Active Learning Using a Constructive Neural Network Algorithm

José L. Subirats, Leonardo Franco, Ignacio Molina, and José M. Jerez

Abstract. Constructive neural network algorithms suffer severely from overfitting noisy datasets as, in general, they learn the set of available examples until zero error is achieved. We introduce in this work a method for detect and filter noisy examples using a recently proposed constructive neural network algorithm. The new method works by exploiting the fact that noisy examples are in general harder to be learnt than normal examples, needing a larger number of synaptic weight modifications. Different tests are carried out, both with controlled and real benchmark datasets, showing the effectiveness of the approach. Using different classification algorithms, it is observed an improved generalization ability in most cases when the filtered dataset is used instead of the original one.

1 Introduction

A main issue at the time of implementing feed-forward neural networks in classification or prediction problems is the selection of an adequate architecture [1, 2, 3].

José L. Subirats
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga,
Campus de Teatinos S/N, 29071 Málaga, Spain
e-mail: `jlsubirats@lcc.uma.es`

Leonardo Franco
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga,
Campus de Teatinos S/N, 29071 Málaga, Spain
e-mail: `lfranco@lcc.uma.es`

Ignacio Molina
Departamento de Tecnología Electrónica,, Universidad de Málaga,
Campus de Teatinos S/N, 29071 Málaga, Spain
e-mail: `aimc@dte.uma.es`

José M. Jerez
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga,
Campus de Teatinos S/N, 29071 Málaga, Spain
e-mail: `jja@lcc.uma.es`

Feed-forward neural networks trained by back-propagation have been widely used in several problems but still the standard approach for selecting the number of layers and number of hidden units of the neural architecture is the inefficient trial-by-error method. Several constructive methods and pruning techniques [1] have been proposed as an alternative for the architecture selection process but it is a research issue whether these methods can achieve the same level of prediction accuracy. Constructive algorithms start with a very small network, normally comprising a single neuron, and work by adding extra units until some convergence condition is met [4, 5, 6, 7, 8]. A totally opposite approach is the used by pruning techniques, as these methods start with large architectures and work by eliminating unnecessary weights and units [9].

Despite the existence of several constructive algorithms, they have not been extensively applied in real problems. This fact is relatively surprising, given that they offer a systematic and controlled way of obtaining a neural architecture together with the set of weights, and also because in most cases they offer the possibility of an easier knowledge extraction procedure. In a 1993 work, Smieja [10] argued that constructive algorithms might be more efficient in terms of the learning process but cannot achieve a generalization ability comparable to back-propagation neural networks. Smieja arguments were a bit speculative rather than based on clear results, but nevertheless might explain the fact that constructive methods have not been widely applied to real problems. In recent years new constructive algorithms have been proposed and analyzed, and the present picture might have changed [8, 7].

One of the problems that affects predictive methods in general, is the problem of overfitting [11, 12]. The problem of overfitting arises when an algorithm specializes in excess in learning the available training data causing a reduction on the generalization ability, computed on unseen data. In particular, overfitting affects severely neural network constructive algorithms as they, in general, learn towards zero error on the training set. One of the strategies used in constructive algorithms for avoiding overfitting is the search of very compact architectures, as models with fewer number of parameters may suffer less from overfitting. Other standard methods to avoid overfitting, like early stopping using a validation set or weight decay, can also be applied to constructive methods but they tend to be computationally costly and sometimes difficult to adapt to work in conjunction with some constructive algorithms. When the input data is noisy, as it is normally the case of real data, the simple use of compact architectures is not enough to avoid overfitting as it will be shown later in this chapter. A possible solution to this problem might be the implementation of methods that exclude noisy instances from the training dataset [13, 14, 15, 16, 17, 18], in a process that is usually considered a pre-processing stage. In this work, we refer to the whole problem of learning and filtering noisy examples as "Active learning", as we considered both stages together in an on-line procedure in which noisy instances are eliminated during the learning procedure. Nevertheless, we also show in this work that the new introduced filtering process can be applied as a separate stage and the selected instances used later with any available predictive algorithm. The usual name given to the process of selecting or filtering some examples from the available dataset is "Instance selection" and we refer to [19] for previous work on the field. Instance selection can be also used for reducing the size of the dataset in order to speed up the training process and can be also lead to prototype selection when the selected data are very much reduced. The approach taken in this chapter is to use the proposed instance selection method as a pre-processing step , as way of improving the generalization ability of predictive algorithms. The method is developed inside a recently introduced constructive neural network algorithm named C-Mantec [20] (Competitive MAjority Network Trained by Error Correction) and leads to an improvement in the generalization ability of the algorithm, permitting also to obtain more compact neural network architecures. The reduced, filtered, datasets are also tested with other standard classification methods like standard multilayer perceptrons, decision trees and support vector machines, analyzing the generalization ability obtained. This chapter is organized as follows: Next we give details about the C-Mantec constructive neural network algorithm and in Section 3 the method for eliminating noisy instances is introduced, to follow with some experiments, results and conclusions.

