Beam Search Extraction and Forgetting Strategies on Shared Ensembles*

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

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. However, there is an important shortcoming associated with ensemble methods. Huge amounts of memory are required to store a set of multiple hypotheses. In this work, we have devised an ensemble method that partially solves this problem. The key point is that components share their common parts. We employ a multi-tree, which is a structure that can simultaneously contain an ensemble of decision trees but has the advantage that decision trees share some conditions. To construct this multi-tree, we define an algorithm based on a beam search with several extraction criteria and with several forgetting policies for the suspended nodes. Finally, we compare the behaviour of this ensemble method with some well-known methods for generating hypothesis ensembles.

Keywords: Ensemble Methods, Decision Trees, Randomisation, Search Space, Beam Search.

1 Introduction

Ensemble methods [7] are used to improve the accuracy of machine learning models. Basically, this technique combines a finite set of hypotheses into a new hypothesis, which is usually more accurate than any of the ensemble components. Examples of well-known ensemble methods are boosting [11], bagging [2], randomisation [8], etc. Although accuracy is significantly increased, a large amount of computational resources is necessary to generate, store, and employ the ensemble due to the large number of different hypotheses that make up the ensemble. For this reason, there are several contexts where these techniques are hard to apply.

Since ensemble methods construct a set of models, one way to overcome the above mentioned drawbacks could be to share the common parts of the models. In a previous work [10], we presented an algorithm which is able to obtain more than one tree. It is based on a structure, called multi-tree, which can contain a set of decision trees that have some of their conditions in common. A multi-tree is generated by applying a splitting criterion on a node and then storing the

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

splits which have not been selected into a list of suspended nodes. The selected split is pursued until a complete tree is constructed. Later, alternative models can be generated by exploring the nodes of the list. This way of exploring the space for generating an ensemble of hypotheses can be considered a beam search [15].

The use of an auxiliary data structure (list of suspended nodes) must lead to establishing a policy to manage it. This policy must cover several aspects such as the definition of a criterion to extract a node in order to further populate the multi-tree. In this paper, we focus on the study of some strategies for selecting the nodes to be explored from the list of suspended nodes. We introduce some methods, and we then study these strategies experimentally. We also compare the performance of this ensemble method with other well-known methods such as bagging [2] and boosting [11].

Despite the advantages of the multi-tree approach, there is a great number of suspended nodes that are never employed. Thus, they occupy memory needlessly and it would be interesting to forget these nodes. We study some techniques to keep only one subset of the nodes under consideration. This *forgetting* technique leads to a better use of resources as we experimentally demonstrate.

This work can be considered as a continuation of [10]. In that work, we presented the basis for and main features of a multi-tree as an ensemble method. An experimental study of some fusion strategies and vector modification techniques was made. In this work, we investigate some multi-tree search and exploration policies, and we present an important optimisation in the algorithm.

The paper is organised as follows. In section 2, we introduce the multi-tree structure. Section 3 includes some experimental evaluations on multi-tree construction. It also explains different forgetting strategies and shows the results of several experiments related to this technique. Finally, section 4 closes the paper with conclusions and future work.

2 Shared Ensembles

Ensemble methods require the generation, storage and application of a set of models in order to predict future cases. This represents an important use of resources in both scenarios: the learning process and the prediction of new cases.

In our previous work [10], we presented a method that allows a set of decision trees to be generated from a single evidence. A collection of trees is called a forest, but our approach is based on a shared ensemble, namely, a collection of trees that share their common parts (*decision multi-tree*). The idea is to generate an AND/OR tree structure, called multi-tree, from which it is possible to extract a set of hypotheses. We perform a greedy search for each solution, but once the first solution is found, the following ones can be obtained keeping in mind that computation time is limited. Therefore, in a certain way, our algorithm can be considered *anytime*. An anytime algorithm is one that can be interrupted at any point during computation to return a result whose quality increases as the computation time increases [6].

