FACTORISATION DE MATRICES AVEC DESCENTE DE GRADIENT DISTRIBUÉE

Mastère Spécialisé Data Science
Année 2017-2018

Ana Albutiu
Assitan Diarra
Flora Ziadi

Table des matières

Introduction 2
1. Présentation des données 3
2. Algorithme de factorisation de matrices 4
Algorithme DSGD - forme séquentielle 4
Algorithme DSGD Forme distribuée 6
3. Résultats 7
4. Extensions 8

Introduction

Avec le développement d'Internet, en particulier l'Internet mobile, l'information devient explosive. Plus de 90% des données dans le monde sont créées ces dernières années. Avec l'inflation de l'information, les gens accèdent à des informations utiles avec plus de difficultés, ce qui est appelé problème de surcharge d'information. Le principal moyen d'attaquer ce problème est par le biais des moteurs de recherche et des systèmes de recommandation.

En fonction des informations de l'utilisateur, comme l'historique des achats ou des films regardés et des informations personnelles de base, le point clé des systèmes de recommandation est de proposer aux utilisateurs les produits les plus susceptibles de répondre à leurs attentes.

Pour ce faire, la factorisation matricielle de rang inferieur a été considérée comme l'un des meilleurs modèles pour les systèmes de recommandation. Plusieurs chercheurs ont créé des algorithmes de factorisation de matrices à grande et petite échelle. Les algorithmes les plus utilisés sont ALS (Alternative Least Square) et SGD (Stochastic Gradiant Descent).

Dans ce projet, nous utiliserons l'algorithme SGD distribué (DSGD) que nous implémenterons sur Spark avec pySpark sur une base de données des préférences des utilisateurs pour des films. Nous présenterons donc dans une première partie les données utilisées puis dans une seconde partie nous expliquerons l'algorithme DSGD et enfin nous présenterons nos résultats et des extensions envisageables.

1. Présentation des données

Dans le cadre de ce projet, nous avons utilisé la base de données MovieLens, qui a été collectée par le groupe de recherche GroupLens à l'Université du Minnesota.

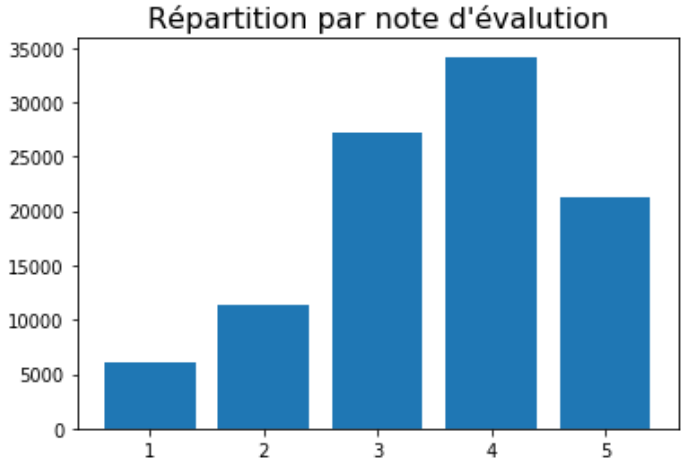
Les données ont été collectées sur le site web de MovieLens sur une période de sept mois entre le 19 septembre 1997 et le 22 avril 1998. Cet ensemble de données comprend 100 000 notations allant de 1 à 5 effectuées par 943 utilisateurs sur 1682 films. Chaque utilisateur a évalué au moins 20 films.

La base de données utilisée contient les informations :

- UserIDs : identifiant de l'utilisateur compris entre 1 et 943
- MovieIDs : identifiant du film compris entre 1 et 1682
- Rating : note donnée par un utilisateur pour un film, compris entre 1(faible) et 5(élevée)
- Timestamp : compteur de temps.

Pour la suite, nous utiliserons les 3 premières colonnes.

