Machine Unlearning for Random Forests

Jonathan Brophy [1] **Daniel Lowd** [1]

Abstract

Responding to user data deletion requests, removing noisy examples, or deleting corrupted training data are just a few reasons for wanting to delete instances from a machine learning (ML) model. However, efficiently removing this data from an ML model is generally difficult. In this paper, we introduce data removal-enabled (DaRE) forests, a variant of random forests that enables the removal of training data with minimal retraining. Model updates for each DaRE tree in the forest are exact, meaning that removing instances from a DaRE model yields exactly the same model as retraining from scratch on updated data.

DaRE trees use randomness and caching to make data deletion efficient. The upper levels of DaRE trees use random nodes, which choose split attributes and thresholds uniformly at random. These nodes rarely require updates because they only minimally depend on the data. At the lower levels, splits are chosen to greedily optimize a split criterion such as Gini index or mutual information. DaRE trees cache statistics at each node and training data at each leaf, so that only the necessary subtrees are updated as data is removed. For numerical attributes, greedy nodes optimize over a random subset of thresholds, so that they can maintain statistics while approximating the optimal threshold. By adjusting the number of thresholds considered for greedy nodes, and the number of random nodes, DaRE trees can trade off between more accurate predictions and more efficient updates.

In experiments on 13 real-world datasets and one synthetic dataset, we find DaRE forests delete data orders of magnitude faster than retraining from scratch while sacrificing little to no predictive power.

[1]Department of Computer and Information Science, University of Oregon, Eugene, Oregon. Correspondence to: Jonathan Brophy <jbrophy@cs.uoregon.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

1. Introduction

Recent legislation (EU, 2016; California, 2018; Canada, 2018) requiring companies to remove private user data upon request has prompted new discussions on data privacy and ownership (Shintre et al., 2019), and fulfilling this "right to be forgotten" (Kwak et al., 2017; Garg et al., 2020) may require updating any models trained on this data (Villaronga et al., 2018). However, retraining a model from scratch on a revised dataset becomes prohibitively expensive as dataset sizes and model complexities increase (Shoeybi et al., 2019); the result is wasted time and computational resources, exacerbated as the frequency of data removal requests increases.

Decision trees and random forests (Breiman et al., 1984; Friedman, 2001) are popular and widely used machine learning models (Lundberg et al., 2018), mainly due to their predictive prowess on many classification and regression tasks (Kocev et al., 2013; Genuer et al., 2017; Wager & Athey, 2018; Linero & Yang, 2018; Biau et al., 2019). Current work on deleting data from machine learning models has focused mainly on recommender systems (Cao & Yang, 2015; Schelter, 2020), K-means (Ginart et al., 2019), SVMs (Cauwenberghs & Poggio, 2001), logistic regression (Guo et al., 2020; Schelter, 2020), and deep neural networks (Baumhauer et al., 2020; Golatkar et al., 2020b; Wu et al., 2020); however, there is very limited work addressing the problem of efficient data deletion for tree-based models (Schelter et al., 2021). Thus, we outline our contributions as follows:

1. We introduce DaRE (**Da**ta **R**emoval-**E**nabled) Forests (a.k.a DaRE RF), a variant of random forests that supports the efficient removal of training instances. DaRE RF works with discrete tree structures, in contrast to many related works on efficient data deletion that assume continuous parameters. The key components of DaRE RF are to retrain subtrees only as needed, consider only a subset of valid thresholds per attribute at each decision node, and to strategically place completely random nodes near the top of each tree to avoid costly retraining.
2. We provide algorithms for training and subsequently removing data from a DaRE forest.
3. We evaluate DaRE RF's ability to efficiently perform

sequences of deletions on 13 real-world binary classification datasets and one synthetic dataset, and find that DaRE RF can typically delete data 2-4 orders of magnitude faster than retraining from scratch while sacrificing less than 1% in terms of predictive performance.

2. Problem Formulation

We assume an instance space $\mathcal{X} \subseteq \mathbb{R}^p$ and possible labels $\mathcal{Y} = \{+1, -1\}$[1]. Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ be a training dataset in which each instance $x_i \in \mathcal{X}$ is a p-dimensional vector $(x_{i,j})_{j=1}^p$ and $y_i \in \mathcal{Y}$. We refer to $P = \{j\}_{j=1}^p$ as the set of possible attributes.

2.1. Unlearning

Our goal is to "unlearn" specific training examples by updating a trained model to completely remove their influence. We base our definition on prior work by Ginart et al. (2019, Def. 3.1). We define a (possibly randomized) *learning algorithm*, $\mathcal{A} : \mathcal{D} \to \mathcal{H}$, as a function from a dataset $\mathcal{D}$ to a model in hypothesis space $\mathcal{H}$. A *removal method*, $\mathcal{R} : \mathcal{A}(\mathcal{D}) \times \mathcal{D} \times (\mathcal{X} \times \mathcal{Y}) \to \mathcal{H}$, is a function from a model $\mathcal{A}(\mathcal{D})$, dataset $\mathcal{D}$, and an instance to remove from the training data (x, y) to a model in $\mathcal{H}$. For *exact unlearning* (a.k.a. *perfect unlearning*), the removal method must be equivalent to applying the training algorithm to the dataset with instance (x, y) removed. In the case of randomized training algorithms, we define equivalence as having identical probabilities for each model in $\mathcal{H}$:

$$P(\mathcal{A}(\mathcal{D} \setminus (x, y))) = P(\mathcal{R}(\mathcal{A}(\mathcal{D}), \mathcal{D}, (x, y))) \quad (1)$$

See §5 for more related work on unlearning.

The simplest approach to exact unlearning is to ignore the existing model and simply rerun $\mathcal{A}$ on the updated dataset, $\mathcal{D} \setminus (x, y)$. We refer to this as the *naive retraining* approach. Naive retraining is agnostic to virtually all machine learning models, easy to understand, and easy to implement. However, this approach becomes prohibitively expensive as the dataset size, model complexity, and number of deletion requests increase.

2.2. Random Forests

A *decision tree* is a tree-structured model in which each leaf is associated with a binary-valued prediction and each internal node is a decision node associated with an attribute $a \in P$ and threshold value $v \in \mathbb{R}$. The outgoing branches of the decision node partition the data based on the chosen attribute and threshold. Given $x \in \mathcal{X}$, the prediction of a decision tree can be found by traversing the tree, starting at the root and following the branches consistent with the attribute values in x. Traversal ends at one of the leaf nodes, where the prediction is equal to the value of the leaf node.

