BDT: Gradient Boosted Decision Tables for High Accuracy and Scoring Efficiency

Yin Lou*
Airbnb, Inc.
yin.lou@airbnb.com

Mikhail Obukhov
LinkedIn Corporation
mobukhov@linkedin.com

ABSTRACT

In this paper we present gradient boosted decision tables (BDTs). A d-dimensional decision table is essentially a mapping from a sequence of d boolean tests to a real value in $\mathbb{R}$.

We propose novel algorithms to fit decision tables. Our thorough empirical study suggests that decision tables are better weak learners in the gradient boosting framework and can improve the accuracy of the boosted ensemble. In addition, we develop an efficient data structure to represent decision tables and propose a novel fast algorithm to improve the scoring efficiency for boosted ensemble of decision tables. Experiments on public classification and regression datasets demonstrate that our method is able to achieve 1.5x to 6x speedups over the boosted regression trees baseline. We complement our experimental evaluation with a bias-variance analysis that explains how different weak models influence the predictive power of the boosted ensemble. Our experiments suggest gradient boosting with randomly backfitted decision tables distinguishes itself as the most accurate method on a number of classification and regression problems. We have deployed a BDT model to LinkedIn news feed system and achieved significant lift on key metrics.

CCS CONCEPTS

• **Computing methodologies → Ensemble methods**;

KEYWORDS

classification; regression; decision table; gradient boosting

1 INTRODUCTION

Gradient boosting is one of the most popular machine learning methods with many applications in classification and regression [6, 7, 9, 14, 19]. It builds an additive model by a greedy stagewise procedure [7, 9] and forms an ensemble of weak regression models, typically regression trees [3]. Standard regression tree learning method grows a tree using the divide-and-conquer algorithm by recursively partitioning the feature space. Each split of an internal node is a binary split on some feature that minimizes the squared error. Multiple stopping criteria can be employed when constructing a tree, e.g., limiting by depth, limiting by number of leaves, etc.

$x_3 <= 5$	$x_6 <= 0$	f
0	0	2
0	1	7
1	0	5
1	1	0

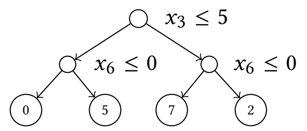


Figure 1: Decision table and its tree equivalent.

A decision table (DT), on the other hand, is a function of the form $f : \mathbb{B}^d \to \mathbb{R}$; it maps a sequence of d boolean tests to a real value in $\mathbb{R}$. We note that a d-dimensional decision table is equivalent to a full (binary) regression tree of depth d where all nodes at the same level share the *same test*. Figure 1 illustrates a decision table and its tree equivalent. It is also easy to see that given an arbitrary regression tree, there exists a decision table equivalent. Therefore, there is no difference in representational capacity between regression trees and decision tables. However, a d-dimensional decision table is more restricted compared to a regression tree of depth d since there can be potentially $2^d - 1$ different splits for such regression tree while the decision table will only have at most d different cuts.

Although regression tree is the *de facto* weak learner in gradient boosting, we note that it is just one implementation of the gradient boosting machine [7]. In this work, we consider gradient boosted decision tables (BDTs) for classification and regression. Decision tables are interesting in the modern setting of gradient boosting for the following reasons.

First, we now have better understandings of the bias and variance tradeoff for gradient boosting; it greatly reduces the bias but may come with the price of increasing variance [1, 10]. It is well known that the predictive power of boosting can be dramatically improved via regularization, such as shrinkage, subsampling, or complexity control of weak models [7, 8]. Common approaches of controlling complexity for regression trees is to limit the tree size. However, trees can still freely cut the feature space by greedy induction. Decision tables, on the other hand, control the model complexity by restricting how to place cuts; when placing a cut, we have to place a full cut in the feature space.

Second, for regression trees, one usually needs to decide how to control the complexity of each tree (e.g., limiting the trees by depth or by number of leaves) in gradient boosting. The most accurate ensemble model is usually selected by including all available options, resulting in a large parameter space. Hence, model selection could be quite expensive. However, due to the restricted structure, dimension is the main tuning parameter for decision tables and therefore model selection is much easier in gradient boosting.

*Work done while at LinkedIn.

KDD '17, August 13-17, 2017, Halifax, NS, Canada
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098175

Third, there is usually a tradeoff between scoring efficiency and model accuracy when deploying boosted ensembles to many real world applications, such as recommendation systems. A decision table can be represented using a compact data structure, which leads to small memory footprint. As we will see in Section 4.5, scoring a d-dimensional decision table is much faster than scoring regression tree of depth d since decision tables are more cache friendly. Therefore, it is easier to train larger (and more accurate) models without increasing the scoring latency.

In this paper, we propose novel algorithms to fit decision tables. Our experiments suggest that boosted decision tables are consistently more accurate than boosted regression trees (BRTs) on classification and regression problems. To further shed the light on the accuracy of various boosted ensembles, we complement our experimental evaluation with a bias-variance analysis that explains how different weak models influence the predictive power of the boosted ensemble. In addition, we develop an efficient data structure to represent decision tables and propose a novel fast algorithm to improve the scoring efficiency of BDTs. Our empirical results demonstrate that our method is able to achieve 1.5x to 6x speedups over boosted regression trees. To evaluate our approach in real world applications, we present the offline and online experimental results on LinkedIn news feed system that illustrate the advantage of our model.

In summary, we make the following contributions in this paper.

- We propose novel algorithms to fit decision tables and introduce boosted decision tables (BDTs).
- We develop an efficient data structure to represent decision tables and propose a novel fast algorithm to improve the scoring efficiency of BDTs.
- We show through a thorough experimental evaluation on public datasets that boosted decision tables are more accurate than boosted regression trees on classification and regression problems.
- We present a bias-variance analysis to show how different weak models influence the generalization error and we find that BDTs are low bias and low variance.
- We find empirically that our method of scoring BDTs is able to achieve 1.5x to 6x speedups over baseline methods.
- We report our successful deployment of boosted decision tables to LinkedIn news feed system and present offline and online experimental results.

The rest of this paper is organized as follows. Section 2 presents preliminaries. In Section 3, we present algorithms for learning decision tables. Experimental results on public datasets and LinkedIn news feed are presented in Section 4 and Section 5, respectively. We present related work in Section 6 and conclude the paper in Section 7.

