Unsupervised Data Classification for Convex and Non Convex Classes

Fairouz Lekhal
LABO LEPAS, FS, University Mohammed I, MAROC

Mohamed El Hitmy
LABO LEPAS, FS, University Mohammed I, MAROC

Ouafae El Melhaoui
LABO LEPAS, FS, University Mohammed I, MAROC

ABSTRACT
We present in this work, a new unsupervised data classification technique based on a three steps system: Split, Clean and Merge. In this system, the classes are represented by a set of subclasses that we call prototypes. The prototypes are created in an incremental way from the initial data set. No prior knowledge on the classes is required. The data are presented to the system one by one in an arbitrary way. The system built on a neural network strategy ends up by acquiring knowledge on the data and gathers the data into a set of real classes which may have a non convex structure. The method proposed is compared to the fuzzy C-means '*FCM'* and fuzzy min max clustering '*FMMC'* methods through a number of simulations. The results obtained by the proposed method are very good.

Keywords
Unsupervised classification, split, clean, merge, convex and non convex classes, fuzzy min max clustering, fuzzy C-means.

1. INTRODUCTION
Unsupervised data classification is an important technique in the field of data analysis; it has played an important role in many scientific fields. The objective of unsupervised data classification is to regroup the data into classes according to a similarity criterion. The data are not labeled and no prior knowledge on the classes is available [1] [10].

Several methods for unsupervised data classification are already very common and these include [4]: hierarchical, partitioning, connexionist, by tree, by graph … etc. The most common among these are the partitioning methods. The algorithms used in partitioning methods include: C-means [5], Fuzzy C-means [6] [11], Isodata [7], competitive neural network [5]… etc. These algorithms optimize iteratively a classification criterion, in order to partition a set of observations into a set of classes. The drawback with these algorithms is to initially set arbitrarily the number of classes *C* to a certain value and the value of *C* is varied until optimality is achieved. Optimality criteria such as Xie and Beni, entropy and partition coefficient are commonly used [8] [11]. Another drawback is the initialization of centers. Starting the algorithm with a different initial value yield to another convergence point different from the first one. Authors have proposed evolutionary algorithms in order to get around these two drawbacks [5] [10]. C-means, fuzzy C-means and competitive neural networks are well suited to convex classes having spherical or elliptical structures; they are not convenient to non convex or complex classes. Many authors have been interested to such a type of problem, regarding the form of the classes, they have proposed a multitude of methods based on existing data classification methods, such as *SVM* [13], Fuzzy pattern matching *FPM* [11], neural networks [12]… etc. The most commonly used algorithm for this problem remains however the one proposed in [3] which is based on a neural network structure and called fuzzy min max clustering *FMMC*.

In this work, we propose a new method for unsupervised data classification, which prove to be valid for convex and non convex type of classes. The method is based on three steps: split, clean and merge. The split technique is based on a neural network with evolving architecture; the network contains two layers which will make it possible to divide the data into several prototypes in an incremental way. The prototypes obtained are characterizing elements of the real classes. The clean step discards the noisy prototypes, which are non representative of the real classes. Merge step is an algorithm based on an evolving neural network, constituted of three layers. The merge technique is well adapted for regrouping the similar prototypes into end classes by means of a merge procedure after that the clean operation has been performed.

2. FUZZY CLASSIFICATION
2.1. Descriptive elements
Let's consider a set of *M* objects $\{O_1, O_2,..,O_i, .., O_M\}$, characterized by *N* parameters regrouped in a line initial vector $V_{init} = (a_1, a_2, .., a_n, .., a_N)$. Let $R_i = (a_{in})_{1 \le n \le N}$ be a line vector of R^N where the n^{th} component a_{in} is the value taken by a_n for the object O_i. R_i is called the observation associated with O_i. R^N is the observation space, or the parameters space. Let $E_V = \{a_1, a_2, .., a_n, .., a_N\}$ be the set of attributes associated with V_{init}.

2.2. Fuzzy C-means algorithm.
Let's consider *M* observations $(R_i)_{1 \le i \le M}$ to be associated to *C* different classes $(CL_s)_{1 \le s \le C}$ with respective centers $(x_s)_{1 \le s \le C}$. Fuzzy C-means algorithm *FCM* associates with each observation R_i its membership degree μ_{is}, to the class CL_s. $\mu_{is} \in [0, 1]$. *FCM* method consists of determining the class centers which minimize the optimization criterion defined by [5] [10]:

$$J_m = \sum_{i=1}^{M} \sum_{s=1}^{C} (\mu_{is})^{df} \ \|R_i - x_s\|^2$$

Under the constraints:

$$\sum_{s=1}^{C} \mu_{is} = 1 \text{ for } i=1 \text{ to } M \quad \text{and } 0 < \sum_{i=1}^{M} \mu_{is} < \text{M for } s=1 \text{ to } C$$

