




Hierarchical Gradient Smoothing for Probability Estimation Trees ^{*}

He Zhang , François Petitjean , and Wray Buntine 

Faculty of Information Technology, Monash University, Australia
{he.zhang, francois.petitjean, wray.buntine}@monash.edu

Abstract. Decision trees are still seeing use in online, non-stationary and embedded contexts, as well as for interpretability. For applications like ranking and cost-sensitive classification, probability estimation trees (PETs) are used. These are built using smoothing or calibration techniques. Older smoothing techniques used counts local to a leaf node, but a few more recent techniques consider the broader context of a node when doing estimation. We apply a recent advanced smoothing method called Hierarchical Dirichlet Process (HDP) to PETs, and then propose a novel hierarchical smoothing approach called Hierarchical Gradient Smoothing (HGS) as an alternative. HGS smooths leaf nodes up to all the ancestors, instead of recursively smoothing to the parent used by HDP. HGS is made faster by efficiently optimizing the Leave-One-Out Cross-Validation (LOOCV) loss measure using gradient descent, instead of sampling used in HDP. An extensive set of experiments are conducted on 143 datasets showing that our HGS estimates are not only more accurate but also do so within a fraction of HDP time. Besides, HGS makes a single tree almost as good as a Random Forest with 10 trees. For applications that require more interpretability and efficiency, a single decision tree plus HGS is more preferred.

Keywords: Probability Estimation Trees · Class Probability Estimation · Hierarchical Probability Smoothing · Hierarchical Dirichlet Process.

1 Introduction

Many critical classification tasks require accurate class probability estimates rather than class labels because probabilities can show people whether the prediction is reliable or not. For instance, the weather forecast not only predicts rain but also tells people how likely it is. In cost-sensitive learning, the misclassification cost could be significantly reduced if more accurate class probability estimates could be obtained [4].

More recently, with the advent of many more learning tasks, such as online learning, or learning where the inference system have low computational

^{*} This research was partially supported by the Australian Research Council's Discovery Projects funding schemes (projects DP190100017 and DE170100037).

resources, a single decision tree is seeing a resurgence. Extremely fast decision trees are one of the top performers for high-data-throughput contexts [11]. Random forests and gradient boosted trees have relatively high computational demands in inference, and thus may not be suitable for wearable or embedded IOT (Internet of Things) applications. So, the problem of making a single tree perform well in inference arises, and one can ask does a single decision tree beat a random forest with 10 trees. Moreover, trees also serve as one of the few global models considered to be interpretable, an increasingly important requirement in applications [12]. Thus, quality single decision tree built efficiently have many uses.

Probability Estimation Tree (PET) is a generalization of a single decision tree by taking the observed frequencies at a leaf node as the class probability estimates for any test examples that fall into this leaf. However, this method may lead to unreliable estimates when the number of training examples associated in a leaf is small [21]. Simple probability smoothing techniques, such as Laplace smoothing and M-estimation, have long been used to improve PETs' class probability estimates by making the estimates at leaves less extreme. However, they ignore the broader context of any leaf node, especially crucial in cases where the datasets are imbalanced.

Hierarchical smoothing has gained attention in the community in recent years. It assumes that the class probability of the leaf node depends on the probabilities of its parents in some hierarchy. To our knowledge, M-branch smoothing [5] is the first and only one hierarchical method for PETs. It smooths the leaf node to its direct parent using M-estimation, with the parent also been smoothed recursively until the root node reached. The results demonstrate that M-branch performs better than M-estimation. Hierarchical Dirichlet Process (HDP) [13] can also be used to smooth the probability at the leaves with its parent, partially mimicking what is done in M-branch, but it uses fully Bayesian inference. A decision tree can be turned into a HDP model tree with each node in the tree associated with a Dirichlet Process (DP). Similar HDP smoothing methods allow Bayesian network classifiers [13] and language models [17] to get state-of-the-art probability estimates but has not been applied to decision trees.

