Simple Mimetic Classifiers*

V. Estruch C. Ferri J. Hernández-Orallo M.J. Ramírez-Quintana

DSIC, Univ. Politècnica de València, Camí de Vera s/n, 46020 Valencia, Spain.
{vestruch,cferri,jorallo,mramirez}@dsic.upv.es

Abstract. The combination of classifiers is a powerful tool to improve the accuracy of classifiers, by using the prediction of multiple models and combining them. Many practical and useful combination techniques work by using the output of several classifiers as the input of a second layer classifier. The problem of this and other multi-classifier approaches is that huge amounts of memory are required to store a set of multiple classifiers and, more importantly, the comprehensibility of a single classifier is lost and no knowledge or insight can be acquired from the model. In order to overcome these limitations, in this work we analyse the idea of "mimicking" the semantics of an ensemble of classifiers. More precisely, we use the combination of classifiers for labelling an invented random dataset, and then, we use this artificially labelled dataset to re-train one single model. This model has the following advantages: it is almost similar to the highly accurate combined model, as a single solution it requires much fewer memory resources, no additional validation test must be reserved to do this procedure and, more importantly, the resulting model is expressed as a single classifier in terms of the original attributes and, hence, it can be comprehensible. First, we illustrate this methodology using a popular data-mining package, showing that it can spread into common practice, and then we use our system SMILES, which automates the process and takes advantage of its ensemble method.

Keywords: Multi-classifier Systems, Stacking, Decision Trees, Comprehensibility in Machine Learning, Rule Extraction.

1 Introduction

Accuracy of classifiers can be improved by combining the predictions of a set (ensemble) of classifiers. These ensembles of classifiers are called multi-classifiers [9]. The effectiveness of combination is further increased the more diverse and numerous the set of hypotheses is [18] and also when several layers are arranged, known as "stacking" [29]. Many techniques for generating and combining classifier ensembles have been introduced: boosting [15, 25], bagging [5, 25], randomisation [10] and windowing [24], as well as several architectures, such as stacking [29] or cascading [16, 17].

* This work has been partially supported by CICYT under grant TIC2001-2705-C03-01 and Acción Integrada Hispano-Alemana HA2001-0059.

Although ensemble methods significantly increase accuracy, they have some drawbacks, mainly the loss of comprehensibility of the model and the large amount of memory required to store the hypotheses [20]. Recent proposals, such as miniboosting [26], have shown that memory requirements can be considerably reduced. Nonetheless, the comprehensibility of the resulting combined hypothesis is not improved since it is still a combination of three hypotheses and not a single one. A combined hypothesis is usually a voting of many hypotheses and it is treated as a black box, giving no insight at all. In the special case of stacking, the new classifier is defined in terms of the outputs of the first-layer classifiers, and hence, it is not a model defined in terms of the original problem attributes.

Instead of this "meta-model", it would be interesting to obtain a single model, with the high accuracy the multi-classifier has, but simple and defined in terms of the original problem attributes. To do this, the main idea is to consider the combination as an "oracle" from which we label a random invented dataset, which is then used to "re-train" a decision tree and, hence, to obtain a single model that is similar to the combination. Although this idea is relatively simple, it is scarce in the literature, and usually related to the extraction of rules from neural networks (see e.g. [4][7][8]). The reason why this idea has not been spread into common practice may be that usually some restrictions are imposed on the oracle, or the decision tree learner employed to capture the semantics of the oracle is very specific, or simply because the random invented dataset has not been constructed properly. As we will see, the key point is precisely the generation of a sufficiently large invented dataset by using a proper distribution but also the reuse of the training dataset, which would permit not only a good approximation/fidelity to the oracle but a good performance in terms of accuracy, what is really aimed to. The decision tree learner need not be specific: any state-of-the-art decision tree learner, such as C5.0, can be used. Hence, the method can be easily used by any data-mining practitioner as we show with an example.

The paper is organised as follows. First, in section 2, we discuss the use of ensemble methods, and how these all end up in accurate but complex, resource-inefficient and incomprehensible classifiers. We describe some previous methods in the literature for reducing the size of multi-classifiers, such as "ensemble pruning" or "mini-boosting", as well as other "black-box" approaches. Section 3 explains how multi-classifiers can be used to label a random dataset and use it for learning a new single classifier, in a way that resembles mimicking or imitating the behaviour of the combined classifier. We illustrate the process with an example using the Clementine data-mining package. In section 4, we address the question of how to generate the invented dataset: uniform distribution or training distribution, appending the training dataset or not. Section 5 presents our system SMILES where the previous process is automated. A thorough experimental evaluation is illustrated in Section 6, which includes the analysis of the random dataset distribution, the relevance of the size of the dataset, the size of the models obtained, and the comparison with direct single classifiers (C4.5/J4.8) and other ensemble methods (bagging/boosting). Finally, the last section presents the conclusions and proposes some future work.

2 Ensemble Methods and their Comprehensibility

Different techniques have been developed for combining the predictions obtained from multiple classifiers. According to the number of layers of classifiers, we can distinguish two different approaches:

- methods that generate a single layer of classifiers and then combine their predictions. This can be done by applying different learning algorithms to a single data set (e.g. [21] or *randomisation* [10]), or a single learning algorithm to different versions of the dataset (*bagging* [5] and *boosting* [15]).
- methods that generate multiple layer classifiers. In this case, the predictions made by the classifiers of one layer are used as input for the generation of the classifiers of the next layer (*stacking* [29] and *cascading* [16, 17]).

