Ensae ParisTech

Eléments logiciels pour le traitement des données massives

Map/Reduce Affinity Propagation Clustering Algorithm

Yann Tekam
Arnaud Pagniez

Map/Reduce Affinity Propagation Clustering Algorithm

TEKAM Yann, PAGNIEZ Arnaud

5 février 2017

1 Introduction

Dans ce projet, nous nous sommes intéressés à l'algorithme développé par *Chih Hung*, *Chun-Yen Chu*, et *Leh Wu* de la National Taiwan University of Science and Technology et *Cheng-Yuan Tang* de la Huafan University. Cet algorithme est présenté dans l'article **"Map/Reduce Affinity Propagation Clustering Algorithm"** paru dans le journal *International Journal of Electronics and Electrical Engineering* paru en Août 2015.

Cet algorithme présente une manière de paralléliser l'exécution de l'algorithme dit **"Affinity Propagation"** développé par *Brendan J. Frey* et *Delbert Dueck* à l'université de Toronto en 2008.

Cet algorithme est une technique de clustering basée sur la théorie des graphes et le passage d'informations entre différents points du graphes.
Les principaux avantages de cet algorithme sont :

1. Applicable à toute base de données organisée en ligne et ne comportant que des variables numériques (peut s'etendre aux variables catégorielles).
2. Ne nécessite pas de spécifier le nombre de clusters désiré en amont de l'exécution.
3. Algorithme relativement intuitif

Le principal défaut de cet algorithme est sa complexité en $O(N^2)$. C'est ce défaut qui a motivé les trois créateurs de l'algorithme d'Affinity Propagation parallélisé. En effet, si ce problème de complexité quadratique peut être résolue par une approche Map/Reduce alors cette technique de clustering devient applicable à toute base de données.

2 Présentation des données choisies

Pour exécuter cet algorithme nous avons créés une base de données à partir de données **Kaggle** concernant les matchs de football et caractéristiques de joueurs tirées du jeu FIFA.

Ces données sont disponibles à l'adresse suivante : *www.kaggle.com/hugomathien/soccer/kernels*

Cette base de données contient de nombreuses données concernant les matchs de football européen durant les dernières années et contient notamment pour un certain nombre de joueurs (11060) leurs statistiques tirées du jeu vidéo FIFA ainsi que d'autres caractéristiques (droitier, gaucher, date de naissance).

La base de données fournie par Kaggle est au format sqlite. Nous avons donc eus recours au logiciel **DB Browser for SQLite** pour exploiter cette base de données et en exporter les tables **Player** et **Player-Attributes** au format csv.

Nous avons ensuite utilisés ces données pour notre travail sur l'algorithme Affinity Propagation.

3 Premiere implémentation

Comme exposé dans le notebook **ELTDM-PAGNIEZ-TEKAM.ipynb** disponible sur github, nous avons commencés par créer notre base de données à partir des fichiers csv avant d'exécuter l'algorithme d'Affinity Propagation simple sur une partie de notre base de données et de visualiser les clusters obtenus par cette approche. L'étude a été réalisée à l'aide de la bibliothèque *scikit-learn*.

Tout les résultats sont exposés et expliqués dans le notebook **ELTDM-PAGNIEZ-TEKAM.ipynb**.

On obtient la figure suivante pour 100 points :

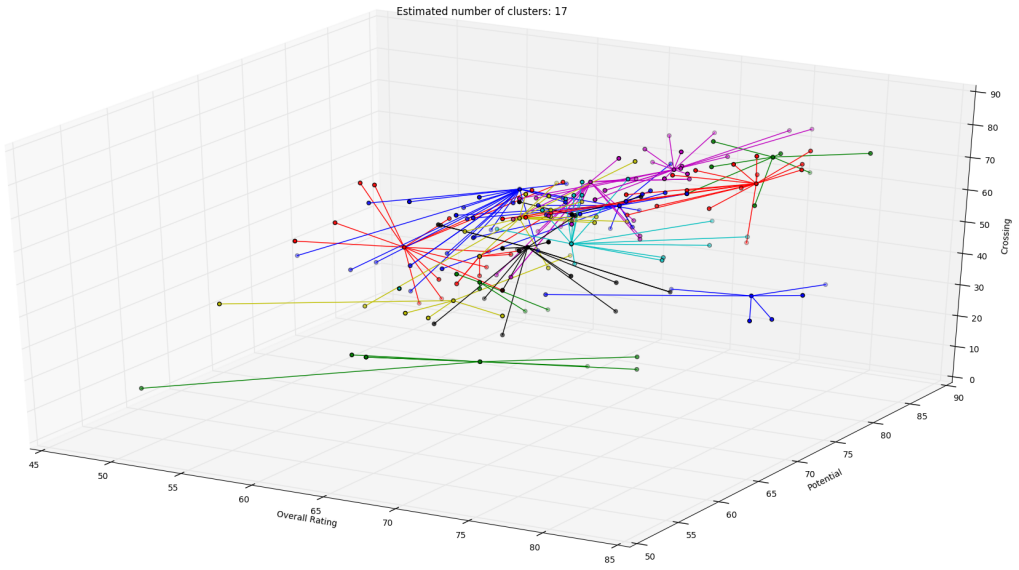

FIGURE *1 – 3D Plot of Clustered points*

NB : Les résultats de ce notebook ne sont pas éxecutables dans pyspark qui ne comporte pas les mêmes bibliothèques qu'Anaconda

