Collaborative Filtering
Problème des Bandits

Emma AMBLARD, Eva BOUBA, Charlotte DURAND

22 Octobre 2020

1 Introduction

Dans ce rapport, nous allons vous présenter notre travail effectué sur le problème des bandits appliqués au Collaborative Filtering. Dans un premier temps, nous expliquerons rapidement le problème des bandits ainsi que les principaux algorithmes de résolution, puis nous vous présenterons les résultats expérimentaux obtenus.

2 Le problème des bandits et ses méthodes de résolution

Le problème des bandits est un exemple d'apprentissage par renforcement. Un agent se trouve face à K machines à sous, et il doit décider quelle machine jouer afin de maximiser son gain. Chaque machine a une espérance de gain μ_i qui est inconnue. Pour simplifier, l'agent doit choisir entre exploration - tester différentes machines en espérant en trouver une qui lui fera gagner plus d'argent - et exploitation, utiliser la machine qui à sa connaissance lui rapporte le plus.

2.1 Définitions mathématiques et notations

On considère un problème de bandits à K bras (K machines à sous). On représente la récompense donnée par le bras i à l'instant t par une variable aléatoire $X_{i,t}$ avec $1 \leq i \leq K$. Chaque variable aléatoire a une espérance μ_i et une distribution de probabilités inconnue. À chaque tour, l'agent active un bras i et reçoit une récompense, dépendant de la distribution du bras i.
On note μ^* l'espérance du meilleur bras.
On définit le regret après n essais comme la différence entre la récompense qu'on obtiendrait en utilisant n fois la meilleure machine et l'espérance de la récompense après n essais effectués selon la politique choisie. On note X_t la récompense obtenue à l'instant t. Le but des différents algorithmes est alors de minimiser le regret :

$$r_n = n\mu^* - \mathbb{E}\left[\sum_{t=1}^{n} X_t\right] \tag{1}$$

2.2 Principe des algorithmes implémentés

2.2.1 La méthode ϵ-greedy

Il s'agit de la méthode la plus intuitive. On définit un paramètre ϵ qui va représenter la part d'exploration. On considère donc une variable type Bernoulli de paramètre ϵ dont on observe la réalisation à chaque instant t afin de choisir entre exploration et exploitation :

— Si on obtient une valeur de 0 (avec une probabilité $1-\epsilon$), on exploite les observations faites lors des précédents tours en activant le bras donnant la meilleure récompense en moyenne

— Si on obtient une valeur de 1, on explore en activant un bras choisi aléatoirement parmi tous les bras.

2.2.2 La méthode UCB (Upper Confidence Bound)

Une autre méthode classique introduit la notion d'intervalle de confiance supérieur (Upper Confidence Bound -UCB). Il part du principe de l'optimisme vis-à-vis d'évènements incertains. Cela signifie que pour chaque bras, les valeurs qui lui sont associées sont reliées à un intervalle de confiance supérieur qui est avec une probabilité élevée une surestimation de l'espérance inconnue. On considère $(X_t)_{t=1}^n$ une suite de variables aléatoires indépendantes d'espérance μ. Soit $\hat{\mu}$ l'estimateur de l'espérance : $\hat{\mu} = \frac{1}{n}\sum_{t=1}^n X_t$ On a alors, avec une probabilité $1-\delta$:

$$\mathbb{P}\left(\mu \geq \hat{\mu} + \sqrt{\frac{2\log(1/\delta)}{n}}\right) \leq \delta \quad \forall \delta \in (0,1) \tag{2}$$

Ainsi, à l'instant t, l'agent a observé $T_i(t-1)$ échantillons issus du bras i et a reçu une moyenne empirique $\hat{\mu}(t-1)$. On peut définir la moyenne du bras i "aussi grande que raisonnablement probable" :

$$UCB_i(t-1,\delta) = \begin{cases} \infty & \text{si } T_i(t-1) = 0 \\ \hat{\mu}_i(t-1) + \sqrt{\frac{2\log(1/\delta)}{T_i(t-1)}} & \text{sinon} \end{cases} \tag{3}$$

L'algorithme UCB est défini comme décrit ci-dessous. Pour un nombre de bras K et une probabilité δ :

Algorithm 1 UCB

for $t \in 1, ..., n$ **do**
  Choisir l'action $A_t = \text{argmax}_i UCB_i(t-1,\delta)$
  Observer la récompense X_t et mettre à jour l'intervalle de confiance
end for

Cet algorithme est connu pour bien marcher lorsqu'il y a plus de deux bras. Il présente un défaut important : il nécessite de connaître l'horizon de l'algorithme, c'est-à-dire l'instant n où on arrête l'algorithme.

