# A. Algorithmic Details

## A.1. Exact Deletion: Proof of Theorem 3.1

We use the following Lemma to help prove the theorem of exact deletion for DaRE forests.

**Lemma A.1.** *The probability of selecting a valid set of thresholds $S$ from a dataset $D$ and then subsequently resampling any invalidated thresholds after the deletion of $(x, y) \in D$ is equivalent to the probability of selecting $S$ from an updated dataset $D \setminus (x, y)$.*

*Proof.* The probability of choosing a valid set of thresholds $S$ from $D \setminus (x, y)$ is $P^A(S) = 1/\binom{n-m}{k}$ in which $n$ is the number of valid thresholds before the deletion, $k$ is the number of thresholds to sample from the set of valid thresholds, and $m$ is the number of thresholds that become invalid due to the deletion of $(x, y)$. The probability of ending up with thresholds $S$ by first choosing some set $S^*$ and then resampling any thresholds invalidated by removing $(x, y)$ is:

$$P^{B+R}(S) = \frac{1}{\binom{n}{k}} \sum_{i=0}^{m} \frac{\binom{m}{i}\binom{k}{i}}{\binom{n-k-(m-i)}{i}},$$

in which $\binom{n}{k}$ is the number of valid threshold sets for $D$; $S^*$ may have up to $m$ invalid thresholds, thus $\binom{m}{i}\binom{k}{i}$ is the number of ways $i$ invalid thresholds out of $k$ chosen thresholds could be resampled from the set of $m$ invalid thresholds; and $\binom{n-k-(m-i)}{i}$ is the number of valid threshold sets that can be resampled to, starting at a set with $i$ invalid thresholds.

In the simplest case, $m = 0$ and no thresholds are invalidated, so the probability of choosing $S$ in the updated dataset and original dataset are identical: $1/\binom{n}{k} = 1/\binom{n-0}{k}$. In the next simplest case, $m = 1$ and only a single threshold is invalidated. Thus, we could arrive at $S$ by first sampling it with the original dataset (probability $1/\binom{n}{k}$) or by first sampling one of the $k$ sets that includes the invalidated threshold and is otherwise identical to $S$, followed by resampling that threshold from the remaining $(n-1)-(k-1)$ valid and unselected thresholds to obtain $S$.

Thus, the total probability (for $m = 1$) is:

$$\begin{aligned}
P^{B+R}(S) &= \frac{1}{\binom{n}{k}} \left(1 + \frac{k}{n-k}\right) \\
&= \frac{1}{\binom{n}{k}} \left(\frac{n}{n-k}\right) \\
&= \frac{k!\,(n-k)!\,n}{n!\,(n-k)} \\
&= \frac{k!(n-1-k)!}{(n-1)!} \\
&= \frac{1}{\binom{n-1}{k}} \\
&= P^A(S)
\end{aligned}$$

For $m > 1$, we can reduce it to the $m = 1$ case by viewing it as a sequence of invalidating one threshold at a time. After invalidating one of the thresholds, the probability remains uniform, so by induction it continues to remain uniform after a second deletion, or a third, or any number. □

**Theorem.** *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.*

*Proof.* Exact unlearning is defined as having the same probability distribution over models by deletion as by retraining (Def. (1)). For discrete attributes, the node statistics used in model updating are precisely those used for learning the initial structure, so as the statistics are updated, the structure is updated to match what would be learned from scratch (in distribution).

For continuous attributes, we first discretize by uniformly sampling $k$ thresholds from the set of all valid thresholds for that attribute. As instances are removed, if one of the sampled thresholds becomes invalid, then those thresholds are resampled to

obtain a set of valid thresholds. Lemma A.1 shows the resulting probability of each set of valid thresholds remains uniform, identical to what it would be if the model were retrained from scratch.

The same logic and lemma also applies for attributes. If a deletion causes one or more attributes to become invalid (i.e. no more valid thresholds to sample), then those attributes are resampled to obtain a set of valid attributes, with all sets of valid attributes being equally likely.

Since each decision node in the tree operates on its own partition of the data $D$, then updating all relevant decision nodes and leaf nodes results in the entire tree being updated to match the updated dataset. The extension to the forest follows since all trees are independent; thus, the probability of a DaRE forest after removing instances is the same as retraining the model from scratch on updated data.

□

### A.2. Training Complexity: Proof of Theorem 3.2

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

*Proof.* When training a DaRE tree, we begin by choosing a split for the root by iterating through all $n$ training instances and scoring $\tilde{p}$ randomly selected attributes. Generalizing this to nodes at other depths, there are (at most) $2^d$ nodes at depth $d$, and each of the $n$ training instances is assigned to one of these nodes. Choosing all splits at depth $d$ thus requires a total time of $\mathcal{O}(\tilde{p}\,n)$ across all depth-$d$ nodes, since we again process every training instance when finding the best split for each node. Summing over all depths, the total time is $\mathcal{O}(\tilde{p}\,n\,d_{\max})$ to train a single DaRE tree or $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$ to train a forest of $T$ trees. □

### A.3. Training Complexity: Proof of Theorem 3.3

**Theorem.** *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 an invalid attribute threshold, 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 $\mathcal{O}(\tilde{p}\,|D|\,(d_{\max} - d)))$.*

*Proof.* Deleting an instance from a DaRE tree (Alg. 2) requires traversing the tree from the root to the a leaf, updating node statistics, retraining a subtree (if necessary), and removing the instance from the tree's set of instances. Since there are $\tilde{p}$ candidate attributes at each node, subtracting the influence of $(x, y)$ from the node statistics and checking for invalid attribute thresholds requires $\mathcal{O}(\tilde{p})$ time. Recomputing the score for each attribute-threshold pair requires $\mathcal{O}(\tilde{p}\,k)$, since we have the necessary statistics and computing the Gini index can be done in constant time for each pair. Across all depths up to $d_{\max}$, this is a total time of $\mathcal{O}(\tilde{p}\,k\,d_{\max})$.

Choosing new thresholds requires making a list of all attribute values at a node, along with the associated labels. This can be done by traversing the subtree rooted at the node, visiting each leaf and collecting the attribute values from the instances at that leaf. Let $|D|$ be the total number of these instances. Since the number of leaves is bounded by the number of instances, traversing the subtree can be done in $\mathcal{O}(|D|)$ time, plus $\mathcal{O}(|D| \log |D|)$ time to sort the values. The remaining work of making a list of valid thresholds, randomly choosing $k$ thresholds, and computing statistics for these $k$ thresholds can all be done in $\mathcal{O}(|D|)$, since each requires (at most) a single pass through all $|D|$ instances. Thus, the total time is $\mathcal{O}(|D| \log |D|)$.

If the best attribute-threshold pair at a node has changed, then the subtree must be retrained. Let $|D|$ be the number of instances at the node and $d$ be its depth. The time for retraining a subtree is identical to the time for retraining a DaRE tree, except that the number of instances is $|D|$ and the maximum depth (relative to this node) is $(d_{\max} - d)$. Thus, the total time is $\mathcal{O}(\tilde{p}\,(d_{\max} - d)\,|D|)$. □

### A.4. Space Complexity: Proof of Theorem 3.4

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

*Proof.* The space complexity of a DaRE tree with a single decision node is $\mathcal{O}(k\,\tilde{p} + n)$ since we need to store a constant $\mathcal{O}(1)$ amount of metadata for $k$ thresholds times $\tilde{p}$ attributes as well as $n$ pointers (one for each training instance) partitioned

across the leaves in the tree. For a single DaRE tree with multiple decision nodes, we need to multiply the first term in the previous result by $2^{d_{\max}}$ since there may be $2^{d_{\max}}$ decision nodes in a single DaRE tree; the second term remains the same as the training instances are still partitioned across all leaves in the tree. Thus, the space complexity of a single DaRE tree is $\mathcal{O}(k\,\tilde{p}\,2^{d_{\max}}\,+n)$. For a DaRE forest, we need to multiply this result by $T$; thus, the space complexity of a DaRE forest is $\mathcal{O}(k\,\tilde{p}\,2^{d_{\max}}\,T+n\,T)$. □

Assuming that we have at least one training instance assigned to each leaf, the number of leaves in each tree is at most $n$, and thus the total number of nodes per tree is at most $2n-1$. This gives us an alternate bound of $\mathcal{O}(k\tilde{p}nT)$, which is proportional to the size of the training data times the number of thresholds and trees in the forest (in the worst case).

