Scalable Feature Selection for (Multitask) Gradient Boosted Trees

Cuize Han **Nikhil Rao** **Daria Sorokina** **Karthik Subbian**
Amazon, Palo Alto, CA {cuize, nikhilsr, dariasor, ksubbian } @ amazon.com

Abstract

Gradient Boosted Decision Trees (GBDTs) are widely used for building ranking and relevance models in search and recommendation. Considerations such as latency and interpretability dictate the use of as few features as possible to train these models. Feature selection in GBDT models typically involves heuristically ranking the features by importance and selecting the top few, or by performing a full backward feature elimination routine. On-the-fly feature selection methods proposed previously scale suboptimally with the number of features, which can be daunting in high dimensional settings. We develop a scalable forward feature selection variant for GBDT, via a novel group testing procedure that works well in high dimensions, and enjoys favorable theoretical performance and computational guarantees. We show via extensive experiments on both public and proprietary datasets that the proposed method offers significant speedups in training time, while being as competitive as existing GBDT methods in terms of model performance metrics. We also extend the method to the multitask setting, allowing the practitioner to select common features across tasks, as well as selecting task-specific features.

1 Introduction

Gradient Boosting methods Friedman [2001] are widely used in several ranking and classification tasks for web-scale data Zheng et al. [2008], Li et al. [2008]. GBDTs allow for efficient training and inference for large datasets Ke et al. [2017], Chen and Guestrin [2016]. Efficient inference is of key importance for applications such as search, where real-time vending of results at web scale in response to a search query is vital.

A key consideration for the models is the number of features used. A large number of features selected in the model severely impacts latency. Selecting a small number of features also allows for better model fitting and helps yield explainable models. While it is generally accepted that fitting a parsimonious model to the data is useful, past work on learning such models for GBDTs have been few and far between. In Ke et al. [2017], sparsity inducing penalties are used to reduce the number of trees; in Chen and Guestrin [2016], a similar technique is applied to penalize the number of leaves in each tree. One common method for feature selection in gradient boosting involves fitting the model on *all* the features, ranking the features in the order of importance Ke et al. [2017], Chen and Guestrin [2016] and selecting the top-s, where s is a positive, predefined number of features that one can handle. This kind of post-hoc thresholding is suboptimal compared to learning a sparse set of features during training itself. A second, and more often used method is backward feature elimination Mao [2004]: recursively fit a model on the (leftover) set of features, and eliminate the least important feature. The second method becomes cumbersome in the case of most real world applications, which have a small number of target features and a large number of potential features to choose from.

To alleviate this, Xu et al. [2014] proposed a forward feature selection method for gradient boosting, based on a sparsity-inducing penalty over the features. The resulting subroutine to select features is linear in the number of features. This is both wasteful and cumbersome in high dimensional settings where the number of features we want to use is significantly smaller than the total number of features available. Moreover, the sparsity penalty in the algorithm does not explicitly account for the distribution of targets in the training data for each tree. The difference in variance across the trees means a sparsity penalty that works well for

Proceedings of the 23[rd]International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

one tree might not work well for subsequent ones.

In this paper, we help address both of the above concerns. We first show how the forward feature selection method for GBDT needs to be modified to account for different variances in the residuals being fit, which we refer to as A-GBM (Adaptive Gradient Boosting Machine), since it adapts to the residual variance while fitting successive trees. The main contribution of our work is the introduction of a scalable variant of A-GBM, called GT-GBM (Group Testing GBM) that uses a group testing procedure to significantly speed up the training procedure for GBDTs. For cases where we want to select s out of d features, we show that so long as the number of samples in a node n to split is at least the order of $\left(\frac{d}{s}\right)^2 \log\log\left(\frac{d}{s}\right)$, GT-GBM selects the optimal feature to split on. GT-GBM also enjoys computational speedups so long as n is $O\left(\exp\left(\frac{d/s}{\log(d/s)}\right)\right)$. Thus, so long as the rather easy-to-satisfy

$$\left(\frac{d}{s}\right)^2 \log\log\left(\frac{d}{s}\right) \lesssim n \lesssim \exp\left(\frac{d}{s}\log^{-1}\left(\frac{d}{s}\right)\right),$$

condition holds, GT-GBM is guaranteed to be fast as well as accurate. This covers a wide range of real world applications. For example in web search cases, the number of samples is in the millions, number of features is in the hundreds and a few tens of features need to be selected.

Another major contribution is the extension of GT-GBM to the multitask setting, where our novel penalization helps us tradeoff between selecting common features across tasks, as well as task specific features. By sharing some features across tasks and selecting a few task-specific features, we can achieve better performance than standard multitask learning. We experimentally show that GT-GBM matches other feature selection methods for GBDT in performance, while being significantly faster. Results on multitask learning show the power of the flexibility to select features provided by our method. GBDT based feature selection has been shown to outperform other baselines such as the L1-regularized linear models and random forests Xu et al. [2014], so we omit these redundant comparisons to those methods in this paper.

