Genetic induction of descriptive fuzzy classifiers with the Logitboost algorithm

Luciano Sánchez[1], José Otero[1], A. López[2]

[1]Universidad de Oviedo. Depto. Informática.
[2]Universidad de Oviedo, DIEECS
Campus de Viesques, 33203 Gijón.

Abstract. Recently, Adaboost has been regarded as a particular case of a previous statistical method: greedy backfitting of extended additive models in logistic regression problems, or "Logitboost". The application of Logitboost to learn fuzzy classifiers from data should improve the performance of the fuzzy classifier in multiclass problems, and reduce the size of the fuzzy rule base. In this work, we propose a GA-based version of Logitboost and discuss some preliminary numerical results of this method.

1 Introduction

Boosting consists in combining low quality classifiers with a voting scheme to produce a classifier better than any of its components. The most common version of Boosting is called AdaBoost [5]. Recently, a close relationship between this method and Generalized Additive Models has been shown. Following [6], Adaboost is a specialization of the backfitting algorithm –used since the 80's to induce generalized additive models– whose greedy version is also known as "matching pursuit" [12] [17]. This relationship explains the mechanisms of Adaboost in terms of iterative approximations to maximum likelihood estimation over a family of additive models.
As a consequence of the new theoretical justification, some corrections were introduced to Adaboost and a new boosting method, called LogitBoost, was proposed in [6]. Logitboost should pose less numerical problems than Adaboost, and it was experimentally shown to improve the former method, being specially efficient in multiclass problems, to which Adaboost was difficult to extend.
Matching pursuit methods have been used to induce fuzzy classifiers and models in different ways. In fact, Iterative Rule Learning of models [4] and classifiers [2] are closely related to matching pursuit algorithms and can be regarded as the precursors of these algorithms, and Adaboost itself was directly applied to induce descriptive [10] and approximate [9] fuzzy classifiers. Backfitting has also been regarded as the counterpart of Adaboost in model estimation and also used to induce fuzzy models in previous works [14].

1.1 Summary

The structure of this paper is as follows: in the next section, fuzzy classifiers are introduced and it is explained how Adaboost can be applied to induce them from data. Then,

the Logitboost algorithm is explained, compared to Adaboost and adapted to learn fuzzy classifiers. The paper finishes with an empirical evaluation of the new algorithm.

2 Boosting Fuzzy Classifiers

2.1 Notation

At this point we introduce the basic notation employed throughout the paper. $\mathbf{X}$ is the feature space, and a feature $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{X}$ is an element of this space. $\mathcal{A} = \{A^j\}_{j=1\ldots N}$, with $A^j \subset \widetilde{\mathcal{P}}(\mathbf{X})$ is a fuzzy partition of $\mathbf{X}$. p is the number of classes. A sample is a set of m pairs $(\mathbf{x}_i, c_i)$, where $1 \leq c_i \leq p$. The set of training examples is indexed by the letter i, the set of rules by j, the features by f and the classes by k. The ranges of these variables are $1 \leq i \leq m$, $1 \leq j \leq N$, $1 \leq f \leq n$ and $1 \leq k \leq p$.

2.2 Descriptive fuzzy classifiers

A fuzzy rule based classifier is a fuzzy relationship defined on $\mathcal{A} \times \{1, \ldots, p\}$. Values of this relationship describe the degree of compatibility between the feature vector $\mathbf{x}_i$ and the class c_i. High values indicate high compatibility, whereas low values do not necessarily imply low compatibility; but rather codify the absence of knowledge about the actual degree of compatibility.
There are different standards when translating the former fuzzy relationship into linguistic statements. We combine p instances of the fuzzy relationship,

$$\text{compatibility}(A^j, c_k) = s_k \quad k = 1, \ldots p,$$

into a single sentence, as follows:

$$\begin{array}{c} \text{if } x_1 \text{ is } A_1^j \text{ and } \ldots \text{ and } x_n \text{ is } A_n^j \\ \text{then truth}(c_1) = s_1 \text{ and } \cdots \text{ and truth}(c_p) = s_p \end{array}$$