2.2.3 La méthode Linear UCB

Dans ce modèle, on suppose que la récompense donnée par chaque bras est linéaire par rapport à un vecteur de contexte associé au bras sélectionné, c'est-à-dire que si on tire un bras à l'instant t, la récompense obtenue X_t vaut :

$$X_t = \langle \theta_*, A_t \rangle + \eta_t \tag{4}$$

où η_t est un bruit de nature gaussienne, $A_t \in \mathbb{R}^d$ est le vecteur de contexte représentant le bras sélectionné à l'instant t, et $\theta_* \in \mathbb{R}^d$ est un vecteur que l'on cherche à apprendre.

De la même manière qu'avec la méthode UCB, à chaque instant t, on sélectionne le bras ayant la plus grande borne de confiance supérieure. Dans notre cas, on sélectionne l'action A_t parmi l'ensemble d'actions possibles $\mathcal{A}_t$ ainsi :

$$A_t = \text{argmax}_{a \in \mathcal{A}_t} \langle \hat{\theta}_t, a \rangle + \sqrt{\beta_t} \|a\|_{V_t^{-1}} \tag{5}$$

où $\beta_t \geq 1$ et les $(V_t)_t$ sont des matrices de dimension $d \times d$ définies ainsi :

$$V_0 = \lambda I \text{ avec } \lambda > 0 \quad \text{et} \quad V_t = V_0 + \sum_{s=1}^{t} A_s A_s^T. \tag{6}$$

$\hat{\theta}_t$ est une estimation de θ_* à l'instant t donnée par :

$$\hat{\theta}_t = V_t^{-1} \sum_{s=1}^{t} A_s X_s \tag{7}$$

Pour des paramètres $(\beta_t)_t \geq 1$, $\lambda > 0$ et un contexte pour les bras donnés, l'algorithme Linear UCB est décrit de la manière suivante :

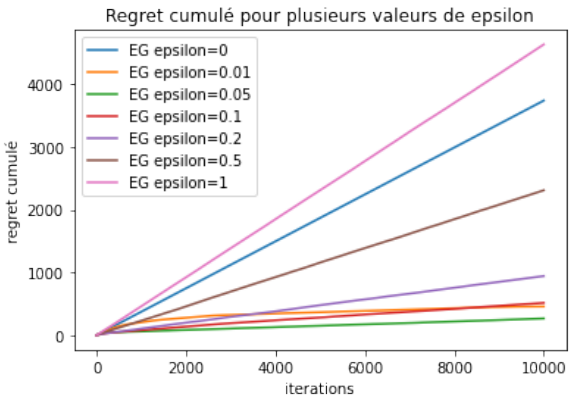
Algorithm 2 LinUCB

$V_0 \leftarrow \lambda I_d$
$\hat{\theta}_0 \leftarrow 0_{\mathbb{R}^d}$
for $t \in 1, ..., n$ **do**
  Observer le contexte A_t pour chaque bras $a \in \mathcal{A}_t$
  for all $a \in \mathcal{A}_t$ **do**
    $UCB_{t,a} \leftarrow \langle \hat{\theta}_{t-1}^T, a \rangle + \sqrt{\beta_t} \sqrt{a^T V_{t-1}^{-1} a}$
  end for
  Sélectionner le bras $A_t = \text{argmax}_{a \in \mathcal{A}_t} UCB_{t,a}$ en choisissant aléatoirement en cas d'égalité, et observer une récompense X_t
  $V_t \leftarrow V_{t-1} + A_t A_t^T$
  $\hat{\theta}_t \leftarrow V_t^{-1} \sum_{s=1}^{t} A_s X_s$
end for