Prior Work : The LARS Efron et al. [2004] and Lasso Tibshirani [1996] methods, along with variants Needell and Tropp [2009], Chen et al. [2001], Rao et al. [2015] allow for highly efficient training and inference on large datasets for linear models. In the nonlinear setting, kernel methods Song et al. [2012] can be trained with methods similar to the above ones, but their computational and memory complexity typically grow super-linearly with the number of samples in the data. The method in Xu et al. [2014] (referred to as GBFS, stands for Gradient Boosted Feature Selection) is a form of forward feature selection in the GBDT setting, but the tree splitting routine(s) still takes linear time with respect to the number of features in the data. We show how to avoid this. We also make a modification to GBFS to make the method more robust to the variances in the residuals as we fit more trees into the model.

Multitask learning (MTL) Caruana [1997] aims to improve model performance across multiple "tasks" by learning joint representations. Such methods are useful in cases where there is not enough data to train individual models, as in the case of neuroscience Rao et al. [2013] or where there are similarities across tasks Chen et al. [2010], Yang et al. [2009]. Work on MTL has focussed on linear models Maurer et al. [2013], where novel sparsity-aware penalties have been proposed to share models, and neural networks (Collobert and Weston [2008] for languages for example); the former being too restrictive in web search and recommendations domain, and the latter not lending itself well to real-time inference.

We formally set up the problem we intend to solve and introduce the GBFS procedure of Xu et al. [2014] in Section 2, and the variance adaptive variant of the same. In Section 3, we introduce our multitask learning method for forward feature selection. In Section 4 we derive a scalable method for forward feature selection in GBDT, and provide theoretical performance guarantees. We conduct extensive experiments in Section 5, and conclude the paper in Section 6.

2 Problem Setup and GBFS

Let $(x_i, y_i)_{i=1}^m$ be a dataset of m samples, with $x_i \in \mathcal{X} \subset \mathbb{R}^d$. y_i is a $T-$ dimensional vector (for the multitask case). Our aim is to train a GBDT model $f_{gbdt}(\mathcal{X}) \to \mathcal{Y}$, by using a small subset of features of size $s \ll d$. We denote by $[d]$ the set $\{1, 2, \cdots, d\}$. $\mathbb{1}\{C\}$ is the indicator function, taking the value 1 if condition C is satisfied, 0 otherwise.

Given $\mu > 0$, Xu et al. [2014] proposed the GBFS method, that penalizes the selection of new features via an additive penalty. Let h correspond to a tree, and $\Omega \subset [d]$ be the set of features used by the model, and g_i be the residual. At iteration k, GBFS solves

$$h_k = \arg\min_{h\in\mathcal{H}} \sum_{i=1}^m \left(g_i - h(x_i)\right)^2 + \mu \sum_{j=1}^d \mathbb{1}\left\{h \text{ uses feature } j \text{ and } j \notin \Omega\right\}, \quad (1)$$

with $\mathcal{H}$ being the space of trees we are optimizing over.

(1) can be solved by modifying the CART algorithm, which builds the tree by choosing the split to minimize the square error loss L. At each node, one chooses the best split among features $j \in [d]$ and split points $s_j \in \{x_{ij},\, i \in [m]\}$ that minimizes

$$L(j, s_j) = SSE_L(j, s_j) + SSE_R(j, s_j) + \mu \mathbb{1}_j$$

We have used the shorthand $\mathbb{1}_j$ to denote the indicator function for the event that feature j has not been previously used.

$$SSE_L(j, s_j) = \sum_i (y_i - \bar{y}_L)^2 \;\; \mathbb{1}\{x_{ij} < s_j\} \text{ and}$$
$$SSE_R(j, s_j) = \sum_i (y_i - \bar{y}_R)^2 \;\; \mathbb{1}\{x_{ij} \geq s_j\}$$

are the sum of squared errors for left and right child if we split at feature j and split points s_j. $\bar{y}_L = \frac{\sum_i y_i \mathbb{1}\{x_{ij} < s_j\}}{\sum_i \mathbb{1}\{x_{ij} < s_j\}}$, $\bar{y}_R = \frac{\sum_i y_i \mathbb{1}\{x_{ij} \geq s_j\}}{\sum_i \mathbb{1}\{x_{ij} \geq s_j\}}$ are the means in the corresponding node.

Adaptive Gradient Boosted Feature Selection : When optimizing to choose h_k, the value of the objective function in the root of the tree being built may have high variance across trees. Consequently, a penalty parameter μ that worked well until iteration $k-1$ might not be good for iteration k. Picking a good penalty parameter μ in this case becomes challenging, since we are using the same parameter for feature selection across all boosting rounds. To alleviate this situation, we propose to scale the loss function being used to fit each tree to account for the current tree root variance. That is, we modify L to be

$$\tilde{L}(j, s_j) = \frac{SSE_L(j, s_j) + SSE_R(j, s_j)}{SSE_r} + \mu \mathbb{1}_j$$

where $SSE_r = \sum_i (y_i - \bar{y})^2$, y_i is the label in the current tree root. We now only need to choose $\mu \in [0, 1]$ since the scaled split criterion $\frac{SSE_L(j,s_j)+SSE_R(j,s_j)}{SSE_r}$ is always $\in [0, 1]$. More importantly, this variance scaling ensures that the behavior of μ remains stable across each fitting round, avoiding the alternative of potentially "re-tuning" μ for each boosting round. We refer to this method as A-GBM (the 'A' referring to adaptive), since the method adapts to the variance on a per-tree bases. A-GBM training proceeds exactly like GBFS, except for the scaling part. We refer the interested reader to Appendix A for the pseudocode.

