

# Is random model better? On its accuracy and efficiency

Wei Fan, Haixun Wang, Philip S. Yu, and Sheng Ma

IBM T.J. Watson Research

Hawthorne, New York 10532

{weifan,haixun,psyu,shengma}@us.ibm.com

## Abstract

*Inductive learning searches an optimal hypothesis that minimizes a given loss function. It is usually assumed that the simplest hypothesis that fits the data is the best approximate to an optimal hypothesis. Since finding the simplest hypothesis is NP-hard for most representations, we generally employ various heuristics to search its closest match. Computing these heuristics incurs significant cost, making learning inefficient and unscalable for large dataset. In the same time, it is still questionable if the simplest hypothesis is indeed the closest approximate to the optimal model. Recent success of combining multiple models, such as bagging, boosting and meta-learning, has greatly improved the accuracy of the simplest hypothesis, providing a strong argument against the optimality of the simplest hypothesis. However, computing these combined hypotheses incurs significantly higher cost. In this paper, we first advert that as long as the error of a hypothesis on each example is within a range dictated by a given loss function, it can still be optimal. Contrary to common beliefs, we propose a completely random decision tree algorithm that achieves much higher accuracy than the single best hypothesis and is comparable to boosted or bagged multiple best hypotheses. The advantage of multiple random tree is its training efficiency as well as minimal memory requirement.*

## 1. Introduction

Inductive learning has wide applications in many different fields. Extensive research has been conducted in the past 2 decades to improve the accuracy of the model. We have found that most of the previously proposed inductive learning approaches, computing either the simplest model that fits the data or combining multiple hypotheses, may be way too complicated and too costly beyond necessity if minimizing a given loss function is the only purpose.

**Optimal Model** For a target function  $t = F(\mathbf{x})$ , given a training set of size  $n$ ,  $\{(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_n, t_n)\}$ , an induc-

tive learner produces a model  $y = f(\mathbf{x})$  to approximate the true function  $F(\mathbf{x})$ . Usually, there exists  $\mathbf{x}$  such that  $y \neq t$ . In order to compare performance, we introduce a loss function. Given a loss function  $L(t, y)$  where  $t$  is the true label and  $y$  is the predicted label, an optimal model is one that minimizes the average loss  $L(t, y)$  for all examples, weighted by their probability. Typical examples of loss functions in data mining are 0-1 loss and cost-sensitive loss. For many problems,  $t$  is nondeterministic, i.e., if  $\mathbf{x}$  is sampled repeatedly, different values of  $t$  may be given. The optimal decision  $y_*$  for  $\mathbf{x}$  is the label that minimizes the expected loss  $E_t(L(t, y_*))$  for a given example  $\mathbf{x}$  when  $\mathbf{x}$  is sampled repeatedly and different  $t$ 's may be given. For 0-1 loss function, the optimal prediction is the most likely label or the label that appears the most often when  $\mathbf{x}$  is sampled repeatedly. For cost-sensitive loss, the optimal prediction is the one that minimizes the empirical risk. In order to make the optimal decision, it is generally agreed that a posterior probability  $P_t(y|\mathbf{x})$  for example  $\mathbf{x}$  to be of class  $y$  is needed.

**Simple Best Hypothesis** There has been significant amount of work in machine learning and data mining to compute “an” (as distinguished from “the”) optimal model. One major interest is on simple hypothesis. According to Occam’s razor, we prefer the simplest hypothesis that fits the data (or “single best hypothesis”). For example, in decision tree construction [12, 4], a decision tree with fewer number of nodes that fits the data is usually preferred. For a chosen model representation and a formal definition of “simplicity,” it is usually NP-hard to find the simplest hypothesis that fits the data. Most of the practical techniques is to use “greedy” to approximate the simplest model with various heuristics. For decision tree, typical heuristics include information gain [12], gini index [10], Kearns-Mansour criterion [9] among others. After the model is completely constructed, it is then further simplified via pruning with different techniques such as MDL-based pruning [11], reduced error pruning [12], cost-based pruning [2], among others.

Although no longer NP-hard, most of these techniques

still require significant amount of computation as well as storage. Most algorithms require multiple scans of the training data and require data to be held in main memory. For example, to compute information gain of every untested feature at each internal node of a decision tree, a brute force decision tree learner either has to group data items according to discrete feature value or sort them for continuous features. After a feature is chosen, the subset of data at the node has to be split into multiple subsets. There have been later proposals to improve the efficiency and scalability of decision tree learning [13, 10, 8]. However, these solutions employ rather sophisticated indexing and hashing data structures and do not completely solve the problems. In the same time, it is still questionable about the true accuracy of the simplest model; it is now a general consensus that the pruned decision tree is not necessarily better than the unpruned decision tree. One well cited example is the KDD-CUP'98 donation dataset; the pruned decision tree will not solicit to any people and consequently will not receive any donation [15].