Since a multi-tree can contain a set of hypotheses, we employ an ensemble method that combines the individual components of the multi-tree. This ensemble method takes advantage of the internal structure of the multi-tree by sharing common parts of the components of the ensemble. In this way the amount of resources required for the learning and the application of the ensemble is reduced. We called this ensemble method: *shared ensemble*.

A decision multi-tree structure is similar to an option tree [3, 4, 12]. The main difference between these methods is the construction strategy. In the multi-tree method, we store all the rejected splits as suspended nodes, and later we continue the multi-tree construction selecting one of these suspended nodes according to a criterion. Option trees are constructed as usual. However, when the difference between the best test and the other alternatives is small at a level, these alternatives are also explored as optional tests (OR-nodes). The creation of optional nodes is limited at the upper levels to restrict the size of the option tree. Therefore, while option trees are constructed by a breadth search, a multi-tree is generated by a beam search, i.e. we store the discarded alternatives in a list of suspended nodes, and later, if we want to further populate the multi-tree, we select one node from the list according to one selection criterion, and at this point the search is continued.

As we have just seen in Section 1, the decision multi-tree can be populated by selecting one suspended OR-node from the list of suspended OR-nodes by means of a selection criterion (*suspended OR-node selection criterion*). The *optimality* computed by the splitting criterion can be used to choose the OR-node to be explored. We have introduced the following suspended OR-node selection criteria:

- **Rival Absolute:** The node with the highest optimality is selected from the list of suspended OR-nodes.
- **Rival Ratio:** For every suspended OR-node, a rival ratio value is computed as the relation between the *optimality* of the OR-node and the best *optimality* of its sibling OR-nodes (i.e. the children of the same AND-node). The suspended OR-node with the highest rival ratio value is chosen.
- **Random:** This criterion just selects nodes pseudo-randomly (uniform distribution) from the list.
- **TopMost:** Select the topmost node with the highest optimality first.
- **Bottom:** Selects the bottom node with the highest optimality first.

Once a multi-tree is constructed, it contains a set of hypotheses which depends on the number of suspended OR-nodes which have been explored. Note that the number of hypotheses increases exponentially w.r.t. the number of nodes explored. Since it could be prohibitive to combine all these models at the top, the combination is performed inside the multi-tree by a majority vote. A study of some fusion methods, most of them based on [13], was presented in [10].

2.1 Node forgetting

Despite the advantages of the multi-tree structure, many suspended nodes are never 'woken', occupying memory needlessly. An additional criterion can be specified to forget some of the suspended nodes and, hence, to use less memory.

We have studied several methods for restricting the nodes to be selected:

- **Constant:** We only store a constant minimum number of nodes.
- **Logarithmic:** A logarithmic number of nodes is selected (a constant minimum is also applied).
- **Logarithmic + depth:** A logarithmic number of nodes is selected but corrected by the depth of the nodes (a constant minimum is also applied). This correction tends to store more nodes at the top positions, where the exploration of suspended nodes permits models to become more diverse.

In the following section, we perform an experimental evaluation of forgetting strategies.

3 Experiments

In this section, we present an experimental evaluation of our approach, as is implemented in the SMILES system [9]. SMILES is a multi-purpose machine learning system which (among many other features) includes the implementation of a multiple decision tree learner.

#	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	Contraceptive Method Choice	1473	3	7	2
6	Dermatology	366	6	33	1
7	Hayes-Roth	106	3	5	0
8	Heart Disease	920	5	8	5
9	Hepatitis	155	2	14	5
10	Horse-colic-outcome	366	3	14	8
11	Horse-colic-surgical	366	2	14	8
12	House Congressional Voting	435	2	16	0
13	Iris Plan	158	3	0	4
14	MONK's1	566	2	6	0
15	MONK's2	601	2	6	0
16	MONK's3	554	2	6	0
17	New Thyroid	215	3	0	5
18	Postoperative Patient	90	3	7	1
19	Segmentation Image Database	2310	7	0	14
20	Teaching Assistant Evaluation	151	3	2	3
21	Thyroid ANN	7200	3	15	0
22	Tic-Tac-Toe Endgame	958	2	8	0
23	Wine Recognition	178	3	0	13

Table 1. Datasets used in the experiments.