Decision trees are typically learned in a recursive manner, beginning by picking an attribute a and threshold v at the root that optimizes an empirical split criterion such as the Gini index (Breiman et al., 1984):

$$G_{D,\mathcal{Y}}(a, v) = \sum_{b \in \{\ell, r\}} \frac{|D_b|}{|D|} \left(1 - \sum_{y \in \mathcal{Y}} \left(\frac{|D_{b,y}|}{|D_b|}\right)^2\right) \quad (2)$$

or entropy (Quinlan, 2014):

$$H_{D,\mathcal{Y}}(a, v) = \sum_{b \in \{\ell, r\}} \frac{|D_b|}{|D|} \left(\sum_{y \in \mathcal{Y}} -\frac{|D_{b,y}|}{|D_b|} \log_2 \frac{|D_{b,y}|}{|D_b|}\right) \quad (3)$$

in which $D \subseteq \mathcal{D}$ is the input dataset to a decision node, $D_\ell = \{(x_i, y_i) \in D \mid x_{i,a} \leq v\}$, $D_r = D \setminus D_\ell$, $D_{\ell,y} = \{(x_i, y_i) \in D_\ell \mid y_i = y\}$, and $D_{r,y} = \{(x_i, y_i) \in D_r \mid y_i = y\}$. Once a and v have been chosen for the root node, the data is partitioned into mutually exclusive subsets based on the value of v, and a child node is learned for each data subset. The process terminates when the entire subset has the same label or the tree reaches a specified maximum depth $d_{\max}$.

A *random forest* (RF) is an ensemble of decision trees which predicts the mean value of its component trees. Two sources of randomness are used to increase diversity among the trees. First, each tree in the ensemble is trained from a bootstrap sample of the original training data, with some instances excluded and some included multiple times. Second, each decision node is restricted to a random subset of attributes, and the split criterion is optimized over this subset rather than over all attributes.

We base our methods on a minor variation of a standard RF, one that does not use bootstrapping. Bootstrapping complicates the removal of training instances, since one instance may appear multiple times in the training data for one tree. There is also empirical evidence that bootstrapping does not improve predictive performance (Zaman & Hirose, 2009; Denil et al., 2014; Mentch & Hooker, 2016), which was consistent with our own experiments (Appendix: §B.2, Table 5). Since predictive performance was already similar, we saw no need to add the extra bookkeeping to handle this complexity.

3. DaRE Forests

We now describe DaRE (**Da**ta **R**emoval-**E**nabled) forests (a.k.a. DaRE RF), an RF variant that enables the efficient removal of training instances.

[1] Our methods can easily be generalized to the multi-class setting, $|\mathcal{Y}| = C$, by storing statistics for $C - 1$ classes instead of just one.

Theorem 3.1. *Data deletion for DaRE forests is* exact *(see Eq. 1), meaning that removing instances from a DaRE model yields exactly the same model as retraining from scratch on updated data.*

This is also equivalent to certified removal (Guo et al., 2020) with $\epsilon = 0$. Proofs to all theorems are in §A of the Appendix.

A DaRE forest is a tree ensemble in which each tree is trained independently on a copy of the training data, considering a random subset of $\tilde{p}$ attributes at each split to encourage diversity among the trees. In our experiments we use $\tilde{p} = \lfloor \sqrt{p} \rfloor$. Since each tree is trained independently, we describe our methods in terms of training and updating a single tree; the extension to the ensemble is trivial.

DaRE forests leverage several techniques to make deletions efficient: (1) only retrain portions of the model where the structure must change to match the updated database; (2) consider at most k randomly-selected thresholds per attribute; (3) introduce random nodes at the *top* of each tree that minimally depend on the data and thus rarely need to be retrained. We present abridged versions for training and updating a DaRE tree in Algorithms 1 and 2, respectively, with full explanations below. Detailed pseudocode for both operations is in the Appendix, §A.8.

3.1. Retraining Minimal Subtrees

We avoid unnecessary retraining by storing statistics at each node in the tree. For decision nodes, we store and update counts for the number of instances $|D|$ and positives instances $|D_{\cdot,1}|$, as well as $|D_\ell|$ and $|D_{\ell,1}|$ for a set of k thresholds per attribute. This information is sufficient to recompute the split criterion of each threshold without iterating through the data. For leaf nodes, we store and update $|D|$ and $|D_{\cdot,1}|$, along with a list of training instances that end at that leaf. These statistics are initialized when training the tree for the first time (Alg. 1). We find this additional overhead has a negligible effect on training time.

When deleting a training instance $(x, y) \in D$, these statistics are updated and used to check if a particular subtree needs retraining. Specifically, decision nodes affected by the deletion of (x, y) update the statistics and recompute the split criterion for each attribute-threshold pair. If a different threshold obtains an improved split criterion over the currently chosen threshold, then we retrain the subtree rooted at this node. The training data for this subtree can be found by concatenating the instance lists from all leaf-node descendants. If no retraining occurs at any decision node and a leaf node is reached instead, its label counts and instance list are updated and the deletion operation is complete. See Alg. 2 for pseudocode.

Algorithm 1 Building a DaRE tree / subtree.

Input: data D, depth d
if stopping criteria reached **then**
  $node \leftarrow$ LEAFNODE()
  save instance counts($node$, D) ▷ $|D|$, $|D_{\cdot,1}|$
  save leaf-instance pointers($node$, D)
  compute leaf value($node$)
else
  if $d < d_{\text{rmax}}$ **then**
    $node \leftarrow$ RANDOMNODE()
    save instance counts($node$, D) ▷ $|D|$, $|D_{\cdot,1}|$
    $a \leftarrow$ randomly sample attribute(D)
    $v \leftarrow$ randomly sample threshold $\in [a_{min}, a_{max})$
    save threshold statistics($node$, D, a, v)
  else
    $node \leftarrow$ GREEDYNODE()
    save instance counts($node$, D) ▷ $|D|$, $|D_{\cdot,1}|$
    $A \leftarrow$ randomly sample $\tilde{p}$ attributes(D)
    for $a \in A$ **do**
      $C \leftarrow$ get valid thresholds(D, a)
      $V \leftarrow$ randomly sample k valid thresholds(C)
      for $v \in V$ **do**
        save threshold statistics($node$, D, a, v)
    $scores \leftarrow$ compute split scores($node$)
    select optimal split($node$, $scores$)
  $D.\ell, D.r \leftarrow$ split on selected threshold($node$, D)
  $node.\ell$ = TRAIN(D_ℓ, $d+1$) ▷ Alg. 1
  $node.r \leftarrow$ TRAIN(D_r, $d+1$) ▷ Alg. 1
Return node

3.2. Sampling Valid Thresholds

The optimal threshold for a continuous attribute will always lie between two training instances with adjacent feature values containing opposite labels; if the two training instances have the same label, the split criterion improves by increasing or decreasing v. We refer to these as *valid* thresholds, and any other threshold as *invalid*. More precisely, a threshold v between two adjacent values v_1 and v_2 for a given attribute a is valid if and only if there exist instances (x_1, y_1) and (x_2, y_2) such that $x_{1,a} = v_1$, $x_{2,a} = v_2$, and $y_1 \neq y_2$.

Only considering valid thresholds substantially reduces the statistics we need to store and compute at each node. We gain further efficiency by randomly sampling k valid thresholds and only considering these thresholds when deciding which attribute-threshold pair to split on. We treat k as a hyperparameter and tune its value when building a DaRE model. One might suspect that only considering a subset of thresholds for each attribute may lead to decreased predictive performance; however, our experiments show that relatively modest values of k (e.g. $5 \leq k \leq 25$) are sufficient to providing accurate predictions, and in some cases lead to improved performance (Appendix: §B.2, Table 5).

When deleting an instance at a given node, we must determine if any threshold has become invalid. To accomplish this efficiently, at each node we also save and update the number of instances in which attribute a equals v_1, the

Algorithm 2 Deleting a training instance from a DaRE tree.

