PARALLEL IMPLEMENTATION OF A PCA-BASED METHOD TO DO MISSING VALUE IMPUTATION IN A NUMERICAL DATASET

João Fontes Gonçalves
Final project - INF560
École Polytechnique
`joao.fontes-goncalves@polytechnique.edu`

May 29, 2020

1 About the method

PCA in the complete case consists in finding a matrix of low rank S that gives the best approximation of a matrix $X_{n\times p}$ with n individuals and p variables assumed to be centered by columns, in the least square sense (with $\|.\|$ the frobenius norm):
$argmin_{\mu}\{\|X_{n\times p}-\mu_{n\times p}\|_2^2 : rank(\mu)\leq S\}$

The PCA solution is the truncated singular value decomposition (SVD) of the matrix $X = U\Lambda^{\frac{1}{2}}V'$ at the order S, namely:
$\hat{\mu}_{ij} = \sum\limits_{s=1}^{S}\sqrt{\lambda_s}u_{is}v_{js}$
PCA has been extended to an incomplete dataset by replacing the least square criterion by an weighted least squares:
$argmin_{\mu}\{\|W\odot(X-\mu)\|_2^2 : rank(\mu)\leq S\}$
where $W_{ij}=0$ when X_{ij} is missing and 1 otherwise and $\odot$ stands for the element-wise multiplication. The aim is to estimate the PCA parameters despite the missing entries. There is no explicit solution for this minimization problem and it's necessary to resort to iterative algorithms. Here, the algorithm used was the one described in (Julie Josse, 2016). It performs:

- 1) Initialization $l=0$: substitute missing values with initial values, such as the mean of the variables with non-missing entries, the imputed matrix is denoted X^0. Calculate M^0, the matrix of the vector containing the mean of the variables of X^0, repeated in each row of M^0.
- 2) Step $l\geq 1$:
  a) perform the SVD of $(X^l - M^l)$ to estimate quantities Λ^l and U^l, V^l.

  b) compute the fitted matrix: $\hat{\mu}^l_{ij} = \sum\limits_{s=1}^{S}\sqrt{\lambda^l_s}u^l_{is}v^l_{js}$ and define the new imputed data as $X^l = W\odot(X-M^l)+(1-W)\odot\hat{\mu}^l$, where $1_{n\times p}$ is a matrix filled with ones. The observed values are the same and the missing ones are replaced with by the fitted values.

  c) $X^l = X^l + M^l$, compute M^l by the new completed matrix X^l.
- 3) steps (2.a), (2.b) and (2.c) are repeated until the change in the imputed matrix falls below a predefined threshold $\sum_{ij}(\hat{\mu}^{l-1}_{ij}-\hat{\mu}^l_{ij})^2\leq\epsilon$ with ϵ equal to 10^{-6} for example.

2 About the possibilities for parallelization

Figure 1 shows the dependency graph for the proposed method.
As the graph shows, there is no parallelism inter-task. Though, we can give a look at the dependency graph for each task.

Task1 consists in a loop through the dataset to get the initialization for the matrix M followed by a new loop to construct the initialization for X. So it's possible to execute it in parallel.

Task2a consists in a singular value decomposition. There is a lot of literature about how to parallelize SVD, though Eigen's library already provide an efficient way to compute it using C++. Hence, I preferred to use it, because SVD's parallelization is not the main objective of this project.

For Task2b, the computation of the fitted matrix can be done with a triple loop (through the dataset and through the number of components). So it's possible to execute it in parallel.

Task2c consists in a loop through the columns of the dataset to get the mean values and it's nested with a loop through the M rows to get the new X for that iteration. So, it can be parallelized.

Finally, Task3 executes the convergence's test, which consists in a nested loop through the squared difference of the fitted matrices from the last two iterations. So, it can also be parallelized.

3 About the parallelism types used

Control parallelism is possible because we can extract the tasks by choosing the right granularity depending on the target execution. The main challenge here is to handle task dependencies and extract the maximum number of independent tasks to increase parallelism.

