Closed-form dual perturb and combine for tree-based models

Pierre Geurts P.GEURTS@ULG.AC.BE
Louis Wehenkel L.WEHENKEL@ULG.AC.BE
University of Liège, Department of Electrical Engineering and Computer Science, B-4000 Liège, Belgium

Abstract

This paper studies the aggregation of predictions made by tree-based models for several perturbed versions of the attribute vector of a test case. A closed-form approximation of this scheme combined with cross-validation to tune the level of perturbation is proposed. This yields soft-tree models in a parameter free way, and preserves their interpretability. Empirical evaluations, on classification and regression problems, show that accuracy and bias/variance tradeoff are improved significantly at the price of an acceptable computational overhead. The method is further compared and combined with tree bagging.

1. Introduction

Ensemble methods are used in machine learning essentially because of the important improvement in accuracy they can bring to learning algorithms like decision trees or neural networks. Among them, the perturb and combine (PC) algorithms (Breiman, 1998) consist in perturbing an algorithm so as to produce different models from a learning sample. The predictions of these models are combined to produce a final prediction potentially better than the individual ones. The most well-known example of PC is bagging (Breiman, 1996) where the different models are produced by resampling the available data before building a model. Most PC algorithms are very effective in terms of accuracy improvement with respect to the original algorithm. However, they suffer from two main drawbacks. First, computing times and memory space are typically multiplied by the number of models which are aggregated (e.g. two orders of magnitude). Second, the aggregation step jeopardizes the interpretability of the original algorithm.

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

In this paper, we consider the generic Dual Perturb and Combine (DPC) algorithm first introduced in (Geurts, 2001). Unlike PC, DPC uses only one model and delays to the prediction stage the generation of multiple predictions by perturbing the attribute vector corresponding to a test case. A closed-form approximation of the asymptotic prediction (obtained with an infinite number of perturbations) of DPC exists for tree-based models. In this paper, classification and regression trees are treated in parallel and a cross-validation technique is proposed to tune automatically the degree of perturbation. The resulting soft-tree algorithm is analyzed from the viewpoint of accuracy, bias/variance tradeoff, and computational overhead.

The paper is structured as follows. Section 2 recalls the closed-form approximation of DPC for tree-based models, and describes the tuning of the degree of perturbation by cross-validation. Section 3 provides experimental results on classification and regression tasks, with DPC on single trees and tree bagging. Section 4 analyses the main properties of the proposed method and Section 5 discusses its relationship with other soft tree models proposed in the literature.

2. Dual Perturb and Combine

We first introduce terminology and recall the standard PC framework.

A (deterministic) learning algorithm builds a function $f_{ls,\pi}(\cdot)$ over an attribute space, to approximate a random variable y in terms of an attribute vector $\underline{x}$. Its inputs are a sample $ls = \{\underline{x}_i, y_i\}_{i=1}^N$ of joint observations and some algorithm specific parameters π. On top of such an algorithm, PC proceeds in the following generic way to derive another one:

- **Learning stage:** for i going from 1 to T:
  - draw a random vector $\underline{\epsilon}^i$ from a distribution $P(\underline{\varepsilon}|ls)$, and perturb the inputs of the learning algorithm yielding $ls_{\underline{\epsilon}^i}$ and $\pi_{\underline{\epsilon}^i}$
  - construct the function $f_{ls_{\underline{\epsilon}^i},\pi_{\underline{\epsilon}^i}}(\cdot)$

- **Prediction stage:** compute the prediction by

$$f^T_{PC}(\underline{x}) = \text{aggr}^T_{i=1}\{f_{ls_{\underline{\epsilon}^i},\pi_{\underline{\epsilon}^i}}(\underline{x})\}, \quad (1)$$

  where "aggr" stands for average in regression and majority vote or average of probability estimates in classification.

Examples of PC methods proposed in the literature are bagging (Breiman, 1996), random subspace (Ho, 1998), output smearing (Breiman, 2000), random trees (Dietterich, 2000), random forests (Breiman, 2001).

2.1. Generic DPC algorithm

The algorithm works as follows (Geurts, 2001):

- **Learning stage:** build a (single) model $f_{ls,\pi}$.
- **Prediction stage:** at a point $\underline{x}$:
  - for $i = 1 \ldots T$, let $x_{\underline{\epsilon}^i}$ denote a perturbed version of $\underline{x}$, where the $\underline{\epsilon}^i$ are drawn independently from a distribution $P(\underline{\varepsilon}|ls)$;
  - compute the average prediction given by:

$$f^T_{DPC}(\underline{x}) = \text{aggr}^T_{i=1}\{f_{ls,\pi}(\underline{x}_{\underline{\epsilon}^i})\}, \quad (2)$$

    where "aggr" is defined as above.

