Bagging Decision Multi-Trees*

V. Estruch C. Ferri J. Hernández-Orallo M.J. Ramírez-Quintana[1]

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

Abstract. Ensemble methods improve accuracy by combining the predictions of a set of different hypotheses. A well-known method for generating hypothesis ensembles is Bagging. One of the main drawbacks of ensemble methods in general, and Bagging in particular, is the huge amount of computational resources required to learn, store, and apply the set of models. Another problem is that even using the bootstrap technique, many simple models are similar, so limiting the ensemble diversity. In this work, we investigate an optimization technique based on sharing the common parts of the models from an ensemble formed by decision trees in order to minimize both problems. Concretely, we employ a structure called decision multi-tree which can contain simultaneously a set of decision trees and hence consider just once the "repeated" parts. A thorough experimental evaluation is included to show that the proposed optimisation technique pays off in practice.
Keywords: Ensemble Methods, Decision Trees, Bagging.

1 Introduction

With the goal of improving model accuracy, there has been an increasing interest in defining methods that combine hypotheses. These methods construct a set of hypotheses (ensemble), and then combine the components of the ensemble in some way (typically by a weighted or unweighted voting) in order to classify examples. The accuracy obtained will be often better than that of the individual components of the ensemble. This technique is known as *Ensemble Methods* [3].

This accuracy improvement of ensemble methods can be intuitively justified because the combined model represents an increase in expressiveness over the single components of the ensemble, and the fact that the combination of uncorrelated errors avoids over-fitting. The quality of the generated ensemble highly depends on the accuracy and diversity of its individual components [9].

Many methods have been proposed to construct a set of classifiers from a single evidence. These techniques have been applied in many different learning algorithms. Dietterich [3] distinguishes different kinds of ensemble construction methods, being probably the methods based on the manipulation of the training examples the most frequently used. The common idea of this kind of methods boils down to several times the same learning algorithm, each time with a different subset or weighting of the training examples, thus generating a different

* This work has been partially supported by CICYT under grant TIC2001-2705-C03-01 and MCyT Acción Integrada HU 2003-0003.

classifier for each set. The relevant issue in these methods is to define a good mechanism to generate subsets from the set of training examples. For instance, *Bagging* [2, 16], *Boosting* [8, 16] and *Cross-validated committees* [13] are ensemble methods of this family.

Bagging is derived from the technique known as *bootstrap aggregation*. This method constructs subsets by generating a sample of m training examples, selected randomly (and with replacement) from the original training set of m instances. The new subsets of training examples are called *bootstrap replicates*.

There have been some works that compare the performance of ensemble methods. Dietterich has compared *Bagging*, *Boosting* and *Randomisation* ensemble methods experimentally in [4]. The conclusions are that in problems without noise *Boosting* gets the best results, while the results of *Bagging* and *Randomisation* are quite similar. With respect to noisy datasets, *Bagging* is the best method, followed by *Randomisation*, and, finally, *Boosting*.

Ensemble methods have also important drawbacks. Probably, the most important problem is that they require the generation, storage, and application of a set of models in order to predict future cases. This represents an important consumption of resources, in both scenarios: learning process and predicting new cases. This important hindrances frequently limit the application of ensemble methods to real problems.

Let us consider the following scenario, given the classical *playtennis* [11] problem; we construct an ensemble of four decision trees by applying *Bagging*, over the C4.5 decision-tree learning algorithm, i.e., we learn four decision trees with C4.5 from four different bootstrap replicates. The four trees learned are shown in Figure 1.

