Efficiently Training Intelligible Models for Global Explanations

Yin Lou
Ant Group
yin.lou@antgroup.com

Yongliang Wang
Ant Group
yongliang.wyl@antgroup.com

Shiwei Liang
Ant Group
shiwei.lsw@antgroup.com

Yang Dong
Ant Group
doris.dy@antgroup.com

ABSTRACT

Generalized additive models (GAMs) are one of the popular methods of building intelligible models on classification and regression problems. Fitting the most accurate GAMs is usually done via gradient boosting with bagged shallow trees. However, such method can be expensive for large industrial applications. In this work, we aim to improve the training efficiency of GAM. To this end, we propose to use subsample aggregating (subagging) in place of bootstrap aggregating (bagging). Our key observation is that subsamples of reasonable size (e.g., 60% of the training set) usually overlap. Such property allows us to explore the computation ordering inside a subagged ensemble and we present a novel algorithm to speed up the computation of subagged ensemble with no loss of accuracy. Our experimental results on public datasets demonstrate that our proposed method can achieve up to 3.7x speedup over bagged ensembles with comparable accuracy. Finally, we demonstrate our methodology of finding global explanations on a real application at Alipay. We have developed several strategies from the findings of those explanations and found those strategies achieved significant lift on key metrics through online experiments.

CCS CONCEPTS

• **Computing methodologies** → **Ensemble methods**.

KEYWORDS

intelligibility; classification; regression; additive models

ACM Reference Format:
Yin Lou, Yongliang Wang, Shiwei Liang, and Yang Dong. 2020. Efficiently Training Intelligible Models for Global Explanations. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3340531.3412702

1 INTRODUCTION

In many applications, what is learned is just as important as the accuracy of predictions. Generalized additive models (GAMs) [11, 22] are one of the popular methods of building intelligible models on classification and regression problems. Typically, GAMs can be written as,

$$g(E[y]) = \beta_0 + \sum_j f_j(x_j), \tag{1}$$

where g is the link function and f_j is called shape function. For identifiability, f_j's are usually centered, i.e., $E[f_j] = 0$.

Since GAM only has univariate components, users can easily visualize those one-dimensional additive components. Hence, it is a completely white-box model and provides global additive explanations. GAM has been demonstrated to be useful and accurate in many mission-critical applications, such as healthcare [5] and bias detection in recidivism prediction [19].

One of the major GAM fitting algorithms employs a combination of bootstrap aggregating (bagging) [3] and gradient boosting [8] as its optimization method; GAM runs gradient boosting with bagged shallow trees that cycle one-at-a-time through its components. Previous research shows that such combination yields the most accurate model on classification and regression problems [14, 24]. Despite the high predictive performance, such ensemble combination makes it relatively expensive to train GAMs, especially for industry-scale applications.

Hence in this work we aim to improve the training efficiency of GAM. We propose to use subsample aggregating (subagging) instead of bootstrap aggregating when building trees for each component. Our key observation is that subsamples of reasonable size (e.g., 60% of the training set) usually overlap.[1] We explore such overlapping property to carefully order the computation inside a subagged ensemble so that consecutive samples share as many data points as possible. This makes it possible to transfer statistics of model construction from previous sample to the next one with a much smaller cost compared to building model from scratch. Our experiments show that we can achieve up to 3.7x speedup compared to bagged ensembles with comparable accuracy.

Finally, we demonstrate our methodology of finding global explanations for real applications at Alipay; predicting whether a user wants to renew his/her financial product. When the financial product is about to mature, Alipay's automated financial assistant will first ask the user whether to renew the financial product. If user decides not to renew now, the system will try to keep the user by asking whether he/she wants to find lower-risk products within the same category or try out other assets. Understanding patterns of customer behaviors is important for improving key metrics, such as click through rate (CTR). In this application, we want to find what characteristics of our customer are leaning towards a particular action in a dialogue flow, and how much they contribute, so that we can develop strategies that promote interaction rate. We have deployed several strategies from the findings of those explanations and found those strategies achieved significant lift on key metrics through online experiments.

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

- We propose to use subagged shallow trees as base learners in GAMs.

[1] Recall that there are about 63% of unique points inside a bootstrap sample.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412702

- We present a novel algorithm that speeds up the computation of subagged ensemble.
- We provide experiments to demonstrate the efficiency and accuracy of our proposed methods.
- We demonstrate our methodology of finding global explanations on a real application at Alipay.

The rest of this paper is organized as follows. Section 2 presents preliminaries. Our approach is discussed in Section 3. Experimental results on public datasets are presented in Section 4. We present online experimental results on a real application at Alipay in Section 5. Related work is presented in Section 6 and we conclude the paper in Section 7.

2 PRELIMINARIES

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ 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. Additionally, each data point may have a weight w_i. 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 $\{1, 2, ..., n\}$.

2.1 GAM Fitting via Cyclic Gradient Boosting

One of the most popular GAM fitting algorithms is cyclic gradient boosting with bagged shallow trees [14]. It follows a variant of multiple additive regression trees (MART) algorithm that aims to find a function $F = \sum_j f_j$ to minimize the following objective function,

$$\mathcal{L}(y, F(\boldsymbol{x})), \tag{2}$$

where $\mathcal{L}(\cdot, \cdot)$ is a non-negative convex loss function. When $\mathcal{L}$ is squared loss, our problem becomes a regression problem and if $\mathcal{L}$ is logistic loss, we are dealing with a binary classification problem.

