GA-stacking: Evolutionary stacked generalization

Agapito Ledezma*, Ricardo Aler, Araceli Sanchis and Daniel Borrajo
Computer Science Deparment, Universidad Carlos III de Madrid, Avda. de la Universidad, 30m 28911 Leganés. Madrid, Spain

Abstract. *Stacking* is a widely used technique for combining classifiers and improving prediction accuracy. Early research in *Stacking* showed that selecting the right classifiers, their parameters and the meta-classifier was a critical issue. Most of the research on this topic hand picks the right combination of classifiers and their parameters. Instead of starting from these initial strong assumptions, our approach uses genetic algorithms to search for good *Stacking* configurations. Since this can lead to overfitting, one of the goals of this paper is to empirically evaluate the overall efficiency of the approach. A second goal is to compare our approach with the current best *Stacking* building techniques. The results show that our approach finds *Stacking* configurations that, in the worst case, perform as well as the best techniques, with the advantage of not having to manually set up the structure of the *Stacking* system.

Keywords: Ensembles of classifiers, stacking, genetic algorithms, meta-classifier

1. Introduction

One of the currently favored lines of research in Machine Learning is the combination of classifiers to improve classification accuracy [9]. This approach is known as *ensembles of classifiers* in the supervised learning area. The main idea behind ensembles, is that they are often much more accurate than the individual classifiers that make them up.

Typically, ensembles are constructed by generating several classifiers with the same learning algorithm [10]. In order to generate different classifiers, there are several methods that can be grouped in: sub-sampling the training examples (e.g. bagging [3] and boosting [18]); manipulating the input features [5]; manipulating the output target (e.g. ECOC [11]); and injecting randomness in the learning algorithm [26]. Once the classifiers have been generated, they are combined, in most cases by voting or weighted voting.

Other research in the area uses different learning algorithms over a dataset to generate the ensemble members. One example of this approach is *Stacking* [39]. *Stacking* uses an algorithm to learn how to combine the outputs of a set of classifiers that have been obtained by different learning algorithms. One of the problems of *Stacking* is how to obtain the right combination of base-level classifiers and the meta-classifier, specially in relation to each specific dataset. If the number of classifiers and algorithms to be used is small, this problem can be solved by a simple method in reasonable time: exhaustive search. But, when the search space is large, it is difficult to find the best *Stacking* configuration. In a previous paper [27], we presented an approach based on genetic algorithms (GAs) [21]. A somewhat related approach uses GAs for ensembles of neural networks, but only to determine the right number of ensemble members and in the context of homogeneous ensembles, not heterogeneous classifier generation algorithms like Stacking [40]. In an approach previous to *Stacking*, Wolpert [38] searched a space of generalizers with a particular version of an evolutionary algorithm. The search was restricted to the base generalizers. Also, English [15] used an evolutionary algorithm to explore the space of base classifiers (recurrent neural networks only). This approach was tested in two time series problems.

There are also many variants of the basic *Stacking* algorithm. The methods that have reported the best results are *Stacking* with multi-response model trees [13] and *Stacking* with multi-response linear regression and a reduced set of attributes [32]. In this paper, we continue our line of thought, based on not imposing strong assumptions on the final *Stacking* configuration and letting the GA find an accurate one freely. In particular, we have extended our *GA-Stacking* approach by enlarging the search space of *Stacking* configurations taking into account not only the learning algorithms but also their parameters. This is easy to achieve in a genetic approach, by coding the parameters in the GA individual. We then compare results with two recent *Stacking* state-of-the-art approaches. We show that our approach is able to find a *Stacking* configuration that performs comparably well and in some case better than those state-of-art *Stacking* systems with the advantage of not having to set manually an appropriate selection of algorithms and parameters.

This paper is organized as follows: Section 2 gives some background on *Stacking* and recent *Stacking* variants. Section 3 introduces the *Stacking* configuration approach, *GA-stacking*. Section 4 describes the experimental evaluation of the *GA-Stacking* viability. Section 5 and Section 6 show the results of the performed experiments carried out. Section 7 draws some conclusions. Finally, Appendices A and B show the parameters of the learning algorithms used by *GA-Stacking*.

2. Stacking Approaches

In this Section, we first describe the *Stacking* framework and then we present a brief description of some recent work on stacking.

2.1. Stacked Generalization

Stacking is the abbreviation that refers to Stacked Generalization [39]. The main idea of *Stacking* is to combine classifiers from different learners such as decision trees, instance-based learners, etc. Since each one uses a different knowledge representation and different learning biases, the hypothesis space will be explored differently, and different classifiers will be obtained. Thus, it is expected that their errors will not be perfectly correlated, and that the combination of classifiers will perform better than the base classifiers.

Once the classifiers have been generated, they must be combined. *Stacking* uses the concept of the meta learner. The meta learner (or level-1 model), tries to acquire, using a learning algorithm, how the outputs of the base classifiers (or level-0 models) should be combined to obtain the final classification. This is achieved by a cross-validation-like process, as described in [39].

More formally, given a data set S, *Stacking* first generates a subset of training sets $S_1, \ldots, S_T$ and then follows something similar to a cross-validation process: it leaves one of the subsets out (e.g. S_j) to be used later. The remaining instances $S^{(-j)} = S - S_j$ are used to generate the level-0 classifiers by applying K different learning algorithms, $k = 1, \ldots, K$, to obtain K classifiers. After the level-0 models have been generated, the S_j set is used to train the meta learner (level-1 classifier). Level-1 training data is created from the predictions of the level-0 models over the instances in S_j, that were left out for this purpose. Level-1 data has K attributes, whose values are the predictions of each one of the K level-0 classifiers for every instance in S_j. Therefore, a level-1 training example is made of K attributes (the K predictions) and the target class, which is the right class for every instance in S_j. Once the level-1 data has been built from all instances in S_j, any learning algorithm can be used to generate the level-1 model. To complete the process, the level-0 models are re-generated from the whole data set S (this way, it is expected that classifiers will be slightly more accurate). To classify a new instance, the level-0 models produce a vector of predictions that is the input to the level-1 model, which in turn predicts the class.

2.2. Related Work

Recent work on *Stacking* addresses the *Stacking* configuration problem: what algorithm and features are to be used in the meta-level. We present here a brief review of these work.

A necessary condition to create a good ensemble of classifiers is that base-level classifiers error predictions are uncorrelated [20]. In [29] a variant of stacking is proposed that uses correspondence analysis in order to detect correlations between the base-level classifiers. Once dependencies have been removed from the original meta-level space, a nearest neighbor method (meta-level algorithm) is then applied over the resulting features space. This approach is called SCANN.

In [34], they use probability distributions for the outputs from level-0 models instead of a simple class prediction as meta-level attributes. By using probability distributions as meta-level data, the authors argue that both, prediction and confidence of the base-level classifiers are used. In order to use this type of meta-level data, the authors proposed to use the *multi-response linear regression* technique (MLR) as the meta-level algorithm.

Another variant of *Stacking* [33] creates a meta-level classifier for each level-0 classifier. The learning task for each level-1 classifier is to predict whether the level-0 classifier prediction will be correct. The meta-level data is composed of base-level attributes and the class values are *correct* or *incorrect*. The predictions of those classifiers are combined by summing up the predicted probability distribution.

In [36] a new variant of *Stacking* is described that uses another learning method in the meta-level. This method, called *meta decision trees* (MDTs), replaces class-value predictions in its leaf nodes by the base level-classifiers. The meta-level data is composed of properties of the probability distributions that reflect the confidence of the base-level classifiers, like entropy and maximum probability, rather than the distributions themselves. Based on these properties, small MDT's are generated.

Based on *Stacking* with MLR (SMLR), [32] proposed to reduce the number of meta-level attributes independently of the number of classes, in order to overcome a weakness of SMLR in domains with more than two classes. This method is called *StackingC* (STC).

Džeroski and Ženko [14] proposed two new variants of stacking. First, based on *Stacking* with MLR, they propose to extend the set of meta-level features augmented with the probability distribution multiplied by the maximum probability and the entropies of the probability distributions. On the other hand, they propose another extension of SMLR in which they use model tree induction instead of linear regression as the meta-level algorithm. This method is called *Stacking* with multi-response model trees (SMRMT).

Comparing *Stacking* approaches overall, SCANN, MDTs, SMLR and SelectBest (selecting the best classifier with cross-validation) seem to perform at about the same level [14]. Moreover, they concluded that *Stacking* with multi-response model trees (SMRMT) performs better than existing *Stacking* approaches, including STC, and selecting the best classifier from the ensemble by cross-validation.

One of the main conclusions of all this previous work is that there are many contradictory results and there is no consensus on which combination of classifiers is the best one. The main differences of previous work with respect to our approach are that we do not select "a priori":

- which meta-classifier to use,
- the parameters of the meta-classifier,
- the number of base classifiers
- which of the available base classifiers to use, nor
- the parameters of these base classifiers

The main advantage of our approach is its flexibility and its non "a-priori" commitments. The system is very extensible. It can benefit from new classifiers since they can be easily incorporated into the pool of available classifiers together with their parameters, and coded in the *GA-Stacking* chromosome. Another advantage of our approach is that it is dataset dependent, so we also adapt to dataset biases and features, while the rest of approaches use the same configuration for all datasets.

3. GA-Stacking

Since an ensemble of classifiers generated from *Stacking* is composed by a group of models created from different learning algorithms, the following question arises: what algorithm should be used to generate the level-0 and the level-1 models?. Wolpert [39] originally indicates that many aspects on the parameters of *Stacking*, including the algorithms that generate the classifiers can be considered as black art. At first any learning algorithm can be used to generate the classifiers for both levels. As we described in the previous section, there have been some work focused on providing answers to these questions. For example, Ting and Witten [35] showed that a linear model is useful to generate the level-1 classifier when outputs of level-0 classifiers are probabilistic, Seewald [32] proposes a variation in the level-1 attributes space, Džeroski and Ženko [14] propose a regression tree as the meta-classifier instead of the linear model proposed by Ting and Witten.

Based on the idea of using probability class distributions as meta-level data proposed by Ting and Witten [35], we propose a new approach based on genetic algorithms which searches for the optimal configuration of *Stacking* parameters in a given problem. In Section 3.1 we describe the general framework of our approach. In Section 3.2 we detail the solution encoding for our GA-based system. Finally, in Section 3.3 we show the evaluation process of solutions found by the GA.

3.1. General Framework

GA-Stacking is the acronym of *Genetic Algorithms for Stacking*. *GA-Stacking* provides answers to the previous questions: which and how many learning algorithms are necessary to generate base-level classifiers and, what algorithm should be used to generate the meta-level classifier? The proposed solution consists on considering them as an optimization problem, which can be solved by applying Genetic Algorithms. Figure 1 shows the proposed general framework.

The application of GAs to an optimization problem requires, mainly, the study of two aspects: the encoding of the candidate solutions and the definition of the fitness function. The process of codification of the solutions takes place before (Fig. 1, Phase I) the execution of the GA. With respect to the fitness evaluation, it is an iterative process that is carried out in each generation of the GA (Fig. 1, Phase II) on all the individuals (p) of the population (P). The encoding-solutions phase is detailed in Section 3.2. The fitness evaluation is detailed in Section 3.3.