For the experimental evaluation, we have employed 23 datasets from the UCI dataset repository [1]. Some details of the datasets are included in Table 1.

For the experiments, we used GainRatio [16] 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. The results present the mean accuracy of the 10 $\times$ 10-fold cross-validation for each dataset, and finally the geometric mean of all the datasets. First, we analyse the behaviour of the quality of the

#	Bottom	Optimal	Random	Rival Ratio	Top most
1	78.18	79.00	82.61	77.63	86.60
2	93.84	94.70	94.88	93.45	94.45
3	92.43	92.55	93.07	93.75	93.75
4	99.62	99.58	99.38	99.64	99.62
5	48.35	48.23	49.73	48.31	51.70
6	91.86	92.92	94.03	92.17	91.31
7	76.44	76.75	76.75	73.06	76.75
8	52.23	52.48	54.43	52.20	59.13
9	76.07	75.87	81.40	75.93	83.67
10	62.72	62.78	67.28	62.69	75.28
11	78.53	78.00	83.36	78.33	85.83
12	94.70	95.53	95.67	94.35	95.23
13	94.13	94.13	95.33	94.20	94.67
14	95.20	95.91	99.78	99.73	100.00
15	75.42	75.33	75.67	71.05	77.55
16	98.02	98.02	97.85	97.38	97.95
17	92.14	93.38	93.29	93.14	92.43
18	64.63	63.75	63.25	69.88	63.00
19	95.91	95.91	96.40	96.06	96.06
20	60.47	60.60	62.93	60.07	63.40
21	99.24	99.37	99.26	99.23	99.19
22	77.21	76.97	82.01	78.13	84.94
23	92.94	92.59	93.06	94.00	90.00
Geomean	80.56	80.70	82.45	80.69	83.61

Table 2. Accuracy of combination depending on the second tree criterion.

shared ensemble depending on some criteria to populate the multi-tree: *Bottom, Optimal, Random, Rival Ratio* and *Top Most*.

Table 2 shows the accuracy comparison between these methods for populating the multi-tree. According to the results, the best methods for building the shared ensemble seem to be *Random* and *Top Most*. The *Top Most* method has an important drawback: the decision trees do not share many components, so the construction and use of the shared structure is slower. This fact is perceived if one observes the consumption of time. Table 3 shows the average learning time for each dataset depending on the method employed to populate the multi-tree. The difference in time between *Random* and *Top Most* is important because the *Top Most* method selects the nodes to be explored at the top positions in the multi-tree. This leads to the generation of very different models because they do not share many conditions. However, this criterion also produces large multi-trees where the advantages in resource saving of the multi-tree structure are practically lost. Consequently, the *Random* criterion can be considered as an optimal trade-off between efficiency and accuracy.

3.1 Node forgetting

In this section, we study the effect of the forgetting optimisation in the performance of the ensemble method. Since we have seen that a good criterion is to select the nodes to be explored randomly, a random strategy of node forgetting will not alter the suspended node selection criterion, and therefore the accuracy of the multi-classifier will not be degraded.

A comparison of some of these criteria according to the accuracy of the classifiers obtained is presented in Table 4. These criteria are: no forgetting, limiting a constant number of nodes (5) in each OR-node, leaving a logarithmic number of nodes, and finally a correction of the last method depending on the

