DART: Data Addition and Removal Trees

Jonathan Brophy, [1] **Daniel Lowd** [1]
[1] University of Oregon
{jbrophy, lowd}@cs.uoregon.edu

Abstract

How can we update data for a machine learning model after it has already trained on that data? In this paper, we introduce DART, a variant of random forests that supports adding and removing training data with minimal retraining. Data updates in DART are exact, meaning that adding or removing examples from a DART model yields exactly the same model as retraining from scratch on updated data.

DART uses two techniques to make updates efficient. The first is to cache data statistics at each node and training data at each leaf, so that only the necessary subtrees are retrained. The second is to choose the split variable randomly at the upper levels of each tree, so that the choice is completely independent of the data and never needs to change. At the lower levels, split variables are chosen to greedily maximize a split criterion such as Gini index or mutual information. By adjusting the number of random-split levels, DART can trade off between more accurate predictions and more efficient updates. In experiments on ten real-world datasets and one synthetic dataset, we find that DART is orders of magnitude faster than retraining from scratch while sacrificing very little in terms of predictive performance.

1 Introduction

Recent legislation (California 2018; EU 2016; Canada 2018) requiring companies to remove private user data upon request has prompted new discussions on data privacy and ownership (Shintre, Roundy, and Dhaliwal 2019). Fulfilling this "right to be forgotten" (Garg, Goldwasser, and Vasudevan 2020; Kwak, Lee et al. 2017) may require updating any models trained on this data (Villaronga, Kieseberg, and Li 2018). Models also need to be updated as new data is acquired, to reflect new patterns and trends. Retraining a model from scratch on a revised dataset becomes prohibitively expensive as the size and complexity of a model increases (Shoeybi, Patwary et al. 2019), taking longer to train a new model and using more computational resources; this problem is exacerbated as the frequency of data updates or removal requests increases.

Decision trees and random forests (Breiman et al. 1984; Friedman 2001) are popular and widely used machine learning models (Lundberg, Erion, and Lee 2018), mainly due to their predictive prowess on many classification and regression tasks (Biau, Scornet, and Welbl 2019; Kocev, Vens et al. 2013; Genuer et al. 2017; Wager and Athey 2018; Linero and Yang 2018). Current work on deleting data from machine learning models has focused mainly on recommender systems (Cao and Yang 2015; Schelter 2020), K-means (Ginart et al. 2019), SVMs (Cauwenberghs and Poggio 2001), logistic regression (Guo et al. 2020; Schelter 2020), and deep neural networks (Baumhauer, Schöttle, and Zeppelzauer 2020; Golatkar, Achille, and Soatto 2020b; Wu, Dobriban, and Davidson 2020) (for a comprehensive list of related work, see Section 6); however, there is currently no work addressing the problem of efficiently deleting data from decision trees and random forests. Thus, we outline our contributions as follows:

- We propose DART (**D**ata **A**ddition and **R**emoval **T**rees), a stochastic variant of decision trees that supports efficient addition and removal of training instances. DART works with discrete tree structures, in contrast to many related works that assume continuous parameters. Our key ideas are to only retrain subtrees as needed and to strategically place completely random nodes near the top of the tree to avoid costly retraining.
- We provide algorithms for training a DART model, and subsequently updating that model given a training example to add or remove.
- We introduce CEDR, a baseline that implements certified removal in decision trees.
- We apply our approach to sequences of additions and deletions, evaluated on ten real-world datasets and one synthetic dataset, and find that our method can be over 50,000 times faster than retraining from scratch while achieving nearly the same predictive performance.

2 Problem Formulation

This section provides a primer on decision tree models, and introduces the problems faced when attempting to efficiently add or delete data from them.

We assume an *instance space* $\mathcal{X}$ with n samples defined over p attributes, $\{x_{i,1}, x_{i,2}, \ldots, x_{i,p}\}_{i=1}^{n}$. Without loss of generality, we assume that all attributes are binary. In binary classification, our goal is to find a function $f : \mathcal{X} \rightarrow \{-1, +1\}$ that maps each instance to either the positive $(+1)$ or negative (-1) class.

