Performance improvements of a Kohonen self organizing classification algorithm on sparse data sets

FRANCESCO MAIORANA
University of Catania, Department of Computer Engineering and Telecomminications
Via A. Doria, 6 95127 Catania
ITALY

Abstract: - This paper presents a variation of a Kohonen self organizing feature map. From the proposed algorithm possible performance improvements are investigated in terms of time and space complexity taking advantage from a sparse input data set. The proposed variation has been tested on different datasets coming from case studies in the field of bioinformatics. The improvements make the application of the algorithms feasible to massive document collections. The application of the proposed improvements for grid implementations could be beneficial to reduce the computing element demand.

Key-Words: - Clustering algorithm, Kohonen Self Organizing Map, Space and Time complexity, performance improvements

1 Introduction

Clustering algorithms are playing a central role in data analysis and exploration. This is true in almost every field of science. In fact, nowadays the amount of data produced and stored is surprisingly increasing especially in life science and in gene expression data collected by microarray experiments [1, 2].

To overcome the increasing computational demand coming from the exponentially rising volume of data generated and, eventually, by the change in space representation (from feature to similarity space representation) the paper proposes and evaluates some performance improvements in terms of computational time and space. These improvements are more effective on sparse datasets. The results in terms of execution time reduction and concordance between the clusters obtained by the improved algorithms against the Kohonenn algorithm without performance improvement, which has been taken as reference point, are presented and evaluated.

The paper is organized as follows: section 2 briefly recalls the Kohonen self organizing feature map algorithms; section 3 revises the literature about fast implementation of the Kohonen algorithms; section 4 describes some implementations of fast Kohonen algorithm and compares the results for exact and approximate algorithms; section 5 draws some conclusions and feature works.

2 Brief review of the SOM algorithm

Kohonen Self Organizing Maps (SOM) are often used to cluster datasets in an unsupervised manner [3, 4, 5]. This paper deals with on–line SOM since the batch version has some disadvantages such as the fact that it often represents an approximation of the on–line algorithm [6].

In the on–line version the weights are updated after the presentation of each input vector. In order to do this, the distance (usually the Euclidean distance) is computed between the input vector and each weight vector as in (1).

$$d_k(t) = \|x(t) - w_k(t)\| \qquad K = 1...no \qquad (1)$$

where no is the number of output neurons.

In the second step the algorithm searches for the winning neuron, $d_{w,}$ i.e., the neuron that best matches the input neuron and is characterized by the minimum distance from the input vector

$$d_w(t) = \min_k(d_k(t)) \quad K = 1...no \qquad (2)$$

In the third phase the algorithm updates the weights of the winning neuron and of the neurons that lie in a user defined neighborhood as follows

$$w_k(t+1) = w_k(t) + \alpha(t) h_{kw}(t) \|x(t) - w(t)\| \quad K = 1..no \quad (3)$$

where $\alpha(t)$ is the learning rate that modulates the weight update, and h_{kw} is the neighborhood function that depends, given a time t, on the winning neuron w and the neuron under consideration k.

Usually the output neurons are arranged in a bi-dimensional array; however, some implementations have been proposed which adopt a different topology of the network where the output neurons are arranged along a single layer (SL configuration) [7, 8].

In the SL configuration the network topology is composed of an input layer with as many nodes as the number of components of the input element, and an output layer with as many nodes as the number of classes.

This means that if, at the final cycle, the winning neuron mostly activated by the i^{th} item is the j^{th} neuron, then the input object belongs to the class j.

In this scenery there is no topological similarity among output neurons since adjacent output neurons do not necessarily represent similar classes.
Let us note that in the SL configuration the updating formula (3) is replaced by a neighborhood function that chooses the winning neurons and the ones (usually two o three neurons) that are mostly activated by the current input object. The neighborhood function is not a topological but a logical one that finds the output neurons closer to the input vector.
As neighborhood function the following one has been proposed in [7]:

$$h_{kw}(t) = \frac{1}{ord(k)^2} \qquad (4)$$

where ord (k) is the rank of weight vector k in the ordered vector of distance computed with formula 1).
The SL clustering algorithms work on both the feature and the similarity space as proposed in In [7, 8]. If the similarity space is considered, the algorithm allows us to perform a final step in which, for each class, it is possible to find both the most relevant features common to the majority of class elements, called positive features, and the features not present in the majority of class elements, called negative features.
An automatic strategy to find the optimal number of classes is also proposed in [7, 8].

