Algorithmes de recommandation séquentielle
Stage de L3 - ENS de Cachan

Mathias Sablé Meyer

Équipe SequeL, LIFL (UMR CNRS), INRIA, Villeneuve d'Ascq, France

dirigé par Romaric Gaudel

Juin à mi-juillet 2014

Résumé

Ce document présente mon approche face aux problèmes d'apprentissage séquentiel, et plus précisément à des problèmes de recommandation *en-ligne*, dans lesquels on ne dispose à l'origine d'aucune information sur l'environnement.

Partant du domaine de la recommandation, on constate que les méthodes classiques tentent, à un instant donné, de coller le plus possible aux données connues. Elles ne prennent pas en compte la façon dont elles ont recueilli ces données, et ne pensent pas à chercher à recueillir de façon optimale ensuite. Mon approche est novatrice puisqu'elle aborde le problème en faisant intervenir un point de vue issu d'un autre domaine, celui des bandits. Ainsi, on va tenir compte de l'effet du système sur la collecte des données, et se placer dans une optique dynamique d'apprentissage.

Cette approche n'est pas moins efficace en termes de temps de calcul, et offre de meilleurs résultats pratiques.

1 Introduction

Le contexte général

La recommandation désigne le domaine de recherche motivé par des problèmes dans lesquels, possédant une certaine quantité d'information à propos d'utilisateurs et d'items, on veut recommander des items à des utilisateurs suivant leurs goûts.

Ce problème trouve des applications immédiates : recommandation de livres, d'article de journaux, de musique. Des services comme Deezer, Spotify, ou Amazon, sont intéressés car ils possèdent de grandes bases de données de feedback de leurs utilisateurs sur les produits proposés et qu'ils cherchent à leur proposer des produits adaptés.

Un indicateur de l'intéret porté dans ce domaine et de la demande de recherche active est le nombre de concours organisés par des organismes privés, ce qui fournit quelques bases de données sur lesquelles travailler (on pense notamment à Netflix, MovieLens, ou encore à Yahoo!).

On peut formaliser ce problème de plusieurs manières, j'ai choisi de le voir comme un problème de complétion de matrices : dans le passé chaque utilisateur s'est vu présenté des produits ; pour chaque présentation, l'utilisateur a fourni un retour (note, achat ou non, etc.) ; ces retours sont stockés dans une matrice dont chaque ligne représente un utilisateur et chaque colonne un produit ; la recommandation d'un produit à un utilisateur passe alors par la détermination des valeurs manquantes dans cette matrice.

Le problème étudié

Dans la pratique, la recommandation est un problème séquentiel : les utilisateurs arrivent, on recommande, on collecte les retours des utilisateurs, on apprend, puis le processus recommence.

Dans ce contexte l'objectif de chaque recommandation est ambivalent : (i) on veut faire plaisir à l'utilisateur en lui recommandant le produit le plus adapté à ses besoin, et (ii) on veut comprendre les goûts de l'utilisateur et des autres utilisateurs sur les différents produits.

Or les façons les plus efficaces de gérer ce choix ne sont à ce jour pas du tout prises en compte par la communauté de la recommandation, qui applique une stratégie gloutonne qui favorise systématiquement le point (i), ce qui, comme on le verra formellement plus tard, n'est pas optimale.

Il s'agissait alors de trouver un ordonnancement optimal des recommandation, sous les contraintes de l'environnement, pour maximiser la somme des gains à travers toute la résolution du problème, et non plus juste à un instant donné.

La contribution proposée

J'améliore les techniques utilisées actuellement en introduisant un concept issu d'un autre domaine de recherche, celui des *bandits*, qui voit ces problèmes de façon dynamique et non statique, le paradigme de ce domaine étant synthétisé sur la figure 1.

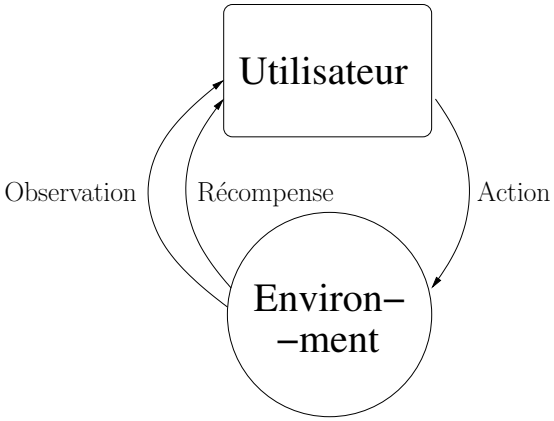

FIGURE 1 – Contexte général de la recommandation séquentielle.

D'une part, j'ai adapté l'une des stratégies les plus efficaces dans le domaine des bandits classiques, $\varepsilon - greedy$, au problème de la recommandation en-ligne, afin de montrer qu'il existe des façons simples de gérer le conflit qui donnent de meilleurs résultats que les méthodes classiques.

D'autre part, j'ai mené une réflexion importante sur la manière d'aborder les calculs pour que la solution que je propose soit viable en terme de temps de calculs par rapport aux méthodes qui sont utilisées actuellement.

Les arguments en faveur de sa validité

Dans le problème que j'ai étudié, mes résultats pratiques sont sensiblement meilleurs que les algorithmes qui existaient déjà sur des données artificielles, et le sont dans une moindre mesure sur des données réelles. Par ailleurs ma façon de faire nécessite peu d'hypothèses, et peut donc se transposer aisément à beaucoup de contextes dans le domaine de la recommandation.

Cependant, dans ce cadre les hypothèses sont assez faibles, en particulier les hypothèses d'indépendance pour les résultats de probabilité, et je n'ai pas pu conduire entièrement la preuve, quoique les idées principales et les intuitions soient là.

Le bilan et les perspectives

Mes résultats étant directement applicables à des bases de données de taille raisonnable, ils peuvent servir *en l'état* dans des cas pratiques de recommandation.

Mais surtout, ma recherche montre que les algorithmes utilisés jusqu'ici ne sont pas particulièrement adaptés, or ils ne sont pas non plus efficaces en termes de temps de calcul.

S'ouvre donc un champs pour la recherche des algorithmes optimaux dans ce domaine. En effet, la vitesse de convergence de l'algorithme proposé n'est optimale que pour l'ordre de grandeur, pas pour la constante. Des algorithmes explorant de façon plus subtiles permettraient de diminuer cette constante [1].

Il convient donc de repenser entièrement la façon classique d'aborder ces problèmes, et la suite de la recherche dans ce domaine pourrait tout à fait être basée là-dessus.

1. À titre d'exemple, la section B.2 de l'annexe montre comment les algorithmes UCB, LinUCB et LinOFUL ont un regret qui évolue selon la même tendance que ε-greedy mais avec des constante plus petite

2 Contexte

Je vais commencer par vous présenter les deux domaines qu'il est important de connaître pour pouvoir ensuite se pencher sur le travail que j'ai effectué.

2.1 Recommandation

Je vais d'abord présenter le contexte général des problèmes de recommandation, avec lesquels il faut se familiariser pour comprendre la suite de mes recherches.

Dans sa formalisation la plus courante , il consiste en la résolution du problème suivant : étant donné une matrice de taille donnée et de rang petit devant sa taille, dont on a masqué un certain nombre de valeurs, comment retrouver la matrice de départ avec la plus grande précision possible.