4 Implémentation dans l'environnement PySpark

Après avoir réussi à installer Spark, nous avons décidés de coder dans l'environnment PySpark une version parallélélisée de l'algorithme Affinity Propagation. Le fait d'avoir installé Spark en local ne nous permettait néanmoins pas de mener l'opération au bout. Nous nous sommes donc efforcés de coder l'algorithme de telle manière à ce qu'il soit exécutable sur Spark relativement facilement.

Tout nos résultats sont disponibles dans le notebook **Spark-ELTDM-PAGNIEZ-TEKAM.ipynb** sur github (ce notebook doit être éxecuté dans un environnement PySpark).

Le code écrit détaille plusieurs tests de l'environnement spark avant de recoder l'algorithme d'Affinity Propagation et toutes les fonctions utiles pour les étapes de Merge, avant de présenter un exemple de Map/ Reduce sur de petits jeux de données. Ce notebook présente donc le code que nous enverrions à tous les noeuds du cluster si nous le pouvions, avant de présenter un exemple du processus de merge des résultats de deux éléments différents du clusters notamment grâce aux tables de hachage que nous avons voulus coder sous forme de dictionnaires.

Après avoir réalisé une première version fonctionelle du code dans le notebook **Spark-ELTDM-PAGNIEZ-TEKAM.ipynb** nous avons voulu améliorer l'implémentation obtenue. En effet, les temps de calculs nous apparaissaient longs et nous voulions produire un code qui utilisait plus les ressources de Spark (RDD, Vectors, etc). Cette version différente est disponible dans le notebook **Affinity-propagation-Spark.ipynb**. Les deux versions sont implémentables dans le contexte de Spark.

5 Algorithme de parallélisation

L'idée principale de cet algorithme de Map/Reduce appliqué à l'algorithme d'Affinity Propagation est la suivante :

1. On part d'une base de données de N lignes telle que N^2 soit trop grand pour permettre un calcul en une fois
2. On découpe cette base de données en K datasets qu'on envoie sur K machines différentes (toutes utilisant Spark dans notre cas)
3. On exécute le même code sur chacune des machines afin d'appliquer l'algorithme d'Affinity Propagation à chacun des K datasets
4. On récupère les tables de hachage de chaque ordinateur
5. On analyse les résultats et on fusionne les clusters trop proches
6. On analyse les clusters obtenus

Dans notre travail, les fonctions *affinity-propagation* et *local-propagation* permettent d'exécuter l'algorithme sur un dataset précis et de créer sa table de hachage à partir des résultats de l'algorithme.

Finalement la fonction *merge* permet de prendre deux datasets et tables de hachage et de reconstituer la base de données initiale tout en ayant maintenant une table de hachage complète pour définir les clusters des données.

Ainsi, les fonctions *affinity-propagation* et *local-propagation* seraient les fonctions envoyées sur chacun des ordinateurs la où la fonction *merge* est la fonction présente uniquement sur la machine finale.

6 Conclusion

Dans ce projet, nous nous sommes efforcés de mettre en place la démarche décrite dans l'article **"Map/Reduce Affinity Propagation Clustering Algorithm"** afin de paralléliser l'exécution de l'algorithme d'Affinity Propagation.

Notre travail s'est décomposé en plusieurs parties :

1. Création d'une base de données utilisable
2. Premier test de l'algorithme (via scikit-learn)
3. Passage à l'environnement Spark
4. Implémentation dans l'environnement Spark en simulant du calcul distribué
5. Amélioration du code

Notre démarche a donc été progressive et nous avons voulus nous rapprocher le plus possible d'un environnement dans lequel nous étions vraiment en train de faire du calcul distribué. C'est pour cela que nous avons cherchés à recoder les fonctions nommées précedemment dans un environnement PySpark afin qu'elles soient tout de suite applicables à un vrai cluster.

Au bilan, nous avons produit un code exécutable sur un cluster réel (après certaines modifications) et nous nous sommes approchés de la démarche présentée dans l'article afin de paralléliser l'étape de compléxité $O(N^2)$ dans l'algorithme d'Affinity Propagation.