MABSplit: Faster Forest Training Using Multi-Armed Bandits

Mo Tiwari[1] **Ryan Kang**[1,*] **Je-Yong Lee**[2,*] **Sebastian Thrun**[1]

Chris Piech[1] **Ilan Shomorony**[#,3] **Martin Jinye Zhang**[#,4]

Abstract

Random forests are some of the most widely used machine learning models today, especially in domains that necessitate interpretability. We present an algorithm that accelerates the training of random forests and other popular tree-based learning methods. At the core of our algorithm is a novel node-splitting subroutine, dubbed MABSplit, used to efficiently find split points when constructing decision trees. Our algorithm borrows techniques from the multi-armed bandit literature to judiciously determine how to allocate samples and computational power across candidate split points. We provide theoretical guarantees that MABSplit improves the sample complexity of each node split from linear to logarithmic in the number of data points. In some settings, MABSplit leads to 100x faster training (an 99% reduction in training time) without any decrease in generalization performance. We demonstrate similar speedups when MABSplit is used across a variety of forest-based variants, such as Extremely Random Forests and Random Patches. We also show our algorithm can be used in both classification and regression tasks. Finally, we show that MABSplit outperforms existing methods in generalization performance and feature importance calculations under a fixed computational budget. All of our experimental results are reproducible via a one-line script at `https://github.com/ThrunGroup/FastForest`.

1 Introduction

Random Forest (RF) is a supervised learning technique that is widely used for classification and regression tasks [28, 15]. In RF, an ensemble of decision trees (DTs) is trained for the same prediction task. Each DT consists of a series of nodes that represent `if/then/else` comparisons on the feature values of a given datapoint and are used to produce an output label. In RF, each DT is typically trained on a *bootstrap* sample of the original dataset and considers a random sample of available features at each node split [14]. The prediction of the each DT is aggregated to provide an output label for the whole RF. By aggregating the prediction of each DT in the ensemble, RFs tend to be more robust to noise and overfitting [52] and are capable of capturing more complex patterns in the data than a single DT [59].

RF has gained tremendous popularity due to its flexibility, usefulness in multi-class classification and regression tasks, high performance across a broad range of data types, natural support for missing features, and relatively low computational complexity [61, 53, 10, 35, 30]. Furthermore, RF inherits

$*$ denotes equal contribution. $\#$ denotes joint supervision. Correspondence should be addressed to M.T. (`motiwari@stanford.edu`), I.S. (`ilans@illinois.edu`), and M.J.Z. (`jinyezhang@hsph.harvard.edu`).

1: Department of Computer Science, Stanford University

2: Oxford University

3: Electrical and Computer Engineering, University of Illinois at Urbana-Champaign

4: Department of Epidemiology, Harvard T.H. Chan School of Public Health

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

the interpretability of decision trees because the prediction of each constituent DT can be explained through a sequence of binary decisions. RFs have been successfully applied in contexts as varied as the prediction of legal court decisions [32], solar radiation analysis [10], and the Higgs Boson classification problem [3]. In the era of big data, a simple and flexible machine learning technique such as RF is expected to play a key role in processing large datasets and providing accurate and interpretable predictions.

The need for training prediction models on massive datasets and doing so on compute-constrained hardware, such as smartphones and Internet-of-Things devices, requires the development of new algorithms that can deliver faster results without sacrificing generalization performance [61]. For this reason, recent work has proposed ways to accelerate the training of RFs, both at the algorithmic level and at the hardware level.

At the algorithmic level, most work focuses on fast construction of each individual DT. Each DT is built by identifying the feature f and threshold t that best split the data points according to the prediction targets. The data points are split into subsets based on whether their feature f has a value less than t or feature f has a value greater than t. The process is then recursed for each resulting subset. Most of the complexity in this process is in identifying the pair (f, t) that provides the best split for a set of N data points, which typically requires $O(N)$ computation per split. Recent proposals include computing (or estimating) f and t from a subsample of the data points and features, or quantizing the feature values. The latter technique creates a histogram of values of each feature across the data points and restricts t to be at the edges of histogram bins.

While existing approaches provide significant speed up in the training of RFs, they often require prespecification of fixed hyperparameters, such as the proportion of data points or features to subsample, and are not adaptive to the underlying data distribution. Moreover, when comparing different candidate features for a split, all features are treated on equal footing and the quality of their split is computed based on the same number of data points. Intuitively, this is wasteful because features that are not informative for the prediction task can be identified based on a smaller number of data points. Alternatively, an adaptive scheme could better allocate computational resources towards a promising set of candidate features and achieve a better tradeoff between computational cost and generalization performance.

In this work, we propose MABSplit, a fast subroutine for the node-splitting problem, which adaptively refines the estimate of the "quality" of each feature-threshold pair (f, t) as a candidate split. Bad split candidates can be discarded early, which can lead to significant computational savings. The core idea behind our algorithm is to formulate the node-splitting task as a multi-armed bandit problem [34, 2, 5, 58], where each pair (f, t) is a distinct arm. The unknown parameter of each arm, μ_{ft}, corresponds to the quality of the split based on feature f and threshold t, where the split quality is measured in terms of how much the split would reduce label impurity. An arm (f, t) can be "pulled" by computing the reduction of label impurity induced by a new data point sampled from the dataset. This allows us to compute an estimate $\hat{\mu}_{ft}$ and a corresponding confidence interval, which can be used in a batched variant of the Upper Confidence Bound (UCB) and successive elimination algorithms [34, 63] to identify the best arm (f, t). Crucially, MABSplit uses the adaptive sampling tools of multi-armed bandits to avoid computing the split qualities over the entire dataset.

We demonstrate the benefits of MABSplit on a variety of datasets, for both classification and regression tasks. In some settings, MABSplit algorithm leads to 100x faster training (a 99% reduction in training time), without any decrease in test accuracy, over an exact implementation of RF that searches for the optimal (f, t) pair via brute-force computation. Additionally, we demonstrate similar speedups when using MABSplit across a variety of forest-based variants, such as Extremely Random Forests and Random Patches.

1.1 Related work

Random Forests were originally proposed by Ho [28]. In its original formulation, RF constructs n_{tree} DTs, where each DT is trained on a bootstrap sample of all N data points and a random subset of the features at each node (a technique known as random subspacing [29]). More recently, the need for training RFs on large datasets has prompted the development of several techniques to accelerate training at both the software and hardware levels.

Software acceleration of RF: Most of the software and algorithmic acceleration techniques focus on the training of each individual DT. FastForest [61] accelerates the node-splitting task using three ideas: subsampling a pre-specified number of data points without replacement (subbagging), subsampling a pre-specified number of features dependent on the current number of data points (Dynamic Restricted Subspacing), and dividing values of a given feature into T bins, where T depends on the number of data points at the node (Logarithmic Split-Point Sampling, inspired by the single-tree SPAARC algorithm [62]). MABSplit is inspired by the ideas in FastForest, but does not require the number of data points or features to be prespecified and, instead, determines them by adapting to the data distribution.

Other recent work has also used adaptivity to identify the best split. For example, Very Fast Decision Trees (VFDTs) [19] and Extremely Fast Decision Trees (EFDTs) [40] are incremental decision tree learning algorithms in which trees can be updated in streaming settings. Acceleration of the node-splitting task is achieved by adaptively selecting a subset of data points sufficient to distinguish the best and second best splits. These approaches are similar to ours, but the sampled data points are used to evaluate all possible splits. MABSplit, in contrast, adaptively discards unpromising splits early. The F-forest [23] algorithm also applies adaptivity and uses an upper bound on the impurity reduction of each split in order to discard candidate splits. This is similar in spirit to the goal of MABSplit, but is based on a deterministic, conservative upper bound on the impurity reduction (as opposed to MABSplit's statistical estimate) and incurs computation linear in the node size for each split, even when considering a fixed number of possible split thresholds per feature.

Other variations of RF have been proposed to improve training time. Random Patches [39] builds trees based on a subset of data points and features that is fixed for each entire forest. ExtraTrees (ETs) [24] draw a random subset of K features at each node and, for each one, chooses a number R of random splits. It then selects the split that yields the largest impurity reduction from among these KR candidate splits.

Other recent work attempts to accelerate RF training by identifying the optimal number of decision trees needed in the forest [45], a form of hyperparameter tuning. The MABSplit subroutine can also be incorporated into these methods.

Hardware acceleration of RF: The training of RFs can also be significantly accelerated through the use of specialized hardware. For instance, the implementation of RF available in Weka [21] allows trees to be trained on different cores and reduces forest training time. A GPU-based parallel implementation of RF has also been proposed in [26]. These solutions require specialized hardware (e.g., GPU-based PC video cards) and are inappropriate for everyday users locally executing data-mining tasks on standard PCs or smartphones. As such, it is still desirable to develop techniques to improve prediction performance and processing speed at an algorithmic, platform-independent level.

2 Preliminaries

We now formally describe the RF algorithm and other tree-based models, all of which rely on a node-splitting subroutine. We consider N data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ where each $\mathbf{x}_i$ is an M-dimensional feature vector and y_i is its target. Following standard literature, we consider flexible feature types such as numerical or categorical. We consider both categorical targets for classification and numerical targets for regression. With a slight abuse of notation, we use $\mathcal{X}$ to mean either the set of indices $\{i\}$ or the values $\{(\mathbf{x}_i, y_i)\}$, with the meaning clear from context.

An RF contains n_{tree} decision trees, each trained on a set of N bootstrapped data points (sampled with replacement) and a random subset of features at each node. The whole RF, an ensemble model, outputs the trees' majority vote in classification and the trees' average prediction in regression [28, 15]. We focus on the top-down approach of constructing DTs by choosing the feature-threshold pair at each step that best splits the set of targets at a given node [52].

We now discuss the method by which the best split is chosen in each node. Consider a node with n data points $\mathcal{X}$ and m features $\mathcal{F}$. Note that n and m at the given node may or may not be the same as N and M of the entire dataset (for example, RF considers a random subset of $m = \sqrt{M}$ features at each node [12]). Let $\mathcal{X}_{\text{L},ft}$ and $\mathcal{X}_{\text{R},ft}$ be the left and right child subsets of $\mathcal{X}$ when $\mathcal{X}$ is split according to feature f at threshold t. The approach finds the split that best reduces the label

impurity; i.e., finds

$$f^*, t^* = \underset{f \in \mathcal{F}, t \in \mathcal{T}_f}{\arg\min} \frac{|\mathcal{X}_{\text{L},ft}|}{n} I(\mathcal{X}_{\text{L},ft}) + \frac{|\mathcal{X}_{\text{R},ft}|}{n} I(\mathcal{X}_{\text{R},ft}) - I(\mathcal{X}), \tag{1}$$

where $I(\mathcal{S})$ measures the impurity of targets $\{y_i\}_{i \in \mathcal{S}}$ and $\mathcal{T}_f$ is the set of allowed thresholds for feature f. Common impurity measures include Gini impurity or entropy for classification, and mean-squared-error (MSE) for regression [16]:

$$\text{Gini} : 1 - \sum_{k=1}^{K} p_k^2, \quad \text{Entropy} : -\sum_{k=1}^{K} p_k \log_2 p_k, \quad \text{MSE} : \frac{1}{n} \sum_{i \in \mathcal{X}} (y_i - \bar{y})^2, \tag{2}$$

where K is the total number of classes and $p_k = \frac{1}{n} \sum_{i \in \mathcal{X}} \mathbb{I}(y_i = k)$ is the proportion of class k in $\mathcal{X}$ in classification, and $\bar{y} = \frac{1}{n} \sum_{i \in \mathcal{X}} y_i$ is the average target value in regression. We note that our proposed algorithm, MABSplit, does not assume any particular structure of $I(\cdot)$.

