance du bénéfice après n essais.

Théorie Universelle du portefeuille

A traver cette théorie, l'idée est de modéliser l'investissement comme un scénario de prise de décision répétée, qui s'intègre bien dans le processus d'optimisation convexe, en mesurant le regret comme une métrique de performance.

Considérons le scénario suivant : à chaque itération $t \in [T]$, le décideur choisit x_t , une distribution de sa richesse sur n actifs, telle que $x_t \in \Delta_n$. Ici $\Delta_n = \{x \in R^{n+}, \sum_i x_i = 1\}$ est le simplex à n dimensions, c'est-à-dire l'ensemble de toutes les distributions sur n éléments. Un adversaire choisit indépendamment les rendements du marché pour les actifs, c'est-à-dire un vecteur $r_t \in R^{n+}$ tel que chaque coordonnée $r_t(i)$ est le rapport de prix pour le i ème actif entre les itérations t et $t + 1$.

Par exemple, si la (i)ième coordonnée est le symbole GOOG de Google négocié sur le NASDAQ, alors

$$r_t(i) = \text{prix de GOOG au temps } (t+1) / \text{prix de GOOG au temps } (t)$$

Soit W_t sa richesse totale à l'itération t .

Puis, en ignorant les coûts de transaction, nous avons

$$W_{t+1} = W_t - r_t^\top x_t$$

Sur T itérations, la richesse totale de l'investisseur est donnée par

$$W_T = W_1 \cdot \prod_{t=1}^T \mathbf{r}_t^\top \mathbf{x}_t$$

L'objectif du décideur, qui est de maximiser le gain de richesse globale W_T / W_0 , gain qui peut être atteint en maximisant le logarithme suivant :

$$\log \frac{W_T}{W_1} = \sum_{t=1}^T \log \mathbf{r}_t^\top \mathbf{x}_t$$

La formulation ci-dessus est déjà très similaire à nos paramètres d'optimisation, bien qu'elle soit formulée comme une maximisation du gain plutôt qu'une minimisation des pertes.

On se ramène donc à l'étude de la fonction :

$$f_t(\mathbf{x}) = \log(\mathbf{r}_t^\top \mathbf{x})$$

L'ensemble convexe est le simplex à n dimensions $K = \Delta^n$, et on définit le regret

$$\text{regret}_T = \max_{\mathbf{x}^* \in \mathcal{K}} \sum_{t=1}^T f_t(\mathbf{x}^*) - \sum_{t=1}^T f_t(\mathbf{x}_t)$$

Les fonctions $f(t)$ sont concaves plutôt que convexes, ce qui est parfaitement correct puisque nous formulons le problème comme une maximisation plutôt qu'une minimisation.

Comme il s'agit d'un problème d'optimisation convexe, on peut utiliser l'algorithme de descente de gradient qui garantie un regret $O(T)$ ou bien la méthode de newton avec un regret $O(\log T)$ pour investir.

Algorithme de Descent Gradient

Algorithm 6 online gradient descent

- 1: Input: convex set \mathcal{K} , T , $\mathbf{x}_1 \in \mathcal{K}$, step sizes $\{\eta_t\}$
- 2: **for** $t = 1$ to T **do**
- 3: Play \mathbf{x}_t and observe cost $f_t(\mathbf{x}_t)$.
- 4: Update and project:

$$\begin{aligned}\mathbf{y}_{t+1} &= \mathbf{x}_t - \eta_t \nabla f_t(\mathbf{x}_t) \\ \mathbf{x}_{t+1} &= \Pi_{\mathcal{K}}(\mathbf{y}_{t+1})\end{aligned}$$

-
- 5: **end for**
-

```

122
123     class oga(algorithm):
124         def __init__(self,
125                      data: str, # data folder
126                      old_dates: str, # old param dates
127                      new_dates: str, # new param dates
128                  ):
129             super().__init__(data, old_dates, new_dates)
130             self.name = "oga"
131             self.params["x"] = ["x"]
132
133     def algorithm(self, X: np.ndarray) -> list:
134         T = X.shape[1] # length
135         d = X.shape[0] # dimension
136         if not self.params: # no previous params
137             self.params["x"] = np.ones([d, 1]) / d # how to invest
138
139         rewards = []
140
141         for t in tqdm(range(1, T), desc=self.name):
142             r_t = X[:, t] / X[:, t - 1]
143             r_t = r_t[:, None]
144
145             multiplier = r_t.T @ self.params["x"]
146             rewards += [np.log(multiplier)[0][0]]
147
148             grad = r_t / multiplier
149             eta = 1 / (d * np.sqrt(t))
150             y = self.params["x"] + eta * grad # + for ascent
151             self.params["x"] = project_simplex(y)
152
153
154         return rewards
155
156