where $A^j = A_1^j \times \ldots \times A_n^j$. The overall classifier comprises as many sentences (fuzzy rules,) as elements in the fuzzy partition $\mathcal{A}$. It is immediate that not every arbitrary fuzzy set can be expressed as a Cartesian product of fuzzy sets defined over projections of the feature space. This restriction limits the choice of the fuzzy partition $\mathcal{A}$. We can restrict the search space even further by requiring the fuzzy sets A_f^j, $f = 1, \ldots, n$ to form n linguistic variables, in which case we obtain a fuzzy rule based *descriptive* classifier; otherwise we obtain *approximate* fuzzy rules. Descriptive fuzzy classifiers allow a linguistic interpretation of rules because the fuzzy sets A_f^j refer to linguistic values that are common to all the rules in the classifier. All terms A_f^j are associated to a linguistic value. Therefore, n linguistic variables $\mathcal{L}_f$, spanning one feature each, must be defined. Each linguistic variable comprises n_f terms, and every term is related to a fuzzy set L_f^q. In other words, $\mathcal{L}_f = \{L_f^q\}_{q=1\ldots n_f}$ and $A_f^j \in \mathcal{L}_f$.
Observe that in a descriptive fuzzy classifier the set of possible rules is finite due to the discrete number of possible linguistic labels associated to each rule. In other words, each fuzzy set has a label associated and the consequents and antecedents in the rule are

taken from that set of labels. Because of this, descriptive classifiers are expected to be more interpretable than approximate ones, since in the latter the meaning of the rules are related with the definition of the fuzzy sets, that change from one rule to another.
Fuzzy reasoning methods define how rules are combined and how to infer from a given input to the corresponding output. Frequently, an instance $\mathbf{x}$ is assigned to the class

$$\arg\max_{k=1,\ldots,p} \bigvee_{j=1}^{N} A^j(\mathbf{x}) \wedge s_k \tag{1}$$

where the "$\wedge$" and "$\vee$" can be implemented by different operators; for example, "$\vee$" can be the maximum operator [11] or the arithmetic sum, so called "maximum voting scheme" [11]. "$\wedge$" is always a t-norm.

2.3 The Adaboost Algorithm

Boosting is a technique that combine several individual classifiers into a "committee" that performs better than any of the single classifier. The first boosting algorithms use a voting scheme to combine the classifiers [15]. Let us define a set $\mathcal{H}$ of very simple, but possibly unreliable classifiers $g_j \in \mathcal{H}$. Boosting consists in combining these low quality classifiers (so called "weak hypotheses" in boosting literature) with a voting scheme to produce an overall classifier that performs better than any of its individual constituents alone. In our case, weak hypotheses correspond to fuzzy rules. Weak hypotheses in confidence rated Adaboost [16] take feature values as input and produce both a class number as well as a degree of confidence in the given classification. In two classes problems, these two outputs can be encoded with a single real number, $g_j(\mathbf{x}) \in \mathbf{R}$, whose sign is interpreted as the label of $\mathbf{x}$ and whose absolute value is interpreted as the confidence in the classification, the higher the better. AdaBoost is intended to produce a linear threshold of all hypotheses

$$H(\mathbf{x}) = \text{sign}\left(\sum_{j=1}^{N} \alpha_j g_j(\mathbf{x})\right) \tag{2}$$

minimizing an upper bound of the number of training errors; as in all machine learning techniques, reducing the training error is of secondary importance, as the primary objective of the classifier is to minimize the generalization error over previously unseen instances.
Adaboost can operate with any learning algorithm that generates a confidence rated classifier given a weighted data set. We will use this property to learn fuzzy rule based classifiers: boosting fuzzy rules is based on an algorithm able to fit one single fuzzy rule to a set of weighted examples. This algorithm will be repeated so many times as rules in the base, and the Adaboost algorithm produces the number of votes each rule is assigned and recalculates the weight of every example when the rule is added to the base.
Figure 1 shows the outline of the Adaboost algorithm adapted to learn fuzzy rules, as proposed in [10].

Given: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m), \quad \mathbf{x_i} \in \mathbf{R}^n,\ y_i \in \{-1, +1\}$
Initialize $D_1(i) = 1/m$, $s_1^j = 0$, $s_2^j = 0$
Select the number of rules N
For $j = 1, \ldots, N$:

- Find the fuzzy membership $A^j \in \mathcal{A}$ that minimizes $Z = \min_{A \in \mathcal{A}} \left(\sum_{i=1}^m D_j(i) \exp(-y_i A(\mathbf{x}_i)),\ \sum_{i=1}^m D_j(i) \exp(y_i A(\mathbf{x}_i)) \right)$
- Find numerically the value α_j that minimizes $Z_j(\alpha) = \sum_{i=1}^m D_j(i) \exp(-\alpha y_i A^j(\mathbf{x}_i))$
- If $\alpha_j > 0$ then $s_1^j = \alpha_j$ else $s_2^j = \alpha_j$.
- Update the weights:
$$D_{j+1}(i) = \frac{D_j(i) \exp(-\alpha_j y_i A^j(\mathbf{x}_i))}{K}$$
where K is another normalization factor, so that D_{j+1} is a distribution.