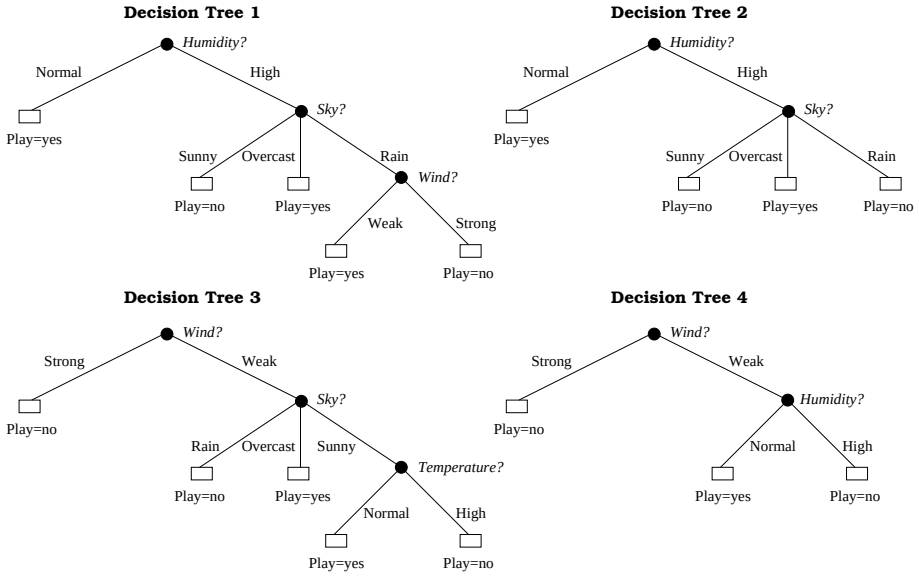

Fig. 1. Four different decision trees for the playtennis problem.

If we observe the four decision trees, we can appreciate that there are many similarities among them. For instance, Decision Tree 1 and Decision Tree 2, as well as Decision Tree 3 and Decision Tree 4, have the same condition at the root. More concretely, Decision Tree 1 and Decision Tree 2 are almost identical, the only difference between both trees is that Decision Tree 1 has an additional split in a node considered as a leaf in Decision Tree 2.

Furthermore, the first consequence of this phenomenon is that most of the solutions are similar, and hence, the errors can be correlated. All these patterns and regularities are not considered when learning ensembles of decision trees, and therefore, this process is also usually expensive in terms of computational cost. Motivated by these two problems, we present in this work an algorithm that exploits these regularities, and therefore, it allows a better dealing of resources when learning ensembles of decision trees. We employ a structure called decision multi-tree that can contain simultaneously a set of decision trees sharing their common parts. In previous works [6], we developed the idea of options trees [10] into the multi-structure, using a beam-search method to populate the multi-tree, mostly based on the randomisation technique, and several fusion techniques to merge the solutions in the multi-tree. One of the most delicate things of our previous approach was the choice of alternate splits. The use of *Bagging* in multi-trees solves this problem and, furthermore, it can allow a fairer comparison between the multi-tree and other ensemble methods. Although the presented algorithm is based on *Bagging*, the same idea could be easily applied to other ensembles methods such as *Boosting*.

The paper is organised as follows. First, in Section 2, we introduce the decision multi-tree structure. Section 3 introduces the algorithm that allows a decision multi-tree to be constructed by employing bootstrap replicates. A thorough experimental evaluation is included in Section 4. Finally, the last section presents the conclusions and proposes some future work.

2 Decision Multi-trees

In this section we present the decision multi-tree structure. This structure can be seen as a generalisation of a classical decision tree. Basically, a decision multi-tree is a tree in which the rejected splits are not removed, but stored as *suspended* nodes. The further exploration of these nodes after the first solution is built permits the generation of new models. For this reason, we call this structure decision multi-tree, rather than a decision tree. Since each new model is obtained by continuing the construction of the multi-tree, these models share their common parts.

Likewise, a decision multi-tree can also be seen as an AND/OR tree [12, 14], if one considers the split nodes as being OR-nodes and considers the nodes generated by an exploited OR-node as being AND-nodes.

Formally, a decision multi-tree is formed by an AND-node on the root, with a set of children which are OR-nodes (each one represents the split considered). Each OR-node can be *active* or *suspended*. The active OR-nodes have a set of children which are AND-nodes corresponding to a descendant of the split. This

schema is repeated for each new descendant node (AND-node), until an AND-node that is not further explored and it is assigned a class (a leaf).