2 The C-Mantec Algorithm

The C-Mantec algorithm is a constructive neural network algorithm that creates architectures with a single layer of hidden nodes with threshold activation functions. For the most general case of input data comprising 2 output classes, the constructed networks have a single output neuron computing the majority function of the responses of the hidden nodes (i.e., if more than half of the hidden neurons are activated the output neuron will be active). The case of multiclass classification will be considered separately below. The learning procedure starts with an architecture comprising a single neuron in the hidden layer and as the learning advances more neurons are added every time the present ones are not able to learn the whole set of training examples. The synaptic weight modification rule used at the single neuron level is the thermal perceptron learning rule proposed by Frean [5, 21]. The thermal perceptron rule is a modification of the standard perceptron rule [22] that incorporates a modulation factor that makes the perceptron to learn only inputs that are similar to the already acquired knowledge, as the introduced factor limits the value of the modifications of the synaptic vector. The idea behind the thermal perceptron is to introduce stability to the standard perceptron for the case of non-linearly separable tasks and this is achieved by permitting large changes of the synaptic vector only at the beginning and later on only allow small modifications.

We consider neurons with a threshold activation function receiving input signals ψ_i through synaptic weights w_i that are active if the synaptic potential, ϕ is larger than zero. The synaptic potential is defined as:

$$\phi = (\sum_i \psi_i * w_i) - b. \tag{1}$$

Note that the definition of the synaptic potential includes the value of the threshold or bias, b, as this will be useful because for wrongly classified inputs the absolute value of the synaptic potential, $|\phi|$, quantifies the error committed as it gives the distance to the bordering hyperplane dividing the classification regions.

The neuron model used is a threshold gate where the output of the neuron, S, is given by a step function depending on the value of the synaptic potential.

$$S = f(p) = \begin{cases} 1 \text{ if } p \geq 0 \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

As said above, the synaptic modification rule that is used by the C-Mantec algorithm is the thermal perceptron rule for which the change of the weights, δw_i, is given by the following equation:

$$\delta w_i = (t - S)\ \psi_i\ \frac{T}{T_0}\ \exp\{-\frac{|\phi|}{T}\}\ , \tag{3}$$

where t is the target value of the example being considered, S represents the actual output of the neuron and ψ is the value of the input unit i. T is a parameter introduced in the thermal perceptron definition, named temperature, T_0 the starting temperature value and ϕ, the synaptic potential defined in Eq. 1. For rightly classified examples, the factor $(t - S)$ is equals to 0 and then no synaptic weight changes take place. The thermal perceptron rule can be seen as a modification to the standard perceptron rule where the change of weights is modified by the factor, m, equals to:

$$m = \frac{T}{T_0}\ \exp\{-\frac{|\phi|}{T}\}\ . \tag{4}$$

At the single neuron level the C-Mantec algorithm uses the thermal perceptron rule, but at a global network level the C-Mantec algorithm incorporates competition between the neurons, making the learning procedure more efficient and permitting to obtain more compact architectures [20]. The main novelty introduced in the new C-Mantec algorithm is the fact that once a new neuron is added to the network, the existing synaptic weights are not frozen, as it is the standard procedure in constructive algorithms. Instead, after an input instance is presented to the network all existing neurons can learn the incoming information by modifying its weights in a competitive way, in which only one neuron will learn the incoming information. The norm in standard constructive algorithms is to freeze weights not connected to the last added neurons in order to preserve the stored information, in the C-Mantec algorithm this is not necessary as the thermal perceptron is a quite conservative learning algorithm and also because the C-Mantec algorithm incorporates a parameter $gfac$ that further controls the size of the allowed changes in synaptic weights, in particular when the Temperature is large when this large changes are allowed at the single neuron level by the thermal perceptron.

The C-Mantec algorithm generates architectures with a single hidden layer of neurons. The output neuron of the network computes the majority function of the activation values of the hidden units and thus the set of weights connecting the hidden neurons with the output are fix from the beginning and not modified during

```
1. Start a network with one hidden neuron and one output
neuron.
2. Input a random training example and check the output of
the network.
3. If the input example is not rightly classified then:
  3a. Compute the value of $\phi$ for all existing hidden neurons
that wrongly classify the input example.
  3b. Modify the weights of the neuron with the smallest
value of $\phi$, provided that the value of the factor $m$ is larger
than the value of the parameter $gfac$. Lower the internal
temperature of the modified neuron.
  3c. If there is no neuron with a value of $m$ larger than $gfac$
then introduce a new neuron that learns the incoming example.
4. Go to instruction 2 until all examples are classified
correctly.
```

Fig. 1 Pseudocode of the C-Mantec algortihm

the learning procedure. As the output neuron computes the majority of the hidden layer activations, a correct functioning of the network is a state in which for every instance in the training set the output of more than half of the hidden units coincides with the respective target value of the instances.