### A.5. Complexity of Slightly-Less-Naive Retraining

The complexity of naive retraining is the same as training a DaRE forest from scratch, $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$, where $n=|\mathcal{D}|$.

A slightly smarter approach is to retrain only the portion of each tree that depends on the deleted node. For example, if the best split at the root of the tree after deleting an instance is the same as it was before, then the data will be partitioned between its two children the same way as before. One part of this partition never contained the deleted instance, and that fraction of the tree is unchanged. The other part of the partition has been changed by this deletion, so we must recurse, but only in that half of the tree.

This is potentially more efficient, but the efficiency gains are still bounded relative to the naive retraining approach. Choosing the split at the root still requires iterating through all training instances to compute statistics for each attribute (and each split of each continuous attribute), for a total time of $\mathcal{O}(T\,\tilde{p}\,n)$ across all $T$ trees.

Since the total time for retraining is at most $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$, the gain from this optimized approach is at *most* a factor of $d_{\max}$ (10-20 in our experiments). This is ignoring the cost of scoring splits at the lower levels in the tree, or retraining the lower levels of the tree (as is often required after data deletion). Thus, the gain will be smaller in practice.

Therefore, a slightly-less-naive approach to retraining random forests could improve over the naive approach, but would still be substantially slower than our methods, which achieve speedups of several orders of magnitude (see Figure 1 and Table 2).

### A.6. Node Statistics

A DaRE tree may consist of three types of nodes: greedy decision nodes, random decision nodes, and leaf nodes. Each stores a constant amount of metadata to enable efficient updates. In addition to the following type-specific statistics, each node stores $|D|$ and $|D_{\cdot,1}|$ the number of instances and the number of positive instances at that node.

- Greedy decision nodes: For each threshold for an attribute, we store $|D_l|$, $|D_{l,1}|$. This is a sufficient set of statistics needed to recompute the Gini index (Eq. 2) or entropy (Eq. 3) split criterion scores. Since a threshold in a greedy decision node is the midpoint between two adjacent attribute values, we also keep track of how many positive instances and the total number of instances are in each attribute value set; by updating this information, a DaRE tree can sample a new threshold when one is no longer valid.
- Random decision nodes: After selecting a random attribute, and then a random threshold within that attribute's min. and max. value range, we only store $|\mathcal{D}_l|$ and $|\mathcal{D}_r|$. Updating these statistics informs the DaRE tree when the threshold value is no longer within the min. and max. value range of that attribute. At that point, the random decision node is retrained.
- Leaf nodes: For each leaf, we store pointers to the training instances that traversed to that leaf. This enables the DaRE tree to collect these training instances when needing to retrain any ancestor decision nodes higher in the tree.

### A.7. Batch Deletion

Batch deletion is almost the same as deleting one instance, except we may need to recurse down multiple branches of each tree to find all relevant instances to delete, and we only retrain a given node (at most) once, rather than (up to) once for each instance deleted. This will naturally be more efficient, but waiting for a large batch may not be possible.

### A.8. Pseudocode

Algorithm 3 provides detailed pseudocode for training a DARE tree and deleting an instance from a trained DARE tree. For even more code details, please check out our code repository at https://github.com/jjbrophy47/dare_rf.

**Algorithm 3** Pseudocode for building a DaRE tree / subtree, and deleting an instance from a DARE tree (unabridged).

**TRAIN**(data $D$, depth $d$):
  **if** stopping criteria reached **then**
    $node \leftarrow$ LEAFNODE($D$)
  **else**
    **if** $d < d_{\text{rmax}}$ **then**
      $node \leftarrow$ RANDOMNODE($D$)
    **else**
      $node \leftarrow$ GREEDYNODE($D$)
    $D.\ell, D.r \leftarrow$ split on selected threshold($node$, $D$)
    $node.\ell, r \leftarrow$ TRAIN($\mathcal{D}_\ell$, $d+1$), TRAIN($\mathcal{D}_r$, $d+1$)
  **return** $node$

**DELETE**($node$, depth $d$, instance to remove $(x, y)$):
  $node \Leftarrow$ update pos. and total count($node$, $(x, y)$)
  **if** $node$ is a LEAFNODE **then**
    $node \Leftarrow$ remove $(x, y)$ from leaf instances
    $node \Leftarrow$ recompute leaf value($node$)
    remove $(x, y)$ from database
  **else**
    $node \Leftarrow$ update decision statistics($node$, $(x, y)$)
    **if** $node$ is a RANDOMNODE **then**
      $node \leftarrow$ RANDOMNODEDELETE($node$, $d$, $(x, y)$)
    **else**
      $node \leftarrow$ GREEDYNODEDELETE($node$, $d$, $(x, y)$)
    $a, v \leftarrow node$.selected attribute, threshold
    **if** no retraining occurred **then**
      **if** $x_{\cdot,a} \leq v$ **then**
        DELETE($node.\ell$, $d+1$, $(x, y)$)
      **else**
        DELETE($node.r$, $d+1$, $(x, y)$)
  **return** $node$

**GREEDYNODEDELETE**($node$, depth $d$, inst. $(x, y)$):
  $A \leftarrow node$.sampled attributes
  $\bar{A} \leftarrow$ get invalid attributes($A$)
  **if** $|\bar{A}| > 0$ **then**
    $D \leftarrow$ get data from leaves($node$) $\setminus (x, y)$ ▷ Cache
    $A^* \leftarrow$ resample invalid attributes($\bar{A}$, $D$)
    $A \leftarrow A \setminus \bar{A} \cup A^*$
  **for** $a \in A$ **do**
    $V \leftarrow a$.sampled valid thresholds
    $\bar{V} \leftarrow$ get invalid thresholds($V$)
    **if** $|\bar{V}| > 0$ **then**
      $D \leftarrow$ get data from leaves($node$) $\setminus (x, y)$ ▷ Cache
      $V^* \leftarrow$ resample invalid thresholds($\bar{V}$, $D$, $a$)
      $V \leftarrow V \setminus \bar{V} \cup V^*$
  $scores \leftarrow$ recompute split scores($node$)
  $node \Leftarrow$ select optimal split($scores$)
  **if** optimal split has changed **then**
    $a, v \leftarrow node$.selected attribute, threshold
    $D_\ell, D_r \leftarrow$ split data on new threshold($D$, $a$, $v$)
    $node.\ell \leftarrow$ TRAIN($\mathcal{D}_\ell$, $d+1$) ▷ Retrain left
    $node.r \leftarrow$ TRAIN($\mathcal{D}_r$, $d+1$) ▷ Retrain right
  **return** $node$

**LEAFNODE**(data $D$):
  $node \leftarrow$ NODE()
  $node \leftarrow$ SAVENODESTATS($node$, $D$)
  $node \Leftarrow$[5]compute leaf value($node$)
  $node \Leftarrow$ save leaf instances($node$, $D$)
  **return** $node$

**RANDOMNODE**(data $D$):
  $node \leftarrow$ NODE()
  $node \leftarrow$ SAVENODESTATS($node$, $D$)
  $a \leftarrow$ randomly sample attribute($D$)
  $v \leftarrow$ randomly sample threshold $\in [a_{\min}, a_{\max})$
  $node \leftarrow$ SAVETHRESHSTATS($node$, $D$, $a$, $v$)
  **return** node

**GREEDYNODE**(data $D$):
  $node \leftarrow$ NODE()
  $node \leftarrow$ SAVENODESTATS($node$, $D$)
  $node \Leftarrow$ randomly sample $\tilde{p}$ attributes($node$, $D$)
  **for** $a \in node$.sampled attributes **do**
    $C \leftarrow$ get valid thresholds($D$, $a$)
    $V \leftarrow$ randomly sample $k$ valid thresholds($C$)
    **for** $v \in V$ **do**
      $node \leftarrow$ SAVETHRESHSTATS($node$, $D$ $a$, $v$)
  $scores \leftarrow$ compute split scores($node$)
  $node \Leftarrow$ select optimal split($scores$)
  **return** $node$

**SAVENODESTATS**($node$, data $D$):
  $node \Leftarrow$ instance count($D$)
  $node \Leftarrow$ positive instance count($D$)
  **return** $node$