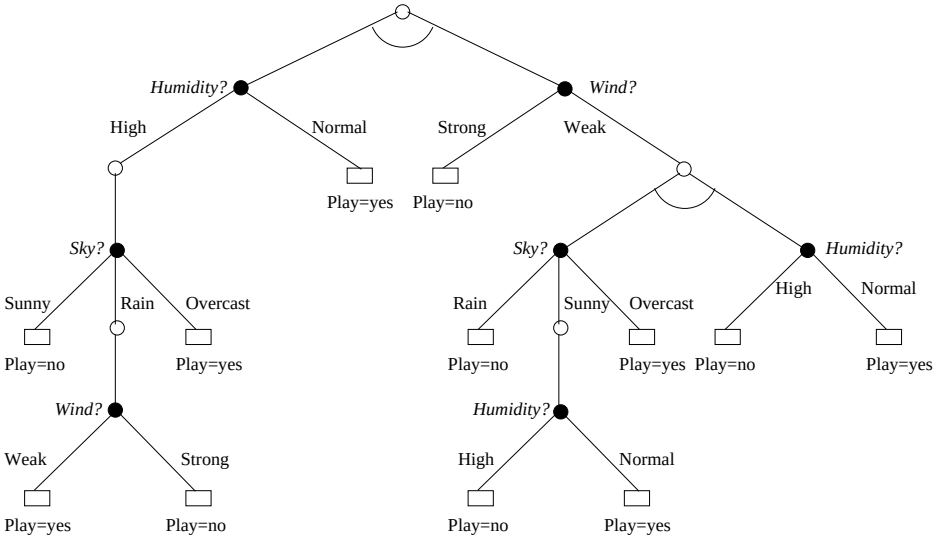

Fig. 2. The multi-tree structure.

Figure 2 shows a decision multi-tree that contains the four decision trees of Figure 1. AND-nodes are represented with no filled circles and they have an arc under the node. Leaves are represented by rectangles. OR-nodes are expressed by black-filled circles. In a decision multi-tree, as we can see, there are alternatively levels of AND-nodes and OR-nodes. Note that a classic decision tree can be seen as a decision multi-tree where only one OR-node is explored at each OR-node level.

In previous work [7], we have introduced an ensemble method that employs the decision multi-tree structure. In that work, the ensemble method performs a beam-search population of the decision multi-tree based on a random selection of the suspended nodes to be explored. However, one of the critical issues about this technique is the criterion for choosing the OR-node to "wake" from its suspended state.

3 Bagging Construction of a Decision Multitree

In this section we present our method to construct a decision multi-tree by using different training sets. As in *Bagging*, each training set is a bootstrap replicate of the original training set. However, as we have mentioned in the introduction, our approach differs from *Bagging* in that we construct a single structure (a multi-tree) that includes the ensemble of decision trees obtained by *Bagging* but without repeating their common parts.

In order to implement this, we use the first bootstrap replicate for filling the multi-tree with a single decision tree. Then, for each new bootstrap replicate

we continue the construction of the multi-tree, but only exploiting those nodes which have not been considered in the previous bootstrap replicates (i.e. they are suspended OR-nodes).

3.1 Algorithm

The following algorithm formalises this process:

Algorithm Bagging-Multi-tree (INPUT E**:dataset,** n**:integer; OUTPUT** M**:multi-tree)** {n is the number of iterations}

```
    M=Initialize_multi_tree(); {M only contains an empty AND-node}
    for i=1 to n do
        D=Bootstrap_replicate(E); {a bootstrap replicate is generated}
        if i=1 then LearnM(M.root, D)
        else LearnMBagg(M.root, D)
    end for
end
```

As we can see, the algorithm begins by generating an empty multi-tree, that is, a multi-tree that only contains one AND-node. Then, a bootstrap replicate is obtained and processed in each step of the main loop. At the first iteration, a multi-tree is constructed (procedure $LearnM$) and, for the following iterations, this structure is populated using a new bootstrap replicate for selecting the suspended OR-node that must be exploited (procedure $LearnMBagg$). In what follows we describe both processes.