3 Reviews of performance improvements of the Kohonen SOM algorithms

Some algorithms belonging to the Kohonen family (here on referred to as Kohonen–like algorithms can be summarized by the following three phases:
Phase D: it computes the distance between input and weight vector by equation (1);
Phase W: it computes the winning neuron by equation (2)
Phase U: it updates the weight by equation (3) using equation (4) for the neighbourhood.
The computational complexity of each iteration of our version of the on-line Kohonen algorithm without any optimization is :

1. Phase D: $O(N^2 * no)$
2. Phase W : $O(N * (no * \log (no)))$ since the vector of distances must ordered if all the output neurons should be updated according to their rank.
3. Phase U: $O (N^2 * no)$

where N is the number of input elements, no is the number of classes or output neurons. The analysis is performed for a classification step involving the similarity matrix that has a dimension of N x N.
In literature several studies concerning performance improvements of the Kohonen like algorithm can be found.
In [9] the authors propose the use of spatial indexing method such as R-Tree in order to speed up the search of the winning neuron to reduce the cost from O(no) to $\log_m$ (no) where m is the node size.
In [10] the authors propose a fast implementation for a batch version of the standard Kohonen algorithm. The optimization they propose has the following computational complexity for the various phases of the Kohonen algorithm:

1) Phase D: $O(no* (N * f_{non\ zero})^2)$
2) Phase W: O (no)
3) Update weight O (no * N * $f_{non\ zero}$)

where $f_{nonzero}$ is the percentage of elements different from zero in the input matrix. Their implementation takes advantage from the batch implementation and from a different arrangement of equation (1) so that in computing the distance in equation 1 only the elements of x different from zero are taken into consideration.
This is possible since in the batch version the weights are updated only at the end of a phase, i.e., after the presentation of all the input elements, so allowing the weights to be pre-computed at the beginning of each epoch.
In [11] the author proposes an on-line implementation of the Kohonen-like algorithm with the following computational complexity:

1) Phase D, computes Distance: $O(no* N^2 * f_{non\ zero})$
2) Phase W, computes Winning neuron: O (N * no)
3) Phase U, Updates weights: O ($no * N^2 * f_{non\ zero}$)

To achieve this result a normalized set of weight $\mathbf{z}_k$ is used such that $\mathbf{w}_k = \beta_k \mathbf{z}_k$. This set of weight $\mathbf{z}_k$ can be updated at a cost proportional to the number of non-zero elements of the input vector. The algorithm does not update all the normalized weights after each presentation of an input vector, but only the weights corresponding to elements of the input vector different from zero, reducing the overall computational cost.
The drawback is that in the update steps it is necessary to update the normalized weights and two constant (β_k and η_k in the paper) with a computational cost proportional to the number of input components different from zero.
Another drawback is that if the value of β_k drops below a predefined threshold (0.01) the updating equations change with a computational cost that is no longer determined by the sparsity of the input matrix.
In [12] the author uses an early stopping strategy in computing the distance between the input element and the output ones. In summing up the squared difference between the components of the i^{th} input element and the weight vector ones, he stops when the summation is above the current minimum. In the paper it is suggested to try to start from the output neuron with the expected smallest distance.
The observation that in many applications, such as speech recognition and image processing, successive vectors exhibit strong correlation, leads to the conclusion that the best matching node (BMN) found for the last input can be the best candidate for the BMN for the successive input.

The author reports a percentage of CPU time saved ranging from 51.3 % to 56.2 % on real speech data composed of 17,179 weighted cepstrum vectors with a dimension of 12 components for a map size ranging from 10x10 to 20x20. .

In [13] the author finds, for a given input vector, the new winner (at phase t + 1) in the vicinity of the old one (at phase t) by storing the old winner in a table containing, for each training vector, a pointer to the winner.