While the conventional RF considers all $(n-1)$ possible splits among the n generally different values in the dataset for a given feature f, in this work we focus on the histogram-based variant that chooses the threshold from a set of predefined values $\mathcal{T}_f$, e.g., $|\mathcal{T}_f|$ equally-spaced histogram bin edges; this variant is substantially more efficient, offers comparable accuracy, and has been used in most state-of-the-art implementations such as XGBoost and LightGBM [51, 17, 33, 62]. A naïve algorithm finds the best feature-threshold pairs in Equationh (1) by evaluating the label impurity reduction for each feature-threshold over all n data points, which incurs computation linear in n.

3 MABSplit

We now discuss MABSplit and how it can reduce the complexity of the node-splitting problem to logarithmic in n.

With the same notation as in Section 2, we note that Equation (1) is equivalent to

$$f^*, t^* = \underset{f \in \mathcal{F}, t \in \mathcal{T}_f}{\arg\min} \frac{|\mathcal{X}_{\text{L},ft}|}{n} I(\mathcal{X}_{\text{L},ft}) + \frac{|\mathcal{X}_{\text{R},ft}|}{n} I(\mathcal{X}_{\text{R},ft}) \tag{3}$$

and so we focus on solving Equation (3). Let $\mu_{ft} = \frac{|\mathcal{X}_{\text{L},ft}|}{n} I(\mathcal{X}_{\text{L},ft}) + \frac{|\mathcal{X}_{\text{R},ft}|}{n} I(\mathcal{X}_{\text{R},ft})$ be the optimization objective for feature-threshold pair (f, t). Omitting the dependence on K, computing μ_{ft} exactly is at least $O(n)$. MABSplit, however, *estimates* μ_{ft} with less computation by drawing $n' < n$ independent samples with replacement from $\mathcal{X}$. As shown in Subsec. 3.1, it is possible to construct a point estimate $\hat{\mu}_{ft}(n')$ and a $(1-\delta)$ confidence interval (CI) $C_{ft}(n', \delta)$ for the parameter μ_{ft}, where n' and δ determine estimation accuracy. The width of these CIs generally scales with $\sqrt{\frac{\log 1/\delta}{n'}}$. To estimate the solution to Problem (3) with high confidence, we can then choose to sample different amounts of data points for different features so as to estimate their impurity reductions to varying degrees of accuracy. Intuitively, promising features-threshold splits with high impurity reductions (lower values of μ_{ft}) should be estimated with high accuracy with many data points, while less promising ones with low impurity reductions (higher values of μ_{ft}) can be discarded early.

The exact adaptive estimation procedure, MABSplit, is described in Algorithm 1. It can be viewed as a batched version of the conventional UCB algorithm [34, 63] combined with successive elimination [22], is straightforward to implement, and has been used in other applications [58, 55, 56, 6, 7]. Algorithm 1 uses the set $\mathcal{S}_{\text{solution}}$ to track all potential solutions to Problem (3); $\mathcal{S}_{\text{solution}}$ is initialized as the set of all feature-threshold pairs $\{(f, t)\}$ and Algorithm 1 maintains the mean objective estimate $\hat{\mu}_{ft}$ and $(1-\delta)$ CI C_{ft} for each potential solution $(f, t) \in \mathcal{S}_{\text{solution}}$.

In each iteration, a new batch of data points $\mathcal{X}_{\text{batch}}$ is used to evaluate the split quality for all potential feature-threshold splits in $\mathcal{S}_{\text{solution}}$, which allows the estimate of each $\hat{\mu}_{ft}$ to be made more accurate. Based on the current estimate, if a candidate's lower confidence bound $\hat{\mu}_{ft} - C_{ft}$ is greater than the upper confidence bound of the most promising candidate $\min_{f,t}(\hat{\mu}_{ft} + C_{ft})$, we remove it from $\mathcal{S}_{\text{solution}}$. This process continues until there is only one candidate in $\mathcal{S}_{\text{solution}}$ or until we have sampled more than n data points. In the latter case, we know that the difference between the remaining candidates in $\mathcal{S}_{\text{solution}}$ is so subtle that an exact computation is warranted. MABSplit then compute those candidates' objectives exactly and returns the best candidate in the set.

3.1 Point estimates and confidence intervals for impurity metrics

We now discuss MABSplit's construction of point estimates and confidence intervals of μ_{ft} based on a set of n' points, $\{(\mathbf{X}_i, Y_i)\}_{i=1}^{n'}$, sampled independently and with replacement from $\mathcal{X}$. We consider two widely used impurity metrics in classification, Gini impurity and entropy, although mean estimates and confidence intervals for other settings and metrics can be derived similarly (more details are provided in Appendix 3).

Let $p_{\text{L},k} \coloneqq \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(x_{if} < t, y_i = k)$ and $p_{\text{R},k} \coloneqq \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(x_{if} \geq t, y_i = k)$ denote the proportion of the full n data points in class k and each of the two subsets created by the split (f, t) (we call these subsets "left" and "right", respectively). Note that

$$\sum_k p_{\text{L},k} = \frac{|\mathcal{X}_{\text{L},ft}|}{n} \quad \text{and} \quad \sum_k p_{\text{R},k} = \frac{|\mathcal{X}_{\text{R},ft}|}{n}. \tag{4}$$

Furthermore, let $\hat{p}_{\text{L},k} \coloneqq \frac{1}{n'}\sum_{i=1}^{n'} \mathbb{I}(X_{if} < t, Y_i = k)$ and $\hat{p}_{\text{R},k} \coloneqq \frac{1}{n'}\sum_{i=1}^{n'} \mathbb{I}(X_{if} \geq t, Y_i = k)$ denote the empirical estimates of $p_{\text{L},k}$ and $p_{\text{R},k}$ based on the n' subsamples drawn thus far. Then $\{\hat{p}_{\text{L},k}, \hat{p}_{\text{R},k}\}_{k=1}^{K}$ jointly follow a multinomial distribution satisfying

$$\mathbb{E}[\hat{p}_{\text{L},k}] = p_{\text{L},k}, \quad \text{Var}[\hat{p}_{\text{L},k}] = \frac{1}{n'} p_{\text{L},k}(1 - p_{\text{L},k}),$$
$$\mathbb{E}[\hat{p}_{\text{R},k}] = p_{\text{R},k}, \quad \text{Var}[\hat{p}_{\text{R},k}] = \frac{1}{n'} p_{\text{R},k}(1 - p_{\text{R},k}).$$

since for each random data point $(\mathbf{X}_i, Y_i)$, exactly one element of the set $\{\hat{p}_{\text{L},k}, \hat{p}_{\text{R},k}\}_{k=1}^{K}$ is incremented. Using Equations (2) and (4), and the definition of μ_{ft} after Equation (3), we write

$$\text{Gini impurity}: \ \mu_{ft} = 1 - \frac{\sum_k p_{\text{L},k}^2}{\sum_k p_{\text{L},k}} - \frac{\sum_k p_{\text{R},k}^2}{\sum_k p_{\text{R},k}}, \tag{5}$$

$$\text{Entropy}: \ \mu_{ft} = -\sum_k p_{\text{L},k} \log_2 \frac{p_{\text{L},k}}{\sum_{k'} p_{\text{L},k'}} - \sum_k p_{\text{R},k} \log_2 \frac{p_{\text{R},k}}{\sum_{k'} p_{\text{R},k'}}, \tag{6}$$

where we can use the empirical parameters $\{\hat{p}_{\text{L},k}, \hat{p}_{\text{R},k}\}_{k=1}^{K}$ as plug-in estimators for the true parameters $\{p_{\text{L},k}, p_{\text{R},k}\}_{k=1}^{K}$ to produce the point estimate $\hat{\mu}_{ft}$.

In MABSplit (Algorithm 1), each batch of B data points is used to update each $\hat{p}_{\text{L},k}$ and $\hat{p}_{\text{R},k}$, which are used in turn to update our point estimates $\hat{\mu}_{ft}$ and the corresponding CIs. The CIs of $\hat{\mu}_{ft}$ are based on standard error derived using the delta method [60]. As in standard applications of the delta method, the estimates $\hat{\mu}_{ft}$ are asymptotically unbiased and their corresponding CIs are asymptotically valid. Appendix 3 provides further details, including a derivation of the CIs and discussion of convergence properties.

3.2 Algorithmic and implementation details

We considered sampling with replacement in MABSplit (Algorithm 1) primarily for the ease of theoretical analysis. In practice, we found sampling without replacement was more computationally efficient and did not significantly change the results, and was used in the actual implementation.

For DTs, in classification tasks, we allow individual DTs to provide soft votes and average their predicted class probabilities to determine the forest's predicted class label, following existing approaches [47]. This is in contrast with hard votes, in which each DT is only permitted to produce its best label and the forest's prediction is determined by majority voting. For the fixed-budget experiments in Section 5, we terminate tree construction and do not split nodes further if doing so would violate our budget constraints.

4 Analysis of the Algorithm

In this section, we prove that MABSplit returns the optimal feature-threshold pair for a node split with high probability. Furthermore, we provide bounds on computational complexity of MABSplit, which can lead to a logarithmic dependence on the number of data points n under weak assumptions.

Algorithm 1 MABSplit ($\mathcal{X}, \mathcal{F}, \mathcal{T}_f, I(\cdot), B, \delta$)

$\mathcal{S}_{\text{solution}} \leftarrow \{(f,t),\ \forall\ f \in \mathcal{F},\ \forall t \in \mathcal{T}_f\}$ ▷ Set of potential solutions to Problem (3)
$n_{\text{used}} \leftarrow 0$ ▷ Number of data points sampled
For all $(f,t) \in \mathcal{S}_{\text{solution}}$, set $\hat{\mu}_{ft} \leftarrow \infty, C_{ft} \leftarrow \infty$ ▷ Initialize mean and CI for each arm
for all $f \in \mathcal{F}$ **do**
    Create empty histogram, h_f, with $|\mathcal{T}_f| = T$ equally spaced bins
while $n_{\text{used}} < n$ and $|\mathcal{S}_{\text{solution}}| > 1$ **do**
    Draw a batch sample $\mathcal{X}_{\text{batch}}$ of size B with replacement from $\mathcal{X}$
    for all unique f in $\mathcal{S}_{\text{solution}}$ **do**
        for all x in $\mathcal{X}_{\text{batch}}$ **do**
            Insert x_f into histogram h_f ▷ Each insertion is $O(1)$
    for all $(f,t) \in \mathcal{S}_{\text{solution}}$ **do**
        Update $\hat{\mu}_{ft}$ and C_{ft} based on histogram h_f ▷ For fixed f, this is $O(T)$
    $\mathcal{S}_{\text{solution}} \leftarrow \{(f,t) : \hat{\mu}_{ft} - C_{ft} \leq \min_{f,t}(\hat{\mu}_{ft} + C_{ft})\}$ ▷ Retain only promising splits
    $n_{\text{used}} \leftarrow n_{\text{used}} + B$
if $|\mathcal{S}_{\text{solution}}| = 1$ **then**
    return $(f^*, t^*) \in \mathcal{S}_{\text{solution}}$
else
    Compute μ_{ft} exactly for all $(f,t) \in \mathcal{S}_{\text{solution}}$
    return $(f^*, t^*) = \arg\min_{(f,t)\in\mathcal{S}_{\text{solution}}} \mu_{ft}$