It has been shown that accuracy is significantly improved with ensemble methods; however, the large amount of hypotheses that are generated makes the use of ensemble methods difficult due to the high resource consumption, in particular the memory required to store the set of models. A few attempts have been made in order to reduce the number of hypotheses. In [26], e.g., a new method, called *miniboosting*, has been proposed. It consists in the reduction of the ensemble to only three decision trees. Although memory requirements are considerably reduced, it obtains 40% less of the improvement that would be obtained by a 10-trial AdaBoost, and the result is still not comprehensible.

With respect to the methods based on several layers, it may seem that the last layer could be in some way comprehensible. However, let us explain in more detail how stacking and cascading work and why this is not the case.

The basic idea in stacking is to create a partition of the learning set L at layer 1, training the first-layer classifiers with one part of the partition and then using the rest for training the second-layer classifier using as attributes the outputs of each first-layer classifier and as class the original class of each example. Originally [29], the partition of L was made in a similar way as cross-validation. However, nowadays, any process of this form (with different partitions, without partition or with probability estimates) is known as *stacked generalisation*. The process as a whole can be iterated, so making up several layers (*multiple stackings*).

Cascade generalisation is a special kind of stacking algorithm which uses sequentially a set of classifiers. At each step, the original data is extended by adding new attributes which represent the probability that an example belongs to a class given by a base classifier. As in the stacked generalisation method, different learning algorithms can also be used to obtain the classifiers.

One of the main drawbacks of the above methods is that comprehensibility is lost. This is due to the use of the outputs of classifiers as input attributes for the next layer. The final layer classifier is thus defined in terms of *artificial* attributes (the outputs of the previous layer classifiers) and not exclusively in terms of the original attributes. Moreover, these "meta-classifiers" require the first layers to be preserved in order to make all the ensemble work.

Recently, a variant of stacking/cascading, known as *ensemble pruning* [22], was introduced in order to discard a subset of the available base classifiers and preserve the most relevant ones, so reducing space (this relevance is determined by using a classical "pruning" method; hence the name). A decision tree is used to "mimic" the semantics of the meta-classifier, but, again, it uses the outputs of the first layer as inputs for the second layer, and not the original attributes.

Nonetheless, it is from the area of rule extraction from neural networks where some ideas can be reused, in particular the use of the ensemble as an oracle in the same way neural networks are the "oracle" in these works, called "non-decompositional" or "black-box" rule extraction methods. For instance, the system TREPAN ([7][8]) constructs a very particular decision tree (with m-of-n expressions) from a network, using a mixture of specific splitting criteria based on fidelity to the neural network, new stopping criteria and partial queries to the network.

A more recent work [4] can be seen as a refinement of the previous work, using more traditional decision trees with a specific pruning and where the random examples are created during the construction of the decision tree. However, there is no justification why a general decision tree learner cannot be used instead, how other alternatives to the generation of random examples could work, and why the original training dataset (the training set which trained the neural network) is not reused. Moreover, the accuracy of the resulting tree is not very close to the accuracy of the neural network, maybe because they try to have high fidelity instead of high accuracy: "we concentrate on increasing fidelity more than increasing accuracy of the extracted decision tree" [4]. Finally, the empirical results are only based on four datasets, so it is very difficult to precisely evaluate the goodness of both methods.

3 Arranging Mimetic Classifiers

In the previous section we have reviewed different ensemble methods. The final model is a combined model, not a single one exclusively described in terms of the original attributes. The motivation of this work comes when we look for a new classifier that could be "semantically" similar to the accurate (and complex) combined classifier but "syntactically" or "structurally" simpler and ultimately comprehensible. In other words, we look for a new classifier that "mimics" or imitates the behaviour of the combined classifier. But how can we do this in a simple and effective manner? The method we present here is based on an additional "random dataset". This dataset can be artificially generated as large as we want, and this is possible just because we want it *unlabelled*, i.e., without class values.

The point is illustrated in Figure 1. After a first complex multi-classifier stage, an unlabelled random dataset is "classified" or "labelled" by using the complex combined classifier (represented by a dotted circle in the figure). What we obtain now is a labelled random dataset that captures or distils (partially) the semantics of the combined classifier. And once here, the final stage is easy: we just join this labelled random dataset with the original training dataset and we train a single comprehensible model, e.g. a decision tree.

Note that the final model is exclusively defined in terms of the original attributes. Hence, the rest of the structure (the multiclassifier and the random data) is an auxiliar step that can be removed from memory. The outcome of the overall process is just a single model (as could be obtained by a simple decision tree learner). However, as we will justify experimentally in section 6, this final model is much more accurate because it is semantically similar to the multi-classifier.