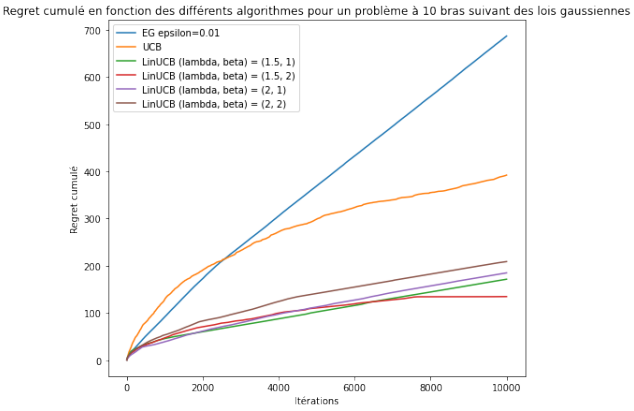
2.3 Evaluation des différents algorithmes sur des datasets d'essai

Dans un premier temps, nous avons voulu coder les 3 algorithmes présentés ci-dessus. Afin de les tester, nous avons utilisé un set de données artificielles créé à partir de variables gaussiennes. On

implémente les différents algorithmes pour $K = 10$ bras, sur 10000 itérations et en moyennant sur 10 runs. On représente à chaque fois le regret cumulé en fonction du nombre d'itérations. Le regret à l'instant n, défini à l'équation 1, doit théoriquement tendre vers 0, lorsque l'on trouve le meilleur bras. Le regret cumulé doit donc tendre vers une asymptote horizontale. Au contraire, lorsque l'on utilise une mauvaise stratégie, le regret cumulé est linéaire, puisque l'on n'améliore pas le regret au cours du temps.
Dans la figure 1a, on va comparer les différentes valeurs de ϵ pour l'algorithme ϵ-greedy. On remarque que la "meilleure" stratégie est celle pour $\epsilon = 0.05$. Au contraire, les stratégies avec $\epsilon = 0$ et $\epsilon = 1$ sont les moins efficaces. $\epsilon = 0$ correspond à une stratégie d'exploitation totale : le premier bras est choisi aléatoirement, puis il est toujours utilisé. $\epsilon = 1$ correspond à une stratégie d'exploration complète : on ne fait que choisir des bras aléatoires, il est donc impossible d'exploiter le meilleur bras.
Dans la figure 1b, on va comparer les 3 différents algorithmes. Pour LinearUCB, le contexte est initialisé avec une matrice générée aléatoirement en suivant une loi uniforme, θ est également généré de façon aléatoire pour $K = 10$ valeurs comprises entre $\frac{-2}{K}$ et $\frac{2}{K}$. Les trois algorithmes sont évalués sur un bandit dont la moyenne de chaque bras est égale au produit scalaire entre θ et son contexte. LinUCB est présenté avec plusieurs couples de paramètres λ et β. Afin de faciliter la comparaison, une valeur de $\epsilon = 0.01$ est choisie pour ne pas écraser les autres courbes.
On peut premièrement remarquer que la stratégie ϵ-greedy est la moins efficace des 3 stratégies proposées. Ensuite, LinUCB semble être une stratégie plus intéressante que UCB, peu importe les paramètres proposés ici. Dans la figure 2, on peut observer à gauche l'influence du paramètre λ et à droite l'influence du paramètre β. λ peut être vu comme un paramètre de régularisation lors de la minimisation de la fonction qui permet de calculer $\hat{\theta}_t$. Il est donc compliqué d'interpréter son évolution. β représente une constante qui est associée à l'intervalle de confiance contenant la moyenne de chaque bras. Si β est trop élevé, on augmente les chances d'erreurs lors du choix du bras. S'il est trop faible, l'algorithme peut penser avoir trouvé le bon bras alors que ce n'est pas le cas. Il faut donc trouver un bon compromis entre ces 2 phénomènes.