2 PRELIMINARIES

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_1^N$ denote a dataset of size N, where $\boldsymbol{x}_i = (x_{i1}, ..., x_{ip})$ is a feature vector with p features and y_i is the response. Let $\boldsymbol{x} = (x_1, ..., x_p)$ denote the features in the dataset. In this paper, we consider regression problems where $y_i \in \mathbb{R}$ and binary classification problems where $y_i \in \{0, 1\}$. Let $[n]$ denote the set $\{1, 2, ..., n\}$.

Algorithm 1 Gradient Boosting

```
1: $F \leftarrow 0$
2: **for** $m = 1$ to $M$ **do**
3:     $r_i \leftarrow$ current (pseudo) residual for $i \in [N]$
4:     $\mathcal{R} \leftarrow \{(\boldsymbol{x}_i, r_i)\}_1^N$
5:     $T_m \leftarrow$ a regression model trained on $\mathcal{R}$
6:     $F \leftarrow F + \nu T_m$
```

Gradient boosting is an ensemble method that builds an additive model by a greedy stagewise procedure [7, 9]. It aims to find an approximation $F(\boldsymbol{x})$ of the unknown true function $F^*(\boldsymbol{x})$ that minimizes the following objective function,

$$L(y, F(\boldsymbol{x})), \tag{1}$$

where $L(\cdot, \cdot)$ is a non-negative convex loss function. When $L = \frac{1}{2N}\sum_{i=1}^N (y_i - F(\boldsymbol{x}_i))^2$, our problem becomes a regression problem and if $L = \frac{1}{N}\sum_{i=1}^N \log(1 + \exp(-y_i F(\boldsymbol{x}_i)))$, we are dealing with a classification problem. Typically regression tree is the main weak learner in gradient boosting.

Algorithm 1 summarizes the general gradient boosting algorithm. At each iteration we form the current residuals (Line 3-4), build a regression model on the residuals (Line 5) and add it to the current ensemble (Line 6); ν is the shrinkage parameter called learning rate. M controls the total number of models in the ensemble. In this paper, we consider MART [7] for regression problems where $r_i = y_i - F(\boldsymbol{x}_i)$ and robust LogitBoost [9, 13] for binary classification problems where $r_i = y_i - p_i$ and $p_i = 1/(1 + \exp(-F(\boldsymbol{x}_i)))$ is the probability of $\boldsymbol{x}_i$ being in positive class.

Gradient boosting usually requires regularization to avoid overfitting, such as shrinkage, subsampling, and model complexity control, etc. Shrinkage reduces the variance of each added model [7]. Subsampling introduces randomness into the algorithm and helps prevent overfitting [8]. Common choices of controlling the tree complexity include limiting depth or limiting number of leaves [10, 19]. Decision tables naturally serve as a class of high-bias predictor to control the model complexity, and dimension d is the main parameter to tune for gradient boosting.

3 DECISION TABLES

In this section, we present data structures for decision table and learning algorithms of fitting decision tables. We also discuss a highly efficient scoring method for boosted decision tables.

3.1 Representation

We first describe our data structure to represent decision tables. Given a d-dimensional decision table, we use an array of length d to store the feature index, another array of length d to store the corresponding cuts, and an array of length 2^d to store the predictions for each combination of the d tests.[1] The prediction value is indexed through a bit vector of length d that represents the results for the d tests. Figure 2 illustrates an example for a 3-dimensional decision table. For input $\boldsymbol{x}_i = (0, 3, 4, 2, 1, 0)$, we go through three tests: $x_3 \leq 5$, $x_6 \leq 0$ and $x_2 \leq 1$ and generate a bit vector $[110]_2 = [6]_{10}$. Now we can locate the prediction as the 7th

[1]Note that d is usually small when used in boosted ensemble.

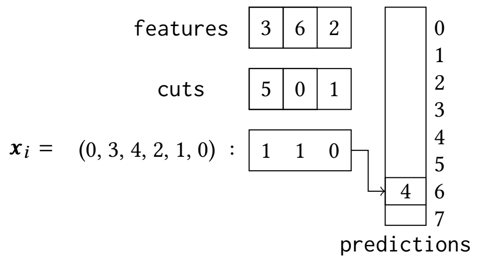


Figure 2: Compact representation of decision tables.

element in the prediction array using the bit vector as index. Such representation leads to very small memory footprint and makes it quite cache friendly.

3.2 Learning Decision Tables

3.2.1 Algorithm. Let us first define how to evaluate a test. Let $\text{dom}(x_j) = \{v_j^1, ..., v_j^{d_j}\}$ be a sorted set of possible values for variable x_j in descending order, where $d_j = |\text{dom}(x_j)|$. Given $d-1$ boolean tests (cuts), the feature space is divided into at most 2^{d-1} partitions. To evaluate the d-th test, we aim to find a cut on some feature so that the squared error can be minimized.

For each partition k, let $\mathcal{D}_k$ denote the subset of points in that partition. For any cut $c \in \text{dom}(x_j)$, let $\mathcal{D}_k^L = \{\boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{D}_k \wedge x_{ij} \leq c\}$ denote the points in $\mathcal{D}_k$ whose value on feature j is smaller than or equal to c and let $\mathcal{D}_k^R = \{\boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathcal{D}_k \wedge x_{ij} > c\}$ denote the points in $\mathcal{D}_k$ whose j-th feature value is larger than c. Let $\mathcal{I}_k^L = \{i \mid \boldsymbol{x}_i \in \mathcal{D}_k^L\}$ denote the indices of points in $\mathcal{D}_k^L$ and similarly we define $\mathcal{I}_k^R = \{i \mid \boldsymbol{x}_i \in \mathcal{D}_k^R\}$. We define the squared error on subset $\mathcal{D}_k$ as $SE_{\mathcal{D}_k} = \sum_{i \in \mathcal{I}_k}(y_i - \overline{y}_k)^2$, where $\overline{y}_k = \frac{1}{|\mathcal{I}_k|}\sum_{i \in \mathcal{I}_k} y_i$.

The gain for a test on feature x_j at cut c is

$$Gain(x_j, c) = SE_{\mathcal{D}} - \sum_k (SE_{\mathcal{D}_k^L} + SE_{\mathcal{D}_k^R}) \tag{2}$$

$$= \sum_k \left(\frac{(\sum_{i \in \mathcal{I}_k^L} y_i)^2}{|\mathcal{I}_k^L|} + \frac{(\sum_{i \in \mathcal{I}_k^R} y_i)^2}{|\mathcal{I}_k^R|} \right) - N\overline{y}^2 \tag{3}$$

where $\overline{y} = \frac{1}{N}\sum_{i=1}^N y_i$.

