Sara Mounir et Charlotte Augé 05.02.2017

ELEMENTS LOGICIELS POUR LE TRAITEMENT DE DONNEES MASSIVES

Projet DIMSUM

Dimension Independent Matrix Square using MapReduce

1. Introduction

Le projet est basé sur les travaux[1] de RezaBosaghZadehet Gunnar Carlsson qui développent une nouvelle manière de calculer le produit d'une matrice et de sa transposée. Nous considérons donc une matrice A de taille $n \times p$ dont les entrées ont valeur dans $[-1; 1]$ avec $n \gg p$. Nous cherchons à calculer la matrice $A^T A$.

L'algorithme « Dimension Independent Matrix Square UsingMapReduce » (DIMSUM) permet de calculer le carré d'une matrice avec une complexité ne dépendant pas de n, cela est donc très utile dans le cas où n est très grand.

Dans ce cas précis, MapReduce est l'outil idéal pour manipuler de grandes quantités de données en effectuant des calculs parallèles et en les distribuant dans un cluster de machines. Comme son nom l'indique, il consiste de deux étapes : map et reduce. L'étape map découpe le RDD, chaque partie va être traitée par une machine qui va appliquer la même fonction. L'étape reduce permet de renvoyer les données à la machine mère en regroupant les données par clé.

Nous allons vous présenter les différents algorithmes pour calculer le carré d'une matrice, puis les différents résultats obtenus et enfin les difficultés rencontrées durant ce projet.

2. Deux méthodes pour calculer $A^T A$

2.1. Méthode naïve

2.1.1. Théorie

La première méthode, dite naïve, consiste à calculer directement le produit matriciel de la transposée de la matrice A par la matrice A dans MapReduce. Notons B, la matrice résultat, nous avons donc

[1]Bosagh-Zadeh, Reza and Carlsson, Gunnar (2013), Dimension Independent Matrix Square using MapReduce, arXiv:1304.1467

$$b_{jk} = \sum_{i=0}^{n} a^T{}_{ji} a_{ik} = \sum_{i=0}^{n} a_{ij}\, a_{ik}$$

Ainsi, grâce à l'outils MapReduce, pour chaque ligne r_i, il nous suffit de calculer le produit $a_{ij}a_{ik}$ (étape map) et de sommer sur toutes les colonnes (étape réduce).Cette méthode est très simple, mais elle dépend de la valeur de n.

2.1.2. Implémentation en Spark

Pour l'implémentation des deux algorithmes suivants, nous avons choisi Spark avec le langage Python.

Algorithm 1 Mapper - méthode naïve

for chaque ligne r_i **do**
  Récupérer chaque paire (a_{ij}, a_{ik})
  Retourner $((j,k), a_{ij}a_{ik})$
end for

Algorithm 2 Reduce - méthode naïve

for chaque ligne $r_i : ((j,k), p_{jk})$ **do**
  Retourner $b_{jk} = \sum p_{jk}$
end for

2.2. Méthode DIMSUM

2.2.1. Théorie

La méthode DIMSUM, expliquée dans l'article, permet de réduire considérablement le nombre de calculs. Elle repose sur les deux algorithmes suivants :

Algorithm 1 Mapper - méthode DIMSUM

for chaque ligne r_i **do**
  Récupérer chaque paire (a_{ij}, a_{ik})

  Avec la propabilité min $\left(1, \frac{\gamma}{||c_j||\cdot||c_k||}\right)$, retourner $((j,k), a_{ij}a_{ik})$
end for

Algorithm 2 Reduce - méthode DIMSUM

for chaque ligne $r_i : ((j,k), p_{jk})$ **do**
  if $\frac{\gamma}{||c_j||\cdot||c_k||} > 1$ **then**
    $b_{jk} \rightarrow \frac{1}{||c_j||\cdot||c_k||} \sum p_{jk}$
  else
    $b_{jk} \rightarrow \frac{1}{\gamma} \sum p_{jk}$
  end if
end for