$\|.\|$ is the Euclidean distance, *df* is the ''fuzzy degree'' often taken equal to 2. In order to minimize J_m, μ_{is} and x_s, must be updated at each iteration according to [10]:

$$\mu_{is} = \frac{\left(\left\|R_i - x_s\right\|^2\right)^{\frac{1}{1-df}}}{\sum_{k=1}^{C}\left(\left\|R_i - x_k\right\|^2\right)^{\frac{1}{1-df}}} \quad \text{and} \quad x_s = \frac{\sum_{i=1}^{M}(\mu_{is})^{df} R_i}{\sum_{i=1}^{M}(\mu_{is})^{df}}$$

Despite the simplicity and the popularity of this algorithm, it suffers from the same drawbacks as the *C*-means does. These are: the number of classes must be known a priori, the initialization problem and the possibility that the convergence point may stack on a local rather than on a global optimum [8] [10] [11].

2.3. Fuzzy min max clustering

In 1993 Simpson introduces the neural algorithm known as fuzzy min max clustering *FMMC* [3]; it is a neural network based algorithm with evolving architecture. It is based on unsupervised learning rules which were initially developed by the same author in 1992 for supervised classification based on a neural fuzzy min max approach [2]. *FMMC* contains three layers, input, output and hidden layers. The number of neurons in the input layer is equal to the dimension of the data representation space; it is the space R^N where N is the size of the attribute vector giving the features of the data. The numbers of neurons in the hidden and output layers increase in time with respect to the creating of prototypes for the hidden layer and creating of classes for the output layer. The synaptic weights associated with the connections between input and hidden layers are formed by two matrices V and W giving the characteristics of the different prototypes in the hidden layer. The synaptic weights associated with the connection between hidden and output layers are formed by a Z matrix characterizing the association of prototypes to the classes. The learning process is made in three steps, expansion overlapping and contraction.

The main objective of this algorithm is to characterize the real classes by a set of fuzzy hyper-cube prototypes, without prior knowledge on the classes. Each prototype P_j is defined by a couple of points (V_j, W_j), where $V_j = (v_{jn})_{1\le n\le N}$ and $W_j = (w_{jn})_{1\le n\le N}$, corresponding respectively to the prototypes influence zone min max. Two parameters need to be fixed before starting the algorithm, the sensitivity γ and the vigilance factor θ of the hyper-cube. *FMMC* algorithm steps are shown in figure 1:

Network initialization
- 1- Initial values for γ and θ.
- 2- The first data observation O_1 of to the network.

Repeat:

For each new observation data O_i,

1- ***Expansion of a hyper-cube***: Identify the hyper-cube P_j for what O_i has the maximum degree of membership $b_j(O_i)$.

$$b_j(O_i) = \frac{1}{N}\sum_{n=1}^{N}[1 - f(a_{in} - w_{jn}) - f(v_{jn} - a_{in})]$$

$f(.)$ is the activation function

$$f(X) = \begin{cases} 1 & if \quad X\gamma \succ 1 \\ X\gamma & if \quad 0 \prec X\gamma \le 1 \\ 0 & if \quad X\gamma \prec 0 \end{cases}$$

2- Expand the prototype P_j.

3- Compute the size T of P_j by means of:

$$v_{jn}^{*} = \min(v_{jn}, a_{in})$$
$$w_{jn}^{*} = \max(w_{jn}, a_{in})$$

If $T \le \theta$ then

$P_j^* = P_j$

Overlap test: Search for overlap between existing prototypes

Contraction: Remove these overlaps by contracting prototypes

Else Create a new prototype

End

Until Stabilization of prototypes

Compute Z matrix

Compute the membership functions of the classes.

Fig 1: *FMMC* algorithm.

The *FMMC* algorithm suffers however from the quality of the adopted model [12]. The hyper cube mode is a loose approximation of the data distribution. Tuning with γ and θ parameters does not solve this problem. Big value for γ gives rise to an over learning while taking θ as a small value creates a lot of prototypes.

3. PROPOSED METHOD

The proposed method is shown in figure 2:

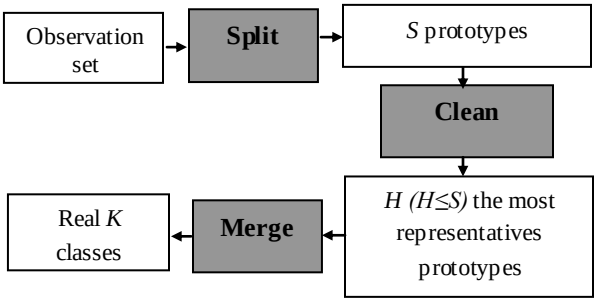


Fig 2: Proposed method.