Il existe plusieurs algorithmes pour traiter de ce problème, j'en ai étudié deux. Le premier est basé sur l'élimination du bruit dans la décomposition en valeurs singulières (SVD). Quoique celui-ci donne des résultats satisfaisants, il peut-être très coûteux pour de grandes matrices, et il n'est pas très robuste à la façon dont on l'initialise. Le second algorithme que j'ai étudié, ALS (Alternating Least Square) est connu dans la communauté et donne de très bon résultats, tout en convergeant vite. Le problème majeur est qu'il faut l'initialiser dans un voisinage raisonnable de la solution pour qu'il converge vers celle-ci. Si ce point n'est pas vérifié, il convergera, mais vers un résultat voisin de celui que l'on cherche. Dans la suite on fera l'hypothèse que l'algorithme converge vers la bonne solution.

Formellement, ALS résout, en alternant la recherche pour U et pour V de rang k, le problème suivant :

$$\arg\min_{U,V} \sum_{i,j\ connus} (M_{i,j} - UV^\top)^2 + \|U\| + \|V\|$$

On remarque que chaque étape de la résolution est constituée d'une multitude de résolution des moindres carrés indépendants — un par ligne/colonne de la matrice — et ce problème étant bien connu, on en connaît des solutions efficaces et parallélisables.

Le fonctionnement de cette méthode est décrit plus en détail dans la la figure 2

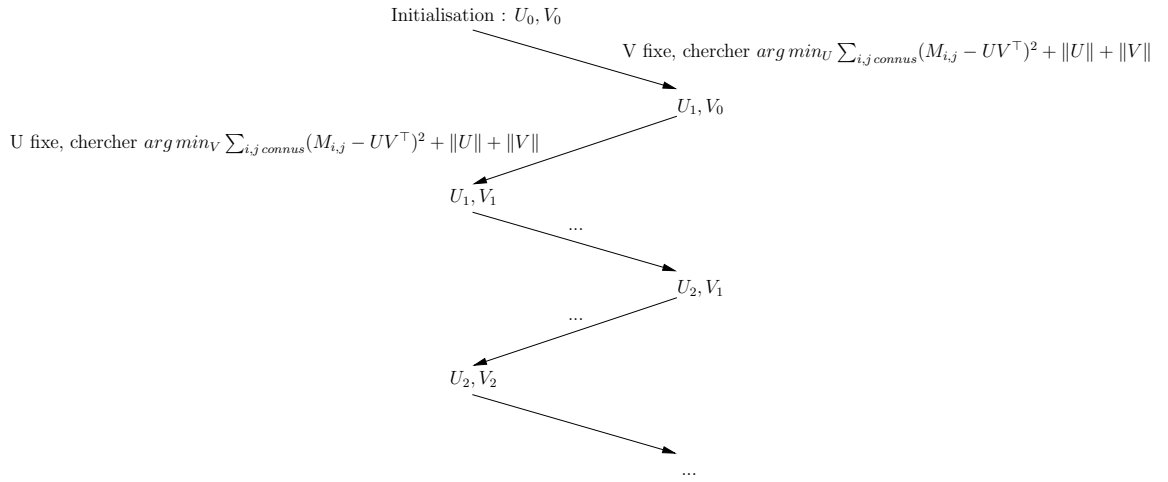

FIGURE 2 – Le fonctionnement d'ALS : une succession de résolution des moindres carrés indépendants.

On peut choisir d'arrêter l'algorithme soit quand la distance entre deux U ou deux V successifs n'est plus significative, soit au bout d'un certain nombre d'itérations — il n'est généralement pas pertinent d'en faire plus d'une vingtaine.

La matrice ainsi décomposée fait apparaître des propriétés intrinsèques de l'environnement. Ainsi si la matrice contient la note $n(u, v)$ à la case (u, v) attribuée par l'utilisateur u à l'item v, cette décomposition va faire apparaître d'une part des propriétés intrinsèques au groupe des utilisateurs, et d'autre part des propriétés intrinsèques au groupe des items. Quoique ceci n'intervienne pas directement dans la suite de la recherche, on obtient de cette façon une intuition de ce que cette décomposition signifie.

2.2 Bandits

Un domaine très exploré dans la recherche en apprentissage est celui des bandits et ses nombreuses variantes. Je vais en exposer brièvement les tenants et les aboutissants, pour comprendre en quoi la démarche pour la résolution de problèmes de recommandation semble naturelle à cette lumière.

Dans sa version la plus simple, on se donne un nombre fini de bras, chaque bras i étant associé à une distribution ν_i d'espérance μ_i. Les distributions et leur espérance sont inconnues. On joue alors n pas de temps. À chaque pas de temps t, on choisit un bras i_t et l'on reçoit une récompense tirée selon ν_{i_t}. Le but est de maximiser le gain cumulé, c'est à dire la somme des récompenses reçues au cours des n pas de temps, en ne disposant comme information que des couples (bras, récompense) de chaque instant passé. On appelle stratégie un algorithme qui choisit le prochain bras à tirer, en fonction de la séquence des couples (bras, récompense) connus.

On remarque qu'un algorithme qui suit un paradigme glouton, c'est à dire qui tente de trouver le meilleur bras mais qui une fois qu'il pense l'avoir trouvé ne fait plus que l'exploiter, n'est pas optimal car il n'est pas robuste en cas d'erreur dans son estimation des bras.

De même, un algorithme qui explore systématiquement n'est pas optimal non plus, puisqu'il va dans son exploration tirer des bras sous-optimaux, et ainsi prendre du retard sur le gain cumulé.

Il y a ici un conflit entre l'exploitation des bras que l'on pense être les meilleurs, et l'exploration pour s'assurer que l'on n'est pas en train de choisir un bras sous-optimal. Il existe de nombreuses façons de résoudre ce conflit, les plus connues étant UCB (fig. 3) (Upper Confidence Bound) ainsi qu'$\varepsilon - greedy$ (fig. 4) (et sa variant $\varepsilon_n - greedy$).

UCB choisit toujours le bras dont la borne supérieure de confiance est la plus élevée, et fait grandir artificiellement celle-ci avec le temps (en quelque sorte, on perd confiance en nos tirages avec le temps), et $\varepsilon - greedy$ explore uniformément avec probabilité ε, et exploite le reste du temps.

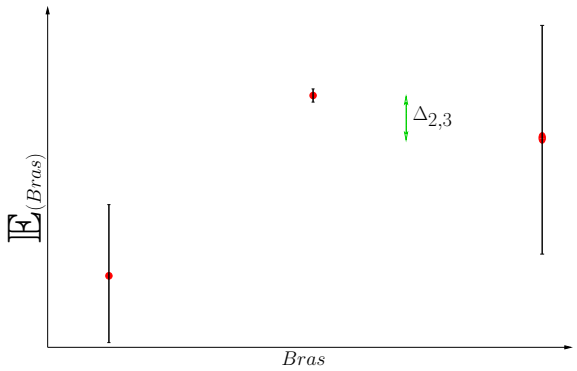

FIGURE 3 – UCB : choisir le bras de plus grande borne haute de confiance.

