Algorithmes pour les systèmes de recommandation : un comparatif

Introduction

Définition : collaborative filtering
Position du problème : Attentes vis-à-vis des algorithmes (précision/temps de réponse/ajout rapide d'un rating/cold start...)
Efficacité peut varier selon la forme du jeu de données (sparsity...)
Objectif : Implémenter (en Matlab) et comparer différents algorithmes en terme de précision et de temps d'exécution sur différents jeux de données, mettre en relief les compromis à faire.

1 Position du problème

Notations ?

2 Les algorithmes utilisés

2.1 Algorithmes témoins

Algorithmes naïfs servant de point de comparaison :
– "Witness" : Estimer tous les ratings inconnus par la moyenne de tous les ratings possibles
– "PerUserAverage" : Estimer tous les ratings inconnus pour un utilisateur donné par la moyenne des ratings connus qu'il a attribués
– "BiasFromMean" : Estimer la note donnée par un utilisateur à un objet en fonction de sa moyenne, mais aussi des écarts entre les notes attribuées à l'objet et les moyennes des utilisateurs qui l'ont noté
– "UnbiasedWitness" : Retirer les biais comme dans le cours, estimer tous les ratings inconnus par 0, remettre les biais

2.2 Algorithmes Slope One

Prédicteur affine, en imposant une pente de 1. Pour tout couple d'objet, on définit une déviation de l'un par rapport à l'autre, un peu comme la mesure de similarité cosinus, mais en beaucoup plus simple (linéaire). La prédiction de la note attribuée à un objet s'obtient alors en ajoutant la moyenne de l'utilisateur et la moyenne des déviations de l'objet aux autres objets notés par l'utilisateur. D'où un prédicteur de forme $f(x) = x + b$.
Calculs simples. Possibilité de garder les déviations en mémoire : traitement d'une requête très rapide. De plus, ces déviations peuvent même être mises à jour sans tout recalculer quand on ajoute un rating.
À quel point est-ce moins précis que d'autres algos plus sophistiqués ?

Variante : donner plus de poids aux déviations des paires d'objets qui ont été notés tous deux à la fois par un grand nombre d'utilisateurs, car elles sont plus fiables.
Algorithmes tirés de [2].

2.3 Algorithmes par similarité cosinus

Implémentation des algorithmes vus en cours : on calcule les biais en résolvant le problème d'optimisation

$$\min_{b^i, b_j} \sum_{i,j \in \Omega} (r_{ij} - \overline{r} - b^i - b_j)$$

via la solution matricielle du cours, puis on calcule des scores de similarité entre utilisateurs ou entre objets en utilisant le cosinus de l'angle comme mesure de similarité entre deux vecteurs et on estime les ratings comme une moyenne pondérée par les scores de similarité.

2.4 SVD

Variations sur la question bonus du DM 2. L'algorithme du DM commence par estimer les ratings inconnus à 0 avant d'appliquer la décomposition en valeurs singulières. L'estimation à 0 permet de faire des calculs sur des matrices creuses quand on connaît peu de ratings, donc un temps de calcul plus court. Les variantes qui suivent opèrent différentes transformations sur la matrice à estimer avant d'appliquer l'algorithme du DM, le but étant que l'estimation à 0 soit aussi pertinente que possible.

2.4.1 ShiftSVD

On calcule le rating moyen $\overline{r}$, on applique l'algorithme du DM à $R - \overline{r}$ et on ajoute $\overline{r}$ au résultat.

2.4.2 UnbiasedSVD

Plutôt que de se contenter de retrancher le rating moyen partout, on calcule les biais en résolvant le problème d'optimisation vu en cours comme en section 2.3, on applique l'algorithme du DM à la matrice non biaisée puis on réintroduit les biais. On s'attend à un résultat plus précis, mais le temps de calcul des biais est non négligeable (il faut inverser une matrice).

2.5 Analyse en composantes principales : algorithme Eigentaste[1]

Idée : s'appuyer sur un petit sous-ensemble d'objets notés par tous les utilisateurs (*gauge set*) pour projeter un utilisateur sur un espace de petite dimension puis estimer ses notes à partir de celles de ses voisins au sens d'un algorithme de clustering. Eigentaste ne s'applique donc qu'aux jeux de données comprenant un tel gauge set, comme le jeu de données Jester.
Notons R la matrice $(n \times m)$ des ratings, G le gauge set, $k = |G|$.
On commence par normaliser les ratings des utilisateurs sur les objets du gauge set : la qualité objective d'un objet importe peu ici car on cherche à obtenir des informations sur les ressemblances entre utilisateurs par analyse de composantes principales.
Pour tout objet j, on calcule donc $\mu_j = \sum_{i=1}^{n} r_{ij}$ la note moyenne de cet objet, et

$$\sigma_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (r_{ij} - \mu_j)$$

sa variance.
On obtient alors la matrice A (de taille $k \times k$) des ratings normalisés sur le gauge set : $a_{ij} = \frac{r_{ij} - \mu_j}{\sigma_j}$.