Sur le graphe suivant, nous présentons la répartition par note d'évaluation.



Les utilisateurs donnent les notes 3 et 4 pour évaluer les films dans 60% des cas.

2. Algorithme de factorisation de matrices

Etant donnée **V** – une matrice sparse de grandes dimensions $m \times z$ m utilisateurs aux n films et r – petit rang, le but est de trouver une matrice **W** de taille $m \times r$ représentant les utilisateurs) et une matrice **H** de taille $r \times n$ (représentant les films) de telle sorte que **V** ≈ **W H.**

La multiplication est utilisée pour prédire les notes manquantes de la matrice **V,** comme illustré ci-dessus :

	Harry Potter (2.24)	Saw (1.92)	Reine des neiges (1.18)
Ana (1.98)	?	**4** (3.8)	**2** (2.3)
Assitan (1.21)	**3** (2.7)	**2** (2.3)	?
Flora (2.3)	**5** (5.2)	?	**3** (2.7)

	Harry Potter (2.24)	Saw (1.92)	Reine des neiges (1.18)
Ana (1.98)	? (4.4)	**4** (3.8)	**2** (2.3)
Assitan (1.21)	**3** (2.7)	**2** (2.3)	? (1.4)
Flora (2.3)	**5** (5.2)	? (4.4)	**3** (2.7)

Pour les m=3 utilisatrices et n=3 films, décrits par un facteur ($r=1$), on a **les vraies notes** et on souhaite avoir les notes prédites pour les films non notés. Par exemple, est 'Saw' un film qui devrait être proposé à Flora ? (Oui, elle lui donnerait une bonne note)

La qualité de cette prédiction est caractérisée par une fonction de perte L à minimiser :

$$argmin_{W;H} L(\boldsymbol{V}, \boldsymbol{W}, \boldsymbol{H})$$

, qui doit pouvoir être décomposée en :

$$L = \sum_{i,j \in V} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$$

, pour un set d'observations $V \subseteq \{1,2,\dots,m\} \times \{1,2,\dots,n\}$ et une fonction locale de perte l, où $\boldsymbol{A}_{i*}$ et $\boldsymbol{A}_{*j}$ représentent respectivement la ligne i et la colonne j de la matrice A.

Il existe trois algorithmes principaux pour résoudre le problème de factorisation de matrice : alternate least square (ALS), coordinate gradient descent(CGD) et stochastic gradien descent (SGD). C'est sur ce dernier que notre travail va porter.

Algorithme DSGD - forme séquentielle

Dans sa forme la plus simple, l'algorithme SGD séquentiel est :

Algorithme 1 SGD pour Factorisation de Matrices

Ayant **V** – matrice de notes, **W** et **H** – matrices d'utilisateurs et de films initialisées avec des valeurs non nulles,
K – nombre d'itérations,
pour $k = 1, \dots, \boldsymbol{K}$ **faire** : sélectionner un point (i,j) de manière aléatoire

$$\boldsymbol{w}_i' = \boldsymbol{w}_i - \beta N \frac{\partial}{\partial \boldsymbol{W}_i} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$$

$$\boldsymbol{h}_i = \boldsymbol{h}_i - \beta N \frac{\partial}{\partial \boldsymbol{H}_i} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$$

$$\boldsymbol{w}_i = \boldsymbol{w}_i'$$

fin pour

Autrement dit, on calcule la note prédite $\boldsymbol{w}_{i*} \times \boldsymbol{h}_{*j}$, puis l'erreur de prédiction comme différence entre la note prédite et la note réelle : $\boldsymbol{e}_{ij} = v_{ij} - \boldsymbol{w}_{i*} \times \boldsymbol{h}_{*j}$ pour ensuite mettre à jour les matrices **W** et **H**.