End For
$s_1^j = s_1^j / \max_{k,j}(s_k^j)$, $s_2^j = s_2^j / \max_{k,j}(s_k^j)$; $k = 1, 2$; $j = 1, \ldots, N$
Generate the rules

$$\text{if } x_1 \text{ is } A_1^j \text{ and } \ldots x_n \text{ is } A_n^j \text{ then } \text{tr}(c_1) = s_1^j \text{ and } \text{tr}(c_2) = s_2^j$$

Fig. 1. Adaboost algorithm applied to the induction of a descriptive, fuzzy rule based classification system. Two classes version.

3 Backfitting additive logistic classifiers: The Logitboost algorithm

The Adaboost algorithm was based on two *a priori* definitions: the concept of classifier (a linear threshold of a set of weak hypothesis,) and certain exponential bound of the number of errors. By the contrary, the Logitboost algorithm is an statistical estimation procedure that relies on an stochastic definition of a classification problem. The bound that Logitboost tries to minimize is the likelihood of a classifier, which is turn is restricted to a parametric family of density functions (a logit transform of an additive model.)

3.1 Generalized and Extended Additive Models

Additive models were introduced in the 80's to improve precision and interpretability of classical nonparametric regression techniques in problems with a large number of inputs. These models estimate an additive approximation to the multivariate regression function, where each of the additive terms is estimated using a univariate smoother.
Individual terms explain the dependence of the output variable with respect to their corresponding input variables, thus there exists a certain degree of interpretability in the model. While this kind of estimation avoids the curse of dimensionality, it is not able to approximate universally. Hastie and Tibshirani addressed this issue and proposed generalized additive models [6]. With these last models it is assumed that the mean of the output depends on a sum of terms through a nonlinear link function, and it is permitted that the response probability distribution is any distribution in the exponential family.

Many statistical models belong to this class, including additive models for Gaussian data and nonparametric logistic models for binary data like the one we are interested in. More formally, let y be the output random variable we wish to model, and let $x = (x_1, \ldots, x_n)$ be the input random vector. The objective of the modeling process consists in estimating the conditional expectation of y given x. Linear regression assumes

$$E(y|x) = f(x_1, \ldots, x_n) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n \tag{3}$$

and obtains $\beta_0, \ldots, \beta_n$ by least squares. Additive models generalize this schema by allowing the use of a sum of nonlinear univariate regressors

$$E(y|x) = f(x_1, \ldots, x_n) = s_0 + s_1(x_1) + \ldots + s_n(x_n) \tag{4}$$

where s_i are smooth functions that are estimated in a nonparametric fashion. Generalized additive models extend additive models by not assuming a Gaussian distribution of the output, but any probability distribution in the exponential family,

$$f_y(t; \theta; \phi) = \exp\left\{\frac{t\theta - b(\theta)}{a(\phi)} + c(t, \phi)\right\} \tag{5}$$

and making the additive component

$$f(x_1, \ldots, x_n) = s_0 + s_1(x_1) + \ldots + s_n(x_n) \tag{6}$$

to depend on the mean of the output by means of a link function g, so that $g(E(y|x)) = f(x_1, \ldots, x_n)$. The most commonly used link function in practice is the canonical link $g(E(y|x)) = \theta$.
Additive models can be generalized furthermore. In extended additive models, the univariate regressors s_i are replaced by functions of more than one feature. In our context, these functions usually depend on a set of parameters γ and a multiplier β,

$$s_i = \beta_i s(x; \gamma_i) \tag{7}$$

thus the additive model becomes

$$E(y|x) = f(x_1, \ldots, x_n) = s_0 + \sum_{i=1}^{n} \beta_i s((x_1, \ldots, x_n); \gamma_i). \tag{8}$$

For example, in radial basis neural networks the functions $s(x, \gamma_i) = \exp\{||x - \gamma_i||^2\}$ are the "basis functions"; γ_i are their centers and β_i are the weights that connect the input layer with the output. In support vector machines, $s(x, \gamma)$ is a kernel, and γ_i are the support vectors. We will show later that a fuzzy rule base can be casted in the same schema under certain mechanisms of approximate reasoning.