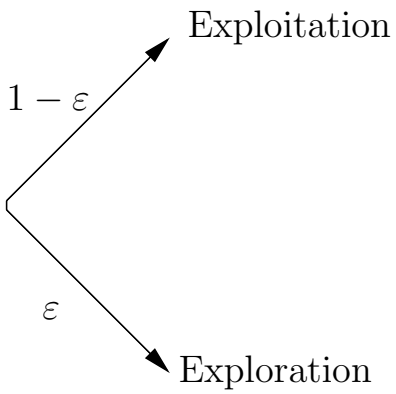

FIGURE 4 – ε doit être fonction du temps pour des résultats optimaux.

La maximisation du gain cumulé nous amène à vouloir minimiser une autre valeur : le regret R_n. C'est ainsi que l'on mesure l'efficacité d'une stratégie : on se donne une stratégie optimale — qui est souvent omnisciente —, et le regret est alors l'écart entre l'espérance de notre stratégie et l'espérance de la stratégie optimale.

Dans le cas le plus simple où il existe **un** *meilleur bras*, on a donc :

$$R_n = n * \mathbb{E}(Meilleur\ bras) - \sum_{t=0}^{n} \mathbb{E}(Bras\ joué\ au\ tour\ t)$$

Les résultats [2] sur ces deux stratégies nous montrent que l'exploration doit se faire en $O(log(t))$ où t est le nombre de bras que l'on a tiré.

Il existe de nombreux autres contextes pour les bandits : les bras peuvent exister dans un espace continu (recherche du maximum d'une fonction [3]), ne pas être tous actifs en même temps (mortal bandits, [5]), ou contenir des informations non seulement sur eux-même, mais sur l'ensemble des bandits (Contextual Bandits, [6], voir en Annexe : LinUCB/LinOFUL).

Ces deux derniers contextes m'ont servis, d'une part pour mieux cerner le problème, et d'autre part pour apporter des éléments de preuve.

On trouvera des exemples des résultats en annexes pour différents algorithmes classiques.

Ces algorithmes trouvent leurs applications dans le cas de recommandation où l'on ne prend en compte qu'un seul utilisateur par exemple (ou alors on choisit de ne pas considérer les différences entre utilisateurs, on s'intéresse à un *utilisateur moyen*), mais on peut aussi penser au domaine de la recherche en médecine [7], dans laquelle on veut converger vers le meilleur résultat en bornant l'exploration, particulièrement coûteuse. Il faudrait alors se pencher sur une version à *budget borné*, c'est à dire dans laquelle on ne peut se permettre qu'une exploration limitée [9].

2.3 Notations

Puisque je suis à l'intersection de deux domaines ayant leurs propres notations, je vais redéfinir mon propre système de notations — hérité des deux autres — pour la suite.

On définit :

- M la matrice *à compléter*, U et V deux matrices à k colonnes, telles que $UV^\top = M$. Comme on l'a vu précédemment, on peut voir U et V comme des façons de représenter des propriétés, d'une part des utilisateurs, d'autre part des produits.
- Les lettre surmontées d'un chapeau ($\hat{U}$, $\hat{V}$, etc.) désigneront toujours l'estimateur, dans notre algorithme, de l'objet que représente la lettre correspondante.
- Le lettres suivies d'une étoile ($M^*_{u,v}$, etc.) désignent toujours le choix optimal ou la **vraie** valeur.
- On désigne par observation le triplet (u, v, r) où u représente l'utilisateur, v le produit, et r la récompense associée (on remarque qu'on a toujours $M_{u,v} = r = M^*_{u,v} + bruit$)

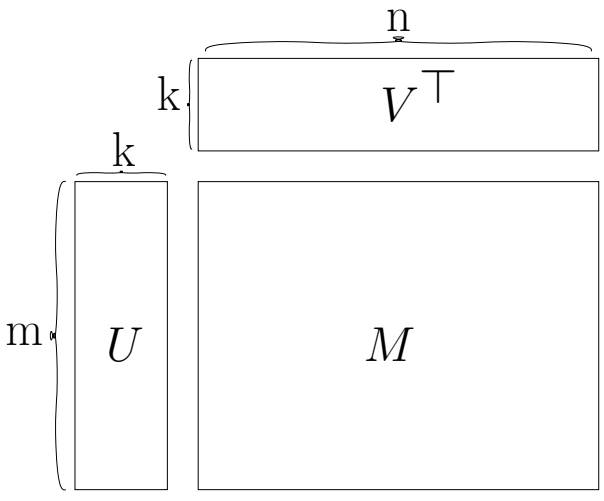

FIGURE 5 – Structure du problème à résoudre

3 Recommandation Séquentielle

Il existe plusieurs façons de considérer la recommandation séquentielle, je vais vous présenter la façon dont je l'ai abordée.

Un site veut recommander un ou plusieurs produits à un utilisateur qui charge une page. On suppose que l'utilisateur retourne alors un avis sur le(s) produit(s) recommandé(s). Le regret mesure alors l'erreur commise par rapport à ce que l'on aurait pu recommander de mieux, ce qui est exactement ce que l'on cherche à minimiser.

On peut poser le problème de la façon suivante : on se donne un environnement qui dispose d'une matrice, et d'une matrice de même dimension dont toutes les valeurs ont été masquées. L'environnement impose une ligne — i.e. un utilisateur — et l'algorithme tente de deviner parmi les valeurs qu'il ne connaît pas encore, la colonne qui maximise la valeur sur la ligne. L'environnement révèle alors la vraie valeur à la case choisie, et calcul le regret (voir formule (1)). L'algorithme prends en compte la nouvelle valeur, et la séquence peut recommencer.

$$R_n = \sum_{k=1}^{n} (\arg\max_{j \, cachés}(M_{u_k,j}) - M_{u_k,C_k}), \tag{1}$$

u_k est l'utilisateur imposé par l'environnement à l'instant t, et C_k est la colonne choisie par l'algorithme à l'instant t.

Cette définition du regret n'est pas exactement la même que dans le cas des bandits, mais elle mesure la même chose, et correspond bien à ce que l'on veut minimiser. À noter que ce calcul du regret introduit une sorte de double punition pour un algorithme qui fait une erreur : d'abord, il y perd la différence avec la meilleur valeur, mais en plus un algorithme qui aurait deviné la bonne ne peut plus jamais se tromper, réduisant ainsi toutes ses erreurs futures.

3.1 $\varepsilon_n - greedy$ pour la recommandation

Les solutions actuellement utilisées pour ce type de problème sont gloutonnes, et notre expérience des bandits nous apprend que ce n'est pas optimal.

Deux algorithmes simples peuvent alors être adaptés à cette situation : LinUCB et $\varepsilon_n - greedy$. Je me suis attaqué à ce second cas car la preuve me semblait plus facile à étendre dans notre nouvelle situation.

Base L'algorithme peut être décrit de la façon suivante : à chaque observation, on effectue un ALS pour trouver $\hat{U}$ et $\hat{V}$. Lorsque l'environnement nous demande de recommander un produit pour un utilisateur, on recommande avec probabilité $\varepsilon(t)$ un produit tiré uniformément au hasard, et avec probabilité $1 - \varepsilon(t)$ de façon gloutonne : (voir algo. 1)

Data: u, M
Result: v vérifiant : $M_{u,v}$ n'est pas connue.
if *rand()* $< \varepsilon(t)$ **then**
  v = randint(0, #Produits)
else
  $\hat{U}$, $\hat{V}$ = ALS(M)
  $v = \arg\max_v \{\hat{U}_u \hat{V}_v^\top | M_{u,v} \, inconnue\}$
end
return v

Algorithm 1: Un $\varepsilon - greedy$ naïf.