Algorithm 1 and Algorithm 2 summarize the optimization procedure for regression and classification problems, respectively. We first create bootstrap samples of size B (Line 2); $\mathcal{S}_b$ is a multi-set of selected indices in $\mathcal{D}$. For each boosting pass, we cycle through all the features (Line 3-4) and construct the (pseudo) residual r_i's w.r.t. feature j (Line 5), the negative gradients in functional space [8]. For regression problems, this is equivalent to $r_i = y_i - F(\boldsymbol{x}_i)$. For classification problems, the pseudo residual is $r_i = y_i - p_i$ and $p_i = 1/(1 + \exp(-F(\boldsymbol{x}_i))$. We then run bootstrap aggregating (bagging) and fit trees on each bootstrap sample to those residuals (Line 6-7). For classification problems, we further take a Newton-Raphson step using a diagonal approximation to the Hession to update the prediction γ_l for leaf l (Line 8 in Algorithm 2),

$$\gamma_l = \frac{\sum_{\boldsymbol{x}_i \in l} r_i}{\sum_{\boldsymbol{x}_i \in l} |r_i|(1 - |r_i|)}. \tag{3}$$

Finally a learning rate ν is applied on the bagged ensemble and we update the shape function f_j (Line 8 in Algorithm 1 and Line 9 in Algorithm 2).

For identifiability, each f_j is usually centered (i.e., $E[f_j] = 0$) and an intercept β_0 is added accordingly to the model as shown in Equation (1).

Algorithm 1 GAM for regression

1: $f_j \leftarrow 0$, for $j \in [p]$
2: $\mathcal{S}_b \leftarrow$ create a bootstrap sample from $[N]$, for $b \in [B]$
3: **for** $m = 1$ to M **do**
4: **for** $j = 1$ to p **do**
5: $r_i \leftarrow y_i - F(\boldsymbol{x}_i)$, for $i \in [N]$
6: **for** $b = 1$ to B **do**
7: $T_{jb} \leftarrow$ a regression tree of L leaves trained on $\{(x_{ij}, r_i) | i \in \mathcal{S}_b\}$
8: $f_j \leftarrow f_j + \nu \frac{1}{B} \sum_{b=1}^B T_{jb}$

Algorithm 2 GAM for binary classification (MART)

1: $f_j \leftarrow 0$, for $j \in [p]$
2: $\mathcal{S}_b \leftarrow$ create a bootstrap sample from $[N]$, for $b \in [B]$
3: **for** $m = 1$ to M **do**
4: **for** $j = 1$ to p **do**
5: $r_i \leftarrow y_i - p_i$, for $i \in [N]$
6: **for** $b = 1$ to B **do**
7: $T_{jb} \leftarrow$ a regression tree of L leaves trained on $\{(x_{ij}, r_i) | i \in \mathcal{S}_b\}$
8: Update T_{jb} using Newton-Raphson step as shown in (3)
9: $f_j \leftarrow f_j + \nu \frac{1}{B} \sum_{b=1}^B T_{jb}$

Algorithm 3 Histogram Construction (MART)

1: **for** $i = 1$ to N **do**
2: $v \leftarrow x_{ij}$
3: $H^{(j)}(v).r \leftarrow H^{(j)}(v).r + r_i w_i$
4: $H^{(j)}(v).h \leftarrow H^{(j)}(v).h + |r_i|(1 - |r_i|)w_i$
5: $H^{(j)}(v).w \leftarrow H^{(j)}(v).w + w_i$

2.2 Line Cutting

For practical implementation, standard regression tree may not be the optimal choice. As suggested by Lou et al. [15], building such single-feature regression tree is equivalent to cutting on a line, which can be made more efficient. We also discretize each feature into at most 256 equi-frequency bins to avoid feature sorting. Such discretization is usually enough to achieve high scalability without hurting accuracy [15].

The essential statistics of line cutting (building single-feature regression tree) are histograms. Let $H^{(j)}(v).r$ and $H^{(j)}(v).w$ denote the weighted sum of responses (first order information) and weights when $x_j = v$, respectively. To facilitate Newton-Raphson step in Algorithm 2, we use $H^{(j)}(v).h$ to denote weighted sum of $|r_i|(1-|r_i|)$ for $x_j = v$ (second order information). Algorithm 3 summarizes the construction of a histogram for MART algorithm. Note that on Line 3, we compute the weighted sum of responses.

We then make greedy cuts to minimize the variance in current branch (same as standard regression tree). Figure 1 illustrates an example of line cutting on a histogram of size 5. To see why line cutting is more efficient than standard regression tree, we note that the histograms only need to be constructed once (as opposed to re-constructing histograms each time after a split in standard regression tree). Thus, histogram is the only statistics that we need to

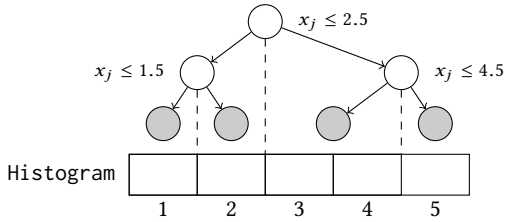


Figure 1: Line cutting on histograms.

build a regression model. For classification problems, note that the nominator in Equation (3) can be easily computed by aggregating $H^{(j)}(v).r$, and the denominator is the aggregation of $H^{(j)}(v).h$.

It is easy to see that when histograms are complete, the complexity of line cutting is $O(|H|)$, where $|H|$ is the size of the histogram.

3 OUR APPROACH

In this section, we present our approach to improve the training efficiency of GAM. We start with subagging in Section 3.1. In Section 3.2, we propose a novel algorithm to speed up the computation of subagged ensemble. For classification problems, we present a variant of LogitBoost algorithm in Section 3.3.

3.1 Subsample Aggregating

In this paper, we introduce subsample aggregating (subagging) in place of bootstrap aggregating (bagging) when building shape functions in GAMs. Unlike bootstrap sampling, which is random sampling *with* replacement, subsample is random sampling *without* replacement.