Thus, the gain for feature x_j is

$$Gain(x_j) = \max_{c \in \text{dom}(x_j)} Gain(x_j, c). \tag{4}$$

With this evaluation function, we can now design the learning algorithm for fitting decision tables. It is easy to see that learning the optimal decision table is NP-complete and therefore we propose a backfitting algorithm as illustrated in Algorithm 2.

We first greedily find d tests one by one for the decision table dt (Line 1-4). By the greedy nature those d tests will not necessarily be optimal. For example, consider an extreme case where we greedily find tests on a parity function, any cuts on such space will generate equal gains and the greedy algorithm will only find random cuts. Therefore, once we have a decision table of d tests, we employ the backfitting algorithm with n passes to correct some suboptimal cuts in the previous run (Line 5). For each pass we optimize for d cuts (Line 6). Each time we pick one of the existing d cuts to optimize according to some criteria (Line 7). We remove that cut from the current decision table (Line 8), compute the best cut that achieves the highest gain given all other cuts (Line 9) and update that cut in the decision table (Line 10-11). In the end, we compute the prediction value as the average value of targets in each partition (Line 12). We note that it is the simple structure of decision table that allows us to perform backfitting, while it is hard to correct suboptimal splits in regression trees.

Algorithm 2 Fit Decision Table of Dimension d

```
1:  for $t = 0$ to $d - 1$ do
2:      $(x_j, c) \leftarrow \arg\max_{x_j, c} Gain(x_j, c)$
3:      $dt$.features[$t$] $\leftarrow j$
4:      $dt$.cuts[$t$] $\leftarrow c$
5:  for $pass = 1$ to $n$ do
6:      for $t = 0$ to $d - 1$ do
7:          $k \leftarrow$ next() {Pick the $k$-th cut to optimize}
8:          remove the $k$-th cut from dt
9:          $(x_j, c) \leftarrow \arg\max_{x_j, c} Gain(x_j, c)$
10:         $dt$.features[$k$] $\leftarrow j$
11:         $dt$.cuts[$k$] $\leftarrow c$
12: update $dt$.predictions
13: return $dt$
```

We now discuss how to pick the next cut to optimize since the `next` method determines how to perform backfitting. In this work, we consider three variants of backfitting as follows.

- Cyclic backfitting. This is the standard backfitting algorithm; we go back to the first cut and re-optimize all cuts in sequential "cyclic" order.
- Random backfitting. We randomly select a cut in the current decision table to backfit.
- Greedy backfitting. This is the greedy stagewise approach where we select a cut to backfit that gives the largest reduction in variance.

It is possible to perform multiple passes of backfitting but as we will see in Section 4.2, additional passes of backfitting result in diminishing returns and can sometimes overfit. In most cases, we perform at most one pass in backfitting.

3.2.2 Implementation Details. Equation (4) is somewhat similar to the splitting criterion in regression tree induction. The main difference is that we can not recursively build cuts since the $Gain(x_j, c)$ requires we have access to all data points in nodes when building d-th cut. Efficient computation of $Gain(x_j)$ is critical to a scalable decision table learning algorithm. Naïve implementation of $Gain(x_j)$ can be expensive. In this section, we discuss an efficient implementation of computing $Gain(x_j)$. Note that since $N\overline{y}^2$ is a constant, we ignore this term when computing $Gain(x_j)$.

We now introduce the data structures needed for efficient computation of $Gain(x_j)$. For each feature x_j we take its unique value set $\text{dom}(x_j)$. Since $\text{dom}(x_j)$ is sorted in descending order, the first element is the largest value for x_j. For each value $v_j^k \in \text{dom}(x_j)$, we define the set $\mathcal{I}(v_j^k) = \{I_{j,1}^k, ..., I_{j,L_j^k}^k\}$ as the indices of points such that $\forall i \in \mathcal{I}(v_j^k), x_{ij} = v_j^k$ and $L_j^k = |\mathcal{I}(v_j^k)|$. Figure 3 illustrates

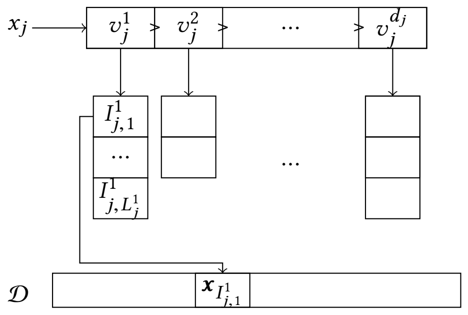


Figure 3: Data structure for DT construction.

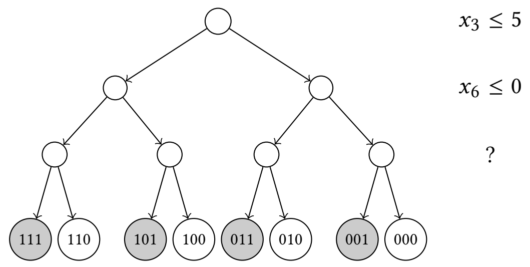


Figure 4: Example of decision table construction.

an example of our data structure. Note that we are essentially re-organizing the data using inverted indexes for each ⟨feature, value⟩ pair.

Assume we want to compute the gain of feature x_j for d-th cut, we will have at most 2^d partitions of the data. The key problem is that for each cut c, we need to quickly identify the points in each partition. To this end, we first label those partitions from 0 to $2^d - 1$ in binary format as we discussed in Section 3.1. Figure 4 illustrates an example using tree representation where the first two tests have been found. We maintain an array L of length N indicating the partition label for each point $\boldsymbol{x}_i \in \mathcal{D}$. For example, if $\boldsymbol{x}_i$ is in partition $[011]_2$, we will have $\mathtt{L}[i] = 3$. For any feature x_j, we start with its largest value v_j^1. This means all points are initially in the shaded circles as illustrated in Figure 4. Suppose we are now testing the cut v_j^2, the second largest value for x_j, we only need to move all points $\boldsymbol{x}_i$ for $i \in \mathcal{I}(v_j^1)$, from the shaded circle to its right. Using our notation for partition labeling, we only need to decrement $\mathtt{L}[i]$ by 1 for each $i \in \mathcal{I}(v_j^1)$.