**SAVETHRESHSTATS**($node$, data $D$, attr. $a$, thresh. $v$):
  $node \Leftarrow$ left branch instance count($D$, $a$, $v$)
  **if** $node$ is a GREEDYNODE **then**
    $node \Leftarrow$ left branch pos. instance count($D$, $a$, $v$)
    $node \Leftarrow$ left / right adj. feature val.($D$, $a$, $v$)
    $node \Leftarrow$ left / right adj. val. set count($D$, $a$, $v$)
    $node \Leftarrow$ left / right adj. val. set pos. count($D$, $a$, $v$)
  **return** $node$

**RANDOMNODEDELETE**($node$, depth $d$, inst. $(x, y)$):
  **if** selected threshold is invalid **then**
    $D \leftarrow$ get data from leaves($node$) $\setminus (x, y)$
    **if** selected attribute ($a$) is still valid **then**
      $v \leftarrow$ resample threshold $\in [a_{min}, a_{max})$
      $D_\ell, D_r \leftarrow$ split data on new threshold($D$, $a$, $v$)
      $node.\ell \leftarrow$ TRAIN($\mathcal{D}_\ell$, $d+1$) ▷ Retrain left
      $node.r \leftarrow$ TRAIN($\mathcal{D}_r$, $d+1$) ▷ Retrain right
    **else**
      $node \leftarrow$ TRAIN($\mathcal{D}$, $d$) ▷ Retrain entire subtree
  **return** $node$

[5]Each node type has several types of counts and statistics to maintain. Rather than name and update each count separately, we use "$node \Leftarrow \ldots$" to denote updates to a node or its data. Details about node statistics for each node type are in §A.6.

## B. Implementation and Experiment Details

Experiments are run on an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.6GHz with 70GB of RAM. No parallelization is used when building the independent decision trees. DaRE RF is implemented in the C programming language via Cython, a Python package allowing the development of C extensions. Experiments are run using Python 3.7. Source code for DaRE RF and all experiments is available at `https://github.com/jjbrophy47/dare_rf`.

### B.1. Datasets

- Surgical (Kaggle, 2018c) consists of 14,635 medical patient surgeries (3,690 positive cases), characterized by 25 attributes; the goal is to predict whether or not a patient had a complication from their surgery.
- Vaccine (Bull et al., 2016; DrivenData, 2019) consists of 26,707 survey responses collected between October 2009 and June 2010 asking people a range of 36 behavioral and personal questions, and ultimately asking whether or not they got an H1N1 and/or seasonal flu vaccine. Our aim is to predict whether or not a person received a seasonal flu vaccine.
- Adult (Dua & Graff, 2019) contains 48,842 instances (11,687 positive) of 14 demographic attributes to determine if a person's personal income level is more than $50K per year.
- Bank Marketing (Moro et al., 2014; Dua & Graff, 2019) consists of 41,188 marketing phone calls (4,640 positive) from a Portuguese banking institution. There are 20 attributes, and the aim is to figure out if a client will subscribe.
- Flight Delays (Research & Administration, 2019) consists of 100,000 actual arrival and departure times of flights by certified U.S. air carriers; the data was collected by the Bureau of Transportation Statistics' (BTS) Office of Airline Information. The data contains 8 attributes and 19,044 delays. The task is to predict if a flight will be delayed.
- Diabetes (Strack et al., 2014; Dua & Graff, 2019) consists of 101,766 instances of patient and hospital readmission outcomes (46,902 readmitted) characterized by 55 attributes.
- No Show (Kaggle, 2016) contains 110,527 instances of patient attendances for doctors' appointments (22,319 no shows) characterized by 14 attributes. The aim is to predict whether or not a patient shows up to their doctors' appointment.
- Olympics (Kaggle, 2018b) contains 206,165 Olympic events over 120 years of Olympic history. Each event contains information about the athlete, their country, which Olympics the event took place, the sport, and what type of medal the athlete received. The aim is to predict whether or not an athlete received a medal for each event they participated in.
- Census (Dua & Graff, 2019) contains 40 demographic and employment attributes on 299,285 people in the United States; the survey was conducted by the U.S. Census Bureau. The goal is to predict if a person's income level is more than $50K.
- Credit Card (Kaggle, 2018a) contains 284,807 credit card transactions in September 2013 by European cardholders. The transactions took place over two days and contains 492 fraudulent charges (0.172% of all charges). There are 28 principal components resulting from PCA on the original dataset, and two additional fetures: 'time' and 'amount'. The aim is to predict whether a charge is fraudulent or not.
- Click-Through Rate (CTR) (Criteo, 2015) contains the first 1,000,000 instances of the Criteo 1TB Click Logs dataset, in which each row represents an ad that was displayed and whether or not it had been clicked on (29,040 ads clicked). The dataset contains 13 numeric attributes and 26 categorical attributes. However, due to the extremely large number of values for the categorical attributes, we restrict our use of the dataset to the 13 numeric attributes. The aim is to predict whether or not an ad is clicked on.
- Twitter uses the first 1,000,000 tweets (169,471 spam) of the HSpam14 dataset (Sedhai & Sun, 2015). Each instance contains the tweet ID and label. After retrieving the text and user ID for each tweet, we derive the following attributes: no. chars, no. hashtags, no. mentions, no. links, no. retweets, no. unicode chars., and no. messages per user. The aim is to predict whether a tweet is spam or not.
- Synthetic (Pedregosa et al., 2011) contains 1,000,000 instances normally distributed about the vertices of a 5-dimensional hypercube into 2 clusters per class. There are 5 informative attributes, 5 redundant attributes, and 30 useless attributes. There is interdependence between these attributes, and a randomly selected 5% of the labels are flipped to increase the difficulty of the classification task.

- Higgs (Baldi et al., 2014; Dua & Graff, 2019) contains 11,000,000 signal processes (5,829,123 Higgs bosons) characterized by 22 kinematic properties measured by detectors in a particle accelerator and 7 attributes derived from those properties. The goal is to distinguish between a background signal process and a Higgs bosons process.

*Table 4.* Dataset summary including the main predictive performance metric used for each dataset, either average precision (AP) for datasets whose positive label percentage $< 1\%$, AUC for datasets between [1%, 20%], or accuracy (Acc.) for all remaining datasets.

| **Dataset** | **No. train** | **Pos. label %** | **No. test** | **Pos. label %** | **No. attributes** | **Metric** |
|---|---|---|---|---|---|---|
| Surgical | 11,708 | 25.30 | 2,927 | 25.00 | 90 | Acc. |
| Vaccine | 21,365 | 46.60 | 5,342 | 45.60 | 185 | Acc. |
| Adult | 32,561 | 24.00 | 16,281 | 23.60 | 107 | Acc. |
| Bank Marketing | 32,951 | 11.40 | 8,237 | 10.90 | 63 | AUC |
| Flight Delays | 80,000 | 18.90 | 20,000 | 19.50 | 648 | AUC |
| Diabetes | 81,412 | 46.00 | 20,353 | 46.50 | 253 | Acc. |
| No Show | 88,422 | 20.14 | 22,105 | 20.41 | 99 | AUC |
| Olympics | 164,932 | 14.60 | 41,233 | 14.60 | 1,004 | AUC |
| Census | 199,523 | 6.20 | 99,762 | 6.20 | 408 | AUC |
| Credit Card | 227,846 | 0.18 | 56,961 | 0.17 | 29 | AP |
| CTR | 800,000 | 2.89 | 200,000 | 2.98 | 13 | AUC |
| Twitter | 800,000 | 16.96 | 200,000 | 16.83 | 15 | AUC |
| Synthetic | 800,000 | 50.00 | 200,000 | 50.00 | 40 | Acc. |
| Higgs | 8,800,000 | 53.00 | 2,200,000 | 53.00 | 28 | Acc. |

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. Table 4 summarizes the datasets after preprocessing.

### B.2. Predictive Performance of DaRE Forests

If extremely randomized trees exhibit the same predictive performance as their greedy counterparts, then adding and removing data can be done by simply updating class counts at the leaves and only retraining if a chosen threshold is no longer within the range of a chosen split attribute for a given decision node. Thus, this section compares the predictive performance of a G-DaRE forest against:

- Random Trees: Extremely randomized trees (Geurts et al., 2006) in which each decision node selects an attribute to split on uniformly at random, and then selects the threshold by sampling a value in that attribute's [min, max] range uniformly at random.
- Extra Trees: Similar to the extremely randomized trees model (Geurts et al., 2006), except each decision node selects $\lfloor \sqrt{p} \rfloor$ attributes at random; a threshold is then selected for each attribute by sampling a value in that attribute's [min, max] range uniformly at random. Then, a split criterion such as Gini index or mutual information is computed for each attribute-threshold pair, and the best threshold is chosen as the split for that node.
- SKLearn RF: Standard RF implementation from Scikit-Learn (Pedregosa et al., 2011).
- SKLearn RF (w/ bootstrap): Standard RF implementation from Scikit-Learn (Pedregosa et al., 2011) with bootstrapping.