On peut alors voir l'exploration pure comme le cas où ε est la fonction constante égale à 1, et l'algorithme glouton comme le cas où c'est la fonction constante égale à zéro.

Peter Auer a montré [2] que dans le cas des bandits, la fonction optimale était $\varepsilon(t) = \frac{C}{t}$, cependant en un sens dans notre cas, l'exploration étant forcée par le problème (si on fait l'analogie avec les bras, on ne peut tirer ceux-ci qu'une fois), on peut craindre d'être obligé d'explorer encore **plus** souvent. On verra plus tard que, même si cette idée peut paraître contre-intuitive, elle est confirmée par la pratique : on serait alors plus proche d'un cas de *bandits adversarial* pour lesquels les tirages ne sont pas i.i.d.

En pratique, l'algorithme d'ALS est très coûteux : en effet, il fixe U, puis fait une recherche d'un minimum en faisant varier V, puis fixe V, fait de même pour U, et ainsi de suite jusqu'à ce qu'une

certaine condition soit remplie. Il ne serait donc pas raisonnable d'effectuer un ALS complet à chaque nouvelle observation.

Mais sa structure même nous permet de ne pas recommencer un calcul complet : on écrit une autre version d'ALS qui n'effectue qu'une passe pour $\hat{U}$ et une passe pour $\hat{V}$, que je noterai *UpdateFullALS*. L'idée est de maintenir les calculs déjà faits le plus longtemps possible, et périodiquement de recommencer à zéro pour s'assurer qu'on n'est pas en train de diverger (voir algo. 2).

À noter que cette étape n'est sans doute pas nécessaire si on se place dans l'hypothèse où l'ALS converge directement vers le résultat qui nous intéresse, mais ce n'est pas exactement le cas en pratique. On prend donc cette sécurité pour s'assurer que le travail que l'on fait ne nous fait pas converger vers une solution sous-optimale.

Data: u, t, *period*, M, $\hat{U}$, $\hat{V}$
Result: v vérifiant : $M_{u,v}$ n'est pas connue.
if *rand()* $< \varepsilon(t)$ **then**
  v = randint(0, #Produits)
else
  if *(t mod period) = 0* **then**
    $\hat{U}$, $\hat{V}$ = ALS(M) ; `// On recalcule tout.`
  else
    $\hat{U}$, $\hat{V}$ = UpdateFullALS(M, $\hat{U}$, $\hat{V}$); `// On incrémente.`
  end
  $v = \arg\max_v \{\hat{U}_u \hat{V}_v^\top | M_{u,v}\, inconnue\}$
end
return v

Algorithm 2: Un $\varepsilon - greedy$ sans ALS systématique.

Vers un ALS stochastique. Malheureusement, sur de grandes matrices, même une unique passe d'ALS est très coûteuse, et cet algorithme devient lui même trop coûteux. En remarquant qu'une observation est un résultat très *local*, il est plus judicieux de faire une mise à jour locale de l'ALS autour de cette nouvelle observation que de mettre à jour l'ensemble des données.

En suivant la philosophie du gradient stochastique [8], on peut se contenter de cette mise à jour locale même si elle ne nous fait pas elle-même converger, du moment que son espérance nous fait converger.

J'ai donc adapté cet état d'esprit à l'algorithme d'ALS en mettant à jour seulement quelques points au voisinage de la dernière observation, je noterai cet algorithme *UpdateStochastALS*. Globalement, l'algorithme est le même, mais l'ALS prend maintenant comme paramètre la dernière observation, et s'en sert pour faire une mise à jour locale de $\hat{U}$ et $\hat{V}$ au lieu de faire une passe complète de l'algorithme, qui mettrait à jour toutes les données.

La fréquence des ALS complets est une question difficile : faire des calculs complexes d'ALS complets au début ne paye pas parce que la quantité d'informations dont on dispose est très faible. De même, il existe une taille telle qu'il n'est plus nécessaire de travailler avec cet algorithme, parce que l'on peut compléter entièrement la matrice avec une grande précision. J'ai pour l'instant opté pour une fréquence de l'ordre de la racine du nombre d'éléments à deviner, mais ce point mériterait approfondissement.

Des résultats théoriques [4] nous donnent une majoration sur le rang pour éviter l'over-fiting, cependant la façon dont celui doit varier avec le temps est une autre question délicate : on peut penser qu'il faut faire des approximation de faible rang au début parce qu'on dispose de peu d'informations, mais que celui-ci doit croître avec le temps.

Dans mon implémentation j'ai choisi une croissance linéaire de sorte qu'on atteigne la majoration de [4] à la fin de l'algorithme.

3.2 Intuitions sur la preuve

Une majoration. L'idée d'une preuve de la validité du résultat est de montrer une majoration sur la probabilité de recommander le mauvais produit à un instant donné.

Voir en annexes la preuve du comportement en *log* de l'algorithme $\varepsilon_n - greedy$ dans le cas des bandits simples, dûe à Auer, pour la structure générale de la preuve, que l'on essaye d'adapter ici à

Data: u, t, *period*, M, $\hat{U}$, $\hat{V}$, u_{t-1}, v_{t-1}
Result: v vérifiant : $M_{u,v}$ n'est pas connue.
if *rand()* $< \varepsilon(t)$ **then**
  v = randint(0, #Produits)
else
  if *(t mod period) = 0* **then**
    $\hat{U}$, $\hat{V}$ = ALS(M)
    `// Recalcul global de` $\hat{U}$ `et` $\hat{V}$
  else
    $\hat{U}$, $\hat{V}$ = UpdateStochastALS(M, $\hat{U}$, $\hat{V}$, u_{t-1}, v_{t-1})
    `// Calcul local uniquement.`
  end
  $v = \arg\max_v \{\hat{U}_u \hat{V}_v^\top | M_{u,v}\, inconnue\}$
end
return v

Algorithm 3: Un $\varepsilon - greedy$ avec ALS local.

notre problème.

Plaçons nous à un instant t et notons u l'utilisateur correspondant, j un choix non optimal, j^* le choix optimal, C le choix effectué par l'algorithme et K le cardinal de l'ensemble $\{n \mid M_{u,n}\, inconnue\}$

Séparons d'abord entre le cas ou le produit est choisi *au hasard*, et le cas où notre estimateur était biaisé :

$$\mathbb{P}(C = j) = \frac{\varepsilon(t)}{K} + \left(1 - \frac{\varepsilon(t)}{K}\right) \mathbb{P}(j \text{ jamais tiré}) \mathbb{P}(U_{u,j} V_{u,j}^\top = \arg\max_{k\, libres} U_{u,k} V_{u,k}^\top)$$

Pour se rapprocher de la preuve de Auer on peut effectuer les majorations suivantes :

$$\begin{array}{ll} & \mathbb{P}(C = j) = \frac{\varepsilon(t)}{K} + \left(1 - \frac{\varepsilon(t)}{K}\right) \mathbb{P}(j \text{ jamais tiré}) \mathbb{P}(U_{u,j} V_{u,j}^\top = \arg\max_{k\, libres} U_{u,k} V_{u,k}^\top) \\ \Longrightarrow & \mathbb{P}(C = j) = \frac{\varepsilon(t)}{m} + (1 - \varepsilon(t)) \mathbb{P}(U_{u,j} V_{u,j}^\top = \arg\max_{k\, libres} U_{u,k} V_{u,k}^\top) \\ \Longrightarrow & \mathbb{P}(C = j) \leqslant \frac{\varepsilon(t)}{m} + (1 - \varepsilon(t)) \mathbb{P}(U_{u,j} V_{u,j}^\top > U_{u,j^*} V_{u,j^*}^\top) \end{array}$$

Il s'agit maintenant de trouver la fonction ε décroissant le plus vite telle que le terme de droite soit négligeable devant celui de gauche, c'est à dire qu'asymptotiquement, la seule source d'erreur soit le terme d'exploration pure.