[14] believe that a thorough study of what are the best smoothing methods for PETs would be a successful contribution to machine learning research, which is also the main aim of this research. We first show that HDP can help PETs get better estimates compared with M-estimation. However, HDP is computationally intensive. A novel, more efficient hierarchical smoothing method called Hierarchical Gradient Smoothing (HGS) is proposed. Unlike HDP and M-branch, HGS smooths the leaf node to all the ancestor nodes at once, where each ancestor has a weight parameter to control the smoothness. We propose Leave-One-Out Cross-Validation (LOOCV) cost to PETs and conduct a gradient descent algorithm [16] on the cost to automatically obtain the parameters. One time smoothing and gradient descent make HGS more efficient than recursive smoothing and sampling. A single PET With HGS makes more than 90/143

UCI datasets obtain the best probability estimates. Besides, HGS makes single tree superior to Random Forest with 7 trees and almost as good with 10 trees.

The remainder of this paper is organized as follows. The related works are reviewed in Section 2. M-branch and HDP are also introduced in this section. The HGS model is developed in Section 3. An extensive experiment results are reported in Section 4.

2 Related work

There are some empirical studies [10,23] of improving the class probability estimation of PETs, covering different tree learning algorithms, probability smoothing techniques and tree ensembles, among which we focus more on probability smoothing techniques.

C4.5, as a traditional decision tree learning algorithm[15], cannot produce accurate class probability estimates because it aims at building small trees with accurate class label predictions rather than accurate class probability estimates. Tree pruning technique is used in C4.5 to achieve this goal by removing the nodes and branches at the bottom of the tree that fitted to noise data. The pruned tree is more accurate on classification but less accurate on class probability estimation [22]. C4.4 is a variant of C4.5 with better class probability estimates by turning off pruning and applying Laplace smoothing method to leaf nodes [14].

Laplace and M-estimation as the two most simple smoothing methods have long been used on PETs [14,22] and cost-sensitive learning [20]. Although more sophisticated smoothing methods such as Kneser-Ney [8] and Modified Kneser-Ney [2] have been used in language modelling for a long time, M-branch was the first hierarchical smoothing method for decision trees proposed in 2003 [6]. A recent smoothing method called Hierarchical Dirichlet Process (HDP) has had great success on language modelling [17] and Bayesian Network Classifiers [13], whereas it has not been used on decision trees. The following part introduces these methods in detail.

M-estimation smoothing M-estimation is more recommended by [22] for class-imbalanced data, which is defined as

$$\hat{\theta}_k^{M-esti} = \frac{n_k + M \times b}{n. + M} \quad (1)$$

where the base rate b is the expected probability without any additional knowledge, and it is usually considered uniform, i.e. $b = \frac{1}{K}$. M is a parameter that controls how much scores are shifted towards the base rate. When $m = K$ and $b = \frac{1}{K}$, it becomes Laplace smoothing.

M-branch Smoothing M-branch is the first hierarchical smoothing method for PETs [5]. It considers each node in the tree is a subsample of the upper parent. This means that the sample used to obtain the probability estimates in a leaf is the result of many sampling steps, as many as the depth of the leaf. Then it is natural to consider all the history of samples when trying to obtain the probability estimates of a leaf.

Let $\langle v_1, v_2, \dots, v_{l-1}, v_l \rangle$ represents all the nodes on the branch that contains the leaf node v_l , where v_{l-1} is the parent of v_l and v_1 is the root. The class probability estimate for class k at node v_l is smoothed by M-estimation in the following way

$$\hat{\theta}_{l,k}^{Mbranch} = \frac{n_{l,k} + m_l \times \hat{\theta}_{l-1,k}^{Mbranch}}{n_{l,\cdot} + m_l} \quad (2)$$

where $n_{l,k}$ denote the observed count of class k and $n_{l,\cdot}$ is the total. The base rate b for M-estimation is the parent estimate $\hat{\theta}_{l-1,k}^{Mbranch}$, which also needs to be smoothed to the parent node at a higher level v_{l-2} . Repeat these steps recursively until the root node v_1 reached. The root node is smoothed to a uniform probability $\hat{\theta}_{0,k} = \frac{1}{K}$. The m parameter for each node has been defined as a function of the node height in the tree. Please refer to [5] for more detail.