The objective is to characterize each class by a set of well representative prototypes in an unsupervised way. Each prototype is represented by a set of data points defining a cloud of loosely a spherical form. The prototype is characterized by the gravity center of all the data points it contains. Three threshold points must be set out initially in order to obtain the prototypes and the classes. These are S_{Split}, S_{Clean}, and S_{Merge}, the three thresholds for the operations Split, Clean and Merge act as decision parameters respectively. A

data observation O_i is considered to belong to a prototype P_j if the distance between O_i and the gravity center of P_j is lower than S_{Split}. Setting S_{Split} to a correct value is not an easy task. It is an application dependant problem. We have set it here, in this work, to a small value in the hope to divide the data into subclasses (prototypes) of very similar points. This is at the expense of obtaining a lot of prototypes. S_{Clean} is a threshold for clean operation. Its task is to remove the bad prototypes. A prototype is considered, in this work to be bad if it contains a reduced number of data. In this case we considered those prototypes with cardinal values less than S_{Clean} as candidates to be removed. S_{Merge} is a threshold parameter designated to merge similar prototypes. We considered the distance between two prototypes to be the minimum distance between all the data points in the two prototypes.

$$d(P_i, P_j)=min(\{d(x, y), x \in P_i, y \in P_j\})$$

Two prototypes P_i and P_j are merging into one subclass if the distance between P_i and P_j is lower than S_{Merge}. We have set S_{Merge} to a small value in order to only merge those prototypes which are very similar.

3.1. Split algorithm

The Split phase is accomplished through a neural network with evolving architecture, figure 3. The network contains two layers. An input layer with N neurons representing the realization of the object O_i, $R_i=(a_{i1},\ldots, a_{in},\ldots, a_{iN})$ where a_{in} are the attribute elements. An output layer having a number of neurons defining the exiting prototypes in a giving instant of time. Each prototype P_j is characterized by a neuron in the output layer. The size of this layer is not fixed initially, it increases with time as long as new prototypes are created when new observation data points is applied to the network. The output layer is fully connected to the input layer. The synaptic weight associated with the connections W corresponds to the gravity centers of the existing S prototypes.

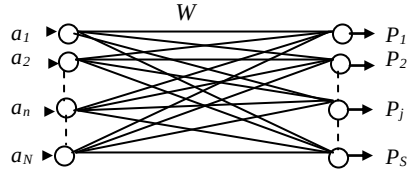


Fig 3: Split neural network architecture.

Mode of operation of the Split process.

Initially at $t=0$, observation O_1 is applied to the network. Output layer contains one neuron and the matrix of synaptic weights M_w is:

$$M_W = W_1 = [w_{11}\, w_{12}\ldots\, w_{1n}\, \ldots..\, w_{1N}]$$
$$= [R_1] = [a_{11}\, a_{12}\ldots a_{1n}\ldots\, a_{1N}]$$

R_1 is the realization of the data point O_1, figure 4. P_1 is formed by O_1.

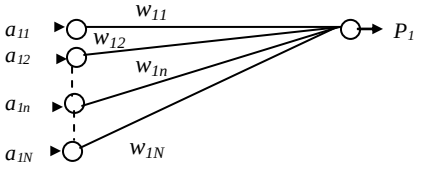


Fig 4: Configuration of the network associated with Split phase at t=0

At iterative t=1, the distance between R_2 and W_1 is computed, R_2 is the realization of O_2, O_2 is the second data point.

- If $d(R_2,W_1) \leq S_{Split}$, then O_2 is attributed to P_1 and the synaptic weight vector W_1 becomes :

$$W_1 = \frac{\sum_{R_i \in P_1} R_i}{card(P_1)} = \frac{R_1 + R_2}{2}$$

- If $d(R_2,W_1) > S_{Split}$, then O_2 is not in P_1, A new prototype P_2 and a new neuron are created (figure 5). The new prototype is characterized by the attribute vector associated with O_2 and the synaptic weights matrix becomes

$$M_W = \begin{bmatrix} M_W^{t=0} \\ W_2 \end{bmatrix} = \begin{bmatrix} W_1 \\ R_2 \end{bmatrix}$$

Where: $M_W^{t=0}$ the weight matrix at t=0.

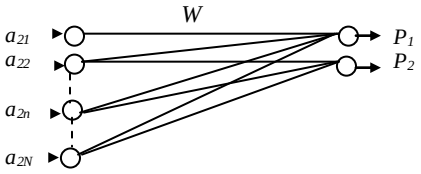


Fig 5: Creation of a new prototype

Proceeding with the algorithm, we consider that at iteration $t=k-1$, we obtain S prototypes associated to S neurons in the network output layer. We pass on to iteration $t=k$, object O_k is applied to the network's input layer through its realization R_k, the distance between R_k and the various W_j are computed, *for* $j=1$ to S, let W_r be the closest to R_k,

$$d(R_k,W_r) = \min_{j=1}^{S}(d(R_k,W_j))$$

- If $d(R_k,W_r) \leq S_{Split}$ then O_k is attributed to P_r and the synaptic weight vector associated to neuron P_r becomes:

$$W_r = \frac{\sum_{R_i \in P_r} R_i}{card(P_r)}$$

The synaptic weights associated with the other neurons $j \neq r$, remain unaffected. .