Une fois trouvée une telle fonction, on sait que le regret est un O de la somme des probabilités ci-dessus, et on obtient une majoration du regret.

Dans le cas des bandits, la fonction $\varepsilon(t) = \frac{1}{t}$ étant solution, et puisque $\sum_{k=0}^{n} \frac{1}{k} \sim ln(n)$, on obtient le résultat classique du comportement en log du regret de l'algorithme $\varepsilon_n - greedy$.

Faisons l'hypothèse (forte) que l'ALS nous retourne un résultat exact, et que notre façon de faire calculer l'ALS n'introduit pas de biais — cette seconde hypothèse est raisonnable puisque l'on ne prend jamais le risque de s'éloigner du voisinage de la solution proposée par le premier ALS.

On a alors :

$$(\hat{U}, \hat{V}) = \arg\min_{U,V} \sum_{i,j\, connus} (M_{i,j} - UV^\top)^2 + \|U\| + \|V\|$$

Malheureusement, je n'ai pas pu aller plus loin de façon consistante dans la preuve, parce que même sous ces hypothèses, je n'arrive pas à majorer de façon *non triviale* le terme de droite de l'expression. En effet, pour cela il faudrait pouvoir faire des hypothèses d'indépendance des tirages, ce qui n'est pas valide dans le cas de notre algorithme, et sans ces hypothèses je ne vois pas de résultat que je pourrais utiliser pour majorer $\mathbb{P}(U_{u,j} V_{u,j}^\top > U_{u,j^*} V_{u,j^*}^\top)$.

Des éléments de réponse peuvent se trouver dans la preuve de la validité de l'algorithme LinOFUL (voir : [1]), mais je n'ai pas pu m'en inspirer pour ma preuve.

4 Résultats pratiques

4.1 Mise en œuvre.

J'ai vérifié la validité de mon algorithme sur deux types de données, des données artificielles générées aléatoirement et des données issues de statistiques réelles.

Données artificielles Pour générer des données aléatoires de sorte à ce que M soit de rang approximativement k, j'ai procédé de deux façons différentes :

- De la façon la plus propice au fonctionnement de l'algorithme : en générant aléatoirement selon une loi uniforme deux matrices U et V de dimensions respectives (n, k) et (m, k), puis en calculant $M = UV^\top$
- D'une façon plus proche de la réalité, en construisant selon des lois uniformes deux vecteurs $\mathbf{u}$ et $\mathbf{v}$ de tailles respective n et m à valeur dans $\{0, \ldots, k\}$, ainsi qu'une matrice $\mathbf{r}$ de taille (k, k), puis en construisant M de la façon suivante : $M_{i,j} = \mathbf{r}_{\mathbf{u}_i, \mathbf{v}_j}$

Les résultats pratiques dans un cas comme dans l'autre sont sensiblement les mêmes, et je ne commenterai pas plus sur la distinction entre les deux. Les courbes présentées basées sur de données artificielles utilisent la seconde façon de faire.

Données réelles. Je me suis servi des bases de données publiques MovieLens [2] : il s'agit de votes d'utilisateurs à propos de différents films. Les (rares) algorithmes existants prenant en compte le compromis exploration/exploitation étaient internes à l'équipe dans laquelle j'ai travaillé, et ne pouvaient traiter que la plus petite base (100 000 votes, 1000 utilisateurs et 1700 films), mon algorithme a pu traiter la seconde (1 million de votes, 6000 utilisateurs et 4000 films), mais la 3e semble pour l'instant hors d'atteinte.

Les résultats sur les premiers pourcentages sont les plus importants, puisqu'ensuite un calcul – coûteux mais unique – permet de recomposer avec une grande précision la matrice.

4.2 Résultats.

Les résultats en termes de temps de calcul sont difficiles à apprécier, puisqu'il existe à ce jour très peu d'algorithmes traitant de ce problème. Cependant, on peut donner une évaluation de sa complexité : si l'algorithme doit *deviner* N valeurs, que la matrice est carré de taille $n * m$, que l'algorithme d'ALS est effectué périodiquement, et que l'on suppose que le rang k donné à l'algorithme est fonction linéaire de n, on obtient une complexité en $O(Nnm)$.

Cependant si, au lieu de le faire périodiquement, on fait l'ALS complet de moins en moins souvent, par exemple en doublant la période à chaque fois, on peut réduire la complexité à $O(nm\,log(N) + Nmax(n, m))$

Ainsi sur la matrice d'un million de valeurs, le temps de calcul sur un ordinateur portable de puissance de calcul moyenne a été de trois jours, donc le calcul pour chaque utilisateur a été de l'ordre du quart de seconde, à l'exception des quelques cas où l'ALS complet était recalculé. Ceci signifie que cet algorithme, sur des ordinateurs plus puissants, peut effectivement être mis en application dans le cadre d'un système de recommandation en ligne de type client/serveur, où le serveur doit calculer la page à afficher.

Le premier graphique (fig. 6) montre le gain cumulé en fonction du temps sur des données artificielles, alors que le second (fig. 7) montre des résultats sur la base de données MovieLens pour les 600000 premières recommandations (sur 1 million, soit 60%).

Dans les deux cas tous les algorithmes se comportent mieux que celui qui recommande aléatoirement, mais on constate aussi que $\frac{1}{\sqrt{n}} - greedy$ se comporte à chaque fois mieux que $greedy - pur$. Enfin, dans le cas des données artificielles, l'écart et sensible et $\frac{1}{n} - greedy$ est déjà meilleur que $greedy - pur$, alors que ces deux courbes sont confondues sur les données réelles. Face aux résultats dans le cas des données réelles, il convient de se demander si l'approximation du faible rang est vérifiée pour cette base de données.