3.2 Backfitting and the Logitboost Algorithm

Extended additive models can be learned with a generalized backfitting algorithm [6]. Given a cost function d, that measures the differences between the conditional expectation and its approximation, this algorithm consists in finding n pairs of values $\{\beta_m, \gamma_m\}$

minimizing each

$$E\left[d\left(y, \sum_{\substack{k=1\ldots n \\ k\neq m}} \beta_k s(x;\gamma_k) + \beta s(x;\gamma)\right)\right] \tag{9}$$

with respect to β, γ [6]. A greedy approach, where the expectation of the output is incrementally approximated, produces good results in practice. Let $f_0(x), f_1(x), \ldots$ be successive approximations to $E(y|x)$; then, let us define

$$\begin{array}{l} \{\beta_m, \gamma_m\} \leftarrow \\ \arg\min_{\beta,\gamma} E[d\,(y, f_{m-1}(x) + \beta s(x;\gamma))] \end{array} \tag{10}$$

where $\{\beta_k, \gamma_k\}_1^{m-1}$ are fixed at their corresponding solution values at earlier iterations. Algorithms that learn a weighted sum of basis functions, by sequentially appending functions to an initially empty basis, to approximate a target function in the least-squares sense, are contained in the family of the *matching pursuit* algorithms [12]. These algorithms have been compared to support vector machines and radial basis neural networks in machine learning problems [17]. One of the most interesting properties of matching pursuit algorithms is that they are good in keeping the sparsity of the solution; this improves the generalization properties of the method and we will also see in the following sections that the same property guarantees a short number of rules in the fuzzy case that will be described later.

We have mentioned that the objective of a binary classification problem is to approximate the value $E(y|x) = p(c = 1|x)$, which we will denote by $p(x)$. The response variable in a classification problem follows the binomial distribution, and the link function is $g(p(x)) = \log \frac{p(x)}{1-p(x)}$ [6]; therefore, the additive model is

$$\log \frac{p(\text{class}(x) = 1)}{p(\text{class}(x) = 0)} = f(x_1, \ldots, x_n) = s_0 + \beta_1 s_1(x) + \ldots \tag{11}$$

and the output of the model, reversing the logistic transform,

$$p(x) = \frac{e^{f(x)}}{1 + e^{f(x)}} \tag{12}$$

If the greedy version of generalized backfitting, mentioned in the preceding section, is applied to this model, it is obtained an algorithm very similar to Adaboost, so called "Logitboost" algorithm [6]. An outline of this algorithm, for binary problems, is shown in Figure 2.

The "smooth" operation [6], consists in estimating the values β_i and γ_i on which the i-th additive term depends, by means of a suitable statistical or machine learning procedure. Every step can be understood as fitting a new term to a weighted set of residuals of the previous submodel. This residual is $z = \frac{y - p_{i-1}(x)}{p_{i-1}(x)(1-p_{i-1}(x))}$, and the weight of the residual at the element x in the sample is $p_{i-1}(x)(1 - p_{i-1}(x))$. The multiclass extension of these operations is inmmediate, but it is not given here because of space considerations.

a. Set $f_0(x) = 0$, $p_0(x) = 1/2$.
b. For step number $i = 1, \ldots, N$
   (a) Compute $\beta_i s_i(x) = \text{smooth}[\frac{y - p_{i-1}(x)}{p_{i-1}(x)(1-p_{i-1}(x))}]$
   with weights $p_{i-1}(x)(1 - p_{i-1}(x))$.
   (b) Update $f_i(x) = f_{i-1}(x) + \beta_i s_i(x)$
   (c) Compute $p_i(x) = \frac{e^{f_i(x)}}{1+e^{f_i(x)}}$
c. Output class$(x) = \arg\max(p_N(x), 1 - p_N(x))$

Fig. 2. Pseudocode of backfitting applied to a logistic extended additive model or *Logitboost*. After solving the step (b.a) as discussed in the text, the algorithm is similar to Adaboost.

