Biclustering using Spark-MapReduce

Tugdual Sarazin
LIPN-UMR 7030
University of Paris 13 - CNRS
99, av. J-B Celmént
F-93430 Villetaneuse, France
Email: firstname.lastname@lipn.univ-paris13.fr

Mustapha Lebbah and Hanane Azzag
LIPN-UMR 7030
University of Paris 13 - CNRS
99, av. J-B Celmént
F-93430 Villetaneuse, France
Email: firstname.lastname@lipn.univ-paris13.fr

Abstract—Biclustering approaches are more complex compared to the traditional clustering particularly those requiring large dataset and Mapreduce platforms. We propose a new approach of biclustering based on popular self-organizing maps, which is one of the famous unsupervised learning algorithms. We have designed scalable implementations of the new topological biclustering algorithm using MapReduce with the Spark platform.

Index Terms—Biclustering; Self-organizing Map; Map-Reduce Spark;

I. INTRODUCTION

Biclustering refers to simultaneous clustering of observations and their features [1], [2]. Biclustering problems have numerous applications and are becoming more challenging as the size of the data increases. Nevertheless, good clustering algorithms are still extremely valuable and we can (and should) rewrite them for parallel clustering using a new Map-Reduce paradigm [3], [4]. Map-Reduce is the most popular and suited for data already stored on a distributed file system, which offers data replication, as well as the ability to execute computations, locally on each data node. However, the existing parallel programming paradigms are too low-level and ill-suited for implementing machine learning algorithms. The authors of [5] present a portable infrastructure that has been specifically designed to enable the rapid implementation of parallel machine learning algorithms. Recently, a MapReduce-MPI library was made available by Sandia Lab to ease porting the set of columns (features) is denoted by \( I \) and \( x_i \) a vector in \( D \), \( \mathbb{R}^d \), \( d \) is the euclidean distance defines a neighbor cell. Let \( \mathbb{R}^d \) be the euclidean data space and \( D \) the matrix of data, where each observation \( x_i = (x_{i1}, x_{i2}, ..., x_{id}) \) is a vector in \( D \subset \mathbb{R}^d \). The set of rows (observations) is denoted by \( I = \{1, ..., N\} \). Similarly, the set of columns (features) is denoted by \( J = \{1, ..., d\} \). We are interested in simultaneously clustering observation \( I \) into \( K \) clusters \( \{P_1, ..., P_k, ..., P_K\} \), where \( P_k = \{x_i, \phi_i(x_i) = k\} \) and features \( J \) into \( L \) clusters \( \{Q_1, ..., Q_l, ..., Q_L\} \) where \( Q_l = \{x^j, \phi_w(x^j) = l\} \). We denote by \( \phi_w \) the assignment function of row (observation) and \( \phi_z \) the assignment function of column (feature). The main purpose of BiTM-MR is to transform a data matrix \( D \) into a block structure organized in a topological map does. In BiTM-MR, each cell \( r \in C \) is associated with a prototype \( g_k = (g_{k1}, g_{k2}, ..., g_{kd}) \), where \( L < d \) and \( g_k \in \mathbb{R}^d \). To facilitate formulation, we define two binary matrices \( Z = [z_{ik}] \) and \( W = [w_{ij}] \) to save the assignment associated respectively to observations and features: \( z_{ik} = \begin{cases} 1 & \text{if } x_i \in P_k, \\ 0 & \text{else} \end{cases} \) et \( w_{ij} = \begin{cases} 1 & \text{if } x^j \in Q_l, \\ 0 & \text{else} \end{cases} \). To cluster \( D \) into \( K \) and \( L \)

1\text{http://spark-project.org/}

REFERENCES


clusters in both observations and features, we propose the new following objective function to optimize in the biclustering process:

\[ R(W, Z, G) = \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{i=1}^{N} \sum_{j=1}^{d} \sum_{r=1}^{K} H(r, k) z_{ij}^{r} (x_{ij}^{r} - g_{ij}^{r})^2 \]

We can detect the block or bicluster of data denoted by \( B_{ij}^{k} = \{(x_{ij}^{r} | z_{ij}^{r} w_{ij}^{r} = 1\}. \) The objective function can be locally minimized by iteratively.

A. BiTM-MR and MapReduce

To handle this huge amount of data, it is necessary to use distributed architecture. This is not a simple task and several difficulties have to be dealt with, including loading data, failure safety, and algorithm design. The MapReduce implementation on Spark takes care of failure-correction, data management and distribution. It has become very important in MapReduce to decompose our problem in elementary functions. The idea is to initiate two Map-Reduce functions for row and column iterations, and a synchronization to update the parameters \( G, W, Z. \) In the case of BiTM-MR algorithm we identified these atomic parts:

- Assign each observation \( x_i \) to the best match unit using expression: \( \phi_c(x_i) = \arg \min_{c} \sum_{j=1}^{d} \sum_{l=1}^{L} w_{ij}^{c} (x_{ij}^{c} - g_{ij}^{c})^2 \)
- Assign each feature \( x_j \) to the best match unit using expression: \( \phi_w(x_j) = \arg \min_{w} \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ij}^{k} (x_{ij}^{k} - g_{ij}^{k})^2 \)
- Accumulate denominator and numerator for each prototype \( g_r \), where \( g_{ij}^{r} = \sum_{k=1}^{K} \sum_{l=1}^{L} H(r, l) \sum_{x_{ij}^{k} \in B_{ij}^{k}} x_{ij}^{k} \)
- Update prototype vectors \( g_r, \forall r \in C \)