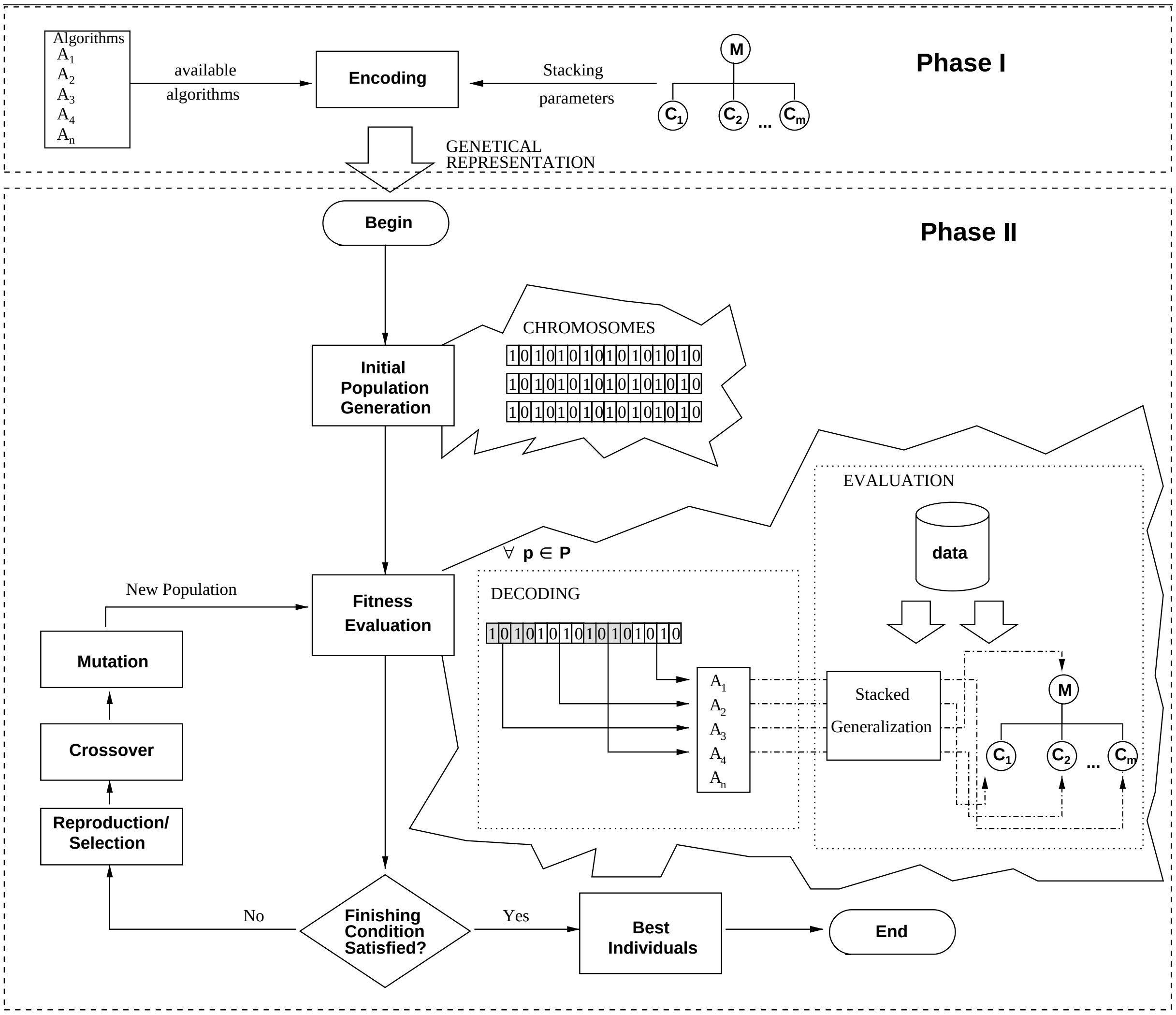


Fig. 1. Proposed framework *GA-Stacking*.

3.2. Encoding of Candidate Solutions

There are different ways to represent the solutions of a problem for a GA (e.g. binary, decimal, hexadecimal, etc. codifications) In order to represent the candidate solutions or individuals in *GA-Stacking*, we decided to use a binary representation, since it allows the use of canonical GAs. The canonical GA is the original form proposed by Holland [21,22] and has a stronger mathematical foundation [19,30]) which relies on binary codifications

The second decision consists on selecting the size of the chromosomes, which is given by two factors in our case:

- the number of algorithms that can be selected in order to generate the base-level and meta classifier (m).
- the maximum number of base-level classifier in the ensemble (n).

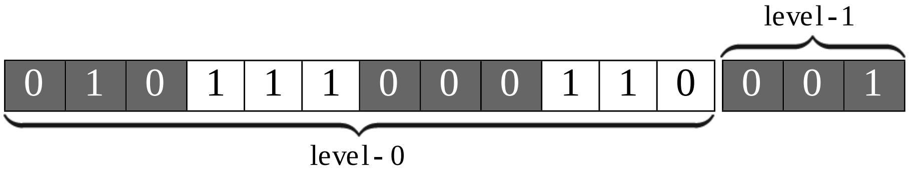


Fig. 2. A binary string representing a combination of four base-level classifiers and one meta-level classifier.

If we consider only the classifier name in the coding process, the gene size depends on the available learning algorithms. For example, if there are seven learning algorithms available to generate the members of the ensemble, we can use three bits to represent a gene. In this case, the gene can represent the use of any of the seven algorithms, and the option of using none. Figure 2 shows an individual in which the first four genes of the chromosome represent the four learning algorithms used to generate the base-level classifiers and, the last gene represents the algorithm used to create the meta-classifier.

In addition to the identifier of an algorithm, we also want to represent the algorithm's parameters (like the number of hidden neurons in a neural network). In that case, more than one gene will be needed per algorithm. If there are n base classifiers and G is the number of genes per classifier (including parameters), the total number of genes per individual, T_c, is given by

$$T_c = G \times (n+1)$$

Concerning the length of the chromosome in bits, it derives from the number and size of genes used to represent the available algorithms able. In other words the length of the chromosome in bits is given by

$$T_b = (n+1) \times \sum_{i=1}^{G} x_i$$

where x_i represents the number of bits used for coding the gene i. For example, the individual shown in Fig. 2 has $G = 1$, $n = 3$ and $x_i = 3$.

3.3. Fitness Evaluation

The fitness evaluation process is carried out in two phases: the decoding and construction phase; and the fitness calculation phase. In the first phase, decoding and ensemble construction (Fig. 3[a]), we generate the ensemble of classifiers using the algorithms represented in the chromosome and the training dataset. The first step is to extract the identifier and the parameters for each algorithm from the chromosome. After that, we use a subset of available domain data (training data) to generate the ensemble.

In the second phase (Fig. 3 [b]), the fitness calculation, we use a different dataset (validation data) in order to estimate the fitness value of each individual. The fitness is the accuracy on the validation dataset.

3.4. Setting the GA parameters

In addition to the codification type and the number of genes in the chromosome, GAs have other parameters. Structural parameters like the population size, and execution parameters like mutation and elite rate, have to be set. Nevertheless, in this work, we have used the most frequently values for this parameters [19] yielding good results.

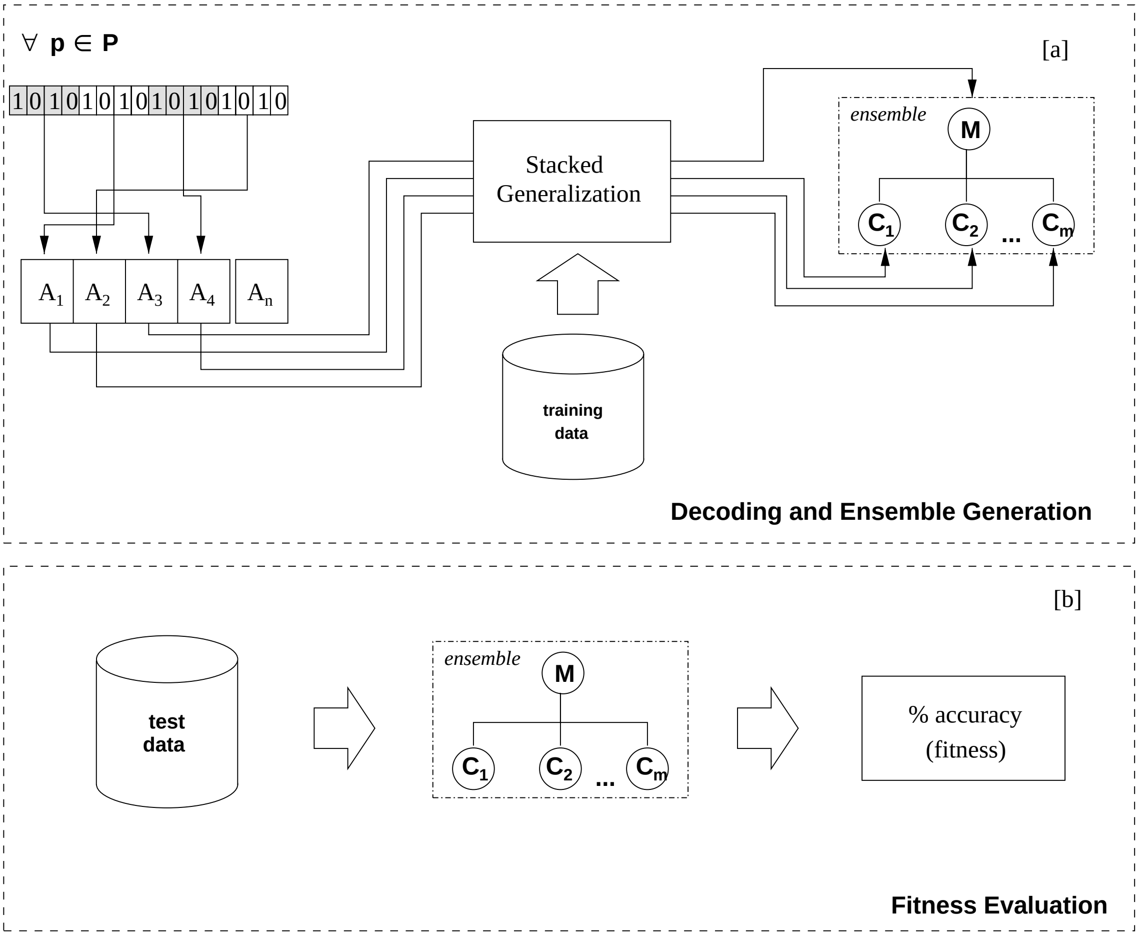


Fig. 3. Fitness computation in *GA-Stacking*.

4. GA-Stacking viability evaluation

In order to evaluate our approach we carried out two simple preliminary experiments to show the initial validity of *GA-Stacking*. In order to carry out this evaluation we used several domains from the UCI repository [2]. At a firs stage we obtained promising results, but there were signs of overfitting In order to avoid overfitting we carried out another set of experiments with a different fitnes evaluation process. In all these experiments, we used the learning algorithms implemented in the WEKA suite [37] (version 3.1.7). WEKA includes all algorithms used for experimentation including the ensemble generation algorithms (i.e. *Bagging*, *Boosting* and *Stacking*).

The implementation of *GA-Stacking* has two parts: the learning algorithms for which we used WEKA; and the GA, for which we used the GAJIT library (*Genetic Algorithm Java Implementation Toolkit*) [16].

4.1. Preliminary results

In this Section we show the results of *GA-Stacking* over two domains from the UCI repository: *ionosphere* and *dermatology*, given that they have been widely used on other comparisons related to ensembles of classifiers Table 1 shows the values of the GA parameters used in this experiment.

Table 1
GA parameters

Parameter	Value
Population	10
Generations	10
Elite rate	0.10
Cull rate	0.40
Mutation rate	0.067

Given that we are only studying the viability of the system and that the time to run the experiments is large, we have only considered here a small population size and a small number of generations. Regarding to the elite, cull and mutation rates, we use the defaults values of the GAJIT library.

We have selected seven algorithms in order to generate the ensemble of classifiers

- C4.5 [31]. It generates decision trees – (C4.5).
- A probabilistic *Naive Bayes* classifier [24] (NB).
- A simple *Naive Bayes* classifier where numerical attributes are modeled by a normal distribution [12] (NBS).
- IB*k* [1]. This is Aha's instance-based learning algorithm – (IB*k*). Default value of k is 1.
- PART [17]. It creates decision lists from partially pruned decision trees, generated using the C4.5 heuristic – (PART).
- *Decision Table* [25]. It is a simple classifier that uses the majority class. – (DT).
- *Decision Stump* [23]. It generates one-level decision trees – (DS).

On the other hand, we have used two algorithms to generate the ensemble in order to compare the results of *GA-Stacking* with other ensembles of classifiers The algorithms are:

- *Bagging*: algorithm for creating homogeneous ensembles of classifiers based on subsampling the training set. In this experiment we used C4.5 as the base classifier.
- *Boosting*: algorithm for creating homogeneous ensembles of classifiers based on weighting training instances. We used *AdaBoostM1* with C4.5 as the base classifier.

In order to evaluate the individual found by *GA-Stacking*, we performed the following:

- Each dataset was randomly divided into two parts, A and B
- Part A (around 85% of available domain instances) was used as the training set, and as the fitness evaluation set.
- In order to compute the fitness of each individual in the population, we generate an ensemble of classifiers using *Stacking*. The *Stacking* parameters (learning algorithms) are codified in the chromosome. Once the ensemble of classifiers has been generated, we use the accuracy of the ensemble over the training set (set A) as the fitness value.
- The dataset B was used as test set. Since these experiments were preliminary, the fitness evaluation process has not been carried out as we described in Section 3.3, where we proposed to split the training data in two sets in order to carry out a cross-validation.

Results are shown in Table 2. Columns two and four show the accuracy on the training data of the ensemble of classifiers in the *ionosphere* and *dermatology* domains. Columns three and five show the results over the test datasets.