3 Multitask A-GBM with Feature Selection

The above modification that adapts to the data variance as we grow the model becomes more crucial in the multitask learning setting, where we now have T different but related tasks. Let $t \in [T]$ denote the task id, and let the data for task t be (x_i^t, y_i^t), $i \in [m_t]$ and the corresponding features be f_j^t, $j \in [d]$. For ease of presentation, we assume that all tasks have the same number of features d. In the case where the tasks have different features, we can 'zero-pad' the data and since there is no variance along these features, they will not be considered for selection in the GBDT model.

As in standard MTL, we can form groups of features, where each group is a single feature grouped across tasks Maurer et al. [2013]. Then we have d groups of features $G_j = \{f_j^t, t = 1, ., T\}, j = 1, 2, .., d$. Assuming the tasks are related, grouping the features in this manner helps us learn a joint set of features that are useful across all tasks. However, this constraint might be too restrictive: we would like to account for slight variations across tasks, and have the flexibility to select task-specific features as well. To this end, we propose to use a group sparse penalty (only penalize if the feature is from a previously unused group of features, see the formula below for details) + sparse penalty for MTL. Note that now, the function to be fit depends on the task as well as the feature, giving:

$$\tilde{L}(j, s_j) = \frac{SSE_L(j, s_j) + SSE_R(j, s_j)}{SSE_r} + \mu_G I\{j \notin \Omega_G\} + \mu_t I\left\{j \notin \Omega^t\right\}, \tag{2}$$

Where Ω_G is the set of features that have been selected across all tasks, and Ω^t is the set of features selected for task t, $t \in [T]$. μ_G, μ_t are respectively the common group sparsity parameter for all the tasks and the task specific sparsity parameter. The pseudocode for this method is presented in Algorithm 1.

Using a combination of the group sparse and sparse penalizations has been shown to be effective in multitask learning settings for linear regression and classification Rao et al. [2013], Simon et al. [2013]. To the best of our knowledge, this has not been proposed before in the tree learning setting.

4 Scalable adaptive Gradient Boosting

The methods described above end up having to compute the $SSE_L(j, s_j)$ and $SSE_R(j, s_j)$ functions defined previously for all the features in the dataset and for each split to be performed while fitting a tree. This procedure is linear in the number of features d, and the number of samples n per node where the split is being computed. For many real world applications in web search and recommendations, the total number of feature is large while the number of feature used by the model is significantly smaller. In these cases,

Algorithm 1 Pseudocode for Multitask A-GBM

Require: Data $\{x_i^t, y_i^t\}$, $i \in [m_t]$, $t \in [T]$, shrinkage ϵ, iterations N, tree growth parameter α, group penalty parameter $0 \leq \mu_G < 1$, individual task penalty parameter $0 \leq \mu_t < 1$, also $\mu_G + \mu_t < 1$

1: **for** $t = 1, 2, \ldots T$ **do**
2: Initialize prediction $H^t = 0$, residues $g_i^t = y_i^t$ and selected feature set $\Omega^t = \emptyset, \quad \Omega_G = \emptyset$
3: **end for**
4: **for** $k = 1, 2, \ldots N$ **do**
5: **for** $t = 1, 2, \ldots T$ **do**
6: Fit a tree h_k^t using α, μ_t, μ_G, data $\{\{x_i^t, g_i^t\}, i \in [m_t]\}$ and loss function (2)
7: $H^t = H^t + \epsilon h_k^t$
8: $g_i^t = y_i^t - H^t(x_i^t)$
9: $\Omega^t = \Omega^t \cup \left\{j \mid \text{tree } h_k^t \text{ uses feature } f_j^t\right\}$
10: $\Omega_G = \Omega_G \cup \left\{j \mid \text{tree } h_k^t \text{ uses feature } f_j^t\right\}$
11: **end for**
12: **end for**
13: Output H^t, $\Omega^t \ \ \forall \ \ t \in [T]$ and Ω_G

we expect that checking all the d features is not only time consuming but also redundant. If we can quickly identify those small number of s good features *without* checking them all during each node split, training time will be reduced greatly. We address this now.

4.1 Group Testing and Binary Search

The idea is to compare groups of randomly selected features and perform a binary search to eliminate the set of features that are relatively uninformative. Random selection helps reduce the bias in the ordering of the features. At each time we can eliminate half of features in this way. This depends on a key consideration: we require a metric that can be computed *efficiently* on a group of features, and one that is also *indicative* of the presence of an important feature in the group. An inefficient method will not yield computational gains, and a non-indicative metric is not going to yield an accurate solution. Suppose we have a function `GT(`G`,`M`)` that takes in a subset of features $G \subset [d]$ and a subset of samples $M \subset [m]$ as input arguments. Suppose the number of operations it takes to evaluate a split for this group of features is $\Phi(|G|, |M|)$: the computational complexity of this procedure depends on the number of samples as well as the number of features.