- If $d(R_k,W_r) > S_{Split}$, then O_k is far away from all the existing prototype at $t=k-1$. In this case a new prototype P_{S+1} and a new neuron are created in the network's output layer. P_{S+1} is characterized by R_k and the weight matrix is:

$$M_W = \begin{bmatrix} M_W^{t=k-1} \\ W_{S+1} \end{bmatrix} = \begin{bmatrix} W_1 \\ \vdots \\ W_S \\ R_k \end{bmatrix}$$

3.2. The clean operation

Finishing off the Split phase, many prototypes sometimes by hundreds are produced. Among the obtained prototypes in the split phase, many of them are non representatives of the real classes and must simply be removed without causing any harm to the initially pointed out objective. Indeed the presence

of these non representative prototypes does not improve the discrimination between the classes and the information they add for representing the classes is negligible if compared to the noise they generate.

In order to avoid this problem, we opted for a clean operation where we delete the non representative prototypes. In this work, the cardinality of the prototype is used as a criterion for removing or keeping the prototype. If the cardinal of a prototype is less than S_{Clean} then the prototype is considered be non representative and is removed. S_{Clean} is a threshold parameter which is set out initially and is taken as a small value after an error and trial procedure.

3.3. The Merge algorithm

The split procedure described previously obtains a high number of prototypes. The main reason for this is that the split algorithm divides the observations in an arbitrary way. The solution is to merge similar prototypes into one. The similarity between prototypes is defined by a similarity criterion based on a distance measure [11]. Merge algorithm is a neural network with evolving architecture figure 6. The network is formed by three layers.

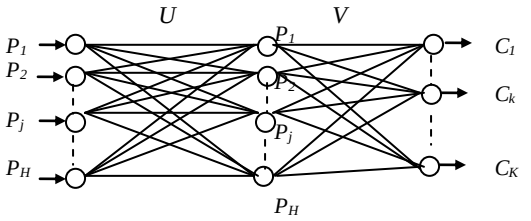


Fig 6: Merge neural network architecture.

The input layer is made of H neurons, where H is the number of prototypes found after the clean operation. The hidden layer is also formed by H neurons corresponding to the H prototypes, the input and hidden layers are fully connected by means of a binary matrix $U=(u_{ij})_{(1\leq i\leq H, 1\leq j\leq H)}$, where u_{ij} define the connexity between the prototypes P_i and P_j. The distance between two prototypes P_i and P_j is defined by [4]:

$$d(P_i, P_j)=min(\{d(x, y), x \in P_i, y \in P_j\})$$

- If $J(P_i, P_j)\leq S_{Merge}$ then P_i is considered to be similar to P_j and $u_{ij}=1$.
- If $J(P_i, P_j)>S_{Merge}$ the two prototypes are considered to be not similar and $u_{ij}=0$.

The output layer contains a number of neurons defining the real existing classes. Each class C_k is characterized by a neuron on the output layer. The number of neurons on the output is not fixed initially, the algorithm starts with one neuron, and the number of neurons increases with time in correspondence with the creation of classes where similar prototypes are regrouped. Hidden and output layers are fully connected by means of a binary matrix $V=(v_{jk})_{(1\leq j\leq H, 1\leq k\leq K)}$,. The prototype P_j is considered to be a member of the class C_k if $v_{jk}=1$, if not, $v_{jk}=0$. $v_{jk}=1$ in two cases, the first case is when the algorithm chooses to open a new class C_k because P_j is not connex to any of the previous $k-1$ classes, the second case is when C_k is already existing and P_j is connex to one of the elements of C_k.

Mode of operation of the Merge process.

At iteration $t=0$, the architecture of the merge network is that of figure 7, one neuron at the output layer corresponding to a real class C_1. C_1 contains P_1 and all the prototypes which are directly or indirectly connex to P_1. Two prototypes P_a and P_b are indirectly connex if there exists a chain of prototypes related by a connexity relationship where the two prototypes P_a and P_b are present. For example, consider the chain:

$$P_1 \longrightarrow P_2 \longrightarrow P_3 \longrightarrow P_4$$

P_1 is directly connex to P_2 but it is indirectly connex to P_3 and P_4.