In this work, we use parameter α to denote the subsampling ratio; we sample αN points for each subsample, e.g., $\alpha = 0.6$ means we subsample 60% of the data. We then build regression models on those subsamples and form an ensemble accordingly. As we will see in Section 4, using subagging in place of bagging yields models with comparable accuracy.

3.2 Fast Subagging

We note that subsamples of reasonable size (e.g., 60% of the training set) usually overlap. We can explore such overlapping property to carefully order the computation inside a subagged ensemble so that consecutive samples share as many data points as possible. Recall that histogram is the only statistics to build regression model in GAM fitting, such overlapping property makes it possible to transfer the histogram from previous sample to the next one with a much smaller cost compared to building model from scratch. We start by defining the cost from one sample to another.

3.2.1 Cost between Two Subsamples. Without loss of generality, let us consider two subsamples $\mathcal{S}_i$ and $\mathcal{S}_j$. Denoting the histogram on $\mathcal{S}_i$ as H_i and histogram on $\mathcal{S}_j$ as H_j, the cost of transferring H_i to H_j can be defined as $|\mathcal{S}_i \cup \mathcal{S}_j| - 2|\mathcal{S}_i \cap \mathcal{S}_j|$. This cost can be further decomposed into two parts; $\Delta_{ij}^+ = \mathcal{S}_j - \mathcal{S}_i$ (the points to add to H_i) and $\Delta_{ij}^- = \mathcal{S}_i - \mathcal{S}_j$ (the points to remove from H_i). When the context is clear, we use $\Delta_{ij} = (\Delta_{ij}^+, \Delta_{ij}^-)$ to denote the additional data points to scan in the dataset, and $|\Delta_{ij}| = |\Delta_{ij}^+| + |\Delta_{ij}^-|$ to denote the cost between two subsamples. We note that, because of this sampling without replacement property, we can achieve a relatively low cost to transfer histogram from one sample to another; the same is not easy for bootstrap samples.

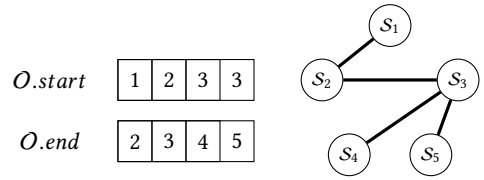


Figure 2: Computation ordering for a set of subsamples.

Algorithm 4 Computing Subagging Ordering

1: $\mathcal{S}_b \leftarrow$ create a subsample from $[N]$, for $b \in [B]$
2: **for** $i = 1$ to $B - 1$ **do**
3: **for** $j = i + 1$ to B **do**
4: $d_{ij} \leftarrow |\mathcal{S}_i \cup \mathcal{S}_j| - 2|\mathcal{S}_i \cap \mathcal{S}_j|$
5: $MST \leftarrow$ minimum spanning tree on $G = (\{\mathcal{S}_b\}, \{d_{ij}\})$
6: $\mathcal{S}_v \leftarrow$ a random element in $\{\mathcal{S}_b\}$
7: $O \leftarrow BFS(\mathcal{S}_v, MST)$

3.2.2 Computation Ordering. With the cost between two samples, it is now possible to carefully order the computation inside a subagged ensemble to minimize the overall cost.

Algorithm 4 summarizes the procedure of generating the computation ordering. We first model the subsamples inside an ensemble (typically of size 50 or 100) as nodes in a graph, with edges representing the cost to transfer from one sample to another (Line 1-4). Such graph is a clique and we observe that the cost minimization problem is equivalent to finding minimum spanning tree in this clique (Line 5). Once we have computed the minimum spanning tree, we randomly pick a node (subsample) as the starting point and run breadth-first search (BFS) to generate the computation ordering O (Line 6-7). We use $O.start$ and $O.end$ to denote the vectors for starting sample and target samples, respectively.

Figure 2 shows an example of 5 subsamples and their computation ordering O. We start with sample $\mathcal{S}_1$, build the histogram H_1 and use Δ_{12} to transfer H_1 into H_2. Once we have H_2, we can use Δ_{23} to compute histogram H_3. Finally, H_3 will be the starting point to compute H_4 and H_5. Note that we only do a full scan once when generating the histogram on $\mathcal{S}_1$. For the remaining histograms, we only need to scan data points as indexed in Δ_{ij} and therefore avoid full scans of the dataset.

To demonstrate the effectiveness of such computation ordering, we run a simulation with $\alpha \in \{0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80\}$. For each α we create 100 subsamples, Figure 3 shows the average ratio of data points to be scanned. As we can see, with such histogram transferring, we only need to scan less than 50% of the data points to build a new histogram on the next sample. We do not consider α lower than 0.5 since it causes us to scan more points for constructing new histograms. As we will see in Section 4, this range of α is enough to produce accurate models while achieving high training efficiency.

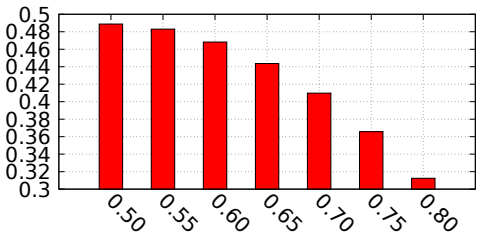


Figure 3: Ratio of points to scan vs. α.