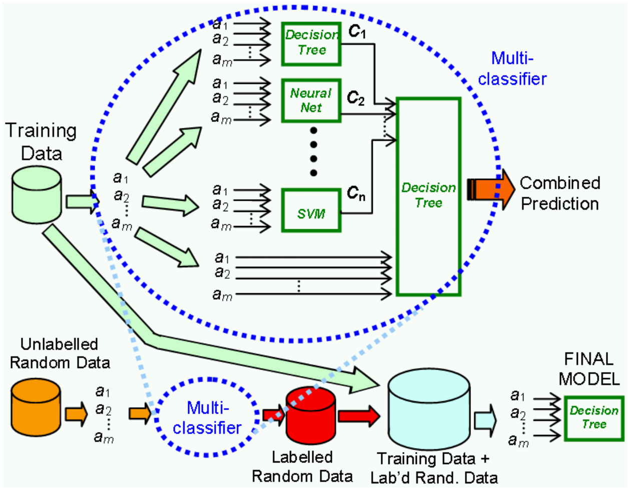

Fig. 1. Arrangement for Mimicking Classifiers

An additional feature comes when we consider that the classifier used to label the random dataset can be any kind of classifier, e.g. a simple neural network. In this case, we have a methodology for giving a comprehensible representation to any other non-comprehensible (black-box) classifier.

3.1 Example

Let us illustrate the previous process on a well-known (commercial) data-mining package, SPSS Clementine 6.0.2 and a single learning problem. We selected the "balance-scale" problem from the UCI repository [2] because it is quite simple to generate random datasets for it in a handicraft way (it has 4 numeric attributes in the range 1-5 and one attribute for the class with 3 possible values). We used a partition of the original 625 examples into two datasets: training set of 325 examples and test set of 300 examples. Using the training set, we learned three different models included in Clementine: a C&R Tree, a C5.0 Tree and a Neural Network. Then, we analysed their quality by using the test set, as it is illustrated in the two topmost streams of Figure 2.

The accuracies of each model with respect to the test set are:

```
C&R Tree: 78%        C5.0 Tree: 79.33%        NeuralNet: 88.33%
```

The neural network seems to be much better than the other two approaches for this specific problem. However, a neural network is not comprehensible.

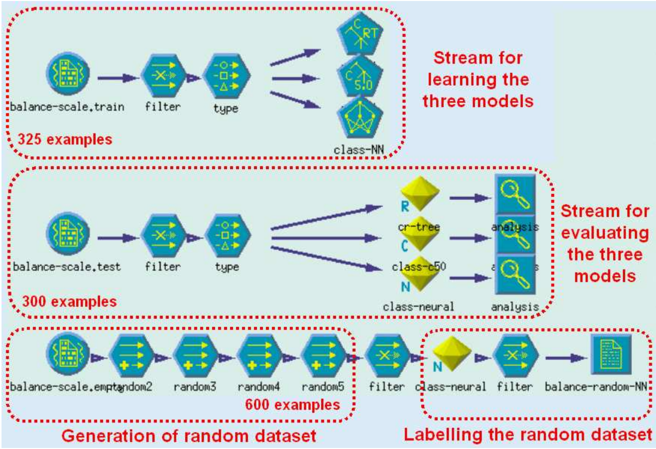

Fig. 2. First Stage of Mimicking using Clementine

Let us use our "mimicking" method to try to obtain a decision tree with similar accuracy to the neural network. For this, an unlabelled random dataset of size 625 examples was generated by using the uniform distribution. Since the neural network gives the best results[1], we label this dataset with it, as can be seen in the bottom part of Figure 2. All this process (generation of random dataset and its labelling) is illustrated at the bottom of the previous Figure 2.

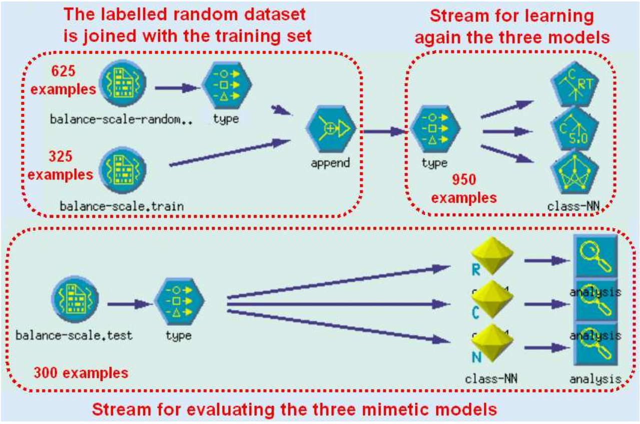

Fig. 3. Second Stage of Mimicking using Clementine

Finally, the second stage boils down to using this new labelled dataset (jointly with the original training dataset) to retrain new models, as it is illustrated in Figure 3. With these new models we have the following accuracies with the same test set[2]:

`C&R Tree: 81.7%` `C5.0 Tree: 86.0%` `NeuralNet: 89.0%`

The decision tree solutions are now closer to the neural network. Thereby, we can choose the C5.0 tree as a better comprehensible solution than that we obtained in the first stage, quite close in accuracy to the neural network.

---

[1] We could have made the labelling with a combination or stacking of classifiers, but we have not done it in this example for the sake of simplicity.

[2] Note that the test set has not been used for learning in any moment.