Require: Start at the root node.
Input: $node$, depth d, instance to remove (x, y).
update instance counts($node$, (x, y)) ▷ $|D|$ and $|D_{\cdot,1}|$
if $node$ is a LEAFNODE **then**
  remove (x, y) from leaf-instance pointers($node$, (x, y))
  recompute leaf value($node$)
  remove (x, y) from database and return
else
  update decision node statistics($node$, (x, y))
  if $node$ is a RANDOMNODE **then**
    if $node$.selected threshold is invalid **then**
      $D \leftarrow$ get data from leaf instances($node$) $\setminus (x, y)$
      if $node$.selected attribute (a) is not constant **then**
        $v \leftarrow$ resample threshold $\in [a_{min}, a_{max})$
        $D.\ell, D.r \leftarrow$ split on new threshold($node$, D, a, v)
        $node.\ell, r \leftarrow$ TRAIN($D.\ell, d{+}1$), TRAIN($D.r, d{+}1$)
      else
        $node \leftarrow$ TRAIN(D, d) ▷ Alg. 1
      remove (x, y) from database and return
  else
    if $\exists$ invalid attributes or thresholds **then**
      $D \leftarrow$ get data from leaf instances($node$) $\setminus (x, y)$
      resample invalid attributes and thesholds($node$, D)
    $scores \leftarrow$ recompute split scores($node$)
    $a, v \leftarrow$ select optimal split($node$, $scores$)
    if optimal split has changed **then**
      $D.\ell, D.r \leftarrow$ split on new threshold($node$, D, a, v)
      $node.\ell, r \leftarrow$ TRAIN($D.\ell, d+1$), TRAIN($D.r, d+1$)
      remove (x, y) from database and return
  if $x_{\cdot,a} \leq v$ **then**
    DELETE($node.\ell$, $d+1$, (x, y)) ▷ Alg. 2
  else
    DELETE($node.r$, $d+1$, (x, y)) ▷ Alg. 2

number in which a equals v_2, and the number of positive instances matching each of those criteria. When an attribute threshold becomes invalid, we sort and iterate through the node data D, resampling the invalid threshold to obtain a new valid threshold.

3.3. Random Splits

The third technique for efficient model updating is to choose the attribute and threshold for some of the decision nodes at random, independent of the split criterion. Specifically, given the data at a particular decision node $D \subseteq \mathcal{D}$, we sample an attribute $a \in P$ uniformly at random, and then sample a threshold v in the range $[a_{\min}, a_{\max})$, the min. and max. values for a in D. We henceforth refer to these decision nodes as "random" nodes, in contrast to the "greedy" decision nodes that optimize the split criterion. Random nodes store and update $|D_\ell|$ and $|D_r|$, statistics based on the sampled threshold, and retrain only if $|D_\ell| = 0$ or $|D_r| = 0$ (i.e. v is no longer in the range $[a_{\min}, a_{\max})$); however, since random nodes minimally depend on the statistics of the data, they rarely need to be retrained. Random nodes are placed in the upper layers of the tree and greedy nodes are used for all other layers (excluding leaf nodes). We introduce d_{rmax} as another hyperparameter indicating how many layers from the top the tree should use for random nodes (e.g. the top two layers of the tree are all random nodes if $d_{\text{rmax}} = 2$).

Intuitively, nodes near the top of the tree contain more instances than nodes near the bottom, making them more expensive to retrain if necessary. Thus, we can significantly increase deletion efficiency by replacing those nodes with random ones. We can also maintain comparable predictive performance to a model with no random nodes by using greedy nodes in all subsequent layers, resulting in a greedy model built on top of a random projection of the input space (Haupt & Nowak, 2006).

In our experiments, we compare DaRE RF with random splits to those without, to evaluate the benefits of adding these random nodes. We refer to DaRE models with random nodes as random DaRE (R-DaRE) and those without as greedy DaRE (G-DaRE). G-DaRE RF can also be viewed as a special case of R-DaRE RF in which $d_{\text{rmax}} = 0$.

3.4. Complexity Analysis

The time for training a DaRE forest is *identical* to that of a standard RF:

Theorem 3.2. *Given $n = |\mathcal{D}|$, T, $d_{\max}$, and $\tilde{p}$, the time complexity to train a DaRE forest is $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$.*

The overhead of storing statistics and instance pointers is negligible compared to the cost of iterating through the entire dataset to score all attributes at each node. The key difference is in the deletion time, which can be much better depending on how much of each tree needs to be retrained:

Theorem 3.3. *Given $d_{\max}$, $\tilde{p}$, and k, the time complexity to delete a single instance $(x, y) \in \mathcal{D}$ from a DaRE tree is $\mathcal{O}(\tilde{p}\,k\,d_{\max})$, if the tree structure is unchanged and the attribute thresholds remain valid. If a node with $|D|$ instances has invalid attribute thresholds, then the additional time to choose new thresholds is $\mathcal{O}(|D| \log |D|)$. If a node with $|D|$ instances at level d needs to be retrained, then the additional retraining time is $\mathcal{O}(\tilde{p}\,|D|\,(d_{\max} - d))$.*

When the structure is unchanged, this is much more efficient than naive retraining, especially if the number of thresholds considered (k) is much smaller than n. In the worst case, if the split changes at the root of every tree, then deletion in a DaRE forest is no better than naive retraining. In practice, this is very unlikely, since different trees in the forest consider different sets of $\tilde{p}$ attributes at the root, and the difference between the best and second-best attribute-threshold pairs is usually bigger than a single instance.

Choosing new thresholds also requires iterating through the training instances at a node. Thresholds only become invalid when an instance adjacent to the threshold is removed, so

this is an infrequent event when k is much smaller than n. To analyze this empirically, we evaluate our methods with both random and adversarially chosen deletions, approximating the average- and worst-case, respectively.

The main storage costs for a DaRE forest come from storing sets of attribute-threshold statistics at each greedy node, and the instance lists for the leaf nodes.

Theorem 3.4. *Given $n = |\mathcal{D}|$, $d_{\max}$, k, T, and $\tilde{p}$, the space complexity of a DaRE forest is $\mathcal{O}(k\,\tilde{p}\,2^{d_{\max}}\,T + n\,T)$.*

In our experiments, we analyze the space overhead of a DARE forest by measuring its memory consumption as compared to a standard RF, quantifying the time/space trade-off introduced by DARE RF to enable efficient data deletion.

4. Experimental Evaluation

Research Questions Can we use G-DaRE RF to efficiently delete a significant number of instances as compared to naive retraining (**RQ1**)? Can we use R-DaRE RF to further increase deletion efficiency while maintaining comparable predictive performance (**RQ2**)?

Datasets We conduct our experiments on 13 publicly-available datasets that represent problems well-suited for tree-based models, and one synthetic dataset we call Synthetic. For each dataset, we generate one-hot encodings for any categorical variable and leave all numeric and binary variables as is. For any dataset without a designated train and test split, we randomly sample 80% of the data for training and use the rest for testing. A summary of the datasets is in Table 1, and additional dataset details are in the Appendix: §B.1.