We will address how to construct such a function in Section 4.2. Assuming for now we do have such a function at our disposal, we give our general procedure of group testing and binary search in GBDT. We refer to this method as GT-GBM, the "GT" referring to the group-testing scheme. The pseudocode for GT-GBM is identical to that of A-GBM (Algorithm 3) except line 3 will be replaced by the subroutine we provide below in Algorithm 2. For the multitask case, line 2 in Algorithm 1 will be replaced by the subroutine.

Algorithm 2 Tree Fitting Subroutine for GT-GBM

Require: (in addition to usual hyperparameters) $s =$ desired number of features, $\delta \in (0, 1)$ (see Theorem 4.1 for details)

1: Check previously used feature set Ω for splitting and record the best standardized MSE. Call it l.
2: Independently generate $es\log(\frac{s}{\delta})$ random subsets from $[d]$ with size $\frac{d}{s}$. If $s = 1$,we just select $[d]$. Assign these to $\mathcal{G}$
3: Initialize candidate set $C = \emptyset$
4: **for** Each random subset $G \in \mathcal{G}$ **do**
5: **while** $|G| > 1$ **do**
6: Binary half split G into G_L, G_R. Let n_G be the samples to consider for this split.
7: $G = \arg\min_{G_L, G_R} (\texttt{GT(}G_L\texttt{,}n_G\texttt{)}, \texttt{GT(}G_R\texttt{,}n_G\texttt{)})$
8: **end while**
9: $C = C \cup G$
10: **end for**
11: check features in C for splitting and record the squared error value with penalty $l' + \mu$ if the feature is not used by previous trees.
12: **if** $l' + \mu < l$ **then**
13: Include best feature from C
14: **else**
15: Use one of the old features from Ω to split
16: **end if**

4.2 Constructing an Efficient `GT()` Function

In general, the number of operations for group testing and binary search in a node splitting step is $O\left(s\log(s)\log(\frac{d}{s})\Phi(d, n)\right)$. Our aim is to construct a function `GT()` such that $\Phi(d, n) \ll O(nd)$. We do this as follows: given a group of features and the samples to make the split, we sum the features up to obtain a new "pseudo-feature" [1]. We will then test this "pseudo-feature" for a split point in a fashion identical to the usual tree-splitting procedure in A-GBM. For speeding up this computation, we can compute the prefixed-sum Cormen et al. [2009] of all the features in data [2]. $\Phi(d, n)$ now is $n\log(n)$ for sorting the pseudofeature and the check for splitting. Comparing GT-GBM with the usual procedure of GBDT, we will gain a boost in

[1] We will standardize feature values by subtracting the min value and dividing the max value, so that all feature values are within [0,1]

[2] after computing the prefixed sum for features in each random subset and storing the result in one-pass, getting the "pseudofeature" value will just be O(1)

Table 1: Complexity comparisons between A-GBM (and hence GBFS) and GT-GBM. P and T refer to the precomputation and training phases respectively.

Algo.	Phase	Time	Space
A-GBM	P	$O\left((n\log n)\,d\right)$	$O\left(nd\right)$
	T	$O\left(nd\right)$	$O\left(nd\right)$
GT-GBM	P	$O\left((n\log s)\,d\right)$	$O\left(nd\log s\right)$
	T	$O\left((s\log(s)\log(d/s))\,n\log n\right)$	$O\left(n\right)$

training speed if (see table 1 for details)

$$s\log(s)\log(n) << \frac{d}{\log(\frac{d}{s})}, \tag{3}$$

which is easy to satisfy in real world applications. More detailed complexity comparision is the following.

The main preprocessing of data for training A-GBM or any other sort-based GBDT algorithm involves getting and storing the sorted value pairs of features and targets. This takes $O\left((n\log n)\,d\right)$ operations and needs $O\left(nd\right)$ space. GT-GBM, however, does not need to precompute the sorted value pairs since target values will be sorted based on the pseudofeature during binary search and split. Instead, it calculates and records the prefixed sum for each of $O\left(s\log s\right)$ random subset of features with size d/s. So the precomputation for GT-GBM takes $O\left((s\log s)\,nd/s\right) = O\left((n\log s)\,d\right)$ time and space. With a bit more space used during precompute, GT-GBM needs $O\left(n\right)$ instead of $O\left(nd\right)$ space since sorted value pairs doesn't need to be stored and passed to child nodes during growing the tree. Table 1 shows this comparison.