Plusieurs fonctions de perte existent pour déterminer le terme de mise à jour des $\boldsymbol{w}_i$et $\boldsymbol{h}_i$,:

- Non Zero Squared Loss : $L_{NZSL} = \sum_{i,j \in V} (\boldsymbol{V}_{ij} - [\boldsymbol{WH}]_{ij})^2$
- Perte avec régularisation : $L_{L2} = L_{NZSL} + \lambda \|\boldsymbol{W}\|_F^2 + \|\boldsymbol{H}\|_F^2$

Cette dernière étant la forme choisie, on aura les termes de mise à jour correspondants :

$$\frac{\partial}{\partial \boldsymbol{W}_{ik}} L_{ij} = -2(\boldsymbol{V}_{ij} - [\boldsymbol{WH}]_{ij})\boldsymbol{H}_{kj} + 2\lambda \frac{\boldsymbol{W}_{ik}}{N_{i*}}$$

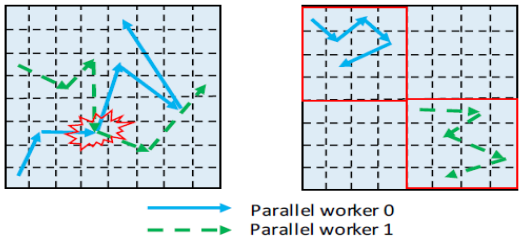
$$\frac{\partial}{\partial \boldsymbol{H}_{kj}} L_{ij} = -2(\boldsymbol{V}_{ij} - [\boldsymbol{WH}]_{ij})\boldsymbol{W}_{ik} + 2\lambda \frac{\boldsymbol{W}_{kj}}{N_{*j}}$$

, où :

- λ – paramètre de régularisation pour éviter le surapprentissage.
- β- taux d'apprentissage (learning rate), qui détermine le pas d'approche de la descente vers le minimum, suffisamment petit en sorte de ne pas rater le minimum et osciller autour de lui ; mais suffisamment grand pour descendre rapidement

SGD est un algorithme intrinsèquement séquentiel, puisque à chaque fois une observation est sélectionnée pour être mise à jour. Ayant un jeu de données avec N observations, une itération implique l'exécution de N mises à jour une après l'autre.

Pour accélérer le processus, il a été proposé de mettre à jour des blocs d'observations en parallèle et plusieurs façons de découper la matrice V en blocs existent :

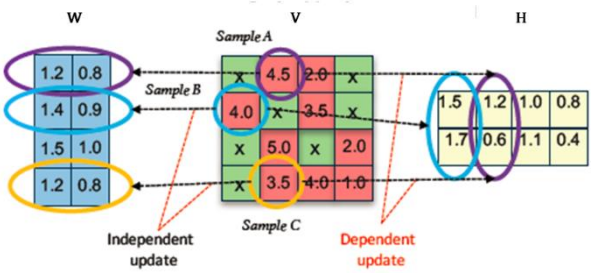


(a) Hogwild! *(b) Matrix Blocking*

Une approche nommée *Hogwild!* permet aux blocs de se partager la même ligne ou colonne, quand la matrice V est très sparse et donc le nombre de conflits serait beaucoup plus petit que le nombre d'observations disponibles, donc la probabilité d'un conflit serait petite.

L'approche qui sera implémentée par la suite dite *matrix blocking* consiste à diviser la matrice dans $d \times d$ sous-blocs indépendants qui ne se partagent ni la même ligne, ni la même colonne pour éviter entièrement les conflits. Idéalement, cette approche SGD peut être parallélisée sans perte de précision et garantissant que l'algorithme converge.

