Boosted Tree Ensembles for Solving Multiclass Problems

Terry Windeatt and Gholamreza Ardeshir

Centre for Vision, Speech and Signal Processing (CVSSP)
University of Surrey, Guildford, Surrey, GU2 5XH, U.K.
T.Windeatt,G. Ardeshir @eim.surrey.ac.uk

Abstract. In this paper we consider the combination of two ensemble techniques, both capable of producing diverse binary base classifiers. Adaboost, a version of Boosting is combined with Output Coding for solving multiclass problems. Decision trees are chosen as the base classifiers, and the issue of tree pruning is addressed. Pruning produces less complex trees and sometimes leads to better generalisation. Experimental results demonstrate that pruning makes little difference in this framework. However, on average over nine benchmark datasets better accuracy is achieved by incorporating unpruned trees.

1 Introduction

Traditionally, the approach that has been used in the design of pattern classification systems is to experimentally assess the performance of several classifiers with the idea that the best one will be chosen. However, the theory of ensemble classifiers represents a departure from the traditional strategy and has been developed to address the problem of designing a system with improved accuracy. Recognising that each classifier may make different and perhaps complementary errors, the aim is to pool together the results from all classifiers in such a way that the ensemble outperforms any constituent (also called base) classifier. There are several categories of techniques capable of producing diversity among base classifiers, which is a necessary condition for improvement by combining. In the first category are methods that reduce dimension of training set to give different feature sets. The second category includes methods that incorporate different types of base classifier or different base classifier parameters. In the third category, which includes Boosting, are techniques that resample the training set and thereby specialise each classifier on a different subset. Finally in the fourth category are Output Coding methods that create complementary two-class problems from multiclass problems.

Ensemble methods from these four categories have been developed and tuned over the past decade. In principle, the problem of creating a good ensemble is solved by jointly optimising the design and fusion of base classiers. However this is a difficult optimisation, and frequently a relatively simple fusion strategy is applied, such as majority or weighted vote. Base classifiers are then designed to

match the fusion rule. In this paper, two ensemble design strategies are combined. Adaboost which is from category three is combined with Output Coding from category four. The idea is to convert a multiclass problem into complementary binary sub-problems and at the same time concentrate on difficult-to-classify patterns by reweighting; the idea was first reported in [20]. In this study, decision trees are chosen as base classifer and the effect of tree pruning on ensemble performance is investigated. An advantage of using decision trees over other base classifiers is that the decision to prune is the only parameter to set. Pruning reduces the complexity of base classifiers, but does not necessarily lead to improved accuracy.

Boosting adaptively changes the distribution of the training set based upon the performance of sequentially constructed classifiers. Each new classifier is used to adaptively filter and re-weight the training set, so that the next classifier in the sequence has increased probability of selecting patterns that have been previously mis-classified. Boosting is a general method for converting a weak learner into one with high accuracy, but it requires that the weak learning algorithm produce hypotheses with accuracy better than random guessing. For a k-class problem, $k >> 2$, it may be difficult to achieve the required accuracy. One solution proposed by Freund and Schapire, referred to as AdaBoost.M2, is based on a pseudo-loss measure in which the weak learner chooses from a set of plausible labels [20]. AdaBoost.M2 not only divides the training set into hard and easy patterns, but also forces the learning algorithm to concentrate on patterns whose labels are hard to distinguish from each other. Another solution is to combine Boosting with Output Coding (Adaboost.OC), in which the weak learner is rerun on the training patterns while reweighting and relabelling on each round [20]. Output Coding is an ensemble method in which a binary code matrix defines the decomposition of the multiclass into binary sub-problems. Each classifier operates on the same training set but the patterns are relabelled according to the columns of the Output Coding matrix. This process of relabelling was referred to as colouring of the patterns in the context of Adaboost.OC. By introducing colouring into Boosting the weak learner no longer uses the same training set on each round. The main advantage is that Adaboost is then only required to solve a two-class problem.