4.3 Theoretical Guarantees for GT-GBM

If $s = 1$, then the important feature will be in either G_L or G_R. All other features will act as random noise. Intuitively the procedure will select the group that contains the relevant feature with high probability as long as it is highly correlated with the target. This process recurses until we find the important feature. If there are multiple relevant features in the same group, however, their effects can cancel each other out. An idea then is to generate several random subsets of features and apply our `GroupTest` to these subsets, as we do in Algorithm 2 (line 2). If a subset contains only one of the important features, then it reduces to the case for one feature and we can find that feature with high probability. The following result bounds this probability as a function of the number of subsets generated:

Theorem 4.1. *Suppose that there are s important features. To ensure that for every important feature there is a random subset that only cover this feature with probability $1-\delta$, it is sufficient to generate p random subsets of features, where $p \geq es\log\left(\frac{s}{\delta}\right)$ and $e = 2.71..$ is the base of the natural logarithm.*

We refer the reader to Appendix B.2 for the Proof.

Next, we show that the method we proposed is guaranteed to recover the correct set of features with high probability, under mild asumptions.

Theorem 4.2. *Suppose $X = (X_1, \ldots, X_d)$ are independent of each other, and $0 \leq X_i \leq 1$, with non-zero variance. Let $B_d^2 = Var(X_1 + .. + X_d)$ and $\mu_d = \mathbb{E}(X_1 + .. + X_d)$. Assume $\lim_{d\to\infty} B_d = \infty$ and $\eta = \lim_{d\to\infty} \frac{\mu_d}{B_d}$. Suppose there is an unknown subset $S^* \subset [d]$, $|S^*| = s$, such that $Y = \mu + \sum_{i\in S^*} f_i(X_i) + \epsilon$, where $\mu = \mathbb{E}Y$ is the population mean and ϵ is noise (mean 0, bounded and independent of all other variables). f_is are unknown univariate monotonic functions with $\mathbb{E}f_i(X_i) = 0$. Suppose at a node we have n i.i.d. samples. Then for $\delta \in (0,1)$, if $d \geq d_0$ and*

$$n \geq C_0 \left(\frac{d}{s}\right)^2 \log\left(\log\left(\frac{d}{s}\right)\frac{\log(1/\delta)}{\delta}\right) \tag{4}$$

GTGBM finds the best split feature with probability at least $1-\delta$., where C_0 and d_0 are positive constants that only depend on the fixed unknown functions f_i, and η.

Note that the assumptions made above are based on Sparse Additive Models Ravikumar et al. [2009], and encompass a wide variety of practical settings.

Proof Sketch. Recall the split criterion of CART algorithm. For a split variable Z (a feature or the pseduo-feature in GTGBM that represents a group of variables) and threshold t, the criterion is to minimize

$$L_n(Z,t) = \frac{1}{n}\left(\sum_{i:Z_i<t}\left(Y_i - \bar{Y}_L\right)^2 + \sum_{i:Z_i\geq t}\left(Y_i - \bar{Y}_R\right)^2\right) \tag{5}$$

The population split criterion (corresponds to when we have infinite amount of data) is to minimize

$$\begin{aligned} L(Z,t) \quad = \quad & \mathbb{E}[(Y - \mathbb{E}\left[Y|Z<t\right])^2 1\{Z<t\} \\ & + (Y - \mathbb{E}\left[Y|Z\geq t\right])^2 1\{Z\geq t\}] \end{aligned} \tag{6}$$

For an important feature index $i \in S^*$, we consider the random subset S generated in GTGBM that only covers i. Then during binary search for active feature within S, we only need to prove for the split subset S_L, S_R (assume S_L contains the important index i), that $\min_t L_n(Z_{S_L}, t) < \min_t L_n(Z_{S_R}, t)$ w.h.p.

Let $Z_S = \sum_{j\in S} X_j$. For the population version, we can prove $L(Z_{S_R}, t) = \mathbb{E}Y^2, \forall t$ (no variance reduction), $\min_t L(Z_{S_L}, t) < \mathbb{E}Y^2$ and the difference only depends on the signal strength of f_i and how correlated are Z_{S_L} and Y. To investigate the sample split criterion, we need to quantify : (a) How the amount of variance reduced decays with the increase of $|S_L|$

(Lemma B.1 states $\approx \frac{1}{|S_L|} \approx \frac{s}{d}$.) (b) How the uniform approximation error between empirical and population split criterion decays with n. (Lemma B.2 states $\sup_t |L_n(Z,t) - L(Z,t)| = O_p(\sqrt{\frac{1}{n}})$)

Combining the above gets us the result. We refer the reader to Appendix B.5 for the detailed proof. □

Combining equations (3) and (4) in Theorem 4.2 show that so long as the number of samples n at a node to split satisfies

$$\left(\frac{d}{s}\right)^2 \log\log\left(\frac{d}{s}\right) \lesssim n \lesssim \exp\left(\frac{d}{s}\log^{-1}\left(\frac{d}{s}\right)\right),$$

GT-GBM will find the correct feature to split significantly faster than GBFS. This condition is easily satisfied in most real world applications, where the number of samples and the number of features are large, and relatively shallow trees are used to train the models which is the case for gradient boosting procedures.