**Combining Multiple Hypotheses** Apart from model comprehensibility and simplicity, it is doubtful whether the efforts to look for the simplest model that fits the data is indeed worthwhile just from accuracy point of view, given the fact that searching this model compromises learning efficiency and scalability. In [6], Domingos criticized one common notion of Occam's razor, "Simplicity leads to better accuracy." Empirical studies [14] have also shown that the simplest tree is not the best. The most evident supporting results are the success of bagging [3], boosting [7] and meta-learning [5]. Besides improved accuracy, one biggest problem for the family of complicated hypotheses is the learning efficiency.

**The Problem** In summary, the observation we have found is that:

*Most of the previously proposed inductive learning approaches, computing either the simplest model that fits the data or combining multiple hypotheses, may be way too complicated and too costly beyond necessity.*

There exist multiple optimal models for a given loss function. To minimize a given loss function, in fact, it is not necessary to have the true probability  $P_t(y|\mathbf{x})$  as long as the estimated probability  $P(y|\mathbf{x})$  is within a tolerable error range of the true value. For a two-class problem, assume that the true probability for  $\mathbf{x}$  to be of class  $y$  is 0.9. An inaccurate estimate of  $p(y|\mathbf{x})$  to be 0.51 will have exactly the optimal prediction under 0-1 loss function.

**The Solution** We propose a multiple random decision tree algorithm. The algorithm works as follows. At each level of the tree, a non-tested feature is "randomly" chosen without using any training data. The tree construction stops as soon as the depth of the tree exceeds a predefined limit. The

structure of the tree is only dependent on the order that the features are randomly chosen, and is completely independent from the training data. After the structure of the tree is built, we use the training data to update the statistics of each node. Each node records the number of examples of different classes that are "classified" through that node. The process is similar to classifying with a decision tree. Since each data item can be read once to update multiple random trees one by one, the update process requires only one complete scan of the data. To classify an example  $\mathbf{x}$ , the probability outputs from multiple random trees are averaged as an estimate to  $P_t(y|\mathbf{x})$ .

## 2. Random Decision Tree

The "strawman" version of random decision tree algorithm is summarized in Algorithm 1. It first builds the structure of  $N$  random decision trees without the data. It then updates the statistics of each node by scanning the training examples one by one. The feature set  $X = \{\mathcal{F}_1, \dots, \mathcal{F}_k\}$  is provided to construct the structure of the tree. For discrete feature,  $\mathcal{F}_i = \{d_1, \dots, d_m\}$  is a list of valid values. For convenience,  $\mathcal{F}_i(\mathbf{x})$  denotes the value of feature  $\mathcal{F}_i$  of example  $\mathbf{x}$ . For continuous feature, we either use discretization or randomly pick a dividing point (i.e.,  $<$  and  $\geq$  as the testing condition). For illustration purpose, we use discretization in the description of the algorithm. When classifying an example  $\mathbf{x}$ , the probability outputs from multiple random trees are averaged to estimate *a posteriori* probability. The loss function is required to make the decision.

To handle missing values in the training data, each example  $\mathbf{x}$  is assigned an initial weight of  $w = 1.0$ . When missing feature value is encountered, the current weight of  $\mathbf{x}$  is distributed across its children nodes. If the prior distribution of known values are given, the weight is distributed in proportion to this distribution. Otherwise, it is equally divided among the children nodes. The tree can grow up to the full number of features and the exact depth can be limited by a given parameter in tree growth. The structure of the tree can be simplified after training. We can remove all *empty* nodes (i.e., nodes with no examples) and their descendants without changing the logic of the tree. There may be practical reasons to keep these empty nodes. For example, if the same tree structure will be reused for new dataset such as streaming data, temporary empty nodes should be kept.