The top rows of the table show the results of using the available individual learning algorithms. In the center rows of the table, we show the results of *Bagging* and *Boosting* and the best individual found

Table 2
GA-Stacking evaluation: preliminary results

Algorithm	Ionosphere		Dermatology	
	Training	Test	Training	Test
Single				
C4.5	98.42	82.86	96.67	92.42
Naive Bayes	85.13	82.86	98.67	**96.97**
Naive Bayes Simple	85.13	82.86	98.67	**96.97**
PART	98.73	88.57	96.67	93.94
IB1	100.00	82.86	100.00	92.42
Decision Stump	84.18	80.00	51.33	45.45
Decision Table	94.94	88.57	96.67	87.88
Ensembles				
Bagging with C4.5	97.78	85.71	97.00	93.94
Boosting with C4.5	100.00	**91.43**	100.00	**96.97**
Final GA-Stacking	100.00	85.71	100.00	95.45
Intermmediate GA-Stacking	97.78	94.29	98.67	98.48

by *GA-Stacking* after all generations. The individual found by *GA-Stacking* obtains a 100% of accuracy over the training set in both domains. However, the results over the test sets (85.71% for *ionosphere* and 95.45% for *dermatology*) show a potential overfitting. In the *ionosphere* domain the result obtained by the final *GA-Stacking* individual is worse than PART, Decision Table and *Boosting*. In the *dermatology* domain the results of the final *GA-Stacking* individual is surpassed by the classifiers generated from *Naive Bayes* and *Boosting*, both obtaining the best results in this domain (96.97%).

However, if we analyze the evolution of the fitness value of intermediate individuals, we can see that in previous generations *GA-Stacking* found individuals that obtained better results than any single or ensemble classifier over the test dataset in both domains (last row of Table 2). This reinforces the idea that *GA-Stacking* overfits to the training data.

Figure 4 shows the fitness evolution in the *dermatology* domain. In the third generation, there is an individual that has the maximum fitness value (100% of accuracy), but the results over the test set get worse from this generation onwards. Figure 4 also shows the average accuracy of the best three individuals of each generation in training/fitness and the results over the test set. The fitness average reaches 100% of success in the sixth generation, but the accuracy in test had already been decreasing.

4.2. Avoiding Overfitting

As we just saw, the *Stacking* configurations found by *GA-Stacking* overfitted. One of the reasons for this behavior, might be that the fitness value is obtained using the same instances employed to generate the ensemble of classifiers by means of *Stacking*. In order to avoid overfitting, we designed a new set of experiments where the fitness value is computed from a new dataset (validation dataset). In other words, the training set was split randomly into two datasets. 80% of the original training instances are used to generate the ensemble of classifiers from the configuration found by *GA-Stacking*, and the rest of instances – the validation dataset – are used to estimate the fitness value. This scheme has the disadvantage that the ensemble of classifiers is generated from fewer instances than previous experiments.

In the following experiments we extended the number of domains in order to evaluate *GA-Stacking*. We used six domains from the UCI repository. These domains are defined in Table 3.

We used the same GA parameters as in the previous set of experiments (see Table 1). A five-fold cross-validation was performed. For experimental reasons we execute the GA only one time for each fold. The classification accuracy of each classifier/ensemble C, over a given domain is the average

Table 3
Domains used for *GA-Stacking* evaluation

Domain	Attributes	Attributes Type	Instances	Classes
dermatology	34	numeric-nominal	366	6
dna-splice	60	nominal	3190	3
heart	13	numeric-nominal	303	2
ionosphere	34	numeric	351	2
musk	166	numeric	476	2
sonar	60	numeric	208	2

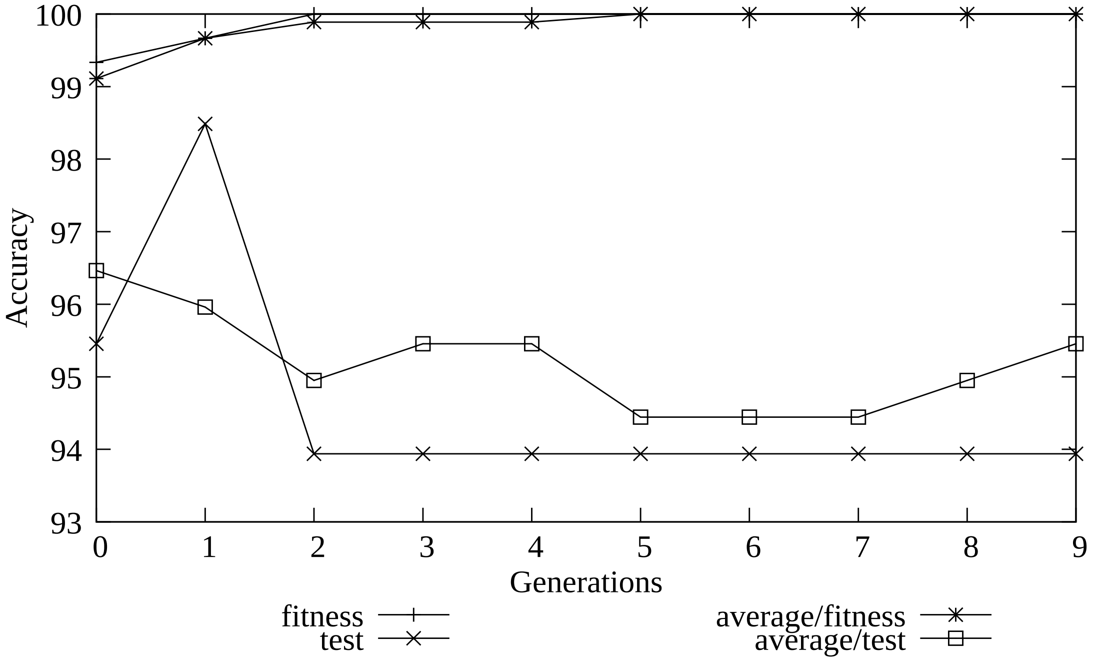


Fig. 4. Fitness evolution in the *dermatology* domain (best individual and the average of the three best individuals in each generation).

of the cross-validation (A_C). In order to compare two learning algorithms, we have used the relative improvement and the paired t-test as it is described next.

In order to obtain the relative improvement of a given classifier/ensembl (C_1) over another classifier/ensemble (C_2), we use $1 - error(C_1)/error(C_2)$. The classifier/ensembl error is given by $1 - A_C$. The average relative improvement (ARI) over all domains is computed using the geometric mean of the error reduction in the single domains as in [14]:

$$1 - geometricMean\left(\frac{error(C_1)}{error(C_2)}\right)$$

The statistical significanc is estimated using a paired t-test with $\alpha = 0.05$: $+/-$ in the results tables indicate that C_1 is better/worse than C_2. Table 4 shows the results of the individual classifier in all domains, and Table 5 shows the results of the ensemble methods, including *GA-Stacking* (best results are in bold).

Table 4
Accuracy rate of individual classifier

Domain	C4.5	Part	Nb	Ib1	Dt	Ds
dermatology	94.33	94.59	97.03	94.59	87.83	50.40
dna-splice	94.12	92.11	95.63	75.62	92.49	62.42
heart	74.00	82.00	**83.00**	78.00	76.33	70.67
ionosphere	90.14	89.30	82.82	87.61	89.30	82.82
musk	83.33	85.21	74.38	86.46	82.08	71.46
sonar	72.38	73.81	67.62	79.52	71.90	73.93

Table 5
Accuracy rate of ensemble methods

Domain	Bagging	Boosting	GA-Stacking
dermatology	94.59	97.03	**97.30**
dna-splice	94.56	94.43	**95.72**
heart	76.33	79.67	80.67
ionosphere	**92.11**	91.83	90.42
musk	87.29	**88.96**	83.96
sonar	80.00	79.05	**80.48**

Table 6
Accuracy relative improvement (in %) of the hypotheses found by *GA-Stacking* comparing them with the different single classifier and homogeneous ensemble methods (Bagging and Boosting) and the statistical significanc (+/− is better/worse, '.' not significant

Domain	C4.5	Part	Nb	Ib1	Dt	Ds	Bagging	Boosting
heart	25.64 .	−7.41 .	−13.73 .	12.12 .	18.31 .	34.09 +	18.31 +	4.92 .
sonar	29.31 .	25.45 .	39.71 +	4.65 .	30.51 .	26.79 .	2.38 .	6.82 .
musk	3.75 .	−8.45 .	37.40 +	−18.46 .	10.47 .	43.80 +	−26.23 .	−45.28 .
ionosphere	5.77 .	7.92 .	42.64 +	20.48 .	7.92 .	42.64 +	−24.96 .	−20.65 .
dermatology	52.39 .	50.01 +	9.12 .	50.01 +	77.78 +	94.54 +	50.01 .	9.11 .
DNA splice	27.27 +	45.82 +	2.16 .	82.45 +	43.10 +	88.62 +	21.39 +	23.16 +
Average	25.94	22.51	22.37	35.92	37.58	68.55	11.08	−1.36
win/lose	1+/0−	2+/0−	3+/0−	2+/0−	2+/0−	5+/0−	2+/0−	1+/0−

With the exception of the *heart* domain, the ensemble methods obtain better results than any single algorithm. Also, the ensembles of classifier generated from the *Stacking* configuratio found by *GA-Stacking* obtain better results in three out of the six domains, showing that it is a viable approach.

Table 6 shows a comparison among the ensemble of classifier found by *GA-Stacking*, the individual learning algorithms and *Bagging* and *Boosting*. The solutions generated from *GA-Stacking* improve, on average, over all the individual classifier and *Bagging*, but not *Boosting*. Nevertheless, if we analyze the statistical significanc of the results, the solutions of *GA-Stacking* are not significantl worse than any of the single classifiers or any of the methods for ensemble generation in any domain. On the other hand, *GA-Stacking* is better, significantl , than any of the single classifier or the techniques for ensemble generation in at least one domain, as it can be seen in the last row of Table 6.

Regarding the evolution of fitness Fig. 5 shows a comparison of the use of the same dataset both to generate the ensemble of classifier and to compute the fitnes value (**a**), and the use of a different dataset in order to evaluate the fitnes (**b**). As we can see, the accuracy rate over the test set increases if training and validation sets are used to generate the ensemble of classifiers Also, if we analyze the evolution of fitness the use of two datasets avoids, to a certain extent, the overfitting

Figures 6 and 7 show the evolution of fitnes together with the training and test accuracies in the domains used in these experiments. 100% accuracy is not reached in any training or fitnes dataset.

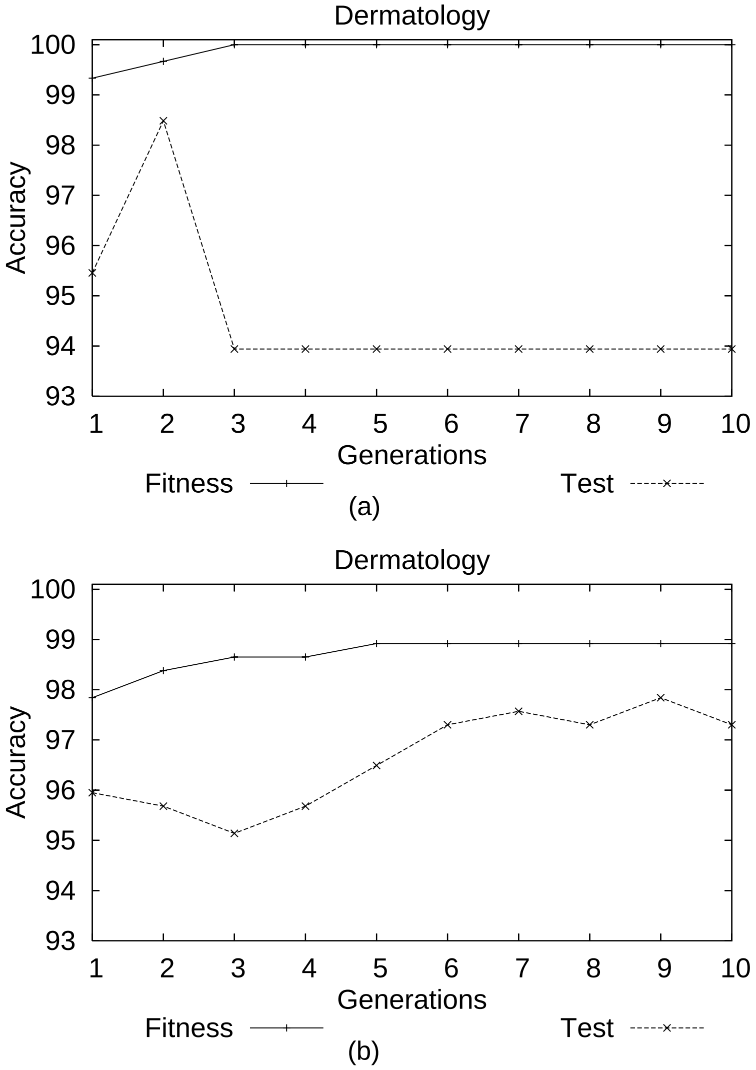


Fig. 5. Comparison of the evolution of fitnes using the same dataset for training and fitnes calculation (*a*) or different datasets (*b*) in the *Dermatology* domain.