A *decision tree* is a tree-structured model in which each leaf is associated with a categorical or binary-valued prediction and each internal node is a decision node associated with an attribute a. The outgoing branches of the decision node define a partition over the values of the chosen attribute. 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 for the root to maximize an empirical *split criterion* such as Gini index (Breiman et al. 1984):

$$G_{\mathcal{D},C}(a) = \sum_{v \in a} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} \left(1 - \left(\sum_{c \in C} \frac{|\mathcal{D}_{v,c}|}{|\mathcal{D}_v|}\right)^2\right) \quad (1)$$

or entropy (Quinlan 2014):

$$H_{\mathcal{D},C}(a) = \sum_{v \in a} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} \left(\sum_{c \in C} -\frac{|\mathcal{D}_{v,c}|}{|\mathcal{D}_v|} \log_2 \frac{|\mathcal{D}_{v,c}|}{|\mathcal{D}_v|}\right) \quad (2)$$

in which $\mathcal{D}$ is the dataset input at a given decision node (in this case, the root node), $\mathcal{D}_v$ are the samples in $\mathcal{D}$ where attribute a takes on value v, and $\mathcal{D}_{v,c}$ are the samples in $\mathcal{D}$ where attribute a takes on value v and class label c. Once an attribute has been chosen for the root node, the data is partitioned into subsets based on the value of the attribute a, and a child node is learned for each data subset. The process terminates when the entire data subset has the same label or the tree reaches a specified maximum depth.

A *random forest* (RF) is an ensemble of decision trees which predicts the mean value . The prediction of the forest is the mean value over the predictions of all trees in the set. Two sources of randomness are used to increase diversity among the trees. The first source of randomness is in the training data: each tree in the ensemble is trained from a bootstrap sample of the original training data, with some examples excluded and some included multiple times. The second source of randomness is in the splits: each decision node is restricted to a random subset of attributes, and the split criterion is maximized over this subset rather than over all attributes.

We base our proposed methods on a minor variation of standard RFs, making two changes that simplify implementation without hurting accuracy. First, we do not use bootstrapping. Bootstrapping complicates adding and removing training examples, since one example may appear multiple times in the training data for one tree. There is empirical evidence that bootstrapping does not improve predictive performance (Denil, Matheson, and De Freitas 2014; Mentch and Hooker 2016; Zaman and Hirose 2009), and indeed our own preliminary experiments did not demonstrate any advantage of a tuned bootstrapped model over a tuned non-bootstrapped model. Second, we choose the random subset of attributes once per tree rather than once per node. In our preliminary experiments, we found that the predictive performance was very similar to the standard implementation in scikit-learn. Our methods could be extended to include RF models with bootstrapping and different feature sets at each node, but with some additional bookkeeping to keep track of the examples for each tree and the attributes for each node. Since predictive performance was already similar, we saw no need to add this complexity.

Naive Retraining Given an instance x to add/remove from a model, one can always add/delete x from the database and retrain from scratch; we denote this the *naive retraining* approach. This method is beneficial in that it is agnostic to virtually all machine learning models, and also easy to understand and implement. However, as previously stated, this approach becomes prohibitively expensive as the dataset size and/or model complexity increases, requiring more computational resources especially as addition/deletion requests increase.

3 DART

We now describe DART (**D**ata **A**ddition and **R**emoval **T**rees), an RF variant that supports efficient addition and removal of training examples. Data updates in DART are *exact*, meaning that adding or removing examples from a DART model yields exactly the same model as retraining from scratch on updated data (assuming a fixed random seed). This is equivalent to certified removal with $\epsilon = 0$.

A DART model is a tree ensemble where each tree is trained independently on a copy of the training data, restricted to a random subset of the attributes to encourage diversity among the trees. 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.

DART uses two techniques to make updates efficient: the first is to only retrain portions of the model where the structure must change to match the updated database; the second is to introduce random nodes that do not depend on the data and thus never need to be retrained. We describe each of these techniques in the following sections, and highlight the qualities and trade-offs they introduce. Pseudocode for training and updating DART models is listed in Algorithms 1 and 3, with full explanations of our methods below.

3.1 Retraining Minimal Subtrees

We first describe how to retrain only the parts of the model that need to be changed to match the updated training data. We do this by storing statistics at each node in the tree. For decision nodes, we store and update counts $\mathcal{D}_{v,c}$ for each attribute. This allows us to recompute the split criterion for each attribute without iterating through the data. For leaf nodes, we store and update the number of times each label occurs, along with a list of all training examples that match that leaf. These statistics are initialized when training the tree for the first time. We find this additional overhead has a negligible effect on training time.