There have been some previous studies that have compared pruned and unpruned ensembles. In [12], ECOC with C4.5 showed no significant difference on pruned versus unpruned ensembles on six out of eight datasets. In [10] over thirty-three datasets, significant difference due to pruning was observed in ten of the datasets for both C4.5 and randomised C4.5, in four of the datasets for Bagged C4.5 and in none for Boosted C4.5. In [23], six pruning strategies were compared and Error-Based Pruning (EBP) performed better overall than any other strategy for both Bagging and Boosting. Similarly in [21] six methods were compared for Output Coding with C4.5 and overall EBP performed better than any other strategy. In this paper the purpose is to study the effect of pruning in the context of output coded ensembles of boosted trees, rather than to compare boosted ensembles with other techniques.

The structure of this paper is as follows: in Section 2 we review the Output Coding method, followed by a review of AdaBoost.OC in Section 3. In section 4 we briefly explain the Error-based Pruning method (EBP), the default strategy for C4.5. EBP has been used for the experiments in Section 5 in which the performance of unpruned and pruned Adaboost.OC is compared with Adaboost.M2.

2 Output Coding and diverse binary classifiers

Output Coding is a two-stage method, the first being the relabelling stage which can be defined as follows. Let Z be the $k \times b$ code matrix with binary elements, where K is the number of classes and b is the number of binary classifiers. Each column provides a map to convert the multi-class problem to binary sub-problems. Specifically, for the jth sub-problem, a training pattern with target class w_i $(i = 1...k)$ is re-labelled either as class Ω_1 or as class Ω_2 depending on the value of Z_{ij} (typically zero or one). Therefore for each column the k classes can be considered to be arranged into two *super-groups* of classes Ω_1 and Ω_2. (The second stage of Output Coding is the decision rule which is based on finding distance to each row of Z that acts as a b-dimensional code word to represent a class).

The Output Coding method was introduced in [11, 12] and named Error-Correcting Output Coding (ECOC). The idea was to base the code on error-correcting principles to facilitate a robust fusion strategy. Various coding strategies have since been proposed, but most code matrices that have been investigated previously are binary and problem-independent, that is pre-designed. Optimal properties of the code matrix for producing diverse classifiers are believed to be maximum Hamming Distance between pairs of columns [12, 24]. Random codes have received much attention, and were first mentioned in [11] as performing well in comparison with error-correcting codes. In [12] random, exhaustive, hill-climbing search and BCH coding methods were used to produce ECOC code matrices for different column lengths. Random codes were also shown in [14] to give Bayesian performance if pairs of code words were equidistant, and it was claimed that a long enough random code would not be outperformed by a pre-defined code. In [25] it is shown that an optimal code performs better than random code as code word length is reduced. Recent developments include investigation of three-valued codes [2], and problem-dependent continuous and discrete codes [8].

The output coding concept has been successfully applied to problems in several domains [1, 3, 4, 15]. It has also been shown to improve performance with different kinds of base classifier including decision tree, multi-layer perceptron, SVM and k-nearest-neighbour.

3 AdaBoost.OC Algorithm

AdaBoost.OC reruns the weak learning algorithm many times and trains it over a new distribution of examples, relabelled as in Output Coding. As can be seen

in figure 1, given a training set with the size m, the weights of patterns are initialised as in AdaBoost.M2, that is

$$\tilde{D}_1(i,\ell) = [\![\ell \neq y_i]\!]/(m(k-1)) \tag{1}$$

where k is the number of classes and $[\![x]\!]$ is 1 when x is true otherwise 0. On each round, patterns are relabelled according to the colouring (μ_t). To minimise the training errors, a colouring scheme should be chosen to maximise U_t, which is defined in step two of figure 1. We have used the scheme from [20] which uses a random assignment of $k/2$ labels to 0 and $k/2$ labels to 1. The weak learner is trained according to D_t and then by calculating the weak hypothesis h_t, all of the labels satisfying $h_t(x) = \mu_t(\ell)$ receive a vote. The weight of this weak hypothesis (α_t), is found from the weighted training error, $\tilde{\epsilon}_t$, is which includes penalty terms for failing to include the correct label in plausible label set and for including any incorrect labels. The weights of patterns are updated so that the learning algorithm is forced to concentrate on hardest patterns and on the labels that are hard to distinguish from each other. Finally the generated hypotheses are combined according to weighted voting so that a test pattern is assigned to the class whose label has received the most votes.