Table 5 reports the predictive performance of each model on the test set after tuning using 5-fold cross-validation. We tune the number of trees in the forest using values [10, 25, 50, 100, 250], and the maximum depth using values [1, 3, 5, 10, 20]. The maximum number of randomly selected attributes to consider at each split is set to $\lfloor \sqrt{p} \rfloor$. For the G-DaRE model, we also tune the number of thresholds to consider for each attribute, $k$, using values [5, 10, 25, 50]. We use 50%, 25%, 2.5%, and 2.5% of the training data to tune the Twitter, Synthetic, Click-Through Rate, and Higgs datasets, respectively, and 100% for all other datasets. Selected values for all hyperparameters are in Table 6.

We find the predictive performance of the Random Trees and Extra Trees models to be consistently worse than the SKLearn and G-DaRE models. We also find that bootstrapping has a negligible effect on the SKLearn models. Finally, we observe that the predictive performance of the G-DaRE model is nearly identical to that of SKLearn RF, in which their scores are within 0.2% on 9/14 datasets, 0.4% on 1/14 datasets, and G-DaRE RF is significantly better than SKLearn RF on the Surgical, Flight Delays, Olympics, and Credit Card datasets.

*Table 5.* Predictive performance comparison of G-DaRE RF to: an extremely randomized trees model (Random Trees) (Geurts et al., 2006), an Extra Trees (Geurts et al., 2006) model, and a popular and widely used random forest implementation from Scikit-Learn (SKLearn) with and without bootstrapping. The numbers in each cell represent either average precision, AUC, or accuracy as specified by Table 4; results are averaged over five runs and the standard error is shown.

| Dataset | Random Trees | Extra Trees | SKLearn RF | SKLearn RF (w/ bootstrap) | G-DaRE RF |
|---|---|---|---|---|---|
| Surgical | 0.783 ± 0.001 | 0.805 ± 0.001 | 0.848 ± 0.001 | 0.846 ± 0.001 | **0.867 ± 0.001** |
| Vaccine | 0.769 ± 0.001 | 0.795 ± 0.001 | **0.796 ± 0.001** | 0.793 ± 0.002 | 0.794 ± 0.001 |
| Adult | 0.802 ± 0.003 | 0.847 ± 0.001 | **0.863 ± 0.000** | **0.863 ± 0.000** | 0.862 ± 0.001 |
| Bank Marketing | 0.879 ± 0.001 | 0.924 ± 0.000 | **0.940 ± 0.001** | **0.940 ± 0.001** | **0.940 ± 0.001** |
| Flight Delays | 0.650 ± 0.009 | 0.725 ± 0.001 | 0.729 ± 0.001 | 0.729 ± 0.000 | **0.739 ± 0.000** |
| Diabetes | 0.551 ± 0.003 | 0.631 ± 0.001 | 0.643 ± 0.000 | 0.642 ± 0.001 | **0.645 ± 0.000** |
| No Show | 0.694 ± 0.001 | 0.710 ± 0.000 | 0.732 ± 0.000 | 0.731 ± 0.000 | **0.736 ± 0.000** |
| Olympics | 0.835 ± 0.001 | 0.820 ± 0.001 | 0.819 ± 0.001 | 0.820 ± 0.000 | **0.871 ± 0.000** |
| Credit Card | 0.799 ± 0.002 | 0.840 ± 0.004 | 0.837 ± 0.002 | 0.831 ± 0.005 | **0.846 ± 0.001** |
| Census | 0.915 ± 0.001 | 0.936 ± 0.000 | 0.945 ± 0.000 | 0.945 ± 0.000 | **0.946 ± 0.000** |
| CTR | 0.668 ± 0.001 | 0.683 ± 0.000 | **0.702 ± 0.000** | 0.700 ± 0.000 | 0.701 ± 0.000 |
| Twitter | 0.883 ± 0.001 | 0.923 ± 0.001 | **0.943 ± 0.000** | 0.942 ± 0.000 | **0.943 ± 0.000** |
| Synthetic | 0.793 ± 0.002 | 0.909 ± 0.001 | **0.946 ± 0.001** | 0.945 ± 0.000 | 0.945 ± 0.000 |
| Higgs | 0.608 ± 0.001 | 0.700 ± 0.000 | **0.746 ± 0.000** | 0.744 ± 0.000 | 0.744 ± 0.000 |

*Table 6.* Hyperparameters selected for the G-DaRE and R-DaRE (using error tolerances of 0.1%, 0.25%, 0.5%, and 1.0%) models. The number of trees ($T$), maximum depth ($d_{\max}$), and the number of thresholds considered per attribute ($k$) are found using 5-fold cross-validation using a greedily-built model (i.e. G-DaRE RF). To build the R-DaRE model, the values for $T$, $d_{\max}$, and $k$ found in the previous step are held fixed, and the value for $d_{\text{rmax}}$is found by incrementing its value by one starting from zero until its 5-fold cross-validation score exceeds the specified error tolerance as compared to the cross-validation score of the G-DaRE model.

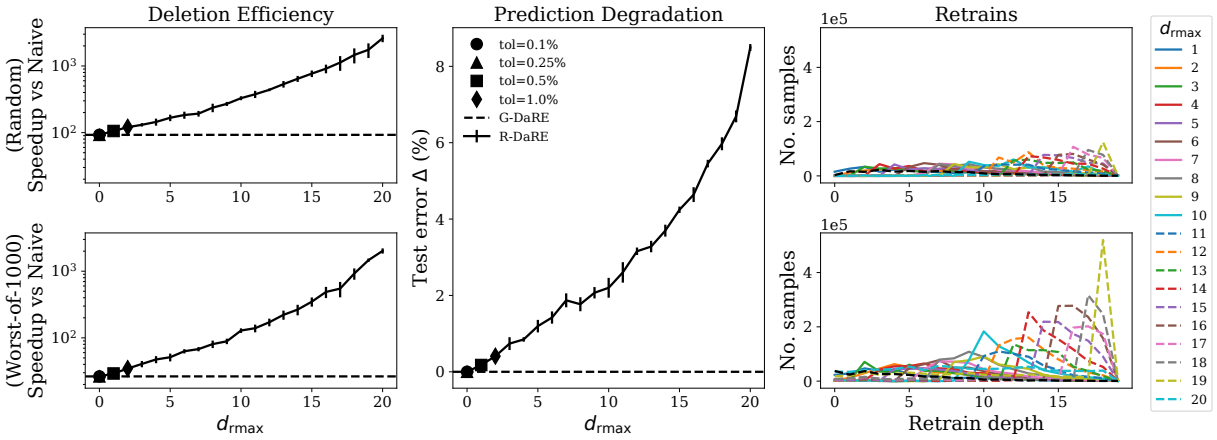
| | **G-DaRE & R-DaRE** | | | **R-DaRE Only** | | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | $T$ | $d_{\max}$ | $k$ | $d_{\text{rmax}}$ **(0.1%)** | $d_{\text{rmax}}$ **(0.25%)** | $d_{\text{rmax}}$ **(0.5%)** | $d_{\text{rmax}}$ **(1.0%)** |
| Surgical | 100 | 20 | 25 | 0 | 1 | 2 | 4 |
| Vaccine | 50 | 20 | 5 | 5 | 7 | 11 | 14 |
| Adult | 50 | 20 | 5 | 10 | 13 | 14 | 16 |
| Bank Marketing | 100 | 20 | 25 | 6 | 9 | 12 | 14 |
| Flight Delays | 250 | 20 | 25 | 1 | 3 | 5 | 10 |
| Diabetes | 250 | 20 | 5 | 7 | 10 | 12 | 15 |
| No Show | 250 | 20 | 10 | 1 | 3 | 6 | 10 |
| Olympics | 250 | 20 | 5 | 0 | 1 | 2 | 3 |
| Census | 100 | 20 | 25 | 6 | 9 | 12 | 16 |
| Credit Card | 250 | 20 | 5 | 5 | 8 | 14 | 17 |
| CTR | 100 | 10 | 50 | 2 | 3 | 4 | 6 |
| Twitter | 100 | 20 | 5 | 2 | 4 | 7 | 11 |
| Synthetic | 50 | 20 | 10 | 0 | 2 | 3 | 5 |
| Higgs | 50 | 20 | 10 | 1 | 3 | 6 | 9 |

*Table 7.* Training times (in seconds) for the G-DaRE model using the hyperparameters selected in Table 6. Mean and standard deviations (S.D.) are computed over five runs.