HDP Smoothing Unlike M-branch, HDP assumes that only leaf nodes have data and all the parent nodes are empty, instead of inheriting data from their children during the inference process. Each leaf passes some subset of its data to its parent, selected during the inference process. The subset passes higher and higher, thinning recursively until the root is reached. Suppose $t_{u,k}$ is the subset of data $n_{u,k}$ that node u passes up to its parent ϕ , the data for node ϕ is collected from all the children so that $n_{\phi,k} = \sum_{u \in \phi} t_{u,k}$ where $u \in \phi$ means u is the child of ϕ . The concentration c_ϕ controls how much data passes up to node ϕ , i.e. $t_{u,k}$. If we expect $\hat{\theta}_{u,k}$ to be very similar to $\hat{\theta}_{\phi,k}$, then choose a bigger c_ϕ that makes most of the data pass up, and the parent probability contribute more to the estimate. If c_ϕ is small, the parent probability contributes less to the estimate. The smoothing formula for node u and class k is defined as follows

$$\hat{\theta}_{u,k}^{HDP} = \frac{n_{u,k} + c_\phi \times \hat{\theta}_{\phi,k}^{HDP}}{n_{u,\cdot} + c_\phi} \quad (3)$$

Here $\hat{\theta}_{\phi,k}^{HDP}$ is the parent estimate which also needs to be smoothed. The class probabilities are calculated from the root to the leaves. Thus, when reaches the leaves, the probability estimates are already properly smoothed. Equ. 3 can also be explained by the Hierarchical Chinese Restaurant Process (CRP) [19]. Please refer to [13] for more detail of HDP smoothing on Bayesian Network Classifiers.

Gibbs sampling is used in HDP to sample the concentration parameters. *iteration* is the number of iterations of Gibbs sampling. *tying* is used to tie some nodes together to share a single concentration in order to reduce the number of parameters. There are four types of tying. *SINGLE* means tying all the nodes together to share a single concentration parameter. *LEVEL* means the nodes on the same depth are tied together. *PARENT* means tying the sibling nodes under one parent. *NONE* means no tying.

3 HGS Algorithm

In this section, we propose a novel efficient hierarchical probability smoothing method for PETs called Hierarchical Gradient Smoothing (HGS). Like all other

hierarchical smoothing methods, HGS considers that the class probability estimate of a leaf node is related to the probability estimates of all parent nodes on the branch that contains the leaf.

3.1 The Hierarchical Computation

HGS is different from HDP and M-branch. Figure 1 can more intuitively express the differences between them. It can be seen from this figure that HDP and M-branch both smooth the leaf node to an upper parent node, then the parent node also needs to be smoothed to a higher node until the root node is reached. Each node has a concentration parameter to control the smoothness, which is c for HDP and m for M-branch. However, unlike HDP and M-branch smoothing, HGS smooths the class probability estimate on a leaf node to all ancestor nodes on the branch at one time, instead of only to the nearest parent node recursively. Each parent node has a weight parameter α to control the degree to which the probability estimates are backed off to the parent. The one-time smoothing makes HGS faster than HDP and M-branch and also allows global optimization of hyper-parameters.

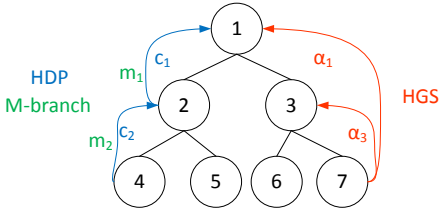


Fig. 1: The difference between HGS, HDP, and M-branch. HGS is represented by red, HDP and M-branch by blue and green, respectively.