Procedure $LearnM$ generates a multi-tree by selecting in each OR-level the best OR-node to be exploited (using, e.g. GainRatio or other splitting criterion). The process is repeated for the descendant of the active OR-node until a leaf is reached.

Procedure $LearnM$ **(INPUT** X**:AND-node,** D**:training dataset; OUTPUT** M**:multi-tree)**

```
    if X=leaf then exit;
    List_of_OR_nodes=Create_OR_nodes(X, D); {generate a list with one OR-
    node for each possible split and their descendants (AND-nodes)}
    B=Select_Best_OR_node(L, D); {the best node according to the split opti-
    mality criterion is selected}
    Activate(B); {the selected OR-node is activated}
    for Y ∈ children_of(B) do
        D'=filter(D, Y); {the examples of D that fall in node Y are selected}
        LearnM(Y, D')
    end for
end procedure
```

As in the above process, Procedure $LearnMBagg$ also selects in each OR-level the best OR-node to be exploited. But in this case the selected OR-node can be active or suspended. If it is an active node (which means that it has been exploited previously) then the procedure continues exploring their children. However, if it is an OR-node suspended, it is activated and their children are added

to the multi-tree.

Procedure *LearnMBagg* **(INPUT** X**:AND-node,** D**:training dataset; OUTPUT** M**:multi-tree)**

if X=leaf **then exit**;
$List\_of\_OR\_nodes$=$Update\_OR\_nodes(X, D)$;{update the split optimality of OR-nodes according to the training set D}
B=$Select\_Best\_OR\_node(L, D)$;
if B is *active* **then**
  for $Y \in children\_of(B)$ **do**
    D'=$filter(D, Y)$;
    $LearnMBagg(Y, D')$
  end for
else
  $Activate(B)$;
  for $Y \in children\_of(B)$ **do**
    D'=$filter(D, Y)$;
    $LearnM(Y, D')$ {the multi-tree is expanded from node Y}
  end for
end procedure

As regards the combination of predictions, there is an important difference with respect to classical ensemble methods: fusion points are distributed allover the multi-tree structure. Concretely, we combine the votes at the AND-nodes using the maximum fusion strategy. This strategy obtains the best results according to the experiments of [6].

3.2 Bagging Decision Trees versus Bagging Decision Multi-trees

Although the algorithm presented in this section is inspired by the *Bagging* method over decision trees, there are some differences between these two techniques and there can be differences in the errors performed by both methods. The most significant differences are how the ensemble is used for predicting new cases.

In *Bagging* decision trees, there is a significant probability (as we will see in the experiments) of learning similar trees. In the prediction phase, the repeated decision trees will be more determinant in the final decision. In the decision multi-tree, since we avoid duplicated trees, all the leaves have identical weight in the final decision. Note that this mechanism of ignoring repeated leaves for the prediction can be intuitively justified by the fact of having a set of models semantically different can help to improve the accuracy of the predictions.

Additionally, in a decision multi-tree the fusion of the predictions is performed internally at the OR-nodes, while in an ensemble of decision trees the voting is performed using each independent decision tree. Performing the fusion in the internal nodes of the multi-tree can alter the colour of the final decision. Furthermore, it represents an important improvement in the response time of the ensemble.

4 Experiments

In this section, we present an experimental evaluation of our approach, as is implemented in the SMILES system [5]. SMILES is a multi-purpose machine learning system which (among many other features) includes the implementation of a multiple decision tree learner.

Table 1. Datasets used in the experiments.