2. `http://grouplens.org/datasets/movielens/`

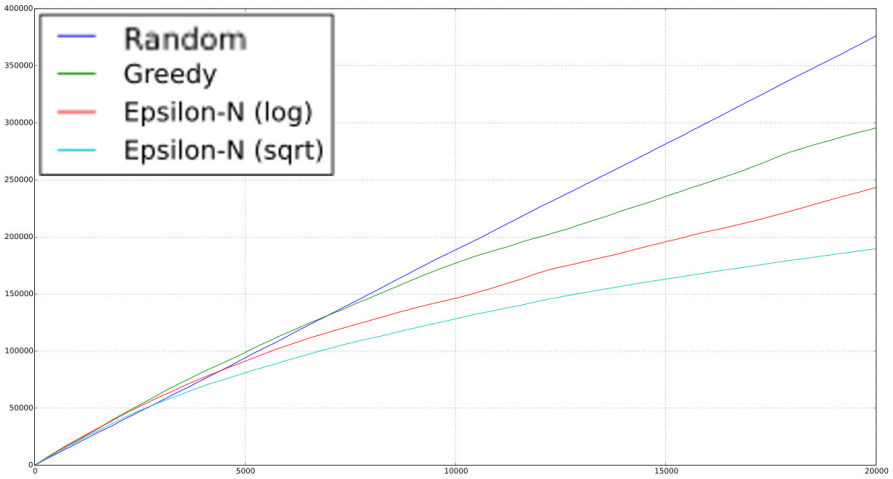

FIGURE 6 – Données artificielles — regret cumulé en fonction du nombre d'itérations : on est évidement meilleurs qu'un algorithme qui répond au hasard, et on est meilleurs que $greedy - pur$ qui est actuellement la norme. Par contre, $\frac{1}{n}$ semble moins bon que $\frac{1}{\sqrt{n}}$ ce qui nous pousse à dire que les hypothèses d'indépendance ne sont pas assez fortes pour mener la preuve comme Auer, et qu'il faudra se contenter d'une convergence en racine.

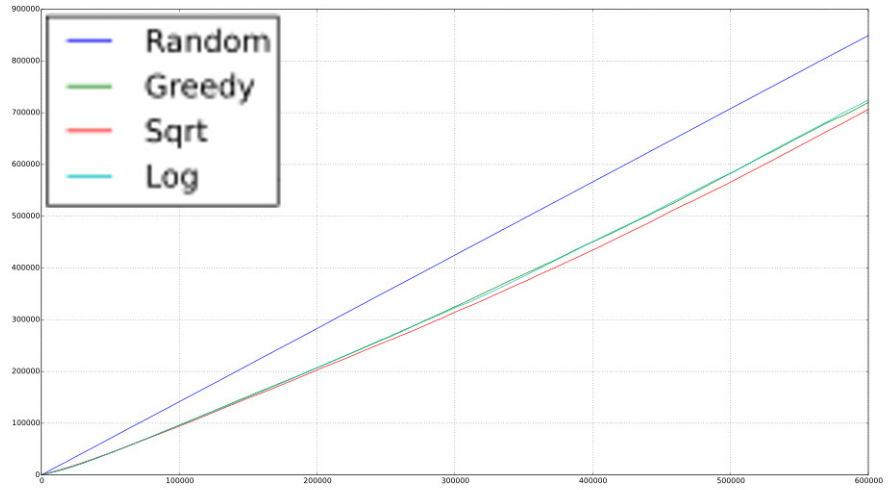

FIGURE 7 – MovieLens, 1 million d'entrées — regret cumulé en fonction du nombre d'itérations. Ce cas là est plus problématique : on est à nouveau, sans surprise, meilleurs que l'algorithme qui répond au hasard, cependant $greedy$ et $\frac{1}{n} - greedy$ se comportent exactement pareil, et $\frac{1}{\sqrt{n}} - greedy$ s'en sort mieux mais pas de façon très sensible.

4.3 Remarques.

La base de données MovieLens a souvent posé des problèmes aux gens qui travaillent en recommandation. Cependant, c'est la seule qui soit publique et gratuite pour tous, pour travailler sur les bases de Yahoo! ou de Netflix il aurait fallu les contacter pour leur demander l'autorisation, ce qui semblait dépasser un petit peu le cadre de mon stage.

On remarque que ma façon de faire a l'avantage de fonctionner sur des données formatées pour d'autres problèmes : il suffit de masquer artificiellement les valeurs, et de considérer que l'on part d'un problème neuf. Ceci m'a permis de travailler sans difficulté sur la base de données de MovieLens par exemple.

Le résultat sur les données artificielles est sans appel, mais à défaut de résultats théoriques sur la *bonne* fonction, il semble maladroit d'affirmer que c'est $n \to \frac{1}{\sqrt{n}}$.

5 Conclusion

La recommandation est un problème intrinsèquement séquentiel : on recommande un produit à partir des retours obtenus lors des recommandations précédentes. Ainsi la meilleur recommandation à un instant donné vise un bon retour dans l'immédiat, mais aussi de bons retours pour les recommandations futures.

Mon étude montre que les méthodes de recommandation actuelles ne répondent pas au problème de façon optimale, car ces méthodes se limitent à optimiser le retour immédiat. Cette observation est faite sur des données synthétiques et des données réelles, et est corroboré par les résultats théoriques de la communauté des bandits.

Cependant, la solution que je propose n'est qu'une première étape vers un algorithme optimal. Dans un premier temps, la fonction qui fixe le taux d'exploration d'ε_n-greedy à chaque pas de temps reste à déterminer.

Un autre point que je n'ai fait que soulever dans cette étude est la façon dont doit croître le rang k de la matrice avec le temps. J'ai temporairement mis une croissance linéaire mais de coefficient très faible, de sorte que sur le million de données de MovieLens on avait à la fin $k \approx 50$, et la façon de faire grandir cette valeur ne me semble pas évidente.

Je pense donc que les prochains points de recherche dans ce domaine seront liés à la transposition d'algorithmes beaucoup plus complexes dans ce domaine et de leurs preuves, ainsi qu'à une façon adaptée de faire varier k avec le temps et les observations successives.

Références

[1] Yasin Abbasi-yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2312–2320. Curran Associates, Inc., 2011.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[3] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *J. Mach. Learn. Res.*, 12:1655–1695, July 2011.

[4] E.J. Candes and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *Information Theory, IEEE Transactions on*, 56(5):2053–2080, May 2010.

[5] Deepayan Chakrabarti, Ravi Kumar, Filip Radlinski, and Eli Upfal. Mortal multi-armed bandits. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 273–280. Curran Associates, Inc., 2009.

[6] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670, New York, NY, USA, 2010. ACM.

[7] William H. Press. Bandit solutions provide unified ethical models for randomized clinical trials and comparative effectiveness research. *Proceedings of the National Academy of Sciences*, 106(52):22387–22392, 2009.

[8] Nicolas L. Roux, Mark Schmidt, and Francis R. Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2663–2671. Curran Associates, Inc., 2012.

[9] Long Tran-Thanh, Archie Chapman, Jose Enrique Munoz De Cote Flores Luna, Alex Rogers, and Nicholas R. Jennings. Epsilon?first policies for budget?limited multi-armed bandits. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1211–1216, April 2010. Event Dates: 11 - 15 July, 2010.

Annexes

A Preuve de $\varepsilon - greedy$ simple.

Quelques notations : les conventions sur les lettres chapeautées et sur les lettres avec une étoile sont les mêmes qu'en page 5. On note de plus :

— X_j la variable aléatoire du bras j et μ_j son espérance.
— $T_j(n)$ le nombre de fois où le bras j a été tiré jusqu'à l'instant n, et $T_j^R(n)$ le nombre de ces tirages qui proviennent du cas de tirage aléatoire uniforme de l'algorithme.
— Δ_j l'écart entre l'espérance du bras j et celle du bras optimal
— K le nombre de bras.

Posons

$$x_0 = \frac{1}{2K}\sum_{t=1}^{n}\varepsilon(t)$$

L'idée est d'établir une majoration de la probabilité de tirer un bras sous-optimal à chaque instant. Puisque le regret est la différence entre la récompense du bras tiré et l'espérance du bras optimal, la somme des $\Delta_j\mathbb{P}(mauvais\,bras)$ sera un majorant du regret et l'on aura un résultat asymptotique du comportement de notre algorithme.