This is particularly true when the SOM is already smoothly ordered although not yet asymptotically stable. In searching for the winning neuron at phase t + 1 the author suggests to locate the winner for phase t in the table and to perform a local search for the winner in the neighbourhood around the located unit. If the best match is found at the edge of this neighbourhood, the search is continued in the surround of the preliminary best match. This principle can be used in both on-line and batch version of the SOM.

The author also suggests to estimate initial value of a map on the basis of the asymptotic values of a map with a much smaller number of units.

In [14] the authors compare the performance of a conventional SOM algorithm and a modification proposed by one of the authors. The modification consists in:

a) selecting the first 2^m +1 neurons (N_w) which best match the input vector;
b) correcting the weight in the set N_w by equation 3)
c) exchanging the weight vectors of BMU's neighbors with the neurons in N_w, so that all the winning neurons will cluster together as neighbors with BMU at the centre.

In their implementation they chose m = 2. The new approach is faster but less stable since the original neighbors around the BMU are forced to leave the original class and thus may disrupt the harmony elsewhere in the map. They report a speed up improvement ranging from 9.68 % to 30.3 % on different datasets with a performance improvement that tends to increase along with the dimensionality of the input data.

In [15] several optimization strategies are proposed for dissimilarity batch self organizing maps, e.g., the possibility to use pre computed values, a monitoring of the clusters that change and an early stopping strategy in computing equation 1) (they stop in computing when the partial sum is above the minimum).

This last optimization however is dependent on the dataset and on the order of presentation of the input elements. To reduce this dependency they propose to first compute the distance of the elements which are the best candidate winning neurons.

In [16] the authors extend the optimization technique by applying the branch and bound principle to reduce the expected cost of the minimization problem by avoiding an exhaustive search. The method introduces some approximations.

The results show, as reported by the authors, that the branch and bound principle reduces a lot the search burden; the speeding up along with the number of classes increases although the speeding up decreases along with the number of elements since the search phase has not a dominant computational cost.

In [17] the authors compared several classification techniques that deal with large datasets by approximation techniques, by sampling the data sets, by randomized search in the solution space, or by a probabilistic parallel randomized search strategy implemented by genetic algorithms.

In [18] the author proposes, for a batch dissimilarity SOM, to work on a random sample of the original data set instead of working on the entire one. The random sample will fit in main memory and will be much smaller than the original data set. The author uses the Chernoff bounds to calculate the minimum sample size for which the sample contains, with high probability, at least a significant fraction of every cluster.

If the clustering algorithm on the reduced dataset finds small clusters, new representations for these clusters are searched among the remaining data.

In [19] the authors suggest a batch learning algorithms that update the weights after processing 15% of the training examples, not only after processing all the training examples as requested by the batch algorithm. The proposed algorithm achieves about half of the acceleration of the batch algorithm without showing its negative effects in term of correct classification rate.

4 Kohonen SOM improvements

This paper uses the Kohonen-like algorithm proposed in [7, 8] as reference point, whereas a sparse dataset of 3,528 rows and 262 columns used in [20] to discover and evaluate hopefully new gene-disease relationships from MEDLINE abstracts has been chosen to give a realistic basis to the results. This dataset represents a vector space representation of the chosen set of abstracts.

From this vector space model representation the similarity space one has been built. This representation is based on the similarity matrix: a symmetric matrix where the element at row i and column j contains the similarity between the i^{th} and the j^{th} element. In this paper it has been adopted a similarity in a broad sense defined by the sum of the minimum of each pair of vector components.

The similarity matrix obtained is normalized between zero and one. Let us note that a strict similarity measure may be obtained by normalizing each row in such a way that the sum of its elements is equal to one.

The similarity matrix used for classifications has a dimension of 3,528 X 3,528. The total number of ones is 1,900, 992 out of 12,446,784 elements equal to 15.27% of ones.

The weighted average number of ones for each column (row) is 539 elements. Figure 1 shows the number of columns with different number of ones.