Algorithm 3 summarizes the procedure of computing the gain of x_j for d-th cut. We need two auxiliary arrays, sum and count, both of length 2^d, where $\mathtt{sum}[k]$ stores the sum of targets of the points in partition k and $\mathtt{count}[k]$ is the number of points in partition k, for $k \in \{0, ..., 2^d - 1\}$ (Line 3-4). For each value v in the sorted set $\mathrm{dom}(x_j)$, we get its indices set $\mathcal{I}(v)$ (Line 5-6). Since we iterate the values in $\mathrm{dom}(x_j)$ in decreasing order, we always move some points from the left circle to the right circle. Therefore, using the partition label array L, we can easily update the sum and count for the two siblings (Line 7-10). Now we compute the current gain for the cut feature x_j at v (Line 12-14) and update the best gain accordingly

Algorithm 3 Compute $Gain(\boldsymbol{x}_j)$

```
1:  $bestGain \leftarrow -\infty$
2:  $c \leftarrow 0$
3:  count $\leftarrow$ auxiliary array of length $2^d$
4:  sum $\leftarrow$ auxiliary array of length $2^d$
5:  **for** $v \in \mathrm{dom}(\boldsymbol{x}_j)$ **do**
6:      **for** $i \in \mathcal{I}(v)$ **do**
7:          count[L[$i$]] $\leftarrow$ count[L[$i$]] $- 1$
8:          sum[L[$i$]] $\leftarrow$ sum[L[$i$]] $- y_i$
9:          count[L[$i$] $- 1$] $\leftarrow$ count[L[$i$] $- 1$] $+ 1$
10:         sum[L[$i$] $- 1$] $\leftarrow$ sum[L[$i$] $- 1$] $+ y_i$
11:     g $\leftarrow 0$
12:     **for** $k = 0$ to $2^d - 1$ **do**
13:         **if** count[$k$] $\neq 0$ **then**
14:             $g \leftarrow g +$ sum[$k$]$^2$/count[$k$]
15:     **if** $g > bestGain$ **then**
16:         $bestGain \leftarrow g$
17:         $c \leftarrow v$
18: **return** $(c, bestGain)$
```

(Line 15-17). Finally we return the best cut c and its corresponding gain (Line 18). It is easy to see the running time for computing the best cut for all features is $O(Np)$, same as that of regression trees.

3.3 Efficient Scoring in Boosted Ensemble

Thanks to the simple structure of decision table, it is possible to scale up the ensemble scoring. Recall that in order to obtain the prediction from a decision table, we only need to know the bit vector that represents the index to look at. Our key observation is that in order to score a boosted ensemble of decision tables, we do not need to go through *all* tests in the decision table ensemble (which is usually the case for boosted regression trees).

Figure 5 illustrates an example of our data structure for fast BDT scoring. For each feature x_j, we collect all tests on x_j in the ensemble and build an index table to store those tests. Each index table has three fields; cut, tid (decision table id) and pos (test position), and the table is sorted by cut in *decreasing* order. Assume the model is an ensemble of d-dimensional decision tables, to score an input feature vector $\boldsymbol{x}$, we first initialize a prediction index array P whose length is the number of the decision tables in the ensemble. All elements in this array is initialized to 0. Now we iterate over all index tables. For each table, we find all entries whose cut values are no less than the current feature value and set the corresponding bit to 1 using tid and pos. Since the table is pre-sorted by cut, we only need to do a linear scan from the largest cut until we find the first cut that is less than the feature value. For example, x_1 is 3 in Figure 5 and we look at its corresponding table. We find that cut values for the first two entries is larger than or equal to 3, we will set the 5th bit in P[7] and the 1st bit in P[1] to 1 and move on to next feature since all tests below the first two entries will be not pass and their corresponding bits are 0 by default. Therefore, we can potentially skip lot of tests.

Algorithm 4 summarizes our approach of scoring BDTs. For each feature x_j, we get its index table cuts (Line 2-3) and scan through cuts. If the current feature value x_j passes the current test, we set

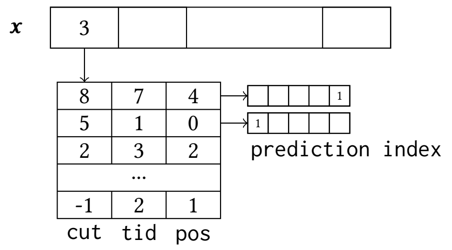


Figure 5: Data structure for fast BDT scoring.

Algorithm 4 Fast Algorithm of Scoring BDTs

```
P ← $\mathbf{0}$
for $j = 1$ to $p$ do
    cuts ← index table for feature $j$
    for $i = 0$ to cuts.$length - 1$ do
        if $x_j <=$ cuts[$i$].$cut$ then
            $t \leftarrow 1 << (d - 1 -$ cuts[$i$].$pos$)
            P[cuts[$i$].$tid$] ← P[cuts[$i$].$tid$] | $t$
        else
            break
$pred \leftarrow 0$
for $i = 0$ to dt.$length - 1$ do
    $pred \leftarrow pred$ + dt[$i$].predictions[P[$i$]]
return $pred$
```

Dataset	Size	Attributes	%Pos
Wine	4989	12	-
CompAct	8192	22	-
Pole	15000	49	-
Bike	17379	15	-
CalHousing	20640	9	-
Magic	19020	11	64.84
Letter	20000	17	49.70
News	39644	59	53.36
Bank	45221	16	11.70
Physics	50000	79	49.72

Table 1: Datasets.

the corresponding bit indicated in that table entry to 1 (Line 5-7), otherwise we can safely leave the bit value for all remaining tests to 0 (Line 9). Once we have iterated through all features, we use the updated prediction index array P to look up the prediction value in each decision table's `predictions` array (Line 10-12).

As we will see in Section 4.5, our approach can achieve 1.5x to 6x speedup compared to standard approach to scoring.

4 EXPERIMENTS ON PUBLIC DATASETS

In this section, we report experimental results of decision tables and their ensembles on public datasets.[2]

[2]Code is available at https://github.com/yinlou/mltk.

4.1 Datasets

Table 1 summarizes the ten datasets used in our experiments. Five are regression problems: The "Wine" and "Bike" datasets are from the UCI repository.[3] "CompAct" is from the Delve repository and describes the state of multiuser computers.[4] "Pole" describes a telecommunication problem [20]. "CalHousing" describes how housing prices depend on census variables [17] . The other five datasets are binary classification problems: The "Magic," "Letter," "News" and "Bank" datasets are from the UCI repository. We convert "Letter" into binary classification using the same method in [6]. "Physics" is from the KDD Cup 2004.[5]

4.2 Decision Tables

Since decision table is a regression model, we run experiments on all five regression problems to study the its accuracy. In this experiment, we train on 80% of the data and hold 20% of the data as test sets. We consider decision tables with cyclic backfitting (DT_{cbf}), random backfitting (DT_{rbf}) and greedy backfitting (DT_{gbf}).