**Single Feature Information Gain Bias of Decision Trees** To the best of our knowledge, state-of-the-art decision algorithms (such as Ross Quinlan's C4.5 and Christian Borgelt's dti among others) use a "simple feature" information gain bias. The algorithm grows a tree *if and only if* there is at least one feature that has information gain. For the following example,

```

TRAIN( $S, X, N$ )
Data      : training set  $S = \{(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_n, t_n)\}$ , set of
              features  $X = \{\mathcal{F}_1, \dots, \mathcal{F}_k\}$ .  $\mathcal{F}$  is a feature descrip-
              tor.  $N$  is the number of random decision trees.
Result    :  $N$  random trees  $\{T_1, \dots, T_N\}$ 
begin
  for  $i \in \{1, \dots, N\}$  do
    BuildTreeStructure( $T_i, X$ );
  end
  for  $(\mathbf{x}, t) \in S$  do
    for  $i \in \{1, \dots, N\}$  do
      UpdateStatistics( $T_i, (\mathbf{x}, t)$ );
    end
  end
  return  $\{T_1, \dots, T_N\}$ 
end

BuildTreeStructure( $T, X$ )
begin
  if  $X = \emptyset$  then
    make the node as leaf;
  end
  else
    randomly choose feature  $\mathcal{F}$ ;
     $\mathcal{F}$  is the testing feature at current node;
    Feature  $\mathcal{F}$  has  $m$  valid values,  $\{d_i\}$ ;
    construct  $m$  children nodes,  $n_j$ , for each valid value  $d_j$ ;
    for  $j \in \{1, \dots, m\}$  do
      BuildTreeStructure( $n_j, X - \{\mathcal{F}\}$ );
    end
  end
end

UpdateStatistics( $T, (\mathbf{x}, t)$ )
begin
   $n[t]$  keeps the number of examples with class label  $t$ ;
   $n[t] \leftarrow n[t] + 1$ ;
  if the node is a not a leaf then
     $d$  is the value of the node's testing feature  $\mathcal{F}$ , i.e.,  $d = \mathcal{F}(\mathbf{x})$ ;
     $n$  is the corresponding child node for the feature value  $d$ ;
    UpdateStatistics( $n, (\mathbf{x}, t)$ );
  end
end

CLASSIFY( $\{T_1, \dots, T_N\}, \mathbf{x}$ )
begin
  For tree  $T_i$ ,  $P_i(y|\mathbf{x}) = \frac{n[y]}{\sum_y n[y]}$  where  $n[y]$  is the count at
  the leaf node that  $\mathbf{x}$  finally reaches to;
  return  $\frac{1}{N} \sum_{i=1}^N P_i(y|\mathbf{x})$  for all class labels  $y$ ;
end

```

**Algorithm 1:** Random decision tree algorithm

$f_1$	$f_2$	label
0	0	<b>F</b>
0	1	<b>T</b>
1	0	<b>T</b>
1	1	<b>F</b>

Neither  $f_1$  nor  $f_2$  carries information gain *independently*. Only when they are examined *jointly*, they are predictive. State-of-the-art decision tree algorithms that only considers one feature at a time will compute a default model that always predicts either **T** or **F**. The empirical reason is that the computation of combined features involves combinatorial explosion and is intractable. The proposed random decision algorithm has the same “single feature information gain bias”.

### 3. Analysis

We first discuss the heuristics to choose tree depth as well as the number of trees, then discuss the training cost and memory requirement.

**Heuristics to choose tree depth** The main reason for multiple random trees is to create diversity. However, when the random trees are too deep, the diversity may actually reduce. In the extreme case, there may be no diversity at all. When the random trees are generated to the depth of the complete feature set, it is very likely that they are equivalent to rote learning and every random tree is virtually the same. The simplest rote learning is to group data items with exactly the same feature values together and use the most frequent class as the prediction. If a fully-generated random tree have no empty nodes, in other words, all the nodes or feature tests in every path has supporting examples, it will degenerate into rote learning. Rote learning tests every feature of an example and a full depth tree with no empty nodes essentially does the same. When every feature is tested, the sequence of how they are tested are no longer significant.

We decide the depth of tree using combinatorics. Assume that there are  $k$  features in total and  $i$  is the number of feature tests.  $C_k^i$  is the number of unique ways to create  $i$  feature tests.  $C_k^i = \frac{k!}{i!(k-i)!}$  is the number of unique combinations to sample  $i$  items with replacement from a population of  $k$ . Since each tested feature cannot be used again, we actually sample features with replacement. For example, when there are 4 features, there are  $C_4^2 = 6$  unique ways to test 2 different features.  $C_k^i$  peaks at  $i = \frac{k}{2}$ . When we limit the random tree's depth to be  $\frac{k}{2}$ , i.e., the depth of tree is half of the number of features, we create the most diversity.

$$\text{Tree Depth: } \frac{k}{2} \quad (1)$$

**Heuristics to choose the number of trees** Since each tree is constructed randomly, it is in effect a random sample from a population of trees of a given depth. The averaging of multiple trees is to approximate the true mean of a large sample. When each feature is binary, the population size is  $2^{\frac{k}{2}}$ . We recast the random decision tree algorithm as a statistical sampling problem. The average probability is the sample mean using  $N$  trees. The problem is to decide if it is worthwhile to have larger samples, i.e., more random trees. Statistically, it is found that, quite surprisingly, a random sample as small as  $N = 10$  gives very good result. When  $N \geq 30$ , the sample's mean (i.e., average probability in our case) usually has small error with at least 99.7% confidence. Because of the “error-tolerance” property in optimal decision making,  $N = 10$  is probably sufficient for most applications.