Thus, overfittin seems to be prevented.

5. Evolving the GA-Stacking parameters

Once the viability of using GAs to obtain good *Stacking* configuration has been established, we intend to evolve, not only better *GA-Stacking* configurations but also the parameters of the base and meta-classifier present in that configuration This would include, for instance, the number of hidden neurons for ANN classifiers In Sections 5.1, 5.2 and 5.3 we detail all parameters and classifier that *GA-Stacking* is able to evolve. In Section 5.4 we show the experimental setup and in Section 5.5 we detail the results obtained when comparing different configurations

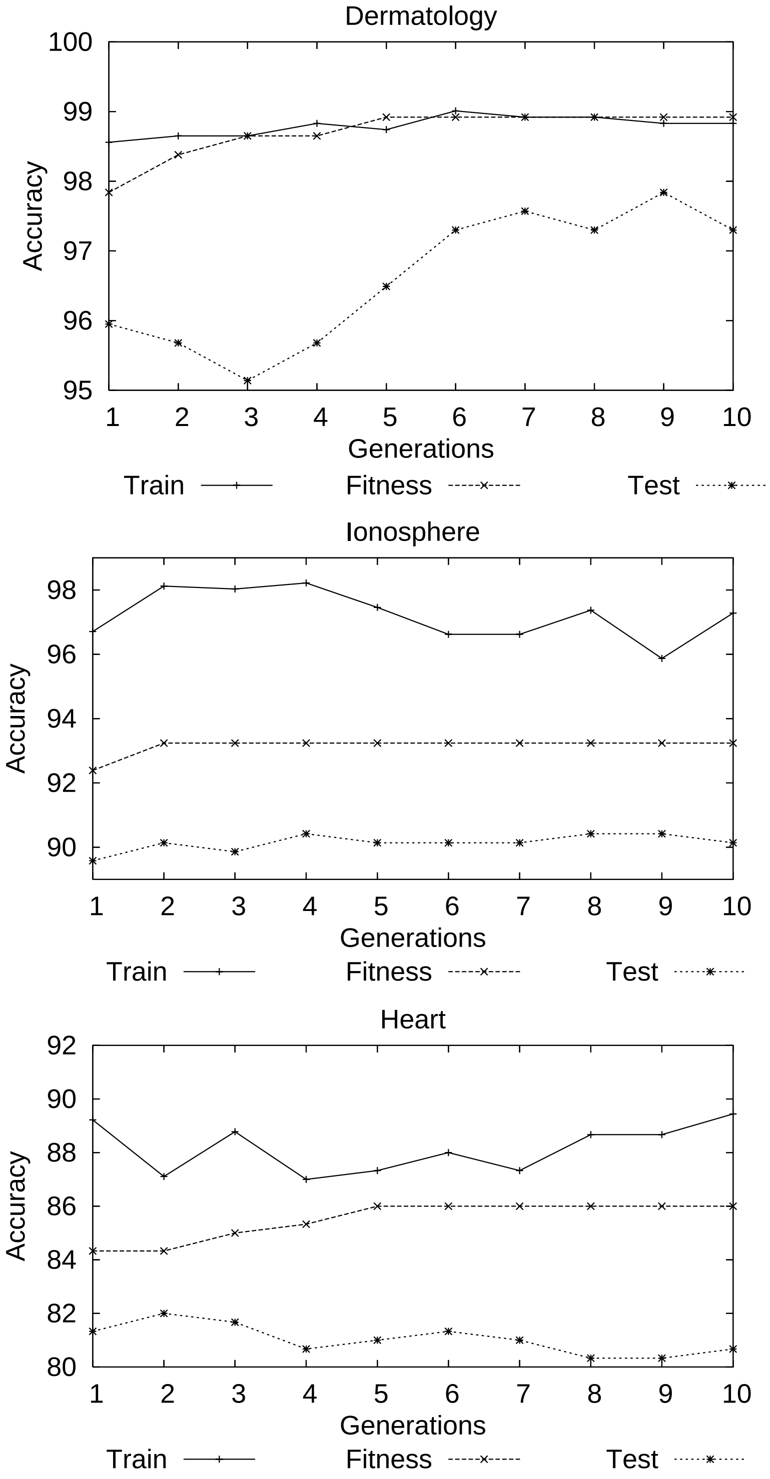


Fig. 6. Fitness evolution compared to the training and test accuracies in *dermatology, ionosphere* and *heart* domains.

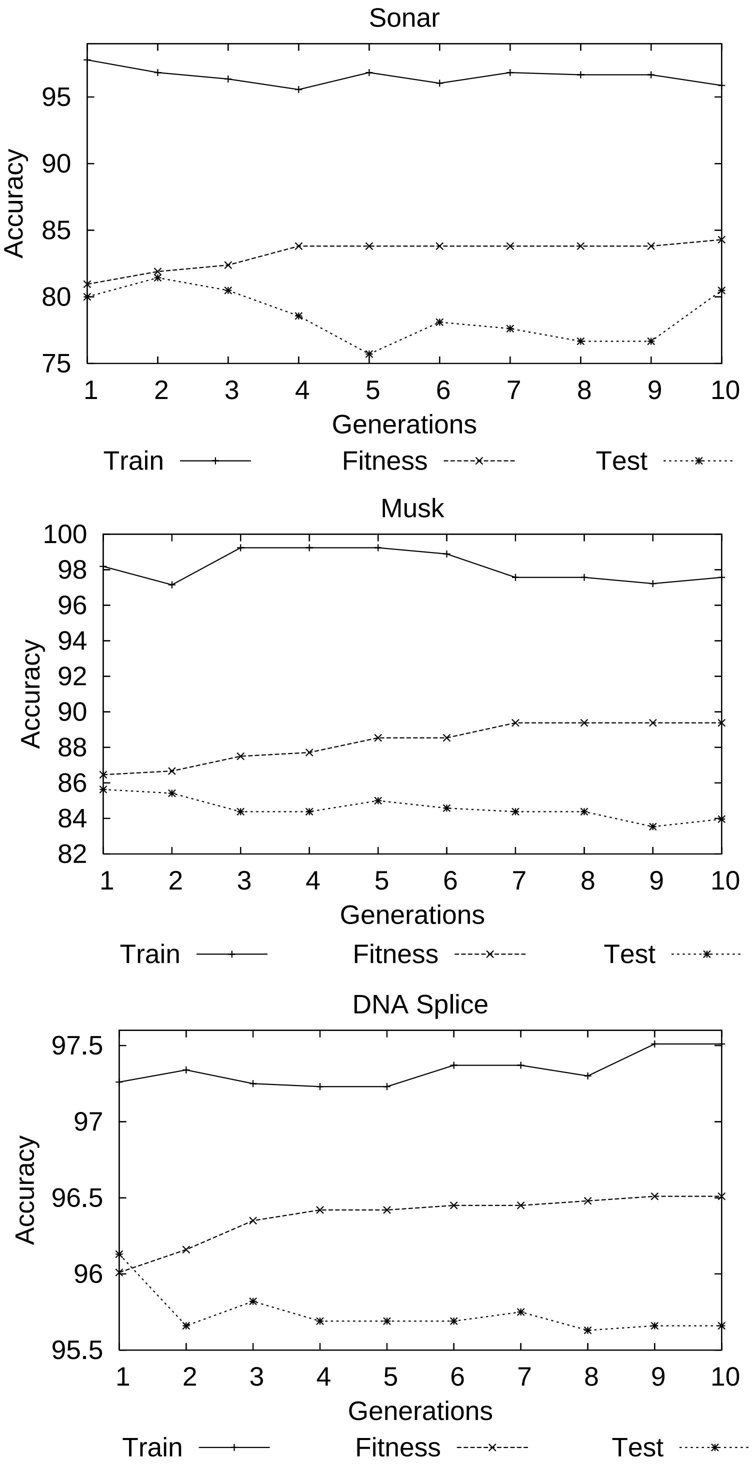


Fig. 7. Fitness evolution compared to the training and test accuracies in the *sonar, musk* and *DNA* domains.

5.1. Learning Algorithms

In order to extend the configuration space in which the GA carries out the search of the good *Stacking* parameters configurations we extended the number of possible learning algorithms that can be used to generate the ensemble members. These algorithms can be used to generate both the base classifier and the meta-classifier. In addition to the algorithms used in the first experiments, C4.5, *Naive Bayes*, IB*k*, PART, *Decision Stump* and *Decision* we have incorporated the following algorithms:

- *Random Forest* [4]. This algorithm constructs a Random Forest by combining many unpruned decision trees, RF.
- *Random Tree* [37]. This algorithm constructs a tree considering K random attributes at each node. It does not carry out any pruning, RT.
- MLR [34]. A multi-response lineal regression, MLR.
- MRMT [13]. A multi-response model tree, MRMT.
- K* [6]. An instance-based algorithm that uses a distance based on the entropy, K*.
- VFI [8]. An algorithm that generates a classifier that classifies an instance based on features intervals, VFI.
- *Conjunctive Rule*. This algorithms generates a simple conjunctive rules classifier, CR.
- *JRip* [7]. An algorithm that generates propositional rules, JRIP.
- *Nnge* [28]. It is a nearest neighbor algorithm that uses non-nested generalized exemplars, NNGE.
- *HyperPipes* [37]. It generates a classifier that constructs a HyperPipe for each category, which contains all the points of that category, HP.

In order to use the MLR and MRMT algorithms, we have used the classification via regression method implemented in WEKA (CVR). Thus, the selection of MLR or MRMT is a learning parameter of *GA-Stacking* (MLR by default).

5.2. Learning parameters of algorithms

In the related work on *Stacking*, the algorithms used by other authors to generate both the base-level classifiers and the meta-classifier employ the default learning parameters. Since these parameters can influence the results of each classifier, *GA-Stacking* searches in the parameter space of the learning algorithms, in addition to selecting the base and meta classifiers. In Appendix A we detail the learning parameters that we have used for every learning algorithm.

5.3. Other parameters

In addition to the learning algorithms and their parameters, there are other aspects related to *GA-Stacking*, such as the size of the ensemble of classifiers, the representation of candidate solutions and the GA parameters. In this section, we describe these parameters in more detail.

5.3.1. Size of the ensemble of classifiers

The number of algorithms that must be used to generate the level-0 classifiers varies from one study to another in the literature. For example, Ting and Witten [34] use three algorithms, whereas Seewald [32] uses six algorithms. Recently Džeroski and Ženko [14] use three and seven base classifiers to carry out the comparison among different methods of ensemble construction.

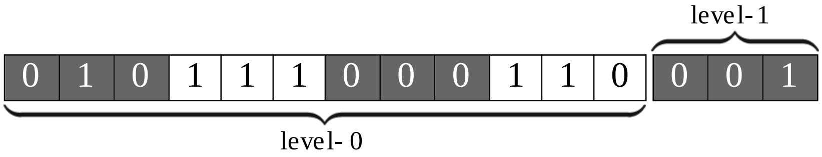


Fig. 8. Binary codification of the GAS5NPI configuration

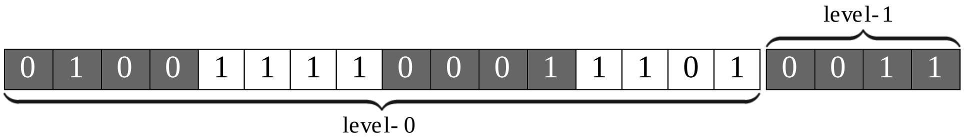


Fig. 9. Binary codification of the GAS5NPII configuration

Since there is no consensus regarding the number of base classifiers that must be members of the ensemble of classifiers, we include the ensemble size in the search space of *GA-Stacking*. We set "a priori" only the maximum number of base classifiers. Thus, we allow in the representation of chromosomes that one or more of the places in the chromosome for the base classifiers can contain the empty-classifier. In this way, the number of base-level classifiers can vary from 0 to the maximum number.

In order to determine the influence of the number of base classifiers in the ensemble, the maximum number of base classifiers has been set to four and ten. Therefore, we include the number of base classifiers used in previous studies in the literature.

5.3.2. Solutions representation

As we detailed in Section 3.2, *GA-Stacking* uses a binary codification in order to represent the solutions. The solution representation depends essentially on three factors previously mentioned: the number of learning algorithms available, the use of learning parameters of algorithms and the maximum number of base classifiers in the ensemble. Taking into account these factors, we have developed six *GA-Stacking* setups with the purpose of determining the best one of them. Next we detail the codification of each configuration.