4 Decision Tree Pruning

In the past much effort has been directed toward developing effective tree pruning methods (for a review see [13]) in the context of a single tree. For a tree ensemble, besides base classifier pruning, it is also possible to consider ensemble pruning. In [9], five ensemble pruning methods used with Adaboost are proposed, but the emphasis there is on efficiency, i.e. finding a minimal number of base classifiers without significantly degrading performance.

Decision tree pruning is a process in which one or more subtrees of a decision tree are removed. The need for pruning arises because the generated tree can be large and complex, so it may not be accurate or comprehensible. Complexity of a univariate decision tree is measured as the number of nodes, and the reasons for complexity are mismatch of representational biases and noise [7]. It means that the induction algorithm is unable to model some target concepts, and also that in some algorithms, (e.g. C4.5), subtree replication causes the tree to be too large and to overfit [17].

According to [13] there are different types of pruning methods but post-pruning is more usual [7]. The disadvantage of pre-pruning methods is that tree growth can be prematurely stopped, since the procedure estimates when to stop constructing the tree. A stopping criterion that estimates the performance gain expected from further tree expansion is applied, and tree expansion terminates when the expected gain is not accessible [13], [18]. A way around this problem is to use a post-pruning method which grows the full tree and retro-spectively prunes, starting at the leaves of the tree. Post-pruning methods remove one or more subtrees and replace them by a leaf or one branch of that subtree. One class of these algorithms divides the training set into a growing set and pruning set.

The growing set is used to generate the tree as well as prune, while the pruning set is used to select the best tree [6]. In the case of shortage of training set, the cross-validation method is used i.e. the training set is divided into several equal-sized blocks and then on each iteration one block is used as pruning set and the remaining blocks used as a growing set. Another class of post-pruning algorithm uses all the training set for both growing and pruning [19]. However it is then necessary to define an estimate of the true error rate using the training set alone.

EBP was developed by Quinlan for use in C4.5. It does not need a separate pruning set, but uses an estimate of expected error rate. A set of examples covered by the leaf of a tree is considered to be a statistical sample from which it is possible to calculate confidence for the posterior probability of mis-classification. The assumption is made that the error in this sample follows a binomial distribution, from which the upper limit of confidence [16] is the solution for p of

$$CF = \sum_{x=0}^{E} \binom{N}{x} p^x (1-p)^{N-x} \tag{2}$$

where N is number of cases covered by a node and E is number of cases which is covered by that node erroneously (As C4.5 we have used an approximate solution for equation 2). The upper limit of confidence is multiplied by the number of cases which are covered by a leaf to determine the number of predicted errors for that leaf. Further the number of predicted errors of a subtree is the sum of the predicted errors of its branches. If the number of predicted errors for a leaf is less than the number of predicted errors for the subtree in which that leaf is, then the subtree is replaced with the leaf. We tried changing the confidence level to vary the degree of pruning but found that this is not a reliable way of varying tree complexity [22]. In the experiments in Section 5 the default confidence level of 25% is used.

5 Experiments

The number of patterns, classes and features for the datasets used in these experiments are shown in table 1. These data sets can be found on UCI web site [5]. The data has been randomly split 70/30% into training/test set and the experiment repeated ten times. C4.5, with Error-Based Pruning, has been used as base classifier and in these experiments ensemble of unpruned and pruned trees built by AdaBoost.M2 and AdaBoost.OC have been compared. In the experiments reported in [20] for comparing AdaBoost.M2 and AdaBoost.OC pruning was turned on.