As above, consider a node with n data points $\mathcal{X}$, m features $\mathcal{F}$, and T possible thresholds for each feature ($|\mathcal{T}_f| = T$ for all f). Suppose $(f^*, t^*) = \arg\min_{f\in\mathcal{F}, t\in\mathcal{T}_f} \mu_{ft}$ is the optimal feature-threshold pair at which to split the node. For any other feature-threshold pair (f,t), define $\Delta_{ft} := \mu_{ft} - \mu_{f^*t^*}$. To state the following results, we will assume that, for a fixed feature-threshold pair (f,t) and n' randomly sampled datapoints, the $(1-\delta)$ confidence interval scales as $C_{ft}(n', \delta) = O(\sqrt{\frac{\log 1/\delta}{n'}})$. (This assumption is justified for the confidence intervals of Gini impurity and entropy under weak assumptions on the $\mu_{f,t}$'s [60]). With this assumption, we state the following theorem:

Theorem 1. *Assume $\exists c_0 > 0$ s.t. $\forall \delta > 0, n' > 0$, we have $C_{ft}(n', \delta) < c_0\sqrt{\frac{\log 1/\delta}{n'}}$. For $\delta = \frac{1}{n^2 mT}$, with probability at least $1 - \frac{1}{n}$, Algorithm 1 returns the correct solution to Equation* (3)*. Furthermore, Algorithm 1 uses a total of M computations, where*

$$\mathbb{E}[M] \leq \sum_{f\in\mathcal{F}, t\in\mathcal{T}_f} \min\left[\frac{4c_0^2}{\Delta_{ft}^2}\log(n^2 mT) + B, 2n\right] + 2mT. \tag{7}$$

Theorem 1 is proven in Appendix 1. Intuitively, Theorem 1 states that with high probability, MABSplit returns the optimal feature-threshold pair at which to split the node. The bound Equation (7) suggests the computational cost of a feature-threshold pair (f,t), i.e., $\min\left[\frac{4c_0^2}{\Delta_{ft}^2}\log(n^2mT) + B, 2n\right]$, depends on Δ_{ft}, which measures how close its optimization parameter μ_{ft} is to $\mu_{f^*t^*}$. Most reasonably different features $f \neq f^*$ will have a large Δ_{ft} and incur computational cost $O(\log(n^2mT))$ that is sublinear in n.

In turn, this implies MABSplit takes only $O(mT\log(n^2mT))$ computations per feature-threshold pair if there is reasonable heterogeneity among them. As proven in Appendix 2 of [5], this is the case under a wide range of distributional assumptions on the μ_{ft}'s, e.g., when the μ_{ft}'s follow a sub-Gaussian distribution across the pairs (f,t). Such assumptions ensure that MABSplit has an overall complexity of $O(mT\log(n^2mT))$, which is sublinear in the number of data points n. We note that in the worst case, however, MABSplit may take $O(n(m+T))$ computations per feature-threshold pair when most splits are equally good, in which case MABSplit reduces to a batched version of the naïve algorithm. This may happen, for example, in highly symmetric datasets where all splits reduce the impurity equally. Other recent work provides further discussion on the conversion between a bound like Equation (7), which depends on the Δ_i's, and a bound in terms of other problem parameters such as $O(mT\log(n^2mT))$ under various assumptions on the μ_{ft}'s [5, 63, 7, 58, 4, 6].

Finally, we note that δ is a hyperparameter governing the error rate. It is possible to prove results analogous to Theorem 1 for arbitrary δ.

5 Experimental Results

We demonstrate the advantages of MABSplit in two settings. In the first setting, the baseline models with and without MABSplit are trained to completion and we report wall-clock training time and generalization performance. In the second setting, we consider training each forest with a fixed computational budget and study effect of MABSplit on generalization performance. We provide a description of each dataset in Appendix 6.

We note that head-to-head wall-clock time comparisons with common implementations of forest-based algorithms, such as Weka's [21] or `scikit-learn`'s [48], would be unfair due to their extensive hardware- and language-level optimizations. As such, we reimplement these baselines in Python and focus on algorithmic improvements. The only difference between our model and the baselines is the call to the node-splitting subroutine (MABSplit in our model, and the exact, brute-force solver for the baseline models); thus, any improvements in runtime are due to improvements in the node-splitting algorithm. This is verified by profiling the implementations and measuring the relative time spent in the node-splitting subroutine versus total runtime (see Appendix 5). Our approach allows us to focus on algorithmic improvements as opposed implementation-specific optimizations. Our implementation of these baselines may also be of independent interest and we verify the quality of our implementations, via agreement with `scikit-learn`, in Appendix 4. We also provide a brief discussion of how an optimized version of our reimplementations may outperform `scikit-learn` in Appendix 7. On the MNIST dataset, our optimized implementation trains approximately 4x faster than `scikit-learn`'s `DecisionTreeClassifier`.

Baseline Models: We compare the histogrammed versions of three baselines with and without MABSplit: Random Forest (RF), ExtraTrees [24], and Random Patches (RP) [39]. In Random Forests, each tree fits a bootstrap sample of N datapoints and considers random subset of $\sqrt{M}$ features at every node split. Extra Trees (also known as Extremely Randomized Forests) are identical to Random Forests except for two differences. First, in regression problems, all features are considered at every node split (in classification problems, we still use only $\sqrt{M}$ features). Second, the histogram edges of a feature are randomly chosen from a uniform distribution over that feature's minimum and maximum value. In classification problems, each histogram has $\sqrt{M}$ bins and in regression problems, each histogram has M bins. These conventions follow standard implementations [47]. Note that in ExtraTrees, the bins in a feature's histogram need not be equally spaced. Random Patches is identical to Random Forests but the training dataset is reduced to α_n of its original datapoints and α_f of its original features, where α_n and α_f are prespecified constants, and the subsampled dataset is fixed for the training of the entire forest. Full settings for all experiments are given in Appendix 6.

5.1 Wall-clock time comparisons

In the first setting, we compare baseline models with and without MABSplit in terms of wall-clock training time. Tables 1 and 2 show that MABSplit provides similar generalization performance but faster training than the usual naïve algorithm for node-splitting for almost all baselines in both classification and regression tasks. Across various tasks, MABSplit leads to approximately 2x-100x faster training, a reduction of training time of 50-99%. These benefits are wholly attributable to MABSplit as the only difference between successive minor rows in each table is the node-splitting subroutine (see Appendix 5 for further discussion).

5.2 Fixed budget comparisons

In the second setting, we consider training models under a fixed computational budget. As before, insertion into a histogram is taken to be an $O(1)$ operation. This is justified if the histogram's thresholds are evenly spaced, wherefore the correct bin in which to insert a value can be indexed into directly. (If the bins are unevenly spaced, we may perform binary searches to locate the correct bin, which is $O(\log T)$ and does not depend on n, or cache the results of these binary searches for an evenly-spaced grid across the range of the given feature's value.)

MNIST Dataset ($N = 60,000$)			
Model	Training Time (s)	Number of Insertions	Test Accuracy
RF	1542.83 ± 5.837	1.44E+08 ± 4.85E+05	0.777 ± 0.005
RF + MABSplit	**40.359 ± 0.246**	**3.37E+06 ± 1.62E+04**	**0.763 ± 0.008**
ExtraTrees	1789.653 ± 2.396	1.68E+08 ± 0.00E+00	0.762 ± 0.003
ExtraTrees + MABSplit	**50.217 ± 0.304**	**4.32E+06 ± 7.69E+03**	**0.755 ± 0.002**
RP	1421.963 ± 8.368	1.32E+08 ± 6.95E+05	0.771 ± 0.003
RP + MABSplit	**38.415 ± 0.245**	**3.17E+06 ± 1.40E+04**	**0.768 ± 0.003**
APS Failure at Scania Trucks Dataset ($N = 60,000$)			
Model	Training Time (s)	Number of Insertions	Test Accuracy
RF	20.542 ± 0.048	3.77E+06 ± 9.66E+03	0.985 ± 0.0
RF + MABSplit	**0.455 ± 0.002**	**6.94E+04 ± 2.19E+02**	**0.985 ± 0.0**
ExtraTrees	18.849 ± 0.027	3.78E+06 ± 0.00E+00	0.985 ± 0.0
ExtraTrees + MABSplit	**0.406 ± 0.001**	**7.00E+04 ± 0.00E+00**	**0.985 ± 0.0**
RP	17.63 ± 0.054	3.22E+06 ± 1.18E+04	0.985 ± 0.0
RP + MABSplit	**0.399 ± 0.003**	**5.96E+04 ± 2.19E+02**	**0.985 ± 0.0**
Forest Covertype Dataset ($N = 581,012$)			
Model	Training Time (s)	Number of Insertions	Test Accuracy
RF	117.351 ± 0.123	1.86E+07 ± 0.00E+00	0.559 ± 0.028
RF + MABSplit	**0.88 ± 0.009**	**3.98E+04 ± 1.79E+02**	**0.505 ± 0.004**
ExtraTrees	117.984 ± 0.119	1.86E+07 ± 0.00E+00	0.539 ± 0.022
ExtraTrees + MABSplit	**2.942 ± 0.856**	**3.69E+05 ± 1.22E+05**	**0.5 ± 0.005**
RP	104.456 ± 0.737	1.62E+07 ± 8.31E+04	0.51 ± 0.008
RP + MABSplit	**0.815 ± 0.004**	**3.50E+04 ± 0.00E+00**	**0.507 ± 0.005**

Table 1: Wall-clock training time, number of histogram insertions, and test accuracies for various models with and without MABSplit. MABSplit can accelerate these models by over 100x in some cases (an 99% reduction in training time) while achieving comparable accuracy. The number of histogram insertions correlates strongly with wall-clock training time, which justifies our focus on accelerating the node-splitting algorithm via reductions in sample complexity.

Beijing Multi-Site Air-Quality Dataset (Regression, $N = 420,768$)		
Model	Training Time (s)	Test MSE
RF	138.782 ± 1.581	1164.576 ± 0.761
RF + MABSplit	**67.089 ± 1.682**	**1109.542 ± 23.776**
ExtraTrees	115.592 ± 3.061	1028.054 ± 11.355
ExtraTrees + MABSplit	**53.607 ± 1.278**	**1015.234 ± 6.535**
RP	108.174 ± 1.24	1128.299 ± 25.78
RP + MABSplit	**60.639 ± 3.642**	**1125.816 ± 33.56**
SGEMM GPU Kernel Performance Dataset (Regression, $N = 241,600$)		
Model	Training Time (s)	Test MSE
RF	32.606 ± 0.859	69733.002 ± 57.401
RF + MABSplit	**16.51 ± 0.224**	**69493.921 ± 73.133**
ExtraTrees	30.624 ± 0.686	69734.948 ± 54.876
ExtraTrees + MABSplit	**14.086 ± 0.295**	**69585.029 ± 80.281**
RP	26.091 ± 0.417	66364.998 ± 894.568
RP + MABSplit	**16.409 ± 0.952**	**66310.138 ± 896.237**

Table 2: Wall-clock training time and test MSEs for various models with and without MABSplit. MABSplit can accelerate these models by up to 2x (an 50% reduction in training time) while achieving comparable results. We omit the number of histogram insertions in favor of wall-clock time for simplicity; unlike in classification, the different baseline regression models have widely varying histogram bin counts. Since the histogram insertion complexity scales as $O(\log T)$, the comparison across models would not be fair.