- GAS5NPI[1]. The first *GA-Stacking* configuration in these experiments is similar to the one used in previous experiments (Section 4). The only difference is that now the non-presence of algorithms (i.e. the presence of empty-classifiers) in a given position of the chromosome is considered. The maximum number of possible base classifiers is four ($n = 4$), and the learning parameters of the algorithms are not included in the solution. The number of algorithms available is seven ($m = 7$). Figure 8 shows the solution codification of this *GA-Stacking* configuration.
  The number of genes in the chromosome is $T_c = 5$, and the size in bits is $T_b = 15$.
- GAS5NPII. This configuration is similar to the previous one, with the only difference that now the number of available algorithms is 15 ($m = 15$). Figure 9 displays the solutions codification in this configuration.

[1] *GA-Stacking* configurations names come from, *GA-Stacking*, "GAS", the maximum number of ensemble members, "5" and "11" (including the meta-classifier), the use or not of the learning parameters of the algorithms, "WP" (with parameters) and "NP" (no parameters), and the experiment version depending on the number of available algorithms, "I" or "II" (7 and 15 respectively).

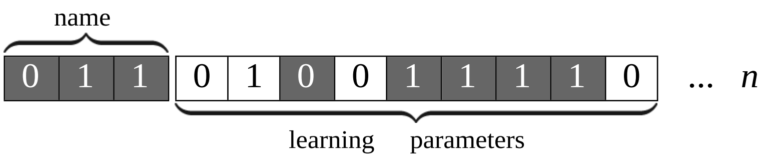


Fig. 10. A classifier binary codification within the GAS5WPI configuration

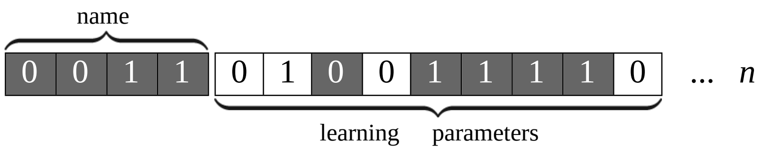
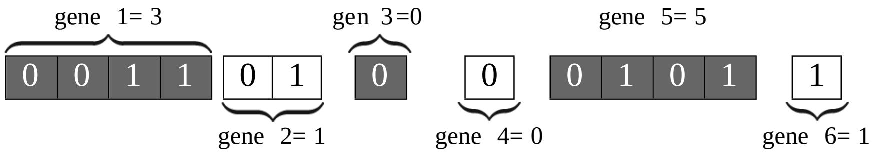


Fig. 11. A classifier binary codification within the GAS5WPII configuration



Gene	Value	Option
1	3	c4.5
2	1	-R
3	0	-
4	0	-
5	5	-C 0.35
6	1	-A

Fig. 12. The representation of C4.5 and its learning parameters using a binary codification

The number of genes in the chromosome is $T_c = 5$ and the size in bits is $T_b = 20$.

- GAS5WPI. In this *GA-Stacking* configuration we included the parameters of each learning algorithm in the genetic search. In these experiments $m = 7$ and $n = 5$. Due to the amount of available algorithms and their parameters, we have chosen a general representation of the solutions that incorporated most of the parameters available. In Appendix B we describe the selected learning parameters of each algorithm and the gene that represents them.
  Since the number and type of learning parameters varies from an algorithm to another, and there might be mutually excluding parameters for the same algorithm, we designed a general codification although this implies that in some cases some genes might not code any parameter of the learning algorithm.
  Figure 10 shows the codification of one algorithm within the chromosome. Six genes are used to represent a classifier, one for the name and five to code the learning parameters. The number of genes in the chromosome is $T_c = 30$ and the size in bits is $T_b = 60$.
- GAS5WPII. In this configuration $m = 15$. Therefore, the size in bits of the chromosome is $T_b = 65$ whereas the number of genes of the chromosome is $T_c = 30$ again. Figure 11 shows the codification and Fig. 12 shows an example of a classifier codification that will be generated from C4.5.
- GAS11NP. In this *GA-Stacking* configuration we extend the number of maximum base classifiers in the ensemble. In this case $m = 10$ and the used codification is similar to the one used by GAS5NPII.

Table 7
Datasets descriptions

Dataset	Attributes	Instances	Part A	Part B	Classes
australian	14	690	345	345	2
balance	4	625	312	313	3
breast-w	9	699	349	350	2
car	6	1728	1382	346	4
chess	36	3196	2876	320	2
diabetes	8	768	384	384	2
echo	6	132	66	66	2
german	20	1000	500	500	2
glass	9	214	107	107	6
heart	13	270	135	135	2
hepatitis	19	155	77	78	2
hypo	25	3163	2846	317	2
image	19	2310	1848	462	7
ionosphere	34	351	175	176	2
iris	4	150	75	75	3
soya	35	683	341	342	19
vote	16	435	217	218	2
wine	13	178	89	89	3

Regarding the length of the chromosome, it is given by $T_c = 11$ and $T_b = 44$ bits.

- GAS11WP. This is the last evaluated configuration of *GA-Stacking*. In this configuration the maximum number of base-level classifiers is ten, like in the previous configuration but in this case we included the learning parameters of each algorithm. The codification used is similar to the one used by GAS5WPII, but taking into account the new value of m. Regarding the chromosome length, it is given by $T_c = 66$ and its length in bits is $T_b = 143$.

5.4. Experimental Setup

In this section we describe in detail the configuration of the experiments carried out with the purpose of evaluating the different versions of *GA-Stacking*.

5.4.1. Domains

For the following experiments we have used 18 data sets from the UCI repository. These datasets have been used widely in other stacking comparative works. In order to evaluate the *GA-Stacking* approach in a different set of instances than the GA training set, we divided each data set into two parts: part A is used to evaluate and compare stacking approaches and part B is used to find the best stacking configuration that is, part B is used during the learning process. Table 7 shows the data sets characteristics.

5.4.2. GA parameters

Due to the increase in size of the configurations search space caused by the increase of available learning algorithms and their parameters, we augmented the size of the population and the number of generations. The parameters used for the GA in the following experiments are shown in Table 8.

5.4.3. Comparison of GA-Stacking versions

In order to evaluate the different versions of *GA-Stacking*, each configuration was executed three times in each domain with dataset B. The best individual found by *GA-Stacking* is taken as the best configuration of *Stacking*. This individual is composed of the name of the algorithms (and possibly

Table 8
GA parameters

Parameter	Value
Population	50
Generations	50
Elite rate	0.10
Cull rate	0.40
Mutation rate	0.10

its parameters) to be used as meta and base classifiers But the individual is not yet a proper stacking system, because these algorithms have yet to be trained. This is done with dataset A, as explained next. All the configuration found by the different versions of *GA-Stacking* are compared to each other by means of a 10-fold stratifie cross-validation using dataset A. A paired t-test is used to test the statistical significanc with a confidenc of 95%.

In order to compute the relative improvement obtained by the configuration of *Stacking* found by a given version of *GA-Stacking* over the rest of versions, we compute the Average Relative Improvement (ARI) for all domains used.

5.4.4. Other parameters

Again, we have used the learning algorithms implemented in WEKA (version 3.4). Also, in order to determine the statistical significanc of the results, we have used the paired t-test implemented in WEKA. As explained in Section 4.2, in order to avoid overfitting *GA-Stacking* carries out a cross-validation procedure to compute the fitnes of an individual. In the rest of experiments, a 2-fold cross-validation is used, as a trade-off between accuracy and computation time. The fitnes value is the average of the cross-validation process. Figure 13 shows the fitnes computation process.

The *Stacking* algorithm carries out another cross-validation-like process, in order to build the meta-level data. By default this process is a 10-fold cross-validation.

5.5. Empirical results

In this section we show the results of comparing different version of *GA-Stacking*. The accuracy of the ensemble of classifier constructed from the *Stacking* configuration found by *GA-Stacking* can be seen in Table 9. In bold we show the best results.

Nevertheless, if we want to compare the *Stacking* configuration performance found by the different versions of *GA-Stacking*, Table 10 is more interesting: it reflect the ARI of X over Y for each solution pair X and Y. Also, this table shows the win/loss summary computed from a paired t-test (10-fold cross-validation). The win/loss summary represents the number of domains where the algorithm in the row X of the table is significantl better (or worse) than the algorithm in column Y. The column "Total" adds all the win:loss summaries in the row.

Analyzing Table 10 we can see that the difference in the ARI in all domains among different solutions found by *GA-Stacking* is low (between -1,86% and 1,82%). Nevertheless, if we analyze the last column, we can see that the *GA-Stacking* versions that include the parameters in the genetic search are better than the others. In other words, GAS5WPI $(+8)$[2], GAS11WP $(+7)$ and GAS5WPII $(+5)$, are better than GAS5NPI (-3), GAS5NPII (-4) y GAS11NP (-13). Thus, the use of the parameters of the algorithms, by extending the search space for the GA, obtains better results than not using them.

[2] The number in brackets represents the absolute difference between wins and loses

Table 9
Results of the *Stacking* configuration found by different versions of *GA-Stacking*

Domains	GaS5npI	GaS5npII	GaS5wpI	GaS5wpII	GaS11np	GaS11wp
australian	86.93	87.21	**88.11**	87.22	88.09	88.08
balance	90.08	94.23	**94.54**	92.95	94.22	94.23
breast	95.42	95.71	**97.14**	95.99	96.57	96.28
car	96.02	95.15	97.03	97.18	95.88	**97.47**
chess	99.27	99.17	99.23	99.24	**99.34**	99.27
diabetes	73.41	73.93	**75.24**	73.93	74.99	**75.24**
echo	86.67	84.17	87.50	**90.00**	84.17	**90.00**
german	74.00	**75.40**	71.00	75.00	71.00	73.60
glass	62.45	63.45	67.27	**76.00**	66.27	74.55
heart	82.75	**83.57**	82.03	82.80	75.44	82.75
hepatitis	**80.89**	**80.89**	79.64	78.57	80.71	79.64
hypho	**99.30**	99.12	99.26	98.98	98.95	99.23
images	97.94	97.89	97.56	**98.16**	97.62	98.10
ionosphere	89.02	90.23	**91.93**	89.61	90.26	90.23
iris	93.57	92.32	93.57	92.32	94.82	**97.32**
sonar	**97.18**	94.27	91.36	93.36	**97.18**	95.27
vote	**95.87**	93.07	95.84	**95.87**	91.75	94.00
wine	94.44	96.67	**98.89**	97.78	94.31	92.22

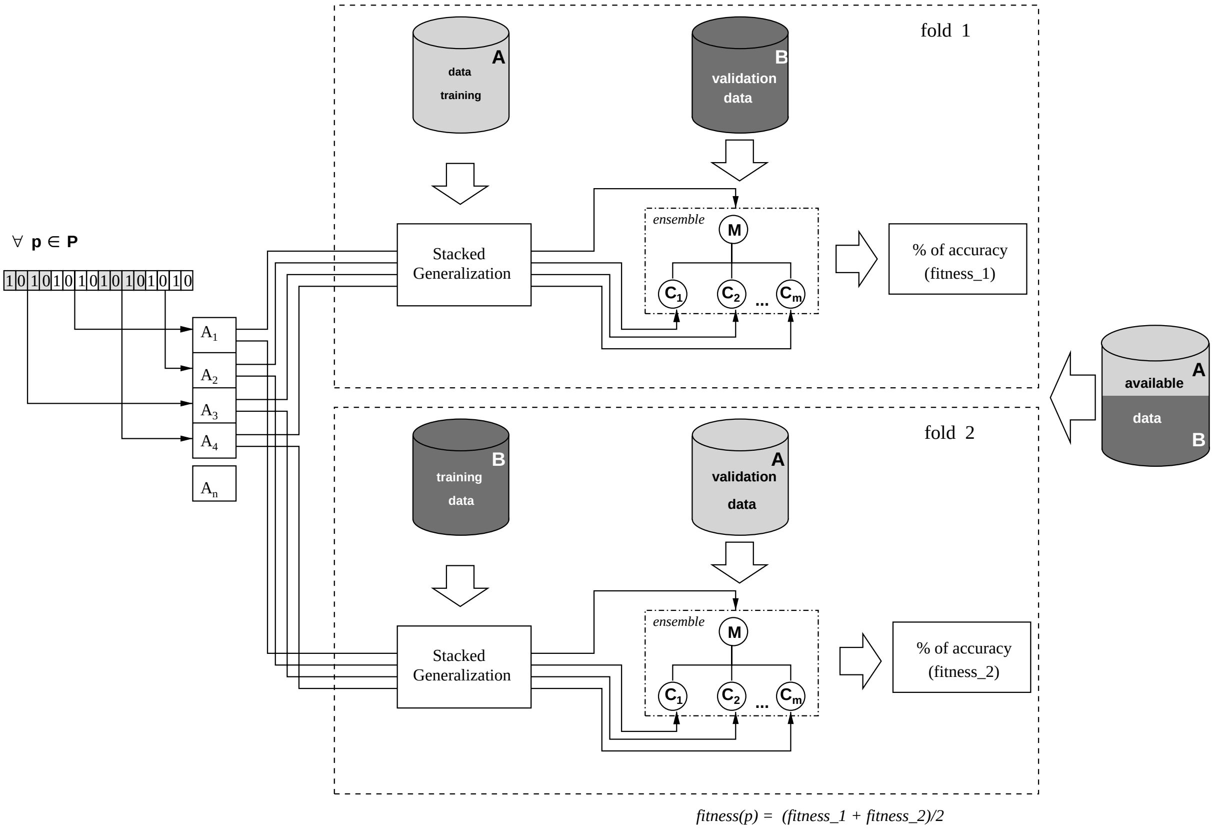


Fig. 13. Fitness computation process using a 2-fold cross-validation.