This *generic* DPC method can be applied on top of any model produced by any learning algorithm.

2.1.1. Perturbation scheme

In practice, several more or less complicated perturbation schemes could be imagined, depending also on the attribute type (numerical or symbolic). In this paper, we consider only numerical attributes and we use the following additive perturbation scheme:

$$\underline{x}_{\underline{\epsilon}^i} = \underline{x} + \underline{\epsilon}^i, \quad (3)$$

where $\underline{\epsilon}^i$ is a realization of a random vector $\underline{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m)$, where each component ε_i is drawn independently from a Gaussian distribution $N(0, \lambda\sigma_i)$, where $\lambda \geq 0$ is a parameter and σ_i is the standard deviation of the attribute x_i in the learning sample.

Obviously, if $\lambda = 0$ the DPC version is identical in terms of predictions to the base model. On the other hand, the higher λ the larger the perturbation.

2.1.2. Effect of parameter T

For finite T, DPC computes a sample estimate of the expectation of the model output according to the perturbation distribution. Therefore T is like the number of models in PC: the higher it is, the better in average. However, for a linear regression model and $T \to \infty$, DPC has no effect at all. Indeed, the average prediction (2) becomes in this case:

$$\begin{aligned} f^\infty_{DPC}(\underline{x}) &= \lim_{T\to\infty} \frac{1}{T}\sum_{i=1}^T \underline{w} \cdot \underline{x}_{\underline{\epsilon}^i} \\ &= \underline{w} \cdot \underline{x} + \lim_{T\to\infty} \frac{1}{T}\sum_{i=1}^T \underline{w} \cdot \underline{\epsilon}^i = \underline{w} \cdot \underline{x}. \end{aligned}$$

This means also that, for finite T, DPC actually increases the average error of linear regression models. Nevertheless, when applied to a nonlinear model, DPC may increase accuracy if T is large enough.

2.2. Closed-form version for tree-based models

For tree-based models it is possible to derive a deterministic closed-form approximation of the prediction corresponding to $T \to \infty$ (Geurts, 2001).

Indeed, a regression tree[1] recursively partitions the input space into (terminal) regions where the prediction is constant. Denoting by $\mathcal{L}_j$ $(j = 1, \ldots, L)$ its leaves and by g_j $(j = 1, \ldots, L)$ some numerical predictions associated to these latter, the prediction given by the tree at a point $\underline{x}$ may be written as:

$$g(\underline{x}) = \sum_j 1(\underline{x} \to \mathcal{L}_j) \cdot g_j, \quad (4)$$

where $1(\underline{x} \to \mathcal{L}_j)$ is the characteristic function of the set of objects reaching $\mathcal{L}_j$. On the other hand, the asymptotic $(T \to \infty)$ prediction given by DPC is:

$$\begin{aligned} g^\infty_{DPC}(\underline{x}) &= E_{\underline{\varepsilon}}\{g(\underline{x} + \underline{\varepsilon})\} & (5)\\ &= \sum_j E_{\underline{\varepsilon}}\{1(\underline{x} + \underline{\varepsilon} \to \mathcal{L}_j)\} \cdot g_j & (6)\\ &= \sum_j P_{\underline{\varepsilon}}(\underline{x} + \underline{\varepsilon} \to \mathcal{L}_j) \cdot g_j, & (7) \end{aligned}$$

where $P_{\underline{\varepsilon}}(\underline{x}+\underline{\varepsilon} \to \mathcal{L}_j)$ is the probability that a perturbation of $\underline{x}$ reaches $\mathcal{L}_j$. Denoting by T_1,T_2,...,T_{N_j}, the tests along the path towards $\mathcal{L}_j$, we have:

$$P_{\underline{\varepsilon}}(\underline{x}+\underline{\varepsilon} \to \mathcal{L}_j) = P_{\underline{\varepsilon}}(T_1(\underline{x}+\underline{\varepsilon}) \wedge \ldots \wedge T_{N_j}(\underline{x}+\underline{\varepsilon})). \quad (8)$$

which can be factored (independence of the ε_i) into:

$$P_{\underline{\varepsilon}}(\underline{x}+\underline{\varepsilon} \to \mathcal{L}_j) = P_{\underline{\varepsilon}}(T_1(\underline{x}+\underline{\varepsilon})) \cdots P_{\underline{\varepsilon}}(T_{N_j}(\underline{x}+\underline{\varepsilon})), \quad (9)$$

under the simplifying assumption that each attribute is tested only once along the path. Finally, assuming that the tests are of the form $[x_i < (\geq) x_{th}]$, the