Algorithm 5 GAM for binary classification (Logit)

1: $f_j \leftarrow 0$, for $j \in [p]$
2: $\mathcal{S}_b \leftarrow$ create a bootstrap sample or subsample from $[N]$, for $b \in [B]$
3: **for** $m = 1$ to M **do**
4: **for** $j = 1$ to p **do**
5: $r_i \leftarrow y_i - p_i$, for $i \in [N]$
6: $w_i \leftarrow p_i(1 - p_i)$, for $i \in [N]$
7: **for** $b = 1$ to B **do**
8: $T_{jb} \leftarrow$ a robust regression tree of L leaves trained on $\{(x_{ij}, r_i)|i \in \mathcal{S}_b\}$ with weight $\{w_i|i \in \mathcal{S}_b\}$
9: $f_j \leftarrow f_j + \nu \frac{1}{B} \sum_{b=1}^{B} T_{jb}$

Algorithm 6 Histogram Construction (Logit)

1: **for** $i = 1$ to N **do**
2: $v \leftarrow x_{ij}$
3: $H^{(j)}(v).r \leftarrow H^{(j)}(v).r + r_i$
4: $H^{(j)}(v).w \leftarrow H^{(j)}(v).w + w_i$

3.3 LogitBoost

For classification problems, we recognize that LogitBoost [9] usually achieves more accurate model than MART [13]. The main difference is that MART uses first order information to grow trees and second order information to compute leaf values whereas LogitBoost uses second order information to do both. Therefore, we propose a variant of LogitBoost for GAM fitting on classification problems.

Algorithm 5 summarizes our procedure. The main difference between Algorithm 2 and 5 is that 1) we train a robust regression tree and 2) we get rid of the Newton-Raphson step in Algorithm 2. Robust regression trees is numerically stable implementation in LogitBoost [13]. In our implementation, we use a robust line cutting procedure for better efficiency and histogram is still the only statistics that we need to build a regression model. Algorithm 6 describes our procedure of building a histogram in robust line cutting. Note that in Line 3, we do *not* use weighted sum of responses. This is because we already set the weight for each point as $p_i(1 - p_i)$ (Line 6 in Algorithm 5). When computing leaf values, we will get the values computed from second order information. We will see in Section 4 that by getting rid of the Newton-Raphson step, we can further improve the training efficiency.

Dataset	Size	Attributes	%Pos
CompAct	8192	22	-
Pole	15000	49	-
Bike	17379	15	-
CalHousing	20640	9	-
Superconductor	21263	82	-
Magic	19020	11	64.84
Letter	20000	17	49.70
Bank	45221	13	11.70
Adult	48842	15	76.07
Credit	150000	11	6.68

Table 1: Datasets.

4 EXPERIMENTS ON PUBLIC DATASETS

In this section we report experimental results on public classification and regression datasets. Our approach is implemented in MLTK.[2]

4.1 Datasets

Table 1 summarizes the ten datasets used in our experiments. Five are regression problems: "CompAct" is from the Delve repository and describes the state of multiuser computers.[3] "Pole" describes a telecommunication problem [20]. The "Bike" datasets is from the UCI repository.[4] "CalHousing" describes how housing prices depend on census variables [17] . "Superconductor" is to predict the superconducting critical temperature [10]. The "Magic", "Letter", "Bank" and "Adult" are binary classification problems: all of them are from the UCI repository. We convert "Letter" into binary classification using the same method in [6]. "Credit" is a dataset for predicting probability that someone will experience financial distress in the next two years.[5] Note that we choose both balanced and unbalanced classification datasets to demonstrate the effectiveness of our method.

4.2 Model Accuracy

In this section, we report experimental results of model accuracy on our datasets. We use the implementation in MLTK as our baseline for GAM fitting algorithm (MART-Bagging). For this set of experiments, we consider subsampling ratio $\alpha \in \{0.50, 0.55, ..., 0.75, 0.80\}$ and we fix the learning rate $\nu = 0.01$. Following [15], we discretize all continuous features into at most 256 bins using equi-frequency discretization and fix the number of leaves for each tree as 3. In all our experiments, we set the bagging or subagging size to be 100. We randomly partition the dataset into training (64% of the data), validation (16% of the data) and test sets (20% of the data). For regression problems we report root mean squared error (RMSE) and for classification problems we report area under ROC curve (AUC). We set the maximum number of iterations to 10,000 and do model selection using validation set. All experiments are repeated 5 times and we report mean and standard deviation of the evaluation results on test sets.

[2] https://github.com/yinlou/mltk
[3] http://www.cs.toronto.edu/~delve/data/datasets.html
[4] http://archive.ics.uci.edu/ml/
[5] https://www.kaggle.com/c/GiveMeSomeCredit/data

Model	CompAct	Pole	Bike	CalHousing	Superconductor
MART-Bagging	2.6635±0.1124	**21.4300±0.1734**	1.0004±0.0121	5.7372±0.1182	11.2624±0.1638
MART-Subagging-0.50	2.6648±0.1131	21.4564±0.1645	1.0004±0.0122	5.7335±0.1144	11.2638±0.1731
MART-Subagging-0.55	2.6684±0.1112	21.4511±0.1712	**1.0003±0.0119**	5.7313±0.1210	11.2711±0.1676
MART-Subagging-0.60	2.6641±0.3750	21.4524±0.1722	1.0005±0.0121	5.7283±0.1170	11.2536±0.1676
MART-Subagging-0.65	**2.6634±0.1161**	21.4507±0.1675	1.0006±0.0119	5.7291±0.1163	**11.2410±0.1572**
MART-Subagging-0.70	2.6648±0.1174	21.4440±0.1709	1.0004±0.0121	5.7258±0.1136	11.2511±0.1491
MART-Subagging-0.75	2.6687±0.1165	21.4457±0.1757	1.0004±0.0122	**5.7239±0.1125**	11.2537±0.1498
MART-Subagging-0.80	2.6678±0.1201	21.4385±0.1823	1.0003±0.0120	5.7252±0.1140	11.2667±0.1522

Table 2: RMSE for regression datasets. Lower is better.