Table 10

Relative improvement of *Stacking* configurations found by different versions of *GA-Stacking*. Elements in X row and Y column show the relative improvement of X over Y (in percentage) and the win/loss summary (according to 1×10 t-test)

	GAS5NPI	GAS5NPII	GAS5WPI	GAS5WPII	GAS11NP	GAS11WP	Total
GAS5NPI		0.12 1:1	0.80 1:3	1.46 2:3	−0.22 3:1	1.60 1:3	8:11
GAS5NPII	−0.14 1:1		0.66 0:2	1.33 0:2	−0.36 2:0	1.47 0:2	3:7
GAS5WPI	−0.81 3:1	−0.67 2:0		−0.67 2:0	−1.03 4:1	0.81 0:1	11:3
GAS5WPII	−1.48 3:2	−1.35 2:0	−0.67 0:2		−1.71 4:1	0.14 2:1	11:6
GAS11NP	0.22 1:3	0.36 0:2	1.02 1:4	1.68 1:4		1.82 0:3	3:16
GAS11WP	−1.63 3:1	−1.49 2:0	−0.82 1:0	−0.14 1:2	−1.86 3:0		10:3

Also, the increase of the number of available algorithms (I vs. II) and the increase of the maximum number of classifiers in the ensemble does not seem to improve very much the solution fitness. This can be explained by the significant increase in the search space, and as we carried out 50 generations of 50 individuals, the number of explored configurations is relatively small compared to the search space.

If the use of the parameters of the algorithms is combined with the increase of the number of available algorithms and the maximum number of members in the ensemble (GAS11WP/10:3), we obtain results that are comparable with the best one of the *GA-Stacking* versions (GAS5WPI/11:3). In other words, if we compare GAS5WPI and GAS11WP, both are significantly worse three times than another *GA-Stacking* configuration. Nevertheless, GAS5WPI has more wins than GAS11WP (11 vs. 10), but if we compare the significant differences between both of them, GAS11WP is the best one of the two versions (+1). Therefore, we select the maximal *GA-Stacking* version for the rest of the comparisons.

6. *GA-Stacking* performance

In order to estimate the *GA-Stacking* performance we have carried out a comparison among *GA-Stacking* (GAS11WP version) and the best generators of homogeneous ensemble of classifiers: *Boosting* and *Bagging*, in addition to other algorithms for ensemble generation based on *Stacking*.

6.1. Experimental setup

In this section we detail the setup of the experiments with the purpose of comparing *GA-Stacking* with other methods of ensemble generation.

6.2. Domains

In these experiments we have used a subset of the domains used for evaluating different versions of *GA-Stacking* (Section 5.4.1), and also used in previous related work. The selection of these domains was carried out taking into account the number of classes, instances and attributes with the purpose of using representative domains. The selected domains are: *australian, balance, car, diabetes* and *glass*.

6.3. GA parameters

As we mentioned earlier, the *GA-Stacking* configuration used in these experiments is GAS11WP. This *GA-Stacking* version can have up to 10 base-level classifiers and includes the parameters of the algorithms used to generate the classifiers. The GA parameters are the same than in a previous section (see Table 8).

6.4. Learning algorithms

We have used two categories of algorithms in these experiments. The first of these categories includes the algorithms for generating the individual classifiers. These algorithms are used by *GA-Stacking* and the other algorithms for ensemble generation in order to obtain the ensemble members. The other category includes the ensemble of classifiers algorithms and classifier combination methods.

The learning algorithms for generating the individual classifiers are the same ones used in the *GA-Stacking* versions comparison (see Section 5.1). Regarding the ensemble generation algorithms and classifier combination methods, we have used the following ones:

- VOTE: Method for combining classifiers using unweighted average of probability estimates.
- BESTCV: Method for selecting a classifier from among several ones using cross-validation on the training data.
- *Bagging*. The base algorithm used is C4.5.
- *Boosting* (*AdaBoostM1*) The base algorithm used is C4.5.
- SMRMT: *Stacking* with multi-response model trees. Given that a study uses groups of three and seven algorithms to generate the base-level classifiers [14], we have used two versions of this algorithm. The first group is composed by C4.5, IB*k* and *Naive Bayes*, while we add to the second group *K**, *Decision Table*, *MLR* and *Kernel Density*. In [14] a comparative among the ensemble construction methods based on *Stacking* was carried out. They concluded that SMRMT has a greater performance than any of the other *Stacking* based methods.
- SCMLR (*StackingC*): *Stacking* with a reduced number of meta-level attributes independently of the number of classes. We have used three groups of algorithms to generate the base-level classifiers. First, we have used the classifiers used by Seewald [32] (*Decision Table*, C4.5, *Naive Bayes*, *Kernel Density*, *MLR* and *K**). Second, we have used the groups used by Džeroski y Ženko [14] described in the previous item.
- SMLR: *Stacking* with multi-response linear regression proposed by Ting y Witten [34]. Like in the other *Stacking*-based algorithms we have used the groups of three and seven algorithms previously described.

6.5. Comparing results

In order to evaluate and compare results of different algorithms, we estimated the classification accuracy with a stratified ten-fold cross-validation. In order to compare the results obtained by *GA-Stacking* with each one of the used algorithms, we have computed the relative improvement of *GA-Stacking* over other algorithms in each domain. In addition, a paired t-test has been carried out (1 × 10) with the purpose of determining if the improvement obtained by *GA-Stacking* is statistically significant.

6.6. Experimental results

The accuracy percentage of the different learning algorithms used in these experiments are reflected in Table 11. As it can be seen, the results obtained by the *Stacking* configuration found by *GA-Stacking* are better in four out of the five domains. Also, in order to compare the *GA-Stacking* performance with each one of the used algorithms, in Table 12 we show the relative improvement that obtains *GA-Stacking* when comparing it with the other algorithms and their statistical significance based on a paired t-test. The results will be analyzed in more detail next.

Table 11
Accuracy rate (%) of ensemble-generation methods and combining-classifier methods

Algorithms	australian	balance	car	diabetes	glass
GaS11wp	**87.10**	**96.96**	**99.42**	75.25	**78.94**
Boosting	83.91	79.21	96.18	71.87	73.33
Bagging	85.80	82.25	93.29	76.30	72.90
BestCV	82.32	86.88	93.75	70.94	70.50
Vote	86.96	86.88	94.85	76.56	76.62
Smrmt3	85.65	92.97	99.13	76.02	68.18
Smrmt7	85.80	92.50	98.96	76.29	73.74
Scmlr5	85.94	92.50	95.02	**77.07**	76.08
Scmlr3	85.51	90.56	93.92	76.15	70.11
Scmlr7	85.65	89.92	95.02	76.94	76.56
Smlr3	85.94	89.92	94.73	76.16	66.75
Smlr7	85.51	89.28	95.49	76.94	76.06

Table 12
Relative improvement (%) of *GA-Stacking* over other ensemble and combined methods. $+/-$ means better/worse, '.' means there is no significant difference

Algorithms	australian	balance	car	diabetes	glass	MRI[3] win/loss
Boosting	19.82 +	85.40 +	84.84 +	12.00 +	21.02 .	58.48 4+/0-
Bagging	9.18 .	82.90 +	91.38 +	−4.43 .	22.28 +	59.52 3+/0−
BestCV	27.05 +	76.85 +	90.74 +	14.82 +	28.61 +	60.58 5+/0−
Vote	1.11 .	76.86 +	88.76 +	−5.58 .	9.91 .	52.39 2+/0−
Smrmt3	10.10 .	56.83 +	33.35 .	−3.22 .	33.81 +	29.29 2+/0−
Smrmt7	9.18 +	59.51 +	44.45 .	−4.37 .	19.80 +	29.76 3+/0−
Scmlr5	8.25 .	69.88 +	88.37 +	−7.95 .	11.95 .	50.22 2+/0−
Scmlr3	11.00 +	67.82 +	90.47 +	−3.77 .	29.54 +	54.29 4+0−
Scmlr7	10.10 .	69.88 +	88.37 +	−7.31 .	10.16 .	50.28 2+/0−
Smlr3	8.25 .	69.87 +	89.01 +	−3.80 .	36.65 +	54.28 3+/0−
Smlr7	11.00 +	71.67 +	87.17 +	−7.34 .	12.03 .	50.23 3+/0−

6.6.1. *Analysis*

First we will analyze the results of *GA-Stacking* comparing it with the homogeneous ensemble-construction methods *Bagging* and *Boosting*. As we can see in Table 12, in four of the five used domains *GA-Stacking* is significantly better than Boosting. In the case of Bagging, *GA-Stacking* is significantly better in three of the five domains. The relative improvement in both cases surpasses 58%.

When comparing *GA-Stacking* with the combination methods by votes (Vote) and the best classifier selection by cross-validation (BestCV), we have also obtained different results. If we compare *GA-Stacking* with BestCV, *GA-Stacking* is significantly better in all domains with a relative improvement over 60%. Nevertheless, when comparing it with the voting scheme, the significant improvements are reduced to two of the five domains, but the ARI continues to be above 52%.

Now we will analyze the results of comparing *GA-Stacking* with other *Stacking*-based algorithms (Smrmt, Scmlr, Smlr). First, when comparing *GA-Stacking* with the two versions of Smrmt (three and seven base classifiers) we can observe that the number of times in which *GA-Stacking* is significantly better varies according to the number of base-level classifiers that are members of the ensembles generated by Smrmt. *GA-Stacking* wins one more time if it compares with Smrmt7 instead of with Smrmt3. And the relative improvement in both is over 29%.

[3] Mean of relative improvement.

Table 13
Average number of base classifiers in the solutions found by *GA-Stacking*

Domain	# of base classifier
australian	9.3
balance	9.4
car	9.5
diabetes	9.4
glass	9.6

In relation to the comparisons with the three versions of SCMLR (three, five and seven base classifiers) the number of domains in which *GA-Stacking* is significantly better are similar in the SCMLR5 and SCMLR7 versions (2+). Nevertheless, the number of domains in which *GA-Stacking* is significantly better doubles when compared with the version of only three base classifiers. The ARI in all cases is over 50%.

Finally, if we compare the results of *GA-Stacking* with the results obtained by SMLR in its two versions (three and seven base classifiers), the ARI is higher than 50%. In addition in three of the five domains, *GA-Stacking* is significantly better than SMLR.

These results demonstrate that the number of base classifiers influence the accuracy, as also does the meta-classifier employed. Given that *GA-Stacking* can automatically select these, among other parameters, and do not incur in overfitting, it has an advantage over the manually preconfigured systems. Also, as we can see in the last row of Table 12, *GA-Stacking* is not significantly worse than any other algorithm in any domain.

6.6.2. Description of the solutions

With the purpose of observing the structure of the *Stacking* configurations found by *GA-Stacking*, we have analyzed the best individuals of each fold in the stratified cross-validation. That is to say, of the three executions of *GA-Stacking* in each fold of the cross-validation, we have analyzed the individual with the greatest value in the fitness function.

Table 13 shows the average number of base classifiers in the solutions. As we can see, the number of base classifiers varies between 9 and 10, where 10 is the maximum number of base classifiers that allows the GAS11WP *GA-Stacking* configuration. On the one hand, it should be expected that the GA tends to use as many base-classifiers as possible, because the probability of the empty classifier is very small, so there is a natural bias towards complex configurations. However, in this case, this bias has no negative influence on the overall accuracy, because this *GA-Stacking* configuration is the best one, in spite of it allowing more base-classsifiers than the rest of *GA-Stacking* configurations.

Regarding the algorithms automatically selected by *GA-Stacking* to generate the base-level classifiers, Fig. 14 shows the number of folds out of 10 (and above 6), where a particular classifier appears as a base classifier in the optimal configuration. As it can be seen, in each domain there are between three and four algorithms that are present at least in 7 of the 10 folds of the cross-validation. For instance, in the *car* domain, there are three base classifiers (K*, MRMT, and PART) that *GA-Stacking* considers to be necessary for that domain, and this fact is automatically determined by our system.