#	Datasets	Size	Classes	Nom. Attr.	Num. Attr.
1	Balance Scale	325	3	0	4
2	Breast Cancer	699	2	0	9
3	Breast Cancer Wisconsin	569	2	1	30
4	Chess	3196	2	36	0
5	Dermatology	366	6	33	1
6	Hayes-Roth	106	3	5	0
7	Heart Disease	920	5	8	5
8	Hepatitis	155	2	14	5
9	Horse-colic-outcome	366	3	14	8
10	Horse-colic-surgical	366	2	14	8
11	House Congressional Voting	435	2	16	0
12	Iris Plan	158	3	0	4
13	MONK's1	566	2	6	0
14	MONK's2	601	2	6	0
15	MONK's3	554	2	6	0
16	New Thyroid	215	3	0	5
17	Postoperative Patient	90	3	7	1
18	Segmentation Image Database	2310	7	0	14
19	Teaching Assistant Evaluation	151	3	2	3
20	Thyroid ANN	7200	3	15	0
21	Tic-Tac-Toe Endgame	958	2	8	0
22	Wine Recognition	178	3	0	13

For the experimental evaluation, we have employed 22 datasets from the UCI dataset repository [1]. Some details of the datasets are included in Table 1.

For the experiments, we used GainRatio [15] as a splitting criterion. Pruning is not enabled. The experiments were performed on a Pentium III-800 Mhz with 180MB of memory running Linux 2.4.2.

First, let us examine how the multi-tree grows with respect to the number of iterations of *Bagging*. Table 2 shows the mean of the number of nodes (AND nodes and OR nodes) in the multi-tree of all the datasets. The results are rather surprising since the number of nodes does not increase as one would expect. This reflects the facts that *Bagging* tends to repeat frequently the same decision trees. We also must remark that we do not remove the suspended OR nodes. This is the reason for the relatively high number of nodes of the multi-tree with just one iteration.

Table 2. Mean size of multi-trees in number of nodes (OR-nodes and And-nodes).

Number of iterations	1	20	40	60
Mean size	4021	9287	12927	15493

Table 3 shows the accuracy comparison (10 × 10-fold cross-validation) between classical *Bagging* as it is implemented in Weka (we call it *BagDT*), and the proposed algorithm as it is implemented in Smiles (we call it *BagMDT*) depending

Table 3. Mean accuracy.

	1		20		40		60	
#	BagMDT	BagDT	BagMDT	BagDT	BagMDT	BagDT	BagMDT	BagDT
1	77,94	79,40	79,26	81,92	80,13	82,85	80,63	82,92
2	93,81	94,12	94,28	96,07	94,64	96,11	94,81	96,28
3	92,43	94,22	93,48	95,61	93,13	95,91	93,29	95,91
4	99,61	99,40	99,37	99,43	99,42	99,40	99,28	99,40
5	91,58	93,80	94,00	96,43	94,31	96,89	94,75	97,05
6	72,13	72,50	73,44	47,31	73,44	54,25	73,19	52,56
7	51,64	47,26	54,58	46,21	54,72	46,45	55,98	46,52
8	76,20	78,98	77,27	83,23	77,47	82,38	77,80	82,76
9	62,72	65,82	64,14	65,66	64,81	65,77	65,33	65,79
10	78,53	83,15	81,92	82,86	82,19	83,12	82,50	83,06
11	94,74	95,47	94,95	96,45	94,77	96,55	94,81	96,58
12	94,13	94,60	95,13	94,57	94,13	94,33	94,47	94,20
13	96,73	95,11	95,93	99,84	96,42	100,00	95,73	100,00
14	70,97	62,66	67,62	66,38	67,45	66,91	66,17	67,19
15	97,47	98,67	98,00	98,74	97,69	98,90	97,93	98,88
16	92,81	92,95	92,76	94,24	93,29	94,56	93,10	94,76
17	66,00	59,89	66,25	63,23	65,88	64,44	65,50	65,00
18	95,91	96,90	95,29	97,67	95,23	97,69	95,23	97,62
19	61,93	56,46	56,93	59,35	57,33	61,50	55,60	60,51
20	99,23	99,72	99,15	99,65	99,06	99,65	98,98	99,65
21	77,16	79,16	79,54	83,17	80,32	83,82	80,34	83,80
22	93,00	93,49	93,12	95,26	93,06	95,79	92,94	95,90
Geomean	82.18	81.66	82.60	81.67	82.73	82.55	82.69	82.46
Average	83.49	83.35	83.93	83.79	84.04	84.42	84.02	84.38

on the number of iterations of the ensemble method. We have employed the version 3.2.3 of Weka [1]. We use $J48$ as base classifier (the Weka version of C4.5), with the default settings, except from pruning, which is not enabled.