[1] For the sake of simplicity, we consider only binary trees using axis-parallel splits.

probability of a test being true is computed by:

$$\begin{aligned} P_{\underline{\varepsilon}}(T(\underline{x}+\underline{\varepsilon})) &= P_{\underline{\varepsilon}}(x_i+\varepsilon_i < (\geq) x_{th}) && (10) \\ &= P(Z < (\geq) \frac{x_{th}-x_i}{\lambda\sigma_i}), && (11) \end{aligned}$$

where Z is a $N(0,1)$ random variable.

For each test node, the probability expressed in (11) can be obtained by table look-up. On the other hand, the computation of all probabilities (8) can be done by propagating once the test case $\underline{x}$ from the root node to the leaves, starting with a probability of 1.0 at the root and multiplying this probability by the probabilities associated to the arcs which are traversed. Then, prediction (5) is obtained by averaging the predictions at leaf nodes according to the probability distribution (8). All in all, the complexity of the computation of (5) is thus proportional to the tree complexity.

The above derivation was made for the case of numerical labels, aggregated by averaging. In the case of classification, one can nevertheless show that for both types of aggregation operators (majority of class votes or averaging of class probability vectors) the scheme is (in asymptotic conditions) equivalent to taking the expected value of a class indicator or a class probability vector and by defining the prediction from it. Hence the closed-form implementation also holds for the two ways of aggregating classifications.

Obviously, this closed-form implementation yields a soft tree: instead of taking the "hard" (or crisp) decision of propagating a test case either to the right or to the left successor of a test node, DPC propagates this case in both directions with some weight depending on the noise level and on the attribute vector.

2.2.1. Tuning of the smoothing parameter λ

Figure 1 illustrates the smoothing effect of DPC on a simple one-dimensional regression task, comparing an un-pruned CART tree (curve labeled $\lambda = 0$) and its DPC versions (for $\lambda = 0.09$ and $\lambda = 0.5$). The graph also shows the learning sample used to grow the tree together with the underlying output function.

We will see in Section 3.4 that the choice of λ has to do with the bias/variance tradeoff, and its optimal value from the accuracy point of view is therefore rather problem specific. Hence, we propose to use 10-fold cross-validation to determine its optimal value for a given dataset. In the case of un-pruned trees, this leads to a significant computational overhead at learning time, since several trees will have to be grown and tested on the different folds. For pruned trees, however, the overhead can be mitigated by re-using the

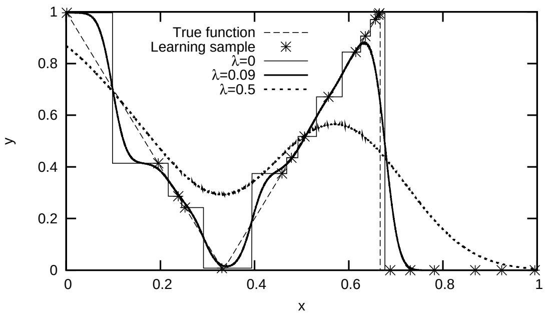

Figure 1. Smoothing effect of DPC on a regression tree.

same set of trees and folds already constructed by the pruning algorithm. On the other hand, in the context of tree bagging, one can use out-of-bag estimates instead of 10-fold cross-validation, and this also mitigates the computational overhead.

In our experimentations, we exploited these two ideas together with a simple bisection search to determine the best value of λ.

3. Experimentations

3.1. Datasets and protocol

Experiments are conducted on 10 classification and 10 regression problems which are summarized in Table 1. The only criterion used to select these problems is that all input variables are numerical. Most datasets are available in the UCI Machine Learning Repository (Blake & Merz, 1998). Friedman1 and Two-norm are two artificial problems introduced respectively in (Friedman, 1991) and (Breiman, 1998). Pumadyn and Hwang come from the DELVE repository[2] and the last three regression problems are from (Torgo, 1999)[3].

To evaluate algorithms, each dataset is split into a learning sample (LS) and test sample (TS) whose sizes are given in Table 1. They are all run on the same learning sample and their errors are estimated on the corresponding test sample. This procedure is repeated 10 times by randomizing the LS/TS split, and errors are averaged over these runs. On smaller datasets (marked by a star in Table 1), 50 runs are used.