We build the decision tables of dimension 2, 4, and 8 on the training sets. Due to space limitation, we only present results on "CalHousing" dataset. Results on other regression datasets are similar. In Figure 6, we report root mean squared error (RMSE) on test sets and running time for constructing the decision tables. We first observe that in most cases, with more backfitting passes the RMSE on test sets does not decrease significantly. In some cases the highly optimized decision tables may slightly overfit. This is a classic behavior of the backfitting algorithm; the very first few passes explain the most of the variance while adding more passes does not lead to meaningful improvement. On the other hand, Figure 6 suggests the training time increases proportionally as the number of passes increases. Thus, from now on we will only use one pass of backfitting for optimizing decision tables.

In addition, we notice that greedy backfitting takes significantly more time than the other two methods. This is because we need to search for the best test to optimize for each iteration, which is usually very expensive. Therefore, we will only consider cyclic backfitting and random backfitting in gradient boosting framework.

4.3 Boosted Decision Tables

In this section, we report experimental results on boosted ensembles. Based on the results in Section 4.2, we consider the following weak learners; number of leaves limited regression tree (BRT_l), depth limited regression tree (BRT_d), decision table without backfitting (BDT), decision table with one pass of cyclic backfitting (BDT_{cbf}), and decision table with one pass of random backfitting (BDT_{rbf}). For BRT_l and BRT_d, we use the implementation in `MLTK`.[6] We also experimented `XGBoost`. Both `MLTK` and `XGBoost` implement gradient boosted trees but `XGBoost` only allows depth limited regression trees and in our experiments, these two packages produce similar accuracy.

For all experiments in this section we fix our shrinkage parameter as 0.01 and build at most 10,000 models. We consider $d \in \{1, 2, \dots, 9\}$

[3]http://archive.ics.uci.edu/ml/
[4]http://www.cs.toronto.edu/~delve/data/datasets.html
[5]http://osmot.cs.cornell.edu/kddcup/
[6]https://github.com/yinlou/mltk

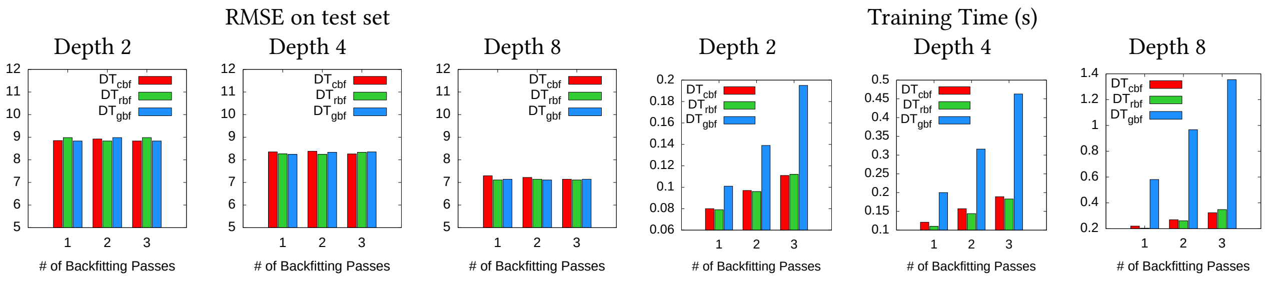


Figure 6: RMSE on test sets and training time for optimized decision tables for "CalHousing" dataset.

for depth limited trees and decision tables, while for number of leaves limited trees, we consider $l \in \{2, 4, ..., 512\}$. We randomly partition the dataset into training (64% of the data), validation (16% of the data) and test sets (20% of the data). We pick optimal parameters (d, l and number of models in the ensemble) using validation set. We do not wish to exhaustively search for the best combination for boosting, but rather show the effectiveness of using different models as weak learners. For regression problems we use RMSE as our metric and we employ error rate for binary classification problems. All experiments are repeated 5 times and we report mean and standard deviation of the evaluation results on test sets.

The regression and classification results are presented in Table 2 and Table 3. We report normalized scores in the last column of the two tables. For both regression and classification problems, we normalize by $\min(\text{BRT}_l, \text{BRT}_d)$. This is the best RMSE or error rate that we can achieve on held-out test data for gradient boosted regression trees by peeking the results on test sets. Note that this sets up a very high baseline.

Interestingly, we observe BDTs are more accurate than BRTs; on average, BDT reduces the error by 4% on regression problems and reduces 3% error rate on classification problems. In our experiments, BDTs are almost always more accurate than BRTs. This suggests that using decision table is a reliable approach to regularization and extra accuracy of boosted ensembles can be obtained.

For BDT_{cbf} and BDT_{rbf}, we can see that they further improve the accuracy on both classification and regression problems. While BDT_{cbf} is sometimes less accurate than BDT (e.g., on "CompAct" dataset), BDT_{rbf} outperforms the other methods most of the time and is the most accurate model on average in the experiment. On average BDT_{rbf} reduces the RMSE by 5% on regression problems and error rate by 4% on classification problems. In the last row of Table 2 and Table 3, we also present the p-value from a paired one-tail t-test of BRT_d and BDT_{rbf}. On most cases the improvement is significant.

4.4 Bias-Variance Analysis

The results in Table 2 and Table 3 show that using decision tables or optimized decision tables in gradient boosting significantly improves accuracy compared to regression trees. The main difference is that decision tables and optimized decision tables have higher bias and lower variance than regression trees.

It is well known that boosting reduces the bias but may lead to an increase in variance [1]. In this section, we perform a bias-variance analysis [11] on regression datasets. The expected squared error is decomposed as follows,

$$E[(y - F(\boldsymbol{x}))^2] = Bias[F(\boldsymbol{x})]^2 + Var[F(\boldsymbol{x})] + \sigma^2, \quad (5)$$

where F approximates the true function and σ^2 is the irreducible error.

We do not perform bias-variance analysis on classification problems since bias-variance decomposition on classification problem is not as well defined. The bias-variance results for all regression datasets are shown in Figure 7. We can see that BRT_l and BRT_d have higher bias and sometimes the variance can be large as well. When we use decision tables as the weak learner, surprisingly we get lower bias for BDT, BDT_{cbf}, and BDT_{rbf}. It is also interesting to see that the variance for gradient boosted decision tables are also smaller, leading to a low bias and low variance model.