Hyperparameter Tuning Due to the range of label imbalances in our datasets (Table 1 and Appendix: §B.1, Table 4) we measure the predictive peformance of our models using average precision (AP) (Zhu, 2004) for datasets with a positive label percentage $< 1\%$, AUC (Hanley & McNeil, 1982) for datasets between [1%, 20%], and accuracy (acc.) for the remaining datasets. Using these metrics and Gini index as the split criterion, we tune the following hyperparameters: the maximum depth of each tree $d_{\max}$, the number of trees in the forest T, and the number of thresholds considered per attribute for greedy nodes k. Our protocol for tuning d_{rmax} is as follows: first, we tune a greedy model (i.e. by keeping $d_{\text{rmax}} = 0$ fixed) using 5-fold cross-validation. Once the optimal values for $d_{\max}$, T, and k are found, we tune d_{rmax} by incrementing its value from zero to $d_{\max}$, stopping when the model's cross-validation score exceeds a specified error tolerance as compared to the greedy model; for these experiments, we tune d_{rmax} using absolute error tolerances of 0.1%, 0.25%, 0.5%, and 1.0%. Selected hyperparameter values are in the Appendix: §B.2, Table 6.

Table 1. Dataset Summary. n = no. instances, p = no. attributes, Pos. % = positive label percentage, Met. = predictive performance metric.

Dataset	n	p	**Pos. %**	**Met.**
Surgical	14,635	90	25.2%	Acc.
Vaccine	26,707	185	46.4%	Acc.
Adult	48,842	107	23.9%	Acc.
Bank Mktg.	41,188	63	11.3%	AUC
Flight Delays	100,000	648	19.0%	AUC
Diabetes	101,766	253	46.1%	Acc.
No Show	110,527	99	20.2%	AUC
Olympics	206,165	1,004	14.6%	AUC
Census	299,285	408	6.2%	AUC
Credit Card	284,807	29	0.2%	AP
CTR	1,000,000	13	2.9%	AUC
Twitter	1,000,000	15	17.0%	AUC
Synthetic	1,000,000	40	50.0%	Acc.
Higgs	11,000,000	28	53.0%	Acc.

4.1. Methodology

We measure relative efficiency or speedup as the number of instances a DaRE model deletes in the time it takes the naive retraining approach to delete one instance (i.e. retrain without that instance); the number of instances deleted gives us the speedup over the the naive approach.[2] We also measure the predictive performance of R-DaRE RF prior to deletion and compare its predictive performance to that of G-DaRE RF. Each experiment is repeated five times.

We determine the order of deletions using two different adversaries: *Random* and *Worst-of-1000*. The random adversary selects training instances to be deleted uniformly at random, while the worst-of-1000 adversary selects each instance by first selecting 1,000 candidate instances uniformly at random, and then choosing the instance that results in the most retraining, as measured by the total number of instances assigned to all retrained nodes across all trees.

4.2. Deletion Efficiency Results

Random Adversary We present the results of the deletion experiments using the random adversary in Figure 1 (top). We find that G-DaRE RF is usually at least two orders of magnitude faster than naive retraining, while R-DaRE RF is faster than G-DaRE RF to a varying degree depending on the dataset and error tolerance. R-DaRE RF is also able to maintain comparable predictive performance to G-DaRE RF, typically staying within a test error difference of 1% depending on which tolerance is used to tune d_{rmax} (Figure 1: bottom).

[2] System hardware specifications are in the Appendix: §B.

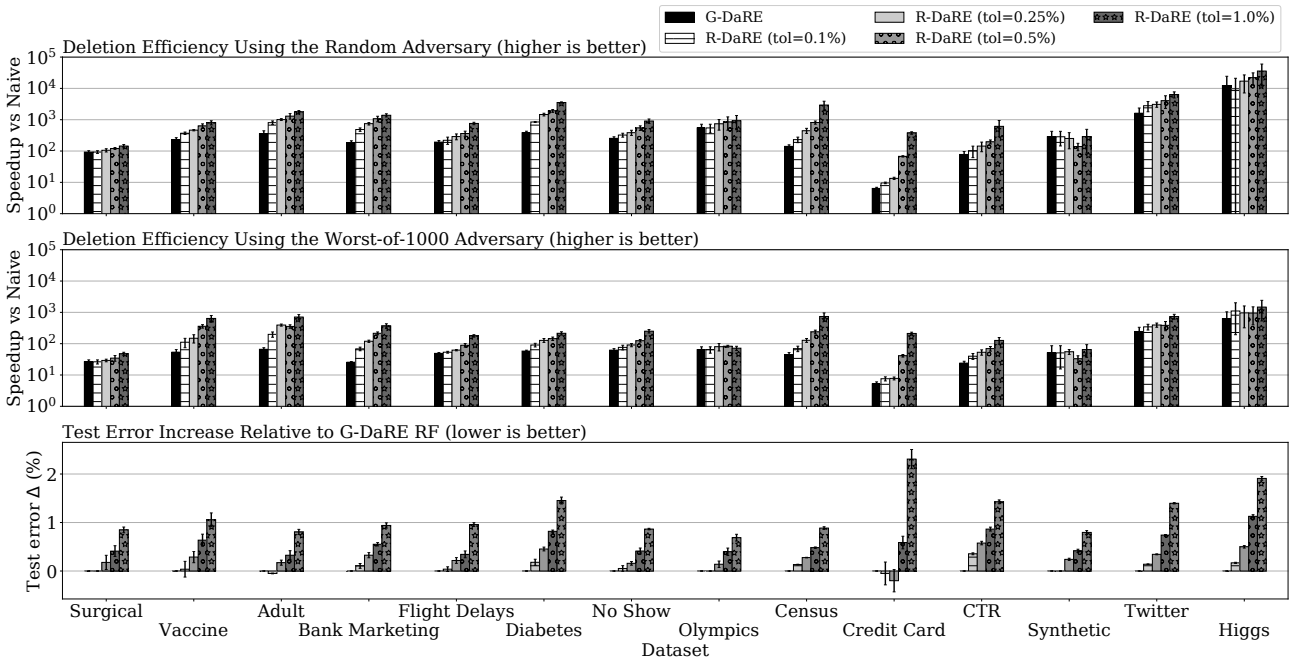

Figure 1. Deletion efficiency of DaRE RF. *Top & Middle*: Number of instances deleted in the time it takes the naive retraining approach to delete one instance using the random and worst-of-1000 adversaries, respectively (error bars represent standard deviation). *Bottom*: The increase in test error when using R-DaRE RF relative to the predictive performance of G-DaRE RF (error bars represent standard error).

As an example of DaRE RF's utility, naive retraining took 1.3 hours to delete a single instance for the Higgs dataset. R-DaRE RF ($tol = 0.25\%$ resulting in $d_{\text{rmax}}= 3$) deleted over 17,000 instances in that time, an average of 0.283s per deletion, while the average test set error increased by only 0.5%. In this case, R-DaRE RF provides a speedup of over four orders of magnitude, providing a tractable solution for something previously intractable.

Worst-of-1000 Adversary Against the more challenging worst-of-1000 adversary (Figure 1 (middle)), the speedup over naive deletion remains large, but is often an order of magnitude smaller. While R-DaRE models also decrease in efficiency, they maintain a significant advantage over G-DaRE RF, showing very similar trends of increased relative efficiency as when using the random adversary.

