Hadoop Project: Non Negative Matrix Factorization

Lottarini Andrea
Virgilio Daniele

November 25, 2011

1 Abstract

2 Non Negative Matrix Factorization

The non negative matrix factorization is a group of algorithm of linear algebra such that a initial matrix A is factorized in two matrix W and H. The matrix A's elements must be non negative (greater or equal by 0) by assumption. The achieved factorization has the property of $A \simeq WH$ and both the matrices W and H are also non negative. This factorization is in general not unique, in fact, let assume that W_1 and H_1 are a factorization of the matrix A, it's possible to find another factorization using a matrix L and its inverse: $A = W_1H_1 = (WL)(L^{-1}H) = W_2H_2$.

The non negative matrix factorization problem can be formulated as follows:

"Given a non negative matrix $A \in \mathbb{R}^{m \times n}$ and a positive integer $k < min\{m,\ n\}$, find non negative matrices $W \in R^{m \times k}$ and $H \in R^{k \times n}$ to minimize the function

$$f(W, H) = \frac{1}{2}||A - WH||_F^2$$

where F is the Frobenious norm defined as $||X||_F = (\sum_i \sum_j (|x_{i,j}|^2))^{1/2}$. "

In this way, the product of W and H is not exactly equal to A but is an approximation. The problem formulation said that the norm of the residual matrix $(D = A - WH)$ must be minimized in order the find a better approximation. Furthermore, choosing a K less than $\frac{m*n}{m+n}$ the space needed to store the matrix decrease.

The Non Negative Matrix Factorization (NMF) is a conical coordinate transformation[5] where the W matrix describe the basis vector and all the element in the dataset are

enclosed in the cone identified by W. Each element, being in the cone, can be expressed as positive linear combinations of vectors in W. The NMF is equivalent to a relaxed form of K-means clustering where the matrix W contains cluster centroids and H contains cluster membership indicators when least square function is used as estimator of the distance.

As said before, the non negative matrix factorization is a group of algorithm. The algorithm used in this project is the Multiplicative Update Algorithm but there exist other type of algorithm such as the Gradient Descendent Algorithm and Alternating Least Square Algorithm.

The Multiplicative Update Algorithm converge to a point that can or cannot be the global minimum and it's impossible to determine unless a better solution is found. A big issue in this problem rely on the initial generation of the W and H matrices: depending on the initial matrices choice the algorithm can converge to a local minimum rather than to another one. The code of the algorithm is listed bellow in a using the MathLab array operator notation.

```
W = rand(m,k);   % initialize W as random dense matrix
H = rand(k,n);   % initialize H as random dense matrix

for i = 1 : number_of_iteration
        H = H .* (W'T A) ./ (W'T W H);
        W = W .* (A H'T) ./ (W H H'T);
end
```

The .* and ./ operators stand for the multiplication and the division element by element in the matrix.

Current research on Non Negative Matrix Factorization regards:

- Algorithmic – How the factors and factor initialization can discover a global minimum instead a local minimum and speed-up the convergence process.
- Scalability – How the problem can scale when the dimension of the matrix increase.
- Online – How update factorization when new data comes without the need of a recomputing from scratch.

As obvious, we focus on the second problem.

3 Map Reduce Algorithm [4]

The multiplicative update formula used works in the case in which the distribution of the internal elements is Gaussian distributed. In order to analyse the map reduce algorithm we split it in in two phase:

• 3.1 Computation of the updated H Matrix

The resulting matrix can be computed obtaining 2 auxiliary matrixes that help in the computation of the updated matrix. So, the algorithm can be divided in three main phases, where are computed:

– 3.1.1 $X = W^T * A$

This schema can be used to multiply any two giant matrices when one is sparse ad the other is narrow. Let x_j denote the j^{th} column of X, then

$$x_j = \sum_{i=1}^{m} A_{i,j} w_i^T = \sum_{i \in \mathbb{O}_i} A_{i,j} w_i^T$$

This operation can be implemented as two set of map/reduce operations.