MNIST Dataset ($N = 60,000$)		
Model	Number of Trees	Test Accuracy
RF	0.2 ± 0.179	0.143 ± 0.026
RF + MABSplit	**15.8 ± 0.179**	**0.83 ± 0.002**
ExtraTrees	0.2 ± 0.179	0.144 ± 0.027
ExtraTrees + MABSplit	**12.0 ± 0.0**	**0.814 ± 0.001**
RP	1.0 ± 0.0	0.253 ± 0.003
RP + MABSplit	**16.8 ± 0.179**	**0.832 ± 0.002**
APS Failure at Scania Trucks Dataset ($N = 60,000$)		
Model	Number of Trees	Test Accuracy
RF	1.0 ± 0.0	0.985 ± 0.0
RF + MABSplit	**5.8 ± 0.179**	**0.989 ± 0.0**
ExtraTrees	1.0 ± 0.0	0.985 ± 0.0
ExtraTrees + MABSplit	**5.6 ± 0.219**	**0.989 ± 0.0**
RP	1.0 ± 0.0	0.985 ± 0.0
RP + MABSplit	**6.8 ± 0.179**	**0.989 ± 0.0**
Forest Covertype Dataset ($N = 581,012$)		
Model	Number of Trees	Test Accuracy
RF	0.4 ± 0.219	0.514 ± 0.019
RF + MABSplit	**99.8 ± 0.179**	**0.675 ± 0.002**
ExtraTrees	0.2 ± 0.179	0.496 ± 0.006
ExtraTrees + MABSplit	**23.4 ± 1.403**	**0.677 ± 0.002**
RP	0.6 ± 0.219	0.534 ± 0.03
RP + MABSplit	**100.0 ± 0.0**	**0.675 ± 0.002**

Table 3: Classification performance under a fixed computational budget (number of histogram insertions) for various models with and without MABSplit. MABSplit allows for more trees to be trained and leads to better generalization performance.

Intuitively, the MABSplit algorithm allows for splitting a given node with less data point queries and histogram insertions than the naïve solution. As such, when the computational budget is fixed, forests trained with MABSplit should be able to split more nodes and therefore train more trees than forests trained with the naïve solver. Prior work suggests that increasing the number of trees in a forest improves generalization performance by reducing variance at the cost of slightly increased bias [27].

Tables 3 and 4 demonstrate the generalization performance of different models as the computational budget is held constant for different classification and regression tasks. When using MABSplit, the trained forests consist of more trees and demonstrate better generalization performance across all baseline models.

5.3 Feature stability comparisons

We also apply MABSplit to compute feature importances under a fixed budget. We follow the common approach of computing the feature importances of multiple forests and then measuring the stability of feature selection across forests using Permutation Feature Importance and Mean Decrease in Impurity (MDI) [42, 49] (see Appendix 6 for a further discussion of these metrics). The forests trained with MABSplit demonstrate better feature stabilities than those trained with the naïve algorithm; see Table 5. Note that the datasets used for these experiments are different from the real-world datasets used in the other experiments and are described in Appendix 6.

6 Discussions and Conclusions

In this work, we presented a novel algorithm, MABSplit, for determining the optimal feature and corresponding threshold at which to split a node in tree-based learning models. Unlike prior models such as Random Patches, in which the subsampling hyperparameters α_n and α_f must be prespecified manually, MABSplit requires no tuning and queries only as much data as is needed by virtue of its

Beijing Multi-Site Air-Quality Dataset ($N = 420,768$)		
Model	Number of Trees	Test MSE
RF	0.0 ± 0.0	3208.93 ± 0.0
RF + MABSplit	$\mathbf{12.0 \pm 0.0}$	$\mathbf{927.013 \pm 2.042}$
RP	0.0 ± 0.0	3208.93 ± 0.0
RP + MABSplit	$\mathbf{11.0 \pm 0.4}$	$\mathbf{875.764 \pm 3.064}$
ExtraTrees	0.0 ± 0.0	3208.93 ± 0.0
ExtraTrees + MABSplit	$\mathbf{9.0 \pm 0.0}$	$\mathbf{834.338 \pm 4.377}$
SGEMM GPU Kernel Performance Dataset ($N = 241,600$)		
Model	Number of Trees	Test MSE
RF	0.0 ± 0.0	131323.839 ± 0.0
RF + MABSplit	$\mathbf{5.6 \pm 0.219}$	$\mathbf{28571.393 \pm 357.433}$
RP	0.8 ± 0.179	102616.047 ± 6647.02
RP + MABSplit	$\mathbf{2.8 \pm 0.593}$	$\mathbf{64876.329 \pm 13350.921}$
ExtraTrees	0.0 ± 0.0	131323.839 ± 0.0
ExtraTrees + MABSplit	$\mathbf{5.0 \pm 0.0}$	$\mathbf{29919.254 \pm 344.409}$

Table 4: Regression performance under a fixed computational budget (number of histogram insertions). MABSplit allows for more trees to be trained and leads to better generalization performance.

Importance Model	Stability Metric	Dataset	Stability
RF	MDI	Random Classification	0.536 ± 0.039
RF + MABSplit	**MDI**	**Random Classification**	$\mathbf{0.863 \pm 0.016}$
RF	MDI	Random Regression	0.134 ± 0.021
RF + MABSplit	**MDI**	**Random Regression**	$\mathbf{0.674 \pm 0.043}$
RF	Permutation	Random Classification	0.579 ± 0.023
RF + MABSplit	**Permutation**	**Random Classification**	$\mathbf{0.69 \pm 0.023}$
RF	Permutation	Random Regression	0.116 ± 0.017
RF + MABSplit	**Permutation**	**Random Regression**	$\mathbf{0.437 \pm 0.044}$

Table 5: Stability scores under a fixed computational budget (number of histogram insertions). MABSplit allows more trees to be trained, which leads to greater feature stabilities across the forests.

adaptivity to the data distribution. Indeed, robustness to choice of hyperparameters is one of primary appeals of algorithms like RF [50].

MABSplit avoids the expensive $O(n\log n)$ sort used in many existing baselines and the $O(n)$ computational complexity of their corresponding histogrammed versions. MABSplit can be used in conjunction with existing software- and hardware-specific optimizations and with other methods such as Logarithmic Split-Point Sampling and boosting [61]. In boosting, MABSplit has the potential advantage of only needing to update data points' targets on-the-fly, as needed by its sampling, as opposed to current approaches that update targets for the entire dataset at each iteration. Additionally, MABSplit may permit easier parallelization due to lower memory requirements than existing algorithms, which may enable greater use in edge computing and may be adaptable to streaming settings.

We also note that $\log n^2$ term in the complexity of MABSplit (Equation 7) comes from a union bound over all possible n^2 confidence intervals computed for each feature-threshold pair across all stages of the algorithm (see the proof of Theorem 1 in Appendix 1). This union bound may be weak. Instead, one can more precisely compute the number of confidence intervals in terms of the arm gaps (the Δ_i's). In this way, the complexity result (Equation 7) can be phrased in terms of the Δ_i's instead of n. Intuitively, this would lead to a complexity bound in terms of the data-generating distribution that is independent of dataset size. Indeed, this is how complexity results are usually stated for multi-armed bandit algorithms [31]. We leave a more detailed investigation of this topic to future work.

Acknowledgements: M. T. was funded by a J.P. Morgan AI Fellowship, a Stanford Indisciplinary Graduate Fellowship, a Stanford Data Science Scholarship, and an Oak Ridge Institute for Science and Engineering Fellowship. M.J.Z. is supported by NIH grant R01 MH115676. I. S. was supported in part by the National Science Foundation (NSF) under grant CCF-2046991.

References

[1] Serhat Arslan, Mo Tiwari, and Chris Piech. Using google search trends to estimate global patterns in learning. In *Proceedings of the Seventh ACM Conference on Learning @ Scale*, L@S '20, page 185–195, New York, NY, USA, 2020. Association for Computing Machinery.

[2] Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. Best arm identification in multi-armed bandits. In *International Conference on Learning Theory*, pages 41–53, 2010.

[3] Mourad Azhari, Abdallah Abarda, Badia Ettaki, Jamal Zerouaoui, and Mohamed Dakkon. Higgs boson discovery using machine learning methods with pyspark. *Procedia Computer Science*, 170:1141–1146, 2020.

[4] Vivek Bagaria, Tavor Z Baharav, Govinda M Kamath, and N Tse David. Bandit-based monte carlo optimization for nearest neighbors. *IEEE Journal on Selected Areas in Information Theory*, 2(2):599–610, 2021.

[5] Vivek Bagaria, Govinda M. Kamath, Vasilis Ntranos, Martin J. Zhang, and David Tse. Medoids in almost-linear time via multi-armed bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 500–509, 2018.

[6] Tavor Z Baharav, Gary Cheng, Mert Pilanci, and David Tse. Approximate function evaluation via multi-armed bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 108–135. PMLR, 2022.

[7] Tavor Z. Baharav and David Tse. Ultra fast medoid identification via correlated sequential halving. In *Advances in Neural Information Processing Systems*, pages 3650–3659, 2019.

[8] Rafael Ballester-Ripoll, Enrique G. Paredes, and Renato Pajarola. Sobol tensor trains for global sensitivity analysis. *CoRR*, abs/1712.00233, 2017.

[9] G. P. Basharin. On a statistical estimate for the entropy of a sequence of independent random variables. *Theory of Probability & Its Applications*, 4(3):333–336, 1959.

[10] Lamara Benali, Gilles Notton, A Fouilloy, Cyril Voyant, and Rabah Dizene. Solar radiation forecasting using artificial neural network and random forest methods: Application to normal beam, horizontal diffuse and global components. *Renewable energy*, 132:871–884, 2019.

[11] Yoshua Bengio, Tristan Deleu, Edward J. Hu, Salem Lahlou, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *CoRR*, abs/2111.09266, 2021.

[12] Simon Bernard, Laurent Heutte, and Sébastien Adam. Influence of hyperparameters on random forest accuracy. In *International workshop on multiple classifier systems*, pages 171–180. Springer, 2009.

[13] Jock Blackard and Denis Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24:131–151, 12 1999.

[14] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[15] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[16] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[17] 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, 2016.

[18] Kaustubh D. Dhole et al. Nl-augmenter: A framework for task-sensitive natural language augmentation, 2021.

[19] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[20] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[21] Frank Eibe, Mark A Hall, and Ian H Witten. The weka workbench. online appendix for data mining: practical machine learning tools and techniques. In *Morgan Kaufmann*. Elsevier Amsterdam, The Netherlands, 2016.

[22] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. In *Journal of Machine Learning Research*, volume 7, pages 1079–1105, 2006.

[23] Yasuhiro Fujiwara, Yasutoshi Ida, Sekitoshi Kanai, Atsutoshi Kumagai, Junya Arai, and Naonori Ueda. Fast random forest algorithm via incremental upper bound. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2205–2208, 2019.

[24] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[25] Christopher Gondek, Daniel Hafner, and Oliver R. Sampson. Prediction of failures in the air pressure system of scania trucks using a random forest and feature engineering. In Henrik Boström, Arno Knobbe, Carlos Soares, and Panagiotis Papapetrou, editors, *Advances in Intelligent Data Analysis XV*, number 9897 in Lecture Notes in Computer Science, pages 398–402, Cham, 2016. Springer International Publishing.

[26] Hakan Grahn, Niklas Lavesson, Mikael Hellborg Lapajne, and Daniel Slat. Cudarf: a cuda-based implementation of random forests. In *2011 9th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pages 95–101. IEEE, 2011.