4 Proposed methodology

4.1 Backfitting fuzzy classifiers

The procedure shown in Figure 2 can be directly applied to learn the fuzzy classifiers defined in section 2.2. The only step in the process that is still undefined is the *smooth* operation that fits one term (one fuzzy rule) to the residual. Then, this rule is added to the base, the residual recalculated and the process repeated until the final number of rules is obtained. A detailed pseudocode of this algorithm is shown in Figure 3. Observe that we have replaced the word *smooth* by *fit one rule*.

```
rule base = emptyset
repeat
 do i=1..N
   votes[i]=inference(rule base,X[i])
   do k=1..p
      p[i][k]=1/suma_{k'=1..p}(exp(votes[i][k]-votes[i][k']))
      weight[i][k]=p[i][k]*(1-p[i][k])
      residual[i][k]=(y[i][k]-p[i][k])/(p[i][k]*(1-p[i][k]))
    end do
 end do
 R = fit_one_rule(X,residual,weight)
 rule base = rule base + R
until rule base contains enough rules
```

Fig. 3. Backfitting a fuzzy classifier is a greedy method that is based on a standard regression algorithm, "`fit_one_rule`". This last procedure finds the descriptive type-3 rule that best fits a weighted sample of residuals.

Observe the similarities between this process and the selection stage in Genetic Iterative Learning [4]. In GIL, the rule which best explains the data is selected and added to the rule base, then the examples covered by the rule are deleted from the training sample. Here, the weight of an example is the residual of the intermediate bank in it, and every

new rule will focus in the points with high error. Since covered points are not deleted but downweighted, GIL's multiselection stage is not needed here and the whole process is very fast.
The smoothing process is itself a standard regression problem: we must search for the fuzzy rule that best approximates the residual, in the least squares sense. Since we are using "type 3" rules, consequents comprise p real values. These values can be analitically determined once we know the antecedent. Therefore, the procedure "`find_one_rule`" is a search in a finite space (the space of valid linguistic antecedents;) this search is guided by a fitness value that measures the squared error of the rule being added over the weighted sample.

4.2 Search of antecedents

Since the space of antecedents is finite, an exhaustive search can be done in many practical problems. For those situations in which the number of features prevents this option, a suboptimal search must be applied.
Binary coded genetic algorithms are the natural choice for this problem, and we have experimentally checked that the rules that the GA finds are close to the optimal ones.
We will use a coding scheme based in [7]. Let us codify a linguistic term with a '1' bit in a chain of so many bits as different terms in the linguistic partition. For example, let {LOW, MED, HIGH} be the linguistic labels of all features in a problem involving three input variables and two classes. The antecedent of the rule

$$\begin{array}{c}\texttt{If } x_1 \texttt{ is High and } x_2 \texttt{ is Med and } x_3 \texttt{ is Low} \\ \texttt{then class is C1 with seg} = \texttt{S1},\ \texttt{C2 with seg} = \texttt{S2}\end{array}$$

is codified with the chain 001 010 100. We extend this encoding to represent rules for which not all variables appear in the antecedent. 'OR' combinations of terms in the antecedent are also allowed. For example, the antecedent of the rule

$$\texttt{If } x_1 \texttt{ is High and } x_3 \texttt{ is Low then ...}$$

is codified with the chain 001 000 100, and

$$\texttt{If } x_1 \texttt{ is}(\texttt{ High or Med}) \texttt{ and } x_3 \texttt{ is Low then ...},$$

will be assigned the chain 011 000 100. With this structure, the GA is also exploited to integrate a rule-wise feature selection process into the search scheme.
Observe that, under the fuzzy reasoning method used here, chains "001 111 100" and "001 000 100" are equivalent. "OR" combinations of rules increase the complexity of the knowledge base and we desire to minimize their number in the final result. Therefore, to promote simpler individuals, it was decided that in case of tie when evaluating the squared error of two different individuals, the one with a lower number of bits is preferred. This way, the search is guided toward rule banks that might not use all features.