Model	Magic	Letter	Bank	Adult	Credit
MART-Bagging	0.9039±0.0066	0.9108±0.0056	0.9156±0.0030	0.9278±0.0022	0.8620±0.0032
MART-Subagging-0.50	0.9047±0.0063	0.9110±0.0058	0.9159±0.0032	0.9281±0.0022	0.8621±0.0032
MART-Subagging-0.55	0.9048±0.0066	0.9111±0.0058	0.9160±0.0032	0.9282±0.0023	0.8621±0.0032
MART-Subagging-0.60	0.9047±0.0066	0.9110±0.0057	0.9159±0.0032	0.9281±0.0023	0.8620±0.0032
MART-Subagging-0.65	0.9044±0.0067	0.9111±0.0059	0.9158±0.0029	0.9282±0.0023	0.8620±0.0032
MART-Subagging-0.70	0.9045±0.0066	0.9112±0.0058	0.9158±0.0031	0.9282±0.0024	0.8620±0.0033
MART-Subagging-0.75	0.9046±0.0065	0.9112±0.0058	0.9158±0.0031	0.9282±0.0024	0.8619±0.0032
MART-Subagging-0.80	0.9044±0.0065	**0.9113±0.0057**	0.9158±0.0031	0.9283±0.0023	0.8619±0.0032
Logit-Bagging	0.9045±0.0066	0.9112±0.0057	0.9159±0.0029	0.9279±0.0024	0.8620±0.0033
Logit-Subagging-0.50	**0.9050±0.0063**	0.9112±0.0059	0.9161±0.0031	0.9280±0.0024	**0.8621±0.0033**
Logit-Subagging-0.55	0.9050±0.0066	0.9113±0.0058	**0.9161±0.0031**	0.9281±0.0023	0.8620±0.0033
Logit-Subagging-0.60	0.9048±0.0066	0.9112±0.0059	0.9160±0.0030	0.9281±0.0024	0.8621±0.0034
Logit-Subagging-0.65	0.9047±0.0069	0.9112±0.0059	0.9159±0.0030	0.9281±0.0023	0.8621±0.0033
Logit-Subagging-0.70	0.9045±0.0068	0.9112±0.0058	0.9159±0.0030	0.9278±0.0022	0.8620±0.0034
Logit-Subagging-0.75	0.9048±0.0066	0.9111±0.0059	0.9159±0.0030	0.9281±0.0023	0.8619±0.0033
Logit-Subagging-0.80	0.9045±0.0066	0.9112±0.0058	0.9158±0.0030	**0.9283±0.0023**	0.8620±0.0033

Table 3: AUC for classification datasets. Higher is better.

Table 2 and Table 3 show the results on regression and classification problems, respectively. We first observe that with subagging, we achieve comparable accuracy of MART-Bagging. In addition, it is interesting to see that GAM fitting algorithms based on LogitBoost also achieve comparable accuracy of MART-Bagging. This observation is different from standard gradient boosting, where LogitBoost demonstrates superior accuracy over MART [13]. This is probably because of the GAM training method; we only update one single feature at a time. Nevertheless, as we will see in the next section that by getting rid of Newton-Raphson step in MART, our variant of LogitBoost algorithm further improves the training efficiency.

4.3 Training Efficiency

In this section, we report experimental results of training efficiency on our datasets. Since we only need to generate the computation ordering once before we enter the gradient boosting stage, this step is usually negligible for the whole training process and we compare the end-to-end training efficiency.

We fix the number of iterations in this set of experiments and running time of subagging is normalized by the running time of MART-Bagging, since MART-Bagging is our baseline method. Figure 4 illustrates the running time ratio compared with MART-Bagging for regression and classification problems, respectively. We see that we can save about 30-40% of time when training subagged tree ensembles for regression problems and we save up to 70% of time when training subagged trees in MART for classification problems. This is because we carefully order the computation inside a subagging ensemble so that key statistics can be transferred from one sample to another in a much lower cost compared. Recall that there are about 63% of unique points inside a bootstrap sample. Even for subsamples of similar size (e.g., $\alpha = 0.6$ or 0.65), we still achieve about 1.67x speedup. We also observe that, for LogitBoost, we can further improve the training efficiency, since we get rid of the need to cumulate second order information and the additional Netwon-Raphson step.

To summarize, with MART-Subagging, we can achieve up to 3.7x speedup, and with Logit-Subagging, we can achieve up to 7x speedup. In real applications, we recommend fixing α to 0.65, which corresponds to an average of 40% saving for regression problems and 70% saving for classification problems.

5 EXPERIMENTS ON ALIPAY'S DATASETS

Over the past two years since it was launched, Alipay's automated online financial assistant has been providing full lifecycle financial services for over 1 million Alipay users. In this work, we focus on the case where a user's financial product is about to mature.

