Distributed Stochastic Gradient Descent pour la factorisation de matrice de faible rang

– ENSAE ParisTech – Elements Logiciels pour le traitement de données massives – Projet –

Mélanie FINAS : melanie.finas@ensae-paristech.fr
Peter MARTIGNY : peter.martigny@ensae-paristech.fr

5 Février 2017

Résumé

La factorisation de matrice de faible rang est un problème très important en machine learning et en particulier pour les systèmes de recommendations. Parmi ces systèmes, les plus aboutis sont basés sur l'obtention d'un modèle de quelques facteurs latents permettant d'expliquer en faible dimension les interactions entre clients et produits (ou utilisateurs et films). Dans ce projet, nous avons implémenté une version du Distributed Stochastic Gradient Descent (DSGD) pour la factorisation de matrice de faible rang dans le framework Spark Apache, présentée dans [NP16]. Nous avons ensuite étudié le temps de calcul de cet algorithme pour la factorisation avec différents nombres de facteurs latents.

1 Introduction aux methodes de factorisation de matrice

Avec le concours Netflix à 1M$ lancé en 2006, les systèmes de recommendations par factorisation de matrice à faible rang ont été largement popularisés (cf . [YZP08]). Le problème de factorisation de matrice de faible rang est le suivant : Soit V une matrice appartenant à $\mathbb{R}^{n*m}$. Il s'agira dans notre cas d'une matrice des ratings avec les lignes qui correspondent aux utilisateurs et les colonnes aux films. Ainsi chaque entrée $V_{(i,j)}$ est le rating du film j attribué par l'utilisateur i. Si n et m sont tous les deux très grands, trouver une factorisation de faible rang peut être très utile pour réduire la dimensionnalité et ainsi le temps de calcul. La méthode de factorisation à faible rang consiste à trouver deux matrices $W \in \mathbb{R}^{n*k}$ (elle correspond à la matrice des utilisateurs et chaque ligne correspond à un utilisateur) et $H \in \mathbb{R}^{k*m}$ (elle correspond à la matrice des films et chaque ligne correspond à un film) de manière à minimiser la fonction de perte entre WH et la matrice de départ V. La méthode la plus courante est de trouver W et H qui minimisent la norme de Frobenius de la différence entre nos deux matrices.

2 Présentation des données utilisées

2.1 Bases de données

Les données utilisées pour ce projet sont un sous-ensemble des données fournies dans le cadre du projet Netflix. Nous avons donc utilisé deux datasets : un dataset de training (training_ratings.txt) et un dataset de test (test_ratings.txt). Chaque ligne de ces fichiers représente le rating d'un film par un utilisateur. Les fichiers ont donc le format suivant : movie_id, user_id, rating. La base d'apprentissage contient 3 255 352 ratings pour 28 978 utilisateurs et 1821 films différents et la base de test contient 100 478 ratings pour 1701 films et 27 555 utilisateurs différents.

2.2 Environnement de code

Nous avons utilisé Spark pour faire la parallélisation et nous avons utilisé 4 processeurs. L'ensemble du code a été réalisé sous pyspark.

3 Distributed Stochastic Gradient Descent

On cherche à résoudre le problème suivant :

$$\min_{W,H} \frac{1}{2} \left\| V - WH^T \right\|_2^2 + \lambda(\|W\|_2^2 + \|H\|_2^2)$$

L'approche la plus classique pour résoudre le problème d'optimisation est de simplement itérer séquentiellement à partir de chaque élément non nul de notre matrice en utilisant un update SGD.

3.1 Algorithme

3.1.1 Stochastic Gradient Descent

Pour chaque point dans la base, on calcule une fonction de perte L2 par rapport à la ligne et la colonne correspondant dans la matrice des facteurs et on utilise un update SGD. Pour cela on calcule les dérivées de notre fonction de perte par rapport à $W_{i,:} et H_{:,j}$:

1.
$$\frac{\partial L(V_{(i,j)}, W_{i,:}, H_{:,j})}{\partial W_{i,:}} = -2 * (V_{i,j} - W_{i,:} * H_{:,j}) * H_{:,j} + 2 * \lambda * W_{i,:}^T$$