The first implementation of the Kohonen-like algorithm was implemented with the possibility to use different metrics to compute the distance between the input elements and the model and many other features.
The first step was to optimize this implementation by using only one distance, a prior check of elements that can be eliminated since equal to other elements, to introduce some optimization techniques to compute the winning neuron, to introduce some short-cuts in the implementation of the algorithms such as avoiding the use of function, avoiding, when possible, the use of power in favour of multiplications, or multiplications in favour of additions, avoiding the use of intermediate variables and so on.
In the second step a compact representation of the similarity matrix is proposed.
This representation consists in maintaining for each row the list of elements greater than zero, storing for each row the column numbers and the values different from zero. A count of the elements greater than zero for each row will allow a compact memorization of the matrix and a faster searching inside this one.

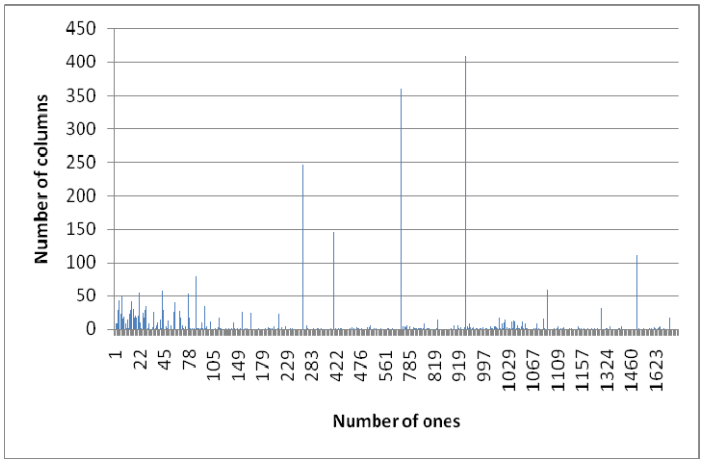


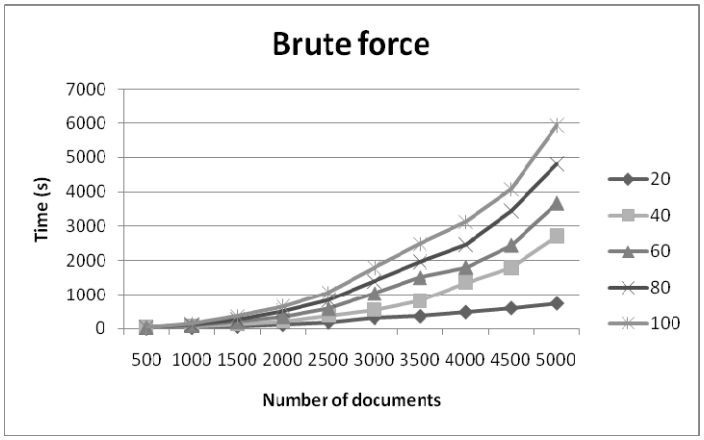
Fig. 1. Number of columns with different number of ones.

The above mentioned representation allows us to consider only the element in the similarity matrix different from zero. The space complexity to store the similarity matrix, with this representation, drops from O (N^2) to O (($N * f_{non\ zero}$)^2). The similarity matrix can be pre-computed and used in all the cycles of the Kohonen-like algorithm.
Moreover, the compact representation of the similarity matrix implies also a time improvement. For example, if 50,000 documents are considered and the similarity matrix is stored as double, its allocations requires more or less 20 G. The time required for its allocation, on an Apple with Intel Xeon Dual Core 2 GHz 64 bit, RAM 4 GB DIMM DDR2 667 MHz, 300GB hard disk, with MacOs X 10.4 operating system was 35 minutes. Using the compact representation, since the numbers of ones in the original similarity matrix is around 15%, of the total number of elements, the space requirement drops to 3 GB thus reducing the virtual memory allocation requirements and hence the overall allocation time becomes insignificant (less than five seconds).
The best optimization in terms of computational time can be obtained if the computation of equation (1) is performed by taking in consideration only the value of s that are different from zero.
Equation 1) can be rewritten as:

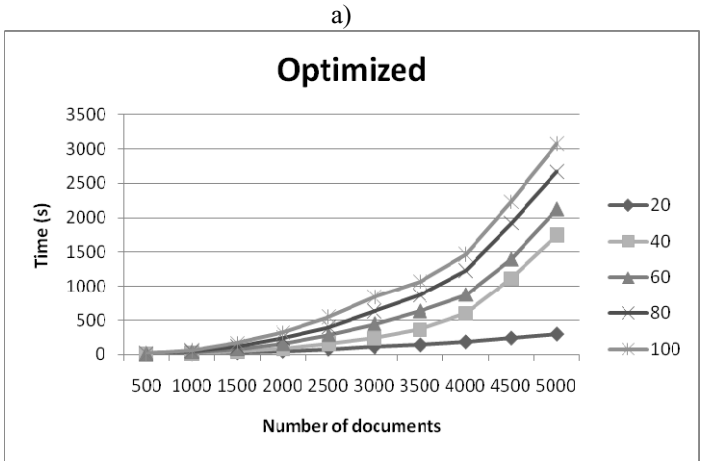
$$d_k(t) = \sum_{x_i \neq 0} x_i(t)(x_i(t) - 2w_{ki}(t)) + \sum_{i=1}^{N} w_{ki}(t)^2 \quad (5)$$

In equation (5) the first summation is computed over the elements of the input matrix that are different from zero thus reducing the complexity from O (N^2 * no) to O (N^2 * $f_{non\ zero}$ * no). The second summation can be pre-computed and updated during Phase U (update phase). The overall running time to compute equation (5) remains O (N^2 * $f_{non\ zero}$ * no) multiplied by a small constant since only 2 * $x_i(t)$ * $w_{ki}(t)$ must be computed in each iteration. The other computations can be done using pre-computed values, only at the beginning of the algorithm, for $x_i(t)^2$; and pre-computed values, during the update phase, for $w_{ki}(t)^2$.
It has also been tried to avoid a pre-computation of the second summation in the update phase and to compute it on the fly in the distance phase using an early stopping strategy. It has also been implemented the possibility, in the computation of equation 5, to start from the winning neuron of the previous phase for the same input elements, in order to improve the gain in the early stopping strategy. This optimization however is dependent on the dataset, and requires some extra computations.
It is important to notice that all the proposed optimizations are exact optimizations that do not produce any classification errors in respect to the original algorithms.
The computational time required by the algorithm can be further reduced by considering, in the computation of the neighborhood function, only the first three neurons with the smallest distance from the examined input vector.
With the chosen neighborhood function the contribution of the third element is 1/9, to be multiplied by the leaning rate that is less than one thus reducing the overall contribution further. In this case the computational complexity of the update phase decreases to O (N^2) with a constant of six since three columns of the weight vector and its squared components must be updated. This case has the advantage to lose the proportionality with the number of classes, which increases along with the number of documents to be classified. This optimization makes the pre computation of the second summation of equations 5) still more appealing.
Various simulations have been performed for different number of documents (ranging from 500 to 5,000 in steps of 500) and different number of classes (ranging from 20 to 100 in steps of 20) of the optimized brute force classification (without any advantage of the sparsity of the matrix), the optimized version of the classification

algorithms by equations 5) with pre computed values for the second summation; the brute force algorithm with a neighborhood of three elements and the optimized version by equation 5) with a neighborhood of three elements.
This choice of the optimization strategy makes the comparison not dependent on the dataset. The results are summarized in figures 2 and 3.
All the classifications were performed on laptop computer with an AMD Turion 64 bit Mobile Technology ML – 30 1.6 GHz with 2 GB DDR2 RAM with Windows XP 32 bit operating system. The classification algorithms were developed in Java. No virtual memory was required in order to run the algorithm.