Notons B la matrice obtenue par les deux algorithmes DIMSUM, elle est de taille $p \times p$ et est appelée la matrice des similarités des cosinus des colonnes de A. Notons D, la matrice diagonale telle que $d_{ii} = \|c_i\|$. Une majeure partie de l'article1 propose la preuve de la convergence des matrices DBD et $A^T A$ donnée par le théorème suivant :

Théorème Soit A une matrice $m \times n$ avec $m > n$,
Soient$\gamma = \Omega\left(\frac{n}{\epsilon^2}\right)$ et Dest une matrice diagonale telle que $d_{ii} = \|c_i\|$,
Alors, la matrice$B$ obtenue par les deux algorithmes DIMSUM vérifie

$$\frac{\|DBD - A^T A\|_2}{\|A^T A\|_2} < \epsilon$$

Ainsi, pour comparer les résultats des deux méthodes, nous calculerons le ratio précédent afin de regarder la performance de nos algorithmes.

2.2.2. Implémentation en Spark

Les normes de chaque colonne ont été calculées à l'aide de la méthode MapReduce. L'étape Map a ainsi permis de calculé le carré de chaque valeur, l'étape Reduce quant à elle a additionné les valeurs par colonne.

Le γ que nous avons utilisé est $2\log(p)/s$ avec s=1, nous avons essayé plusieurs valeurs qui ont fait augmenter l'erreur, nous avons donc gardé cette valeur.

3. Résultats

Comme expliqué au paragraphe 2.2.1, afin de comparer les deux méthodes, nous avons comparé les deux matrices suivantes :DBD et $A^T A$. Nous avons donc calculé la moyenne des erreurs quadratiques. Nous avons testé la performance sur une matrice de taille 100*10 et nous obtenons une différence de l'ordre de 1'232% entre les deux méthodes. Nous nous rendons bien compte que l'erreur est très élevée, cependant nous n'avons pas réussi à optimiser le γ.

Nous avons aussi comparé les temps de calculs, nous remarquons que la méthode DIMSUM met beaucoup plus de temps que la méthode naïve. Cependant nous pensons que cela est du au fait qu'il s'agit de notre premier projet en spark, et que par conséquent nous n'avons pas su optimiser convenablement nos fonctions.

Taille matrice		Temps de calcul (s)	
m	n	Méthode naïve	Méthode DIMSUM
10^2	10	0.82	2,33
10^3	10^2	20,00	1115,81
10^4	10^2	170,08	>1h

4. Conclusion

Un peu laborieux, mais pour deux personnes qui ont commencé à coder en python en Septembre, nous sommes assez contentes du rendu. Nous aurions aimé terminer le programme Affinity Propagation, mais nous avons choisi de le continuer en dehors de ce cours, en dehors de toute pression de rendu.

Nous avions initialement choisi le projet « Affinity Propagation avec MapReduce ». Comme le montre le code Annexe1_AffinityPropagation.ipynb, sans passer par Spark, nous avons réussi à créer une fonction qui nous permettait de déterminer les différents centres du jeu de données, nous l'avons comparée à la fonction AffinityPropagation desklearn, les résultats étaient sensiblement identiques.

Notre problème est survenu lorsque nous avons voulu implémenter cette fonction sous MapReduce et notamment pour calculer la matrice de similarité. Afin de la calculer, nous avons pensé à recopier pour chaque ligne du RDD toutes les autres lignes du RDD. Cette méthode est très coûteuse car elle engendre donc des RDD de taille nb_sample*(nb_features*nb_sample). Nous avons passé énormément de temps à créer ce code sous Spark, en vain. Nous nous sommes donc rendues à l'évidence et nous avons donc choisi de changer de projet quelques jours avant la date du rendu afin de présenter quelque chose de correct.

Nous nous permettons cependant de vous joindre le code Annexe1_AffinityPropagation.ipynb afin de vous montrer la fonction python AffinityPropagation qui nous permettait de calculer les centres du jeu de données.