We note that although bagging can be used to reduce variance for gradient boosted regression trees [10, 18], it usually requires more resources to train and the model becomes very large to score. From the results in Figure 7, we do not expect bagging would help gradient boosted decision tables since the variance of the model is so low to begin with. In addition, the results suggest that using a high-bias-low-variance weak learner, we can take advantage of the bias reduction from gradient boosting framework and the variance reduction from the weak learner, leading to a single small yet accurate model.

4.5 Scoring Efficiency

In this section, we study the scoring efficiency of various boosted ensembles.

Scoring Single Model. In this experiment, we build regression trees and decision tables on different d's (depth for regression trees and dimension for decision tables) and compare their scoring time. We only present results on "CalHousing" dataset due to space limitation, but we observe similar patterns on other datasets. Figure 9 illustrates the scoring time for regression tree and decision table as d increases. The scoring time is averaged over 1000 trials. We can see that for the same d (i.e., same number of tests to score), scoring decision table is much faster using the bit vector representation presented in Section 3.1 as it is more cache friendly. In addition, the scoring latency of decision table increases much slower than that of regression trees.

Model	Wine	CompAct	Pole	Bike	CalHousing	Mean
BRT$_l$	0.6328±0.0279	2.3322±0.3245	4.3816±0.1417	3.1260±0.1685	4.6020±0.0782	1.0035±0.0056
BRT$_d$	0.6308±0.0234	2.3399±0.3142	4.7898±0.2798	3.2174±0.2154	4.6576±0.0552	1.0305±0.0362
BDT	0.6290±0.0168	2.1383±0.0704	3.9809±0.2291	3.0139±0.4308	4.5304±0.0398	0.9591±0.0400
BDT$_{cbf}$	0.6283±0.0250	2.1520±0.0770	3.8516±0.1270	**2.9253±0.5048**	4.5073±0.0806	0.9474±0.0485
BDT$_{rbf}$	**0.6197±0.0246**	**2.1224±0.0700**	**3.8338±0.1174**	3.0190±0.5913	**4.4998±0.0493**	**0.9468±0.0476**
	(0.1097)	(0.0969)	**(0.0002)**	(0.2395)	**(0.0002)**	

Table 2: RMSE for regression datasets. Average normalized score on five datasets is shown in the last column, where the score is calculated as relative improvement over min(BRT$_l$, BRT$_d$). The last row shows p-value from t-test of BRT$_d$ and BDT$_{rbf}$.

Model	Magic	Letter	News	Bank	Physics	Mean
BRT$_l$	11.8297±0.3886	2.4000±0.2284	32.5135±0.8996	9.0661±0.1218	27.0120±0.2699	1.0068±0.0044
BRT$_d$	11.7875±0.3750	2.5250±0.2000	32.4034±0.8517	9.0907±0.1576	27.0920±0.2293	1.0176±0.0288
BDT	11.3985±0.1423	2.1200±0.2168	32.3076±0.6184	9.0156±0.2594	26.8480±0.3897	0.9726±0.0438
BDT$_{cbf}$	11.3933±0.3530	2.0850±0.3105	32.2238±0.5648	8.9624±0.2444	26.8480±0.3252	0.9679±0.0491
BDT$_{rbf}$	**11.3675±0.1793**	**2.0150±0.2690**	**32.1461±0.6896**	**8.9071±0.2189**	**26.7560±0.3502**	**0.9586±0.0615**
	(0.0092)	**(0.0003)**	(0.0710)	**(0.0136)**	**(0.0051)**	

Table 3: Error rate for classification datasets. Average normalized score on five datasets is shown in the last column, where the score is calculated as relative improvement over min(BRT$_l$, BRT$_d$). The last row shows p-value from t-test of BRT$_d$ and BDT$_{rbf}$.

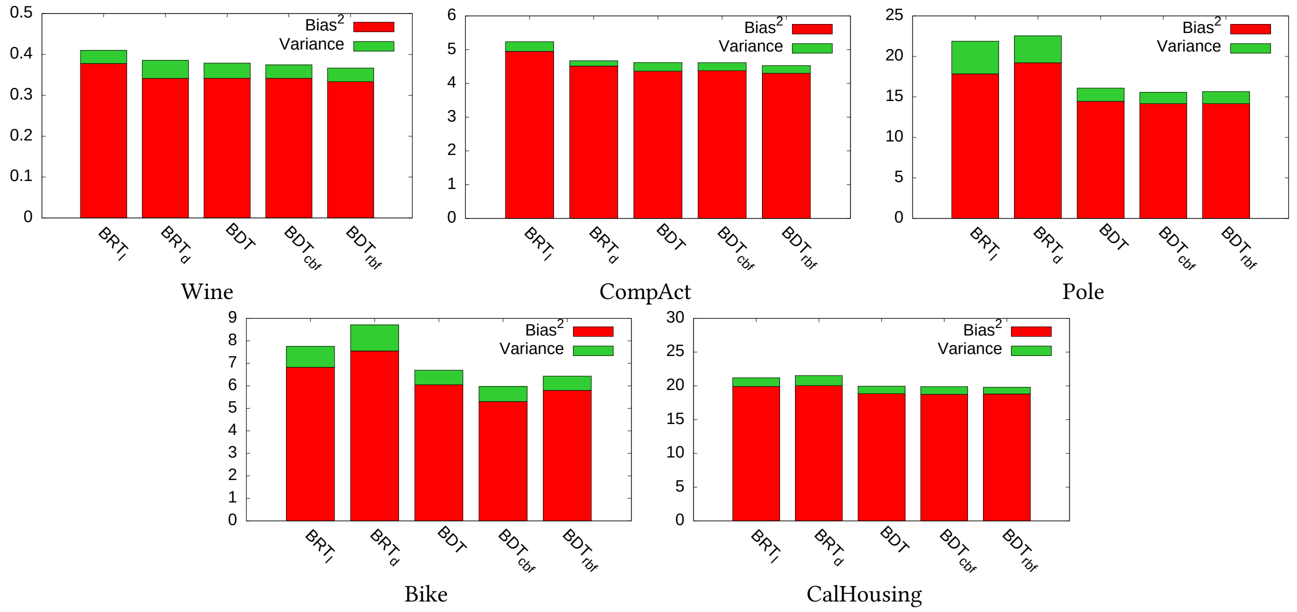


Figure 7: Bias-variance decomposition on regression datasets.