#	Bottom	Optimal	Random	Rival Ratio	Top most
1	0.05	0.06	1.10	0.08	31.41
2	0.81	0.34	2.47	0.29	21.92
3	0.09	0.56	11.86	8.77	136.39
4	0.39	0.63	5.78	0.91	0.30
5	0.56	0.46	5.64	40.11	939.99
6	0.30	0.33	3.51	1.09	16.03
7	0.04	0.02	0.07	0.03	0.03
8	0.40	0.36	5.00	5.64	255.20
9	0.06	0.07	1.52	0.13	7.92
10	0.21	0.18	3.59	0.58	83.75
11	0.17	0.14	3.19	0.26	41.93
12	0.02	0.08	0.54	0.07	2.62
13	0.03	0.12	0.35	0.05	1.59
14	0.16	0.02	0.16	0.04	0.26
15	0.02	0.03	0.29	0.07	1.05
16	0.03	0.02	0.15	0.02	0.18
17	0.01	0.16	0.91	0.96	6.93
18	0.01	0.02	0.21	0.03	0.49
19	1.32	2.71	15.92	28.78	15.82
20	0.03	0.04	0.85	11.40	118.58
21	1.24	2.17	11.31	7.56	14.62
22	0.05	0.05	0.46	0.06	9.02
23	0.06	0.57	1.99	0.68	880.00
Geomean	0.10	0.14	1.34	0.46	9.12

Table 3. Average learning time (in seconds) depending on the second tree criterion.

node depth. The results of Table 4 indicate that, as expected, the forgetting process does not produce significant modifications on the quality of the classifiers.

Table 5 contains the average learning time for each classifier and dataset, and the geometric mean of all the datasets. The experiments demonstrate the usefulness of node forgetting, because it reduces the learning time. The best forgetting method is the one based on the selection of a logarithmic number of all possible nodes. The corrected version (log + depth) is a bit slower because it selects nodes at the top of the multi-tree. The method based on a constant number also improves the learning time. However, the results in accuracy are slightly worse than the logarithmic method.

With regard to the improvement in memory, Table 6 shows the memory (in Kilobytes) required by the system for learning the multi-tree for some datasets. We also include the percentage of memory employed by the forgetting method with respect to the original method (no forgetting). For large and medium problems (4, 28) the use of forgetting drastically reduces the required memory. However, for small problems (15) the reduction is more limited. The forgetting method that requires less memory is the "logarithmic" method.

According to these experiments, the best suspended-node forgetting criterion seems to be to randomly leave a logarithmic number of nodes, without taking depth into account.

3.2 Comparison with other ensemble methods

Let us compare the behaviour of the shared ensemble technique (random extraction + logarithmic forgetting) with other popular ensemble methods: bagging and boosting (Adaboost). We have employed the Weka (version 3.2.3)[1] implementation of these two ensemble methods. The ensemble methods use $J48$ as

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

#	No forgetting	Const=5	Logarithmic	Log. + depth
1	82.61	81.40	82.24	85.11
2	94.88	95.16	94.93	94.81
3	93.07	93.36	93.41	93.80
4	99.38	99.39	99.37	99.34
5	49.73	49.30	49.51	52.65
6	94.03	94.53	94.42	94.39
7	76.75	76.75	76.44	75.94
8	54.43	54.09	54.26	55.58
9	81.40	81.53	80.67	81.67
10	67.28	66.00	67.42	67.06
11	83.36	83.36	83.19	83.67
12	95.67	95.86	95.91	95.79
13	95.33	95.07	95.00	95.13
14	99.78	99.78	99.89	97.56
15	75.67	75.67	76.38	71.50
16	97.85	97.85	97.75	97.67
17	93.29	93.29	93.57	92.29
18	63.25	62.00	61.88	62.13
19	96.40	96.31	96.61	96.43
20	62.93	62.40	62.87	63.40
21	99.26	99.28	99.33	99.24
22	82.01	82.22	81.78	83.96
23	93.06	92.88	93.76	92.76
GeoMean	82.45	82.22	82.38	82.55

Table 4. Accuracy of the combination in the multi-tree depending on the forgetting method.

base classifier (the Weka version of C4.5), and we have used the default settings. Pruning is only enabled for boosting since this method requires pruning to get good results.

Figure 1 shows the average accuracy (10 × 10-fold cross-validation) obtained by the three methods depending on the number of iterations. Initially, the best results are obtained by boosting, whereas bagging and *Multi-tree* results are slightly lower. This is probably due to the fact that they do not use pruning. As expected, when the number of iterations of the ensemble methods is increased, all the methods improve in accuracy. In this case, we can see that the *multi-tree* method is the one that most enhances the results. It even surpasses boosting at 80 iterations or more. Note that the accuracy of bagging with 100 classifiers is not shown in the figure. The reason for this is that the computer ran out of memory for such a configuration.

