HIGH PERFORMANCE COMPUTING
MATRIX FACTORIZATION FOR RECOMMENDER SYSTEMS

MAHA ELBAYAD

June 10, 2015

1- INTRODUCTION

This project aims to build a recommender sysetm based on parallel matrix factorization. The data used is the one provided by Netflix for their famous challenge (link).

1.1 THE DATA

The Data is over 100 million ratings from 480 000 randomly-chosen users of 18 000 movie titles.
It can be downloaded from

2- THE ALGORITHM

Let us consider our Rating matrix:

$$\mathbf{R} = \{r_{ij}\}_{\substack{1 \leqslant i \leqslant n_u \\ 1 \leqslant j \leqslant n_m}}$$

Where n_u is the total number of users and n_m is the total number of movies.
Some of the matrix $\mathbf{R}$ values are missing, and the objective is to estimate them based on the known ones.
To reduce the problem dimensionality, we map each of the users and the movies into a new space of dimension n_f, hence we build two matrices: (1): The users matrix $\mathbf{U} = [u_i]$, $\forall i \in [\![1, n_u]\!]\ u_i \in \mathbb{R}^{n_f}$.
(2): The movies matrix $\mathbf{M} = [m_i]$, $\forall j \in [\![1, n_m]\!]\ m_j \in \mathbb{R}^{n_f}$.
Our prediction of the user i rating of movie j would be the dot product $(u_i|m_j)$.
To build the matrices $\mathbf{U}$ and $\mathbf{M}$ we minimze a loss function J on the known ratings.

$$J(U, M) = \frac{1}{|K|} \sum_{(i,j) \in K} (r_{ij} - (u_i|m_j))^2$$

Where K is the subset of known values (i, j).
To find $(\mathbf{U}^*, \mathbf{M}^*) = \arg\min (U, M) J(U, M)$ we use the Alternative Least Squares method (ALS):

a) Initialize $\mathbf{M}$ by assigning the average rating for a movie to the first compenent of the corresponding column m_j, the rest are random small values.

b) While fixing $\mathbf{M}$, solve $\mathbf{U}$ by minizing $J(U, M)$.

c) While fixing $\mathbf{U}$ this time, we solve $\mathbf{M}$ by minimizing $J(U, M)$.

d) We loop over the 2 previous steps until a stopping criterion is met.

2.1 REGULARIZATION

To avoid overfitting the training set and performing poorly on the unseen data, we add a regularizaion parameter to the cost function J as follows:

$$J(U, M) \propto \sum_{(i,j)\in K} (r_{ij} - (u_i|m_j))^2 + \lambda\Big(\sum_i n_{u_i}\|u_i\|^2 + \sum_i n_{m_i}\|m_i\|^2\Big)$$

Where n_{u_i}, n_{m_i} are the total ratings for user i and movie i respectively.

2.2 OPTIMIZATION PROBLEMS

2.2.1 *Solving the Users matrix (step b):*

$$\begin{aligned}\forall (k,i) \in [\![1, n_f]\!] \times [\![1, n_u]\!],\ \frac{\partial J(\mathbf{U}, \mathbf{M})}{\partial u_{ki}} = 0 &\Rightarrow \forall (k,i),\ \sum_{j\in K_i} ((u_i|m_j) - r_{ij}) m_{kj} + \lambda n_{u_i} u_{ki} \\ &\Rightarrow \forall i,\ (M_{K_i} \times {}^t M_{K_i} + \lambda n_{u_i} \mathbb{I}_{n_f}) u_i = M_{K_i} \times {}^t R(i, K_i) \\ &\Rightarrow \forall i,\ u_i = (M_{K_i} \times {}^t M_{K_i} + \lambda n_{u_i} \mathbb{I}_{n_f})^{-1} \times M_{K_i} \times {}^t R(i, K_i).\end{aligned}$$

Where:

- K_i the subset of known rating from user i.
- M_{K_i} is submatrix of M with only the columns $j \in K_i$.
- $R(i, K_i)$ the rating row of user i (of length $|K_i|$)
- $\mathbb{I}_{n_f}$ is an identity matrix of size (n_f, n_f).

2.2.2 *Solving the Movies matrix (step c):*

Similarly

$$\forall j \in [\![1, n_m]\!],\ m_j = (U_{K_j} \times {}^t U_{K_j} + \lambda n_{m_j} \mathbb{I}_{n_f})^{-1} \times U_{K_j} \times {}^t R(K_j, j).$$

$R(K_j, j)$ is the rating column of movie j.

2.3 PARALLELIZATION

In a parallelized version of ALS, each processor (p) will handle a chunk of users and a chunk of movies. As seen in 2.2.1, we only need the user i's ratings to update u_i, but we need to have the whole matrix $\mathbf{M}$ as $\cup_{i\in p}\{j \in K_i\}$ could possibly cover all the database movies. Similarly the processor responsible for updating m_j will need the movie's rating and the whole matrix $\mathbf{U}$.