As mentioned before, the algorithm also incorporates a parameter named growing factor, *gfac*, as it adjustment affects the size of the resulting architecture. Once an instance is presented and the output of the network does not coincide with the target value, a neuron in the hidden layer will be selected to learn it if some conditions are met. The selected neuron will be the one with the lowest value of ϕ among those neurons whose output is different from the target one, but only if the value of m (see Eq. 4) is larger than the *gfac* value, set at the beginning of the learning process. Thus, the *gfac* parameter will prevent the learning of misclassified examples that will involve large weight modifications, as for high values of T the thermal perceptron rule would not avoid these large changes, that can cause instability to the algorithm. After a neuron modifies it weights, its internal temperature value is lowered. In the case in which for a wrongly classified instance there are no neurons available for learning, a new neuron is added to the network and this unit will learn the current input, ensuring the convergence of the algorithm. After a new unit is added to the network the temperature, T, of all neurons is reset to the initial value T_0 and the learning process continues until all training examples are correctly classified. In Fig. 1 a pseudocode of the algorithm is shown, summarizing the most important steps of the C-Mantec algorithm and in Fig. 2 a flow diagram of the algorithm is shown.

Regarding the setting of the two parameters of the algorithm, T_0 and *gfac*, several experiments have shown that the C-Mantec algorithm is quite robust against changes of these two parameters and the finding of some optimal values is not difficult. The parameter T_0 (initial temperature) ensures that a certain number of learning iterations will take place, permitting an initial phase of global exploration for the weights values, as for high temperature values larger changes are easier to be

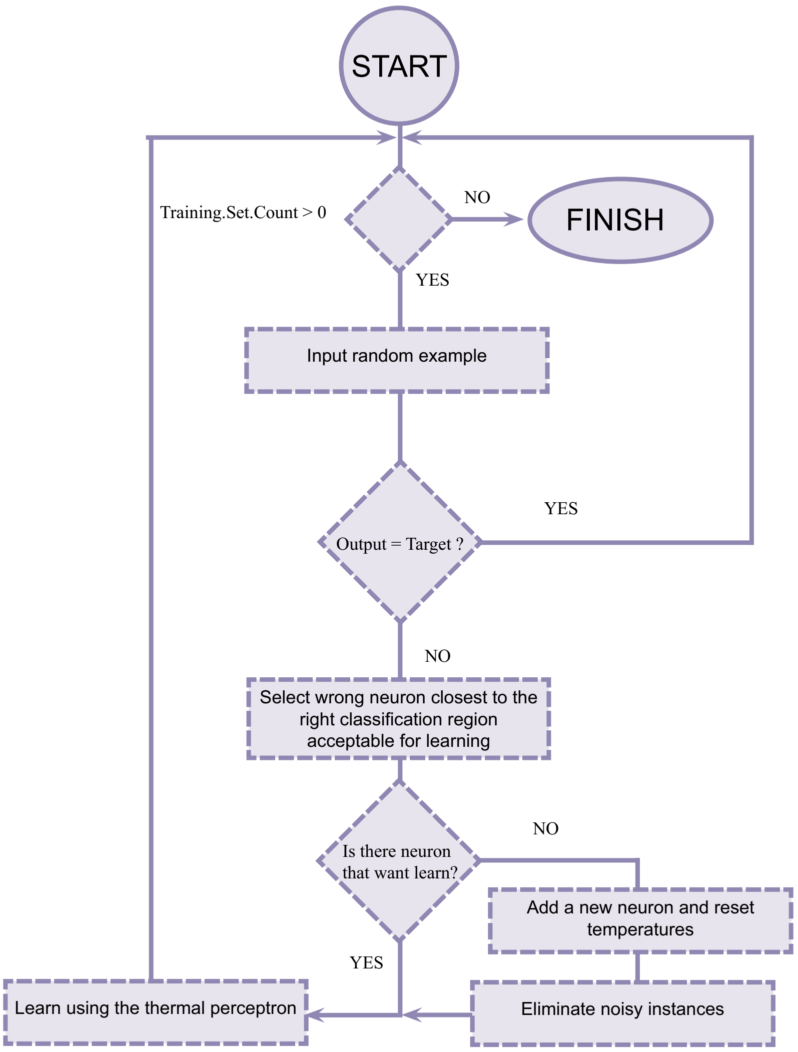

START
Training.Set.Count > 0
NO
FINISH
YES
Input random example
YES
Output = Target ?
NO
Select wrong neuron closest to the right classification region acceptable for learning
NO
Is there neuron that want learn?
Add a new neuron and reset temperatures
YES
Learn using the thermal perceptron
Eliminate noisy instances

Fig. 2 Flow diagram corresponding to the C-Mantec constructive algorithm.

accepted. The value of the parameter $gfac$ affects the size of the final architecture, and it has been observed that different values are needed in order to optimize the algorithm towards obtaining more compact architectures or a network with a better generalization ability.