```

À chaque itération, l'algorithme effectue un pas à partir du point précédent dans la direction du gradient du coût précédent. Cette étape peut aboutir à un point situé en dehors de l'ensemble convexe sous-jacent. Dans ce cas, l'algorithme projette le point dans l'ensemble convexe, c'est-à-dire qu'il trouve son point le plus proche dans l'ensemble convexe.

Algorithme de Newton Step

Algorithm 9 online Newton step

- 1: Input: convex set \mathcal{K} , T , $\mathbf{x}_1 \in \mathcal{K} \subseteq \mathbb{R}^n$, parameters $\gamma, \varepsilon > 0$, $A_0 = \varepsilon \mathbf{I}_n$
- 2: **for** $t = 1$ to T **do**
- 3: Play \mathbf{x}_t and observe cost $f_t(\mathbf{x}_t)$.
- 4: Rank-1 update: $A_t = A_{t-1} + \nabla_t \nabla_t^\top$
- 5: Newton step and projection:

$$\mathbf{y}_{t+1} = \mathbf{x}_t - \frac{1}{\gamma} A_t^{-1} \nabla_t$$

$$\mathbf{x}_{t+1} = \underset{\mathcal{K}}{\text{II}}(\mathbf{y}_{t+1})$$

- 6: **end for**
-

```

156
157
158     class ons(algorithm):
159         def __init__(self,
160                      data: str, # data folder
161                      old_dates: str, # old param dates
162                      new_dates: str, # new param dates
163                  ):
164             super().__init__(data, old_dates, new_dates)
165             self.name = "ons"
166             self.params_strs = ["x", "A", "b", "beta"]
167
168         def algorithm(self, X: np.ndarray, beta: float = 2.0) -> list:
169             self
170             T = X.shape[1]
171             d = X.shape[0]
172             if not self.params: # no previous params
173                 self.params["x"] = np.ones([d, 1]) / d
174                 self.params["A"] = np.zeros([d, d])
175                 self.params["b"] = np.zeros([d, 1])
176                 self.params["beta"] = beta
177
178             rewards = []
179
180             for t in tqdm(range(1, T), desc=self.name):
181                 r_t = X[:, t] / X[:, t - 1]
182                 r_t = r_t[:, None]
183
184                 multiplier = r_t.T @ self.params["x"]
185                 rewards += [np.log(multiplier)[0][0]]
186
187                 grad = r_t / multiplier
188                 hess = grad @ grad.T
189                 self.params["A"] += hess
190                 self.params["b"] += (
191                     hess @ self.params["x"] + (1 / self.params["beta"]) * grad
192                 ) # + for ascent
193                 self.params["x"] = project_A(
194                     self.params["A"], np.linalg.pinv(self.params["A"]) @ self.params["b"]
195                 )
196
197             return rewards
198
199 """

```

A chaque itération, cet algorithme choisit un vecteur qui est la projection de la somme du vecteur choisi à l'itération précédente et d'un vecteur supplémentaire. Alors que pour l'algorithme de descente de gradient, ce vecteur supplémentaire était le gradient de la fonction de coût précédente, pour l'étape de de Newton, ce vecteur est différent : il rappelle la direction dans laquelle la méthode de Newton-Raphson procéderait s'il s'agissait d'un problème d'optimisation pour la fonction de coût précédente. L'algorithme de Newton-Raphson se déplacerait dans la direction du vecteur qui est la Hessienne inverse multiplié par le gradient. Dans la méthode du pas de Newton, cette direction est $A^{-1}\nabla t$, où la matrice A est liée à la Hessienne.