Finally, although the shared ensemble method has suitable properties in terms of accuracy of the combined classifier, the most attractive feature of this algorithm is the sharing of some parts of the ensemble. This fact allows for a good allocation of computational resources. Figure 2 shows the average training time of bagging, boosting, and Multi-tree (shared ensemble) depending on the size of the ensemble. While bagging and boosting[2] show a linear increase in time, the shared ensemble technique shows a sub-linear increase. Note that the implementation features of the methods clearly affect the training time (Weka is implemented in Java, while SMILES is implemented in C++). However, this does not produce significant changes in the asymptotic behaviour of the algorithms when varying the number of iterations.

[2] The observed non-linear behaviour of Boosting is due to a technique implemented in it that stops the algorithm when the accuracy does not improve further from iteration to iteration.

#	No forgetting	Const=5	Logarithmic	Log + depth
1	1.10	0.60	0.62	1.71
2	2.47	1.62	1.66	2.82
3	11.86	12.47	12.00	15.20
4	5.78	5.68	4.77	9.97
5	5.64	3.74	2.48	12.49
6	3.51	3.36	3.10	4.02
7	0.07	0.07	0.02	0.01
8	5.00	2.66	2.64	5.54
9	1.52	1.38	1.36	1.87
10	3.59	2.34	2.45	3.70
11	3.19	2.33	2.28	3.58
12	0.54	0.51	0.47	0.94
13	0.35	0.30	0.27	0.53
14	0.16	0.17	0.14	0.04
15	0.29	0.30	0.29	0.08
16	0.15	0.16	0.12	0.03
17	0.91	0.90	0.74	1.27
18	0.21	0.18	0.16	0.08
19	15.92	17.34	16.17	17.44
20	0.85	0.63	0.57	1.28
21	11.31	12.14	9.79	15.36
22	0.46	0.44	0.37	0.55
23	1.99	2.06	1.70	2.05
GeoMean	1.34	1.16	0.99	1.23

Table 5. Average learning time for the combined solution in the multi-tree depending on the forgetting method.

#	Original	Const=5	%	Logarithmic	%	Log + depth	%
4	49512.00	10892.00	22.0%0	6388.00	12.90%	8328.00	16.82%
15	4880.00	4820.00	98.77%	1484.00	30.41%	3472.00	71.15%
23	18800.00	3268.00	17.38%	2272.00	12.09%	2608.00	13.87%

Table 6. Average memory (in Kbytes) for the combined solution in the multi-tree depending on the forgetting method.

4 Conclusions

This work has presented some strategies for constructing an ensemble based on a beam search. This search considers the alternative options that are rejected by classical greedy decision-tree construction algorithms. The main feature of the technique is the use of a structure called multi-tree that permits sharing common parts of the single components of the ensemble. For this reason, we call it *shared ensemble.*

We have introduced some criteria to populate the multi-tree. These criteria are variations of a beam search over the multi-tree structure. We have implemented this algorithm and an experimental evaluation has been performed in order to analyse the performance of these criteria. We have also studied an optimisation that permits a better use of resources based on a filter of the nodes to be stored (forgetting).

The beam search employed for the generation of the multi-tree structure is related to the rule learning system CN2 [5]. This algorithm performs a search in which the algorithm maintains a list of k candidates at each selection, instead of considering just a single option. On each search step, the algorithm generates descendants from these k candidates, and the resulting set is again reduced to the k most promising members. This can be seen as a global forgetting technique that only stores a constant number k of options. Additionally, CN2 just explores one option and does not perform combination.