An experiment on synthetic data shows the bound in Theorem 4.2 is quite conservative. Figure 1 indicates that the dependence between n and d is potentially linear. We leave the tightening of the bound for future work. For the experiment, we fix $s = 3, \delta = 0.1$ and generate $y = 2x_1 - 3 * 2^{x_2} + \log_2(1 + x_3) + \epsilon$ where x_1, x_2, x_3 and other irrelevant features are i.i.d uniform on $[0, 1]$ and $\epsilon \sim \mathcal{N}(0, 1)$. We replicate each experiment 50 times and calculate the ratio of success (success means the candidate feature set found by GT-GBM contains both x_1, x_2, x_3).

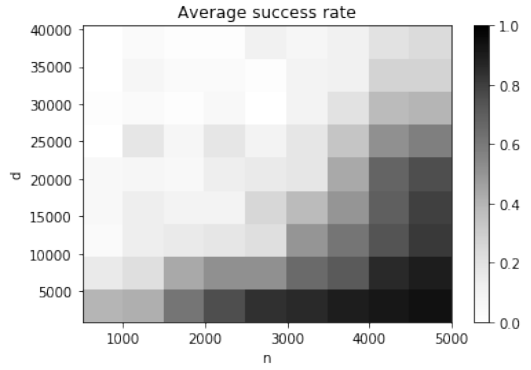

Figure 1: Average success rate as a function of ambient dimension d and sample size n. Dark regions indicate values near 1, and light closer to 0. Note the near linear dependence between n and d.

5 Experiments and Results

First, we extensively test A-GBM and GT-GBM on publicly available datasets. Next, we apply the methods to proprietary datasets, and evaluate GT-GBM for ranking and multiclass classification tasks. Results on an internal dataset for classification are provided in Appendix E. We compare our methods with other GBDT feature selection methods, as that is the main focus in this paper.

5.1 Public Datasets and Baselines

We compare A-GBM and GT-GBM methods with GBFS Xu et al. [2014] and the GBDT method with ranking all features, and retraining with K most important features (referred to as GBDT-topK here). For GBDT-topK, we use LightGBM Ke et al. [2017] and use it's default feature scoring mechanism to rank the features by importance. We train the models on the Gisette[3] , Epsilon[4], and the Flight Delay[5] datasets. They are all for classification tasks. For the latter, we use the variant with 100K samples, and the same script to generate the data as provided in the repository. Details for all the datasets are provided in Table 2.

Table 2: Experimental datasets

Dataset	# samples	# features
Gisette	6000	5000
Epsilon	80000	2000
Flight	100000	634

For each of the methods we use, we tune all the parameters on a held out validation set, and report the results on a separate test set. For GBFS, A-GBM and GT-GBM, we choose the corresponding μ that achieves the best performance on the validation dataset, regardless of the number of features they select. For this reason, we end up picking different number of features for different methods. For GBDT-topK, we train on all the features, and pick top K features, where K is the maximum of the number of features picked by the 3 other methods. We then retrain the model with these K and report results on the test set. Optimal hyperparameter values to reproduce our results are provided in Appendix C.

Speed and Performance Comparisons : First, we show that the proposed methods perform either comparatively, or outperform the baselines. Table 3 shows the performance metrics for the methods we compare, indicating that there's very little performance loss over the baseline methods. For the sake of

[3] https://archive.ics.uci.edu/ml/datasets/Gisette

[4] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

[5] https://github.com/szilard/benchm-ml

completeness, we also report the results obtained from training the GBDT model on all the features, with no feature selection in Appendix D. Furthermore, the flight delay dataset has a large number of categorical features, and a large number of data points compared to features. Even in this case, GT-GBM outperforms the other baselines.

Table 3: Performance comparison on various datasets. Note that GT-GBM consistently picks fewer features while still outperforming or competing with A-GBM and GBFS. As expected, GBDT-topK suffers from poor approximation as a result of picking top K features after fitting on the whole set of features.

Dataset	Method	# feats	RMSE	AUC_ROC
Gisette	GBDT-topK	178	0.187	97.88
	GBFS	172	0.183	99.01 (+1.15%)
	A-GBM	178	0.182	99.18 (+1.33%)
	GT-GBM	170	0.182	99.19 (+1.34%)
Epsilon	GBDT-topK	306	0.377	91.8
	GBFS	306	0.363	93.0 (+1.99%)
	A-GBM	250	0.366	93.2 (+2.21%)
	GT-GBM	255	0.373	93.2 (+2.21%)
Flight	GBDT-topK	67	0.391	71.1
	GBFS	67	0.389	71.6 (+0.70%)
	A-GBM	48	0.389	71.7 (+0.84%)
	GT-GBM	45	0.390	71.6 (+0.70%)

Next, we compare the training time for all the methods in Figure 2. The Figure shows that GT-GBM is significantly faster than the competing methods on all the datasets, by an order of magnitude for Gisette, and two orders of magnitude for Epsilon. The gap is smaller for Flight dataset, since the ratio of the number of samples to the number of features is much smaller.

Evaluating Correlations : In Figure 3 we show that the features selected by the GT-GBM methods are less correlated than those picked by fitting all the features, and selecting the top K (via the feature importance scores obtained via GBDT). We fix $K = 20$, and plot the Pearson correlation coefficient for the Gisette data. When the number of features we want to select is constrained, it is important to select features that are as uncorrelated from each other as possible, as this allows for maximal information gain.