[27] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *Elements of Statistical Learning*. Springer, 2009.

[28] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[29] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[30] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[31] Kevin Jamieson and Robert Nowak. Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting. In *Annual Conference on Information Sciences and Systems*, pages 1–6, 2014.

[32] Daniel Martin Katz, Michael J Bommarito, and Josh Blackman. A general approach for predicting the behavior of the supreme court of the united states. *PloS one*, 12(4):e0174698, 2017.

[33] 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. *Advances in neural information processing systems*, 30, 2017.

[34] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[35] SK Lakshmanaprabu, K Shankar, M Ilayaraja, Abdul Wahid Nasir, V Vijayakumar, and Naveen Chilamkurti. Random forest for big data classification in the internet of things using optimal features. *International journal of machine learning and cybernetics*, 10(10):2609–2618, 2019.

[36] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery.

[37] Gabriele Lanaro, Quan Nguyen, and Sakis Kasampalis. *Advanced Python Programming: Build high performance, concurrent, and multi-threaded apps with Python using proven design patterns*. Packt Publishing.

[38] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[39] Gilles Louppe and Pierre Geurts. Ensembles on random patches. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 346–361. Springer, 2012.

[40] Chaitanya Manapragada, Geoffrey I Webb, and Mahsa Salehi. Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1953–1962, 2018.

[41] Ali Mohsen and Mo Tiwari. Image compression and classification using qubits and quantum deep learning, 2021.

[42] Kristin K. Nicodemus. Letter to the editor: On the stability and ranking of predictors from random forest variable importance measures. *Briefings in Bioinformatics*, 12(4):369–373, 04 2011.

[43] Sarah Nogueira, Konstantinos Sechidis, and Gavin Brown. On the stability of feature selection algorithms. *J. Mach. Learn. Res.*, 18(1):6345–6398, 2017.

[44] Cedric Nugteren and Valeriu Codreanu. Cltune: A generic auto-tuner for opencl kernels. In *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 195–202, 2015.

[45] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. How many trees in a random forest? In *International workshop on machine learning and data mining in pattern recognition*, pages 154–168. Springer, 2012.

[46] Liam Paninski. Estimation of entropy and mutual information. *Neural Comput.*, 15(6):1191–1253, jun 2003.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] Miriam Piles, Rob Bergsma, Daniel Gianola, Hélène Gilbert, and Llibertat Tusell. Feature selection stability and accuracy of prediction models for genomic prediction of residual feed intake in pigs using machine learning. *Frontiers in Genetics*, 12, 2021.

[50] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: data mining and knowledge discovery*, 9(3):e1301, 2019.

[51] Sanjay Ranka and Vineet Singh. Clouds: A decision tree classifier for large datasets. In *Proceedings of the 4th knowledge discovery and data mining conference*, volume 2, 1998.

[52] Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, 2005.

[53] Eduarda MO Silveira, Sergio Henrique G Silva, Fausto W Acerbi-Junior, Monica C Carvalho, Luis Marcelo T Carvalho, Jose Roberto S Scolforo, and Michael A Wulder. Object-based random forest modelling of aboveground forest biomass outperforms a pixel-based approach in a heterogeneous and mountain tropical environment. *International Journal of Applied Earth Observation and Geoinformation*, 78:175–188, 2019.

[54] Aarohi Srivastava et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2022.

[55] Mo Tiwari, Ryan* Kang, Je-Yong* Lee, Sebastian Thrun, Martin J Zhang, and Ilan Shomorony. Faster maximum inner product search in high dimensions. *Preprint*.

[56] Mo Tiwari, Ryan* Kang, Luke* Lee, Sebastian Thrun, Martin J Zhang, Ilan Shomorony, and Chris Piech. Intelligent caching for sequential, permutation-invariant multi-armed bandits. *Preprint*.

[57] Mo Tiwari, Chris Piech, Medina Baitemirova, Namperumalsamy V Prajna, Muthiah Srinivasan, Prajna Lalitha, Natacha Villegas, Niranjan Balachandar, Janice T Chua, Travis Redd, Thomas M Lietman, Sebastian Thrun, and Charles C Lin. Differentiation of active corneal infections from healed scars using deep learning. *Ophthalmology*, 129, 2022.

[58] Mo Tiwari, Martin J Zhang, James Mayclin, Sebastian Thrun, Chris Piech, and Ilan Shomorony. Banditpam: Almost linear time k-medoids clustering via multi-armed bandits. *Advances in Neural Information Processing Systems*, 33:10211–10222, 2020.

[59] Kagan Tumer and Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection science*, 8(3-4):385–404, 1996.

[60] Aad W Van der Vaart. *Asymptotic statistics*, volume 3. Cambridge university press, 2000.

[61] Darren Yates and Md Zahidul Islam. Fastforest: Increasing random forest processing speed while maintaining accuracy. *Information Sciences*, 557:130–152, 2021.

[62] Darren Yates, Md Zahidul Islam, and Junbin Gao. Spaarc: a fast decision tree algorithm. In *Australasian Conference on Data Mining*, pages 43–55. Springer, 2018.

[63] Martin Zhang, James Zou, and David Tse. Adaptive monte carlo multiple testing via multi-armed bandits. In *International Conference on Machine Learning*, pages 7512–7522, 2019.

[64] Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and Song Xi Chen. Cautionary tales on air-quality improvement in beijing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We have all reviewed the claims in the abstract and introduction and confirmed that they reflect the paper's contributions and scope.
   (b) Did you describe the limitations of your work? [Yes] We discuss the assumptions necessary for MABSplit to work in Section 4 and also discuss failure modes when these assumptions are violated. We also describe further room to improve our work in Section 6
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We discuss potential negative societal impacts in the Future Work section
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have all reviewed the ethics review guidelines and ensured that our paper conforms to them.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] The statement of Theorem 1 contains a full description of assumptions, which is further expounded upon before and after the theorem statement.
   (b) Did you include complete proofs of all theoretical results? [Yes] The proof of Theorem 1 is provided in Appendix 1
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Yes, full specifications of all experiments are provided in Section 5 and Appendix 6, and full, documented source code is provided to reproduce our results.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Yes, full specifications of all experiments are provided in Section 5 and Appendix 6, and full, documented source code is provided to reproduce our results.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Every numerical measurement in Tables 1, 2, 3, and 4 contains confidence intervals

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Full specifications of computing resources are provided in 6

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] Yes, the only existing assets we used were publicly available datasets, which we cited in Section **??**

(b) Did you mention the license of the assets? [Yes] We explicitly mention the licenses in Section 5.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Yes, the only new assets created are in the form of code, which is entirely submitted in the supplemental material

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] Yes, both datasets we use are publicly available and we mention the licenses in Section 5.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The two datasets are the MNIST dataset and a mathematical dataset; neither have any personally identifiable information or offensive content.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not use crowdsourcing or conduct research with human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We did not use crowdsourcing or conduct research with human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We did not use crowdsourcing or conduct research with human subjects.

Appendix

1 Proofs

In this section, we present the proof of Theorem 1.

Proof. Following the multi-armed bandit literature, we refer to each feature-threshold pair (f, t) as an arm and refer to its optimization objective μ_{ft} as the arm parameter. Pulling an arm corresponds to evaluating the change in impurity induced by one data point at one feature-threshold pair (f, t) (i.e., arm) and incurs an $O(1)$ computation. This allows us to focus on the number of arm pulls, which translates directly to sample complexity.

First, we show that, with probability at least $1 - \frac{1}{n}$, all confidence intervals computed throughout the algorithm are valid, in that they contain the true parameter μ_{ft}. For a fixed (f, t) and a given iteration of the algorithm, the $(1 - \delta)$ confidence interval satisfies

$$\Pr\left(|\mu_{ft} - \hat{\mu}_{ft}| > C_{ft}\right) \leq \delta.$$

Let B denote the batch size chosen for MABSplit. Note that there are at most $\frac{n}{B}$ rounds in the main `while` loop (Line 6) of Algorithm 1 and hence at most $\frac{nmT}{B} \leq nmT$ confidence intervals computed across all arms and all steps of the algorithm. With $\delta = \frac{1}{n^2 mT}$, we see that $\mu_{ft} \in [\hat{\mu}_{ft} - C_{ft}, \hat{\mu}_{ft} + C_{ft}]$ for every arm (f, t) and for every step of the algorithm with probability at least $1 - \frac{1}{n}$, by a union bound over at most nmT confidence intervals.

Next, we prove the correctness of Algorithm 1. Let $(f^*, t^*) = \arg\min_{f \in \mathcal{F}, t \in \mathcal{T}_f} \mu_{ft}$ be the desired output of the algorithm. Since the main `while` loop in the algorithm can only run $\frac{n}{B}$ times, the algorithm must terminate. Furthermore, if all confidence intervals throughout the algorithm are correct, it is impossible for (f^*, t^*) to be removed from the set of candidate arms. Hence, (f^*, t^*) (or some (f, t) with $\mu_{ft} = \mu_{f^*t^*}$) must be returned upon termination with probability at least $1 - \frac{1}{n}$. This proves the correctness of Algorithm 1.

Finally, we consider the complexity of Algorithm 1. Let n_{used} be the total number of arm pulls computed for each arm remaining in the set of candidate arms at a given point in the algorithm. Notice that, for any suboptimal arm $(f, t) \neq (f^*, t^*)$ that has not left the set of candidate arms, we must have $C_{ft} \leq c_0 \sqrt{\frac{\log 1/\delta}{n_{\text{used}}}}$ by assumption. With $\delta = \frac{1}{n^2 mT}$ as above and $\Delta_{ft} = \mu_{ft} - \mu_{f^*t^*}$, if $n_{\text{used}} > \frac{4c_0^2}{\Delta_{ft}^2} \log(n^2 mT)$ then

$$2(C_{ft} + C_{f^*t^*}) \leq 2c_0 \sqrt{\log(n^2 mT)/n_{\text{used}}} < \Delta_{ft} = \mu_{ft} - \mu_{f^*t^*},$$

and

$$\begin{aligned}
\hat{\mu}_{ft} - C_{ft} &> \mu_{ft} - 2C_{ft} \\
&= \mu_{f^*t^*} + \Delta_{ft} - 2C_{ft} \\
&\geq \mu_{f^*t^*} + 2C_{f^*t^*} \\
&> \hat{\mu}_{f^*t^*} + C_{f^*t^*}
\end{aligned}$$

which means that (f, t) must be removed from the set of candidate arms at the end of that iteration. Hence, the number of data point computations M_{ft} required for any arm $(f, t) \neq (f^*, t^*)$ is at most

$$M_{ft} \leq \min\left[\frac{4c_0^2}{\Delta_{ft}^2} \log(n^2 mT) + B, 2n\right].$$

Notice that this holds simultaneously for all arms (f, t) with probability at least $1 - \frac{1}{n}$. We conclude that the total number of arm pulls M satisfies

$$\begin{aligned}
\mathbb{E}[M] &\leq \mathbb{E}[M | \text{ all confidence intervals are correct}] + \frac{1}{n}(2nMT) \\
&\leq \sum_{f \in \mathcal{F}, t \in \mathcal{T}_f} \min\left[\frac{4c_0^2}{\Delta_{ft}^2} \log(n^2 mT) + B, 2n\right] + 2mT,
\end{aligned}$$

where we used the fact that the maximum number of computations for any arm is $2n$. As argued before, since each arm pull involves an $O(1)$ computation, M also corresponds the total number of computations. $\square$