Table 1. Mean test errors (10 repetitions of the experiment)

	LIN	QUA	NEU	1NN	WM	HL	PM	GIL	ABD	LBD
pima	0.227	0.252	0.255	0.289	0.287	0.301	0.464	0.269	0.241	0.239
cancer	0.044	0.051	0.047	0.048	0.129	0.058	0.087	0.099	0.040	0.037
gauss	0.239	0.190	0.200	0.267	0.477	0.304	0.457	0.205	0.213	0.203
glass	0.403	-	0.439	0.354	0.453	0.503	0.647	0.363	0.359	0.354
image	0.084	-	0.090	0.049	0.329	0.833	0.755	0.130	0.136	0.105
gauss5	0.317	0.317	0.323	0.413	0.539	0.344	0.931	0.338	0.327	0.325

5 Preliminary numerical results

We have selected six benchmarks. Four of them are widely used in the machine learning literature and are based on real problems: Iris (multiclass, low noise), Pima (two classes, moderate noise), Cancer (two clases, low noise), Glass (multiclass, high noise), and two synthethic sets of gaussian data: Gauss dataset [8] (two overlapping Gaussian distributions) and Gauss5 (our own generalization of Gauss, comprising five overlapping Gaussian clouds.) We applied an extension of the WM method modeling method to classification [1, 4], HL and PM methods [13]. Statistical classification methods were linear and quadratical discriminant analysis, neural networks and nearest neigbour.
In Table 1 the mean error rates over the test set are given. Besides this set of experiments is preliminar, it seems that Logitboost uniformly improves Adaboost for these five problems.

6 Concluding Remarks

GA-based Logitboost is not as general as other genetic methods that learn fuzzy classifiers. Not all fuzzy reasoning methods are suitable for it: the t-norm must be the product, and the votes of the rules must be added instead of being combined with a s-norm. But, when these limitations are accepted, greedy backfitting with Logitboost is very precise, and faster than other genetic fuzzy systems; it is remarkable than it can also be faster than many "ad-hoc" methods while being comparable in precision and learning time to neural networks.

References

1. Chi, Z., Yan, H., Pham, T. (1996) *Fuzzy Algorithms: With Applications to Image Processing and Pattern Recognition*. World Scientific.
2. Cordón, O., Del Jesus, M. J., Herrera, F. (1999) "A proposal on reasoning methods in fuzzy rule-based classification systems". International Journal of Approximate Reasoning **20**(1), pp. 21-45.
3. Cordón, O., Herrera, F. and Peregrin, A. (1997) "Applicability of the fuzzy operators in the design of fuzzy logic controllers," Fuzzy Sets and Systems, 86, pp. 15-41.
4. Cordón, O., Herrera, F. (2000) "A proposal for improving the accuracy of linguistic modeling". IEEE Transactions on Fuzzy Systems, 8, 3, pp. 335-344.

5. Freund, Y., Schapire, R. (1996) ”Experiments with a new boosting algorithm”. In Machine Learning, Proc. 13th International Conference, pp 148-156.
6. Friedman, J., Hastie, T., Tibshirani, R. (1998) “Additive Logistic Regression: A Statistical View of Boosting”. Machine Learning.
7. Gonzalez, A., Perez, R. (1996) “Completeness and consistency conditions for learning fuzzy rules,” Fuzzy Sets and Systems, vol 96, pp 37-51.
8. Haykin, S. (1999) *Neural Networks*. Prentice Hall.
9. Hoffmann, F. Boosting a Genetic Fuzzy Classifier, IFSA/NAFIPS 2001, Vancouver, Canada, July 25-28th, 2001
10. L. Junco and L. Sanchez. (2000) Using the adaboost algorithm to induce fuzzy rules in classification systems. ESTYLF 2000, pp 297-301.
11. Kuncheva, L. I. (2000) *Fuzzy Classifier Design*, Springer-Verlag, Heidelberg.
12. Mallat, S. Zhang, Z. (1993) “Matching pursuits with time-frequency dictionaries”. IEEE Trans on Signal Processing 41, pp 3397-3415.
13. Pal, S. K., Mandal, D. P. (1992) “Linguistic recognition system based in approximate reasoning”. *Information Sciences* **61**, pp. 135-161.
14. Sanchez, L. (2001) A Fast Genetic Method for Inducting Linguistically Understandable Fuzzy Models. Fuzzy Sets and Systems. To appear.
15. Schapire R. E. (1990) The strength of weak learnability. Machine Learning, 5(2):197–227.
16. Schapire, R., Singer, Y. (1999) “Improved Boosting algorithms using confidence-rated predictions”. Machine Learning, 37(3):297-336.
17. Vincent P., Bengio, Y. (2001) Kernel Matching Pursuit, Machine Learning Journal, Special Issue on New Methods for Model Combination and Model Selection.
18. Wang, L. X., Mendel, J. (1992) “Generating fuzzy rules by learning from examples”. IEEE Trans. on Systems, Man and Cybernetics, 25, 2, pp. 353-361.