(a) Evaluation de ϵ-greedy pour plusieurs valeurs de ϵ



(b) Evaluation des différents algorithmes sur un set de données générées à partir de variables gaussiennes

FIGURE 1 – Comparaison des différents algorithmes

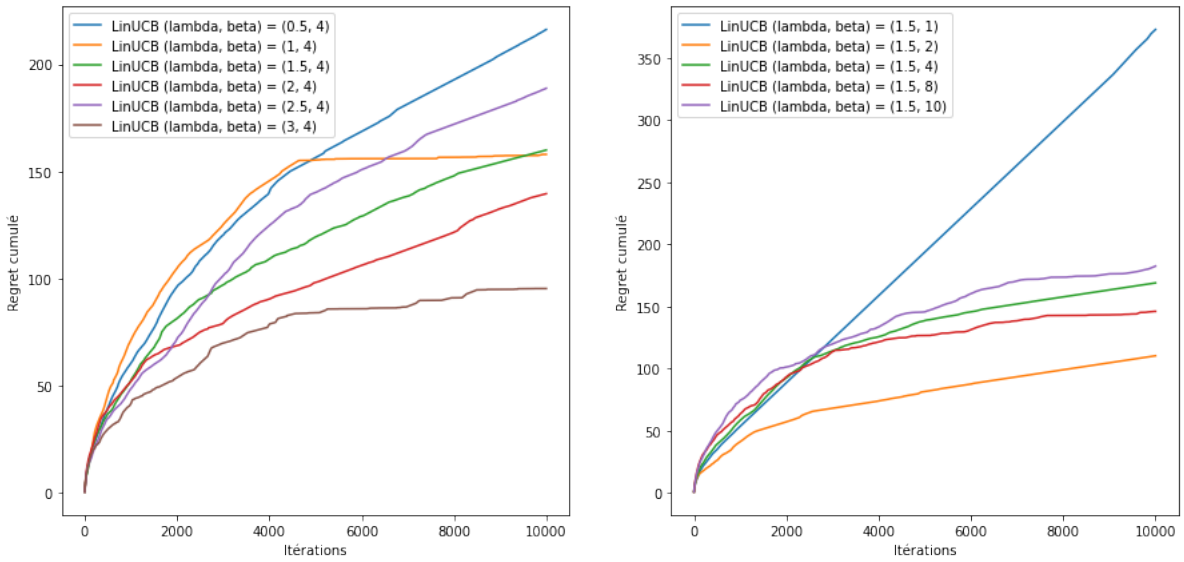


FIGURE 2 – Influence des paramètres β et λ sur le programme LinUCB

3 L'application des bandits au problème de recommandation

La recommandation personnalisée de films à des utilisateurs peut être modélisée sous la forme d'un problème de bandits. Les machines à sous sont alors représentées par les films et l'objectif du système de recommandation est de maximiser la somme des notes que les utilisateurs mettent aux films qu'il leur recommande. Cet objectif peut être reformulé comme la minimisation de la différence entre la meilleur note que mettrait l'utilisateur à un film qu'il n'a pas encore vu, et la note du film qu'on lui propose. Contrairement au problème de bandits initial où l'on peut jouer plusieurs fois la même machine, le système ne peut recommander plusieurs fois le même film au même utilisateur. L'utilisation d'informations préalable s'avère alors nécessaire. Ces informations peuvent être sur les utilisateurs (âge, profession, nationalité...), sur les films (genre, année, nationalité...) ou bien éventuellement obtenues grâce aux notes déjà récoltées précédemment. C'est pour cette troisième méthode que nous avons opté.

3.1 Pré-traitement des données

Soient n utilisateurs, k films et la matrice $R = (r_{ij})_{i\in[\![n]\!],j\in[\![k]\!]}$ des notes données par les utilisateurs aux films jusqu'à présent. Nous divisons R en 3 matrices :

— La matrice $M = (r_{ij})_{i\in[\![p]\!],j\in[\![k]\!]}$ qui sert à la création du contexte

— La matrice d'entraînement $M' = (m'_{ij})_{i\in[\![n-p]\!],j\in[\![k]\!]}$

— La matrice de test $M'' = (m''_{ij})_{i\in[\![n-p]\!],j\in[\![k]\!]}$

Avec $\forall(i,j), (m'_{ij} = 0 \wedge m''_{i,j} = r_{p+i,j}) \vee (m'_{ij} = r_{p+i,j} \wedge m''_{i,j} = 0)$. Nous avons donc $R = \left[\begin{array}{c} M \\ \hline M' + M'' \end{array}\right]$

3.1.1 Le contexte

Nous avons décidé de construire notre contexte à partir d'un historique de notes données par les utilisateurs. Afin de réduire la dimension de ce contexte et ainsi augmenter la vitesse des calculs nécessaires par la suite, nous factorisons M. Pour cela, nous choisissons $d \in [\![1, min(p, k)]\!]$ et nous effectuons une décomposition en valeurs singulières $M = U\Sigma V^*$. C'est la matrice V^* ainsi obtenue qui nous sert de contexte.