When we compared the structures of the first C5.0 decision tree obtained in figure 2 with the second one obtained in figure 3, we saw that the second one had almost double number of rules. However, it was overfitted to the neural net and not overfitted to the original training set. More importantly, the structure of both trees was different (the second one was not a simple specialisation of the first one), as we can see if we compare the topmost three levels of the first decision tree with the second one:

```
FIRST DECISION TREE
original training set

field5 =< 1 [Mode: L]
  field2 =< 2 [Mode: L]
    field3 =< 3 [Mode: L]
    field3  > 3 [Mode: L]   -> L
  field2  > 2 [Mode: L]   -> L
field5  > 1 [Mode: R]
  field2 =< 3 [Mode: R]
    field4 =< 2 [Mode: R]
    field4  > 2 [Mode: R]
  field2  > 3 [Mode: L]
    field3 =< 2 [Mode: R]
    field3  > 2 [Mode: L]
```

```
SECOND DECISION TREE
labelled random set + training set

field3 =< 2 [Mode: R]
  field5 =< 1 [Mode: L]
    field2 =< 3 [Mode: R]
    field2  > 3 [Mode: L] -> L
  field5  > 1 [Mode: R]
    field4 =< 1 [Mode: R]
    field4  > 1 [Mode: R]
field3  > 2 [Mode: L]
  field4 =< 2 [Mode: L]
    field2 =< 3 [Mode: L]
    field2  > 3 [Mode: L] -> L
  field4  > 2 [Mode: R]
    field5 =< 2 [Mode: L]
    field5  > 2 [Mode: R]
```

With this example, we have shown that the “mimicking” methodology can obtain different and more accurate comprehensible models than a direct single approach. But, more importantly, it can be seen as a way to capture or discover a symbolic representation to any black-box learning algorithm.

Obviously, although the previous process can be done manually in a few minutes using a data-mining package and can become mainstream for data-mining practitioners, it is tedious when repeated several times. Moreover, it would be impossible to scientifically evaluate this method if we do not automate the process and apply it for several datasets by using good evaluation techniques such as cross-validation. This is presented in the next three sections.

4 Random Datasets

A very important issue in the mimicking process is the generation of the random dataset in order to capture “extensionally” the semantics of the oracle. But, how should it be constructed?

Let us consider that the examples are equations of the form $f(\cdots) = c$, where f is a function symbol and c is the class of the term $f(\cdots)$. Given a function f with a arguments/attributes, an unlabelled random example is any instance of the term $f(X_1, X_2, \cdots, X_a)$, i.e., any term of the form $f(v_1, v_2, \cdots, v_a)$ obtained by replacing every attribute X_i by values v_i from the attribute domain (attribute type). Note that an unlabelled random example is not an equation (a full example) because we include no information about the correct class.

Apparently, there are many ways to generate a random dataset. One possibility is to ignore any previous training set and just consider the types of each attribute (if known). Then we can use a uniform distribution for generating each of the attributes. One problem here arises with continuous attributes, because the *min* and *max* limits must be known. Even though it is usual that they are not known in general, the factual *min* and *max* limits can be still obtained from the training set. More precisely:

– **Uniform Distribution**: for nominal attributes, one of the seen values is randomly chosen according to a uniform distribution. For numeric attributes, one random value is generated by using a uniform distribution on the interval $\{min, max\}$, where min is the lowest value and max is the highest value observed for that attribute in the training dataset.

A second possibility is to use the training set as a reference and use the values of the attributes that appear in the training set.

– **Prior distribution**: each attribute X_i of a new example is obtained as the value v_i in a different example $f(v_1, \ldots, v_i, \ldots, v_a)$ selected from the training set by using a uniform distribution. This procedure of generating instances assumes that all the attributes are independent. Consequently, the method just maintains the probabilities of appearance of the different values observed in each attribute of the training dataset (prior).

Both ways of obtaining the dataset are easy, although the uniform distribution can be directly implemented in any data-mining package (as shown in the previous section) or even using any spreadsheat application. Although there are more sophisticated methods (e.g. kernel density estimation methods [27]), we will just use and compare these two simple approaches.

5 Automatisation within the SMILES System

In order to evaluate the mimicking technique in general and, in particular, the best random data generation method and the best size of the random dataset, we have implemented the whole process in our SMILES system. SMILES is a multi-purpose machine learning system which includes (among many other features) the implementation of a multi-tree learner [11]. The main algorithm of SMILES is based on the usual construction of decision trees, although the rejected splits are not removed, but stored as *suspended* nodes. The further exploration of these nodes after the first solution has been built allows new models to be extracted from this structure. Since each new model is obtained by continuing the construction of the multi-tree, these models share their common parts. For this reason, a decision multi-tree can also be seen as an AND/OR tree or an option tree [6, 19], if one consider the alternative nodes as OR-nodes, and the nodes generated by an exploited OR-node as AND-nodes. The result is a multi-tree rather than a forest, with the advantage that a multi-tree shares the common parts and the forest does not. We perform a greedy search for each solution, but once the first solution is found, further trees can be obtained. The number of trees (OR-nodes) to be explored and how to select them determines the resulting multi-tree. With all these opened branches we can do two things: select one solution or combine a set of solutions. SMILES implements several criteria to combine a set of solutions given in the multi-tree. For more details on the structure and the system, we refer to [11] or the web page, `http://www.dsic.upv.es/~flip/smiles/` where its user manual and several examples are freely available.