The probability smoothing formula for leaf l and class k using HGS is as follows,

$$\hat{\theta}_{l,k}^{HGS} = \frac{n_{l,k} + \sum_{p \in \text{anc}(l)} \alpha_p \hat{\theta}_{p,k}}{n_{l,\cdot} + \sum_{p \in \text{anc}(l)} \alpha_p}, \tag{4}$$

where $\text{anc}(l)$ represents all the ancestor nodes on the branch that containing the leaf node l . Each ancestor node $p \in \text{anc}(l)$ has a weight parameter α_p controls the degree of smoothness. The probability estimate of p is calculated by the Maximum Likelihood Estimation (MLE), which is defined as $\hat{\theta}_{p,k} = \frac{n_{p,k}}{n_{p,\cdot}}$. The term $\sum_{p \in \text{anc}(l)} \alpha_p \hat{\theta}_{p,k}$ is the weighted combined probability of all the ancestors. The term $\sum_{p \in \text{anc}(l)} \alpha_p$ is the sum of the weights.

3.2 Working with LOOCV

Let α denote the vector that contains all the weight parameter α s with the size being the number of internal nodes in the tree. How can one set the α properly, for instance, how should it be optimized?

Before going into the details of how to set α , it is worth briefly introducing LOOCV and incremental LOOCV. LOOCV is a special case of k fold cross-validation, where k equals to the number of training examples N . In each fold of the cross-validation, an example is treated as a test example while the others are the training examples. LOOCV could be sped up using incremental LOOCV [9,7]. The idea is instead of training a model during each fold of the cross-validation, first, train a model on the full dataset, then delete the one example that is left out, test on that example, then insert it into the model again. This delete-test-insert phase is repeated for each of the N folds. Incremental LOOCV can be conducted on any algorithm that supports incremental learning, allowing for dynamically adding or removing examples from the model. Decision tree learning is such an algorithm. Note incremental LOOCV means the model structure remains unchanged.

If one looks at a cross-validation in LOOCV, a test example at leaf l with true class k should be left out from the tree, which means the data count of both l and $\text{anc}(l)$ should be reduced by 1. The total count becomes $n_{l,\cdot} - 1$. This Leave-One-Out (LOO) probability estimate for this test with class k becomes

$$\theta_{l,k}^{LOO} = \frac{n_{l,k} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p \theta_{p,k}^{LOO}}{n_{l,\cdot} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p} \quad (5)$$

Here $\theta_{p,k}^{LOO} = \frac{n_{p,k}-1}{n_{p,\cdot}-1}$. $n_{l,k} \geq 1$ must be satisfied so that there is at least one example to be moved out. For other classes $c \in \mathcal{K}, c \neq k$, the probability estimate is formed without subtracting one in the numerator.

If one performs an incremental LOOCV on the tree, the examples in every leaf $l \in \mathcal{L}$ with every class $k \in \mathcal{K}$ need to be left out once. The LOOCV measure using log loss of all the examples in the tree becomes

$$LOOCV(\alpha) = \frac{1}{N} \sum_{l \in \mathcal{L}} \sum_{k \in \mathcal{K}} n_{l,k} \cdot \log \left(\frac{1}{\theta_{l,k}^{LOO}} \right) \quad (6)$$

and note we have also tested a squared error loss $(1 - \theta_{l,k}^{LOO})^2$ yielding similar results. For more information about loss functions please refer to [18]

Now a Gradient Descent algorithm [16] can be performed on the $LOOCV(\alpha)$ cost to optimize the parameters α . The gradient of each α_p is

$$\frac{\partial}{\partial \alpha_p} LOOCV(\alpha) = \frac{1}{N \ln 2} \sum_{l \in \text{des}(p)} \sum_{k \in \mathcal{K}} \beta_{l,k} \quad (7)$$

where $\beta_{l,k}$ is referred to

$$\beta_{l,k} = \frac{n_{l,k} \cdot (\theta_{l,k}^{LOO} - \theta_{p,k}^{LOO})}{\left(n_{l,\cdot} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p \right) \cdot \theta_{l,k}^{LOO}} \quad (8)$$

Here $\text{des}(p)$ represents the descendent leaves under p .