Scoring Ensemble. In real world applications, boosted ensembles are usually built to achieve the best accuracy within the constraint of scoring latency. With scoring decision table being faster, we can afford boosting more decision tables in an ensemble to achieve a higher level of accuracy. However, note that a smaller ensemble of decision table does not imply that it is less accurate. In this experiment, we compare the scoring efficiency for the most accurate BRT$_d$ and BDT$_{rbf}$ that we obtained in Section 4.3. Figure 9 demonstrates the averaged speedup from 100 trials on each dataset. We see that our algorithm of scoring boosted decision tables described in Section 3.3 is able to achieve 1.5x to 6x speedup compared to BRT$_d$, since our scoring of BDTs leverages the bitwise operation which makes it highly scalable to score a single decision table and we do not need to go through all the tests in a boosted decision table ensemble.

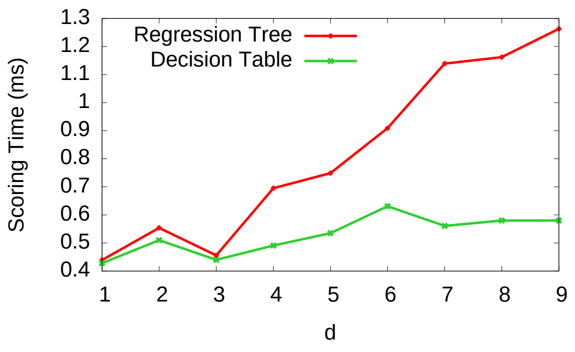
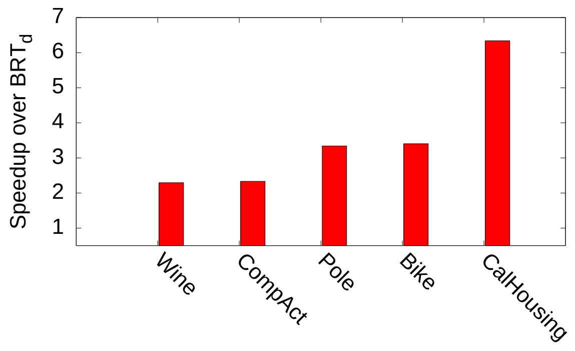
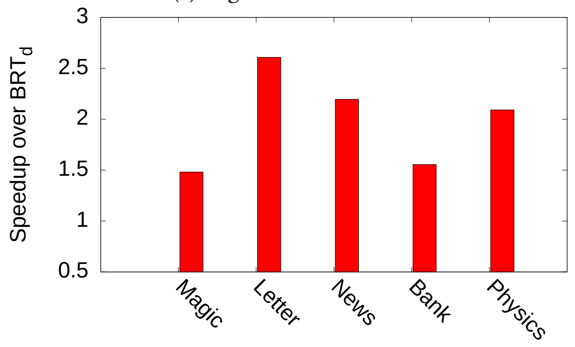


Figure 8: Scoring time on "CalHousing" dataset.



(a) Regression datasets.



(b) Classification datasets.

Figure 9: Speedup of BDT$_{rbf}$ over BRT$_d$.

5 EXPERIMENTS ON LINKEDIN FEEDS

LinkedIn dynamically delivers update activities from a user's interpersonal network to more than 400 million members in the personalized feed that ranks activities according their "relevance" to the user. In this section, we describe our offline and online experiments on boosted decision tables using LinkedIn news feed dataset.

5.1 Offline Evaluation

We collect a subset of the feed activities in March 2016. The dataset has around 40 million points with roughly 1500 features. We use a separate validation set to select parameters and report results on a held-out test set. We use `XGBoost`[7] to build BRT$_d$ for scalability.

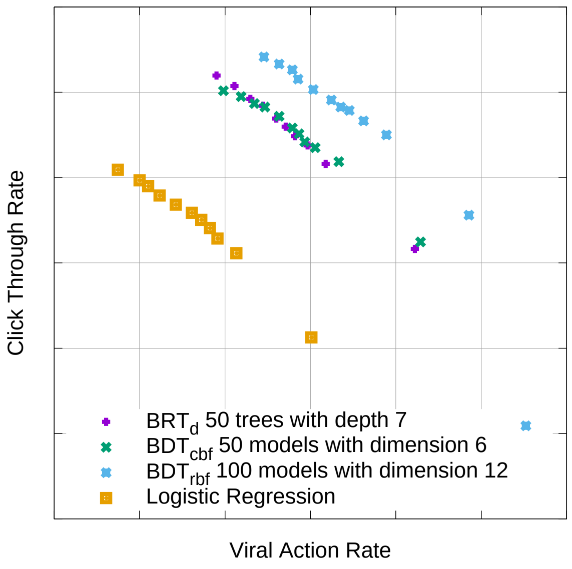


Figure 10: Offline results on LinkedIn news feed.

In this experiment, we study the tradeoff between click through rate (CTR) and viral action ("like," "share," "comment," etc.) rate. We compare logistic regression, BRT$_d$, BDT$_{cbf}$ and BDT$_{rbf}$. The features in the training set are "raw" features and those are the features used in the boosted ensembles. For logistic regression, we apply the feature transformation and feature interaction that are currently used in production. Given that our dataset is large, there is a limited number of weak models in a boosted ensemble that we can train within reasonable time. In addition, due to the online scoring latency constraint, we limit the size of BRT$_d$ to 50 trees with depth 7. However, higher offline accuracy of BRT$_d$ can be obtained by boosting more trees.

Figure 10 shows the results on our held-out test set. The resolution for each cell is 0.05×0.05.[8] We first observe that all boosted ensembles outperform the logistic regression model on both CTR and viral action rate, illustrating the advantage of boosted ensembles over handcrafted linear models. Second, we see that the BDT$_{cbf}$ using 50 decision tables with dimension 6 has similar predictive performance compared to the largest BRT$_d$ model that we can obtain. In our experiment, BDT$_{cbf}$ and BDT$_{rbf}$ result in similar predictive performance and therefore for this setting we only present BDT$_{cbf}$ here. Note that although predictive performance is similar on both BRT$_d$ and BDT$_{cbf}$, BDT$_{cbf}$ is smaller than BRT$_d$ (for using smaller d). The best model from our offline evaluation is BDT$_{rbf}$ using 100 decision tables with dimension 12. We can see that this model significantly outperforms other models in terms of predictive performance.