The convergence of the algorithm is ensured because the learning rule is very conservative in their changes, preserving the acquired knowledge of the neurons and given by the fact that new introduced units learn at least one input example. Tests performed with noise-free Boolean functions using the C-Mantec algorithm show that it generates very compact architectures with less number of neurons than existing constructive algorithms [20]. However, when the algorithm was tested on real datasets, it was observed that a larger number of neurons was needed because the algorithm overfit noisy examples. To avoid this overfitting problem the method introduced in the next section is developed in this work .

3 The "Resonance Effect" for Detecting Noisy Examples

We introduce in this section a method designed to eliminate instances considered noisy, as a way to increase the classification ability of predictive algorithms. It is worth mentioning that deciding whether an input example is a true input datum or a noise-contaminated one is a difficult issue that can in principle be carried out only if one knows a priori the level of noise present in the system. However, a reasonable approach is to discard suspicious noisy inputs and test the generalization ability obtained, without making claims about whether the eliminated instances are noise or not. The filtering method to be introduced is developed from an effect observed during the application of the C-Mantec algorithm to real datasets. The effect, named "resonance effect" can be exemplified by the picture displayed in Fig. 3, where an schematic drawing shows the effect that is produced when a thermal perceptron tries to learn a set of instances containing a contradictory pair of examples. In Fig. 3, the set of "good" examples is depicted in the left part of the figure, while the contradictory pair is on the right. When a single neuron tries to learn this set, the algorithm will find an hyperplane from a beam of the possible ones (indicated in the figure) that classifies correctly the whole set except for one of the noisy examples. Further

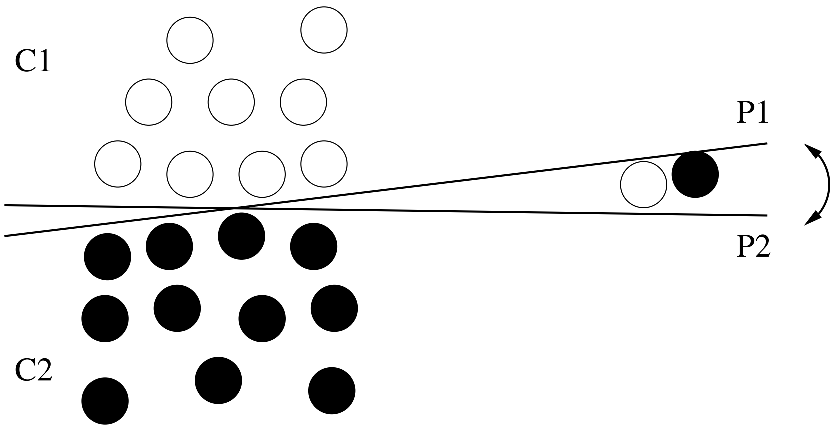
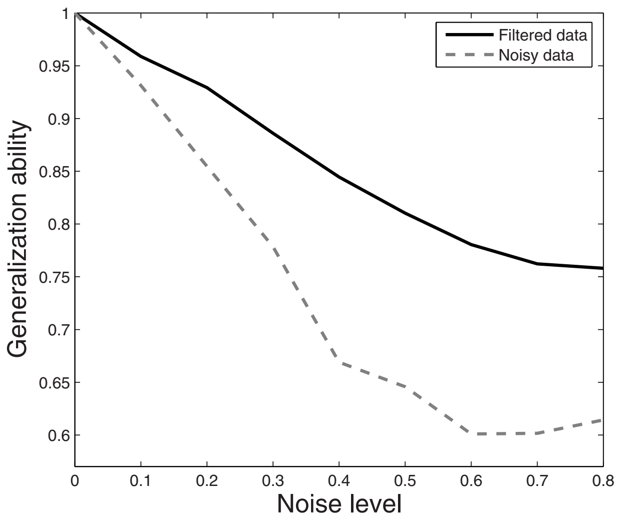


Fig. 3 Schematic drawing of the "Resonance effect" that occurs when noisy examples are present in the training set. A thermal perceptron will learn the "good" examples, represented at the left of the figure, but will classify rightly only one of the noisy samples. Further learning iterations in which the neuron tries to learn the wrongly classified example will produce an oscillation of the separating hyperplane. The number of times the synaptic weights are adjusted upon presentation of an example can be used to detect noisy inputs.