Summary A summary of the deletion efficiency results is in Table 2. When instances to delete are chosen randomly, G-DaRE RF is more than 250x faster than naively retraining after every deletion (taking the geometric mean over the 14 datasets). By adding randomness, R-DaRE models achieve even larger speedups, from 360x to over 1,200x, depending on the predictive performance tolerance (0.1% to 1.0%). The more sophisticated worst-of-1000 adversary can force more costly retraining. In this case, G-DaRE RF is more than 50x faster than naive retraining, and R-DaRE RF ranges from 80x to 260x depending on the tolerance.

Table 2. Summary of the deletion efficiency results. Specifically, the minimum, maximum, and geometric mean (G. mean) of the speedup vs. the naive retraining method across all datasets.

Model	Min.	Max.	G. Mean
Random Adversary			
G-DaRE	6x	12,232x	257x
R-DaRE (tol=0.1%)	10x	9,735x	366x
R-DaRE (tol=0.25%)	13x	17,044x	494x
R-DaRE (tol=0.5%)	68x	22,011x	681x
R-DaRE (tol=1.0%)	145x	35,856x	1,272x
Worst-of-1000 Adversary			
G-DaRE	5x	626x	52x
R-DaRE (tol=0.1%)	8x	1,106x	79x
R-DaRE (tol=0.25%)	8x	961x	102x
R-DaRE (tol=0.5%)	33x	950x	139x
R-DaRE (tol=1.0%)	47x	1,476x	263x

4.3. Effect of d_{rmax} and k on Deletion Efficiency

Figure 2 details the effect d_{rmax} has on deletion efficiency under each adversary for the Bank Marketing dataset[3]. As expected, we see that deletion efficiency increases as d_{rmax} increases. Predictive performance degrades as d_{rmax} increases, but initially degrades gracefully, maintain-

[3]Other datasets show similar trends; see the Appendix: §B.3.

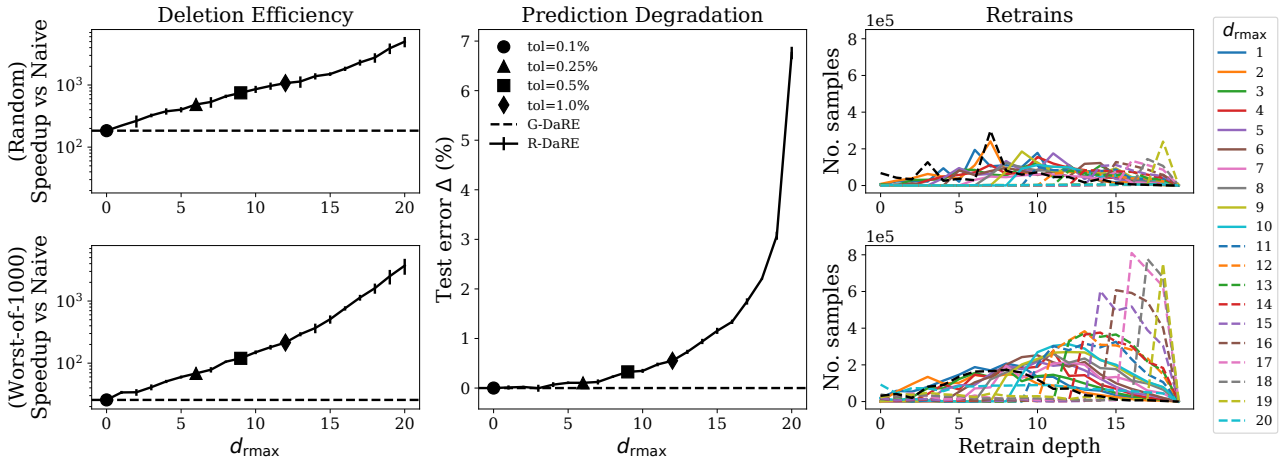

Figure 2. Effect of increasing d_{rmax}on deletion efficiency (left), predictive performance (middle), and the cost of retraining (right) using the random (top) and worst-of-1000 (bottom) adversaries for the Bank Marketing dataset. The predictive performance is independent of the adversary, as performance is measured before any deletions occur. Error bars represent standard deviation and standard error for the left and middle plots, respectively. In short, we see that increasing d_{rmax} increases deletion efficiency but initially gradually degrades predictive performance. Similar analysis for other datasets are in the Appendix: §B.3.

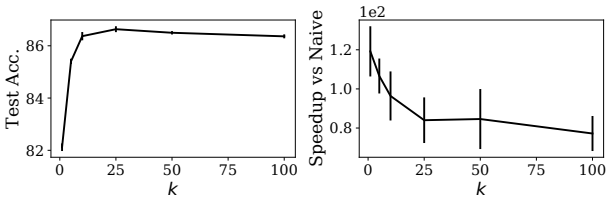

Figure 3. Effect of increasing k on predictive performance (left) and deletion efficiency (right) for the Surgical dataset using the random adversary; d_{rmax} is held fixed at 0. Error bars represent standard error and standard deviation for the left and right plots, respectively. Analysis for other datasets is in the Appendix: §B.4.

ing a low increase in test error even as the top ten layers of each tree are replaced with random nodes (+0.346% test error).

Figure 2 also shows the number of instances retrained at each depth, across all trees in the model. We immediately notice the increase in retraining cost when switching from the random (top-right plot) to the worst-of-1000 (bottom-right plot) adversary, especially at larger depths. This matches our intuition since nodes deeper in the tree have fewer instances; each instance thus has a larger influence on the resulting split criterion over all attributes at a given node and increases the likelihood that a chosen attribute may change, resulting in more subtree retraining.

Figure 3 shows the effect increasing k has on predictive performance and deletion efficiency for the Surgical dataset[4]. In general, we find k introduces a trade-off between predictive performance and deletion efficiency. However, our experiments show that modest values of k can achieve competitive predictive performance while maintaining a high degree of deletion efficiency and incurring low storage costs.

4.4. Space Overhead

This section shows the space overhead of DARE forests by breaking the memory usage of G-DARE RF into three constituent parts: 1) the structure of the model that is needed for making predictions, 2) the additional statistics stored at each decision node, and 3) the additional statistics and training-instance pointers stored at each leaf node. Parts 2) and 3), plus the size of the data, constitute the space needed by G-DARE RF to enable efficient data removal.

Table 3 shows the space overhead of G-DARE RF after training. We also show the training set size for each dataset, and the total memory usage of an SKLearn RF model using the same values for T and d_{max} as G-DARE RF.

As expected, decision-node statistics often make up the bulk of the space overhead for G-DARE RF; two exceptions are the Credit Card and CTR datasets, in which the size of the training-instance pointers outweigh the relatively low number of decision nodes (an average of 238 and 726 per tree, respectively) for those models. The total memory usage of the G-DARE RF *model* is 10-113x larger than that of the SKLearn RF model. However, since both approaches require the training data to enable deletions (G-DARE RF may need to retrain subtrees; SKLearn RF needs to retrain using the naive approach), the relative overhead of G-DARE RF is the ratio of (data + G-DARE RF) to (data + SKLearn RF); this results in an overhead of 6–26x, quantifying the time/space trade-off for efficient data deletion.