Figure 15 shows the algorithms selected to be the meta-classifier in each domain. As it can be seen, the best individuals of each fold tend to use the same algorithm to generate the level-1 classifier. For example, in the *balance* and *car* domains, the algorithm selected to generate the meta-level classifier in all cross-validation folds is the *Random Forest*, while in the *australian* domain *NaiveBayes* is the

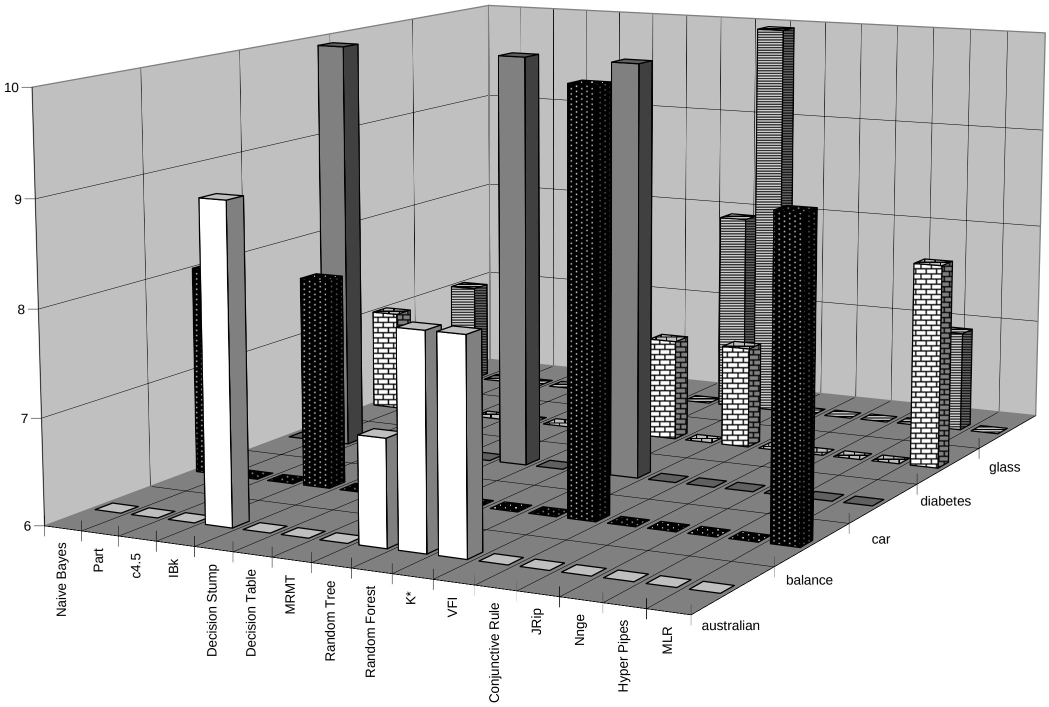


Fig. 14. Number of folds (six or more) in the cross-validation in which a specific classifier appears in each domain.

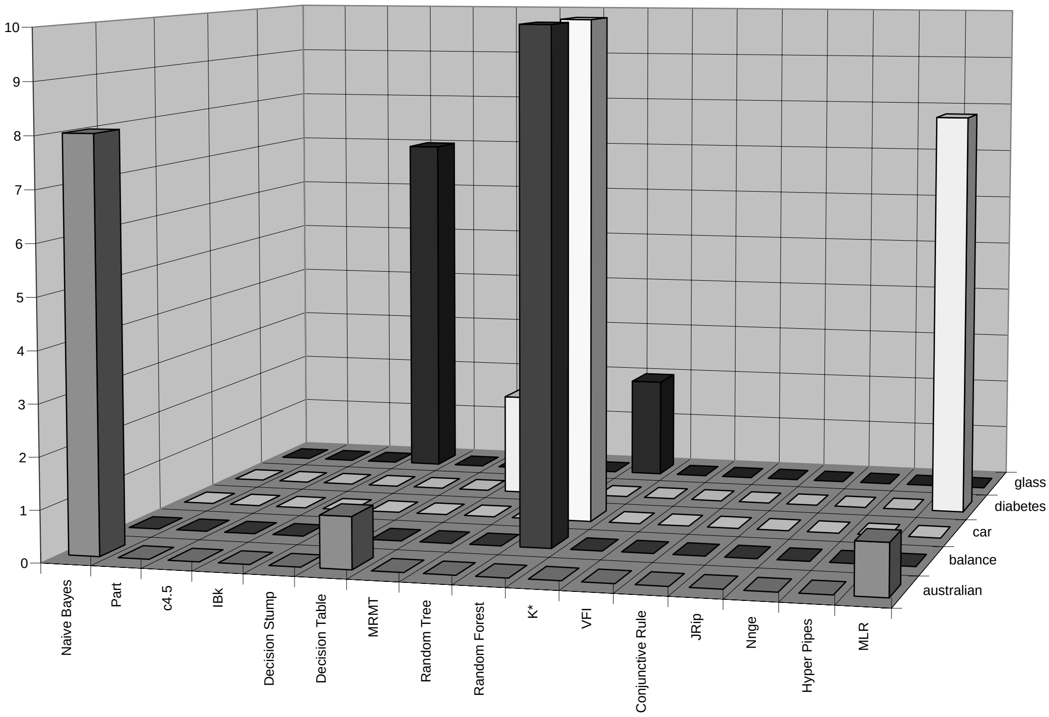


Fig. 15. Number of cross-validation folds in which a particular algorithm appears at the meta-level.

preferred option most of the times. Other researchers claim that *MLR* [34,32] or *MRMT* [13] yield good results when used to build the meta-classifier regardless of the base algorithms and the domain. However, the algorithms selected by *GA-Stacking* to generate the meta-classifier vary according to the domain. For example, *Naive Bayes* (8/10) for *australian*, *Random Forest* (10/10) for *car* and *balance* domains, *MLR* (8/10) in *diabetes* and *Ibk* (7/10) in *glass*.

This shows two key aspects: in each domain, a specific algorithm is preferred over the rest to be the meta-classifier, and this clearly highlights the advantage of *GA-Stacking* over the rest of algorithms in that it can select automatically the best algorithm for each domain.

6.6.3. Evolution of the fitness

Another aspect to take into account in these experiments is the behavior of the fitness function. In Fig. 16 we can see the evolution of the fitness function for each one of the used domains.

The fitness behavior is very similar in all domains: the largest increase takes place in the first generations and soon it stays in constant growth, even when arriving at the last generation. This indicates an evolution in the solutions found by *GA-Stacking*. In addition, given that fitness keeps increasing, it would be useful to increase the number of generations for the purpose of finding better individuals.

7. Conclusions and future work

In this paper, we have presented the *GA-Stacking* algorithm, an approach to find good *Stacking* configurations by means of genetic search. *GA-stacking* not only determines which meta-level, and which (and how many) base classifiers must be present, but also their parameters. The main advantage of *GA-Stacking* over other techniques is its flexibility and extensibility. It can easily incorporate new learning algorithms, and it is not restricted by “a priori” assumptions. Another advantage of *GA-Stacking* is that the solutions that it finds are domain dependent. Thus, *GA-Stacking* adapts the *Stacking* configuration to the domain biases and characteristics, while all the other approaches use the same *Stacking* configuration independently from the domain in which they are applied.

We would like to highlight that in addition to *GA-Stacking* being able to find accurate *Stacking* configurations, it also provides some automation of the data mining process. Typically, *Stacking* has to be configured by hand. Even though there are some general guidelines on how to build *Stacking* configurations, in some cases a process of trial and error is required from the user for selecting the base and the meta learning classifiers and also their parameters. With *GA-Stacking*, the intervention of the user is reduced to selecting the set of available classifiers. Our experiments show that even if this set is large, results do not degrade.

Empirical results in domains currently used in this field show that *GA-Stacking* is comparable to the best results reported so far, and it is never significantly worse than the other tested systems (with the advantage that paremeters such as the number of base classifiers or the algorithms available to be used as base, need not be specified in advance). With respect to accuracy, if we add the relative improvements over the other systems across all the domains, positive differences are always obtained, quite large in some cases. Therefore, if accuracy is very important for a given task, we believe *GA-Stacking* should be used.

However, *GA-Stacking* requires a longer execution time than the rest of approaches, because several generations of individuals must be evaluated in order to obtain a good individual. Even if for most domains this is not crucial, given that most classification tasks do not require to work in real time, it could be relevant for others. Also, adding incremental capabilities can be hard to implement. Finally,

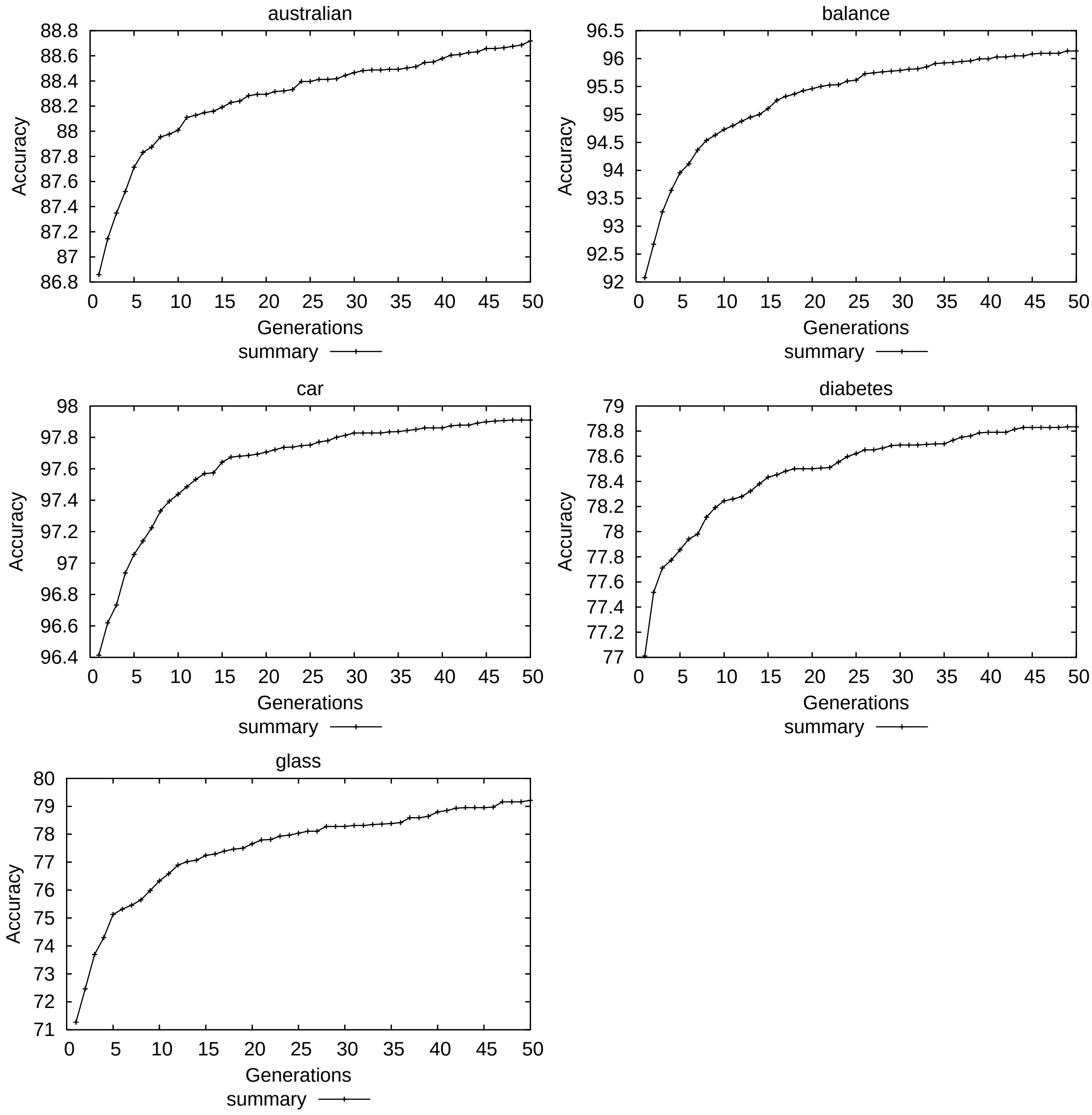


Fig. 16. Evolution of the fitness function in each domain.

another drawback common to all *Stacking* research is the understandability by the human of the final classifier scheme.

Accuracy is not always the only useful feature to consider, although it is usually the only aspect considered in *Stacking* research. Configuration size, on-line classification speed, etc., can also be relevant issues in some domains. In the future, we plan to investigate the flexibility of *GA-Stacking* so that these issues are also considered. For instance, we intend to add selective pressure towards accurate, but also simple (few base classifiers) *Stacking* configurations. Also, we would like to add information to the chromosome about the meta-level data to be used, so that the *Stacking* configuration can use the most appropriate representation for each domain.