* MAP-I: Map $\langle i, j, A_{i,j} \rangle$ and $\langle i, w_i \rangle$ on i such that tuples with the same i are shuffled together in the form of $\langle i, \{w_i, (j, A_{i,j}) \forall j \in \mathbb{O}_i\} \rangle$.
* REDUCE-I: Take $\langle i, \{w_i, (j, A_{i,j})\} \rangle$ and emit $\langle j, A_{i,j} w_i^T \rangle$ $\forall j \in \mathbb{O}_i$.
* MAP-II: Map the $\langle j, A_{i,j} w_i^T \rangle$ on j such as tuples with the same j are shuffled together in the form of $\langle j, \{A_{i,j} w_i^T\} \forall i \in \mathbb{O}^j \rangle$.
* REDUCE-II: Take $\langle j, \{A_{i,j} w_i^T\} \forall i \in \mathbb{O}^j \rangle$, and emit $\langle j, x_j \rangle$ where $x_j = \sum_{i \in \mathbb{O}^j} A_{i,j} w_i^T$.

– 3.1.2 $Y = W^T * W * H$

It's wise to compute Y by first computing $C = W^T W$ and then $Y = CH$ because it maximizes the parallelism while requiring fewer multiplications than $Y = W^T(WH)$. With the partitioning of W along the short dimension, calculation of $W^T W$ can be fully parallelized because

$$W^T W = \sum_{i=1}^{m} w_i^T.$$

It means that each machine can first compute $w_i^T w_i$ (a small k × k matrix) for all the w_i's it hosts, and then send them over for a global summation. The C matrix must be make available to all the entities of the following phase as global shared data.

* MAP-III: Map $\langle i, w_i \rangle$ to $\langle 0, w_i^T w_i \rangle$ where 0 is a dummy key value for data shuffling.
* REDUCE-III: Take $\langle 0, \{w_i^T w_i\}_{i=1}^m \rangle$ and emit $\sum_{i=1}^m w_i^T w_i$.
* MAP-IV: Map the $\langle j, h_j \rangle$ to $\langle j, y_j = C h_j \rangle$.

– 3.1.3 $X = H. * X./Y$

* MAP-V: Map $\langle j, h_j \rangle$, $\langle j, x_j \rangle$ and $\langle j, y_j \rangle$ on j such that tuples with the same j are shuffled together in the form of $\langle j, \{h_j, x_j, y_j\} \rangle$.
* REDUCE-V: Take $\langle j, \{h_j, x_j, y_j\} \rangle$ and emit $\langle j, h_j^{new} \rangle$ where $h_j^{new} = h_j. * x_j./y_j$.

• 3.2 Computation of the updated W Matrix

This phase is very similar to the one described in the above phase.

– 3.2.1 $X = A * H^T$

This schema can be used to multiply any two giant matrices when one is sparse ad the other is narrow. Let x_i denote the i^{th} row of X, then

$$x_i = \sum_{j=1}^{m} A_{i,j} h_j = \sum_{j \in \mathbb{O}_i} A_{i,j} h_j$$

This operation can be implemented as two set of map/reduce operations.

* MAP-I: Map $\langle i, j, A_{i,j} \rangle$ and $\langle j, h_j \rangle$ on j such that tuples with the same i are shuffled together in the form of $\langle j, \{h_j, (i, A_{i,j}) \forall i \in \mathbb{O}_j\} \rangle$.
* REDUCE-I: Take $\langle j, \{h_j, (i, A_{i,j})\} \rangle$ and emit $\langle i, A_{i,j} h_j \rangle$ $\forall i \in \mathbb{O}_j$.
* MAP-II: Map the $\langle i, A_{i,j} h_j \rangle$ on i such as tuples with the same i are shuffled together in the form of $\langle i, \{A_{i,j} h_j\} \forall j \in \mathbb{O}_i \rangle$.
* REDUCE-II: Take $\langle i, \{A_{i,j} h_j\} \forall j \in \mathbb{O}_i \rangle$, and emit $\langle i, x_i \rangle$ where $x_i = \sum_{j \in \mathbb{O}^i} A_{i,j} h_j$.

- **3.2.2** $Y = W * H * H^T$

  It's wise to compute Y by first computing $C = HH^T$ and then $Y = WC$. The C matrix must be make available to all the entities of the following phase as global shared data.