3.3 Algorithm Description

The HGS algorithm $HGS(\mathcal{T}, b, v)$ takes a decision tree T , a learning rate b and a precision parameter ϵ as inputs, and a HGS smoothed tree as output. It has three steps in total. First, initialize α to be the vector of parameters with the length to be the number of internal nodes and all the values to be 1. Second, conduct a standard gradient descent algorithm to get the optimized parameters α . Last, traverse the tree top-down in level-order to calculate the HGS smoothed probability estimates $\hat{\theta}_{i,k}^{HGS}$ for all the leaves. The top-down traverse method used here is the same as the first tree top-down traverse method in Algorithm 2 (line 1–9), except that the probability estimates are calculated using Equ. 4 in line 8.

The second step in the HGS algorithm uses a standard gradient descent algorithm to optimize the parameters, as shown in Algorithm 1. Gradient descent needs many iterations to reduce the cost until the cost difference between two iterations is less than a given ϵ . Algorithm 2 is called in every iteration to calculate the cost and gradients by going through the tree twice. First, traverse the tree top-down to calculate the LOO estimate $\theta_{p,k}^{LOO}$ using Equ. 5 and the cost $LOOCV(\alpha)$ using Equ. 6 (line 1–9). $\alpha_p^* = \sum_i \alpha_i$ and $\theta_{p,k}^* = \sum_i \alpha_i \theta_{i,k}^{LOO}$. Second, traverse the tree bottom-up to calculate the gradients for each internal node level by level (line 10–16). Last, return the cost.

The complexity of HGS smoothing on a PET is $O(I \cdot S \cdot K)$, where S is the total number of nodes and K is the number of classes. We call I the number of iterations for gradient descent. In practice, we use the standard stopping criterion corresponding to an improvement of less than ϵ . Each iteration of gradient descent has a complexity that is linear to the size of the tree S (total number of nodes).

Algorithm 1: *gradientDescent*($\mathcal{T}, \alpha, b, \epsilon$)

Input : a decision tree \mathcal{T} , an initialized vector α , a learning rate b , a learning rate ϵ

Output: an optimized vector α

```

1 costDiff = Double.max;
2 while costDiff >  $\epsilon$  do
3   cost  $\leftarrow$  calculateGradientsAndCost( $\mathcal{T}, \alpha$ );
4   for each node  $p \in \mathcal{T}$  and  $p \notin \mathcal{L}$  do
5     // internal nodes
6      $\alpha_p := \alpha_p - b \frac{\partial}{\partial \alpha_p} LOOCV(\alpha)$ ;
7   cost'  $\leftarrow$  calculateGradientsAndCost( $\mathcal{T}, \alpha$ );
8   costDiff  $\leftarrow$  cost - cost';
9 return  $\alpha$ 

```

4 Experiments

The aim of this section is to show the performance of HGS smoothing compared with other existing smoothing methods for C4.5 trees. Section 4.1 gives

Algorithm 2: *calculateGradientsAndCost*(\mathcal{T}, α)

Input : a tree \mathcal{T} with depth d and leaves \mathcal{L}
Input : a vector α
Output: *cost*

```

1 /* Traverse 1: Calculate cost top-down. */
2 cost  $\leftarrow$  0;
3 for  $h \leftarrow 0$  to  $d$  do
4   for each node  $p$  in level  $h$  and each class  $k \in \mathcal{K}$  do
5     if  $p \notin \mathcal{L}$  then
6       calculate  $\theta_{p,k}^{LOO}$ ,  $\alpha_p \theta_{p,k}^{LOO}$ ,  $\alpha_p^*$  and  $\theta_{p,k}^*$ ;
7     else
8       calculate  $\theta_{l,k}^{LOO}$  using Equ. 5;
9       cost  $\leftarrow$  cost +  $n_{l,k} \cdot \log \frac{1}{\theta_{l,k}^{LOO}}$ ;
10 /* Traverse 2: update gradients bottom-up. */
11 for  $h \leftarrow d$  to 0 do
12   for each node  $p$  in level  $h$  and each class  $k \in \mathcal{K}$  do
13     if  $p \notin \mathcal{L}$  then
14       Calculate  $\frac{\partial}{\partial \alpha_p} LOOCV(\alpha)$  using Equ. 7;
15     else
16       Calculate  $\beta_{l,k}$  using Equ. 8;
17 return cost  $\leftarrow \frac{1}{N \cdot I n 2} \cdot$  cost
```

the general experimental settings. The remaining sections then detail individual experiments.