2.4 DATA PREPROCESSING

Since we have collected data on 4.8×10^5 users and 1.7×10^4 movies with only 10^6 ratings, the matrix $\mathbf{R}$ is sparse and loading the whole 8.16×10^9 potential values would take a tremendous time. Hence the need to choose a more appropriate data structure.
We define a struct "Node" holding the movie and the user IDs plus the rating. The matrix $\mathbf{R}$ would be

replaced by an array of "Nodes". We'll need two arrays, one sorted by users and a second one sorted by movies for efficient memeory access. Those two arrays would be assessed with 2 index arrays to tell the processor where in the large array to look for its assigned values.
The 2 built arrays would then be writing into binary files for the processors to read Node objects directly. Each of the processors will contribute in building the movies metadata consistig of "Ratings count" and "Ratings average" per movie. The rating average of each movie would serve to initialize the matirx M.
Besides, the users being chosen randombly from the netflix dataset we need to map them to $[\![1, \text{Number of users}]\!]$ and save the mapping for the prediction phase. This shuffled mapping aims to recalibrate the ratings distribution among the proprocessors, since the first users in the dataset tend to have far more ratings than the following ones.
While preparing the binary files I excluded some ratings and saved them for test since I couldn't find the test results used on the netflix challenge. This alternative is sufficient even if it dosen't allow comparing with other published performances.

2.5 THE STOPPING CRITERION

We stop the alternating least squares whenever we consider the RMSE[1] ($= \sqrt{J(U, M)}$) has converged. For computational reasons we would estimate the RMSE only on a subset of the training set. In practice, the RMSE is said to have converged if its variation between two consecutive iterations is no more than a small value defined as tolerance (TOL) in the code.
Each processor computes the residual sum of squares on a randomly[2] chosen subset from its training set (the algorihtm has already seen these values). This local sum is then reduced by the MASTER processor to compute an overall RMSE and decide whether to continue updating the matrices or to break out of the loop.

2.6 LINEAR ALGEBRA

For all the opeartions on matrices I used BLAS and LAPACK namely:

- (BLAS) dsyrk: multiplies a symmetric matrix by its transpose and adds a second matrix for computing $A = M_{K_i} \times {}^{t}M_{K_i} + \lambda n_{u_i} I_{n_f}$ and its equivalent for the users as well.
- (BLAS) dgemv: Multiplies a matrix by a vector to compute $B = M_{K_i} \times {}^{t}R(i, K_i)$
- (LAPACK) dsysv: computes the solution to a real system of linear equations $A * X = B$, where A is an N-by-N symmetric matrix and X and B are vectors of length N. In our case X would be the new user/movie vector.

[1] RMSE for Roote Mean Square Error
[2] generated at the first iteration then maintained

2.7 PSEUDOCODE

```
Preporcessing: /* Used only once to build ratings list.                    */
if MASTER then
    /* Sequentially processing the netflix data.                           */
    Load training set and change users indexing.
    Write binary file for ratings ordered by movie.
    Reorder the ratings by user.
    Write binary file for ratings ordered by user.
    Write binary files for offsets on the two ratings files.
    Write binary file for users mapping.
Main ALS:
if MASTER then
    Load the indices arrays.
    Load the mapping array.
Broadcast(Movies Indices, Users indices and users mapping).
Each processor get assigned a part from the ratings files.
Read Ratings file with determined offset and length.
Build Movies Metadata
Initialize the corresponding part of M
Allgatherv(Movies Matrix M)
while Δ_i RMSE < TOL do
    Update U.
    Barrier.
    Allgatherv U.
    update M.
    Barrier.
    Allgatherv M.
    Compute RMSE on follow-up subset.
Prediction:
Allgatherv Movies Metadata /* Useful to predict ratings for unseen users.  */
Seek assigned part of the test set
Predict ratings and compare with true values to compute participation in the RMSE
Reduce the RMSE for printing output
```

3- PERFORMANCE

With the actual code under LAMBDA=0.04, total iterations of around 25 and only 12 features, we get a RMSE of 0.98. The running time is still considerable with 19s for a single ALS iteration when using 5 processors.

REFERENCES

[1] Zhou, Y., Wilkinson, D., Schreiber, R., & Pan, R. (2008). *Large-scale parallel collaborative filtering for the netflix prize*. In Algorithmic Aspects in Information and Management (pp. 337-348). Springer Berlin Heidelberg.

[2] Yu, H. F., Hsieh, C. J., & Dhillon, I. (2012, December). *Scalable coordinate descent approaches to parallel matrix factorization for recommender systems*. In Data Mining (ICDM), 2012 IEEE 12th International Conference on (pp. 765-774). IEEE.