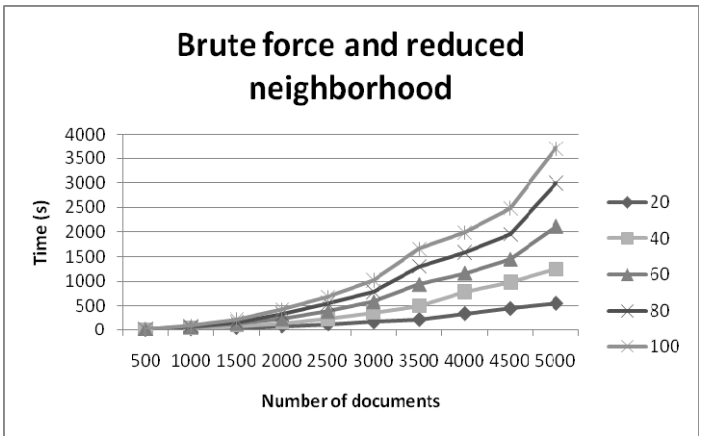
a)



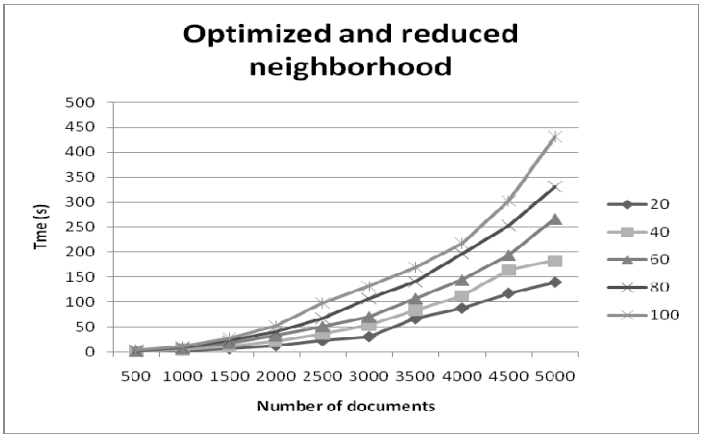
b)

Fig. 2. Execution time for different number of documents and classes: a) brute force; b) optimized version

From the figures the optimization gained with the proposed algorithms can be observed: the time required in the optimized version is halved, and in the optimized version with reduced neighborhood is almost reduced to 1/14. The gain increases with the number of elements to be classified or with the number of classes.
An approximation version which quantizies the values in the similarity matrix and in the weight matrix in 10 classes or quantizied ranges starting from 0.1 to 1 with 0.1 quantization steps has also been designed and implemented.



a)



b)

Fig. 3. Execution time for the classification algorithms with reduced neighborhood for different number of documents and classes: a) brute force; b) optimized version.

This quantization allows us to pre-compute the number of elements in each quantizied range. The knowledge of the number of elements in each interval allows us, in the distance and update phases, to perform an operation per quantization interval, by multiplying the changes by the number of elements in the considered interval. When the matrices are quantizied, a performance improvement of 12.7 % over the optimized version has been observed, but the price paid is a correct classification rate of 0.92 when compared with the exact algorithm. From the data it is possible to conclude that, for massive document collections, when a grid infrastructure is not available, the quantization of s gives a performance improvement with a reasonable classification error

5 Conclusion and future work

This paper proposed several optimization strategies of the Kohonen-like algorithm that takes advantage from the sparsity of the input matrix.
The algorithm has been applied in a similarity space, but the same considerations can be made for the feature one.
Several exact optimization strategies have been presented in both time and space. Moreover, the approximate

optimization strategies have been evaluated with respect to classification errors.

From the analysis the following conclusions can be drawn:

- Use of a compressed notation for the similarity matrix can lead to a drastic drop in space requirements. This also gives a time improvement due to less virtual memory allocation time necessary.
- Optimization strategies that take advantage from the sparsity of the input matrix can drop the time requirement by a factor of ten or more.

Non exact optimization strategy can lead to a time improvement of 15% from the previous case but with a correct classification rate of 0.92.

It could be useful to apply the proposed algorithm to other dataset to evaluate the time and space requirements even if it is expected that the improvements depend on the grade of sparsity of either the feature or the similarity space.

Studies are planned to find an optimization strategy which takes advantage, in computing the distance between the input element and the winning neuron by equation (1), from the expected correlation of subsequent elements and the expected locality of the winning neuron between one phase and the successive one.

How to speed up grid implementations of the Kohonen like algorithm, such as the one proposed in [20], is for further studies.

6 Acknowledgment

This work was supported by the program "ICT per l'Eccellenza dei territory - Intervento 1 – Piano ICT per l'Eccellenza del settore Hi-Tech nel territorio Catanese (ICT-E1)" promoted by the Italian Ministry of Innovation and by Catania Municipality.