4.1 Experiment Design and Setting

Design An extensive set of experiments are conducted on 143 standard datasets from the UCI archive [3], where 20 have more than 10,000 instances, 52 have between 1,000 and 10,000, and 71 have less than 1,000 instances. A missing value is treated as a separate attribute value. The datasets and the table that summarizes the characteristics of each dataset, including the number of instances, attributes and classes, can be found and downloaded from [Github](#)¹.

Evaluation Measure The results are assessed by RMSE and 0-1 Loss. RMSE is the most important measure because it measures how well-calibrated the probability estimates are, and these better support tasks with unequal costs or imbalanced classes. 0-1 Loss refers to classification accuracy. For both RMSE and 0-1 Loss, the smaller, the better. Win-Draw-Loss (WDL) is reported when comparing two methods. A two-tail binomial sign test is used to determine the significance of the results. A difference is considered to be significant if $p \leq 0.05$.

Software To ensure reproducibility of our work and allow other researchers to build on our research easily, we have made our source code for HGS smoothing on PETs available on [Github](#)¹.

Compared methods Different smoothing methods are compared for the C4.4 decision tree (C4.5 without pruning), including Laplace smoothing, M-estimation,

¹ <https://github.com/icesky0125/DecisionTreeSmoothing>

M-branch, HDP and HGS. The parameters of HDP are set to be $iteration = 1,000$ and $tying = SINGLE$, which are tested by us. All the methods are evaluated using 10-fold cross-validation.

4.2 HGS Parameter Tuning

HGS is basically a parameterless algorithm. The only two parameters are the learning rate b and the minimum cost difference threshold ϵ between two iterations needed in the gradient descent algorithm [16]. In this experiment, we tried different values $b = 0.01, 0.001$ and $\epsilon = 0.001, 0.0001$ and found that their results were all the same with only slight differences in training time. Based on these results, in the following experiment we choose standard values of $b = 0.01$ and $\epsilon = 0.0001$.

4.3 HGS vs. Existing Methods

This experiment evaluates the advantages of HGS compared with single-layer smoothing methods, including MLE, Laplace correction and M-estimation, and hierarchical smoothing methods, including M-branch and HDP. Table 1 and Table 2 are the table of WDL and the averaged value respectively. The error bars of all these models are 0.012 on RMSE and 0.015 on 0-1 Loss, respectively. It can be seen from these tables that HGS is significantly better than all of the existing methods on RMSE and better on 0/1 Loss.

The training time for HGS, as a hierarchical smoothing method, is similar to single-level methods, which is 1.1 seconds. While compared with the existing hierarchical methods, HGS is approximately 5 times faster than HDP and 9 times faster than M-branch on average. This indicates that HGS makes PET get both very accurate probability estimates and classification results efficiently.

Table 1: Win-Draw-Loss results (The boldface values are significant).

<i>Method</i>	<i>RMSE</i>	<i>0-1 Loss</i>
HGS vs. MLE	108-2-33	69-22-52
HGS vs. Laplace	111-4-28	68-22-53
HGS vs. M-esti	98-4-41	66-23-54
HGS vs. M-branch	96-3-44	59-32-52
HGS vs. HDP	92-1-50	64-21-58

Table 2: Averaged results

<i>Methods</i>	<i>RMSE</i>	<i>0-1 Loss</i>	<i>Runtime</i>
MLE	0.2596	0.2093	1.1
Laplace	0.2499	0.2093	1.1
M-estimation	0.2485	0.2068	1.1
HDP	0.2436	0.2078	4.9
M-branch	0.2428	0.2062	9.3
HGS	0.2410	0.2059	1.1