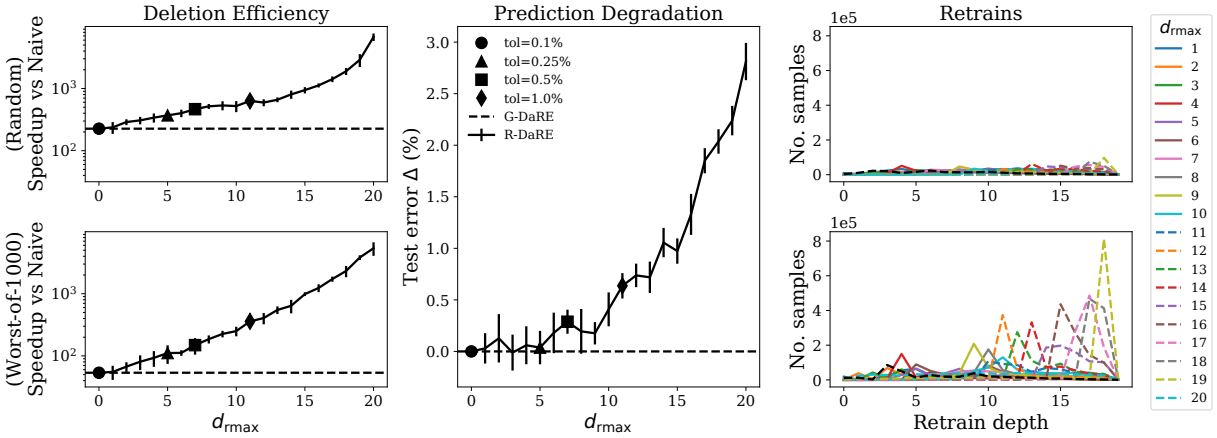
| **Dataset** | **Mean** | **S.D.** |
|---|---|---|
| Surgical | 5.68 | 2.97 |
| Vaccine | 17.08 | 11.86 |
| Adult | 6.76 | 1.17 |
| Bank Marketing | 8.79 | 3.37 |
| Flight Delays | 262.00 | 50.39 |
| Diabetes | 141.91 | 39.12 |
| No Show | 77.65 | 20.33 |
| Olympics | 596.27 | 157.70 |
| Census | 127.40 | 9.57 |
| Credit Card | 616.65 | 166.00 |
| Twitter | 152.34 | 12.32 |
| Synthetic | 732.05 | 231.70 |
| CTR | 121.64 | 37.13 |
| Higgs | 5,016.44 | 146.34 |

### B.3. Effect of $d_{\text{rmax}}$ on Deletion Efficiency

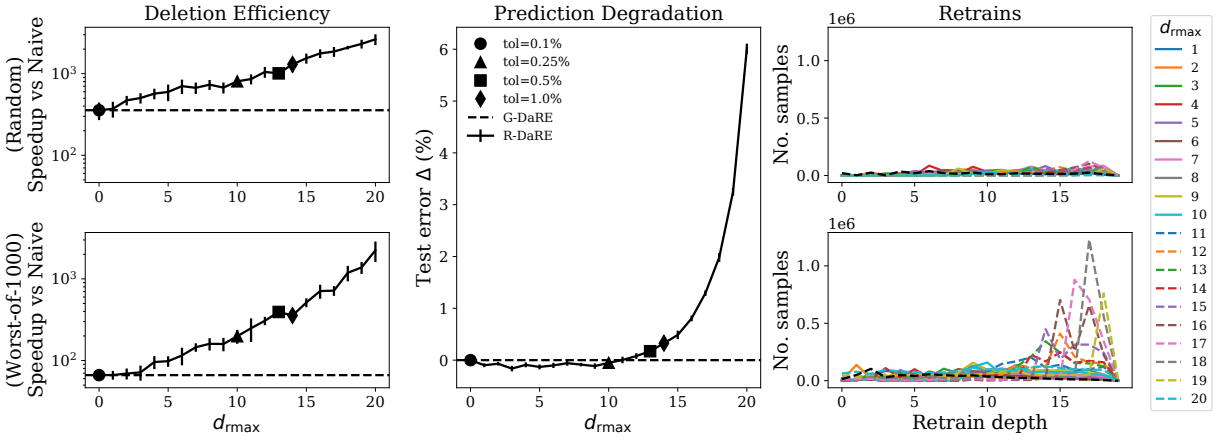
Figure 4 presents additional results on the effect $d_{\text{rmax}}$ has on deletion efficiency for different datasets.