Figure 2 gives, for each dataset, the results obtained by several algorithms alone or combined with DPC. Each graph provides average error (error rate in classification, mean square-error multiplied by the factor given in the last column of Table 1 in regression) and its standard deviation over the 10 or 50 runs. In left to right order we provide the results of (un-pruned) Single

[2] `http://www.cs.utoronto.ca/~delve`.
[3] `http://www.liacc.up.pt/~ltorgo`.

Table 1. Dataset summaries

Dataset	# Atts	LS size	TS size	# Class
Waveform⋆	21	300	4700	3
Two-norm⋆	20	300	9700	2
Vehicle⋆	18	761	85	4
Vowel⋆	10	891	99	11
Segment⋆	19	2079	231	7
Spambase	57	3221	1380	2
Satellite	36	4435	2000	6
Pendigits	16	7494	3498	10
Dig44	16	9000	9000	10
Letter	16	10000	10000	26
Dataset	# Atts	LS size	TS size	Err×
Friedman1⋆	10	300	9700	1
Housing⋆	13	455	51	1
Hwang-f5	2	2000	11600	10^3
Hwang-f5n	2	2000	11600	10^2
Pumadyn-32fh	32	2000	6291	10^4
Pumadyn-32nm	32	2000	6291	10^5
Abalone	8	3133	1044	1
Ailerons	40	5000	8750	10^8
Elevators	18	5000	11559	10^6
Poletelecomm	48	5000	10000	10^{-1}

Trees (ST), Pruned Single Trees (PST) and Tree Bagging (TB), interleaved with the corresponding (closed-form) DPC variants (ST^d, PST^d, and TB^d).

The score measure used for growing regression trees is the amount of variance reduction, whereas in classification we used a normalized version of Shannon entropy (Wehenkel, 1998). Single trees are grown fully in classification, whereas in regression we stopped splitting nodes having less than 5 observations. Trees are pruned by the cost-complexity procedure (Breiman et al., 1984), using ten-fold cross-validation. Bagged tree ensembles contain 50 un-pruned trees. In all cases, we use the closed-form version of DPC (Section 2.2) and λ is tuned as suggested in Section 2.2.1.

To analyze the results, we carried out paired t-tests with the correction proposed in (Nadeau & Bengio, 2003) and a significance level of 0.05. Table 2 summarizes the results of the hypothesis tests in terms of Win/Draw/Loss reports separately for classification and regression problems. On Figure 2 we have also marked with a cross (×) those cases where the DPC version is significantly more accurate than the corresponding base learner.

3.2. Accuracy results

We first observe from Table 2 that the DPC method when combined with ST, PST, and TB never leads to a significant increase of errors, but quite often leads to a significant improvement. It appears also that the improvement is more often significant on regression problems than on classification problems. For example, on classification problems ST^d wins 5 times (and draws 5 times) with respect to ST, while on regression problems it wins 8 times (and draws 2 times). We also notice, specially on regression problems, that the improvement by DPC is stronger on ST than on PST, and stronger on PST than on TB.

Table 2. Win/Draw/Loss reports of column-method w.r.t. row-method (top classification, bottom regression)

	ST	ST^d	PST	PST^d	TB	TB^d
ST	-	5/5/0	2/8/0	5/5/0	7/3/0	7/3/0
ST^d	0/5/5	-	1/4/5	0/10/0	6/4/0	6/4/0
PST	0/8/2	5/4/1	-	5/5/0	7/3/0	8/2/0
PST^d	0/5/5	0/10/0	0/5/5	-	6/4/0	7/3/0
TB	0/3/7	0/4/6	0/3/7	0/4/6	-	3/7/0
TB^d	0/3/7	0/4/6	0/2/8	0/3/7	0/7/3	-
ST	-	8/2/0	5/4/1	8/2/0	9/1/0	9/1/0
ST^d	0/2/8	-	0/3/7	2/8/0	5/5/0	6/4/0
PST	1/4/5	7/3/0	-	7/3/0	8/2/0	8/2/0
PST^d	0/2/8	0/8/2	0/3/7	-	3/7/0	5/5/0
TB	0/1/9	0/5/5	0/2/8	0/7/3	-	6/4/0
TB^d	0/1/9	0/4/6	0/2/8	0/5/5	0/4/6	-

Comparing ST^d with PST^d, we observe that on classification problems they show no significant difference, while on regression problems they draw on 8 datasets (and PST^d wins on 2). This means, that from the accuracy viewpoint, pruning is rather redundant with DPC. On the contrary, bagging and DPC appear as complementary. On regression problems the TB^d version wins 5 times over PST^d and 6 times over TB.