We analyze the effect of the algorithm for the *worst case* scenario to derive the lower bound of the *confidence* that

multiple random tree is the optimal model. The worst case is a dataset that has only one relevant feature. All remaining  $k - 1$  features are irrelevant. This is the worst case because the single best tree will ignore all irrelevant feature and chooses the single relevant feature, resulting in an optimal model. However, the random tree may use all but the relevant feature. We derive the number of random trees ( $N$ ) needed to ensure its optimality with confidence  $p$ . Please note that the example in Section 2 is not a worst case. When there is no “single” feature that has information gain independently, none of the state-of-the-art decision tree learner (eg. C4.5 and dti) that considers one feature at a time will be able to compute a correct model.

The *a posteriori* probability estimate by any number of irrelevant features approximates the default probability in the training data. If at least one random tree uses the only good feature to classify  $x$ , the averaged probability will be better than the default probability. The averaged probability may not be the same as the probability output of the single best tree. However, due to the error-tolerance property, it may not make a difference. Since every feature is chosen equally during tree construction, it has the same probability to appear in a decision path from the root to the leaf. Therefore, the probability for one feature to appear in a decision path is  $p = \frac{1}{k}$ . Considering all  $N$  random trees, the probability for at least one path to test the only relevant feature is at least

$$\text{Confidence Lower Bound: } p(k, N) = 1 - \left(1 - \frac{1}{k}\right)^N \quad (2)$$

In other words, with confidence of at least  $p(k, N)$ , the random decision tree is optimal. This probability estimate is under the worst assumption that the decision path of different trees is completely independent. However they may be partially correlated due to the feature values of  $x$ . The actual probability will be higher than the above estimate. In practice, when there are a large number of features, the feature sets should be filtered to remove irrelevant features. Please note that Eq(2) does not contradict Eq(1). In the worst case assumption of Eq(2), the optimal decision is the prediction by the single relevant feature. Under this situation, tree depth is no longer important. This is because as long as the single relevant feature appears in a decision path of arbitrary length, a random tree will make the optimal decision.

**Training Cost and Memory Requirement** The training cost comes from updating the statistics in each node. This is similar to classifying a dataset. The main overhead comes from scanning the training data. The algorithm can be easily implemented to update multiple random trees from one complete data scan. For realistic problems, the multiple random trees can be held entirely in main memory, and all updates are done in main memory. Since there is no utility to keep a data record afterwards, we only need to keep the

multiple random trees and one data item at any given time. Since the random trees are iso-depth, one can construct one random tree from the data dictionary to measure its memory requirement and multiply by the total number of trees to compute the total memory requirement. In rare cases that the limited memory cannot hold multiple trees simultaneously, multiple trees can be computed from multiple scans. Learning a single best tree may not even be possible since it requires the entire data to be held in main memory.

## 4. Experiment

Since the random decision tree algorithm is contrary to common beliefs, the biggest concern is on its actual accuracy. We use both 0-1 loss and cost-sensitive loss functions to evaluate its performance. We compare its accuracy with the single best unpruned and pruned decision trees, bagging, boosting, meta-learning, MetaCost as well as decision trees with calibrated probability, i.e., Laplace smoothing, curtailment and their combination. We also study the effect of applying various calibrated probability methods to smooth the probability estimates of random trees. We varied both the number of random trees and the depth of each tree simultaneously to study the change in its performance. We recorded the size of each tree and the size of the training set. We also generated an artificial dataset with only one good feature and all  $k - 1$  remaining features are irrelevant. We count the probability that the random trees are the optimal model as a function of the number of trees.