[4]Other datasets show similar trends; see the Appendix: §B.4.

Table 3. Memory usage (in megabytes) for the training data, G-DARE RF, and an SKLearn RF trained using the same values of T and d_{max} as G-DARE RF. The total memory usage for the G-DARE RF model is broken down into: 1) the structure of the model needed for making predictions (Structure); 2) the additional statistics stored at each decision node (Decision Stats.); and 3) the additional statistics and training-instance pointers stored at each leaf node (Leaf Stats.). The space overhead for G-DARE RF to enable efficient data deletion is measured as a ratio of the total memory usage of (data + G-DARE RF) to (data + SKLearn RF). Results are averaged over five runs and the standard error is shown in parentheses.

		G-DARE RF					
Dataset	**Data**	Structure	Decision Stats.	Leaf Stats.	**Total**	**SKLearn RF**	**Overhead**
Surgical	4	15 (0)	388 (1)	14 (0)	417 (1)	31 (0)	12.0x
Vaccine	16	18 (0)	426 (1)	14 (0)	458 (2)	37 (0)	8.9x
Adult	14	9 (0)	227 (1)	16 (0)	252 (1)	18 (0)	8.3x
Bank Marketing	8	23 (0)	455 (2)	33 (0)	511 (2)	51 (0)	8.8x
Flight Delays	207	37 (0)	3,030 (4)	171 (0)	3,238 (5)	66 (0)	12.6x
Diabetes	83	125 (0)	4,968 (12)	199 (0)	5,292 (12)	257 (1)	15.8x
No Show	35	91 (0)	2,511 (5)	203 (0)	2,805 (5)	187 (1)	12.8x
Olympics	663	27 (0)	3,196 (22)	338 (0)	3,561 (23)	57 (0)	5.9x
Census	326	33 (0)	1,737 (14)	169 (0)	1,939 (14)	63 (0)	5.8x
Credit Card	27	5 (0)	105 (1)	457 (0)	567 (0)	7 (0)	17.5x
CTR	45	6 (0)	485 (2)	642 (0)	1,133 (0)	10 (0)	21.4x
Twitter	48	186 (1)	2,450 (11)	693 (0)	3,329 (12)	332 (0)	8.9x
Synthetic	131	128 (1)	5,661 (36)	357 (0)	6,146 (37)	114 (1)	25.6x
Higgs	1,021	935 (4)	39,416 (168)	3,787 (1)	44,138 (173)	1,325 (9)	19.3x

5. Related Work

Exact Unlearning There are a number of works that support exact unlearning of SVMs (Cauwenberghs & Poggio, 2001; Tveit et al., 2003; Duan et al., 2007; Romero et al., 2007; Karasuyama & Takeuchi, 2009; Chen et al., 2019) in which the original goal was to accelerate leave-one-out cross-validation (Shao, 1993). More recently, Cao & Yang (2015) developed deletion mechanisms for several models that fall under the umbrella of non-adaptive SQ-learning (Kearns, 1998) in which data deletion is efficient and exact (e.g. naive Bayes, item-item recommendation, etc.); Schelter (2020) has also developed decremental update procedures for similar classes of models. Ginart et al. (2019) introduced a quantized variant of the k-means algorithm (Lloyd, 1982) called Q-k-means that supports exact data deletion. Bourtoule et al. (2021) and Aldaghri et al. (2020) propose training an ensemble of deep learning models on disjoint "shards" of a dataset, saving a snapshot of each model for every data point; the biggest drawbacks are the large storage costs, applicability only to *iterative* learning algorithms, and the significant degradation of predictive performance. Schelter et al. (2021) enable efficient data removal for extremely randomized trees (ERTs) (Geurts et al., 2006) without needing to save the training data by precomputing alternative subtrees for splits sensitive to deletions; aside from only being applicable to ERTs, they assume a very small percentage of instances can be deleted.

Approximate Unlearning In contrast to exact unlearning, a promising definition of approximate unlearning (a.k.a statistical unlearning) guarantees $\forall \mathcal{S} \subseteq \mathcal{H}, \mathcal{D}, (x, y) \in \mathcal{D}$: $e^{-\epsilon} \leq P(\mathcal{R}(\mathcal{A}(\mathcal{D}), \mathcal{D}, (x, y)) \in S) \ / \ P(\mathcal{A}(\mathcal{D} \setminus (x, y)) \in S) \leq e^{\epsilon}$ (ϵ-certified removal: Guo et al. (2020), Eq. 1). Golatkar et al. (2020c;b) propose a scrubbing mechanism for deep neural networks that does not require any retraining; however, the computational complexity of their approach is currently quite high. Guo et al. (2020), Izzo et al. (2020), and Wu et al. (2020) propose different removal mechanisms for linear and logistic regression models that can be applied to the last fully connected layer of a deep neural network. Golatkar et al. (2020a) perform unlearning on a linear approximation of large-scale vision networks in a mixed-privacy setting. Fu et al. (2021) propose an unlearning procedure for models in a Bayesian setting using variational inference.

Mitigation Although not specifically designed as unlearning techniques, the following works propose different mechanisms for mitigating the impact of noisy, poisoned, or non-private training data. Baumhauer et al. (2020) propose an output filtering technique that prevents private data from being leaked; however, their approach does not update the model itself, potentially leaking information if the model were still accessible. Wang et al. (2019) and Du et al. (2019) fine-tune their models on corrected versions of poisoned or corrupted training instances to mitigate backdoor

attacks (Gu et al., 2017) on image classifiers and anomaly detectors, respectively. Although both approaches show promising empirical performance, they provide no guarantees about the extent to which these problematic training instances are removed from the model (Sommer et al., 2020). Tople et al. (2019) analyze privacy leakage in language model snapshots before and after they are updated.

Differential Privacy Differential privacy (DP) (Dwork, 2006; Chaudhuri et al., 2011; Abadi et al., 2016) is a sufficient condition for approximate unlearning (in the case of a single deletion, sequential deletions may require using group DP (Dwork et al., 2014)), but it is an unnecessary and overly strict one since machine unlearning does not require instances to be private (Guo et al., 2020). Furthermore, differentially-private random forest models often suffer from poor predictive performance (Fletcher & Islam, 2015; 2019); this is because the privacy budget (typically denoted ϵ or β) must be split among all the trees in the forest, and among the different layers in each tree. This typically results in a meaningless privacy budget (e.g. $\epsilon > 10$) (Fletcher & Islam, 2019), a relaxed definition of DP (Rana et al., 2015), extremely randomized trees (Geurts et al., 2006; Fletcher & Islam, 2017), or very small forests (e.g. $T = 10$) (Consul & Williamson, 2020).

6. Discussion

Since data deletions in DaRE models are exact, membership inference attacks (Yeom et al., 2018; Carlini et al., 2018) are guaranteed to be unsuccessful for instances deleted from the model. DaRE models also reduce the need for deletion verification methods (Shintre & Dhaliwal, 2019; Sommer et al., 2020). However, one must be aware that DaRE models (as well as any unlearning method) can leak which instances are deleted if an adversary has access to the model before *and* after the deletion (Chen et al., 2020). Although privacy is a strong motivator for this work, there are a number of other useful applications for DaRE forests.