As figures 2 and 3 show, the test error of AdaBoost.OC for both pruned and unpruned trees reduces as number of classifier increases over all datasets. However, the test error of AdaBoost.M2 appears somewhat insensitive to number of rounds although error increases slightly for Car, Glass and Soybean-Large. This appears to show that the pseudo-loss based algorithm AdaBoost.M2 with

C4.5, in contrast to the error-based Adaboost.M2 with C4.5, only takes a few rounds to learn the training set and in some cases overfits as number of rounds increases.

For AdaBoost.OC the performance of pruned ensembles is similar to the performance of unpruned ensembles, although appears slightly better for Anneal, Audiology, Glass, Iris. However the difference did not show up as significant (McNemar 5%). For AdaBoost.M2, the pruned ensemble outperforms the unpruned ensemble except for Dermatology.

In an attempt to see overall performance the composite curve over all datasets is plotted in figure 4. The plot is normalised with respect to the best classification rate for each particular dataset as follows. The best classification rate is the mean over the last x rounds, where x is judged to be the number of classifiers above which there is no further improvement. From the composite performance we see that the mean classification rate over the nine datasets for Adaboost.OC is higher for unpruned than pruned. For AdaBoost.M2, ensemble of unpruned decision trees outperforms pruned ensemble.

6 Discussion

As explained in Section 2 the choice of code matrix may affect performance of Output Coding. In AdaBoost.OC, we have used a random approach to dividing the labels on each round. This may mean that diversity is not as great as it could be, since the Hamming Distance between columns has not been maximised. Providing that the code matrix has enough columns random code may perform as well as an optimal code. However, since each round of Boosting depends on the previous round it may be important for efficiency reasons to minimise the total number of rounds. Pursuing optimal codes is a possible way forward in this regard and would likely modify the curves shown in figure 4 at the low end. AdaBoost.M2 is a time-consuming algorithm because on each round it constructs classifiers according to the same number of classes and in comparison AdaBoost.OC has removed this complexity.

7 Conclusion

In this paper, we have applied two multi-class versions of Boosting to nine datasets by using pruned and unpruned decision trees as base classifiers, and we have seen that AdaBoost.OC outperforms AdaBoost.M2. For AdaBoost.OC, the performance of pruned ensemble is similar to the unpruned ensemble but on average over all datasets it is better not to prune if accuracy is the main consideration.

References

1. D.W. Aha and R. L. Bankert. Cloud classification using error-correcting output codes. *Artificial Intelligence Applications: Natural Resources, Agriculture, and Environmental Science*, 11(1):13–28, 1997.

Table 1. Specification of Datasets

Name	Data size	Class	Attributes	
			Cont.	Disc.
Anneal	898	6	9	29
Audiology	200	24		69
Car	1728	4		6
Dermatology	366	6	1	33
Glass	214	6	9	
Iris	150	3	4	
Segmentation	2310	7	19	
Soybean-Large	683	19		35
Vehicle	846	4	18	

2. E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing multi-class to binary: A unifying approach for margin classifiers. *Machine learning research*, 1:113–141, 2000.
3. G. Bakiri and T. Dietterich. Achieving high-accuracy text-to-speech with machine learning, 1999.
4. A. Berger. Error-correcting output coding for text classification. In *IJCAI'99, Workshop on machine learning for information filtering*, 1999.
5. C.L. Blake and Merz C.J. Uci repository of machine learning databases. Technical report, Irvine, Univ. of Calif., Inf. and Comp. Scienc, 1998.
6. Leo Breiman, J. H. Freidman, R. A. Olshen, and C.J. Stone. Classification and regression trees. *Wadsworth International Group*, 1984.
7. L. A. Breslow and D. W. Aha. Simplifying decision trees: A survey. *Knowledge Engineering Review*, pages 1–40, 1997.
8. K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, to appear.
9. T. G. Dietterich D. Margineantu. Pruning adaptive boosting. In *International Conference on Machine Learning*, pages 211–218. Morgan Kaufmann, 1997.
10. T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–158, 2000.
11. T.G Dietterich and G. Bakiri. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 572–577. AAAI Press, 1991.
12. T.G. Dietterich and G Bakiri. Solving multi-class learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
13. F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.
14. G. M. James and T. Hastie. The error coding method and PICT's. *Computational and Graphical Statistics*, 7:377–387, 1998.
15. J. Kittler, R. Ghaderi, T. Windeatt, and G. Matas. Face verification using error correcting output codes. In *Computer Vision and Pattern Recognition CVPR01*, volume 1, pages 755–760, Hawaii, December 2001. IEEE Press.