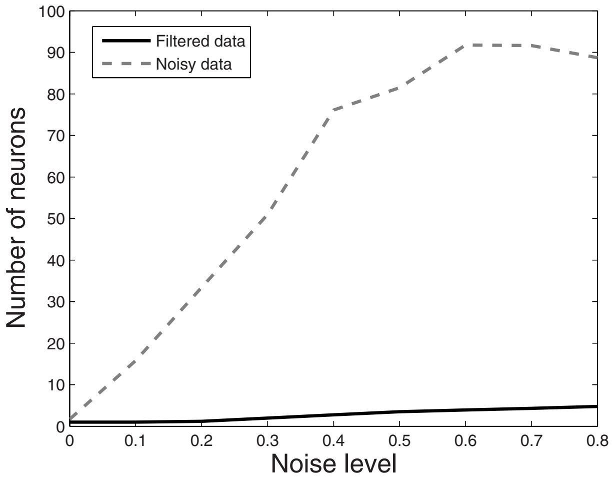


Fig. 4 The effect of adding attribute noise. Top: Generalization ability as a function of the level of attribute noise for the "modified" Pima indians diabetes dataset for the C-Mantec algorithm applied with and without the filtering stage. Bottom: The number of neurons of the generated architectures as a function of the level of noise. The maximum number of neurons was set to 101.

learning iterations produce a resonant behavior, as the dividing hyperplane oscillates trying to classify correctly the wrong example. Eventually, the iterations will end as the whole set cannot be learnt by a simple perceptron and a new neuron will be added to the network. It was observed that these noisy examples make the network to grow in excess, degrading the generalization ability. The filtering method works by counting the number of times each training example is presented to the network,

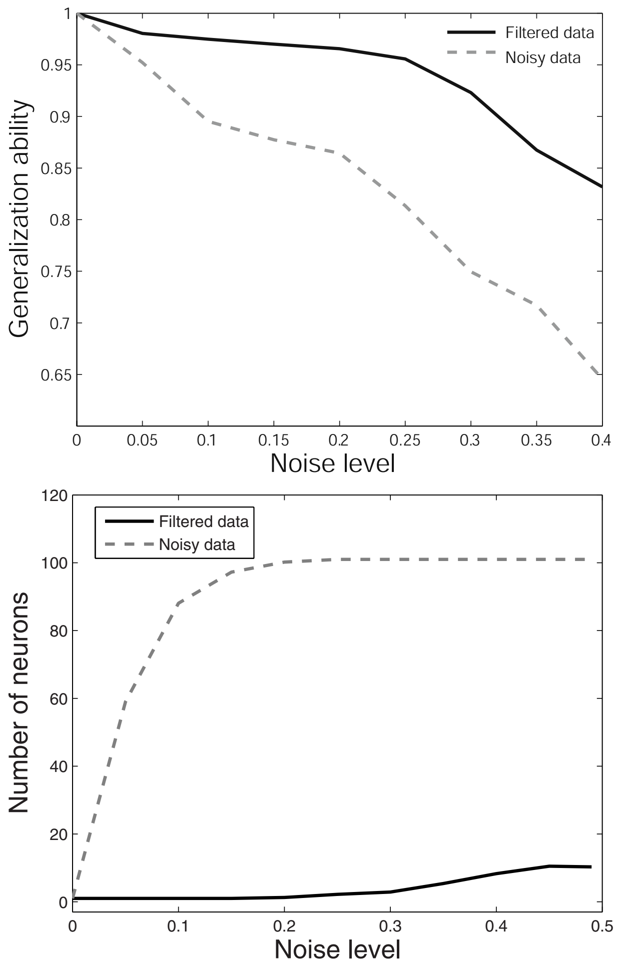


Fig. 5 The effect of adding class noise to the dataset. Top: Generalization ability as a function of the level of class noise for the modified Pima indians diabetes dataset for the cases of implementing the filtering stage and for the case of using the whole raw dataset. Bottom: The number of neurons of the generated architectures for the two mentioned cases of the implementation of the C-Mantec algorithm.

and if the number of presentations for an example is larger by two standard deviations from the mean, it is removed from the training set. The removal of examples is made on-line as the architecture is constructed and a final phase is carried out where no removal of examples is allowed.