Overall, the best method in terms of accuracy is TB^d. On the 20 datasets, it never loses with respect to any of the other methods (with or without DPC). The second best is TB, but with respect to this latter the PST^d version does a decent job, since they draw on 11 problems out of 20 while PST only draws 5 times.

Considering the standard deviations of error estimates (see Figure 2), we observe that they are rather problem dependent[4], but otherwise of similar magnitude for the different methods. Most of the regression problems have larger test sets, and therefore smaller standard deviations, which partially explains why the sometimes small improvements of DPC are nevertheless declared significant by the *t*-test.

3.3. Smoothing level λ

Figure 3 shows how the (tuned) values of the smoothing level λ vary according to different conditions.

Figure 3(a) correlates the (average) tuned value of λ with the relative improvement of the average error,

[4]They are larger for Vehicle, Vowel, Segment, and Housing, which have smaller test samples (see Table 1).

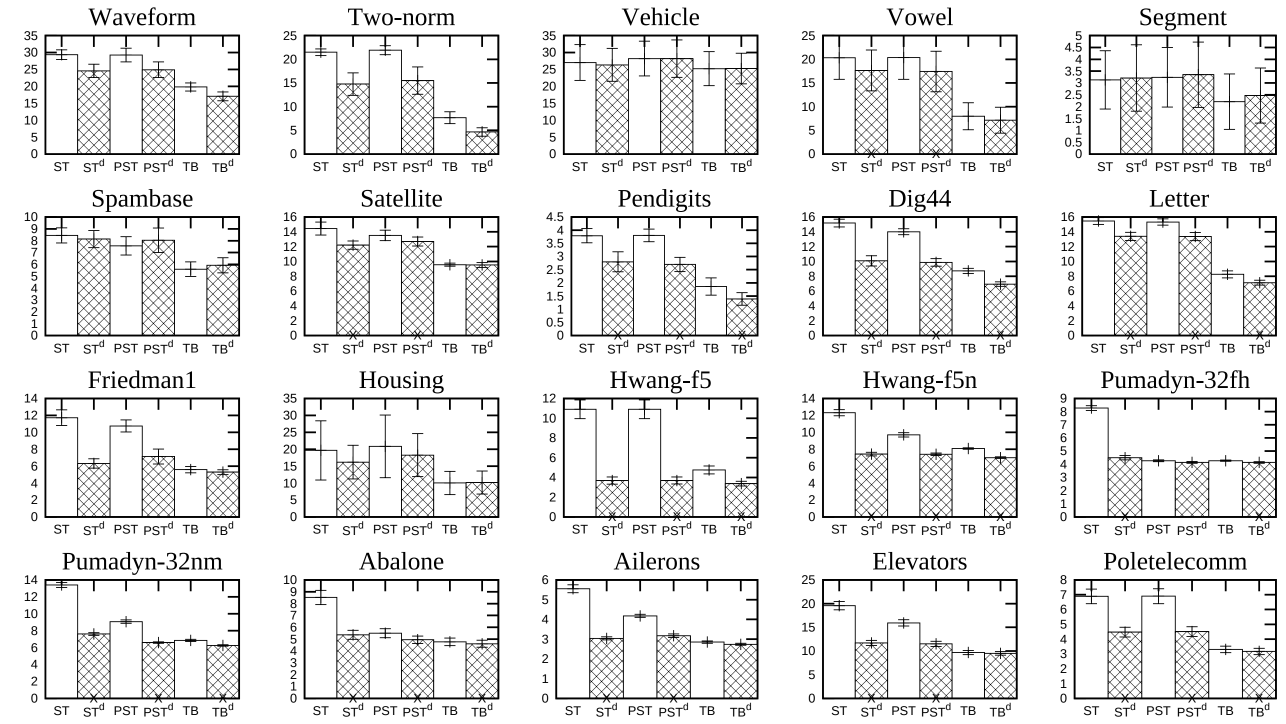

Figure 2. Errors of single (full and pruned) trees and bagging with and without DPC (top classification, bottom regression)