When adding/deleting a sample x, these statistics are updated and used to check if a particular subtree needs retraining. Specifically, decision nodes affected by the addition/deletion of x update the statistics and recompute the split

Algorithm 1 Training a DART tree.

```
function TRAIN(dataset 𝒟, depth d)
    if stopping_criterion_reached then
        node ← LEAF_NODE(𝒟)                          ▷ Alg. 2
    else
        if d < topd then
            node ← RANDOM_NODE()                     ▷ Alg. 2
        else
            node ← DETERMINISTIC_NODE(𝒟)             ▷ Alg. 2
        node.left ← TRAIN(𝒟.left, d + 1)
        node.right ← TRAIN(𝒟.right, d + 1)
    return node
```

criterion for each attribute. If a different attribute obtains an improved split criterion over the currently chosen attribute, then we retrain the subtree rooted at this node. The training data for this subtree can be found by concatenating the example 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 example list are updated and the addition/deletion operation is complete. See Algorithm 3 for full pseudocode.

However, in the worst case, sequential additions or deletions could force a decision node to retrain with every update as the best attribute alternates between two. To reduce the severity of this worst case, we can add randomness.

3.2 Random Splits

Our second technique for efficient model updating is to choose the attribute for some of the decision nodes uniformly at random, independent of the split criterion. We refer to these as "random" nodes, in contrast to "deterministic" decision tree nodes that deterministically maximize the split criterion. Since random nodes do not depend on the statistics of the data, they never need to be retrained. DART uses random nodes for the upper layers of the tree and deterministic nodes for all other layers (excluding leaf nodes). Thus, we introduce *topd* as a hyperparameter indicating how many layers from the top each tree should use for random nodes.

Intuitively, nodes near the top of each tree contain more samples than nodes near the bottom, making them more expensive to retrain if necessary. Thus, we can significantly increase addition/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 deterministic nodes in all subsequent layers, essentially resulting in a greedy model built on top of a random projection of the input space (Haupt and Nowak 2006).

In our experiments, we compare DART models with random splits to those without, to evaluate the benefits of adding these random nodes. We refer to DART models with random nodes as random DART (R-DART) and those without as deterministic DART (D-DART). D-DART can also be viewed as a special case where the number of random layers, $topd$, is set to zero.

Algorithm 2 Creating deterministic, random, and leaf nodes.

```
function DETERMINISTIC_NODE(dataset 𝒟)
    node ← Node()
    node.meta ← save_statistics(𝒟)
    node.scores ← compute_scores(node.meta)
    node.best ← find_best_attribute(node.scores)
    return node

function RANDOM_NODE()
    node ← Node()
    node.best ← select_random_attribute()
    return node

function LEAF_NODE(dataset 𝒟)
    node ← Node()
    node.meta ← save_statistics(𝒟)
    node.value ← compute_leaf_value(node.meta)
    return node
```

4 Differential Privacy-Inspired Unlearning

Adding randomness is also a popular technique used to train differentially-private (DP) models (Dwork 2006; Chaudhuri, Monteleoni, and Sarwate 2011; Abadi et al. 2016), including decision trees and random forests (Fletcher and Islam 2019). However, DP-trained decision trees often have poor predictive performance due to the fact that the privacy budget ϵ must be split among all the trees in the forest, and among the different layers in each tree; the resulting model adds a significant amount of noise or uses a privacy budget too large to provide any meaningful guarantees (Fletcher and Islam 2015, 2019). Models that have achieved high predictive utility using DP-trained decision trees either have a weakened definition of DP (Rana, Gupta, and Venkatesh 2015) or use totally randomized trees (Fletcher and Islam 2017). When using totally randomized trees (Geurts, Ernst, and Wehenkel 2006), efficient data deletion becomes trivial; this is because no privacy is lost when all decision nodes are not dependent on the data to make splits, and leaf node statistics can be updated exactly. However, we find that using totally randomized trees does not often result in optimal predictive performance (Section B.2 in the Appendix), and that a middle-ground exists between totally random and totally greedy trees that provides efficient data deletion with high utility.

As previous work has noted, differential privacy is a sufficient condition for efficient data deletion, but not a necessary one (Ginart et al. 2019). Guo et al. (2020) have developed a framework for designing efficient removal-enabled models called ϵ-certified removal. The idea is that, given an instance to delete x, the resulting model is "close" to a model trained without x after using a carefully designed deletion mechanism M on x.

As another baseline, we developed a new DP-inspired ϵ-certified model that uses the exponential mechanism (Dwork 2006) to provide semi-random decision node splits (for details on this approach, see section B.3 in the appendix). We refer to this method as *CEDR* (**CE**rtified **D**ata **R**emoval), and include it in our empirical evaluation. However, we generally

Algorithm 3 Updating a DART tree.