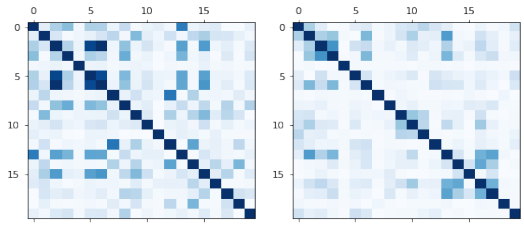

Figure 3: Pairwise pearson correlations for the top 20 features selected by GBDT-topK (left) and GT-GBM (right) methods. The lighter squares indicate values closer to 0.

5.2 Performance on Proprietary datasets

Next, we apply the GT-GBM and A-GBM methods on proprietary datasets. We use aggregated data sets containing only de-identified data from search logs of an e-commerce engine (i.e. they don't include personally identifying information about individuals in the dataset). We make use of 4 datasets across 2 tasks. C1 and C2 are classification tasks, and R1 and R2 are ranking tasks. Results on C1 and C2 are in Appendix E, since the previous experiments already evaluated GT-GBM on classification data. In all the cases below, we choose 20 as the desired number of features in our models so as to illustrate an example where extreme latency constraints are enforced.

The ranking task is akin to the standard relevance task in a search engine: in response to a query, and a set of items that are matched, the job is to rank the items in the order of relevance. Since this is a ranking task, we report the Mean Reciprocal Rank (MRR) for the datasets. Again, we see that GT-GBM is competitive with the other methods (while being faster) (Table 4).

Table 4: Comparison of various methods on the Ranking tasks (R1 and R2). Similar to the classification setting, GT-GBM is competitive with the baselines, and achieves the same result in significantly less time.

Dataset	Measure	GBDT-topK	GBFS	GT-GBM
R1	**MRR**	0.530	0531	**0.532**
	RMSE	0.159	**0.158**	**0.158**
R2	**MRR**	0.496	**0.499**	0.498
	RMSE	0.103	**0.101**	**0.101**

5.3 Multitask Feature Selection

Finally, we test the multitask variant of our algorithm on two other proprietary datasets: M1 and M2. M1 is a classification dataset that categorizes a query into 3 categories (head, torso, tail). The idea is to see if there are highly predictive features in one task that can be used in other tasks where there is a lack of data. At the same time, there might be task-specific features that are useful, which our model accounts for as well. M2 is a dataset that uses query-items across countries, similar to the dataset used in Chapelle et al. [2010]. Due to space constraints, details about M2 and results are provided in Appendix F.

We tune the two parameters μ_G and μ_t which control the proportion of common active features and task-specific important features via cross-validation, and report the results on a held out test set. In Figure 4 (and 5 in Appendix), SingleTask refers to training the model on the combined training data in the single task mode with the task number used as a categorical feature. Multitask_GroupSparse refers to the Multi-