2.
$$\frac{\partial L(V_{(i,j)}, W_{i,:}, H_{:,j})}{\partial H_{:,j}} = -2 * (V_{i,j} - W_{i,:} * W_{i,:}^T) + 2 * \lambda * H_{:,j}$$

Ainsi à chaque itération, on met à jour nos matrice W et H ainsi :

1.
$$W_{i,:}^{new} = W_{i,:} - \epsilon_n * \frac{\partial L(V_{(i,j)}, W_{i,:}, H_{:,j}}{\partial W_{i,:}}$$

2.
$$H_{:,j}^{new} = H_{:,j} - \epsilon_n * \frac{\partial L(V_{(i,j)}, W_{i,:}, H_{:,j})}{\partial H_{:,j}}$$

3.1.2 Distributed Stochastic Gradient Descent

En général, distribuer SGD est compliqué parce que chaque étape dépend des autres. Cependant, dans notre cas SGD Distribué est basé sur le principe que deux matrices blocs i*j et i'*j' sont interchangeables si $i \cap i' = \emptyset$ et $j \cap j' = \emptyset$. Le but de l'algorithme de DSGD est donc de trouver des blocs interchangeables (strata) de manière à ce que chaque bloc à l'intérieur de ce stratum peut être mis à jour en parallèle en utilisant SGD. Ainsi, tant que nos paramètres n'ont pas convergé, on considère un nouveau stratum et on alloue les bloques aux différents processeurs. Voici figure 1 un exemple d'ensemble de strata possible pour une matrice découpé en des 3*3 matrices bloques tiré de [RGS11].

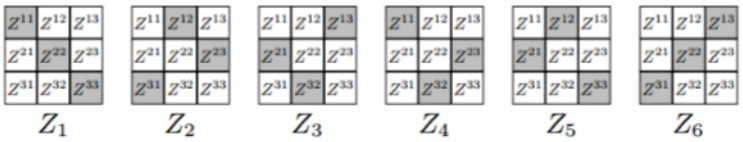

Figure 1 – Ensemble de stratum possible pour une matrice Z divisé en 3*3 bloques

Ainsi, notre fonction de perte peut être décomposée en une somme pondérée de fonctions de perte locales associées à chaque stratum. Le poids associé à chaque fonction de perte de chaque stratum sera égal au nombre de valeurs non nulles dans le stratum divisé par le nombre total de valeurs non nulles.

4 Présentation des résultats

Nous présentons ici les temps de calculs en fonction du nombre de workers et en fonction du nombre de facteurs latents, pour 1000 itérations du DSGD. Nous effectuons cette expérimentation sur la base entière et sur une base plus petite.