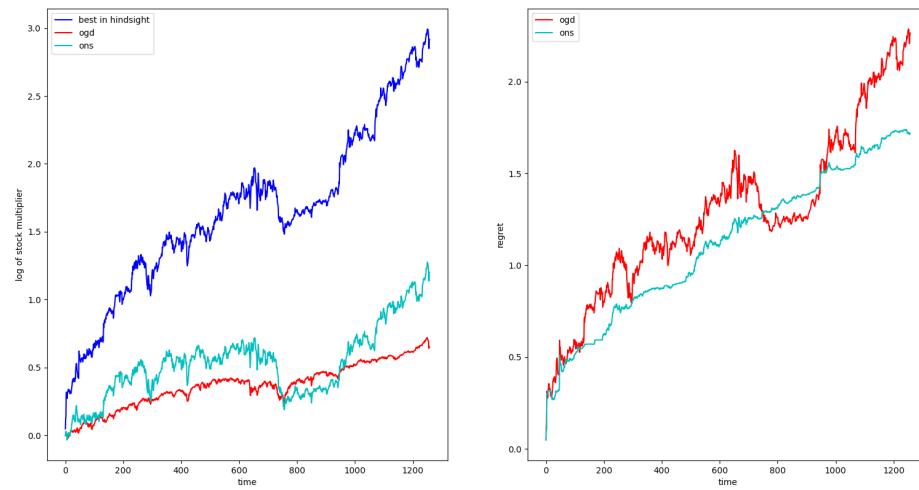
Puisque l'ajout d'un multiple du vecteur de Newton $A^{-1}\nabla t$ au vecteur actuel peut aboutir à un point situé en dehors de l'ensemble convexe, une étape de projection supplémentaire est nécessaire pour obtenir $x(t)$, la décision au temps t . Cette projection est différente de la projection euclidienne standard utilisée par la descente de gradient.

Il s'agit de la projection selon la norme définie par la matrice A , plutôt que selon la norme euclidienne.

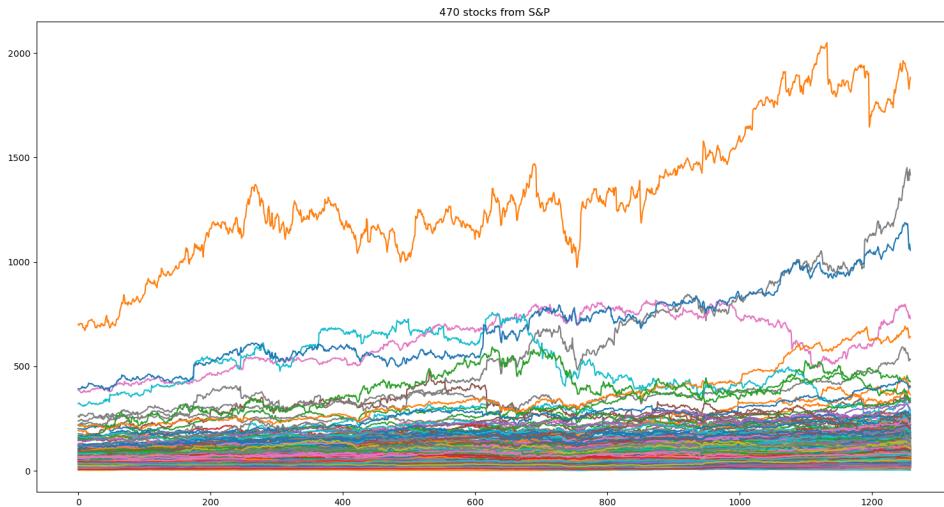
L'avantage de l'algorithme du pas de Newton est sa garantie de regret logarithmique pour les fonctions exp-concaves et permet de le borner.

Résultats

Le code Python provient d'un lien Github où les projections ont été réalisé sur les entreprises de l'index S&P500 à travers un fichier csv comportant les données des actions (ouverture, fermeture, haut, bas, volume, nom) de ces entreprises.



Ces courbes affichent les tendances des algorithmes ogd (online gradient descent) et ons (online newton step) et met en évidence leur performance au fil du temps dans le choix des actifs avec le meilleur rendement et un regret optimal.



Le retour est en échelle logarithmique, donc si la courbe > 0 : l'algorithme gagne de l'argent.

Références

<https://arxiv.org/pdf/1805.07430.pdf>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.296.6481&rep=rep1&type=pdf>

https://github.com/tomnorman/online_portfolio_selection

[https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/2006-Logarithmic
Regret Algorithms for Online Convex Optimization.pdf](https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/2006-Logarithmic_Regret Algorithms for Online Convex Optimization.pdf)

<https://arxiv.org/pdf/1003.0034.pdf>

<https://arxiv.org/pdf/1909.05207.pdf>