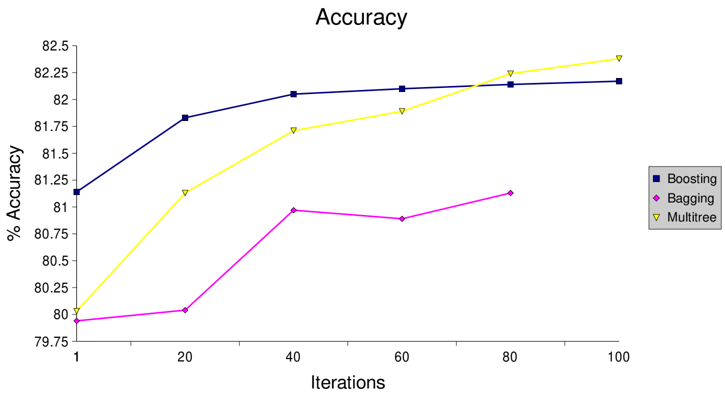

Fig. 1. Accuracy obtained by the ensemble methods depending on the number of iterations.

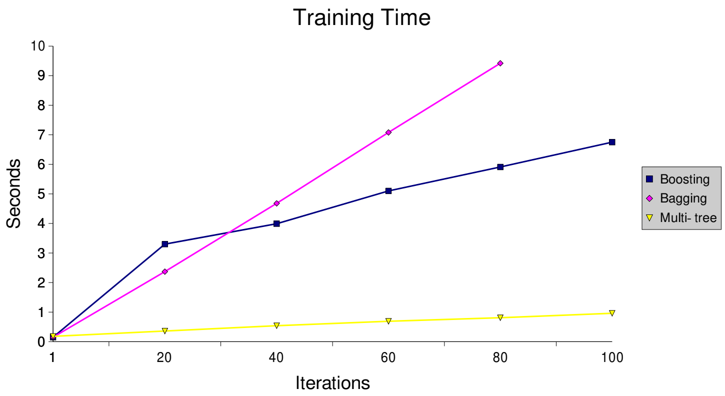

Fig. 2. Training time required by the ensemble methods depending on the number of iterations.

We construct the multi-tree by a random extraction from the node list. Therefore, our technique is also related to randomisation [8]. This technique constructs an ensemble of decision trees by injecting randomness into the splitting criterion. The idea is to randomly select among the top 20 best splits. In this way, the method generates a different decision tree for each iteration. However, randomisation builds forests, i.e. an ensemble of completely independent decision trees.

Lastly, we have included a comparison of the ensemble method with some well-known ensemble methods, namely boosting and bagging. Due to the sharing of the common parts, much less time is required than with classical ensemble approaches to perform the same number of iterations. Consequently, our system is very appropriate for complex problems where other ensemble methods such as boosting or bagging require huge amounts of memory and time.

As future work, we propose the study of a new strategy for generating trees. This strategy would be different from the current random technique we have employed to explore OR-nodes, and would probably be based on the semantic discrepancy of classifiers (using kappa function [14], for instance). This technique would provide a way to improve the results of our ensemble method with fewer iterations.

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. W. Buntine. *A Theory of Learning Classification Rules.* PhD thesis, School of Computing Science in the University of Technology, Sydney, February 1990.
4. W. Buntine. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall,London, 1993.
5. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
6. T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. of the 7th National Conference on Artificial Intelligence*, pages 49–54, 1988.
7. T. G Dietterich. Ensemble methods in machine learning. In *First International Workshop on Multiple Classifier Systems*, pages 1–15, 2000.
8. 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.
9. 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.
10. 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 *Lecture Notes in Computer Science*, pages 204–213, 2002.
11. 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.
12. R. Kohavi and C. Kunz. Option decision trees with majority votes. In *Proc. 14th International Conference on Machine Learning*, pages 161–169. Morgan Kaufmann, 1997.
13. L. Kuncheva. A Theoretical Study on Six Classifier Fusion Strategies. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(2):281–286, 2002.
14. L. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Submitted to Machine Learning, 2002.
15. N.J. Nilsson. *Artificial Intelligence: a new synthesis.* Morgan Kaufmann, 1998.
16. J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.