$Given (x_1, y_1), ..., (x_m, y_m) where x_i \in X, y_i \in Y.$
For $t = 1, ..., T$

1. Compute colouring $\mu_t : Y \rightarrow \{0, 1\}$.
2. Let $U_t = \sum_{i=1}^{m} \sum_{\ell \in Y} \tilde{D}_t(i, \ell) [\![\mu_t(y_i) \neq \mu_t(\ell)]\!]$
3. Let $D_t(i) = \frac{\sum_{\ell \in Y} \tilde{D}_t(i,\ell) [\![\mu_t(y_i) \neq \mu_t(\ell)]\!]}{U_t}$
4. Train weak learner on $(x_1, \mu_t(y_1)), ..., (x_m, \mu_t(y_m))$ weighted according to D_t
5. Get weak hypothesis $h_t : X \rightarrow \{0, 1\}$.
6. Let $\tilde{h}_t = \{\ell \in Y : h_t(x) = \mu_t(\ell)\}$.
7. Let $\tilde{\epsilon}_t = \frac{1}{2} \sum_{i=1}^{m} \sum \tilde{D}_t(i, \ell) \cdot ([\![y_i \notin h(\tilde{x}_i)_t]\!] + [\![\ell \in h(\tilde{x}_i)_t]\!])$
8. Let $\alpha_t = \frac{1}{2} \ln(\frac{1-\tilde{\epsilon}_t}{\tilde{\epsilon}_t})$
9. $\tilde{D}_{t+1}(i, \ell) = \frac{\tilde{D}_t(i,\ell) \cdot exp(\alpha_t([\![y_i \notin \tilde{h}_t(x_i)]\!] + [\![\ell \in \tilde{h}_t(x_i)]\!]))}{Z_t}$
   where Z_t is a normalisation factor chosen so that $\tilde{D}_{t+1}$ will sum to 1

Output the final hypothesis:

$$H_{final}(x) = arg \max_{l \in Y} \sum_{t=1}^{T} \alpha_t [\![h_t(x) = \mu_t(\ell)]\!].$$

Fig. 1. The AdaBoost.OC algorithm

16. J. Ross Quinlan. Personal communication from Quinlan.
17. R. Quinlan. Induction of decision tree. *Machine Learning*, 1:81–106, 1986.
18. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo,California, 1993.
19. R.J. Quinlan. Simplyfying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.
20. R.E. Schapire. Using output codes to boost multiclass learning problems. In *14th International Conf. on Machine Learning*, pages 313–321. Morgan Kaufman, 1997.
21. T. Windeatt and A. Ardeshir. Tree pruning for output coded ensembles. In *International Conference of Pattern Recognition*, Quebec, Canada, 2002. submitted.
22. T. Windeatt and G. Ardeshir. Boosting unpruned and pruned decision trees. In *Applied Informatics, Preceedings of the IASTED International Symposia*, pages 66–71, 2001.
23. T. Windeatt and G. Ardeshir. An empirical comparison of pruning methods for ensemble classifiers. In *IDA 2001*. Springer-Verlag, Lecture notes in computer science, 2001.
24. T. Windeatt and R. Ghaderi. Multi-class learning and error-correcting code sensitivity. *Electronics Letters*, 36(19):1630–1632, Sep 2000.
25. T. Windeatt and R. Ghaderi. Binary labelling and decision level fusion. *Information Fusion*, 2(2):103–112, 2001.