(a) Surgical

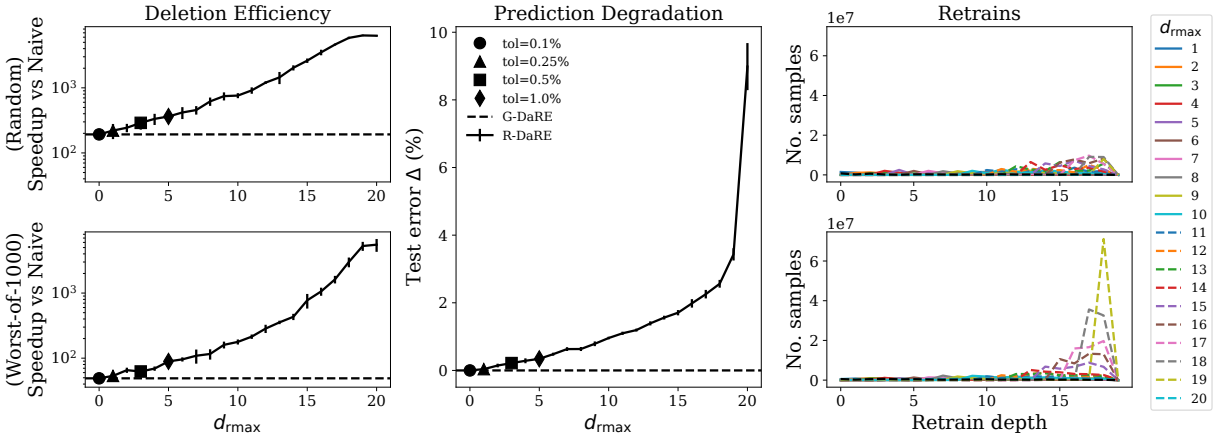



(b) Vaccine

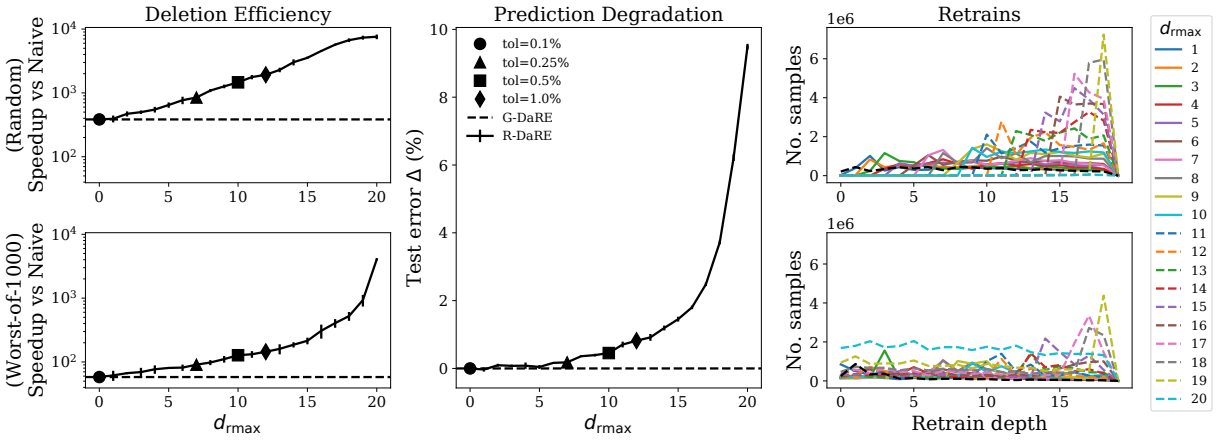



(c) Adult

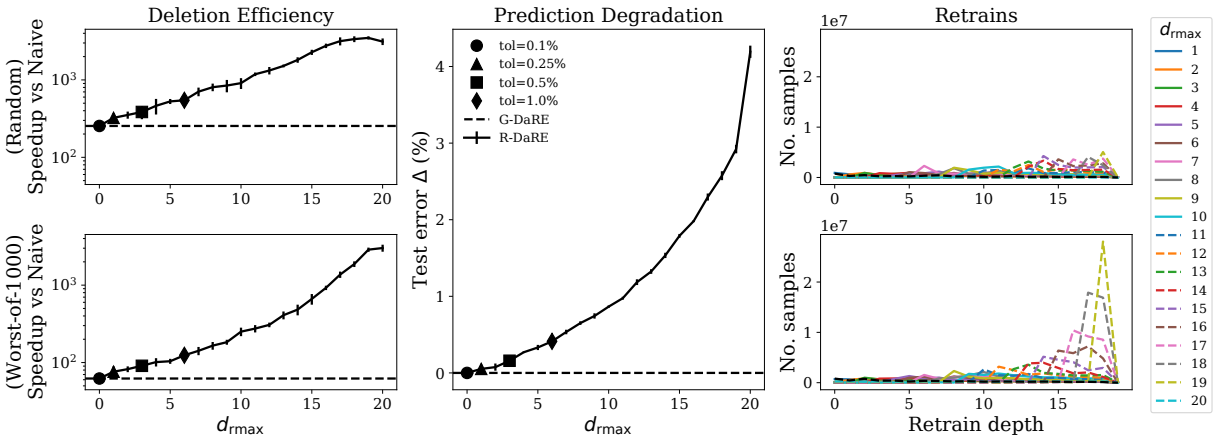
*Figure 4.* Effect of $d_{\text{rmax}}$ on deletion efficiency.




(d) Flight Delays

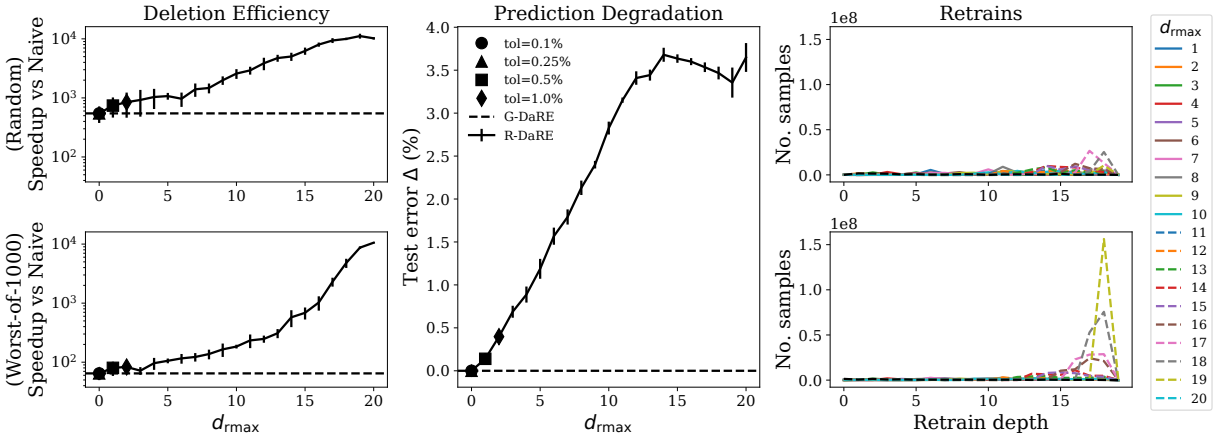



(e) Diabetes

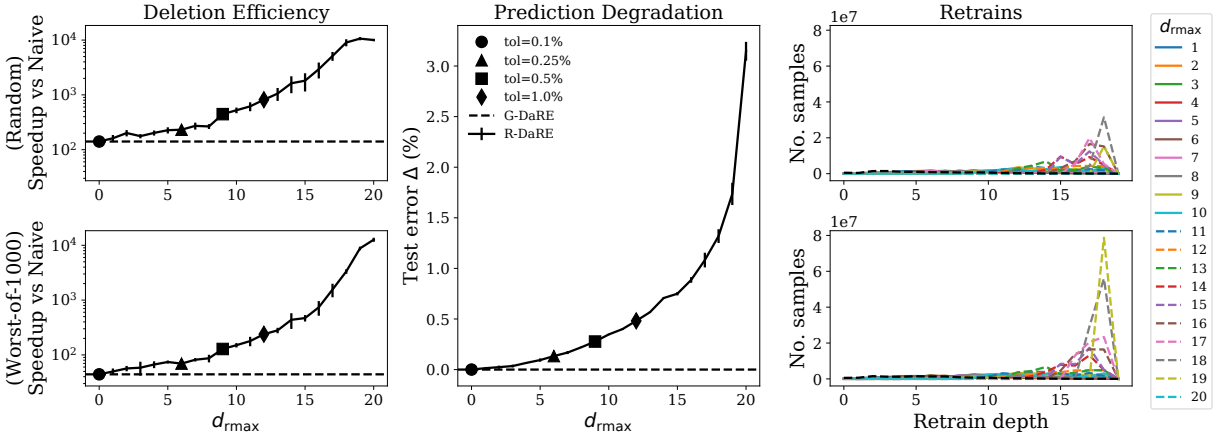



(f) No Show

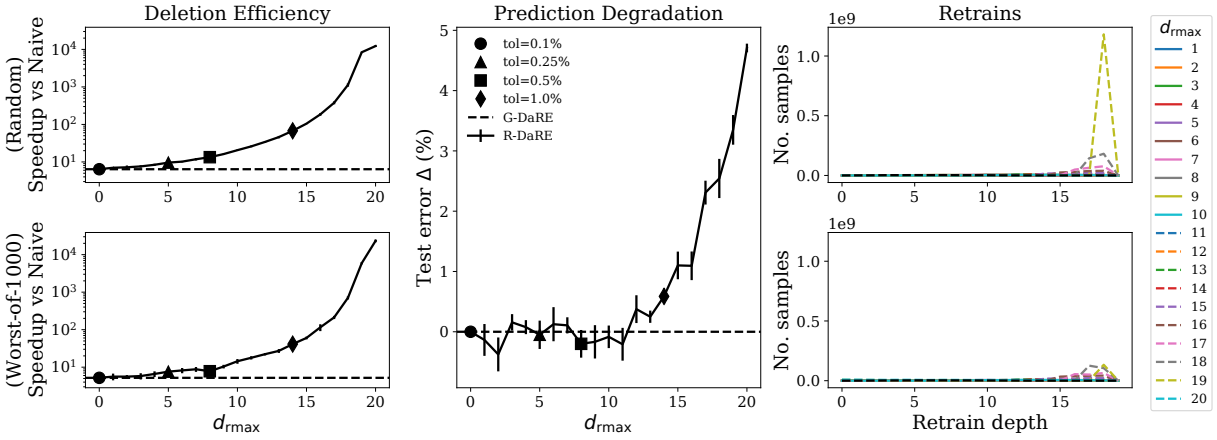
*Figure 4.* Effect of $d_{\text{rmax}}$ on deletion efficiency.