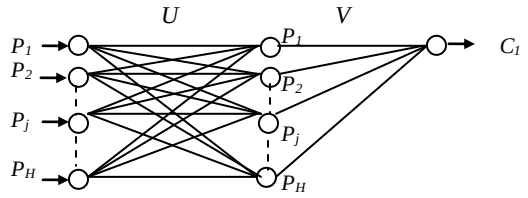


Fig 7: Configuration of the network associated with Merge phase at $t=0$

At iteration $t=k-1$, the output layer contains $k-1$ neurons corresponding to $k-1$ real classes.

At iteration $t=k$, the algorithm looks for the first prototype not yet classified, the search is looked at in an increasing way from P_1, upwards until P_j where P_j does not belong to neither of the $k-1$ already set classes. This means that $(v_{jc})_{(1\leq c\leq k-1)}$ are all equal to zero. A k^{th} neuron is then added to the output layer characterizing the class C_k and containing P_j and those prototypes which are not yet classified and which are directly or indirectly connex with P_j. This process is carried out until all the prototypes are attributed to their specific classes.

4. EXPERIMENTAL RESULTS

Two experiments are carried out in this section, one considers classes of convex type and the classes are non convex for the second one. The data for both experiments are represented in a two dimensional space.

4.1 First experiment: Classes of convex form.

In this experiment, two types of simulation are carried out. For both simulations the classes have convex form, we have three classes in each case, and the data have been generated by a Gaussian distribution routine through Matlab. The difference between the two simulations can be seen in the number of data, the level of overlapping between the classes and the noisy data introduced in the second simulation. For both simulations we set $S_{Split}=0.8$, $S_{Clean}=10$ and $S_{Merge}=0.2$

4.1.1 Simulation 1

The data to be classified for this simulation are shown in figure 8. The overlapping between the classes in this case is null, the noisy data are not there, each class contains 150 data.

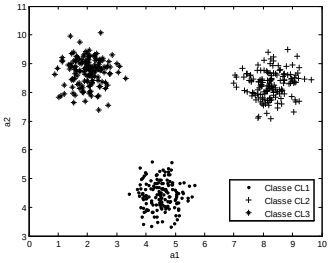


Fig 8: Repartition of the classes in the observation space.

The Split phase runes quickly and obtains 14 prototypes. In the clean process three prototypes are removed. The 11

remaining and most representative prototypes are gathered in the set $E_S = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}\}$. Table 1 and figure 9 show the results obtained.

Table 1. Obtained results

Prototypes	Gravity center		Cardinal
P_1	4.7749	4.0530	49
P_2	4.5089	4.8449	68
P_3	3.9775	4.1385	30
P_4	8.1658	8.1620	72
P_5	8.4336	8.9454	23
P_6	8.9692	8.2134	26
P_7	7.3966	8.4909	15
P_8	8.0904	7.5075	14
P_9	2.4272	8.8034	53
P_{10}	1.7079	8.9593	60
P_{11}	1.9110	8.0437	33

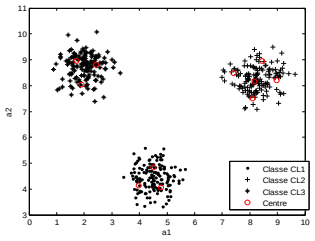


Fig 9: Prototypes with corresponding centers obtained by the proposed method.

The set E_s is applied to the Merge algorithm. The synaptic weights between the hidden and output layers obtained at convergence are

$$V = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Convergence is achieved when the output layer contains three neurons. Each neuron represents a real class which is formed by several prototypes. The number of classes obtained is then 3 which coincides with the true one. The repartition of prototypes in the classes is:

- $CL_1 = \{P_1, P_2, P_3\}$
- $CL_2 = \{P_4, P_5, P_6, P_7, P_8\}$
- $CL_3 = \{P_9, P_{10}, P_{11}\}$

4.1.2 Simulation 2

The second simulation is illustrated by figure 10. There is some overlapping between the classes and many noisy data in each class are there. Each class are there contains 250 observations.

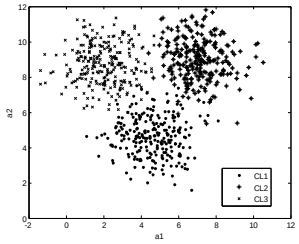


Fig 10: Repartition of the classes in the observation space.

The split process runs quickly and obtains 70 prototypes. The clean operation removes 38 prototypes and 32 prototypes remain as the most representatives of the real classes.

The final stage is the merge process which obtains at convergence a synaptic weight matrix between hidden and output layers given by:

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The algorithm obtains 3 neurons for the output layer; we then obtain three real classes which coincide with the true number of classes. The repartition of prototypes between the classes is:

- $CL_1 = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}\}$
- $CL_2 = \{P_{12}, P_{13}, P_{25}, P_{26}, P_{27}, P_{28}, P_{29}, P_{30}, P_{31}, P_{32}\}$
- $CL_3 = \{P_{14}, P_{15}, P_{16}, P_{17}, P_{18}, P_{19}, P_{20}, P_{21}, P_{22}, P_{23}, P_{24}\}$

Figure 11 shows graphically the obtained results for data classification.