There are two major entries in Alipay's app that can trigger the dialogue flow of the chatbot for this application, as illustrated in Figure 5. The first entry is on total asset page. The message in the

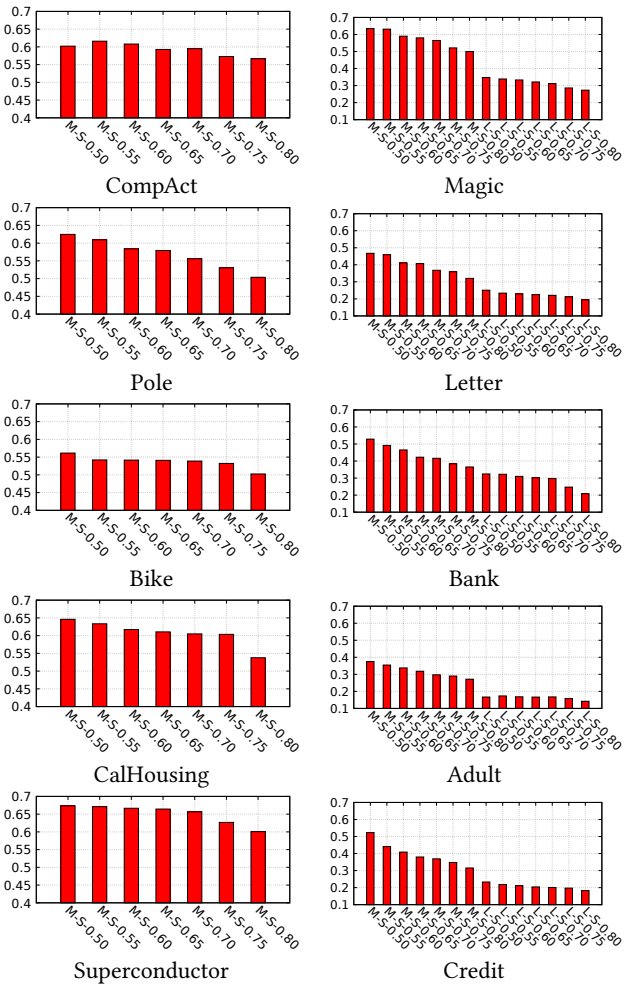
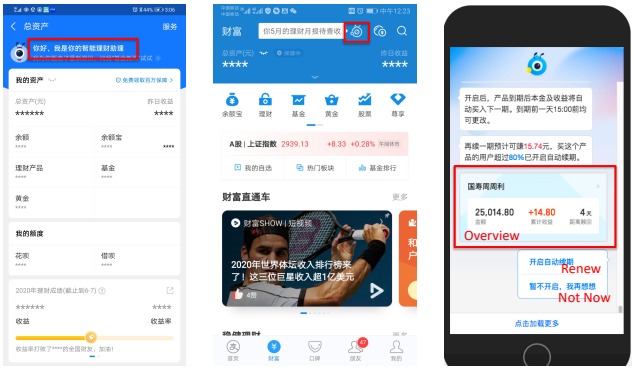


Figure 4: Training efficiency on regression and classification problems. Running time of each method is normalized by that of MART-Bagging.

Type	No.	Example
Portfolio feat.	12	# of mutual funds
Page-level feat.	304	# of clicks on news pages in 3 days
Region-level feat.	104	# of clicks on account positions in 15 days
Gain/Loss feat.	5	Total gain of mutual funds in 7 days
User trading behavior feat.	38	# of trades in 30 days
Market-level feat.	37	# of positive events in 7 days
Promotional feat.	16	Whether a coupon was redeemed
User profile feat.	121	Education level

Table 4: Features for Alipay's Datasets.

red box of Figure 5(a) will notify the user about close-to-mature financial product. There is no message for entry 2 as shown in Figure 5(b). Whenever the user clicks the box area in these two entries, he/she will be redirected to enter a dialogue flow. Figure 5(c) shows an example of a dialogue and Figure 6 illustrates the complete flow that a user might go through via the chatbot. "Overview" shows the details of the close-to-mature financial product. "Renew" and



(a) Entry 1 (b) Entry 2 (c) Dialogue Flow

Figure 5: Product Overview. Both entries can trigger the dialogue flow on the right.

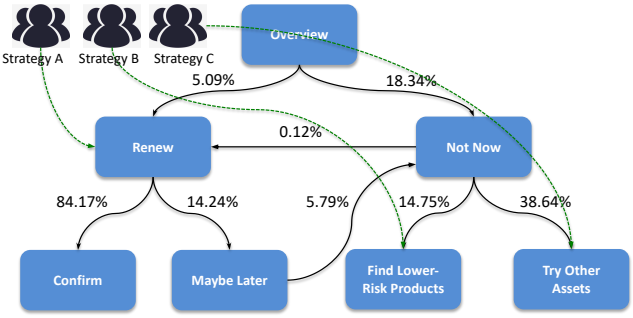


Figure 6: Illustration of financial product renewal flow.

"Not Now" are the two choices for the user to select. If the user decided not to renew now, the system would try to keep the user by asking whether he/she wants to find lower-risk products within the same category or try out other assets. Empirical transition probabilities are shown on the edges in Figure 6. Note that at any stage a user might drop, so the outgoing edges may not necessarily sum to 1.

5.1 Global Explanations

To provide personalized experience interacting with the chatbot, we do not want users to always start with "Overview", as it might be tedious for users with particular needs to go through the whole flow from the beginning. Hence, our goal in this work is to use GAM to find global explanations to gain insights from the data, so that we can provide different users with different entries in the dialogue. To this end, we have collected logs of users interacting with our system from 01/08/2020 to 02/02/2020 (users always started from the beginning in this period).[6] Table 4 shows the different types of our 637 features with examples.

[6]The authors declare that 1) the data set does not contain any Personal Identifiable Information (PII), 2) the data set is desensitized and encrypted, 3) adequate data protection was carried out during the experiment to prevent the risk of data copy leakage, and the data set was destroyed after the experiment, and 4) the data set is only used for academic research, it does not represent any real business situation.

Table 5 lists 4 scenarios that we are interested in for this particular application. We apply our fast GAM training algorithm (LogitBoost-Subagging-0.65) to those 4 scenarios to find global explanations with maximum 100 iterations. Hyperparamers are tuned on validation set. We also compare the accuracy of GAMs with XGBoost [7]. We tune the tree depth from 2 to 10 on validation set. All experiments are repeated 5 times. Table 5 demonstrates the AUC on the test sets for our fast GAM and XGBoost. We see that the accuracy of GAM is comparable to that of XGBoost, while the intelligibility of GAM allows us to find global explanations of the model.