(g) Olympics

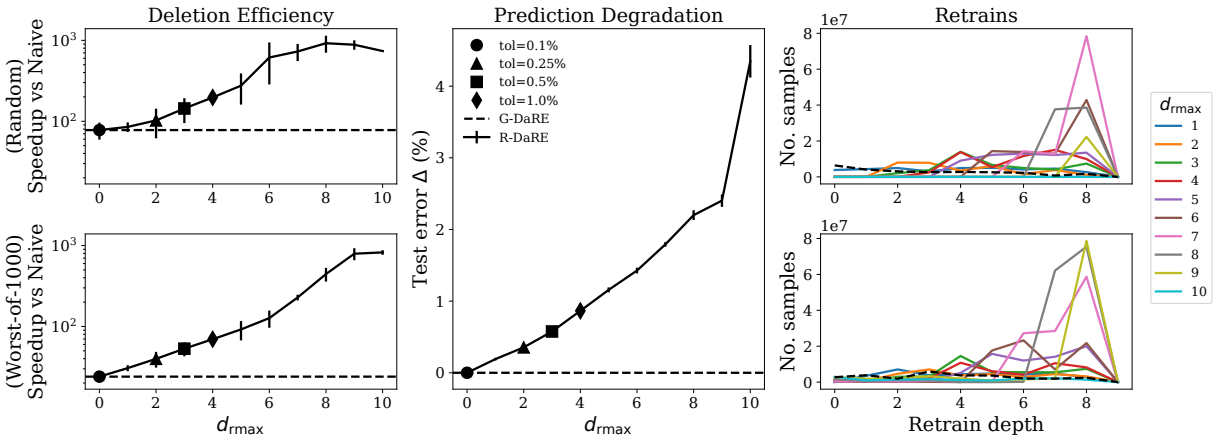



(h) Census

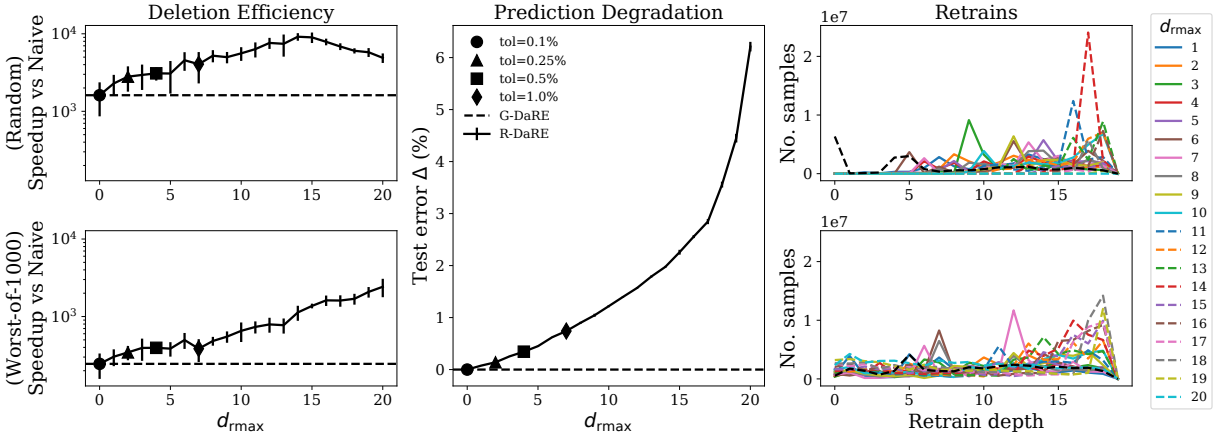



(i) Credit Card

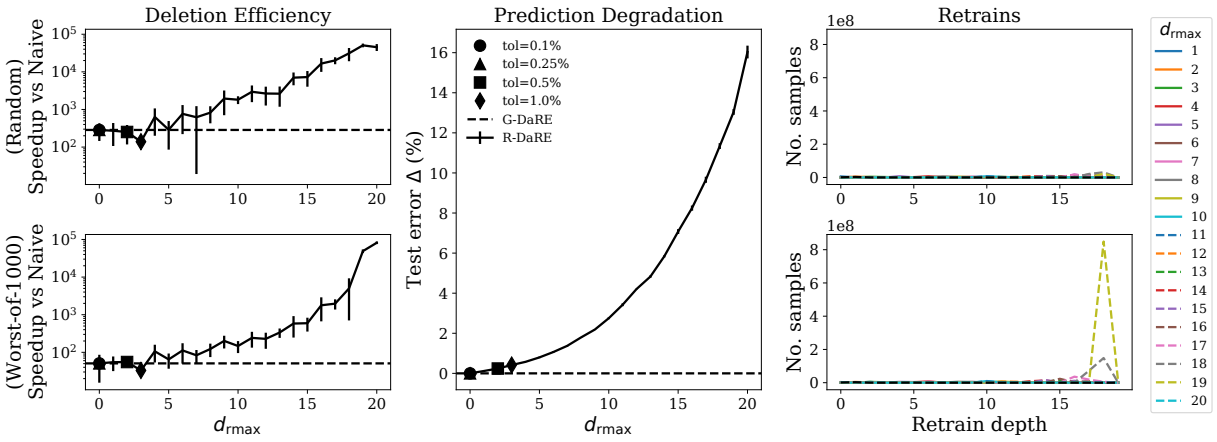
*Figure 4.* Effect of $d_{\text{rmax}}$ on deletion efficiency.




(j) CTR

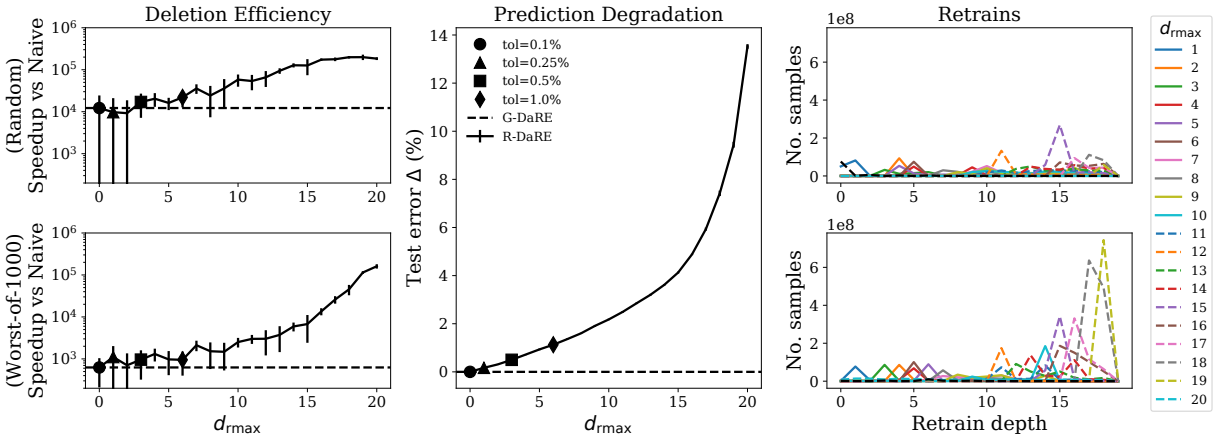



(k) Twitter




(l) Synthetic

*Figure 4.* Effect of $d_{\text{rmax}}$ on deletion efficiency.

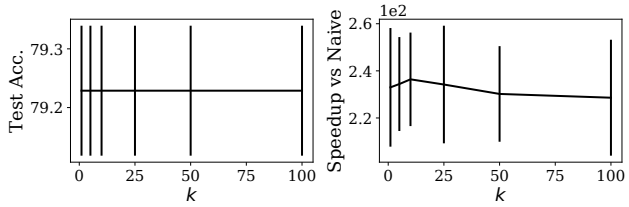



(m) Higgs

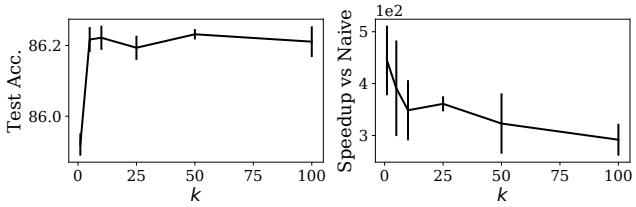
*Figure 4.* Effect of $d_{\text{rmax}}$ on deletion efficiency.

### B.4. Effect of $k$ on Deletion Efficiency

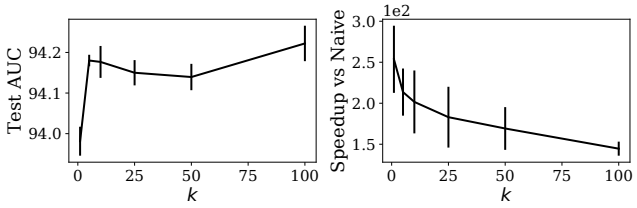
Figure 5 presents additional results on the effect $k$ has on deletion efficiency for different datasets. For $k$, we tested values [1, 5, 10, 25, 50, 100].