On pose $C = \frac{1}{n-1} A^T A$ (matrice de corrélation de Pearson). Elle est symétrique définie positive, on note Λ la matrice diagonale contenant ses valeurs propres triées par ordre décroissant et E la matrice orthogonale de vecteurs propres associée. On a $EE^T = Id$, $C = E\Lambda E^T$.
On peut alors projeter les vecteurs correspondant aux utilisateurs sur un petit sous-espace (de dimension v) contenant l'essentiel de la variance en posant $X = AE_v^T$. En particulier, pour $v = k$, on a $C_X = \frac{1}{n-1} X^T X = ECE^T = \Lambda$ diagonale : les vecteurs sont décorrélés. On prend $v = 2$ dans la suite.

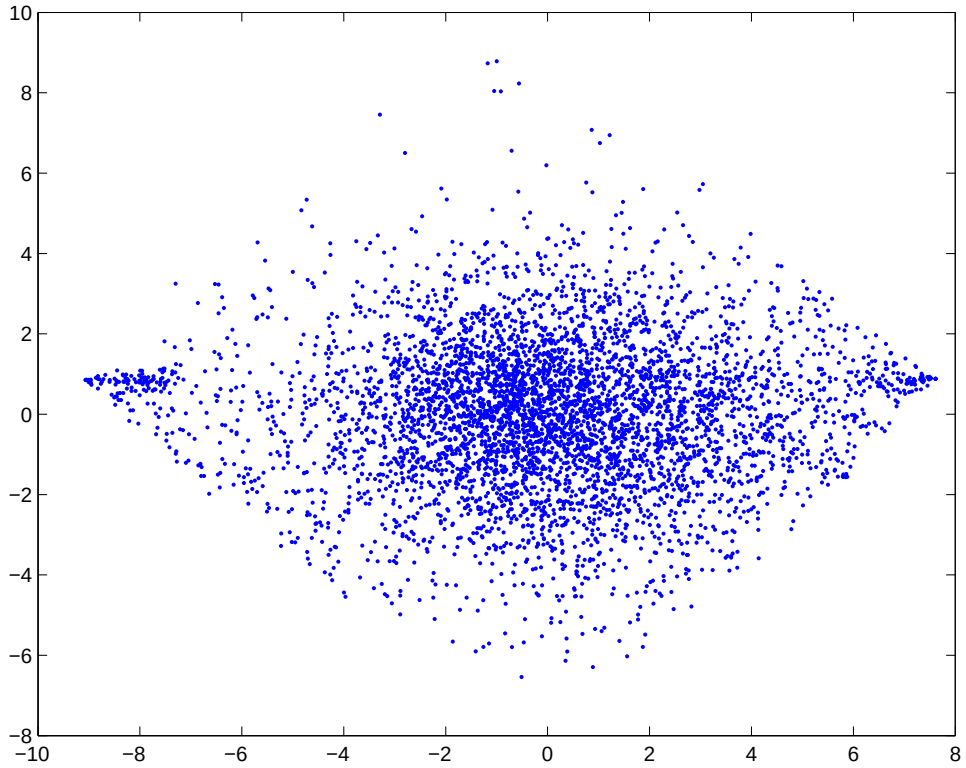

FIGURE 1 – Projection pour $v = 2$ sur un sous-ensemble du jeu de données Jester

On regroupe ensuite les utilisateurs proches par amas en utilisant un algorithme de clustering ([1] utilise un algorithme récursif rectangulaire, nous avons utilisé les fonctions Matlab *linkage* et *cluster* dans notre implémentation).
On estime ensuite un rating de l'utilisateur i sur l'objet j par la moyenne des notes pour j des utilisateurs se trouvant dans le même cluster que i.

Cet algorithme a l'inconvénient de ne s'appliquer qu'à certains jeux de données : pour ajouter un nouvel utilisateur, il faut d'abord lui faire noter l'intégralité du gauge set (typiquement une dizaine d'objets). De plus, on ne peut rien dire sur un utilisateur seul dans son amas : le démarrage à froid pose problème.
En revanche, ajouter un utilisateur qui a noté le gauge set est peu coûteux en termes de temps de calcul : on peut lui recommander rapidement des objets dès qu'on a choisi son cluster, l'idée étant de ne pas modifier immédiatement les projections des anciens utilisateurs mais plutôt de tout remettre à jour périodiquement.

Dans la suite, on se propose de comparer Eigentaste avec d'autres algorithmes de recommandation en termes de précision et de temps de calcul, sur le jeu de données Jester.