  * MAP-III: Map $\langle j, h_j \rangle$ to $\langle 0, h_j h_j^T \rangle$ where 0 is a dummy key value for data shuffling.
  * REDUCE-III: Take $\langle 0, \{h_j h_j^T\}_{j=1}^n \rangle$ and emit $\sum_{j=1}^n h_j h_j^T$.
  * MAP-IV: Map the $\langle i, w_i \rangle$ to $\langle i, y_i = w_i C \rangle$.

- **3.2.3** $X = W. * X./Y$

  * MAP-V: Map $\langle i, w_i \rangle$, $\langle i, x_i \rangle$ and $\langle i, y_i \rangle$ on i such that tuples with the same j are shuffled together in the form of $\langle i, \{w_i, x_i, y_i\} \rangle$.
  * REDUCE-V: Take $\langle i, \{w_i, x_i, y_i\} \rangle$ and emit $\langle i, w_i^{new} \rangle$ where $w_i^{new} = w_i. * x_i./y_i$.

4 Assumption

In our project, as its done in the paper, we assume that the matrix that we need to factorized is a sparse matrix. So an appropriate internal representation can be used specifing element by element in the A matrix.

5 Implementation

In this section we analysed the main implementation choices, specifying the Hadoop's key features used.

5.1 Types

In this project, we decide to create our types of data that can be transmitted as key or value type. In order to achieve this goal, each classes representing an object in our application domain extend the class *WritableComparable* of the Hadoop framework and override the method:

- *ReadFields() / WriteFields()* - for reading/writing the data in a serializated format.
- *compareTo()* - for comparing two element in the case the type is used as key.

Also dual methods for reading/writing in human-readable format is defined in the method *parseLine()* and *toString()*. The mathematical operations that can be done on the different types are implemented directly in the class as instance method (e.g. multiplication matrix per vector, vector per vector and so on).

The files representing the data in the computation are in a serialized form, both in input and in output. It impose that the Map/Reduce phases need to read/write directly the data from a codified datafile. This task is accomplished by setting the input/output SequenceFile through the *Job* class. The choice of adopting an internal data representation has been done in order to reduce the dimension of the files stored at the end of each phase of the computation and so it improve the performance of the read/write on the Hadoop's Distributed File System. The user, obviously, is able to write the input data only in a readable format so different tool (written as of Map/Reduce programs) for converting the data from readable to codified and vice-versa are provided.

In the computation of the auxiliary matrix X (Phase1) happens that, as output of the maper must be emitted a type that can be either a *SparseVectorElement* or *NMFVector*. So, what we need is a mechanism to abstract from the real type of the element emitted. The first attempted trial was the use the polymorphism to bypass the problems. However, unluckily, the Hadoop framework doesn't support the polymorphism in the emitted type because it does only a run-time checking on the real type and not, as usual, on the apparent type. Anyway, the framework provides a solution that can be used in these cases: the generic types. With the generic type, a new class that extends *GenericWritable* must be implemented. The main goal of this class is return, when is called the method *getTypes()*, an array of all the supported classes. Then, it can be scanned sequentially to look for the rigth type and read the data using the introspection mechanism. This control must be done at every read of an element because every type emitted can have different type.

5.2 Framework Mechanism

Multiple input file

Phase1/5 output map problem: different type + ordering ——¿ la chiave degli elementi vmiene modificata e si prende solo la prima Secondary sorter

Context variable

Combiner

Null Writable

DFS operation in phase 4

Record reader

!! The computation of the H phase is very similar to the W's ones and most of the phase remain unchanged.

6 Result

7 Conclusion

References

[1] Michael W. Berry and Murray Browne. Email surveillance using nonnegative matrix factorization, 2005.

[2] Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. In *Computational Statistics and Data Analysis*, volume Volume 52, pages 155–173, 2006.

[3] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *In NIPS*, pages 556–562. MIT Press, 2001.

[4] Chao Liu, Hung-chih Yang, Jinliang Fan, Li-Wei He, and Yi-Min Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *Proceedings of the 19th international conference on World wide web*, pages 681–690. ACM, 2010.

[5] Ralf Nikolaus. Learning the parts of objects using non-negative matrix factorization (nmf), 2007.

[6] Wikipedia. Non-negative matrix factorization. `http://en.wikipedia.org/wiki/Non-negative_matrix_factorization`.