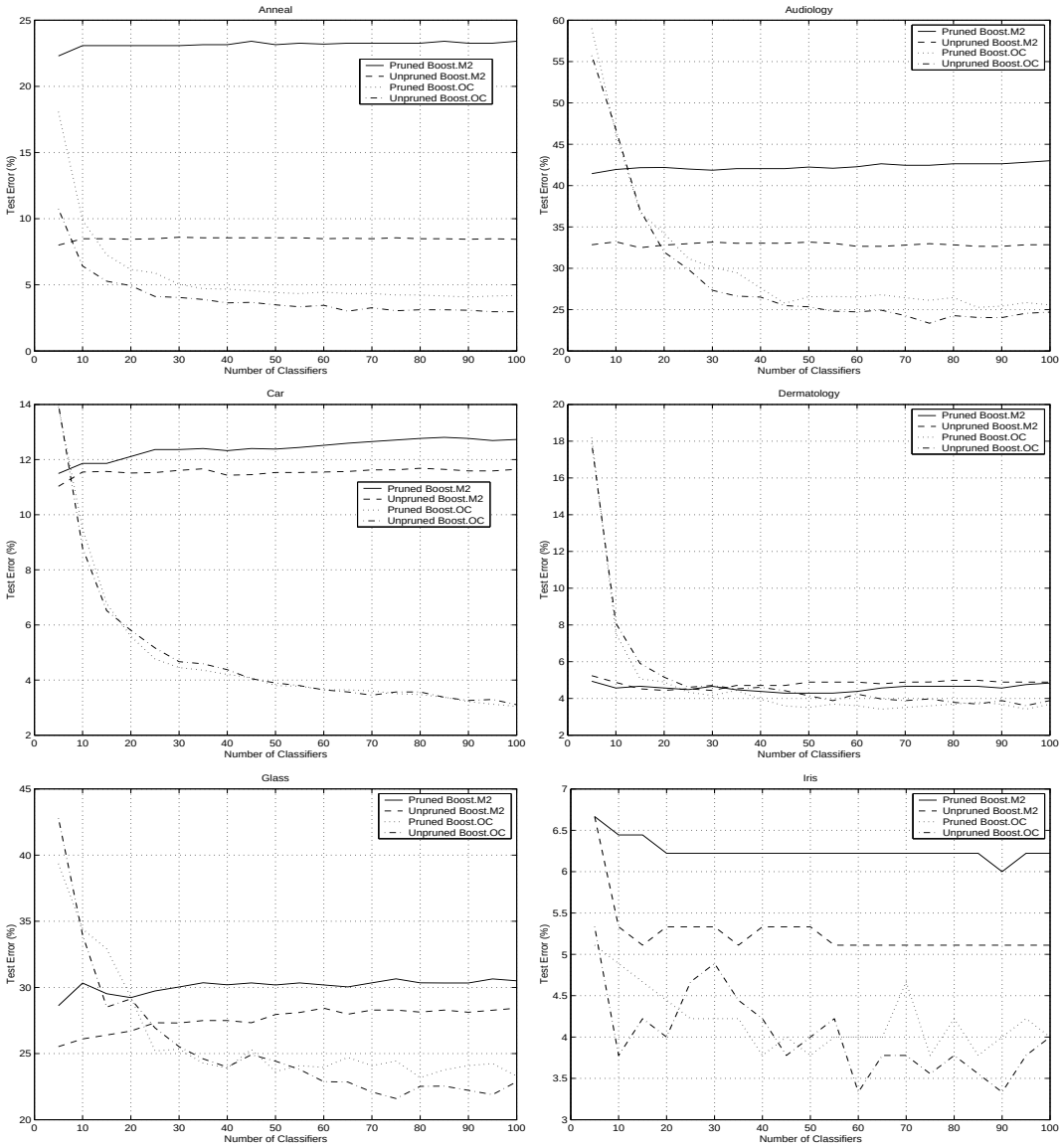

Fig. 2. Test Error of AdaBoost.M2 and AdaBoost.OC

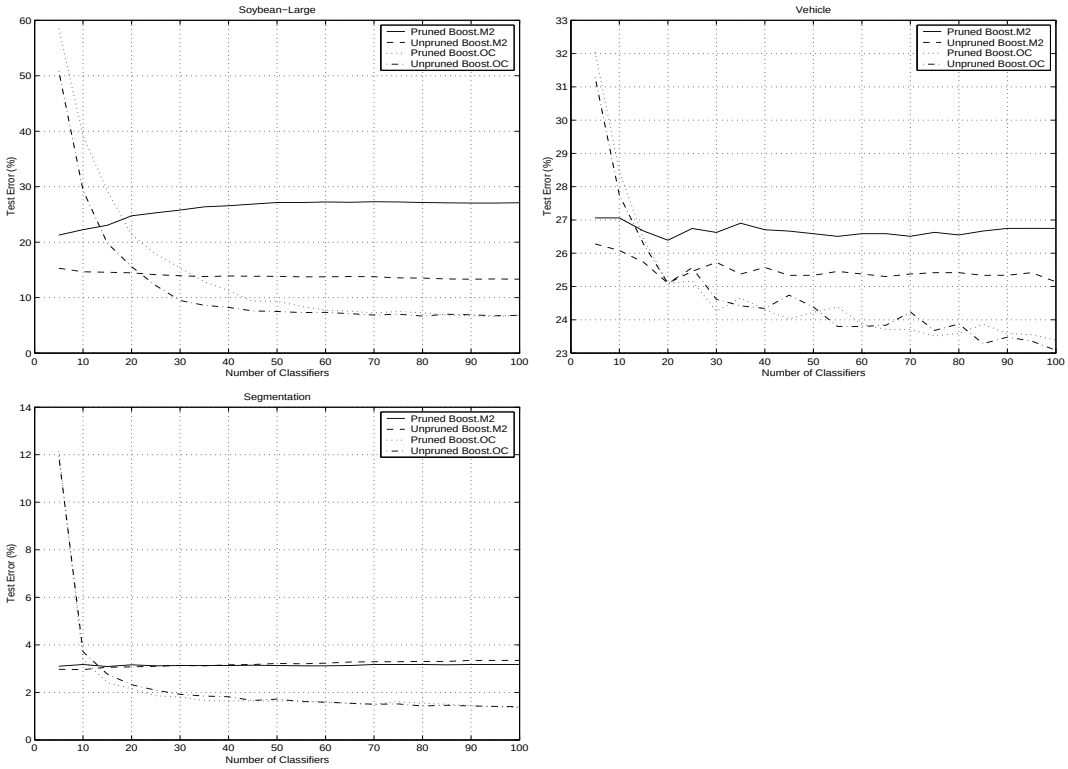

Fig. 3. Test Error of AdaBoost.M2 and AdaBoost.OC

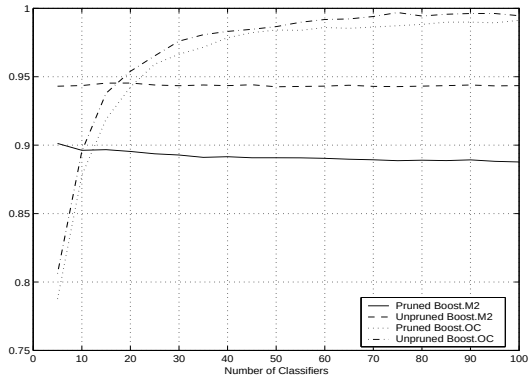

Fig. 4. Composite correct Classification of AdaBoost.M2 and AdaBoost.OC for pruned and unpruned decision trees