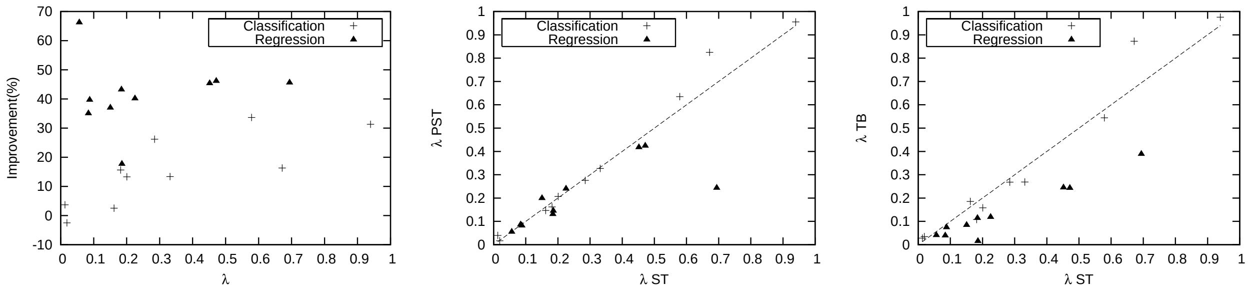

Figure 3. (a) λ for ST^d versus error improvement. (b) λ for ST^d versus λ for PST^d. (c) λ for ST^d versus λ for TB^d

for classification and regression datasets. On classification problems, the higher values of λ correspond to the stronger improvements. On regression problems, the correlation is however marginal, and while the improvement is stronger than in classification, it appears to happen often for λ values rather close to zero.

Figure 3(b) shows that for 19 problems out of 20, the tuned λ-value is not strongly affected by tree pruning. The outlier in this respect, corresponds to the Pumadyn-32fh problem (located at (0.69, 0.24)), where the pruned version needs significantly less smoothing than the un-pruned one. This problem is very noisy, and thus pruning alone already strongly reduced the variance (it reduces the average complexity of the trees from 1509 to 18 terminal nodes).

Finally, Figure 3(c) mainly shows that tree bagging tends to decrease λ on regression problems, while in classification it does not seem to affect λ.

Overall, these figures show that the optimal λ-value is problem dependent and thus justify the use of a cross-validation procedure to tune it automatically.

3.4. Bias/variance analysis of DPC

Figure 4 shows how bias, variance, and average square-error depend on the value of the smoothing parameter λ, both for ST^d and TB^d versions. These curves have been obtained by splitting the Friedman1 dataset into

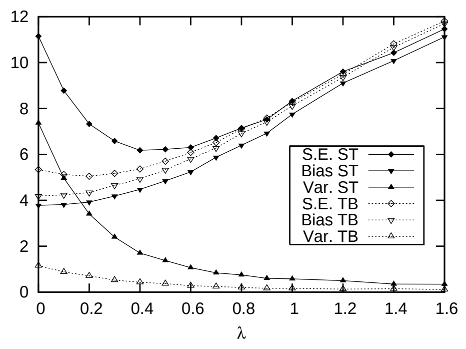

Figure 4. Bias/variance tradeoff with λ (Friedman1).

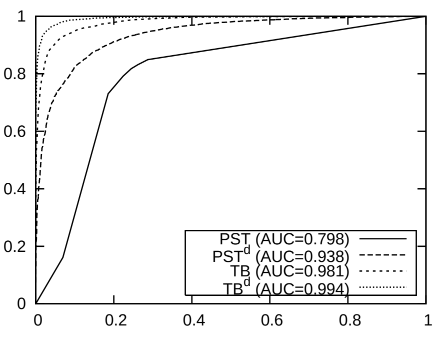

Figure 5. Effect of DPC on ROC curves (Two-Norm).

two parts: a pool of 8000 cases and a test sample of 2000 cases. 100 models are built from 100 learning samples of size 300 randomly drawn from the pool. Then, bias, variances, and mean errors are estimated on the test sample by means of these 100 models for a range of λ values between 0.0 and 1.6. Notice that the bias values displayed concern actually the square value of bias incremented by the residual error (which is independent of the algorithm and value of λ).

The left most point on each curve of Figure 4 corresponds to $\lambda = 0$, i.e. the base algorithm. When λ increases, the variance of single trees decreases quickly, and that of bagging in a less pronounced way. On the other hand, the bias of two algorithms remains very close over the full range of values (the slight advantage of single trees is due to the fact that bagging uses bootstrap samples, which leads to smaller trees).

For ST the bias variance tradeoff yields a reduction of average error of about 50% for $\lambda = 0.4$. On tree bagging the optimum is reached for $\lambda = 0.2$, and the improvement is slim. Comparing, TB and ST^d in terms of variance reduction, we see that within the range of nearly optimal values of $\lambda \in [0.4, 0.5]$ for the latter, its variance reduction is smaller than that of bagging.