The results of both algorithms are similar. Initially, there is a slight advantage for the *BagMDT* due to some small differences between the two systems in the implementation of the C4.5 algorithm. When the number of iterations is increased, both learning methods improve the accuracy, and the difference between them is partially reduced (even *BagDT* obtain better results if we consider the arithmetic average). Nonetheless, it seems that *BagMDT* reaches the saturation point earlier, probably because the repeated "good" branches are not weighted more for higher number of iterations.

Table 4 contains the average learning time for each classifier and dataset, and the geometric and arithmetic mean of all the datasets. From a practical point of view, it is resource consumption where we see the advantages of using decision multi-trees when learning ensembles of decision trees, since the training time is significantly reduced. To appreciate better this feature, Figure 3 shows the geometric average training time of *Bagging*, using Weka, and Smiles depending on the size of the ensemble. While *Bagging* decision trees shows a linear increase in time, *Bagging* decision multi-trees shows also a linear increase with a significant lower slope.

5 Conclusions

In this work, we present an algorithm that reduce the high computational cost characteristic of *Bagging* method. The technique is based on the use of the multi-

[1] `http://www.cs.waikato.ac.nz/~ml/weka/`

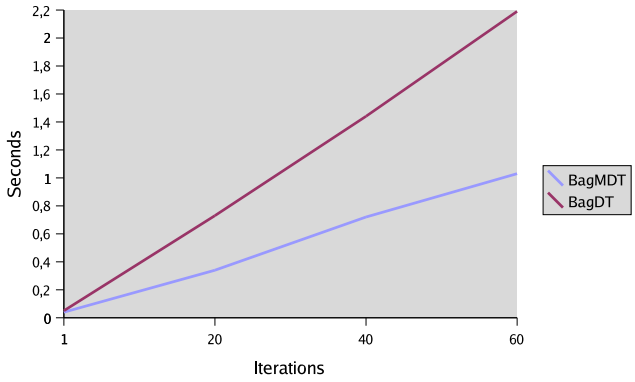

Fig. 3. Training time comparison.

Table 4. Mean training time.

	1		20		40		60	
#	BagMDT	BagDT	BagMDT	BagDT	BagMDT	BagDT	BagMDT	BagDT
1	0,04	0,08	0,31	1,07	0,59	2,29	0,85	3,24
2	0,08	0,06	0,67	1,07	1,29	2,32	1,87	3,19
3	0,32	0,29	2,50	4,24	4,81	8,73	6,94	13,02
4	0,26	0,53	2,28	6,39	4,43	12,44	6,51	18,72
5	0,08	0,04	0,60	0,65	1,11	1,28	1,60	1,93
6	0,01	0,01	0,01	0,18	0,03	0,35	0,04	0,57
7	0,27	0,18	2,53	5,41	5,96	10,43	21,95	15,88
8	0,04	0,03	0,26	0,36	0,48	0,75	0,72	1,13
90	0,15	0,03	1,01	0,51	2,30	1,10	2,82	1,36
10	0,12	0,04	0,87	1,16	1,61	2,06	2,36	3,13
11	0,02	0,03	0,15	0,39	0,39	0,78	0,43	1,18
12	0,01	0,01	0,04	0,09	0,08	0,18	0,12	0,27
13	0,01	0,02	0,06	0,25	0,16	0,47	0,17	0,74
14	0,01	0,04	0,13	0,50	0,33	0,91	0,35	1,50
15	0,01	0,01	0,04	0,15	0,12	0,29	0,13	0,44
16	0,02	0,02	0,15	0,24	0,28	0,49	0,40	0,75
17	0,01	0,01	0,03	0,10	0,07	0,19	0,08	0,31
18	0,85	1,17	8,93	16,90	19,41	33,56	40,00	51,31
19	0,02	0,02	0,15	0,28	0,28	0,55	0,40	0,85
20	1,13	0,69	9,84	9,53	20,59	18,95	31,99	28,54
21	0,03	0,06	0,28	0,74	0,72	1,43	0,78	2,35
22	0,04	0,03	0,27	0,44	0,52	0,88	0,74	1,33
Geomean	0.04	0.05	0.34	0.73	0.72	1.44	1.03	2.19
Average	0.16	0.15	1.41	2.30	2.98	4.57	5.51	6.90