Pseudo code below describes the implementation of the BiTM-MR algorithm with MapReduce Spark. In most of the cases in big dataset the number of observations is bigger than the number of features \( (N >> d) \). Thus, we consider that a column vector \( x_{ij}^{r} \) (column) of the data matrix \( D \) couldn’t be used as an observation vector \( x_i \) (row). Hence for the row assignment function we define row map function (Algorithm 4), which computes for each \( x_i \) the best match unit. In order to reduce the time complexity we compute the denominator and the numerator of the prototype \( g \). This allows the row reduce function to focus only on the sum of different numerators and denominators provided by each row map function. For the column assignment, we define a map function (2) and a reduce function (Algorithm 5) in order to reduce the memory consumption. Thus the column assignment function splits the column distance of one column into multiple outputs \( (d \times K) \). The map function computes the distance between each element \( x_{ij}^{r} \) and all prototypes. Hence the column reduce function sums the "distances" provided by each column map function. At the end of this MapReduce state the best match prototype for features is computed in the main function (Algorithm 1). Therefore, the update of the prototype is computed.

III. EXPERIMENTS

Our performance study is based on the synthetic datasets. The parameter \( T_{max} \) and \( T_{min} \) were fixed as traditional

---

**Algorithm 1 BiTM-MR - Main**

**Initialization**

{Random initialization of prototypes and \( W \) }

{Main loop}

while \( t \leq t_{max} \) do

{Assignment of columns}

for all \( x_i \in D \) do

\( (\langle J, L \rangle; V) \leftarrow \text{ColMapper}(x_i) \)

end for

for each column reduce value \( j \in J \) do

\( \phi_w(x_j) = \arg \min ((\langle J, L \rangle; V)) \)

end for

{Assignment of rows}

for all \( x_i \in D \) do

\( (\langle \text{CM, CN} \rangle) = \text{RowMapper}(x_i) \)

end for

end while

**Algorithm 2 ColMapper(x_i)**

for each column \( j = 1..d \) do

for each feature prototype \( l = 1..L \) do

emit (\( \langle j, l \rangle; (x_{ij}^{l} - g_{ij}^{l})^2 \))

end for

end for

**Algorithm 3 ColReducer(key(j,l), V)**

\( s_u = 0 \)

for each map value \( v \in V \) do

\( s_u = s_u + v \)

end for

emit (\( \langle j, l \rangle; s_u \))

**Algorithm 4 RowMapper(x_i)**

{MAP rows distance: distributed loop over all input vectors \( x_i = (x_{i1}, x_{i2}, \ldots x_{id}) \)}

for each \( k = 1..K \) do

\( \text{bmu}(k) = \| x_i - g_k \|^2 \)

end for

\( \phi_c(x_i) = \arg \min (\text{bmu}) \)

{Contruct a new compressed vector : \( \text{cn} \) with \( 1 \times L \) dimensions}

\( \text{cn}(l) = \sum_{x_{ij}^{k} \in B_{ij}^{k}(x_i)} x_{ij}^{k} \)

CM = \( \text{h}^{\phi_c} \times \text{cn} \)

{So CM and CN are matrices of size \( K \times L \})

emit (\( \langle \text{CM, CN} \rangle \))

**Algorithm 5 RowReducer(V(CM, CN))**

Initialize \( \text{CMs} \leftarrow 0, \text{CNs} \leftarrow 0 \)

for each \( (\text{CM}, \text{CN}) \in V \) do

\( \text{CMs} += \text{CM} \)

\( \text{CNs} += \text{CN} \)

end for

emit (\( \langle \text{CMs}, \text{CNs} \rangle) \)
topological map \( (T_{max} \text{ is fixed as the radius of the map}) \). We used 4 synthetic datasets generated with 1 million and 2 million observations with 20 and 40 features each. This datasets are used for speedup test. An open source Spark library that include the topological biclustering and our implementation of SOM (self-organizing map) is available on Github https://github.com/TugdualSarazin/spark-clustering.

We made scaling experiments on the Magi cluster of Paris 13 university (http://www.univ-paris13.fr/calcul/wiki/) with 1 to 10 machines of the cluster. Each machine on this cluster has 12 cores (two Xeon X5670 at 2.93GHz), 24GB of RAM and they are connected by an InfiniBand network. All the experiments are done in Spark Platform. We generate different dataset using the Scala random normal function. We apply a different factor for each class. \( x = a + N \times b \). In each of the two classes, \( a \) and \( b \) are different. But neither the number of class nor the data generation function influences significantly the execution time of the algorithm. We trained a 10 \times 10 BiTM-MR map with different core counts in each run. The run time is indicated in milliseconds. In all figures below we added an ideal curve. It had been determined by the execution time for one machine divided by the number of machines. This curve represents a perfect scaling architecture. When the size of the data matrix is low the performances of the initialization of the system decrease. But with a huge data matrix the initialization tends to be insignificant proportionally to the global run time. As we can see in the following figures 1(a), 1(b), 1(c), 1(d). The implementation exhibited excellent linear scaling, and is close to ideal times.

**IV. CONCLUSION**

We have presented some experiments to promote the new biclustering algorithm using Spark-MapReduce. The obtained preliminary results indicate that the design used for the BiTM-MR algorithm can be extended to the other biclustering algorithms using MapReduce. In the future, we plan also to develop a method that can automatically divide features into cluster using the weighted biclustering process. We plan also to investigate the applicability of this work on real and more difficult dataset in order to propose a complete library of clustering and biclustering in Spark.

**REFERENCES**