Majorons la probabilité de tirer un bras sous-optimal j à l'instant n:

$$\mathbb{P}(I_n = j) = \frac{\varepsilon_n}{K} + \left(1 - \frac{\varepsilon_n}{K}\right)\mathbb{P}(j = \arg\max_i \hat{X}_{i,T_{(n-1)}}) \tag{2}$$

Soit, en oubliant tous les bras sauf celui qui nous intéresse et le bras optimal :

$$\mathbb{P}(I_n = j) \leqslant \frac{\varepsilon_n}{K} + \left(1 - \frac{\varepsilon_n}{K}\right)\mathbb{P}(\hat{X}_{j,T_{(n-1)}} \geqslant \hat{X}^*_{T^*(n-1)}) \tag{3}$$

Or puisque $\mu_j + \frac{\Delta_j}{2} = \mu^* - \frac{\Delta_j}{2}$:

$$\mathbb{P}(\hat{X}_{j,T_{(n)}} \geqslant \hat{X}^*_{T^*(n)}) \leqslant \mathbb{P}\left\{\hat{X}_{j,T_{(n)}} \geqslant \mu_j + \frac{\Delta_j}{2}\right\} + \mathbb{P}\left\{\hat{X}^*_{T^*(n)} \leqslant \mu^* - \frac{\Delta_j}{2}\right\} \tag{4}$$

On remarque alors que les deux membres de droite sont égaux. Il va nous suffire d'en établir une majoration pour obtenir le résultat qui nous intéresse.

Établissons-en une majoration : on peut commencer par séparer la probabilité selon les valeurs possibles de $T_j(n)$:

$$\mathbb{P}\left\{\hat{X}_{j,T_{(n)}} \geqslant \mu_j + \frac{\Delta_j}{2}\right\} = \sum_{t=1}^{n}\mathbb{P}\left\{T_j(n) = t \wedge \hat{X}_{j,t} \geqslant \mu_j + \frac{\Delta_j}{2}\right\} \tag{5}$$

Que l'on peut séparer en un produit de probabilité, puis en utilisant la borne de Chernoff-Hoeffding sur le second membre :

$$\mathbb{P}\left\{\hat{X}_{j,T_{(n)}} \geqslant \mu_j + \frac{\Delta_j}{2}\right\} \leqslant \sum_{t=1}^{n} e^{-\Delta_j^2 t/2}\mathbb{P}\left\{T_j(n) = t | \hat{X}_{j,t} \geqslant \mu_j + \frac{\Delta_j}{2}\right\} \tag{6}$$

Alors, en considérant une majoration sur la somme des exponentielles, en en ne considérant **que** les cas de tirages *au hasard* des bras, on obtient :

$$\mathbb{P}\left\{\hat{X}_{j,T_{(n)}} \geqslant \mu_j + \frac{\Delta_j}{2}\right\} \leqslant x_0\mathbb{P}\left\{T_j^R(n) \leqslant x_0\right\} + \frac{2}{\Delta_j^2}e^{-\Delta_j^2\lfloor x_0\rfloor/2} \tag{7}$$

En utilisant l'inégalité de Bernstein (voir [2]), on obtient :

$$\mathbb{P}\left\{T_j^R(n) \leqslant x_0\right\} \leqslant e^{-x_0/5} \tag{8}$$

Or la définition de x_0 nous donne en posant $n' = \frac{cK}{d^2} \leqslant n$:

$$x_0 = \frac{1}{2K}\sum_{t=1}^{n'}\varepsilon(t) + \frac{1}{2K}\sum_{t=n'+1}^{n}\varepsilon(t) \tag{9}$$

$$x_0 \geqslant \frac{n'}{2K} + \frac{c}{d^2} ln\left(\frac{n}{n'}\right) \tag{10}$$

D'où, par définition de n' :

$$x_0 \geqslant \frac{c}{d^2} ln\left(\frac{nd^2 e^{1/2}}{cK}\right) \tag{11}$$

En remontant à notre problème initial, on obtient :

$$\mathbb{P}(I_n = j) \leqslant \frac{\varepsilon(n)}{K} + 2x_0 e^{-x_0/5} + \frac{4}{\Delta_j^2} e^{-\Delta_j^2 \lfloor x_0 \rfloor/2} \tag{12}$$

Puis en injectant (11) dans ce résultat, et en prenant $\varepsilon(n) \sim \frac{1}{n}$, on remarque que le terme dominant est le $\frac{\varepsilon(n)}{K}$, et on a bien :

$$\mathbb{P}(I_n = j) \leqslant f(n) \text{ où } f(n) \sim \frac{1}{t} \tag{13}$$

D'ou il vient, par somme de ces quantités :

$$R_n \leqslant \sum_{i=1}^{n} f(i) \text{ où } f(i) \sim \frac{1}{i} \tag{14}$$

C'est à dire :

$$R_n \leqslant g(n) \text{ avec } g(n) \sim ln(n) \tag{15}$$

B Résultats pratiques des algorithmes classiques.

B.1 Recommandation

Vous trouverez ci-après deux graphiques, l'un pour exposer le problème de l'*over-fiting* dans le cas où l'on ne choisi pas judicieusement le rang des matrices à obtenir, et l'autre comme comparaison générale des différents algorithmes de recommandation.

L'abscisse représente l'erreur à minimiser, et l'ordonnée le rang attendu de la matrice.

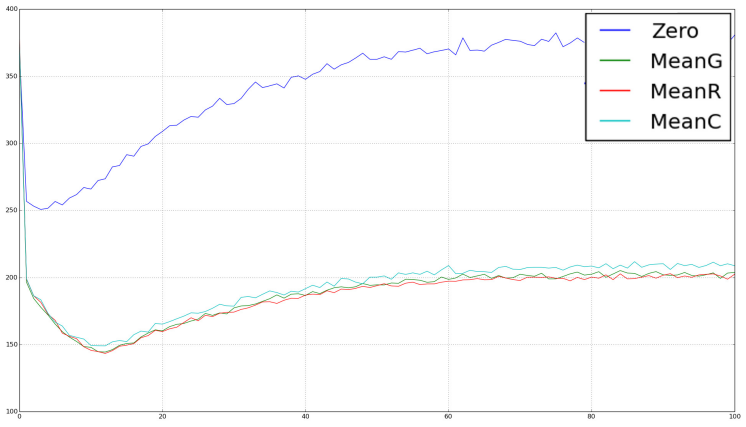

FIGURE 8 – Ce graphique illustre le problème d'*over-fiting* : si on augmentent trop le rang, on finit par garder du bruit comme information, et on perd à nouveau par rapport à la valeur optimale. À noter que sur ce graphique, le rang de la matrice à compléter est de 10, éventuellement un peu moins, ce qui se retrouve sur le graphique. La courbe "Zero" correspond au cas où l'on remplit la matrice avec des zéros avant la SVD, les autres correspondent aux cas où l'on remplace par la moyenne (générale/par ligne/par colonne)