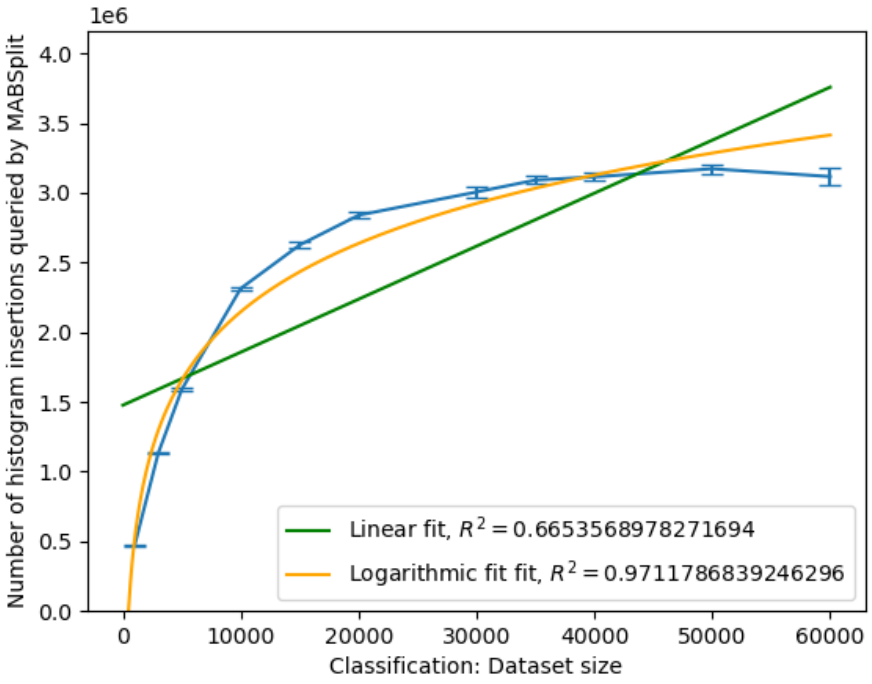
2 $O(\log n)$ scaling of MABSplit

In Theorem 1, we demonstrated that MABSplit scales logarithmically in dataset size. In this section, we empirically validate this claim.

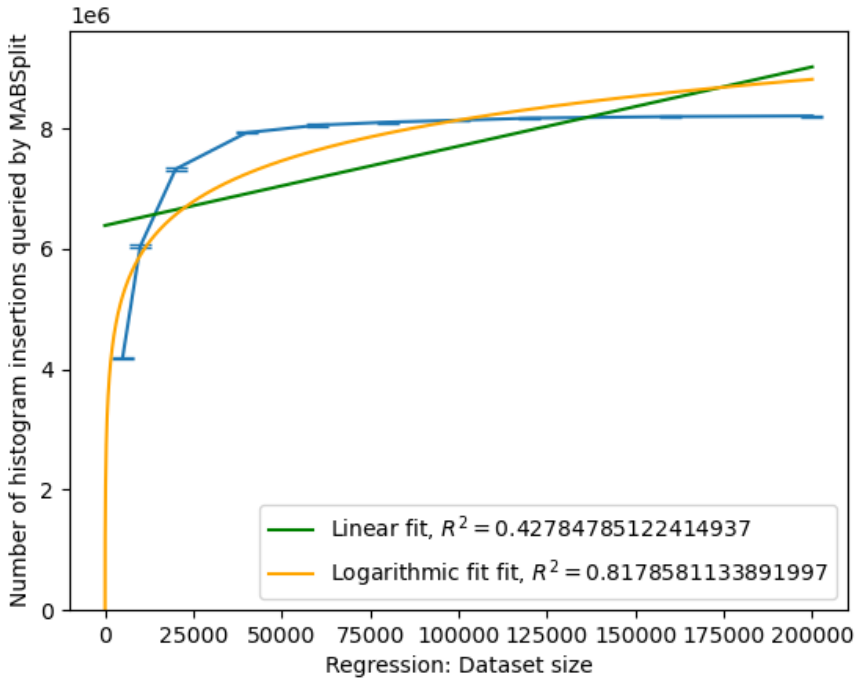
Appendix Figure 1 (a) demonstrates the number of data points queried by MABSplit for a single node split, i.e., a single call to MABSplit, as the dataset size increases, for various subset sizes of MNIST. For each sample size, a sample is drawn with replacement from the original MNIST dataset. The model is trained using Gini impurity in for the usual digitr classification task.

Appendix Figure 1 (b) demonstrates the same plot for various subset sizes of the Random Linear Model dataset. The Random Linear Model dataset consists of 200,000 datapoints with 50 features, 6 of which are correlated with the targets and 44 of which are pure noise, using `scikit-learn`'s `make_regression` function [47]; 160,000 datapoints are used for training and the remaining 40,000 for test.

Appendix Figure 1 also shows the best linear and logarithmic fits to each dataset. The relatively high R^2 values of the logarithmic fits ($R^2 = 0.97$ and $R^2 = 0.82$) compared to that of the linear fits ($R^2 = 0.66$ and $R^2 = 0.43$) suggests that the scaling of MABSplit is logarithmic (and therefore sublinear) with dataset size.



(a)



(b)

Appendix Figure 1: Scaling of MABSplit with different data subset sizes for (a) the MNIST digit classification task and (b) the Random Linear Model regession task. In both tasks, MABSplit appears to scale logarithmically, not linearly, with dataset size.

3 Mean Estimation and Confidence Interval Constructions

In this section, we discuss the estimation of the means μ_{ft} and construction of their confidence intervals via plug-in estimators and the delta method.

Let $p_{\mathrm{L},k}$, $p_{\mathrm{R},k}$, $\hat{p}_{\mathrm{L},k}$, and $\hat{p}_{\mathrm{R},k}$ be the same as defined in Subsection 3.1. Furthermore, let $\mathbf{p} = [p_{\mathrm{L},1}, \cdots, p_{\mathrm{L},K}, p_{\mathrm{R},1}, \cdots, p_{\mathrm{R},K}]^T$ and $\hat{\mathbf{p}} = [\hat{p}_{\mathrm{L},1}, \cdots, \hat{p}_{\mathrm{L},K}, \hat{p}_{\mathrm{R},1}, \cdots, \hat{p}_{\mathrm{R},K}]^T$. Then, $n'\hat{\mathbf{p}}$ follows a multinomial distribution with parameters $(n', 2K, \mathbf{p})$.

Let $\boldsymbol{\theta} = [p_{\mathrm{L},1}, \cdots, p_{\mathrm{L},K}, p_{\mathrm{R},1}, \cdots, p_{\mathrm{R},K-1}]^T$ and $\hat{\boldsymbol{\theta}} = [\hat{p}_{\mathrm{L},1}, \cdots, \hat{p}_{\mathrm{L},K-1}, \hat{p}_{\mathrm{R},1}, \cdots, \hat{p}_{\mathrm{R},K-1}]^T$. Then, by the Central Limit Theorem,

$$\sqrt{n'}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}) \overset{D}{\sim} \mathcal{N}(0, \Sigma), \tag{8}$$

where $\Sigma_{ii} = \theta_i(1-\theta_i)$ and $\Sigma_{ij} = -\theta_i\theta_j$.

Next, we write μ_{ft} in terms of $\boldsymbol{\theta}$ for the impurity metrics as

$$\text{Gini impurity}: \ \mu_{ft}(\boldsymbol{\theta}) = 1 - \frac{\sum_{k=1}^{K}\theta_k^2}{\sum_{k=1}^{K}\theta_k} - \frac{\sum_{k=K+1}^{2K-1}\theta_k^2 + (1-\sum_{k=1}^{2K-1}\theta_k)^2}{1-\sum_{k=1}^{K}\theta_k}, \tag{9}$$

$$\begin{aligned}\text{Entropy}: \ \mu_{ft}(\boldsymbol{\theta}) = &-\sum_{k=1}^{K}\theta_k \log_2 \frac{\theta_k}{\sum_{k'=1}^{K}\theta'_k} - \sum_{k=K+1}^{2K-1}\theta_k \log_2 \frac{\theta_k}{1-\sum_{k'=1}^{K}\theta'_k} - \\ &(1-\sum_{k=1}^{2K-1}\theta_k)\log_2 \frac{(1-\sum_{k=1}^{2K-1}\theta_k)}{1-\sum_{k=1}^{K}\theta_k}.\end{aligned} \tag{10}$$

For a given impurity metric, let $\nabla\mu_{ft}(\boldsymbol{\theta})$ be the derivative of μ_{ft} with respect to $\boldsymbol{\theta}$. From the delta method,

$$\sqrt{n'}(\hat{\mu}_{ft}(\boldsymbol{\theta}) - \mu_{ft}(\boldsymbol{\theta})) \overset{D}{\sim} \mathcal{N}(0, \nabla\mu_{ft}(\boldsymbol{\theta})^T \Sigma \nabla\mu_{ft}(\boldsymbol{\theta})), \tag{11}$$

where the CIs can be constructed accordingly. These CIs are asymptotically valid as $n', n \to \infty$. For other impurity metrics such as MSE, the CIs can be similarly derived by writing the corresponding μ_{ft} in terms of $\boldsymbol{\theta}$ and computing $\nabla\mu_{ft}(\boldsymbol{\theta})$.

Model	Task and Dataset	Performance Metric	Test Performance
RF (ours)	Classification: 20 Newsgroups	Accuracy	$74.1 \pm 2.8\%$
RF (`scikit-learn`)	Classification: 20 Newsgroups	Accuracy	$76.2 \pm 1.7\%$
ExtraTrees (ours)	Classification: 20 Newsgroups	Accuracy	$66.5 \pm 5.1\%$
ExtraTrees (`scikit-learn`)	Classification: 20 Newsgroups	Accuracy	$62.6 \pm 2.8\%$
RF (ours)	Regression: California Housing	MSE	0.679 ± 0.022
RF (`scikit-learn`)	Regression: California Housing	MSE	0.672 ± 0.028
ExtraTrees (ours)	Regression: California Housing	MSE	0.696 ± 0.055
ExtraTrees (`scikit-learn`)	Regression: California Housing	MSE	0.695 ± 0.082

Appendix Table 1: Comparison of our re-implementation of baselines with the the implementations available in `scikit-learn`. No statistically significant differences are apparent, which suggests that our re-implementations are accurate.

4 Comparison of baseline implementations and scikit-learn

In this section, we compare our re-implementation of common baselines to those in popular packages to verify the accuracy of our re-implementation. Specifically, we compare our implementations of Random Forest Classifiers, Random Forest Regressors, Extremely Random Forest Classifiers, and Extremely Random Forest Regressors to those of `scikit-learn`. We omit comparisons of the Random Patches models because their correctness is implied by that of the Random Forest model, as the Random Patches model consists of applying the Random Forest model to subsampled data and features.

For classification, we compare our implementations on the 20 newsgroups dataset filtered to two newsgroups, `alt.atheism` and `sci.space`. The dataset is embedded via TF-IDF and projected onto their top 100 principal components, following standard practice [47]. The train-test split is the standard one provided by `scikit-learn`.

For all classification problems, we average the predicted probabilities of each tree in the forest ("soft voting") as opposed to only allowing each tree to vote for a single class ("hard voting"), following the implementation in `scikit-learn` [47].

For regression, we compare our implementations on the California Housing dataset, subsampled to 1,000 points as performing the regression on the full dataset of approximately 20,000 points is computationally prohibitive. The train-test split is the standard one provided by `scikit-learn` [47].