Instance-Based Interpretability A popular form of interpretability looks at how much each training instance contributes to a given prediction (Koh & Liang, 2017; Yeh et al., 2018; Sharchilev et al., 2018; Pruthi et al., 2020; Chen et al., 2021). The naive approach to this task involves leave-one-out retraining for every training instance in order to analyze the effect each instance has on a target prediction, but this is typically intractable for most machine learning models and datasets. DaRE models can more efficiently compute the same training-instance attributions as the naive approach, making leave-one-out retraining a potentially viable option for generating instance-attribution explanations for random forest models.

Dataset Cleaning Aside from removing user data for privacy reasons, one may also wish to efficiently remove outliers (Rahmani & Li, 2019; Dong et al., 2019) or training instances that are noisy, corrupted, or poisoned (Mozaffari-Kermani et al., 2014; Steinhardt et al., 2017).

Continual Learning Our methods can also be used to *add* data to a random forest model, allowing for continuous updating as data is added and removed. This makes them well-suited to continual learning settings with streaming data (Chrysakis & Moens, 2020; Knoblauch et al., 2020). However, the hyperparameters may need to be periodicially retuned as the size or distribution of the data shifts from adding and/or deleting more and more instances.

Eco-Friendly Machine Learning Finally, this line of research promotes a more economically and environmentally sustainable approach to building learning systems; if a model can be continuously updated only as necessary and avoid frequent retraining, significant time and computational resources can be spared (Gupta et al., 2020).

7. Conclusion

In this work, we introduced DaRE RF, a random forest variant that supports efficient model updates in response to repeated deletions of training instances. We find that, on average, DaRE models are 2-3 orders of magnitude faster than the naive retraining approach with no loss in accuracy, and additional efficiency can be achieved if slightly worse predictive performance is tolerated.

For future work, there are many exciting opportunities and applications of DaRE forests, from maintaining user privacy to building interpretable models to cleaning data, all without retraining from scratch. One could even investigate the possibility of extending DaRE forests to boosted trees (Chen & Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018). At its best, DaRE RF was more than four orders of magnitude faster than naive retraining, so it has the potential to enable new applications of model updating that were previously intractable.

Acknowledgments

We would like to thank Zayd Hammoudeh for useful discussions and feedback and the reviewers for their constructive comments that improved this paper. This work was supported by a grant from the Air Force Research Laboratory and the Defense Advanced Research Projects Agency (DARPA) — agreement number FA8750-16-C-0166, subcontract K001892-00-S05. This work benefited from access to the University of Oregon high performance computer, Talapas.

References

Abadi, M., Chu, A., Goodfellow, I., et al. Deep learning with differential privacy. In *CCS*, 2016.

Aldaghri, N., Mahdavifar, H., and Beirami, A. Coded machine unlearning. *arXiv preprint arXiv:2012.15721*, 2020.

Baldi, P., Sadowski, P., and Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 2014.

Baumhauer, T., Schöttle, P., and Zeppelzauer, M. Machine unlearning: Linear filtration for logit-based classifiers. *arXiv preprint arXiv:2002.02730*, 2020.

Biau, G., Scornet, E., and Welbl, J. Neural random forests. *Sankhya A*, 2019.

Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C., et al. Machine unlearning. In *IEEE Symposium on Security and Privacy*, 2021.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. CRC Press, 1984.

Bull, P., Slavitt, I., and Lipstein, G. Harnessing the power of the crowd to increase capacity for data science in the social sector. In *ICML #Data4Good Workshop*, 2016.

California. California consumer privacy act. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375, 2018. [Online; accessed 16-April-2020].

Canada. Pipeda. https://www.priv.gc.ca/en/opc-news/news-and-announcements/2018/an_181010/, 2018. [Online; accessed 19-August-2020].

Cao, Y. and Yang, J. Towards making systems forget with machine unlearning. In *IEEE Symposium on Security and Privacy*, 2015.

Carlini, N., Liu, C., et al. The secret sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv preprint arXiv:1802.08232*, 2018.

Cauwenberghs, G. and Poggio, T. Incremental and decremental support vector machine learning. In *NeurIPS*, 2001.

Chaudhuri, K., Monteleoni, C., and Sarwate, A. D. Differentially private empirical risk minimization. *JMLR*, 2011.

Chen, M., Zhang, Z., et al. When machine unlearning jeopardizes privacy. *arXiv preprint arXiv:2005.02205*, 2020.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *KDD*, 2016.

Chen, Y., Xiong, J., Xu, W., and Zuo, J. A novel online incremental and decremental learning algorithm based on variable support vector machine. *Cluster Computing*, 2019.

Chen, Y., Li, B., Yu, H., Wu, P., and Miao, C. Hydra: Hypergradient data relevance analysis for interpreting deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Chrysakis, A. and Moens, M.-F. Online continual learning from imbalanced data. In *ICML*, 2020.

Consul, S. and Williamson, S. Differentially private median forests for regression and classification. *arXiv preprint arXiv:2006.08795*, 2020.

Criteo. Criteo click-through rate prediction. https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/, 2015. [Online; accessed 25-Januaray-2021].

Denil, M., Matheson, D., and De Freitas, N. Narrowing the gap: Random forests in theory and in practice. In *ICML*, 2014.

Dong, Y., Hopkins, S., and Li, J. Quantum entropy scoring for fast robust mean estimation and improved outlier detection. In *NeurIPS*, 2019.

DrivenData. Flu shot learning: Predict h1n1 and seasonal flu vaccines. https://www.drivendata.org/competitions/66/flu-shot-learning/data/, 2019. [Online; accessed 12-August-2020].

Du, M., Chen, Z., Liu, C., Oak, R., and Song, D. Lifelong anomaly detection through unlearning. In *CCS*, 2019.

Dua, D. and Graff, C. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2019.

Duan, H., Li, H., He, G., and Zeng, Q. Decremental learning algorithms for nonlinear langrangian and least squares support vector machines. In *OSB*, 2007.

Dwork, C. Differential privacy. In *ICALP*, 2006.

Dwork, C., Roth, A., et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 2014.

EU. Regulation (eu) 2016/679. https://eur-lex.europa.eu/eli/reg/2016/679/oj, 2016. [Online; accessed 16-April-2020].

Fletcher, S. and Islam, M. Z. A differentially private decision forest. In *AusDM*, 2015.

Fletcher, S. and Islam, M. Z. Differentially private random decision forests using smooth sensitivity. *Expert Systems with Applications*, 2017.

Fletcher, S. and Islam, M. Z. Decision tree classification with differential privacy: A survey. *ACM Computing Surveys*, 2019.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001.

Fu, S., He, F., Xu, Y., and Tao, D. Bayesian inference forgetting. *arXiv preprint arXiv:2101.06417*, 2021.

Garg, S., Goldwasser, S., and Vasudevan, P. N. Formalizing data deletion in the context of the right to be forgotten. *arXiv preprint arXiv:2002.10635*, 2020.

Genuer, R., Poggi, J.-M., Tuleau-Malot, C., and Villa-Vialaneix, N. Random forests for big data. *Big Data Research*, 9:28–46, 2017.

Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine learning*, 2006.

Ginart, A., Guan, M., Valiant, G., and Zou, J. Y. Making AI forget you: Data deletion in machine learning. In *NeurIPS*, 2019.