Acknowledgements

This work has been partially supported by the Spanish MCyT under projects TRA2007-67374-C02-02 and TIN-2005-08818-C04. Also, it has been supported under MEC grant by TIN2005-08945-C06-05. We thank anonymous reviewers for their helpful comments.

Appendices A Parameters of the algorithms

We present next the parameters of the learning algorithms used by *GA-Stacking* in order to generate the ensemble of classifiers (See Table 14.)

Table 14

Algorithm	Option	Description
Algorithm	Option	Description
Naive Bayes	–K	Use a kernel estimator for numeric attributes rather than a normal distribution.
PART	–B	Use binary splits on nominal attributes when building the partial trees.
	–C	Set the confidence factor used for pruning (default: 0.25).
C4.5	–R	Use reduced error pruning. No subtree raising is performed.
	–U	Use unpruned tree.
	–B	Use binary splits for nominal attributes.
	–S	Do not perform subtree raising.
	–C	Set confidence threshold for pruning. (Default: 0.25).
	–A	If set, Laplace smoothing is used for predicted probabilites.
IB*k*	–F	Neighbors will be weighted by their similarity when voting. (default equal weighting).
	–D	Neighbors will be weighted by the inverse of their distance when voting. (default equal weighting)
	–N	Turns off normalization.
	–S	When K is selected by cross-validation for numeric class attributes, minimize mean-squared error. (default mean absolute error)
	–K	Set the number of nearest neighbors to use in prediction (default 1).
Decision Stump	–	It does not have configurable parameters.
Decision Table	–I	Use nearest neighbor instead of global table majority.
Classification by Regression	-W	Specify the name of a numeric predictor as the basis for the classifier.
Random Tree	–S	Sets the random number seed used for selecting attributes. (default: 1).
	–K	Sets the number of randomly chosen attributes at each node.
	–M	Sets the minimum total weight of the instances in a leaf.
Random Forest	–I	Set the number of trees in the forest (default 10).
	–K	Set the number of features to consider. If < 1 (the default) will use $logM$+1, where M is the number of inputs.

Table 14 Continue.

Algorithm	Option	Description
	–S	Random number seed (default 1).
K*	–B	Manual blend setting (default 20%).
	–E	Enable entropic auto-blend setting.
	–M	Specify the missing value treatment mode (default average).
VFI	–B	Set exponential bias towards confiden intervals. (default 1.0).
	–C	Do not weight voting intervals by confidence
Conjunctive Rule	–S	Set the seed of randomization (default 1).
	–R	Set if NOT uses randomization (default:use randomization).
	–E	Set whether consider the exclusive expressions for nominal attributes (default false).
	–N	Set number of folds for REP One fold is used as pruning set. (default 3).
	–M	Set the minimal weights of instances within a split. (default 2.0).
JRIP	–O	Set the number of runs of optimizations. (Default: 2).
	–P	Whether NOT use pruning (default: use pruning).
	–E	Whether NOT check the error rate $>= 0.5$ in stopping criteria (default: check).
	–F	Set number of folds for REP One fold is used as pruning set. (default 3).
	–N	Set the minimal weights of instances within a split. (default 2.0).
NNGE	–G	Set the number of attempts of generalization (default 5).
	–I	Set the number of folders to use in the computing of the mutual information (default 5).
Hyper Pipes	–	It does not have configurabl parameters.

Appendices B Genetic representation of the learning parameters

Next, we present the correspondence between the parameters of the algorithms used by *GA-Stacking* and the gene that represents them within the binary codificatio (See Table 15).

Table 15

Algorithm[4]		NA[5]	NB	PART	c4.5	IB*k*	DS	DT	CPR[6]	RT	RF	K*	VFI	CR	JRIP	NNGE	HP
	Valor	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Gene # 2	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	1	–	–	–	–R	–F	–	–	–	-S 1	-S 1	-B 10	-B 2.0	-S 2	-O 3	-G 2	-
	2	–	–	–	-U	-D	–	–	–	-S 2	-S 2	-B 30	-B 3.0	-S 3	-O 4	-G 3	–
	3	–	–	–	–	–	–	–	–	-S 3	-S 3	-B 40	-B 4.0	-S 4	-O 5	-G 4	–
Gene # 3	0	–	–	–	–	–	–	–	-W A	–	–	–	–	–	–	–	–
	1	–	-K	-B	-B	-N	–	-I	-W B	-K 10	-K 10	-E	-C	-R	-P	–	–
Gene # 4	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	1	–	–	–	-S	-S	–	–	–	–	–	–	–	-E	-E	–	–
Gene # 5	0	–	–	-C 0.10	-C 0.10	-K 1	–	–	–	–	–	–	–	–	–	–	–
	1	–	–	-C 0.15	-C 0.15	-K 2	–	–	–	-K 5	-I 2	-M m	-	-N 4	-F 4	-I 2	–
	2	–	–	-C 0.20	-C 0.20	-K 3	–	–	–	-K 10	-I 4	-M n	–	-N 5	-F 5	-I 3	–

[4] It is represented by Gene #1.

[5] No Algorithm.

[6] The gene #3 options $-A$ y $-B$ correspond to use MLR and MRMT respectively.

Table 15 Continue.

Algorithm		NA	NB	PART	C4.5	IB*k*	DS	DT	CPR	RT	RF	K*	VFI	CR	JRIP	NNGE	HP
	Valor	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	3	-	–	-C 0.25	-C 0.25	-K 4	–	–	–	-K 15	-I 6	-M a	-	-N 6	-F 6	-I 4	–
	4	–	–	-C 0.30	-C 0.30	-K 5	–	–	–	-K 20	-I 8	-M d	-	-N 7	-F 7	-I 5	–
	5	–	–	-C 0.35	-C 0.35	-K 6	–	–	–	-K 25	-I 10	–	–	-N 8	-F 8	-I 6	–
	6	–	–	-C 0.40	-C 0.40	-K 7	–	–	–	-K 30	-I 12	–	–	-N 9	-F 9	-I 7	–
	7	–	–	-C 0.45	-C 0.45	-K 8	–	–	–	-K 35	-I 14	–	–	-N 10	-F 10	-I 8	–
	8	–	–	-C 0.50	-C 0.50	-K 9	–	–	–	-K 40	-I 16	–	–	–	–	-I 9	–
	9	–	–	-C 0.50	-C 0.50	-K 10	–	–	–	-K 45	-I 18	–	–	–	–	-I 10	–
	10	–	–	-C 0.50	-C 0.50	-K 11	–	–	–	-K 50	-I 20	–	–	–	–	–	–
	11	–	–	-C 0.50	-C 0.50	-K 12	–	–	–	-K 55	-I 22	–	–	–	–	–	–
	12	–	–	-C 0.50	-C 0.50	-K 13	–	–	–	-K 60	-I 24	–	–	–	–	–	–
	13	–	–	-C 0.50	-C 0.50	-K 14	–	–	–	-K 65	-I 26	–	–	–	–	–	–
	14	–	–	-C 0.50	-C 0.50	-K 15	–	–	–	-K 70	-I 28	–	–	–	–	–	–
	15	–	–	-C 0.50	-C 0.50	-K 16	–	–	–	-K 75	-I 30	–	–	–	–	–	–
Gene # 6	0	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	1	–	–	–	-A	–	–	–	–	-M 5	–	–	–	-M 3	-N 3	–	–

References

[1] D. Aha, D. Kibler and M. Albert, Instance-based learning algorithms, *Mach Learn* **6**(1) (Jan 1991), 37–66.

[2] C. Blake and C. Merz, UCI repository of machine learning databases. databases http://www.ics.uci.edu/∼mlearn/MLRepository html, 1998.

[3] L. Breiman, Bagging predictors, *Mach Learn* **24**(2) (1996), 123–140.

[4] L. Breiman, Random forests, *Mach Learn* **45**(1) (2001), 5–32.

[5] K. Cherkauer, Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks, in: *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, 1996, pp. 15–21.

[6] J. Cleary and L. Trigg, K*: an instance-based learner using an entropic distance measure, in: *Proceedings of the 12th International Conference on Machine Learning*, 1995, pp. 108–114.

[7] W. Cohen, Fast effective rule induction, in: *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[8] G. Demiroz and H. Guvenir, Classification by voting feature intervals, in: *Proceedings of the 9th European Conference on Machine Learning*, 1997, pp. 85–92.

[9] T. Dietterich, Machine-learning research:four current directions, *AI Mag* **18**(4) (1997), 97–136.

[10] T. Dietterich, Ensemble methods in machine learning, in: *Multiple Classifiers Systems: first international workshop; proceedings/MCS 2000*, J. Kittler and F. Roli, eds, volume 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, June 2000. Springer, pp. 1–15.

[11] T. Dietterich and G. Bakiri, Solving multiclass learning problems via error-correcting output codesk, *J Artif Intell Res* **2** (1995), 263–286.

[12] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*, Addison-Wesley, 1973.

[13] S. Dzeroski and B. Zenko, Stacking with multi-response model trees, in: *Proceedings of Multiple Classifier Systems, Third International Workshop, MCS 2002*, J.K. Fabio Roli, ed., Lecture Notes in Computer Science, Cagliari, Italy, 2002 Springer.

[14] S. Dzeroski and B. Zenko, Is combining classifiers better than selecting the best one? *Mach Learn* **54**(3) (2004), 255–273

[15] T. English, Stacked generalization and simulated evolution, *Biosystems* **39**(16) (1996), 3–18.

[16] M. Faupel, http://www.micropraxis.com/gajit/index html, 1998.

[17] E. Frank and I. Witten, Generating accurate rule sets without global optimization, in: *Proceedings of the Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, 1998, pp. 144–151.

[18] Y. Freund and R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, in: *Proceedings of the Second European Conference on Computational Learning Theory* Springer-Verlag, ed., 1995, pp. 23–37.

[19] D. Goldberg, *Genetic Algorithms in search, optimization, and machine learning*, Addison-Wesley, 1989.

[20] L. Hansen and P. Salamon, Neural network emsembles, *IEEE Trans Pattern Anal Mach Intell* **12**(10) (1990), 993–1001.

[21] J. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, 1975.

[22] J. Holland, *Adaptation in Natural and Artificia Systems*, MIT Press, 2^a edition, 1992.

[23] W. Iba and P. Langley, Induction of one-level decision trees, in: *Proceedings of the Ninth International Conference on Machine Learning*, Morgan Kaufmann, 1992, pp. 233–240.

[24] G. John and P. Langley, Estimating continuous distribution in bayesian classifiers in: *Proceedings of the Eleventh Conference on Uncertainty in Artificia Intelligence*, M. Kaufmann, ed., 1995, pp. 338–345.

[25] R. Kohavi, The power of decision tables, in: *Proceedings of the Eighth European Conference on Machine Learning*, 1995.

[26] J. Kolen and J. Pollack, Back propagation is sensitive to initial conditions, in: *Advances in Neural Information Processing Systems*, 1991, pp. 860–867.

[27] A. Ledezma, R. Aler and D. Borrajo, *Data Mining: a Heuristic Approach*, chapter Heuristic Search Based Stacking of Classifiers Idea Group Publishing, 2001.

[28] B. Martin, Instance-based learning: Nearest neighbor with generalization. Master's thesis, University of Waikato, 1995.

[29] C. Merz, Using correspondence analysis to combine classifiers *Mach Learn* **36**(1–2) (1999), 33–58.

[30] T. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[31] J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[32] A. Seewald, How to make stacking better and faster while also taking care of an unknown weakness, in: *Proceedings of the Nineteenth International Conference on Machine Learning* (*ICML 2002*), A.G.H. Claude Sammut, ed., Sidney, Australia, July 2002. Morgan Kaufmann.

[33] A. Seewald and J. Fürnkranz, An evaluation of grading classifiers in: *Advances in Intelligent Data Analysis, 4th International Conference, IDA 2001, Proceedings*, F. Hoffmann, D.J. Hand, N.M. Adams, D.H. Fisher and G. Guimarães, eds, Lecture Notes in Computer Science, 2001, pp. 115–124.

[34] K. Ting and I. Witten, Issues in stacked generalization, *J Artif Intell Res* **10** (1999), 271–289.

[35] K.M. Ting and I. Witten, Stacked generalization: when does it work? in: *Proceedings of the International Joint Conference on Artificia Intelligence*, 1997.

[36] L. Todorovski and S. Dzeroski, Combining multiple models with meta decision trees, in: *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, 2000, pp. 54–64.

[37] I. Witten and E. Frank, *Data mining: practical machine learning tools and techniques with Java implementations*, Morgan Kaufmann, 2000.

[38] D. Wolpert, A mathematical theory of generalization: Part ii, *Complex Syst* **4** (1990), 201–249.

[39] D. Wolpert, Stacked generalization, *Neural Netw* **5** (1990), 241–259.

[40] Z. Zhou, J. Wu and W. Tang, Ensembling neural networks: Many could be better than al, *Artif Intell* **137**(1–2), 2002.