Table 1 presents our results. In all cases, our re-implemented baselines do not present a statistically significant difference in performance from the models present in `scikit-learn`, which suggests that our re-implementations are correct. Performance is measured over 20 random seeds to compute averages and standard deviations.

5 Profiles

In this work, we focused on the reducing the runtime at the *algorithmic* level, i.e., reducing the complexity of computing the best feature-threshold split. In this section, we justify this choice by demonstrating that most of the time spent in our re-implementation of the baseline algorithms is spent in computing the best feature-threshold split.

Appendix Figure 2 demonstrates the wall-clock time spent inside various functions when fitting a Random Forest classifier without MABSplit on two subsets of the MNIST dataset of sizes 5,000 and 10,000. Most of the time is spent inside the computation of the best feature-threshold split, which scales approximately as dataset size and motivates our focus on improving the performance of the split-identification subroutine. When using MABSplit, the time spent to identify the best feature-threshold split is reduced significantly (Appendix Figure 3).

Appendix Figure 4 also contains an example callgraph demonstrating callers and callees for the fitting procedure of a Random Forest, for easier interpretation of Appendix Figures 2 and 3.

Incl.	Self	Called	Function	Location
28 530 355 227	63 786 876	(0)	fit	forest_base.py
28 463 927 621	3 352 727	5	fit	tree_base.py
28 080 739 176	51 124 826	1 174	calculate_best_split	node.py
28 029 614 350	10 035 680	154	solve_exactly	solvers.py
27 757 770 220	49 001 781	154	sample_targets	solvers.py
25 297 154 221	8 489 709 632	2 790	add	histogram.py
16 709 128 009	13 411 464	2 790	__call__	function_base.py
16 695 716 545	16 563 117 384	2 790	_vectorize_call	function_base.py
2 390 857 717	244 556 769	2 790	get_impurity_reductions	criteria.py
1 694 515 913	1 080 737 273	64 170	get_gini	criteria.py

(a)

Incl.	Self	Called	Function	Location
62 475 103 619	130 137 743	(0)	fit	forest_base.py
62 341 106 050	4 011 875	5	fit	tree_base.py
61 569 426 508	112 705 528	1 320	calculate_best_split	node.py
61 456 720 980	12 413 551	155	solve_exactly	solvers.py
61 137 274 586	78 318 817	155	sample_targets	solvers.py
58 108 915 088	19 636 444 229	3 037	add	histogram.py
38 288 693 888	27 780 631	3 037	__call__	function_base.py
38 260 913 257	38 008 248 809	3 037	_vectorize_call	function_base.py
2 922 821 783	300 147 575	3 037	get_impurity_reductions	criteria.py
2 069 337 645	1 329 814 899	69 851	get_gini	criteria.py

(b)

Appendix Figure 2: Profiles for the node-splitting algorithm using the exact solver/naïve computation, the canonical algorithm for computing the best feature-threshold split, for 5,000 (top) and 10,000 (bottom) data point subsets of MNIST. The "Function" column is the name of the called function, the "Incl." column is the time spent in the function and any called subroutines, and the "Self" column is the time (in nanoseconds) spent in only the function and *not* in any callees. All times are in nanoseconds. When increasing the dataset size, the overhead spent outside of the `solve_exactly` function grows negligibly from about 0.5 seconds to about 1 second. However, the time spent in the `solve_exactly` function and any called subroutines grows from about 28 seconds to about 61 seconds and constitutes approximately 98% of the increase in wall-clock time. This observation motivates our focus on improving the subroutine used to identify the best feature-threshold split. This profile was generated with `cProfile` and visualized with `pyprof2calltree` [37].

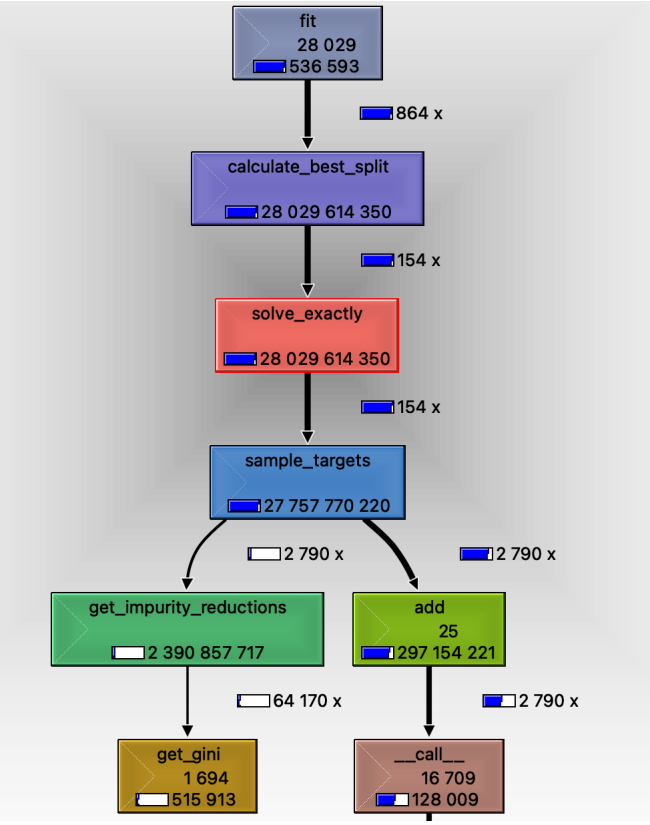
Incl.	Self	Called	Function	Location
19 661 963 096	41 969 883	(0)	fit	forest_base.py
19 617 643 699	3 092 763	5	fit	tree_base.py
19 230 991 254	50 655 813	1 235	calculate_best_split	node.py
19 180 335 441	23 821 486	153	solve_mab	solvers.py
18 876 412 727	54 401 292	171	sample_targets	solvers.py
16 264 631 640	5 516 196 393	2 874	add	histogram.py
10 655 097 887	14 682 830	2 874	__call__	function_base.py
10 640 415 057	10 522 707 847	2 874	_vectorize_call	function_base.py
2 525 591 375	263 120 240	2 874	get_impurity_reductions	criteria.py
1 780 190 442	1 136 355 365	64 900	get_gini	criteria.py

(a)

Incl.	Self	Called	Function	Location
35 191 513 750	146 189 424	(0)	fit	forest_base.py
35 037 137 603	4 275 199	5	fit	tree_base.py
34 213 584 573	148 596 702	1 302	calculate_best_split	node.py
34 064 987 871	33 510 699	155	solve_mab	solvers.py
33 666 053 648	99 185 719	201	sample_targets	solvers.py
30 175 621 314	10 467 512 322	3 286	add	histogram.py
19 550 499 483	28 674 755	3 286	__call__	function_base.py
19 521 824 728	19 318 153 735	3 286	_vectorize_call	function_base.py
3 336 452 440	354 850 749	3 286	get_impurity_reductions	criteria.py
2 340 068 097	1 498 557 797	72 086	get_gini	criteria.py

(b)

Appendix Figure 3: Profiles for the node-splitting algorithm using MABSplit, for 5,000 (top) and 10,000 (bottom) datapoint subsets of MNIST. The "Function" column is the name of the called function, the "Incl." column is the time spent in the function and any called subroutines, and the "Self" column is the time (in nanoseconds) spent in only the function and *not* in any called subroutines. All times are in nanoseconds. When increasing the dataset size, the time spent in the `solve_mab` function and any called subroutines only grows from approximately 20 seconds to approximately 35 seconds to identify the best feature-threshold split. This profile was generated with `cProfile` and visualized with `pyprof2calltree` [37].



Appendix Figure 4: Example call graph of the `fit` subroutine for the forest-based models in our re-implementation when the forest includes a single tree to be split only once. The `fit` method of the forest calls the `fit` method of its only tree, which calls `calculate_best_split` method of the root node, which calls the respective solver (`solve_exactly` for the brute-force algorithm or `solve_exactly` for MABSplit), where the majority of wall-clock time is spent.

6 Experiment Details

Here we provide full details for the experiments in Section 5. All experiments were run on 2021 MacBook Pro running MacOS 12.5.1 (Monterey) with an Apple M1 Max processor, and 64 GB RAM.

6.1 Datasets

Classification Datasets: We use the MNIST [38], APS Failure at Scania Trucks [25, 20], and Forest Cover Type [13, 20] datasets. The MNIST dataset consists of 60,000 training and 10,000 test images of handwritten digits, where each black-and-white image is represented as a 784-dimensional vector and the task is to predict the digit represented by the image. The APS Failure at Scania Trucks dataset consists of 60,000 datapoints with 171 features and the task is to predict component failure. The Forest Covertype dataset consists of 581,012 datapoints with 54 feature and the task is to predict the type the forest cover type from cartographic variables.

Regression Datasets: We use the Beijing Multi-Site Air-Quality [64, 20] and the SGEMM GPU Kernel Performance [8, 44, 20] datasets. The Beijing Multi-Site Air-Quality dataset consists of 420,768 datapoints with 18 features and the task is to predict the level of air pollution. The SGEMM GPU Kernel Performance dataset consists of 241,600 datapoints and the task is to predict the running time of a matrix multiplication.

For all datasets except MNIST (which has predefined training and test datasets), all datasets were randomized into 9:1 train-test splits. All datasets are publicly available.

6.2 Runtime Experiments

For the runtime experiments presented in Tables 1, all performances were measured from 5 random seeds. For all datasets, the maximum depth was set to 1 except for the MNIST dataset, in which the maximum depth was set to 5. The number of trees in each model was set to 5. All experiments used the Gini impurity criterion and the minimum impurity decrease required from performing a split was set to 0.005. For the Random Patches (RP) model, α_n was set to 0.7 and α_f was set to 0.85.

For the regression runtime experiments presented in Table 2, all performances were measured from 5 random seeds. For the Beijing Multi-Site Air-Quality Dataset, the maximum depth was set to 1 and for the SGEMM GPU Kernel Performance Dataset, the maximum number of leaf nodes was set to 5. The number of trees in each model was set to 5. All experiments used the MSE impurity criterion and the minimum impurity decrease required from performing a split was set to 0.005. For the Random Patches (RP) model, α_n was set to 0.7 and α_f was set to 0.85.

6.3 Budget Experiments

For the classification budget experiments presented in Table 3, all performances were measured from 5 random seeds. The budget for each model on the MNIST, APS Failure at Scania Trucks, and Forest Covertype datasets were set to 10,192,000, 784,000, and 9,408,000, respectively. For the Random Patches (RP) model, α_n was set to 0.6 and α_f was set to 0.8. The maximum number of trees in any model was set to 100 and the maximum depth of each tree was set to 5.

For the regression budget experiments presented in Table 4, all performances were measured from 5 random seeds. The budget for each model on the Beijing Multi-Site Air-Quality Dataset was set to 76,800,000 and the budget for each model on the SGEMM GPU Kernel Performance Dataset was set to 24,000,000. For the Random Patches (RP) model, α_n was set to 0.8 and α_f was set to 0.5. The maximum number of trees in any model was set to 100 and the maximum depth of each tree was set to 5.

6.4 Stability Experiments

Two metrics for calculating feature importance are used in Table 5: out-of-bag Permutation Importance (OOB PI) and Mean Decrease in Impurity (MDI) [42, 49]. For a feature f, the OOB PI is calculated by measuring the difference between the trained model's out-of-bag error on the original data with its out-of-bag error on all the data with all out-of-bag datapoints' f values shuffled. The

MDI for a feature f is the average decrease in impurity of all nodes where f is selected as the splitting criterion.

Once feature importances have been calculated, the top k most important features for the model are selected and the stability of these k features is measured via standard stability formulas [43].