**Data Sets** We have carefully selected three datasets from real world applications. The first one is the famous donation dataset that first appeared in KDDCUP’98 competition. Suppose that the cost of requesting a charitable donation from an individual  $x$  is \$0.68, and the best estimate of the amount that  $x$  will donate is  $Y(x)$ . Its benefit matrix (converse of loss function) is:

	predict <i>donate</i>	predict <i>¬donator</i>
actual <i>donate</i>	$Y(x) - \$0.68$	0
actual <i>¬donate</i>	-\$0.68	0

The accuracy is the total amount of received charity minus the cost of mailing. Assuming that  $p(\text{donate}|x)$  is the estimated probability that  $x$  is a donor, we will solicit to  $x$  iff  $p(\text{donate}|x) \cdot Y(x) > 0.68$ . The data has already been divided into a training set and a test set. The training set consists of 95412 records for which it is known whether or not the person made a donation and how much the donation was. The test set contains 96367 records for which similar donation information was not published until after the KDD’98 competition. We used the standard training/test set splits to compare with previous results. The feature subsets (7 features in total) were based on the KDD’98 winning submission. To estimate the donation amount, we employed the multiple linear regression method. As suggested

in [15], to avoid over estimation, we only used those contributions between \$0 and \$50.

The second data set is a credit card fraud detection problem. Assuming that there is an overhead  $v \in \{\$60, \$70, \$80, \$90\}$  to dispute and investigate a fraud and  $y(x)$  is the transaction amount, the following is the benefit matrix:

	predict <i>fraud</i>	predict $\neg$ <i>fraud</i>
actual <i>fraud</i>	$y(x) - v$	0
actual $\neg$ <i>fraud</i>	$-v$	0

The accuracy is the sum of recovered frauds minus investigation costs. If  $p(\text{fraud}|\mathbf{x})$  is the probability that  $\mathbf{x}$  is a fraud, *fraud* is the optimal decision iff  $p(\text{fraud}|\mathbf{x}) \cdot y(x) > v$ . The dataset was sampled from a one year period and contains a total of .5M transaction records. The features (20 in total) record the time of the transaction, merchant type, merchant location, and past payment and transaction history summary. We use data of the last month as test data (40038 examples) and data of previous months as training data (406009 examples).

The third dataset is the adult dataset from UCI repository. We artificially associate a benefit of \$1 ~ \$10 to class label **F** and a fixed benefit of \$1 to class label **N**, as summarized below:

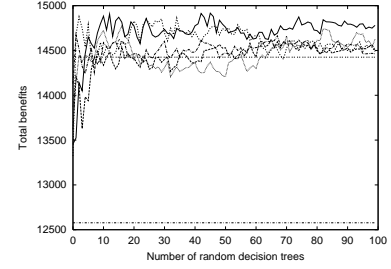
	predict <b>F</b>	predict <b>N</b>
actual <b>F</b>	\$1 ~ \$10	0
actual <b>N</b>	0	\$1

We use the natural split of training and test sets, so the results can be easily duplicated. The training set contains 32561 entries and the test set contains 16281 records. The feature set contains 14 features that describe the education, gender, country of origin, marital status, capital gain among others.

**Decision Tree Learner** The experiments were based on C4.5 release 8 [12]. The single best tree is constructed by running C4.5 with all the default settings. It computes both the unpruned and pruned decision trees. The random decision tree algorithm is a modified version of C4.5 that does not compute information gain and constructs the structure of the tree prior to scanning the data.

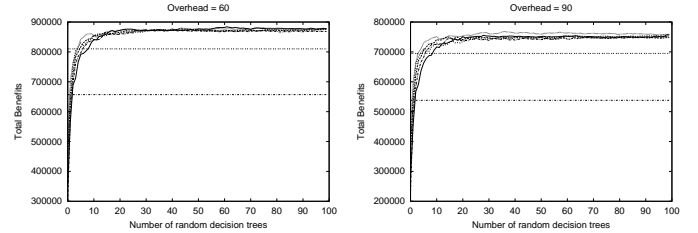
**Accuracy Comparison** Since random decision tree may introduce large variation in accuracy, we run each test 5 times. For each test, up to 100 random trees are constructed. The results are plotted and summarized in Figures 1 to 3. We use the term accuracy to mean number of correctly predicted examples for 0-1 loss and total benefits for cost-sensitive loss.

The result for the donation dataset is shown in Figure 1. A pruned decision tree by C4.5 has only one default rule that says no one is a donor, and will not send any solicitation. Therefore, it will not receive any donation. In Figure 1, we plot the total benefits of all 5 test runs of random deci-



Model	Total Benefits
pruned	\$0
unpruned	\$12577.61
calibrated	\$14424.9
10 random tree	\$14567

Figure 1. Random Tree on Donation Dataset



Model	Total Benefits			
	\$60	\$70	\$80	\$90
pruned	\$810115	\$760256	\$728482	\$694955
unpruned	\$696596	\$614644	\$576634	\$537903
calibrated	\$801034	\$761244	\$731655	\$691044
10 random tree	\$830431	\$779410	\$751455	\$712314