SMILES has been modified in order to implement mimicking. In a first stage, the whole training set is used for training the multi-tree, obtaining a combined solution, i.e., a multi-classifier of, e.g., 100 trees. This combination, which will be used as "oracle", is usually significantly more accurate than any single classifier[11]. The next step is the generation of an unlabelled random dataset of length, e.g., 10,000 examples. This can be done automatically by SMILES in any of the two ways described in the previous section. Next, this dataset is labelled using the "oracle" (the combination). The labelled random dataset is preserved in memory and all the rest (the multi-tree structure) is freed from memory. In a second stage, we join the original training set with the labelled random dataset, so making up an even greater dataset. Finally, the last step is quite easy, we just train a *single* tree (not a multi-tree) using this final dataset. The result is, as we will see, a single tree which is approximately as accurate as the "oracle" and much more accurate than any single tree obtained by traditional means. This single tree is exclusively defined in terms of the original attributes.

In this work we have considered unpruned models for the first stage. This is because the use of models that are not 100% accurate *for the training set* could yield inconsistencies when joining the training set and the labelled random dataset. This is a limitation of the current implementation of SMILES (which does not allow inconsistencies in the class of two identical examples), but it is not a limitation of the approach (these inconsistencies could be purged or a robust decision-tree learner could be used).

6 Experiments

In this section we present an experimental evaluation of our approach by using the implementation in SMILES. For the experiments, we used GainRatio [24] as splitting criterion (for both the first stage and second stage). We chose a random method [11][12] for populating the multi-tree (after a solution is found, a suspended OR-node is woken at random) and we used the *maximum* strategy for combination [11]. Pruning is not enabled unless stated. The number of suspended OR-nodes explored in the first stage (for constructing the multi-classifier) is 100.

We used several datasets from the UCI dataset repository [2]. Table 1 shows the dataset name, the size in number of examples, the number of classes, and the number of nominal and numerical attributes.

Instead of fixed train-test partitions we have performed the experiments with 10-fold cross-validation. This means that the whole dataset is partitioned into 10 sub-datasets, nine are used as training set and the remainder one is reserved initially as test set, and is then used to evaluate the results in the second stage. This is done for the ten possible subdatasets. Since there are many sources of randomness, we have repeated the experiments 10 times. This makes a total of 100 runs (each one with a different first-stage multi-tree construction, random dataset construction and labelling, and second stage process) for each dataset. We show the average (in %) of these 100 runs. The last row of each table will also show the geometric mean of all the datasets.

#	Dataset	Size	Classes	Nom.Attr.	Num.Attr.
1	monks1	566	2	6	0
2	monks2	601	2	6	0
3	monks3	554	2	6	0
4	tic-tac	958	2	8	0
5	house-votes	435	2	16	0
6	breast-cancer-wisc	699	2	0	9
7	chess-kr-vs-kp	3196	2	36	0
8	hepatitis	155	2	14	5
9	balance-scale	625	3	0	4
10	new-thyroid	215	3	0	5
11	tae	151	3	2	3
12	iris	150	3	0	4
13	wine	178	3	0	13
14	hayes-roth	160	3	4	0
15	cmc	1473	3	7	2
16	horse-colic-surgical	366	2	14	8

Table 1. Information About Datasets Used in the Experiments.

The first thing studied is the method for generating the dataset. In Table 2 we show the accuracy of SMILES with different configurations. The first column ("1st") shows the results when learning a single tree (no ensemble, no mimicking). This column is similar to the results obtained by a single decision-tree learner, such as C4.5, and it is included just as a reference. The second column ("Comb") shows the accuracy of the combination of 100 trees and is, hence, the accuracy achieved by the "oracle". The next two columns ("2nd Prior" and "2nd Uniform") show the results of the mimetic classifier, i.e. the single tree obtained in the second stage, learned by using a random dataset (of size 10,000 examples) labelled by the "oracle", jointly with the training set. These two columns have used the "prior distribution" and the "uniform distribution" respectively.

#	1st	Comb	2nd Prior	2nd Uniform	2nd Prior (no Train)
1	95.2	100	99.9	100	100
2	71.0	76.5	76.1	75.8	75.8
3	97.5	97.9	97.8	97.9	97.9
4	77.1	82.1	82.5	81.7	82.2
5	94.7	95.7	95.5	95.0	95.3
6	93.8	94.9	94.5	93.4	94.4
7	99.6	99.4	99.4	99.5	98.3
8	75.8	81.7	79.5	76.5	79.4
9	77.9	82.6	82.9	82.8	82.9
10	92.0	92.9	92.5	92.2	92.0
11	60.6	63.1	63.3	62.7	61.1
12	94.1	95.5	94.7	94.8	94.7
13	93.0	93.0	92.4	90.5	92.0
14	74.1	76.8	76.8	76.8	76.1
15	48.3	49.7	49.1	47.9	49.4
16	78.5	83.2	81.8	77.6	82.3
gmeans	81.3	84.0	83.6	82.7	83.3

Table 2. Comparison of Methods for Generating the Invented Dataset.

The first observation from the previous table is that since we are using a good oracle (84.0 mean accuracy wrt. 81.3 original accuracy), the mimicking method can approach the better accuracy results of the oracle. However, it seems that the "prior distribution" is much more effective than the "uniform distribution". This was expected, because the former preserves the original distribution of the problem space. In fact, the use of 10,000 examples according to the prior distribution gets extremely close to the combination accuracy (83.6 vs. 84.0). According to these results, we will use the "prior distribution" in all the following experiments. The last column shows the effect if we do not use the training set in the second stage, i.e. we only use the labelled random dataset. Performance is slightly reduced. This may explain why related approaches that do not use the training set [4][7][8] have less improvement than that shown here (we use both datasets).