To test the new method for removal of noisy examples a "noise-free" dataset is created from a real dataset, and then controlled noise was added to the attributes (input variables) and to the class (output), in separate experiments to analyze whether there is any evident difference between the two cases [23]. Knowing the origin of the noise is an interesting issue with practical applications, as it can help to detect the sources of noise and consequently help to eliminate it. The dataset chosen for this analysis is the Pima Indians Diabetes dataset, selected because it has been widely studied and also because it is considered a difficult set with an average generalization ability around 75%. To generate the "noise-free" dataset, the C-Mantec algorithm was run with a single neuron that classified correctly approximately 70% of the dataset, and then the "noise-free" dataset was constructed by presenting the whole set of inputs through this network to obtain the "noise-free" output. Two different experiments were carried out: in the first one, noise was added to the attributes of the dataset and the performance of the C-Mantec algorithm was analyzed with and without the procedure for noisy examples removal. In Fig. 4 (top) the generalization ability for both cases is shown for a level of noise between 0 and 0.8 and the results are the average over 100 independent runs. For a certain value of added noise, x, the input values were modified by a random uniform value between $-x$ and x. The bottom graph shows the number of neurons in the generated architectures when the filtering process was and was not applied as a function of the added attribute noise. It can be clearly seen that the removal of the noisy examples helps to obtain much more compact architectures while a better generalization ability is observed. The second experiment consisted in adding noise to the output values and the results are shown on Fig. 5. In this case the noise level indicate the probability of modifying the class value to a binary value, chosen randomly between 0 or 1.

From the experiments carried out with the two types of noise introduced to the Diabetes dataset we can observe that the resonance effect helps to detect and eliminate the noisy instances in both cases, helping to increase the generalization ability, even if the change is not enough to recover the generalization ability obtained in the noise-free case. It can also be observed that the size of the neural architectures obtained after the removal of the noisy instances is much lower than the size of the architectures needed for the noisy cases. Also, it has to be said that the experiments did not lead to a way of differentiating the sources of noise, as the results obtained for the two noise-contaminated datasets considered were not particularly different.

4 Experiments and Results on Public Domain Datasets

We tested the noise filtering abilities of the method introduced in this work using the C-Mantec constructive algorithm on a set of 11 well known benchmark functions [24]. The set of analyzed functions contains 6 two-classes functions and 5 multi-class problems with a number of classes up to 19. The C-Mantec algorithm was run with a maximum number of iterations of 50.000 and an initial temperature value (T_0) equals to the number of inputs of the analyzed functions. It is worth noting that different tests showed that the algorithm is quite robust to changes on these parameter values. The results are shown in Table 1, where it is shown the number of neurons of the obtained architectures and the generalization ability obtained, including the standard deviation values, computed over 100 independent runs. The last column of Table 1 shows, as a comparison, the generalization ability values

Table 1 Results for the number of neurons and the generalization ability obtained with the C-Mantec algorithm using the data filtering method introduced in this work. The last column shows the results from [25] (See text for more details).

Function	Inputs	Classes	Neurons	Generalization C-Mantec	Generalization NN [25]
Diab1	8	2	3.34 ± 1.11	76.62 ± 2.69	74.17 ± 0.56
Cancer1	9	2	1 ± 0.0	96.86 ± 1.19	97.07 ± 0.18
Heart1	35	2	2.66 ± 0.74	82.63 ± 2.52	79.35 ± 0.31
Heartc1	35	2	1.28 ± 0.57	82.48 ± 3.3	80.27 ± 0.56
Card1	51	2	1.78 ± 0.87	85.16 ± 2.48	86.63 ± 0.67
Mushroom	125	2	1 ± 0.0	99.98 ± 0.04	100.00 ± 0.0
Thyroid	21	3	3 ± 0.0	91.91 ± 0.59	93.44 ± 0.0
Horse1	58	3	3 ± 0.0	66.56 ± 5.08	73.3 ± 1.87
Gene1	120	3	3.03 ± 0.22	88.75 ± 1.07	86.36 ± 0.1
Glass	9	6	17.84 ± 1.19	63.75 ± 6.38	53.96 ± 2.21
Soybean	82	19	171 ± 0.0	91.63 ± 1.89	90.53 ± 0.51
Average	50.27	4.18	18.99 ± 0.43	84.21 ± 2.03	82.50 ± 0.63

obtained by Prechelt [25] in a work where he analyzed in a systematic way the prediction capabilities of different topologies neural networks. The size of training and test sets were chosen in a similar way in both compared cases: the training set comprises 75% of the total number of instances and the remaining 25% was used for testing the generalization ability. The results obtained with the C-Mantec algorithm outperforms the ones obtained by Prechelt in 6 out of 11 problems and on average the generalization ability is 2.1% larger. Regarding the size of the networks obtained using the new method introduced, the architectures are very small for all problems with 2 or 3 classes, for which the architectures contain less than 4 neurons. For multi-class problems the algorithm generates networks with a larger number of hidden neurons but this is because of the method used to treat multiclass problems that will be reported elsewhere [20].