The conclusions drawn from this example remain valid for other regression problems and also for the analysis of bias and variance of average square error estimates of the class-probabilities predicted by tree-based classification models. However, in the latter case the optimal smoothing is adjusted to minimize error rates and the resulting values are typically larger than those which yield the best tradeoff from the regression point of view. This phenomenon is related to the fact that class probabilities can be strongly biased without affecting error rates, provided that their average value remains on the right side of the decision threshold (Friedman, 1997). This explains our observation that optimal λ-values are larger in classification problems.

3.5. Effect of DPC on ROC curves

In classification DPC yields smoother probability estimates than single or bagged trees and it should therefore also improve ROC curves. To illustrate this feature, Figure 5 shows ROC curves obtained for class 1 of the Two-Norm problem, for ST, TB and their DPC variants ST^d and TB^d, together with the value of the AUC (area under the ROC curve) values. The curves are obtained by plotting (for a decision threshold varying from 0 to 1) how the proportion of false-alarms (cases of class 2 erroneously classified as class 1) and the proportion of correct detections (cases of class 1 classified as class 1) vary. We observe that the DPC method indeed strongly improves the ROC curves of both methods as well as their AUC values.

4. Comparison of DPC and Bagging

Variance reduction. The propagation of randomized attribute vectors through a fixed tree is equivalent to propagating a fixed attribute vector through a tree with randomized cut-points. Hence DPC of tree-based models is equivalent to cut-point randomization in PC. On the other hand, tree bagging randomizes trees through bootstrap re-sampling, which affects cut-points, but also tree structure and leaf labels. Thus, bagging takes into account all variance sources in tree induction while DPC only takes into account the cut-point variance. This certainly explains why bagging is able to reduce more strongly the variance of tree-based models with only a small increase of bias.

Computational efficiency at the testing stage. The computation of (7) (which is proportional to the tree complexity) is intrinsically more complex than the computation of one prediction with a classical tree (which is proportional to the tree depth). Since tree complexity grows typically much faster with the learning sample size than tree depth, for very large datasets, the closed-form approximation will eventually become much slower than the finite sample based estimate of Eqn (2) with a reasonable number of terms T. Furthermore, in this latter approach the number T can be adjusted to obtain prediction times compatible with available resources, possibly at the price of a loss of accuracy. Quite evidently, it is preferable to use the PST^d version from this point of view. To fix ideas, in our datasets, the slow down of testing by PST^d (in our implementations) ranged between a factor 4 (on very small datasets, such as Waveform) to a factor 90 (for the letter dataset). In comparison, the testing times of ensembles of 50 bagged trees are about 50 times slower than PST. Thus, on large datasets, the main computational advantage of either DPC variants with respect to bagging is their reduced storage requirement (also, roughly a factor 50).

Computational efficiency at the learning stage. The tuning of λ by cross-validation requires to build several trees and hence mitigates the computational advantage of DPC with respect to other ensemble methods. Note that this comment also applies to pruning per se, and so, when DPC is combined with pruning by cross-validation, the computational overhead of the tuning of λ is relatively reduced. Also, in the context of large training samples where this overhead could become penalizing we suggest to use hold-out estimates both for pruning and tuning of λ. In this case, the computational overhead of DPC with respect to standard trees becomes negligible. To fix ideas, we indicate that with our implementations the increase of learning times of PST^d vs PST ranged between a factor of 1.1 (on small datasets, as Waveform) to 7 (on large datasets, as Letter). Tree bagging was about 5 times slower than PST on all datasets.

Interpretability. In terms of interpretability it is also preferable to apply DPC on top of already pruned trees, since they are simpler and hence easier to interpret. In this combination, we believe that DPC preserves and even improves the interpretability of pruned trees, since it provides smoother output values, and in classification, a better indication of the proximity of a test case to the classification boundary.

5. Related Work on Soft Tree Models

The soft tree model is not new. In the machine learning community, (Carter & Catlett, 1987) first have proposed to propagate examples which are close to the discretization thresholds to both successors of a decision tree node and weight the predictions of the corresponding subtrees according to the distance of these examples to the discretization threshold. Following them, (Quinlan, 1986) provides in C4.5 a rudimentary way to soften discretization thresholds. (Friedman, 1996) proposes a method which recursively divides the learning sample into overlapping subsets and uses voting schemes to aggregate competing predictions. Markov tree models (Jordan, 1994) justify soft decisions in extended tree models by a probabilistic framework. In a fuzzy decision tree, the weight used to propagate an instance to the left and right successors of a test node is interpreted as a fuzzy set membership degree (see (Olaru & Wehenkel, 2003) and the references therein).

We think that our bias/variance analysis extends to such soft tree methods, and believe that they mainly improve the accuracy of classical trees by reducing their variance. The relationship between PC and DPC also links soft trees with tree-based ensemble methods.