The results of the stability experiments are shown in Table 5. The Random Classification dataset is generated via `scikit-learn`'s `datasets.make_classification` function with `n_samples=10000`, `n_features=60`, and `n_informative=5`. The Random Regression dataset is generated by `scikit-learn`'s `datasets.make_regression` with `n_samples=10000`, `n_features=100`, and `n_informative=5`.

MNIST Dataset (Classification, $N = 60,000$, maximum depth $= 8$)		
Model	Wall-clock Training Time (s)	Accuracy (%)
`scikit-learn` Decision Tree Classifier	34.665 ± 1.266	91.061 ± 0.0
Histogrammed decision tree (Exact solver, ours)	86.514 ± 2.839	90.923 ± 0.0
Histogrammed decision tree (MABSplit solver, ours)	**8.538 ± 0.079**	**90.629 ± 0.234**

Appendix Table 2: Comparison of accuracy and wall-clock training time of `scikit-learn`'s Decision Tree Classifier with our implementation on the MNIST digit classification task. Our implementation of the histogrammed decision tree is slower than `scikit-learn`'s, but our optimized implementation is about 4x faster than `scikit-learn`'s. The slight performance degradation is likely due to discretization of the data during histogramming; this effect is also seen when histogramming the data and using the exact solver (i.e., when not using MABSplit). A more heavily optimized version of our histogrammed decision tree when using MABSplit would likely result in even lower training times. Performance was measured over 5 random seeds.

7 Limitations

7.1 Theoretical Limitations

Crucial to the success of MABSplit are the assumptions described before and after Theorem 1. In particular, we assume that their is reasonable heterogeneity amongst the true impurity reductions of different feature-value splits. Such assumptions are common in the literature and have been validated on many real-world datasets [5, 63, 7, 58, 4, 6].

We also note that the assumptions that each CI scales as $\sqrt{\frac{\log 1/\delta}{n'}}$ may be violated when using certain impurity metrics. For example, the derivative of the entropy impurity criterion with respect to some p_k approaches ∞ when $p_k \to 0$. In this case, we cannot apply the delta method from Appendix 3 to compute finite CIs that scale in the way we require. In such settings, it may be necessary to compute the CIs in other ways, e.g., following [46] or [9].

We note that in the worst case, even when all assumptions are violated, MABSplit is never worse than the naïve algorithm in terms of sample complexity. In the worst case, it is a batched version of the naïve algorithm.

7.2 Practical Limitations

We note that MABSplit may perform worse than naïve node-splitting on very small datasets, where the overhead of sampling the data in batches outweighs any potential benefits in sample complexity (see Appendix 8 for further discussion).

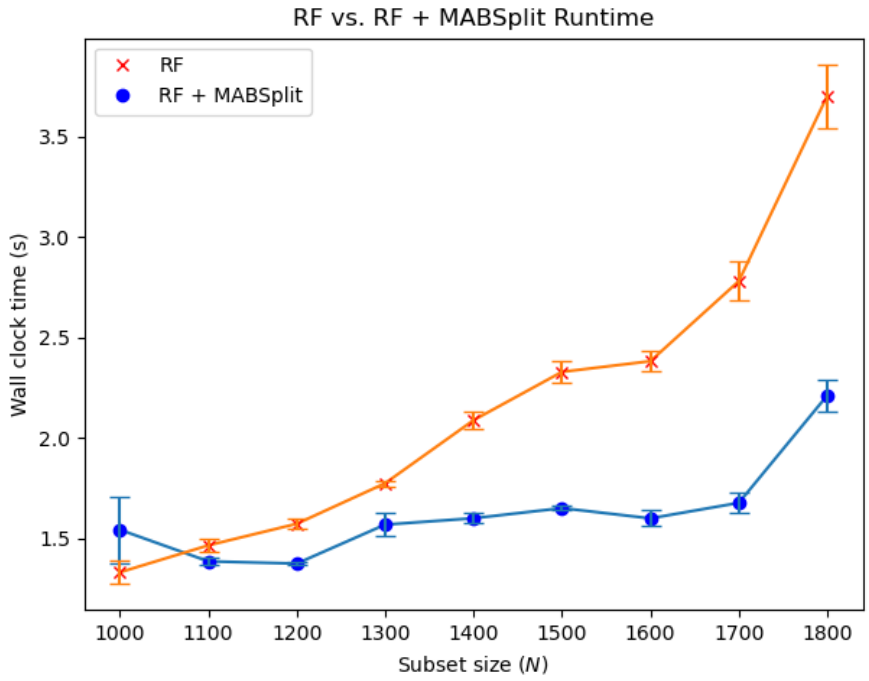
In this work, we avoided a direct runtime comparison with `scikit-learn` because `scikit-learn` utilizes a number of low-level implementation optimizations that would make the comparison unfair. To provide a brief comparison to the popular `scikit-learn` implementation, however, we attempted to optimize our implementation using `numba` [36], a package that translates Python code to optimized machine code. Our `numba`-optimized implementation is 4x faster than `scikit-learn`'s `DecisionTreeClassifier` and achieves comparable performance on the MNIST dataset; see Appendix Table 2.

In order for practitioners to take full advantage of MABSplit, however, it may be necessary to implement MABSplit within the `scikit-learn` library. In doing so, it may be possible that MABSplit makes it difficult or impossible to use existing optimizations in the `scikit-learn` library. An example of this is vectorization: because the naïve node-splitting algorithm queries the data in a predictable way, each datapoint can be queried more quickly than in MABSplit. Despite MABSplit's advantages in sample complexity, the disadvantages of being unable to use implementation optimizations like vectorization may outweigh MABSplit's benefits. Many of these risks may be ameliorated by addressed MABSplit into existing RF implementations such as the one in `scikit-learn`. We anticipate that many optimizations will still apply: for example pre-fetching data to have it in caches close to the CPU, manual loop unrolling, etc. We leave an optimization implementation of MABSplit inside the `scikit-learn` library to future work.

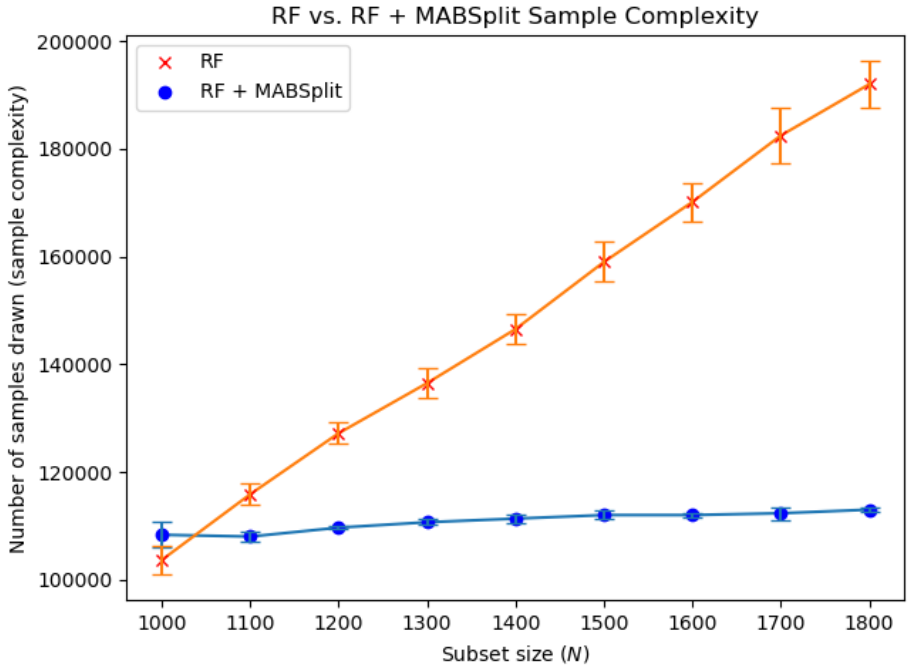
8 Comparison on Small Datasets

In this section, we investigate the performance of MABSplit on small datasets. Appendix Figure 5 demonstrates the performance of MABSplit, both in wall-clock training time and sample complexity, for various subset sizes of MNIST. Our results that RF+MABSplit outperforms the standard RF algorithm, in both sample complexity and wall-clock time, when the dataset size exceeds approximately 1100 datapoints.

However, we also note that the main use case for MABSplit is when the data size is large and it is computationally challenging to run standard forest-based algorithms. Indeed, the use of big data in many applications that necessitate sampling was the primary motivation for our work [11, 54, 18, 57, 41, 1].



(a)



(b)

Appendix Figure 5: (a) Wall-clock training times and (b) sample complexities of a random forest model with and without MABSPlit, for various subset sizes of MNIST. For dataset sizes below approximately 1000, the exact random forest model performs better in terms of sample complexity and wall-clock time. Above 1100 datapoints, the MABSplit version demonstrates better sample complexity and wall-clock time. Error bars were computed over 3 random seeds. Test performances were not different at a statistically significant level.

9 Description of Other Node-Splitting Algorithms

For completeness, we provide a brief description of various baseline models' node-splitting algorithms here to enable easier comparison with MABSplit.

Consider a node with n datapoints each with m features, and T possible thresholds at which to split each feature. We discuss the classification setting for simplicity, though the same arguments apply to regression.

A very naïve approach would be to iterate over all mT feature-value splits, and compute the probabilities $p_{L,k}$ and $p_{R,k}$ from all n datapoints. This results in complexity $O(mTn)$, which is $O(mn^2)$ when $T = n$ (for example, $T = n$ in the un-histogrammed setting).

Instead, the usual RF algorithm sorts all n datapoints in $O(n\log n)$ time for each of the m features, resulting in total computational cost $O(mn\log n)$. Then the algorithm scans linearly from lowest value to highest value for each feature and update the parameters $p_{L,k}$ and $p_{R,k}$ via simple counting to find the best impurity reduction for each of the T potential splits. The complexity of this step is $O(mT + mn)$, where the "$+mn$" comes from the allocations of each data point to the left or right node during the scan (each data point is re-allocated only once per feature). Thus the total complexity of this approach is $O(mn\log n + mT + mn) = O(mn\log n + mT)$. This is $O(mn\log n)$ when $T = n$.

The binned (a.k.a. histogrammed) method does not require the per-feature sort and avoids the $O(mn\log n)$ computation when $T < n$,. Instead, each of the n points must be inserted into the correct bin (which can be done in $O(1)$ time for each datapoint if the bins are equally spaced) for each of the m features, incurring total computational cost $O(mn)$. Then, the same linear scanning approach as in the "standard" algorithm is performed with complexity $O(mT + mn)$. The total complexity of this approach is $O(mn + mT + mn) = O(m(n + T))$. This is $O(mn)$ when $T = n$.

In general, we do not assume $T = n$, i.e., that every feature value is a potential split point, unless otherwise specified. In our paper, the "standard" approach refers to the **un**binned approach which requires an $O(mn\log n)$ sort and "linear" refers to the binned approach that is $O(m(n + T))$, which is $O(mn)$ when $T = O(n)$.

Crucially, when $T = o(N)$ (as is often the case in practice, e.g., for a constant number of bins) and the necessary gap assumptions are satisfied, MABSplit scales as $O(mT\log n)$. In many cases, this is much better than $O(m(n + T))$, e.g., for large datasets, because the dependence on n is reduced from linear to logarithmic. More concretely, treating T as a constant and ignoring the dependence on m, we reduce the complexity of the binned algorithm from $O(n)$, what we refer to as "linear," to $O(\log n)$.