Require: Start at the root node.

function UPDATE(node, depth d, add/remove sample x)
  if node is a leaf **then**
    node.meta $\leftarrow$ update_statistics(node.meta, x)
    node.value $\leftarrow$ update_leaf_value(node.meta)
    update_database(x)
  else
    finished $\leftarrow$ False
    if node is deterministic **then**
      node.meta $\leftarrow$ update_statistics(node.meta, x)
      node.scores $\leftarrow$ compute_scores(node.meta)
      best $\leftarrow$ find_best_attribute(node.scores)

      if best $\neq$ node.best **then**
        $\mathcal{D} \leftarrow$ get_data(node)
        $\mathcal{D} \leftarrow$ add_or_remove($\mathcal{D}$, x)
        node $\leftarrow$ TRAIN($\mathcal{D}$, d) ▷ Alg. 1
        update_database(x)
        finished $\leftarrow$ True

    if not finished **then**
      if $x_{\cdot,\text{node.best}}$ is True **then**
        node.left $\leftarrow$ UPDATE(node.left, $d+1$, x)
      else
        node.right $\leftarrow$ UPDATE(node.right, $d+1$, x)
  return node

find this approach to add too much noise to be useful, or require a budget too large to be meaningful.

5 Experiments

We attempt to answer the following research questions (RQ) whilst comparing DART to a set of meaningful baselines.

RQ1 Can we use D-DART to efficiently add/delete a significant number of examples as compared to naive retraining?

RQ2 Can we use R-DART to further increase addition/deletion efficiency over both naive retraining and D-DART while maintaining comparable predictive performance to these models?

Datasets Our experiments are conducted on ten 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; for numeric variables, we first partition their values into 5 quantiles, and then generate one-hot encodings based on these bin values. Also, 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 Section B.1 of the Appendix.

Hyperparameter Tuning Due to the range of label imbalances in our datasets (Appendix: Table 3), we measure the predictive peformance of our models using average precision (AP) (Zhu 2004) for datasets with a positive label percentage $< 1\%$, AUC (Hanley and McNeil 1982) for datasets between [1%, 20%), and accuracy for the remaining datasets. Using these measures, we tune the following hyperparameters: the maximum depth of each tree D and the number of trees in the forest T. Our protocol for tuning $topd$ is as follows: first, we tune a greedy model (i.e. by keeping $topd = 0$ fixed) using 5-fold cross-validation. Once the optimal values for D and T are found, we tune $topd$ by incrementing its value from zero to D, stopping when the model's cross-validation score exceeds a specified error tolerance as compared to the greedy model; for these experiments, we tune $topd$ using absolute error tolerances of 0.1%, 0.25%, 0.5%, and 1.0%. The following experiments use Gini index as the split criterion; however, we find similar results using entropy as well, which can be found in Section C.1 of the Appendix.

Table 1: Dataset Summary.

Dataset	n	p
Surgical	14,635	101
Vaccine	26,707	186
Bank Marketing	41,188	112
Adult	48,842	116
Flight Delays	100,000	658
Diabetes	101,766	568
Olympics	206,165	1,016
Credit Card	284,807	150
Census	299,285	414
Synthetic	1,000,000	210
Higgs	11,000,000	100

5.1 Removing Data

In this section, we focus on removing data and evaluate how efficiently R-DART can delete a sequential stream of training samples. Specifically, we measure how many samples our method can delete in the time it takes the naive retraining approach to delete one sample (i.e. retrain without that sample); the number of samples deleted gives us the speedup over the the naive approach. We also measure the predictive performance of each model prior to deletion and compare their predictive performance to the performance of the greedy model; we then repeat each experiment five times.

We determine the order of deletions using two different adversaries: *Random* and *Worst-of-1000*. The random adversary selects training samples to be deleted uniformly at random, while the worst-of-1000 adversary selects each sample by first selecting 1,000 candidate samples uniformly at random, and then choosing the sample that causes the greatest number of samples to be retrained across all trees in the forest.