To compare the three hierarchical smoothing methods more intuitively, we take the RMSE of HGS as the benchmark for each dataset and subtract RMSE of M-branch and HDP, and drew Figure 2. The X-axis represents the datasets arranged from large to small in terms of data size. The Y-axis represents the RMSE difference between HGS and each method, which is the lower, the better. The points in the grey area represent the datasets that perform better with HGS. The lower the point is, the stronger the advantage of HGS is. The following

conclusions can be drawn. First, there are more points in the grey area, which indicates that HGS makes the majority of the datasets with better estimates. Second, the * points are mostly centred around the line $y = 0$, while o points are more diffuse than *. This means HDP has high variance compared with M-branch. Last, among the top 20 largest datasets with more than 10,000 examples on the far left of the figure, HDP makes 14 out of them performs better than HGS and M-branch. This indicates that HDP is more helpful on large datasets.

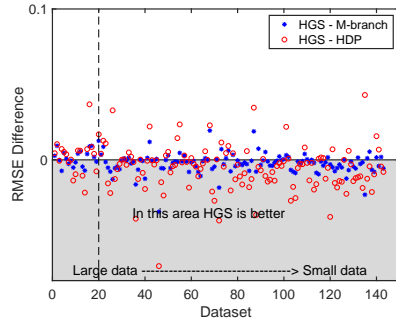


Fig. 2: Compare the probability estimates of HGS with HDP and M-branch.

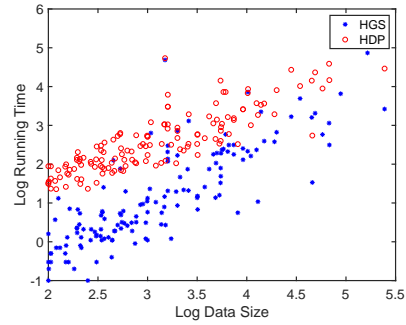


Fig. 3: Training time comparison according to log data size.

4.4 Running Time vs. Data Size

One of the most important motivations of HGS is its efficiency, i.e. running time. Table 1 only gives the averaged running time over all the datasets. Here it is also interesting to investigate the running times based on different data size. Figure 3 is the running time versus data sizes plot for HGS and HDP evaluated on all the datasets. We take the log of both the data sizes and the running times to make the figure more intuitive. It is evident that the blue * are almost always lower than the red circles, which indicates that the training time of HGS on datasets of different sizes is basically shorter than that of HDP.

4.5 HGS on Random Forest

We sought to determine how many trees are needed in RF to beat HGS on a single tree, and the impact of smoothing with RF. Previously, [1] suggested that a non-corrected probability estimate should be used in RF. Figure 4 shows the RMSE changing with the forest size. RF_HGS represents random forest using HGS smoothing, while RF means no smoothing. C4.5 with HGS smoothing is represented by line $y = 0.24$. This figure shows that HGS makes RF worse after three trees because smoothing can reduce the diversity of RF. A single C4.5 tree using HGS yields RMSE better or close to RF with 7 trees and comparatively for 10 trees. While single trees with HGS smoothing cannot beat RF, a single tree is preferred if one is more interested in interpretability.

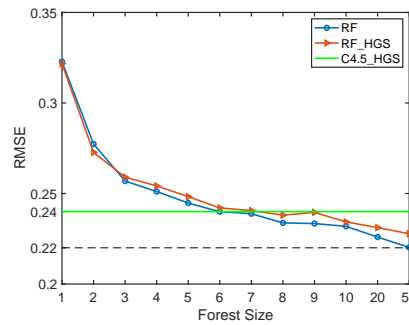


Fig. 4: HGS Smoothing on Random Forest in RMSE.

4.6 Conclusions

It is well known that probability smoothing beats pruning, and M-branch showed us that hierarchical smoothing could further improve performance. This paper, however, develops a new hierarchical algorithm, HGS, and tests out a recent algorithm, HDP smoothing on trees for the first time. The originality of HGS is in removing recursive smoothing and efficient pre-computation of key statistics, which also allow better optimization of hyper-parameters. This experimental evaluation demonstrates three significant contributions.