Golatkar, A., Achille, A., Ravichandran, A., Polito, M., and Soatto, S. Mixed-privacy forgetting in deep networks. *arXiv preprint arXiv:2012.13431*, 2020a.

Golatkar, A., Achille, A., and Soatto, S. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. *arXiv preprint arXiv:2003.02960*, 2020b.

Golatkar, A., Achille, A., and Soatto, S. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *CVPR*, 2020c.

Gu, T., Dolan-Gavitt, B., and Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *Machine Learning and Computer Security Workshop*, 2017.

Guo, C., Goldstein, T., Hannun, A., and van der Maaten, L. Certified data removal from machine learning models. In *ICML*, 2020.

Gupta, A., Lanteigne, C., and Kingsley, S. SECure: A social and environmental certificate for AI systems. In *ICML Deploying and Monitoring Machine Learning Systems Workshop*, 2020.

Hanley, J. and McNeil, B. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 1982.

Haupt, J. and Nowak, R. Signal reconstruction from noisy random projections. *IEEE Transactions on Information Theory*, 2006.

Izzo, Z., Smart, M. A., Chaudhuri, K., and Zou, J. Approximate data deletion from machine learning models: Algorithms and evaluations. *arXiv preprint arXiv:2002.10077*, 2020.

Kaggle. Medical appointment no shows. https://www.kaggle.com/joniarroba/noshowappointments, 2016. [Online; accessed 25-Januaray-2021].

Kaggle. Credit card fraud detection. https://www.kaggle.com/mlg-ulb/creditcardfraud/, 2018a. [Online; accessed 27-July-2020].

Kaggle. 120 years of olympic history: Athletes and events. https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results, 2018b. [Online; accessed 28-July-2020].

Kaggle. Dataset surgical binary classification. https://www.kaggle.com/omnamahshivai/surgical-dataset-binary-classification/version/1#, 2018c. [Online; accessed 29-July-2020].

Karasuyama, M. and Takeuchi, I. Multiple incremental decremental learning of support vector machines. In *NeurIPS*, 2009.

Ke, G., Meng, Q., et al. LightGBM: A highly efficient gradient boosting decision tree. In *NeurIPS*, 2017.

Kearns, M. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM (JACM)*, 1998.

Knoblauch, J., Husain, H., and Diethe, T. Optimal continual learning has perfect memory and is NP-hard. In *ICML*, 2020.

Kocev, D., Vens, C., et al. Tree ensembles for predicting structured outputs. *Pattern Recognition*, 2013.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *ICML*, 2017.

Kwak, C., Lee, J., et al. Let machines unlearn–machine unlearning and the right to be forgotten. *SIGSEC*, 2017.

Linero, A. R. and Yang, Y. Bayesian regression tree ensembles that adapt to smoothness and sparsity. *Journal of the Royal Statistical Society*, 2018.

Lloyd, S. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 1982.

Lundberg, S. M., Erion, G. G., and Lee, S.-I. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

Mentch, L. and Hooker, G. Quantifying uncertainty in random forests via confidence intervals and hypothesis tests. *Journal of Machine Learning Research*, 2016.

Moro, S., Cortez, P., et al. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 2014.

Mozaffari-Kermani, M., Sur-Kolay, S., et al. Systematic poisoning attacks on and defenses for machine learning in healthcare. *Journal of Biomedical and Health Informatics*, 2014.

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. Scikit-learn: Machine learning in Python. *JMLR*, 2011.

Prokhorenkova, L., Gusev, G., et al. CatBoost: Unbiased boosting with categorical features. In *NeurIPS*, 2018.

Pruthi, G., Liu, F., Kale, S., and Sundararajan, M. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33, 2020.

Quinlan, J. R. *C4.5: Programs for Machine Learning*. Elsevier, 2014.

Rahmani, M. and Li, P. Outlier detection and robust PCA using a convex measure of innovation. In *NeurIPS*, 2019.

Rana, S., Gupta, S. K., and Venkatesh, S. Differentially private random forest with high utility. In *ICDM*, 2015.

Research and Administration, I. T. Airline on-time performance and causes of flight delays. https://catalog.data.gov/dataset/airline-on-time-performance-and-causes-of-flight-delays-on-time-data, 2019. [Online; accessed 16-April-2020].

Romero, E., Barrio, I., and Belanche, L. Incremental and decremental learning for linear support vector machines. In *International Conference on Artificial Neural Networks*, 2007.

Schelter, S. "amnesia" - machine learning models that can forget user data very fast. In *CIDR*, 2020.

Schelter, S., Grafberger, S., and Dunning, T. Hedgecut: Maintaining randomised trees for low-latency machine unlearning. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, 2021.

Sedhai, S. and Sun, A. Hspam14: A collection of 14 million tweets for hashtag-oriented spam research. In *SIGIR*, 2015.

Shao, J. Linear model selection by cross-validation. *Journal of the American Statistical Association*, 1993.

Sharchilev, B., Ustinovskiy, Y., Serdyukov, P., and de Rijke, M. Finding influential training samples for gradient boosted decision trees. In *ICML*, 2018.

Shintre, S. and Dhaliwal, J. Verifying that the influence of a user data point has been removed from a machine learning classifier. https://patents.google.com/patent/US10225277B1/en, 2019.

Shintre, S., Roundy, K. A., and Dhaliwal, J. Making machine learning forget. In *Annual Privacy Forum*, 2019.

Shoeybi, M., Patwary, M., et al. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Sommer, D. M., Song, L., Wagh, S., and Mittal, P. Towards probabilistic verification of machine unlearning. *arXiv preprint arXiv:2003.04247*, 2020.

Steinhardt, J., Koh, P. W. W., and Liang, P. S. Certified defenses for data poisoning attacks. In *NeurIPS*, 2017.

Strack, B., DeShazo, J. P., et al. Impact of HbA1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records. *BioMed research international*, 2014, 2014.

Tople, S., Brockschmidt, M., et al. Analyzing privacy loss in updates of natural language models. *arXiv preprint arXiv:1912.07942*, 2019.

Tveit, A., Hetland, M. L., and Engum, H. Incremental and decremental proximal support vector classification using decay coefficients. In *DaWaK*, 2003.

Villaronga, E. F., Kieseberg, P., and Li, T. Humans forget, machines remember: Artificial intelligence and the right to be forgotten. *Computer Law & Security Review*, 2018.

Wager, S. and Athey, S. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 2018.

Wang, B., Yao, Y., et al. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *IEEE Symposium on Security and Privacy*, 2019.

Wu, Y., Dobriban, E., and Davidson, S. B. DeltaGrad: Rapid retraining of machine learning models. In *ICML*, 2020.

Yeh, C.-K., Kim, J., et al. Representer point selection for explaining deep neural networks. In *NeurIPS*, 2018.

Yeom, S., Giacomelli, I., et al. Privacy risk in machine learning: Analyzing the connection to overfitting. In *CSF*, 2018.

Zaman, F. and Hirose, H. Effect of subsampling rate on subbagging and related ensembles of stable classifiers. In *PReMI*, 2009.

Zhu, M. Recall, precision and precision, average. *University of Waterloo, Waterloo*, 2004.