Figure 7 shows some selected shape functions for those scenarios.[7] On scenario 1, we see that women is more likely to click "Renew". On scenario 2, we see that users who have fewer clicks on holdings page are more likely to click "Renew"; this is probably because for such users, they belong to set-and-forget category, who probably care more on long-term capital gains. On the 3rd scenario, there is a strong positive correlation between account balance and clicking "Find Lower-Risk Products". This is reasonable since for users with large account balance, they are more risk averse. On the last scenario, we see that users who frequently visit information pages of different mutual funds are more likely to click "Try Other Assets".

5.2 Online Evaluation

Based on the findings in last section, we have developed 3 strategies as follows. Note that each strategy will only come into play if a user has already gone through part of the dialogue flow but left with no action, and each strategy only affects the starting point of the dialogue flow when users come back and are redirected from the two entries as shown in Figure 5.

- Strategy A: For woman users or users who have fewer clicks on account position page , skip the "Overview" stage and directly ask them whether to renew the product. On Entry 1, we will also change the displayed message to "Do you want to renew your financial product?".
- Strategy B: For users with high account balance, the dialogue will start directly with "Find Lower-Risk Products". Entry 1 will display "Let us find lower-risk financial products for you".
- Strategy C: For users who frequently visit information pages of different mutual funds, the dialogue will jump to "Try Other Assets". Entry 1's message will be "Do you want to find other assets?".

It is possible that different strategies might overlap, and we set strategy A with highest priority and strategy B with lowest priority. We have deployed these strategies online and conducted A/B testing from 03/28/2020 to 04/03/2020. All experiments except CTR on entry 2 are statistically significant with p-value < 0.00001. Figure 8(a) shows the CTR on Entry 1. By employing personalized messages on Entry 1, the average lift is about 19%. Figure 8(b) illustrates (1 - bounce rate) for dialogue flow triggered by Entry 1; the average lift is about 9%. Bounce rate (BR) is a measure of users who enter and leave rather than continuing interacting with our system. We see that by employing personalized entry and dialogue flow, we increase the retention rate of user interacting with our system. Figure 9(a) shows the CTR on Entry 2. Since there is no message showing on these two versions, we see little difference on CTR as expected. Figure 9(b) illustrates (1 - bounce rate) for dialogue flow triggered by Entry 2. We see a lift of about 25%, suggesting the effectiveness of our strategies. These strategies are now live in production.

[7] Some features are normalized so as not to disclose the actual range.

6 RELATED WORK

6.1 Fitting Generalized Additive Models

Tan et al. compared tree ensembles with other local and global explanation methods and argued that global additive explanations provide a more complete view than feature attribution [18]. GAM was employed as their global additive explanation method. There are a variety of choices to train GAMs. Shape functions in GAMs can be represented by different functions, including splines [21], regression trees, or tree ensembles [2]. There are also two popular methods of fitting GAMs (independent of shape functions): Backfitting [11] and gradient boosting [8]. Previous research showed that gradient boosting with ensembles of shallow regression trees is the most accurate method among a number of alternatives [14]. Recently researchers are trying neural nets as shape functions [1, 23]. Our work focus on tree ensembles as shape functions with the goal of improving training efficiency.

6.2 Subsample Aggregating

Subsample aggregating (subagging) employs subsampling instead of bootstrap sampling for the aggregation. The complete subagging is aggregated by enumerating all possible samples of size M ($M < N$). When $M = N/2$, it is called half subagging. Buja and Stuetzle showed that when the estimator is a U-statistic [12], half subagging is exactly equivalent to bagging [4]. They also observed that half subagging yields very similar empirical results to bagging when the estimator is decision tree.

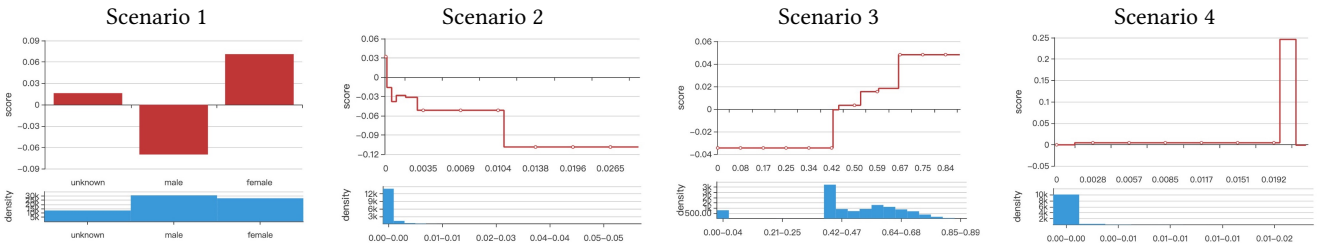
However, building $\binom{N}{M}$ models is computationally infeasible and in practice one usually only aggregates B subsamples. Mentch and Hooker demonstrated such method is asymptotically normal and they explored such property to quantify uncertainty in random forests [16].

7 CONCLUSION

In this work, we propose to use subsample aggregating (subagging) in place of bagging in the GAM fitting algorithm. We show empirically that using subagged shallow trees can lead to models with comparable accuracy. In addition, we observe that subsamples of reasonable size usually overlap. We explore such property and present a novel algorithm to speed up the computation of subagged ensemble with no loss of accuracy. For classification problems, we propose a variant of LogitBoost algorithm that can further improve the training efficiency. Our extensive experiments on public datasets demonstrate that our proposed method can achieve up to 3.7x speedup over bagged ensembles comparable accuracy. Finally, we demonstrate our methodology of finding global explanations on a real application at Alipay. We have developed several strategies from the findings of those explanations and found those strategies achieved significant lift on key metrics through online experiments.