The next thing that needs to be examined is the relevance of the size of the invented dataset. The last columns in Table 3 show the results of the whole process with different random dataset sizes, from 100 to 100,000.

#	1st	Comb	2nd 100	2nd 1000	2nd 10000	2nd 100000
1	95.2	100	97.3	99.5	99.9	99.9
2	71.0	76.5	72.3	75.4	76.1	76.0
3	97.5	97.9	97.5	97.8	97.8	97.9
4	77.1	82.1	77.8	79.1	82.5	82.1
5	94.7	95.7	94.7	95.3	95.5	95.7
6	93.8	94.9	93.7	93.8	94.5	94.9
7	99.6	99.4	99.6	99.5	99.4	99.4
8	75.8	81.7	77.6	80.4	79.5	81.4
9	77.9	82.6	78.6	81.3	82.9	82.4
10	92.0	92.9	92.6	92.3	92.5	92.7
11	60.6	63.1	62.5	62.7	63.3	62.7
12	94.1	95.5	93.9	94.5	94.7	95.3
13	93.0	93.0	90.4	91.5	92.4	92.7
14	74.1	76.8	74.8	76.8	76.8	76.8
15	48.3	49.7	48.2	48.9	49.1	49.4
16	78.5	83.2	78.3	80.2	81.8	82.1
gmeans	81.3	84.0	81.7	82.9	83.6	83.8

Table 3. Comparison of the Size of the Invented Dataset for Mimicking.

As expected, that the greater the random dataset, the closer that the second-stage tree will be with respect to the oracle. Obviously, this has a price; the larger the random dataset, the slower the process. In practice, the optimal size of the random dataset depends on the problem; large training sets (in both number of examples and number of attributes) will require larger random datasets for good approximations. The generation and labelling of random datasets is usually a more efficient process than learning, and can be tuned to the system's resources quite comfortably. In what follows, we will use 10,000 random examples.

The previous results are remarkably positive, but a natural question arises. Are the single trees obtained in the second stage simple? Although a large decision tree can always be examined partially (top-down) and we can still extract knowledge from it, it is quite clear that the simpler the tree the easier to be understood. In order to study this issue, the first four columns of Table 4 show the accuracy and size (number of rules) of a single tree ("1st"), as could be obtained without combination in a first stage and the accuracy and size of the second-stage tree ("2nd"). All these results are without pruning. According to these first columns, it is clear that the approximation to the oracle (the increase in accuracy) is obtained by increasing the size of the tree (from 70.9 mean number of rules to 252.9 number of rules). This is quite a pity, because although this tree is much better than the original tree, it is less comprehensible.

Nonetheless, a different portrait can be seen if we enable pruning (we have used Pessimistic Error Pruning, [23]). The next column ("1st-Pruning") shows the

#	1st		2nd		1st-Pruning		2nd-Pruning					
	Acc	Rules	Acc	Rules	Acc	Rules	Acc (0.7)	Rules (0.7)	Acc (0.8)	Rules (0.8)	Acc (0.9)	Rules (0.9)
1	95.2	87.0	99.9	55.0	95.2	86.6	100.0	60.9	100.0	60.9	100.0	60.8
2	71.0	278.5	76.1	284.1	68.2	242.6	75.8	285.6	75.8	285.6	74.8	271.8
3	97.5	37.1	97.8	38	99.1	14	97.9	20.5	98.2	19.5	98.7	14.9
4	77.1	338.7	82.5	823.5	77.6	250.0	81.3	398.1	81.3	210.3	77.5	44.5
5	94.7	48.5	95.5	316.0	95.8	17.8	95.9	61.2	95.8	30.0	95.8	7.3
6	93.8	43.2	94.5	248.7	94.0	35.6	94.7	91.4	94.5	54.4	92.9	21.8
7	99.6	47.9	99.4	113.8	99.6	46.3	99.3	52.9	99.1	39.6	97.7	23.5
8	75.8	80.7	79.5	974.7	79.6	19.3	79.7	670.4	79.7	664.4	79.2	663.7
9	77.9	139.0	82.9	142.7	78.2	126.6	82.9	142.7	82.9	142.4	82.6	119.9
10	92.0	19.1	92.5	192.6	92.3	17.2	92.4	181.6	92.4	178.9	91.9	176.3
11	60.6	63.8	63.3	483.5	61.3	57.3	61.1	325.6	59.1	267.5	55.3	195.8
12	94.1	11.2	94.7	71.0	93.8	10.6	94.8	37.1	94.7	27.1	94.5	17.0
13	93.0	14.5	92.4	453.9	92.8	14.0	92.5	175.9	92.2	85.7	90.6	27.6
14	74.1	44.0	76.8	48.8	74.3	32.7	76.8	48.6	76.8	48.6	76.6	48.4
15	48.3	929.6	49.1	2399.3	49.0	627.8	51.4	565.4	52.7	204.6	43.5	7.8
16	78.5	146.1	81.8	1310.0	82.1	64.8	83.1	42.3	82.2	8.9	76.3	2.8
gmeans	81.3	70.9	83.6	252.9	81.9	48.8	83.7	119.8	83.6	82.1	81.2	40.7

Table 4. Comparison of the Size of the Models.

best[3] results when we enable pruning on the first-stage single tree. This is only shown for comparison, to realise that the difference with the mimetic classifier in size could even be larger (from 48.8 to 252.9) by using a traditional decision tree with pruning. The interesting thing comes, fortunately, when we use pruning on the mimetic classifier. The rightmost columns ("2nd-Pruning") show the results when pruning is enabled for the second-stage tree. In this case, and depending on the degree of pruning (0.7, 0.8 or 0.9), we see that we can obtain short trees with high accuracy, which was the motivation of this work (from 252.9 to 82.1 number of rules with similar accuracy).