3.1.2 Les données d'entraînement et de test

La partie de la matrice R qui n'a pas été utilisée pour créer le contexte nous fournit les données d'entraînement et de test. La séparation se fait en affectant chaque valeur non nulle de la matrice aléatoirement soit à la matrice M', soit à la matrice M'' mais pas au deux.

3.2 L'entraînement et l'évaluation du bandit

Le but de l'algorithme est d'apprendre le vecteur θ_u associé à chaque utilisateur non présent dans M. Pour chaque utilisateur, nous effectuons T itérations de LinUCB. Les $(\theta_i)_{i=1,...T}$ obtenus à chaque itération nous permettent de calculer l'estimation des notes que donnerait u à chaque film. Nous pouvons alors comparer ces valeurs aux valeurs présentent dans M' et M'' pour évaluer le bandit sur les données d'entraînement et de test respectivement. L'erreur de prédiction au temps t sur les données d'entraînement pour l'utilisateur u est calculée comme suit :

$$err^{(e)}(u)_t = \left(\frac{\sum_{i|m'_{ui} \neq 0} (\langle \theta_t, V^* \rangle_i - M'_{u,i})^2}{\#\{m'_{ui} \neq 0 \mid i = 1, ...k\}} \right)^{1/2}$$

Celle pour les données de test comme suit :

$$err^{(t)}(u)_t = \left(\frac{\sum_{i|m''_{ui} \neq 0} (\langle \theta_t, V^* \rangle_i - M''_{u,i})^2}{\#\{m''_{ui} \neq 0 \mid i = 1, ...k\}} \right)^{1/2}$$

Finalement, les erreurs totales au temps t sont :

$$err_t^{(e)} = \left(\frac{\sum_u \sum_{i|m'_{ui} \neq 0} (\langle \theta_t, V^* \rangle_i - M'_{u,i})^2}{\#\{m'_{ui} \neq 0\}} \right)^{1/2}$$

et

$$err_t^{(t)} = \left(\frac{\sum_u \sum_{i|m''_{ui} \neq 0} (\langle \theta_t, V^* \rangle_i - M''_{u,i})^2}{\#\{m''_{ui} \neq 0\}} \right)^{1/2}$$

En plus de ces erreurs, il est toujours possible de considérer le regret cumulé pour étudier la convergence de l'algorithme. Chaque utilisateur donne lieu à un vecteur de regret que nous agrégeons en moyenne. Plusieurs paramètres ont une influence sur l'erreur et le regret cumulé, notamment la dimension d de factorisation, le nombre de notes dans M', les paramètres λ et β de LinUCB. D'après nos tests, un trop faible β induit moins d'exploration ce qui a pour effet de fixer l'algorithme plus rapidement sur un film et de donc moins apprendre des notes mises à des films moins aimés. Cela

entraîne un regret cumulé plus faible, mais en revanche la RMSE tend à être plus élevée comme le montrent les figures 3 et 4. Les choix des valeurs de d, λ et β semblent également être liés avec la nécessité de diminuer λ ou d'augmenter β lorsque d diminue sans quoi l'exploration est trop faible. Les résultats semblent s'améliorer lorsque d augmente comme le montre la figure 5. Cela vient probablement du fait qu'augmenter d signifie représenter les utilisateurs par un vecteur de plus grande dimension et donc plus d'informations. Néanmoins, cela augmente les temps de calcul et nécessite plus d'itération pour converger.