3 Observations expérimentales

3.1 Jeux de données

- Matrice de la question bonus du DM (pleine, on observe seulement une certaine fraction des ratings)
- Jeu de données Jester (ratings d'une centaine de blagues, dix blagues sont notées par toutes les utilisateurs) utilisé pour Eigentaste.
- Jeu de données plus grand (MovieLens) pour mesurer les difficultés liées aux temps d'exécutions dans des conditions plus réalistes ?

3.2 Mesures d'erreur

p=	0.2	0.4	0.6	0.8
Witness	0.0148	0.0112	0.0074	0.0037
UnbiasedWitness	0.0085	0.0062	0.0041	0.0020
PerUserAverage	0.0149	0.0111	0.0075	0.0037
BiasFromMean	0.0086	0.0062	0.0041	0.0021
PlainSVD	0.0940	0.0490	0.0221	0.0084
ShiftSVD	0.0106	0.0065	0.0037	0.0016
UnbiasedSVD	0.0087	0.0060	0.0036	0.0016
UserCosSim	0.0094	0.0083	0.0076	0.0070
ItemCosSim	0.0091	0.0077	0.0062	0.0049
SlopeOne	0.0086	0.0062	0.0042	0.0020
WeightedSlopeOne	0.0085	0.0062	0.0041	0.0020

Sur la matrice du DM, on obtient ce tableau d'erreurs MSE pour différentes valeurs de p :
Eigentaste et quelques autres ne sont pas encore implémentés. à venir aussi : une autre mesure d'erreur, et des mesures sur les autres jeux de données.
L'algorithme simple "BiasFromMean" obtient d'excellents résultats ! Les variantes de Slope One obtiennent aussi des résultats très comparables aux algorithmes SVD plus complexes à mettre en œuvre.

3.3 Temps d'exécution

Les algorithmes qui enlèvent et remettent les biais comme dans le cours s'exécutent beaucoup plus lentement : il faut inverser une matrice creuse, et on n'a pas réussi à vectoriser une partie du calcul de l'argmin pour les biais. Il faudra essayer l'expression plus naïve des biais et voir ce qu'on gagne en temps/perd en précision.
Il faut aussi tenir compte du fait que certains algos (Slope One oui, SVD non) calculent des choses "une fois pour toutes", et c'est rapide d'ajouter un utilisateur et/ou de faire une requête.

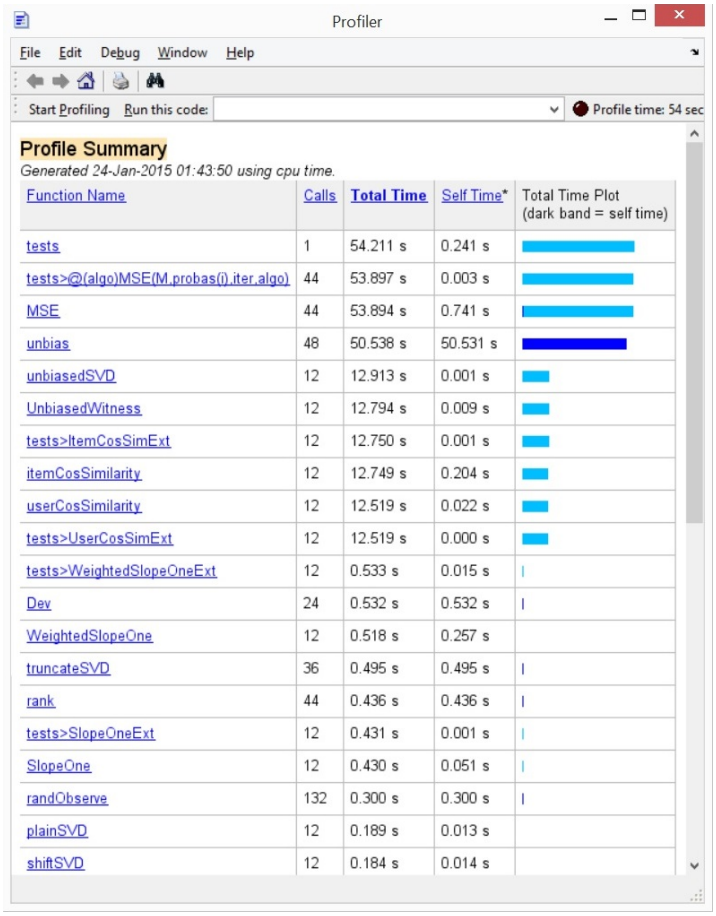

FIGURE 2 – Profiling fourni par Matlab

Références

[1] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste : A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2) :133–151, July 2001.

[2] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. *CoRR*, abs/cs/0702144, 2007.