Finally, in order to compare with the state of the art in ensemble methods, let us compare our results with the most successful ensemble methods: bagging and boosting. For that, we use the implementation of both included in the WEKA data mining package [28]. We have run bagging and boosting (ADA-Boost) with J4.8 (the Java version of C4.5) with their default parameters. Bagging was run without pruning and for boosting we enabled pruning. The results are shown in Table 5. The first two columns show the accuracy of J4.8 without pruning and with pruning. The next two columns show the results of 80 iterations of bagging[4] and 100 iterations of boosting. The final four columns show the results of SMILES with one tree (first-stage), with the combination of 100 trees (first-stage) and with the second-stage tree (mimicking with 10,000 random examples wrt. SMILES combination), without and with a slight pruning.

The results are quite encouraging. The technique presented in this paper is able to obtain single and short trees which are comparable or even excel the accuracy of the best current ensemble methods (83.7 vs. 82.4 and 83.8).

7 Conclusions

We have introduced a simple but effective method for obtaining highly accurate but still comprehensible models from evidence. The idea is based on two

[3] The best results for varying degrees of pruning.

[4] For more than 80 iterations WEKA ran out of memory.

	J4.8		Bagging	Boosting	SMILES			
#	no prune	pruning	80	100	1	100	100-2nd	100-2nd-0.7prune
1	95.1	98.4	100	99.5	95.2	100	99.9	100
2	62.7	64.1	67.0	82.2	71.0	76.5	76.1	75.8
3	98.7	98.9	98.9	97.9	97.5	97.9	97.8	97.9
4	79.1	80.3	83.8	82.6	77.1	82.1	82.5	81.3
5	95.5	96.5	96.6	95.1	94.7	95.7	95.5	95.9
6	94.1	94.5	96.3	96.7	93.8	94.9	94.5	94.7
7	99.4	99.4	99.4	99.6	99.6	99.4	99.4	99.3
8	79.0	79.2	82.6	84.7	75.8	81.7	79.5	79.7
9	79.4	77.7	82.9	76.0	77.9	82.6	82.9	82.9
10	93.0	93.1	94.9	95.3	92.0	92.9	92.5	92.4
11	56.5	55.7	60.7	64.8	60.6	63.1	63.3	61.1
12	94.6	94.7	94.3	94.5	94.1	95.5	94.7	94.8
13	93.5	93.4	95.9	96.9	93.0	93.0	92.4	92.5
14	72.5	74.2	56.1	65.9	74.1	76.8	76.8	76.8
15	50.0	52.0	52.7	50.4	48.3	49.7	49.1	51.4
16	83.2	81.5	83.1	81.5	78.5	83.2	81.8	83.1
gmeans	81.3	81.8	82.4	83.8	81.3	84.0	83.6	83.7

Table 5. Comparison with State-of-the-art Ensemble Methods.

stages. In the first stage, we use whatever highly accurate but incomprehensible method (e.g. stacking, cascading, boosting, bagging, neural networks, support-vector machines, Bayesian methods, etc.) to learn an "oracle" that is employed for labelling a randomly generated dataset. In the second stage, this dataset (jointly with the original training dataset) is used to train a single decision tree, which is, as we have shown, almost as accurate as the "oracle". Pruning can be enabled to simplify the tree. No extra validation set is used during the process.

Our approach is different from stacking and cascading because we can use whatever "oracle" at the first layer to label the random dataset, which is later used to learn a single model that does not use the outputs of the previous layer as inputs of the following one and hence, it just uses the original attributes. It is also different and simpler than other "black-box" approaches [4][7][8] because any kind of classifier can be used for the first stage but also for the second stage. For instance, a rule learner or an ILP system could be used in the second stage.

Since any classifier can be used in either stage, we have shown that this idea of mimicking can be put into practice quite easily by using any data-mining package. Additionally, the technique has been automated into our system SMILES, which uses an ensemble of shared decision trees in the first stage and a simple decision tree learner in the second stage. This implementation has been used to thoroughly evaluate the method. We have shown that using the prior distribution for generating the random dataset is preferable over the uniform distribution. It has been empirically demonstrated that the size of the dataset is extremely important and that the size (in number of rules) of the second-stage classifier depends on it, but that it can be significantly reduced by using classical pruning techniques, maintaining its accuracy. We have shown that this single solution is extremely close to that of the combination (also better than other approaches, such as archetyping [14]). In comparison with other methods, the technique presented in this paper obtains, to our knowledge, the highest accuracy of existing comprehensible classifiers (decision trees and rule learners).