A further set of experiments was carried out using as classification algorithms other standard methods in machine learning. Three different available algorithms were used and tested on the original datasets and on the filtered dataset where the noisy examples were eliminated. The three algorithms used were standard multilayer perceptrons (MLP) trained by backpropagation, Support Vector Machines (SVM) [27] and the C4.5 algorithm based on decision trees (C4.5) [26], all implemented under the WEKA package ([28]) using the default parameter settings. The results are presented in table 2 where the generalization ability obtained for six different datasets are shown for the two cases considered: filtered and original datasets. It can be observed that when multilayer perceptron are used, in all analyzed cases the generalization ability obtained with the filtered dataset was larger than with the original set and the difference was on average larger by a 1.31%, with values in some cases as large as 2.32%. For the case of the C4.5 algorithm the results with and without filtering instances were similar with an average difference of 0.18% in favor of the filtered case. When the support vector machines were tested with both

Table 2 Results for generalization ability obtained using standard multilayer perceptrons (MLP), decision trees (C4.5) and support vector machines (SVM) algorithms using both the filtered and original datasets (See the text for more details).

	Filtered data			Original data		
	MLP	C4.5	SVM	MLP	C4.5	SVM
Diab1	75.63 ± 4.21	75.21 ± 2.85	76.36 ± 3.43	74.58 ± 2.36	73.54 ± 2.78	77.81 ± 1.57
Cancer1	95.06 ± 0.78	93.91 ± 2.05	95.63 ± 1.00	94.84 ± 1.28	93.93 ± 1.83	95.90 ± 0.95
Heart1	82.49 ± 2.87	79.44 ± 1.15	81.80 ± 2.70	80.17 ± 2.23	78.70 ± 2.94	81.74 ± 2.38
Heartc1	83.20 ± 4.43	78.13 ± 3.22	82.13 ± 4.58	81.05 ± 2.70	79.47 ± 5.23	84.74 ± 4.21
Card1	85.81 ± 2.50	85.81 ± 3.68	86.74 ± 2.71	83.72 ± 1.81	85.93 ± 2.55	86.05 ± 1.90
Mushroom	100.00 ± 0.00	99.95 ± 0.05	100.00 ± 0.00	100.00 ± 0.00	99.86 ± 0.07	100.00 ± 0.00
Average	87.04 ± 3.65	85.41 ± 3.99	87.11 ± 3.69	85.73 ± 3.95	85.23 ± 4.09	87.71 ± 3.48

datasets, the generalization ability observed decreases with the filtered instances in average by approximately 0.61%, but noting that in 2 out of the 6 cases considered the prediction improved.

Regarding the generalization ability obtained by the different methods, we first note that the average generalization ability for the 6 functions shown in table 2 is of 87.29 ± 3.72 for the C-Mantec algorithm with the active learning procedure incorporated. Thus, the best method with these limited set of 6 functions turns out to be the SVM approach, close followed by the constructive C-Mantec algorithm and by the MLP; while the C4.5 came last with a lower generalization ability.

The number of instances in the filtered datasets was on average 2.73% smaller than the original sets, being the smaller ones found in those for which the generalization ability was lower, as for Diabetes dataset. The standard deviation of the results shown in tables 1 and 2 is computed over 5 randomly selected datasets, using 75% of the examples for training the models and the remaining 25% for testing the generalization ability.

5 Discussion

We introduced in this chapter a new method for filtering noisy examples using a recently developed constructive neural network algorithm. The new C-Mantec algorithm generalizes very well on free-noise dataset but have shown to overfit with noisy datasets and thus, a filtering scheme for noisy instances have been implemented. The filtering method devised is based on the observation that noisy examples needs more number of weights updates than regular ones. This "resonant effect" observed, permits to distinguish these instances and eliminate them in an on-line procedure. Simulations performed show that the generalization ability and size of the resulting networks are very much improved after the removal of the noisy examples. A comparison of results was done against previous reported values obtained using standard feed-forward neural networks [25] and showed that the generalization ability was on average a 2.1% larger, indicating the effectiveness of the C-Mantec algorithm implemented with the new filtering stage. The introduced method of data selection can also be used as a pre-processing stage for other prediction algorithms, and for this reason a second comparison was carried out using three well known predictive algorithms: MLP, C4.5 decision trees and SVM. The results obtained and shown in table 2 indicate that the instance selection procedure appears to work quite well with MLP and less with the other two algorithms. It might be possible, given the neural nature of the C-Mantec algorithm, that the filtering stage developed works better with neural-based algorithms but further studies might be needed to extract a final conclusion. Overall we have observed that the active learning procedure implemented using the new C-Mantec algorithm is working very efficiently in the task of avoiding overfitting problems and that comparable results to those obtained using MLP's and SVM's can be obtained with a constructive neural network algorithm.