In (Ling & Yan, 2003), a method based on multiple propagation is proposed to improve the AUC of decision trees. In this method, a case is propagated along all branches emerging from an internal node, with a smaller weight s for the branches which are not satisfied by the test. Weights are multiplied along the paths towards leaves and predictions are averaged. In this method, the weight s is independent on the deviation of attribute values from discretization thresholds. Therefore, this method does not provide smooth input-output models, contrary to DPC. Its computational overhead, on the other hand, is at least as important as that of DPC.

6. Conclusion and Extensions

This paper has developed a new wrapper technique which consists in smoothing model output at the prediction stage by randomly perturbing attribute values and aggregating the corresponding randomized predictions. A closed-form implementation of this idea has been developed, in combination with tree based models, and evaluated on classification and regression problems. When combined with single pruned trees, this method gives substantial improvements of accuracy on many problems, while preserving interpretability. Also, the computational overhead is acceptable provided datasets are not too large.

From a more general point of view, this algorithm also suggests that with one model, it is only possible to reproduce part of the improvement of accuracy of ensemble methods. However, this technique has several advantages over ensemble methods. The algorithm is generic and very simple to implement. It only depends on one parameter which value can be tuned either by cross-validation or using an independent test sample. Furthermore, since (in this latter case) it does not require to rerun the learning algorithm, it may be an interesting solution to improve a model for which we do not have access to the learning algorithm and/or learning data.

There remain several possible extensions of our work. First, DPC could be combined with other PC algorithms or with tree boosting, and more generally with any other non-linear supervised learning algorithm. In particular, some preliminary studies not reported here show that this method can improve the accuracy when applied on the top of multi-layer perceptrons. Another useful and at the same time rather obvious extension would consist in modifying the algorithm so as to cope also with symbolic attribute values. Finally, although the perturbation scheme proposed here has the advantage of being generic and independent of the particular interpretation of the attributes, better improvement could possibly be gained by using knowledge about the application problem to imagine different perturbation schemes. Along this idea, let us cite (Dahmen et al., 2001) where a handwritten digit recognition system is improved by propagating into one model several shifted versions of a digit image and aggregating the resulting set of predictions. We interpret this algorithm as an ad hoc version of DPC for image classification.

Acknowledgments

P. Geurts is a post-doctoral researcher at the Belgian National Fund of Scientific Research (FNRS).

References

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/`.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*, 123–140.

Breiman, L. (1998). Arcing classifiers. *Annals of statistics*, *26*, 801–849.

Breiman, L. (2000). Randomizing outputs to increase prediction accuracy. *Machine Learning*, *40*, 229–242.

Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5–32.

Breiman, L., Friedman, J., Olsen, R., & Stone, C. (1984). *Classification and regression trees*. Wadsworth International (California).

Carter, C., & Catlett, J. (1987). Assessing credit card applications using machine learning. *IEEE Expert*, *Fall*, 71–79.

Dahmen, J., Keysers, D., , & Ney, H. (2001). Combined classification of handwritten digits using the "virtual test sample method". *Proc. of the Second International Workshop on Multiple Classifier Systems, Cambrige, UK* (pp. 109–118).

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, *40*, 139–157.

Friedman, J. (1991). Multivariate adaptive regression splines. *Annals of Statistics*, *19*.

Friedman, J. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, *1*, 55–77.

Friedman, J. H. (1996). *Local learning based on recursive covering* (Technical Report). Department of Statistics, Stanford University.

Geurts, P. (2001). Dual perturb and combine algorithm. *Proc. of the Eighth International Workshop on Artificial Intelligence and Statistics* (pp. 196–201). Key-West, Florida.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 832–844.

Jordan, M. I. (1994). A statistical approach to decision tree modeling. *Proc. of the Seventh Annual ACM Conference on Computational Learning Theory. New York.*. ACM Press.

Ling, C., & Yan, R. (2003). Decision trees with better ranking. *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)* (pp. 480–487). Washington DC.

Nadeau, C., & Bengio, Y. (2003). Inference for the generalization error. *Machine Learning*, *52*, 239–281.

Olaru, C., & Wehenkel, L. (2003). A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, *138*, 221–254.

Quinlan, J. (1986). *C4.5: Programs for machine learning*. Morgan Kaufmann (San Mateo).

Torgo, L. (1999). *Inductive learning of tree-based regression models*. Doctoral dissertation, University of Porto.

Wehenkel, L. (1998). *Automatic learning techniques in power systems*. Boston: Kluwer Academic.