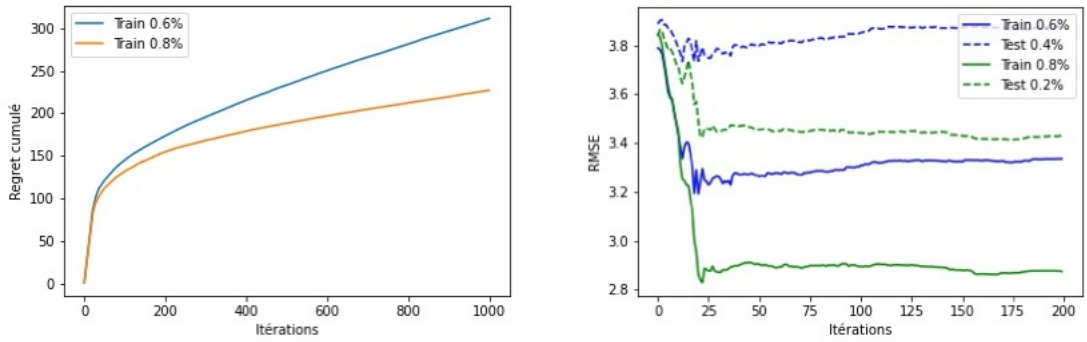


FIGURE 3 – LinUCB pour la recommandation de films avec $d = 20$ et $\beta = 20$

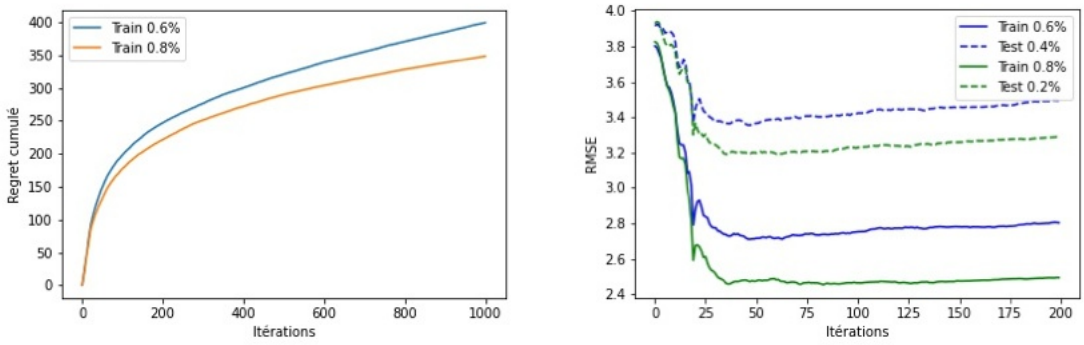


FIGURE 4 – LinUCB pour la recommandation de films avec $d = 20$ et $\beta = 250$

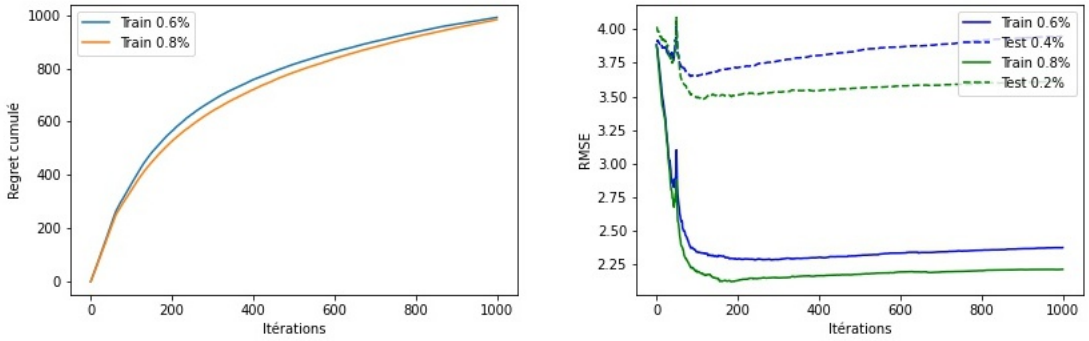


FIGURE 5 – LinUCB pour la recommandation de films avec $d = 50$ et $\beta = 100$

L'intérêt principal de l'algorithme des bandits est qu'il permet de faire des recommandations à des utilisateurs non connus. Notre implémentation ne permet que du warm start, c'est à dire qu'en plus du contexte, il nous faut également au moins une note d'un utilisateur pour apprendre son vecteur. Les prédictions obtenues sont tout de même assez fidèles. En comparaison, la RMSE de la factorisation sur l'ensemble des notes est de 2.4 pour $d = 20$ et de 1.08 pour $d = 50$.