Figure 2. Random Tree on Credit Card

sion tree. We have also plotted two baselines in the same figure as a comparison. The first baseline, shown as the straight line in the bottom, is the total benefits ( $=\$12577.61$ ) of the single unpruned decision tree. The second baseline, shown as the straight line at the top of the figure, is the total benefit ( $=\$14424.9$ ) of the best calibrated probability methods. We experimented with Laplace smoothing, curtailment as well as curtailment plus smoothing [15]. The best calibrated method for donation dataset is curtailment plus smoothing. The detailed numbers are summarized in the table of Figure 1. We compare the results of 10 random trees (averaged over 5 test runs) with the single-best unpruned tree, pruned tree as well as unpruned tree with calibrated probability output.

There are several important observations. First, when the number of random trees exceeds 10, the average total benefits of the random tree algorithm is more than that of the best method. The average total benefits of 10 random trees is \$14567, but the best single decision tree method (un-

Model	Accuracy	
	$c(F)=1$	$c(F)=10$
pruned	13996	43277
unpruned	13733	41710
calibrated	13733	42127
10 random tree	13966.2	44137.8

Figure 3. Random Tree on Adult Dataset

pruned decision tree with “curtailment plus smoothing”) is \$14424.9. Comparing the result of unpruned single best tree with random decision tree, the unpruned single best tree receives a donation of only \$12577.61. The random method receives approximately \$2000 more in donation. Another important observation is that the accuracy of the random trees increases significantly when the number of trees grows from 1 to 10 and starts to stabilize after 10. It starts to fluctuate after 10, but most of them are still higher than the best single decision tree method.

The results on credit card fraud dataset are shown in Figure 2. We plot the total benefits with increasing number of decision trees when the overhead is either \$60 or \$90. The plots for overhead \$70 and \$80 are similar. As a comparison baseline, we also plot the total benefits of the unpruned decision tree (the straight line in the bottom) and pruned decision tree (the one at the top). The phenomenon is similar to the donation dataset. The total benefits of 10 random trees significantly exceed the best of the single model methods. As a summary, the total benefits of the 10 random trees is about \$20000 to \$30000 or 2.5% to 3% higher than the corresponding best single classifier method. As shown in the table of Figure 2, the best single classifier method is either the pruned single decision tree or unpruned tree with calibrated probability. Similar to the donation dataset, when there are more than 10 random trees, the total benefits of multiple random trees stabilize. Computing more random trees doesn’t produce a more accurate model.

The results on the adult dataset is shown in Figure 3. We only show the result for both 0-1 loss (or  $c(F) = 1$ ) and cost-sensitive loss when  $c(F) = 10$ . The phenomenon for other loss functions, i.e.,  $c(F) = 2, \dots, 9$ , are similar. In summary, the random tree significantly improves the accuracy of the best single classifier method and its accuracy stabilizes when the number of trees is more than 10.

**Depth of Random Decision Trees** We have discussed in Section 3 that the optimal depth of random tree is half the number of features. We have varied the depth of the tree from 10% up to the full tree with 10% increment. The results on adult datasets are shown in Figure 4. The results closely confirms our theoretical analysis that when the tree depth is half of the features, the accuracy is the highest. As we can see from the plots, the accuracy starts to increase when the tree depth increases from 10% to 50% of the total number of features, peaks around 50% ~ 60% and be-

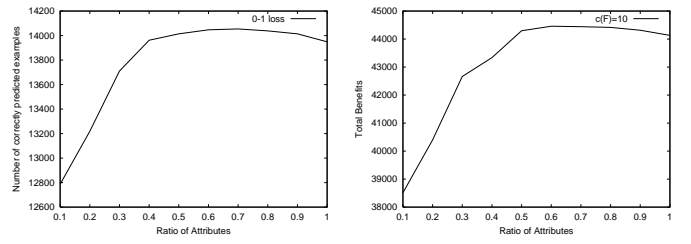


Figure 4. Tree Depth and Accuracy on Adult Dataset

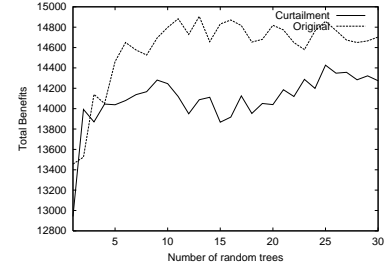


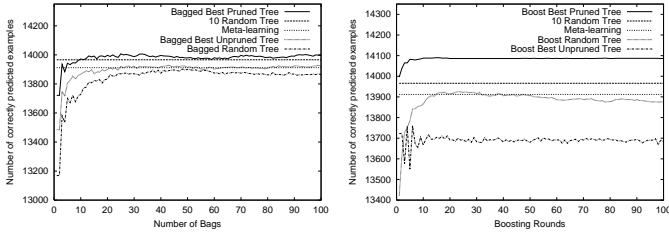
Figure 5. Curtailment on Random Trees - Donation Dataset

gins to drop afterwards.