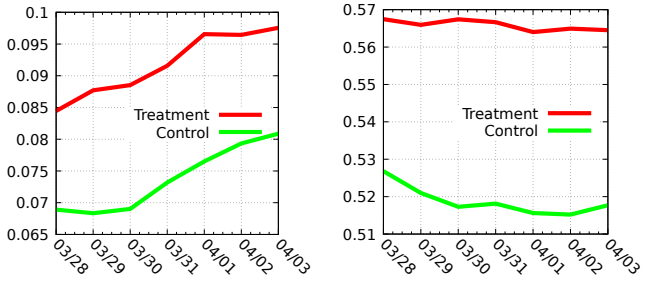
Name	Description	# of Pos	# of Neg	%Pos	GAM	XGBoost
Scenario 1	Renew vs. Not Now + Drop	5540	82511	6.29	0.7151±0.0166	0.7212±0.0130
Scenario 2	Renew vs. Not Now	5540	14247	28.00	0.7732±0.0157	0.7835±0.0153
Scenario 3	Find Lower-Risk Products vs. Not Now + Drop	1801	12116	12.94	0.6782±0.0294	0.6710±0.0281
Scenario 4	Try Other Assets vs. Not Now + Drop	1064	12116	8.07	0.6215±0.0303	0.6185±0.0293

Table 5: Scenario descriptions and AUC results. Higher is better.



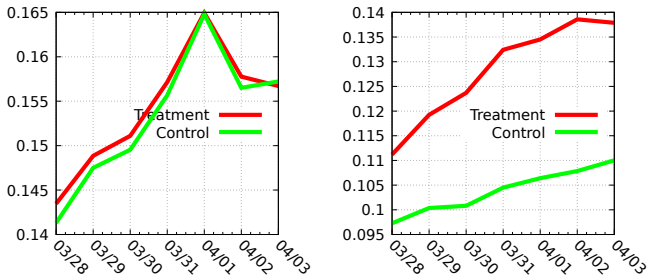
(a) Gender (b) # of clicks on account positions in 3 days (c) Account balance (d) # of clicks of product detail in 3 days

Figure 7: Selected feature shape plots on scenario 1-4. Continuous features are normalized.



(a) CTR on the entry (b) 1- BR for dialogue flow

Figure 8: Online experimental results on Entry 1.



(a) CTR on the entry (b) 1- BR for dialogue flow

Figure 9: Online experimental results on Entry 2.

REFERENCES

[1] R. Agarwal, N. Frosst, X. Zhang, R. Caruana, and G.E. Hinton. 2020. Neural additive models: Interpretable machine learning with neural nets. *arXiv preprint arXiv:2004.13912* (2020).

[2] E. Bauer and R. Kohavi. 1999. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning* 36, 1-2 (1999), 105–139.

[3] L. Breiman. 1996. Bagging predictors. *Machine learning* 24, 2 (1996), 123–140.

[4] A. Buja and W. Stuetzle. 2006. Observations on bagging. *Statistica Sinica* 16, 2 (2006), 323–351.

[5] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad. 2015. Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-day Readmission. In *KDD*.

[6] R. Caruana and A. Niculescu-Mizil. 2006. An empirical comparison of supervised learning algorithms. In *ICML*.

[7] C. Chen, T.and Guestrin. 2016. Xgboost: A scalable tree boosting system. In *KDD*.

[8] J.H. Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of Statistics* 29 (2001), 1189–1232.

[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] K. Hamidieh. 2018. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science* 154 (2018), 346–354.

[11] T.J. Hastie and R.J. Tibshirani. 1990. *Generalized additive models*. Chapman & Hall/CRC.

[12] A.J. Lee. 1990. *U-statistics: Theory and Practice*. CRC Press.

[13] P. Li. 2010. Robust logitboost and adaptive base class (abc) logitboost. In *UAI*.

[14] Y. Lou, R. Caruana, and J. Gehrke. 2012. Intelligible Models for Classification and Regression. In *KDD*.

[15] Y. Lou, R. Caruana, J. Gehrke, and G. Hooker. 2013. Accurate Intelligible Models with Pairwise Interactions. In *KDD*.

[16] L Mentch and G Hooker. 2016. Quantifying uncertainty in random forests via confidence intervals and hypothesis tests. *The Journal of Machine Learning Research* 17, 1 (2016), 841–881.

[17] R.K. Pace and R. Barry. 1997. Sparse spatial autoregressions. *Statistics & Probability Letters* 33, 3 (1997), 291–297.

[18] S. Tan, R. Caruana, G. Hooker, P. Koch, and A. Gordo. 2018. Learning global additive explanations for neural nets using model distillation. *arXiv preprint arXiv:1801.08640* (2018).

[19] S. Tan, R. Caruana, G. Hooker, and Y. Lou. 2018. Distill-and-compare: Auditing black-box models using transparent model distillation. In *AIES*.

[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.

[21] S.N. Wood. 2003. Thin plate regression splines. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 65, 1 (2003), 95–114.

[22] S.N. Wood. 2006. *Generalized additive models: an introduction with R*. CRC Press.

[23] Z. Yang, A. Zhang, and A. Sudjianto. 2020. GAMI-Net: An Explainable Neural Network based on Generalized Additive Models with Structured Interactions. *arXiv preprint arXiv:2003.07132* (2020).

[24] X. Zhang, S. Tan, P. Koch, Y. Lou, U. Chajewska, and R. Caruana. 2019. Axiomatic Interpretability for Multiclass Additive Models. In *KDD*.