Illustration des observations dépendantes et indépendantes pour un cas avec deux facteurs (*r=2)* :



Algorithme DSGD Forme distribuée

L'algorithme distribué avec les blocs indépendants est :

Algorithme 2 DSGD pour Factorisation de Matrices

Ayant $\mathbf{V}$ – matrice de notes, $\mathbf{W}$ et $\mathbf{H}$ –matrices d'utilisateurs et de films initialisées avec des valeurs non nulles,
d – nombre de workers, $\mathbf{K}$ – nombre d'itérations,
$\mathbf{W} = \mathbf{W_0}$
$\mathbf{H} = \mathbf{H_0}$
découper $\mathbf{V}$ / $\mathbf{W}$ / $\mathbf{H}$ en blocs indépendants
générer d règles couvrant tous les blocs de V et chaque règle s'appliquant aux blocs indépendants
pour $k = 1, \dots, \boldsymbol{K}$ **faire** :
    sélectionner β pas d'apprentissage
    ordonner les règles de manière aléatoire ou séquentielle
    pour chaque règle $s = 1, \dots, d$ **faire** :
        choisir d blocs $\{V^{1j_1}, \dots, V^{dj_d}\}$ pour former un stratum
        pour chaque bloc $b = 1, \dots, d$ **faire** :
            exécuter en parallèle l'Algorithme1 SGD sur les points de V^{bj_b}
        fin pour
    fin pour
fin pour

En pratique, on construit les index des lignes et colonnes à inclure dans chaque sous-bloc à l'aide des règles, pour couvrir la matrice $\mathbf{V}$ en entier. Illustration pour un cas avec un facteur et une matrice V de taille 3 x 3.

Illustration d'une matrice découpée en 3 x 3 blocs et ses 6 règles pour des blocs indépendants :

(a)

v_{00}	v_{01}	v_{02}
v_{10}	**v_{11}**	v_{12}
v_{20}	v_{12}	**v_{22}**

(b)

v_{00}	v_{01}	**v_{02}**
v_{10}	v_{11}	v_{12}
v_{20}	**v_{12}**	v_{22}

(c)

v_{00}	**v_{01}**	v_{02}
v_{10}	v_{11}	**v_{12}**
v_{20}	v_{12}	v_{22}

(d)

v_{00}	**v_{01}**	v_{02}
v_{10}	v_{11}	v_{12}
v_{20}	v_{12}	**v_{22}**

(e)

v_{00}	v_{01}	v_{02}
v_{10}	v_{11}	**v_{12}**
v_{20}	**v_{12}**	v_{22}

(f)

v_{00}	v_{01}	**v_{02}**
v_{10}	**v_{11}**	v_{12}
v_{20}	v_{12}	v_{22}

Illustration d'une itération contenant 3 règles couvrant l'ensemble de la matrice $\mathbf{V}$:

v_{00}	v_{01}	v_{02}
v_{10}	v_{11}	v_{12}
v_{20}	v_{12}	v_{22}

=

w_0
w_1
w_2

h_0^T	h_1^T	h_2^T

worker1	**v_{00}**	v_{01}	v_{02}	$w_0h_0^T$	v_{00}	**v_{01}**	v_{02}	$w_0h_1^T$	v_{00}	v_{01}	**v_{02}**	$w_0h_2^T$
worker2	v_{10}	**v_{11}**	v_{12}	$w_0h_1^T$	v_{10}	v_{11}	**v_{12}**	$w_1h_2^T$	**v_{10}**	v_{11}	v_{12}	$w_1h_0^T$
worker3	v_{20}	v_{12}	**v_{22}**	$w_0h_2^T$	**v_{20}**	v_{12}	v_{22}	$w_2h_0^T$	v_{20}	**v_{12}**	v_{22}	$w_2h_1^T$