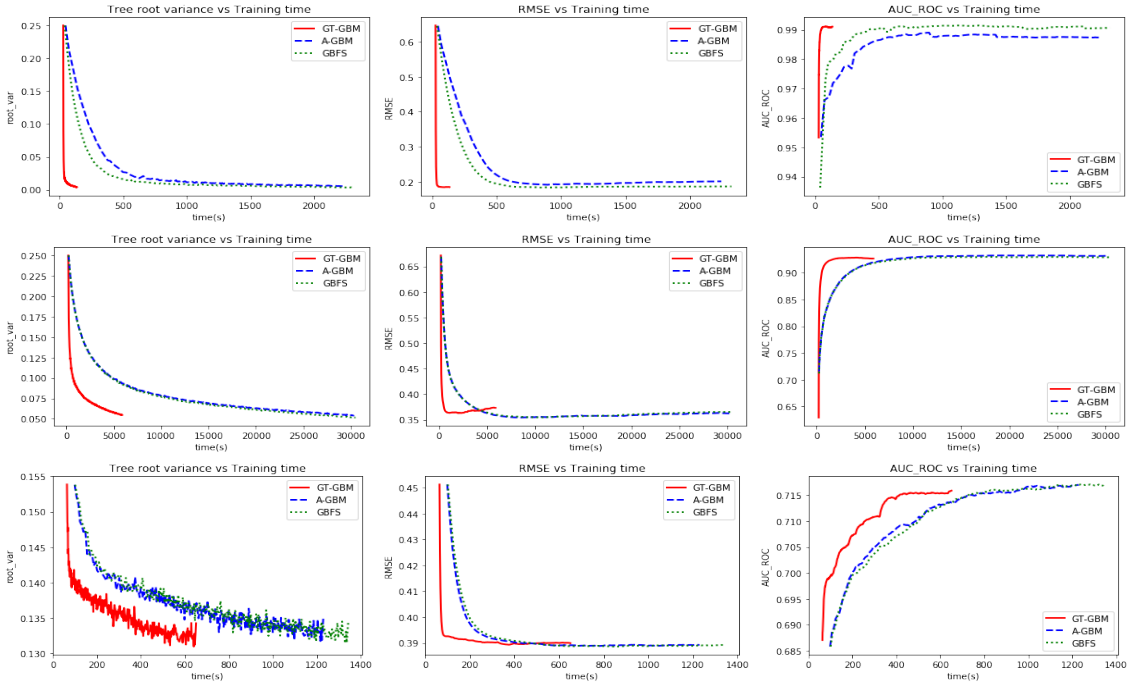

Figure 2: Timing comparisons of all the methods on various datasets, Gisette (top), Epsilon (middle), and Flight (bottom). In all the cases, we see that GT-GBM outperforms the other methods, by orders of magnitude. We plot the tree root variance (left), RMSE (middle) and Area under ROC curve (right) for all datasets as a function of time.

task model we developed, but forcing all the features across tasks to be the same, which is the standard multitask learning framework (effectively $\mu_t = 0$). Multitask refers to the model that has the full flexibility, where both sparse and group sparse parameters can be nonzero. "Total" refers to the overall metric, after taking a weighted average of the scores across the tasks, weighted proportional to the number of samples in each task. The figures show that the Multitask model outperforms both the other methods, across all tasks as well as overall.

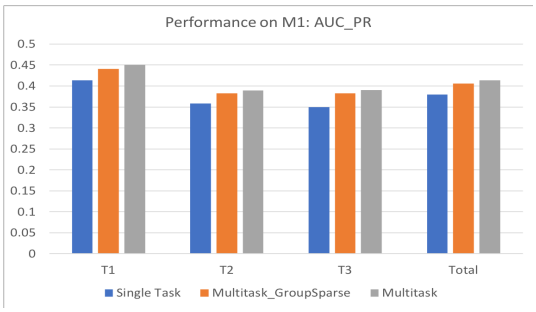

Figure 4: Performance on M1, for Area Under Precision-Recall curves. We see that having the flexibility to choose both task specific and common features across tasks helps boost performance. T1, T2, T3 refer to the three query level tasks respectively.

6 Conclusions

In this paper, we developed a feature selection procedure for gradient boosted decision trees that adapts itself to the variations in the data, and built a scalable version of the same. The scalable algorithm we developed uses a novel group testing and binary search heuristic to achieve significant speedups over baseline methods, with almost no change in performance. We provided theoretical performance guarantees that establish both the speedup and correctness, and empirical results corroborating the same. We also developed a multitask variant of this algorithm, that is flexible enough for the practitioner to transition between choosing the same set of features and training independent models across tasks. Experiments on multiple ranking and classification datasets show that the developed method compares to state of the art methods in performance, while at the same time takes significantly less time to train.

References

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Olivier Chapelle, Pannagadatta Shivaswamy, Srinivas Vadrevu, Kilian Weinberger, Ya Zhang, and Belle

Tseng. Multi-task learning for boosting with application to web search ranking. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1189–1198. ACM, 2010.

Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

Xi Chen, Seyoung Kim, Qihang Lin, Jaime G Carbonell, and Eric P Xing. Graph-structured multi-task regression and an efficient optimization method for general fused lasso. *arXiv preprint arXiv:1005.3579*, 2010.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2008.

Kezhi Z Mao. Orthogonal forward selection and backward elimination algorithms for feature subset selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):629–634, 2004.

Andreas Maurer, Massi Pontil, and Bernardino Romera-Paredes. Sparse coding for multitask and transfer learning. In *International Conference on Machine Learning*, pages 343–351, 2013.

Deanna Needell and Joel A Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and computational harmonic analysis*, 26(3):301–321, 2009.

Nikhil Rao, Christopher Cox, Rob Nowak, and Timothy T Rogers. Sparse overlapping sets lasso for multitask learning and its application to fmri analysis. In *Advances in neural information processing systems*, pages 2202–2210, 2013.

Nikhil Rao, Parikshit Shah, and Stephen Wright. Forward–backward greedy algorithms for atomic norm regularization. *IEEE Transactions on Signal Processing*, 63(21):5798–5811, 2015.

Pradeep Ravikumar, John Lafferty, Han Liu, and Larry Wasserman. Sparse additive models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(5):1009–1030, 2009.

Noah Simon, Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013.

Le Song, Alex Smola, Arthur Gretton, Justin Bedo, and Karsten Borgwardt. Feature selection via dependence maximization. *Journal of Machine Learning Research*, 13(May):1393–1434, 2012.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

Zhixiang Xu, Gao Huang, Kilian Q Weinberger, and Alice X Zheng. Gradient boosted feature selection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 522–531. ACM, 2014.

Xiaolin Yang, Seyoung Kim, and Eric P Xing. Heterogeneous multitask learning with joint sparsity constraints. In *Advances in neural information processing systems*, pages 2151–2159, 2009.

Zhaohui Zheng, Hongyuan Zha, Tong Zhang, Olivier Chapelle, Keke Chen, and Gordon Sun. A general boosting method and its application to learning ranking functions for web search. In *Advances in neural information processing systems*, pages 1697–1704, 2008.