[7]https://github.com/dmlc/xgboost
[8]Actual numbers are omitted in accordance with LinkedIn's non-disclosure policy.

5.2 Online Evaluation

In many recommendation systems, such as LinkedIn news feed, input features are usually divided into groups, e.g., user features, engagement features, content features, interaction features, etc. Typically, the model needs to generate scores for a list of recommendation items for the same user, which suggests that the user features will stay the same when scoring his/her recommendation items. We note that further scoring efficiency can be achieved by pre-setting the bit vectors that index the predictions on user features, and only set relevant bits on other feature groups. Therefore, we only need to score user features once, which cannot be easily done when scoring boosted regression trees. With those optimization methods for scoring, we are able to run an online evaluation that compares boosted decision tables with the baseline production model, a highly engineered logistic regression, without increasing latency.

During offline evaluation we found that BDT$_{cbf}$ and BDT$_{rbf}$ led to similar predictive performance under the same setting and therefore we deployed BDT$_{cbf}$ using 50 decision tables with dimension 7 and BDT$_{cbf}$ using 100 decision tables with dimension 12 to LinkedIn news feed system. In the online experiment, each model served 2% of the members. We collected online experimental results for two weeks in July 2016. We found that both BDT models achieved similar performance; our best BDT model achieved a 4.7% lift on CTR and 6.3% lift on viral action rate, both were significant with p-value = 0. We deployed the model using 50 decision tables with dimension 7 to the production system since this model is much smaller.

6 RELATED WORK

Decision tables have been previously used for model interpretability. Kohavi and Sommerfield have proposed to use decision tables to better understand the data for business users [12]. Lou *et al.* have developed a fast method of detecting pairwise feature interactions for intelligibility using 2-dimensional decision tables, although not explicitly stated [15]. Decision table is also related to oblivious regression trees. An oblivious regression uses the same feature to split one the same level. Capannini *et. al* have proposed Oblivious λ-MART that boost oblivious regression trees for ranking problems [5]. The difference between our work and theirs lies in that we are interested in boosting (optimized) decision tables for accuracy and scoring efficiency.

Gradient boosted regression trees have been successful in many applications [4, 6, 7, 9, 14]. Typically regularization is necessary to improve the accuracy of boosted trees. It is known that gradient boosting is a low bias but high variance method [1] and bagging [2] can be used to wrap around gradient boosting to reduce the variance [10, 18]. In this work, we focus on controlling the model complexity by employing decision tables and therefore, our method is orthogonal to those previous work. Scoring gradient boosted regression trees is typically slow. Novel methods have been developed to significantly speed up the prediction running time via efficient representation of ensembles of binary regression trees [16]. However, those advanced methods usually require additional engineering efforts. Our approach of scoring boosted decision tables is intuitive and easy to implement.

7 CONCLUSION

We present gradient boosted decision tables (BDTs) in this paper. A d-dimensional decision table maps a sequence of d boolean tests to a real value in $\mathbb{R}$ and is equivalent to a full binary regression tree where all nodes on the same level share the same test. It serves as an excellent regularization method for gradient boosting. We propose novel algorithms to fit decision tables and our thorough empirical study suggests BDTs consistently outperform the standard gradient boosting using regression trees (BRTs) on classification and regression problems. We present a bias-variance analysis to show how different weak models influence the generalization error and we find that BDTs are low bias and low variance. In addition, we develop an efficient data structure to represent decision tables and propose a novel fast algorithm to improve the scoring efficiency of boosted ensemble of decision tables. We find empirically that our method of scoring BDTs is able to achieve 1.5x to 6x speedups over baseline methods. Our empirical results suggest gradient boosting with randomly backfitted decision tables distinguishes itself as the most accurate method. We have deployed our model to LinkedIn news feed system and demonstrate the effectiveness of the model through offline and online evaluation.

REFERENCES

[1] E. Bauer and R. Kohavi. 1999. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning* 36, 1 (1999), 105–139.
[2] L. Breiman. 1996. Bagging predictors. *Machine learning* 24, 2 (1996), 123–140.
[3] L. Breiman, J.H. Friedman, C. Stone, and R. Olshen. 1984. *Classification and regression trees.* CRC press.
[4] C.J.C Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11 (2010), 23–581.
[5] G. Capannini, C. Lucchese, F.M. Nardini, S. Orlando, R. Perego, and N. Tonellotto. 2016. Quality versus efficiency in document scoring with learning-to-rank models. *Information Processing & Management* 52, 6 (2016), 1161–1177.
[6] R. Caruana and A. Niculescu-Mizil. 2006. An empirical comparison of supervised learning algorithms. In *ICML.*
[7] J.H. Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of Statistics* 29 (2001), 1189–1232.
[8] J.H. Friedman. 2002. Stochastic gradient boosting. *Computational Statistics and Data Analysis* 38 (2002), 367–378.
[9] J.H. Friedman, T. Hastie, and R. Tibshirani. 2000. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Annals of Statistics* 28 (2000), 337–407.
[10] Y. Ganjisaffar, R. Caruana, and C.V. Lopes. 2011. Bagging gradient-boosted trees for high precision, low variance ranking models. In *SIGIR.*
[11] S. Geman, E. Bienenstock, and R. Doursat. 1992. Neural networks and the bias/variance dilemma. *Neural computation* 4, 1 (1992), 1–58.
[12] R. Kohavi and D. Sommerfield. 1998. Targeting Business Users with Decision Table Classifiers.. In *KDD.*
[13] P. Li. 2010. Robust logitboost and adaptive base class (abc) logitboost. In *UAI.*
[14] P. Li, C.J.C. Burges, and Q. Wu. 2007. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In *NIPS.*
[15] Y. Lou, R. Caruana, J. Gehrke, and G. Hooker. 2013. Accurate Intelligible Models with Pairwise Interactions. In *KDD.*
[16] C. Lucchese, F.M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. 2015. QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees. In *SIGIR.*
[17] R.K. Pace and R. Barry. 1997. Sparse spatial autoregressions. *Statistics & Probability Letters* 33, 3 (1997), 291–297.
[18] D.Y. Pavlov, A. Gorodilov, and C.A. Brunk. 2010. BagBoo: a scalable hybrid bagging-the-boosting model. In *CIKM.*
[19] S. Tyree, K.Q. Weinberger, K. Agrawal, and J. Paykin. 2011. Parallel boosted regression trees for web search ranking. In *WWW.*
[20] S.M. Weiss and N. Indurkhya. 1995. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research* 3 (1995), 383–403.