Flow parallelism is not possible because we don't have different tasks applied on each stream of data.

Data parallelism is possible because during the cross-validation step we need to repeat actions through the same data.

4 About the parallelism paradigms used

Shared-memory programming model is applicable because here there is no problem in considering that the parallel tasks have the same view of memory.

Distributed-memory programming model is also applicable because there is no problem in requiring that parallel tasks work on their own memory space.

5 Parallelization with MPI

For point-to-point communication, it's possible to use **Send** and **Recv** to distribute the loops of all tasks among processes.

For collective communications, **Reduce** can be used on the inner loop of Task2c to compute the mean values that will fill the matrix M and on Task3 to compute the sums in the inner loop.

6 Parallelization with OpenMP

For Task1, we can parallelize the outer loop (through the columns) using **#pragma omp parallel for**. The first inner loop (to calculate mean) can be parallelized with **#pragma omp parallel for reduction(+:mean)** and the second inner loop with **#pragma omp parallel for**.

For Task2b, we can use **# pragma omp parallel for collapse(3)** to parallelize the three nested loops.

For Task2c, we can use **#pragma omp parallel for collapse(2)** to parallelize the two nested loops in order to get the matrix X for the current iteration. After that, it's possible to use **#pragma omp parallel for** in the outer loop of the procedure to get the matrix M and **#pragma omp parallel for reduction(+:mean)** in the inner loop.

For Task3, it's possible to use **# pragma omp parallel for reduction(+:sum) collapse(2)** for the two nested loops to get the squared sum of the difference between the values in the fitted matrix in the two last iterations.

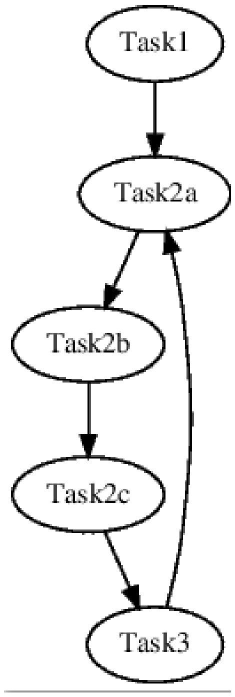

Figure 1: Dependency graph for the parallelization of the proposed method

7 Results

By running the code with the following matrices it's possible to compare the results with parallelization with the results without.

$$M_{4\times3} = \begin{bmatrix} 2.0 & 1.0 & 4.0 \\ 3.0 & 0.0 & 1.0 \\ 0.0 & 4.0 & 3.0 \\ 6.0 & 0.0 & 0.0 \end{bmatrix}$$

$$M_{5\times3} = \begin{bmatrix} 2.0 & 1.0 & 4.0 \\ 3.0 & 0.0 & 1.0 \\ 0.0 & 4.0 & 3.0 \\ 6.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 9.0 \end{bmatrix}$$

$$M_{6\times3} = \begin{bmatrix} 2.0 & 1.0 & 4.0 \\ 3.0 & 0.0 & 1.0 \\ 0.0 & 4.0 & 3.0 \\ 6.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 9.0 \\ 2.0 & 3.0 & 0.0 \end{bmatrix}$$

$$M_{7\times3} = \begin{bmatrix} 2.0 & 1.0 & 4.0 \\ 3.0 & 0.0 & 1.0 \\ 0.0 & 4.0 & 3.0 \\ 6.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 9.0 \\ 2.0 & 3.0 & 0.0 \\ 0.0 & 1.9 & 1.0 \end{bmatrix}$$

With parallelization	Without parallelization
0.497641 s	0.784052 s
5.93 s	6.8902 s
21.183 s	23.7057 s
57.7612 s	62.0069 s

As we can conclude, the time for missing value imputation decreases considerably with the use of parallelization techniques. A possible future work is the complete implementation of missMDA package and its integration with R programming language via shared libraries.