**Calibration on Random Trees** Previous studies have shown that curtailment is the most effective probability calibration method. Curtailment stops searching down the tree if the number of examples in the current node is less than  $m$ .  $m$  is set as 100 in our experiments. Our results have shown that curtailment has no effect on the accuracy for random decision trees on adult and credit card dataset, but it is detrimental to donation dataset. The total benefits is reduced by about \$100 ~ \$800 for different number of random trees as shown in Figure 5.

**Boosting, Bagging, Meta-learning and MetaCost** As a study of accuracy, we compared multiple random decision trees to boosting, bagging, meta-learning as well as MetaCost. However, it is important to emphasize that none of boosting, bagging and MetaCost are scalable methods. The comparison with boosting, bagging and MetaCost is only limited to accuracy without efficiency and memory concerns. Meta-learning scans the complete training set approximately twice, but the proposed random tree scans the data subset exactly once.

We have applied boosting and bagging to both single best (pruned and unpruned) tree and single random decision tree. We applied the advanced version of AdaBoost that uses the numerical estimation and continuous predictive confidence. For meta-learning, we divided the training data into 8 sub-



**Figure 6. Boosting and Bagging on Adult Dataset (order of legends is the same as the curves)**

sets. Two base classifiers are combined in the same time, resulting tree of classifiers with depth equal to 3. We experimented on the adult dataset with 0-1 loss function. The results are plotted in Figure 6. The x-axis is the number of rounds and the y-axis is the number of correctly predicted examples. The y-axis starts from 13000 in order to “magnify” the trends. The left plot is the result of bagging and meta-learning. Only bagged best pruned tree after 10 round exceeds the accuracy of 10 random trees by only  $\pm 50$  examples (out of 16281). The other 3 bagged methods, i.e., bagged unpruned tree and bagged random tree, are less accurate than 10 random trees. The accuracy of meta-learning is slightly lower than the random tree (by  $\pm 50$  examples). The right plot is the result of boosting. Except for boost best pruned tree, all the other boosting methods, i.e., boost random tree and best best unpruned tree, are significantly less accurate than 10 random tree. Boost best pruned tree exceeds the accuracy of 10 random tree by about  $\pm 80$  examples (out of 16281). It is important to emphasize that boosting and bagging best pruned tree are significantly more costly than computing 10 random trees.

As discussed in [15], MetaCost is significantly more costly for cost-sensitive learning than “empirical risk” based method. It uses multiple classifier bagging to estimate the *a posteriori* probability, intentionally changes the label of some examples to optimize the classification frontier and trains a best classifier at the end. We have applied MetaCost (using 10 round bagging to estimate *posteriori* probability and unpruned best decision tree) on the donation dataset. The total charity received is \$11950 which is significantly less than both the multiple random tree (\$14567) and single best unpruned tree (\$12577.61).

**Training Efficiency** The total time to train the single best decision tree using C4.5 is 2 mins, 4 mins and 90 mins for adult, donation and credit card dataset (full training set) respectively. However, the training cost for random tree is 0.5 min, 0.6 min and 2 mins. If the training data cannot be held in main memory, the training time for the single best decision tree will be significantly longer due to memory swap.

DataSet	Training Size	Tree Size	Ratio
donation	2766942	2233	1239
credit card	27005476	18573	1454
adult	3466402	188945	18

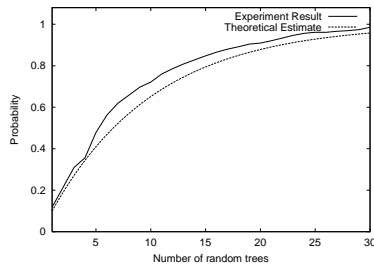
**Table 1. Memory Requirement**

**Memory Requirement** We measure the size of a decision tree from the size of its stored file in the file system. It is a good estimate for the tree kept in main memory. The data size is the file size of the training data stored in the file system. In Table 1, we list the size of the training data, the size of one single random tree (the depth is half of the number of features), and their ratio (closest smaller integer). For the credit card and donation dataset, the same amount of memory to hold the complete dataset can hold more than 1200 random trees. Computing multiple random trees has significantly less memory requirement but can achieve higher accuracy. For the adult data set, the same amount of memory to hold the training data can hold at least 18 random trees. Computing 10 random trees is sufficient. It still results in about 50% less memory requirement.