- HDP smoothing developed in [13] is shown to be comparable to M-branch, and evidence suggests it is the superior algorithm for large data sets.
- HGS is an order of magnitude faster than M-branch and HDP smoothing, and significantly better in RMSE.
- HGS is generally superior to a random forest with 7 trees and almost as good with 10 trees, which makes HGS a single tree alternative to a random forest with 10 trees or less.

There has been a renewed interest in decision trees, and thus also PETS. For applications that require more interpretability and efficiency, such as online and embedded applications, a PET built using a single decision tree plus HGS is thus suitable for these.

References

1. Bostrom, H.: Estimating class probabilities in random forests. In: Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on. pp. 211–216. IEEE (2007)
2. Chen, S.F., Goodman, J.: An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* **13**(4), 359–394 (1999)
3. Dua, D., Graff, C.: UCI machine learning repository (2017)
4. Elkan, C.: The foundations of cost-sensitive learning. In: International Joint Conference on Artificial Intelligence. vol. 17, pp. 973–978. Lawrence Erlbaum Associates Ltd (2001)

5. Ferri, C., Flach, P., Hernández-Orallo, J.: Decision trees for ranking: effect of new smoothing methods, new splitting criteria and simple pruning methods. Technical report, DSIC 2003 (2003)
6. Ferri, C., Flach, P.A., Hernández-Orallo, J.: Improving the AUC of probabilistic estimation trees. In: European Conference on Machine Learning. pp. 121–132. Springer (2003)
7. Joulani, P., Gyorgy, A., Szepesvári, C.: Fast cross-validation for incremental learning. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
8. Kneser, R., Ney, H.: Improved backing-off for m-gram language modeling. In: ICASSP. vol. 1, p. 181e4 (1995)
9. Kohavi, R.: The power of decision tables. In: European Conference on Machine Learning. pp. 174–189. Springer (1995)
10. Liang, H., Zhang, H., Yan, Y.: Decision trees for probability estimation: An empirical study. In: 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06). pp. 756–764. IEEE (2006)
11. Manapragada, C., Webb, G.I., Salehi, M.: Extremely fast decision tree. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1953–1962. KDD '18, ACM, New York, USA (2018)
12. Murdoch, W.J., Singh, C., Kumbier, K., Abbasi-Asl, R., Yu, B.: Definitions, methods, and applications in interpretable machine learning **116**(44), 22071–22080 (2019)
13. Petitjean, F., Buntine, W., Webb, G.I., Zaidi, N.: Accurate parameter estimation for Bayesian network classifiers using hierarchical Dirichlet processes. Machine Learning (May 2018)
14. Provost, F., Domingos, P.: Tree induction for probability-based ranking. Machine Learning **52**(3), 199–215 (2003)
15. Quinlan, J.R.: C4.5: Programs for machine learning. The Morgan Kaufmann Series in Machine Learning, San Mateo, CA: Morgan Kaufmann,— c1993 (1993)
16. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)
17. Shareghi, E., Haffari, G., Cohn, T.: Compressed nonparametric language modelling. In: Proc. of the Twenty-Sixth International Joint Conference on Artificial Intelligence. pp. 2701–2707 (2017)
18. Shen, Y.: Loss functions for binary classification and class probability estimation. Ph.D. thesis, University of Pennsylvania (2005)
19. Teh, Y.W., Jordan, M.I.: Hierarchical Bayesian nonparametric models with applications. Bayesian Nonparametrics **1** (2010)
20. Wang, T., Qin, Z., Jin, Z., Zhang, S.: Handling over-fitting in test cost-sensitive decision tree learning by feature selection, smoothing and pruning. Journal of Systems and Software **83**(7), 1137–1147 (2010)
21. Zadrozny, B., Elkan, C.: Learning and making decisions when costs and probabilities are both unknown. In: Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 204–213. ACM (2001)
22. Zadrozny, B., Elkan, C.: Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In: ICML. vol. 1, pp. 609–616. Citeseer (2001)
23. Zhang, K.: Probability estimation trees: empirical comparison, algorithm extension and applications. Ph.D. thesis, Tulane University (2006)