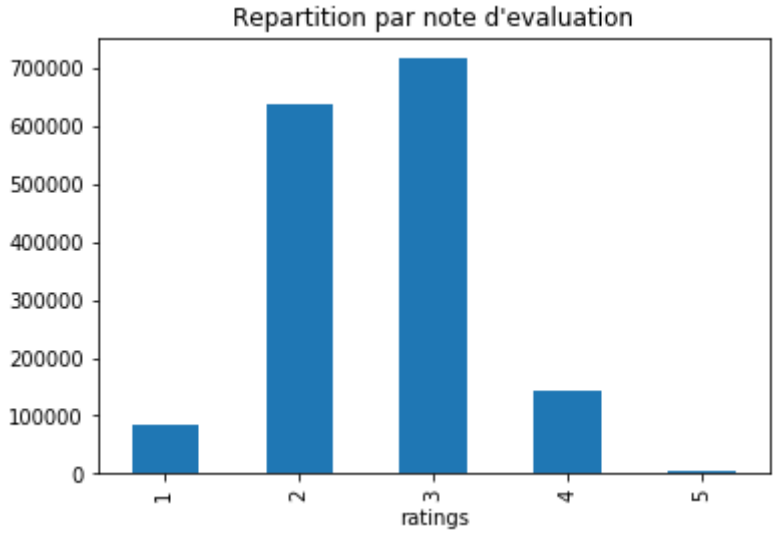
Plus le nombre d'itérations augmente et l'on s'approche du minimum, nous souhaitons que le taux d'apprentissage soit diminué pour ne pas manquer le minimum.

3. Résultats

Apres avoir implémenté l'algorithme, nous avons sélectionné les valeurs optimales des paramètres (nombre d'itérations avant d'atteindre un seuil de convergence fixé, nombre de facteurs, pas d'apprentissage, paramètre de régularisation) qui minimisent le MSE. Nous obtenons les valeurs suivantes :

	Valeur optimale	**MSE**
Nombre d'itérations	41	0.8802
Nombres de facteurs	10	0.9231
Pas d'apprentissage (β)	0.6	0.9214
Paramètre de régularisation (λ)	0.1	0.9188

En prenant compte des paramètres optimaux, nous obtenons un MSE de 0.9216. Pour ce modèle final, la répartition des notes d'évaluation prédites est :



Ratings	**Avant**	**Apres**	**%Avant**	**%Apres**
1	6110	**85695**	7%	4,75%
2	11370	506793	11%	26%
3	27145	635955	27%	32 %
4	34174	715319	34%	36%
5	21201	4935	21%	0,25%
Total	100000	1586126	100%	100%

Nous remarquons que nous prédisons mal la note 5 car sa prédiction est inférieure à la valeur de la base de données initiale et la note 3 représente la note la plus prédite.

4. Extensions

L'algorithme DSGD est une première forme de parallélisation de la descente stochastique de gradient pour la factorisation de matrices. Cette méthode n'est pas la plus performante, parce qu'elle requiert que toutes les blocs soient calculés avant de passer à une nouvelle itération, ce qui fait que le temps d'attente dépend de la vitesse du worker le plus lent. En plus, les blocs sont transmis aux workers à chaque itération.

Parmi les algorithmes plus avancés on peut citer :

- FPSGD (Fast Parallel Stochastich Gradient Descent) : adresse le problème de mémoire discontinue et utilise un environnement de mémoire distribuée
- FDSGD (Fast Distributed Stochastic Gradient Descent) : exécution dans un DCE (Distributed Computing Environment), évite le problème de verrou
- cuMF_SGD : accélération en utilisant la bande passante de mémoire et la connexion intra-nœuds des GPUs

Ressources

"Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent" - Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis

"Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent (Revision of IBM Technical Report RJ10481)" - Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis (version plus détaillée de l'article)

"A Fast Distributed Stochastic Gradient Descent Algorithm for Matrix Factorization" – Fanglin Li, in Wu, Liutong Xu, Chuan Shi, and Jing Shi

"CuMF_SGD: Fast and Scalable Matrix Factorization" – Xiaolong Xie, Wei Tan, Liana L. Fong, and Yun Liang

Support de cours de "Machine Learning with Big Datasets" de Carnegie Mellon University - William W. Cohen : http://www.cs.cmu.edu/~wcohen/10-605/sgd-for-mf.pdf (consulté le 5 février 2018)