4.1 Base entière

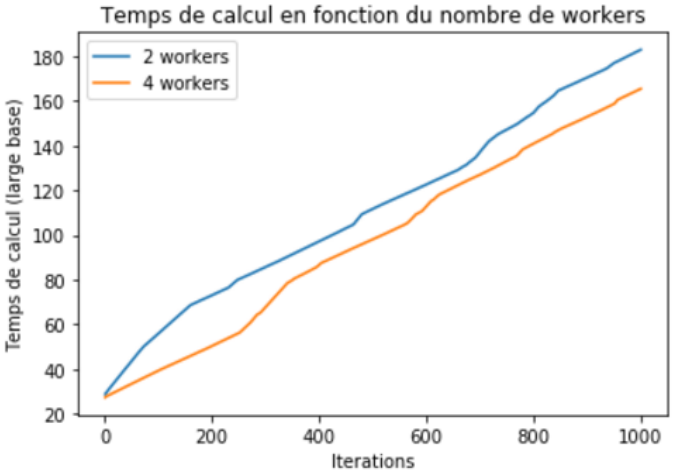

Figure 2 – Temps de calcul en fonction du nombre de workers pour la grande base

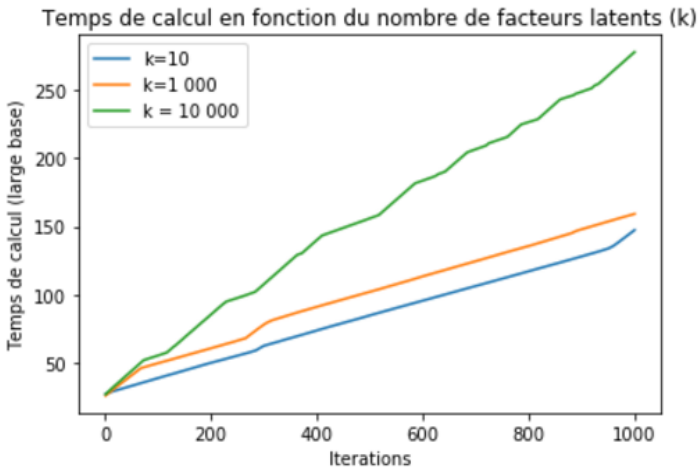

Figure 3 – Temps de calcul en fonction du nombre de facteurs latents pour la grande base

4.2 Base réduite

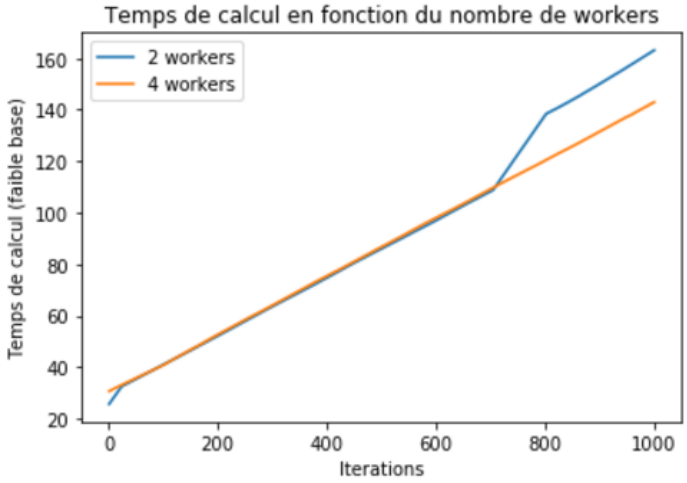

Figure 4 – Temps de calcul en fonction du nombre de workers pour la petite base

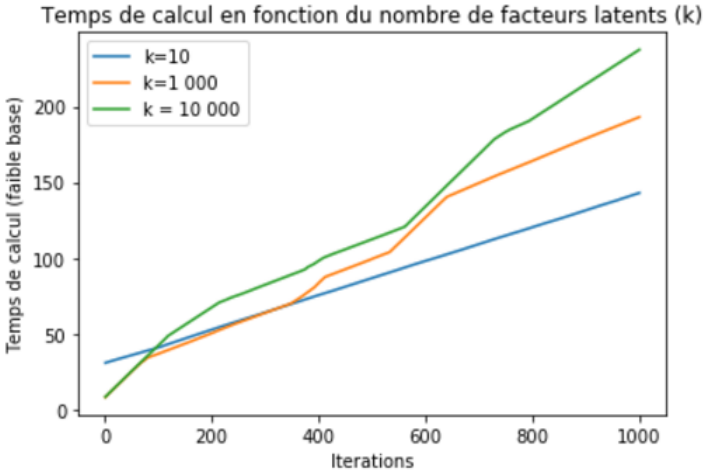

Figure 5 – Temps de calcul en fonction du nombre de facteurs latents pour la petite base

5 Conclusion

A l'aide de Spark, nous avons implémenté Stochastic Gradient Descent Distribué avec 4 processeurs utilisés pour la parallélisation et nous avons comparé les différents temps de calculs de notre algorithme en changeant les paramètres : nombre de workers et nombre de facteurs latents. Comme attendu, on remarque que l'augmentation du nombre de workers diminue le temps de calcul sur nos deux datasets et au contraire l'augmentation du nombre de facteurs latents augmente le temps de calcul. Il aurait également pu être intéressant de comparer le temps de calcul de notre algorithme avec l'algorithme ALS (Alternating Least Squares) pour la factorisation de matrice, implémenté dans la librairie MLlib de Spark.

References

[NP16] P.P. Tea-mangkornpan N. Parthasarathy. Low-rank matrix factorization using distributed sgd in spark. 2016.

[RGS11] P. J. Haas R. Gemulla, E. Nijkamp and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. 2011.

[YZP08] R. Schreiber Y. Zhou, D. Wilkinson and R. Pan. Large-scale parallel collaborative ltering for the netix prize. 2008.