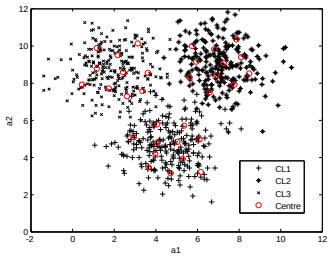


Fig 11: Prototypes and their corresponding centers obtained by the proposed method.

4.1.2.1 Comparison with the FCM

The aim is to compare the proposed method with the *FCM*. The comparison is conducted through simulation 2 case study.

Unsupervised *FCM* classification does not know what is the number of classes initially. Several criteria have been proposed in the literature for optimally selecting this number. We have chosen the Xie and Beni criterion which make use of compacity and separability notions [10]. The *FCM* algorithm is run for several values of C (number of classes) and the Xie and Beni criterion value F_{XB} is computed for each value chosen for C. figure 12 shows the evolution of F_{XB} with respect to the number of classes.

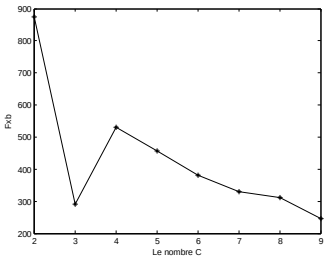


Fig 12: Evolution of F_{XB} with respect to the number of classes.

The figure shows that the optimal number is $C_{opt}=9$, which does not coincide with the real number which is 3, this shows that when the degree of overlapping between the classes is high the Xie and Beni criterion may not obtain the correct number of classes. Our proposed method without initially

knowing the number of classes has converged to the optimal solution.

We take *C=3* for *FCM* and carry out a comparison between the two methods. The results obtained are summarized in table 2.

Table 2: Classification results

	number of misclassified observations	Classifcation rate
FCM (C=3)	24	96.8%
Proposed method	18	97.6%

A higher classification rate is obtained by the proposed method. 18 observations were misclassified by the proposed method from among the initial 750 ones, and a classification rate of 97.6%, the *CMF* has obtained 24 observations misclassified and a classification rate of 96.8%.

4.1.2.2 Comparison with fuzzy Min Max clustering FMMC

We have compared the proposed method with *FMMC* through simulation 2. Figure 13 shows the classification results obtained for *FMMC* method. Table 3 shows the performances of the two methods.

Table 3. Comparison between the two methods.

	Number of classes	Learning time	Number of prototypes
FMMC	1	22,6 s	278
Proposed method	3	1,2 s	32

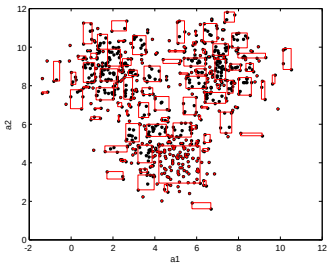
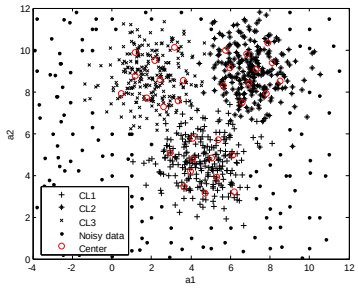


Fig 13: Prototypes obtained by the *FMMC* method.

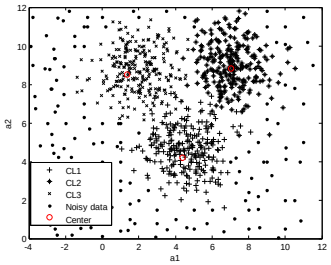
The number of classes obtained by *FMMC* does not coincide with the true one; this is due to the high overlapping between the classes where prototypes are created in the overlapping zone which mix all the classes to one. For the proposed method the clean operation has acted in such a way that the classes have remained separated. We also noticed that the running time of *FMMC* method is largely greater than that for the proposed method.

4.1.2.3 Case with noisy data

We are interested to study the behaviors of the proposed method when there one a lot of noisy data embedded in the set of initial data to be classified, and when the degree of overlapping between the classes is high. We have used the data of simulation 2 to which we have added 150 noisy observations. Figure 14 and table 4 show the results obtained by the proposed and *FCM* methods.



(a)



(b)

Fig 14: (a) Prototypes obtained by the proposed method. (b) Prototypes obtained by *FCM*.

Table 4: Results of data classification

	number of misclassified observations	Classifcation rate
FCM (C=3)	30	96%
Proposed method	18	97.6%

The classification for *FCM* has slightly decreased due to the noisy data effect, while it remains the same as in 4.1.2.1 for the proposed method.