References:

[1] Y. Zhao, G. Karypis, "Data clustering in life science", *Molecular Biotechnology*, vol. 31, no. 1, 2005, pp. 55—80

[2] R. Xu, D. Wunsch "Survey of Clustering Algorithms", *IEEE Transactions on Neural Networks*, vol. 16, no. 3, 2005.

[3] T. Kohonen, "*Self Organizing Maps*", Springer 1995

[4] S. Kaski, J. Kangas, T. Kohonen, "Bibliography of self organizing map (SOM) papers: 1981 – 1997, *Neural Computing Survey*, vol. 1, no. 3, 1998, pp. 102—350

[5] M. Oja, S. Kaski, T. Kohonen, "Bibliography of self organizing map (SOM) papers: 1998 – 2001 Addendum, *Neural Computing Survey*, vol. 3, no. 1, 2003, pp. 1—156

[6] M. Cottrel, J. C. Fort, P. Letremy, "Advantages and drawbacks of the batch Kohonen Algorithm", 10th European Symp. On Artificial Neural Network. Bruges (Belgium), 2005, pp. 223—230.

[7] A. Faro, D. Giordano, F. Maiorana, "Discovering complex regularities by adaptive Self Organizing classification", *Enformatika*, vol. I, 2005, pp. 27--30

[8] A. Faro, D. Giordano, F. Maiorana, "Discovering complex regularities from tree to semi – lattice classifications". *International Journal of Computational Intelligence*, vol. 2, no. 1, 2005, pp. 34—39

[9] E. C. Vargas, R. Francelin Romero, K. Obermayer, "Speeding up algorithms for SOM family for large and high dimensional databases", Proceedings of the Workshop on Self Organizing Maps Hibikino (Japan), 2003, pp. 167-172.

[10] R. D. Lawrence, G. S. Almasi, H. F. Rushmejer, "A scalable parallel algorithm for Self organizing maps with applications to sparse data mining problems", *Data Mining and Knowledge Discovery*, vol. 3, no. 171, 1999, pp 171 – 195.

[11] R. Natarajan, "Exploratory data analysis in large sparse datasets", IBM Research Report RC 20749, IBM Research, Yorktown Heights, NY, 1997.

[12] Z. Zhao, "Improvements to Kohonen self-organising algorithm", *Electronics Letters*, vol. 30, no. 6, 1994, pp. 502 – 503.

[13] T. Kohonen, T. "Speedup of SOM Computation"

[14] B. K. Y. Chan, W. W. S. Chu, L. Xu, "Empirical comparison between two computational strategies for topological self-organization", *Intelligent Data Engineering and Automated Learning* (LNCS), vol 2690, Springer, 2003, pp. 410-414

[15] B. C. Guez, F. Rossi, A. E. Golli, "Fast algorithm and implementation of dissimilarity self organizing maps", *Neural Networks*, vol. 19, 2006, pp. 855 – 863.

[16] B. C. Guez, F. Rossi, "Speeding up the dissimilarity Self Organizing Maps by Branch and Bound", *Computational and Ambient Intelligence* (LNCS), vol. 4507, Springer, 2007, pp 203-210.

[17] C. Wei, Y. Lee, C. Hsu, " Empirical comparison of fast partitioning-based clustering algorithms for large data sets", *Expert Systems with applications*, vol 24, 2003, pp. 351 – 363.

[18] A. El Golli, "Speeding up the self organizing map for dissimilarity data", Proceedings of International Symposium on Applied Stochastic Models and Data Analysis, Brest, France, 2005, pp. 709-713.

[19] M. Nocker, F. Morchen, A. Ultsch, "An algorithm for fast and reliable ESOM learning", Proceedings of 14th European Symposium on Artificial Neural Networks, Bruges, Belgium, 2006, pp. 131-136

[20] A. Faro, D. Giordano, F. Maiorana, C.Spampinato, "Discovering Genes–Diseases Associations from Specialized Literature using the GRID", to appear on IEEE *Transaction on Information Technology in Biomedicine*.