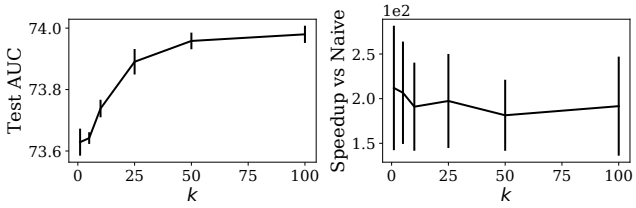

(a) Vaccine: All attributes are binary, thus $k$ has no effect.



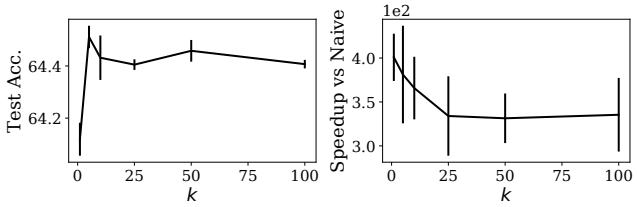

(b) Adult



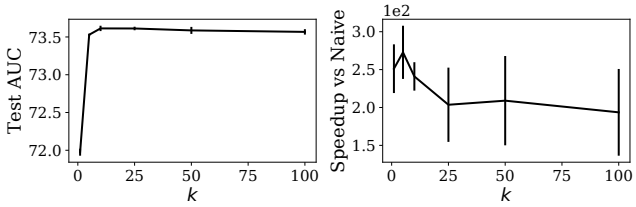

(c) Bank Marketing



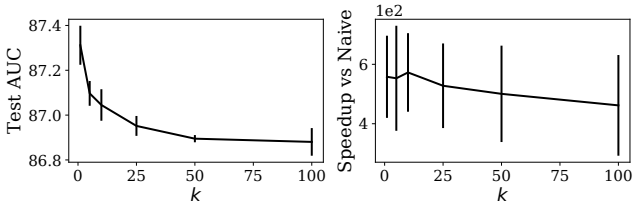

(d) Flight Delays



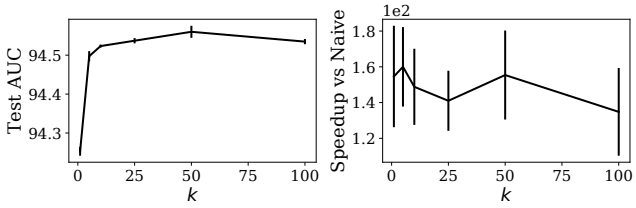

(e) Diabetes



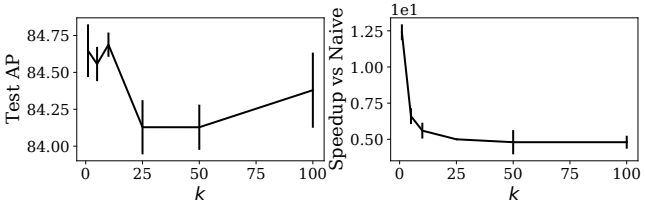

(f) No Show



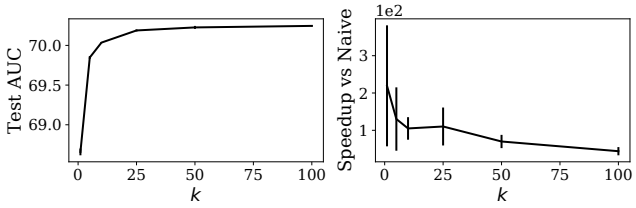

(g) Olympics: In this case, the randomness induced by a low $k$ value actually helps predictive performance.




(h) Census

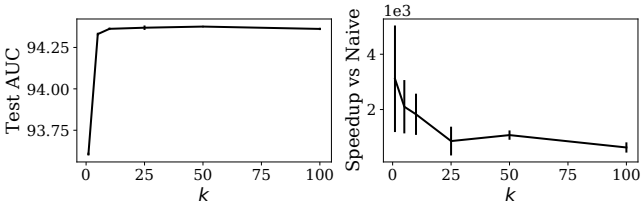



(i) Credit Card

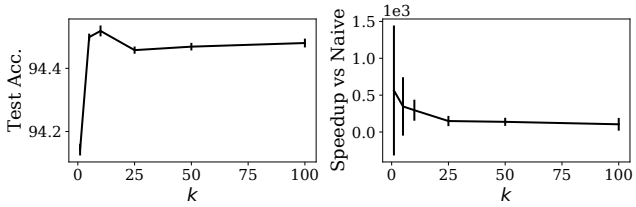



(j) CTR

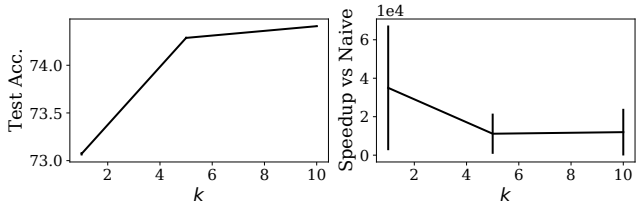
*Figure 5.* Effect of $k$ on deletion efficiency.




(k) Twitter




(l) Synthetic




(m) Higgs: G-DaRE RF exceeded available memory for the settings in which $k \in \{25, 50, 100\}$.

*Figure 5.* Effect of $k$ on deletion efficiency.

# C. Additional Experiments

## C.1. Entropy as the Split Criterion

We repeat our experiments using entropy instead of Gini index as the split criterion and find similar results as when using Gini index. In terms of predictive performance, we find nearly identical results to those in Table 5 with selected hyperparameters shown in Table 8. As for deletion efficiency, a summary of the deletion efficiency results is in Table 9; overall, we find the same trends as those in Table 2.

*Table 8.* Hyperparameters selected using entropy as the split criterion.

| | **G-DaRE & R-DaRE** | | | **R-DaRE Only** | | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | $T$ | $d_{\max}$ | $k$ | $d_{\text{rmax}}$ **(0.1%)** | $d_{\text{rmax}}$ **(0.25%)** | $d_{\text{rmax}}$ **(0.5%)** | $d_{\text{rmax}}$ **(1.0%)** |
| Surgical | 100 | 20 | 50 | 1 | 1 | 2 | 4 |
| Vaccine | 250 | 20 | 5 | 6 | 9 | 11 | 15 |
| Adult | 50 | 20 | 5 | 9 | 12 | 14 | 15 |
| Bank Marketing | 100 | 10 | 10 | 1 | 1 | 3 | 4 |
| Flight Delays | 250 | 20 | 50 | 1 | 3 | 5 | 10 |
| Diabetes | 100 | 20 | 5 | 4 | 10 | 11 | 14 |
| No Show | 250 | 20 | 10 | 1 | 3 | 6 | 9 |
| Olympics | 250 | 20 | 5 | 0 | 1 | 2 | 4 |
| Census | 100 | 20 | 25 | 5 | 8 | 11 | 15 |
| Credit Card | 250 | 10 | 25 | 1 | 2 | 3 | 4 |
| CTR | 100 | 10 | 25 | 2 | 3 | 4 | 6 |
| Twitter | 100 | 20 | 5 | 3 | 5 | 8 | 11 |
| Synthetic | 50 | 20 | 10 | 1 | 2 | 3 | 6 |
| Higgs | 50 | 20 | 10 | 0 | 2 | 5 | 8 |

*Table 9.* Summary of the deletion efficiency results using entropy as the split criterion.

| **Model** | **Min.** | **Max.** | **G. Mean** |
|---|---|---|---|
| **Random Adversary** | | | |
| G-DaRE | 81x | 9,986x | 715x |
| R-DaRE (tol=0.1%) | 93x | 9,986x | 834x |
| R-DaRE (tol=0.25%) | 93x | 21,104x | 1,177x |
| R-DaRE (tol=0.5%) | 99x | 16,897x | 1,265x |
| R-DaRE (tol=1.0%) | 128x | 37,953x | 1,819x |
| **Worst-of-1000 Adversary** | | | |
| G-DaRE | 19x | 790x | 76x |
| R-DaRE (tol=0.1%) | 24x | 790x | 101x |
| R-DaRE (tol=0.25%) | 25x | 1,348x | 135x |
| R-DaRE (tol=0.5%) | 29x | 1,473x | 175x |
| R-DaRE (tol=1.0%) | 37x | 1,783x | 262x |