4.2 Second experiment: Classes of non convex or complex form.

We have envisaged for this experimentation two simulations, each of them considers two classes of non convex form. The form in simulation 3 is rectangular while it is circular for simulation 4. The observation space for the two simulations is of dimension 2. The data have been synthesized by authors and this has made use of the rand routine in Matlab. For both simulations we set $S_{Split}=0.6$, $S_{Clean}=10$ *and* $S_{Merge}=0.2$

4.2.1 Simulation 3

Simulation 3 uses 1200 observations distributed into two classes of rectangular non convex form as shown in figure 15. Each class contains 600 observations.

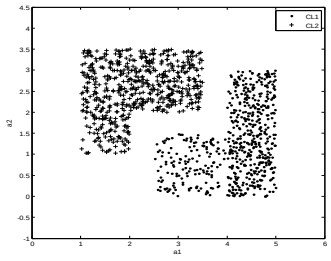


Fig 15: Distribution of data in the observation space.

Split algorithm runs quickly and obtains 22 prototypes. The clean operation removes one of them. The 21 remaining

prototypes are applied to the merge algorithm and the V matrix obtained is:

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

The merge algorithm obtains two neurons in its output layer and the repartition of the prototypes obtained for each neuron is:

- $CL_1 = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_{20}, P_{21}\}$
- $CL_2 = \{P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}, P_{16}, P_{17}, P_{18}, P_{19}\}$

Figure 16 shows the obtained results.

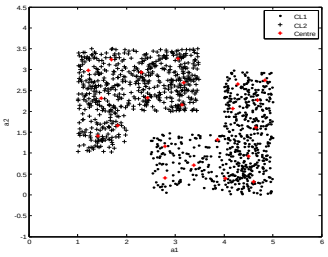
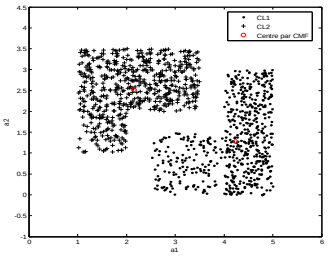
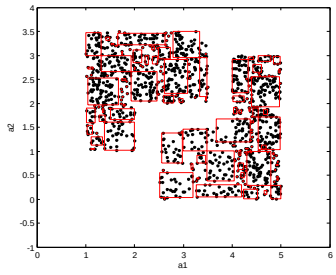


Fig 16: Prototypes and their corresponding centers obtained by the proposed method.

The results obtained by the proposed method are compared to those obtained by *FCM* and *FMMC*, figure 17 and table 6.



(a)



(b)

Fig 17: (a) Prototypes obtained by the *FCM*. (b) Prototypes obtained by *FMMC*.

Table 6 : Classification results

	Number of classes	Learning time (s)	Number of prototypes	Classification rate
FCM (C=2)	2	3,34	2	98.5%
FMMC	2	23,6	229	100%
Proposed method	2	1,07	21	100%

Both *FMMC* and the proposed method have obtained better results than *FCM*, but the running time for *FCM* and *FMMC* are much bigger than that required by the proposed method.

4.2.2 Simulation 4

In this simulation, we have used 1050 observations distributed into 2 classes of circular non convex form (figure 18). The first class contains 100 observations and the second one contains 950 remaining observations.

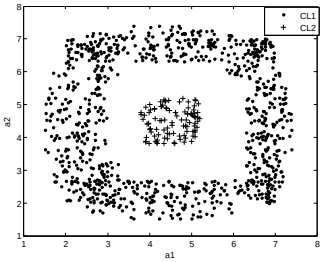


Fig 18: Distribution of data in the observation space.

Split algorithm runs quickly and obtains 17 prototypes. The clean operation does not remove any of the prototypes and the V matrix obtained is:

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

The merge algorithm obtains two neurons in its output layer and the repartition of the prototypes obtained for each neuron is:

- $CL_1 = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}\}$
- $CL_2 = \{P_{16}, P_{17}\}$

Figure 19 shows the obtained results.

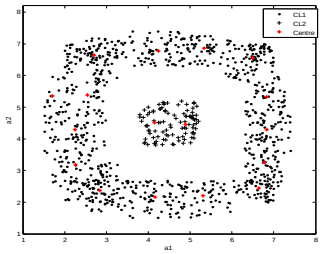
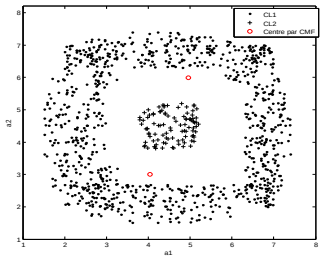
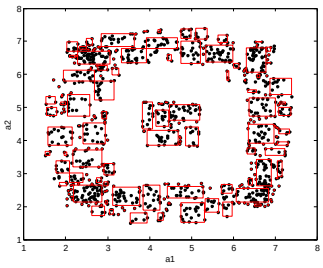


Fig 19: Prototypes and their corresponding centers obtained by the proposed method.