In this paper, we have concentrated on presenting the method and an experimental evaluation of the method. We refer to [13] for some theoretical results concerning the mimetic method. Some of these results are: first, we show that when all the arguments of the function to be learned are nominal, a sufficiently large random dataset allows a loyal mimetic classifier to capture exactly the semantics of the oracle, as expected. Secondly, and more interestingly, we show that if the function to be learned is probabilistcally pure (i.e. not fractal) and the classifier is loyal and fence-and-fill then for a sufficiently large random dataset then the error made by the the mimetic classifier will approach zero. From here, we particularise the results to decision trees, to problems with whatever combination of nominal and numerical attributes and also whatever discrete oracle (such as combined classifiers and neural networks), as the setting shown here.

As future work we would like to compare this technique (in terms of accuracy, fidelity and comprehensibility) with existing decompositional rule extraction techniques used to convert neural networks (or other incomprehensible models) into comprehensible models. On the other hand, although we have concentrated on classification, it is clear that a similar technique could also be used for regression models. The iteration of "mimicking", by using more stages or different partitions on the training dataset can also be studied in the future. Another open question is the automatic adaptation of the size of the invented dataset to the size of the problem. The generation of more refined random datasets can also be improved, especially for numerical attributes where the range of some attribute is not known a priori. In [13] we have also studied some theoretical issues on the generation of the dataset. In particular, it would be interesting that the method for generating the invented dataset is exhaustive.

Other scenarios could also be considered, such the use of existing unlabelled data. This situation is obviously better, since the given unlabelled data complements the training set prior distribution. Techniques from the field of learning from unlabelled labelled data and, especially from co-training, could be used here [3]. In fact, mimicking can be seen as an asymmetrical (i.e. one way) co-training where one of the two classifiers is considered much better than the other and where unlabelled data is generated rather than obtained from a pool. Some ideas, e.g., the notion of ranking the examples such as that more confident predictions are chosen first could also be used for the oracle-based random dataset in our mimetic framework. This is being investigated in [13].

As a more ambitious future work, we consider the use of query learning [1] directly to the oracle (in a wider sense than [7][8]), instead of using the random dataset, as a better way of "mimicking".

References

1. D. Angluin. Queries and concept learning. *Machine Learning*, 2:319, 1987.
2. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
3. A. Blum and T. Mitchell. Combining Labeled and Unlabeled Data with Co-Training. In *Proc. of the 1998 Conf. on Computational Learning Theory*, 1998.
4. O. Boz. Extracting decision trees from trained neural networks. In *8th ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, 2002.
5. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

6. W. Buntine. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall,London, 1993.
7. M. W. Craven. *Extracting Comprehensible Models from Trained Neural Networks.* PhD thesis, Dep. of Computer Sciences, University of Wisconsin-Madison, 1996.
8. M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing*, 8, 1996.
9. T. G Dietterich. Ensemble methods in machine learning. In *First International Workshop on Multiple Classifier Systems*, pages 1–15, 2000.
10. T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157, 2000.
11. V. Estruch, C. Ferri, J. Hernández, and M.J. Ramírez. Shared Ensembles using Multi-trees. In *8th Iberoamerican Conf. on Artificial Intelligence, Iberamia'02*, volume 2527 of *Lecture Notes in Computer Science*, pages 204–213, 2002.
12. V. Estruch, C. Ferri, J. Hernández, and M.J. Ramírez. Beam search extraction and forgetting strategies on shared ensembles. In *Fourth Workshop on Multiple Classifier Systems (MCS2003)*, volume to appear of *Lecture Notes in Computer Science*, 2003.
13. V. Estruch and J. Hernández. Theoretical Issues of Mimetic Classifiers. Technical report, Dep. Information Systems and Computation, Tech. Univ. Valencia, "`http://www.dsic.upv.es/~flip/`", 2003.
14. C. Ferri, J. Hernández, and M.J. Ramírez. From Ensemble Methods to Comprehensible Models. In *The 5th Intl Conf on Discovery Science*, volume 2534 of *LNCS*, pages 164–177, 2002.
15. Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th Intl Conf Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.
16. J. Gama. Combining classifiers with constructive induction. In C. Nedellec and C. Rouveirol, editors, *Proc. of ECML-98*, volume 1398, pages 178–189, 1998.
17. J. Gama and P. Brazdil. Cascade Generalization. *Machine Learning*, 41(3):315–343, 2000.
18. T.K. Ho. C4.5 decision forests. In *Proc. of 14th Intl. Conf. on Pattern Recognition, Brisbane, Australia*, pages 545–549, 1998.
19. R. Kohavi and C. Kunz. Option decision trees with majority votes. In *Proc. 14th Intl. Conference on Machine Learning*, pages 161–169. Morgan Kaufmann, 1997.
20. D.D. Margineantu and T.G. Dietterich. Pruning adaptive boosting. In *14th Int. Conf. on Machine Learning*, pages 211–218. Morgan Kaufmann, 1997.
21. C.J. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36(1/2):33–58, 1999.
22. A.L. Prodromidis and S.J. Stolfo. Cost complexity-based pruning of ensemble classifiers. *Knowledge and Information Systems*, 3(4):449–469, 2001.
23. J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.
24. J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
25. J. R. Quinlan. Bagging, Boosting, and C4.5. In *Proc. 30th Natl. Conf. on AI and 8th Innovative Apps. of AI Conf.*, pages 725–730. AAAI Press / MIT Press, 1996.
26. J. R. Quinlan. Miniboosting decision trees. Submitted to JAIR, 1998.
27. B.W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, 1986.
28. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers, 1999.
29. D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.