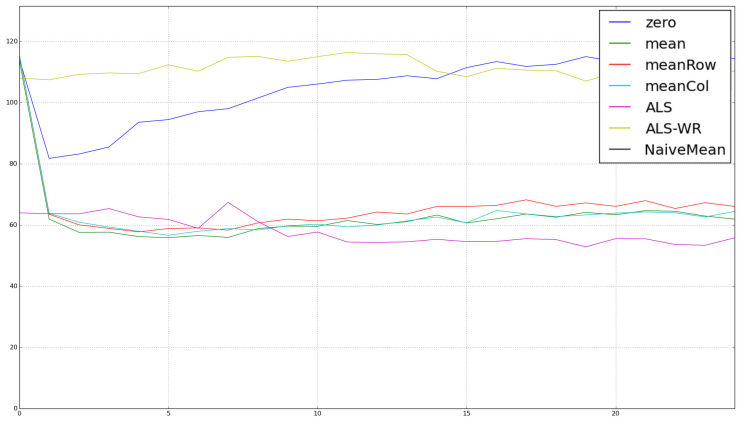

FIGURE 9 – Cette fois ci j'oppose ALS à SVD (ALS-WR est une autre façon d'appliquer une régularisation pour éviter l'over-fiting, mais qui fonctionne mal sur des données artificielles, c'est pourquoi le comportement n'est pas convaincant.). On voit qu'aux alentours du rang réel de la matrice, et ensuite, l'ALS donne de meilleurs résultats que la SVD.

B.2 Bandits

Je propose ici une comparaison entre les différents algorithmes connus pour les bandits, d'abord dans le cas le plus simple, puis dans le cas linéaire.

L'abscisse représente le regret, et l'ordonnée le temps : le but est de visualiser le comportement des différents algorithmes avec le temps, et de se convaincre de leur efficacité.

Dans chaque cas, l'algorithme qui tire au hasard correspond à la droite passant par l'origine, tangente à toutes les autres : le gain est donc très conséquent.

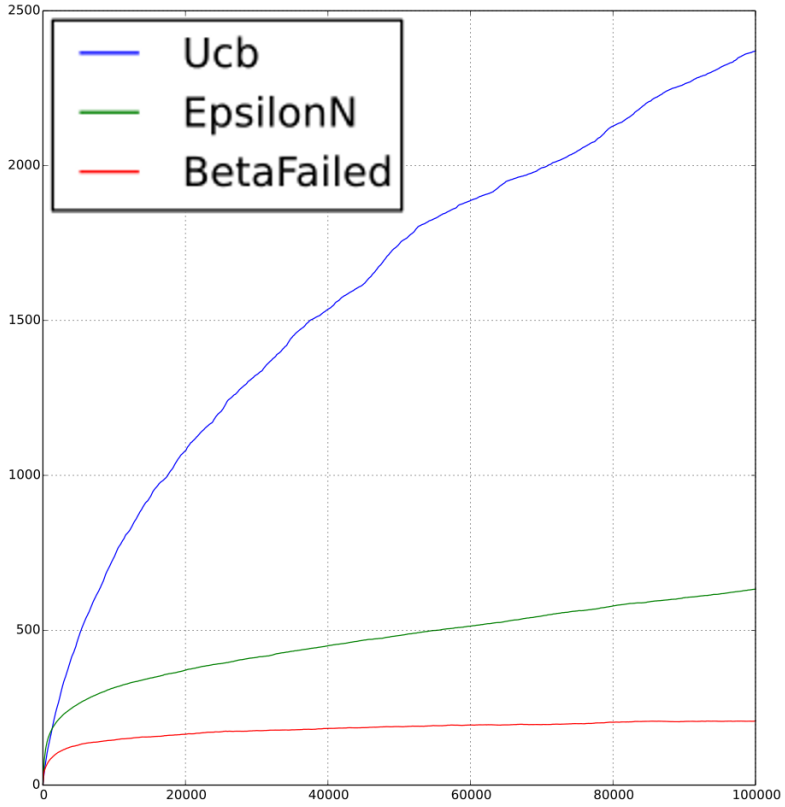

FIGURE 10 – Les différents algorithmes connus dans le domaine des bandits simples. Ce run illustre à la fois les avantages et les défauts : UCB et EpsilonN vont converger en $O(log(t))$, mais avec des constantes différentes, BetaFailed est une sorte de $greedy - pur$: lorsqu'il a trouvé le meilleur bras, il l'exploite et son regret est une constante. Il s'agit cependant — comme son nom l'indique — d'une erreur de ma part : il peut lui arriver de se tromper et donc d'avoir un comportement linéaire. Enfin, on remarque que EpsilonN semble admettre une droite de pente faible comme asymptote, ce qui nous permet de comprendre ce qui s'est passé pendant ce run : les deux meilleurs bras doivent être très proche, et il exploite le mauvais, ce dont il va se rendre compte asymptotiquement grâce à l'exploration, c'est un side-effect du fait que le run n'est **pas assez long** ...

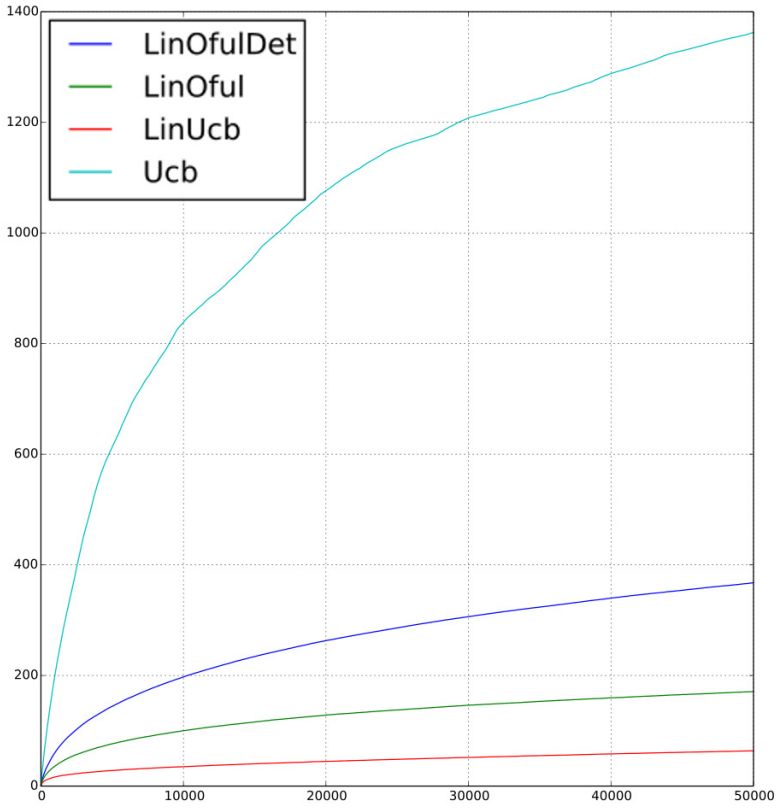

FIGURE 11 – Cette fois dans le domaine des bandits linéaires, où les informations de chaque bras nous informent sur les autres. Je n'ai pas beaucoup parlé de ce problème dans le présent rapport, mais notre problème de recommandation est plus proche de celui-ci que du cas *non-linéaire*, et l'on voit bien qu'il existe des algorithmes très efficaces adaptés au cas linéaire, quoiqu'ils soient tous en $O(log(t))$. Les prochains algorithmes de recommandation exploiteront peut-être cette caractéristique, cependant les preuves sont beaucoup plus difficiles à mener.