tree structure. This structure allows trees to share the common parts. Therefore, the more structurally similar the trees are the better the improvement of the computational resources made by the multi-tree. Additionally, the multi-tree also makes it possible to enhance a key parameter in ensemble techniques: diversity. Note that by applying *Bagging* over c.4.5, the set of decision trees obtained presents many structural similarities (low diversity), so that misclassification errors can be easily correlated. But if these decision trees are organised into a multi-tree the redundancy is reduced and theoretically the accuracy should be better. In fact, *Bagging* multi-tree reaches the saturation point earlier than classical *Bagging*. However, a collateral effect arises when multi-tree is used, because this structure does not take into account how many times a branch tree appears. Therefore, the frequent branches and unusual ones have the same weight in the classification stage. The experimental evaluation makes clear that

it would be feasible to get a trade-off between the redundancy and the diversity, by using the multi-tree structure.

Summing up, the multi-tree structure can be viewed as a feasible and elegant way to overcome the main inherent drawbacks (huge amount of the computational resources and redundancy) of *Bagging*.

As future work, it would be interesting to investigate how we can improve accuracy by an adequate adjustment of the diversity and redundancy parameters in *Bagging* multi-trees. Additionally, we also plan to study whether the multi-tree is able to enhance other well-known ensemble methods, such as boosting.

References

1. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
2. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
3. T. G Dietterich. Ensemble methods in machine learning. In *First International Workshop on Multiple Classifier Systems*, pages 1–15, 2000.
4. 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.
5. V. Estruch, C. Ferri, J. Hernández, and M. J. Ramírez. SMILES: A multi-purpose learning system. In *Logics in Artificial Intelligence, European Conference, JELIA*, volume 2424 of *Lecture Notes in Computer Science*, pages 529–532, 2002.
6. V. Estruch, C. Ferri, J. Hernández, and M.J. Ramírez. Shared Ensembles using Multi-trees. In *the 8th Iberoamerican Conference on Artificial. Intelligence, Iberamia'02*, volume 2527 of *LNCS*, pages 204–213, 2002.
7. V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez. Beam Search Extraction and Forgetting Strategies on Shared Ensembles. In *4th Workshop on Multiple Classifier Systems, MCS2003*, LNCS, 2003.
8. Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.
9. Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(10):993–1001, October 1990.
10. Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In *Proc. 14th International Conference on Machine Learning*, pages 161–169. Morgan Kaufmann, 1997.
11. T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
12. N.J. Nilsson. *Artificial Intelligence: a new synthesis*. Morgan Kaufmann, 1998.
13. Bambang Parmanto, Paul W. Munro, and Howard R. Doyle. Improving committee diagnosis with resampling techniques. In *Advances in Neural Information Processing Systems*, volume 8, pages 882–888. The MIT Press, 1996.
14. J. Pearl. *Heuristics: Intelligence search strategies for computer problem solving*. Addison Wesley, 1985.
15. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
16. J. R. Quinlan. Bagging, Boosting, and C4.5. In *Proc. of the 30th Nat. Conf. on A.I. and the 8th Innovative Applications of A.I. Conf.*, pages 725–730. AAAI Press / MIT Press, 1996.