Acknowledgements. The authors acknowledge support from CICYT (Spain) through grants TIN2005-02984 and TIN2008-04985 (including FEDER funds) and from Junta de Andalucía through grants P06-TIC-01615 and P08-TIC-04026. Leonardo Franco acknowledges support from the Spanish Ministry of Science and Innovation (MICIIN) through a Ramón y Cajal fellowship.

References

1. Haykin, S.: Neural Networks: A Comprehensive Foundation. Macmillan/IEEE Press (1994)
2. Lawrence, S., Giles, C.L., Tsoi, A.C.: What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation. In: Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, Univ. of Maryland (1996)
3. Gómez, I., Franco, L., Subirats, J.L., Jerez, J.M.: Neural Networks Architecture Selection: Size Depends on Function Complexity. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006. LNCS, vol. 4131, pp. 122–129. Springer, Heidelberg (2006)
4. Mezard, M., Nadal, J.P.: Learning in feedforward layered networks: The tiling algorithm, J. Physics A 22, 2191–2204 (1989)
5. Frean, M.: The upstart algorithm: A method for constructing and training feedforward neural networks. Neural Computation 2, 198–209 (1990)
6. Parekh, R., Yang, J., Honavar, V.: Constructive Neural-Network Learning Algorithms for Pattern Classification. IEEE Transactions on Neural Networks 11, 436–451 (2000)
7. Subirats, J.L., Jerez, J.M., Franco, L.: A New Decomposition Algorithm for Threshold Synthesis and Generalization of Boolean Functions. IEEE Transactions on Circuits and Systems I 55, 3188–3196 (2008)
8. Nicoletti, M.C., Bertini, J.R.: An empirical evaluation of constructive neural network algorithms in classification tasks. International Journal of Innovative Computing and Applications 1, 2–13 (2007)
9. Reed, R.: Pruning algorithms - a survey. IEEE Transactions on Neural Networks 4, 740–747 (1993)

10. Smieja, F.J.: Neural network constructive algorithms: trading generalization for learning efficiency? Circuits, systems, and signal processing 12, 331–374 (1993)
11. Bramer, M.A.: Pre-pruning classification trees to reduce overfitting in noisy domains. In: Yin, H., Allinson, N.M., Freeman, R., Keane, J.A., Hubbard, S. (eds.) IDEAL 2002. LNCS, vol. 2412, pp. 7–12. Springer, Heidelberg (2002)
12. Hawkins, D.M.: The problem of Overfitting. Journal of Chemical Information and Computer Sciences 44, 1–12 (2004)
13. Angelova, A., Abu-Mostafa, Y., Perona, P.: Pruning training sets for learning of object categories. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005, vol. 1, pp. 494–501 (2005)
14. Cohn, D., Atlas, L., Ladner, R.: Improving Generalization with Active Learning. Mach. Learn. 15, 201–221 (1994)
15. Cachin, C.: Pedagogical pattern selection strategies. Neural Networks 7, 175–181 (1994)
16. Kinzel, W., Rujan, P.: Improving a network generalization ability by selecting examples. Europhys. Lett. 13, 473–477 (1990)
17. Franco, L., Cannas, S.A.: Generalization and Selection of Examples in Feedforward Neural Networks. Neural Computation 12(10), 2405–2426 (2000)
18. Sánchez, J.S., Barandela, R., Marqués, A.I., Alejo, R., Badenas, J.: Analysis of new techniques to obtain quality training sets. Pattern Recognition Letters 24, 1015–1022 (2003)
19. Jankowski, N., Grochowski, M.: Comparison of Instances Seletion Algorithms I. Algorithms Survey. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 598–603. Springer, Heidelberg (2004)
20. Subirats, J.L., Franco, L., Jerez, J.M.: Competition and Stable Learning for Growing Compact Neural Architectures with Good Generalization Abilities: The C-Mantec Algorithm (2009) (in preparation)
21. Frean, M.: Thermal Perceptron Learning Rule. Neural Computation 4, 946–957 (1992)
22. Rosenhlatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65, 386–408 (1959)
23. Zhu, X., Wu, X.: Class noise vs. attribute noise: a quantitative study of their impacts. Artif. Intell. Rev. 22, 177–210 (2004)
24. Merz, C.J., Murphy, P.M.: UCI Repository of Machine Learning Databases. Department of Information and Computer Science. University of California, Irvine (1998)
25. Prechelt, L.: Proben 1 – A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms. Technical Report (1994)
26. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kauffman, CA (1992)
27. Shawe-Taylor, J., Cristianini, N.: Support Vector Machines and other kernel-based learning methods. Cambridge University Press, Cambridge (2000)
28. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, San Francisco (2000), `http://www.cs.waikato.ac.nz/ml/weka`