**Artificial Dataset** In Section 3, we theoretically derive the confidence probability  $p = 1 - (1 - \frac{1}{k})^N$  that the multiple random tree is the optimal model under the pessimistic condition where there is only one good feature and all  $k - 1$  remaining features are irrelevant. To verify this result, we have created an artificial dataset. The class label is binary,  $F$  and  $N$ . There are 50%  $F$ ’s and 50%  $N$ ’s in the training data. There is one good feature that is strictly correlated with the class label. In other words, using this feature alone can predict the true label with 100% accuracy. We then introduced 9 irrelevant features whose values are generated randomly. We compute  $N = 1 \sim 30$  random trees. Tree depth ranges from 10% to 100% of the total number of features with 10% increment. (In each test,  $N$  trees of the same depth are used to classify an example.) One complete test generates  $30 \times 10$  trees. We have completed the experiment 1000 times and measured the probability that the random tree has exactly the same accuracy (100%) as the single best tree. The result is plotted in Figure 7. As we can see the empirical results closely matches the theoretical analysis.

## 5. Related Work

One important distinction of the proposed random tree method from Amit and Geman’s “randomized trees” [1] is as follows. In randomized trees, Amit and Geman randomly generates feature subset from the complete feature set; from each feature subset, they run a conventional decision tree learner to compute the single best tree with different kinds of heuristics such as information gain among others. However, in our proposed method, the random tree uses



**Figure 7. Probability that the random tree is the optimal model when there is only 1 useful feature. All remaining features are pure noise**

the complete feature set. The structure of the tree is constructed by randomly picking an untested feature and splitting point. Our method does not use any heuristic, e.g., information gain, to choose a feature at any given step. The choice of feature at each step is completely stochastic. Our method scans the the training set exactly once. Amit and Geman's randomized trees scans the training set multiple times (each one with a different feature subset). Our proposed random tree and Amit and Geman's randomized tree are orthogonal methods.

## 6. Conclusion

Contrary to common beliefs, we propose a random decision tree algorithm that builds multiple iso-depth random decision trees. When classifying  $x$ , the probability outputs from multiple trees are averaged to estimate *a posteriori* probability. We discuss the heuristics to choose the depth of each random tree; when the tree depth is half of the number of features, the multiple random trees create the most diversity. We then discuss the minimal number of random trees to guarantee that the multiple random tree is an optimal model. We derive the lower bound of the confidence, as a function of the number of averaged random trees, that the multiple random tree is the optimal model. In practice, 10 trees are sufficient for most problems. We empirically evaluate the algorithm on both real world and artificial datasets. The empirical results have shown that the random decision tree algorithm achieves significantly higher accuracy than the single best hypothesis for both 0-1 loss and cost-sensitive problems. Comparing with more expensive combining techniques such as boosting, bagging, meta-learning and MetaCost, averaging random tree algorithm still achieves higher or comparable accuracy. The structure of a random tree is constructed completely independent of the training data. The training data is scanned exactly once to update the statistics in multiple random trees. The memory requirement is to store multiple random trees and one data item at any given time. With the necessary number of

random trees to ensure optimality, the memory requirement of the random tree algorithm is significantly less than learning a single best tree even if the very large training data could have been held entirely in main memory to compute the single best tree.

## References

- [1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588, 1997.
- [2] J. P. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. E. Brodley. Pruning decision trees with misclassification costs. In *European Conference on Machine Learning*, pages 131–136, 1998.
- [3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [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. Chan. *An Extensible Meta-learning Approach for Scalable and Accurate Inductive Learning*. PhD thesis, Columbia University, Oct 1996.
- [6] P. Domingos. Occam's two razors: The sharp and the blunt. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1998.
- [7] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Computer and System Sciences*, 55(1):119–139, 1997.
- [8] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh. BOAT-optimistic decision tree construction. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, 1999.
- [9] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the Annual ACM Symposium on the Theory of Computing*, pages 459–468, 1996.
- [10] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Extending Database Technology*, pages 18–32, 1996.
- [11] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 216–221, 1995.
- [12] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [13] J. Shafer, R. Agrawl, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of Twenty-second International Conference on Very Large Databases (VLDB-96)*, pages 544–555, San Francisco, California, 1996. Morgan Kaufmann.
- [14] J. Shawe-Taylor and N. Cristianini. Data-dependent structural risk minimisation for perceptron decision trees. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 336–342. MIT Press, 1998.
- [15] B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD01)*, 2001.