The results obtained by the proposed method are compared to those obtained by *FCM* and *FMMC*, figure 20 and table 7.



(a)



(b)

Fig 20: (a) Prototypes obtained by the *FCM*. (b) Prototypes obtained by *FMMC*.

Table 7 : Classification results

	Number of classes	Learning time (s)	Number of prototypes	Classification rate
FCM	2	2 ,77	2	40.7%
FMMC	2	17,66	209	100%
Proposed method	2	0,73	17	100%

FCM has failed in this case because the center of classes obtained by *FCM* is not good. *FMMC* and proposed method have both converged to the true solution but the running time of the *FMMC* is largely high than that required for the proposed method.

5. CONCLUSION

In this work, we have proposed a new method of unsupervised data classification based on a cascade system containing three blocks: split, clean and merge processes. The split process divides the initial set of data into several subclasses; it is based on a neural network containing two layers which builds the subclasses in a progressive way. The clean operation removes the non representative subclasses which contain noisy data; the cardinal value of these subclasses is small. Merge process regroups similar subclasses into a real class according to a chosen similarity criterion. The proposed method have been validated through many examples studied by simulations, the form of the classes chosen for these simulations is convex for some of them and non convex for the others. The results obtained are good; the method has always converged to the optimal solution in a reduced running time. The simulation results have shown that although the degree of overlapping between the classes is high and although the noisy data are present in the initial set of data the method proposed still converge to the expected result. The method does not infer any initial knowledge on the classes especially the number of classes is not initially set by the method. This number is obtained after that the method converges. The simulation results have also shown a net improvement obtained by the proposed method when it is compared to the *FCM* and *FMMC* methods of classification.

6. REFERENCES

[1] P. Jiang, F. Ren and N. Zheng. 2009. A new approach to data clustering using a computational visual attention model. *International Journal of Innovative Computing, Information and Control* Volume 5, Number 12(A). (December 2009)

[2] P.K. Simpson. 1992. Fuzzy min-max neural networks. Part 1: classification. IEEE Transactions on Neural Networks, vol. 3, pp: 776-786.

[3] P.K. Simpson. 1993. Fuzzy min-max neural networks. Part 2: clustering. IEEE Transactions on Fuzzy Systems, vol. 1(1), pp: 32-45.

[4] M. Boubou. 2007. Contribution aux méthodes de classification non supervisée via des approches prétopologiques et d'agrégation d'opinions. Thése de Doctorat, Université Claude Bernard – Lyon I.

[5] H. Ouariachi. 2001. Classification non supervisée de données par réseaux de neurones et une approche évolutionniste: application à la classification d'images. Thèse de doctorat, Université Mohamed 1, Maroc

[6] M. Nasri, M. El Hitmy, H. Ouariachi and M. Barboucha. 2003. Optimization of a fuzzy classification by evolutionary strategies''. In proceedings of SPIE Conf., 6th international Conference on Quality Control by Artificial Vision, Vol. 5132, pp. 220-230, USA, 2003. Repulished as an SME Technical paper by the Society of manufacturing engineers (SME), Paper number MV03-233, ID TP03PUB135, Dearborn, Michigan, USA, pp. 1-11, (24 June 2003).

[7] L. Khodja. 1998. Contribution à la classification floue non supervisée. Thése de doctorat, université de savoie.

[8] K. R. Zalik and B. Zalik. 2010. Validity index for clusters of different sizes and densities. Pattern Recognition Letters, 221-234, (18 September 2010).

[9] C. Duo, L. Xue and C. Du-Wu. 2007. An Adaptive Cluster Validity Index For The Fuzzy C-means. IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.2, (February 2007).

[10] M. Nasri. 2004. Contribution à la classification des données par approches evolutionnistes : simulation et application aux images de textures. Thèse de Doctorat, Université Mohammed premier, Faculté des sciences Oujda, Maroc.

[11] M. S. Bouguelid. 2007. Contribution à l'application de la reconnaissance des formes et la théorie des possibilités au diagnostic adaptatif et prédictif des systèmes dynamiques. Thése de doctorat, Université de Reins Champagne- Ardenne.

[12] H. A. Boubacar. 2006. Classification dynamique de données non-stationnaires apprentissage et suivi de classes evolutives. Thése de doctorat, université des sciences et technologies de lille.

[13] O. Aya, M. Saued-Mouchaweh et P. Billaudel. 2010. Sélection dynamique des classifieurs pour l'amélioration du taux de classification dans les zones d'ambiguïtés ''. 6éme conférences internationale Francophone d'Automatique (CIFA 2010).