Random Adversary We present the results of the deletion experiments using the random adversary in Figure 1 (top). We find that D-DART is at least two orders of magnitude faster than the naive retraining approach, while R-DART is faster than D-DART and all other methods to a varying degree depending on the dataset and error tolerance. We also

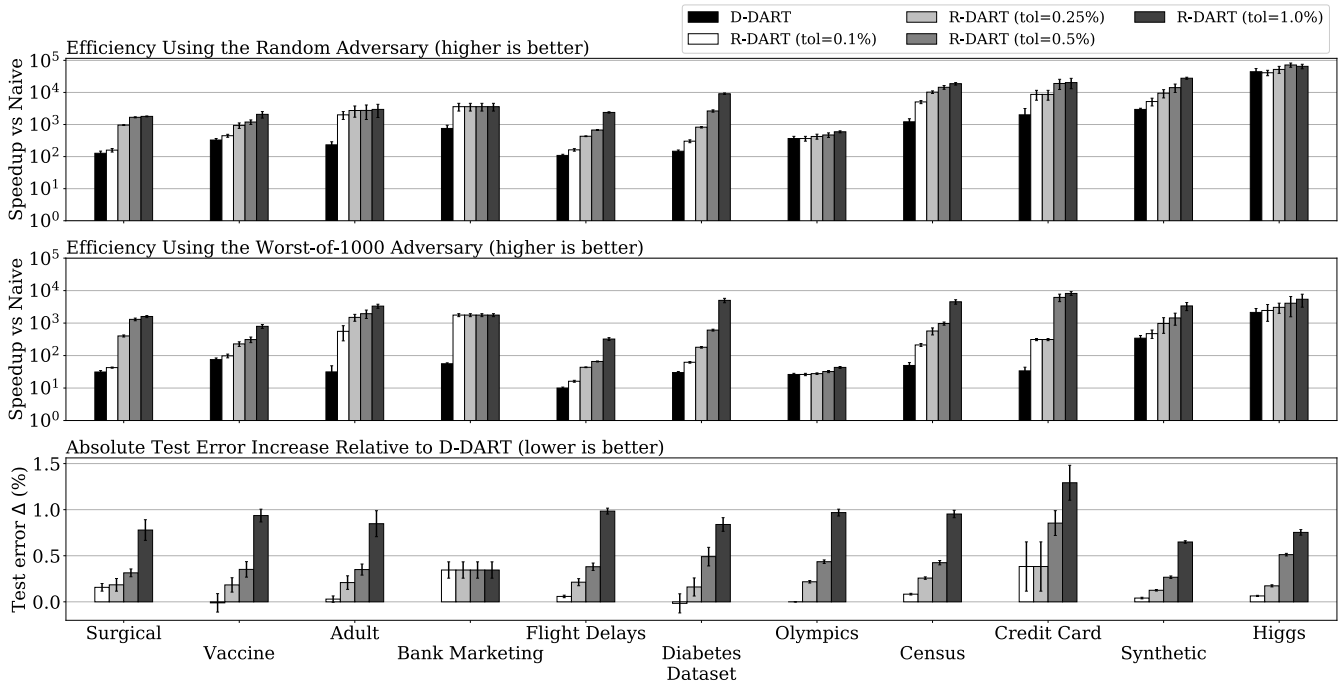

Figure 1: Deletion efficiency and utility using the random and worst-of-1000 adversaries. *Top, Middle*: Number of samples deleted in the time it takes the naive unlearning approach to delete one sample in the random and worst-of-1000 settings, respectively. *Bottom*: The increase in absolute test error relative to the greedy model.

note that R-DART is able to maintain comparable predictive performance to D-DART, typically staying within a test error difference of 1% depending on which tolerance is used to tune $topd$ (Figure 1: bottom).

We also report that the naive retraining approach took roughly 1.25 hours to delete a single example for the Higgs dataset. R-DART with $tol = 0.25\%$ (resulting in $topd = 2$) deleted over 50,000 samples in that time, an average of 0.09s per deletion, while the average test set error increased by only 0.174%. In this case, R-DART provides a speedup of over four orders of magnitude, providing a tractable solution for something previously intractable.

Worst-of-1000 Adversary The results for the deletion experiments using the worst-of-1000 adversary are in Figure 1 (middle). First, we notice the efficiency of D-DART drops significantly under this much more challenging adversary, often resulting in speedups of over one order of magnitude as compared to the naive retraining approach. While the R-DART models also decrease in efficiency, they maintain a significant efficiency advantage over D-DART by an order of magnitude for most datasets and by over two orders of magnitude for the Credit Card dataset.

Summary of Deletion Results A summary of the deletion efficiency results are in Table 2. When the instances to delete are chosen randomly, D-DART is more than 600x faster than naively retraining after every deletion (taking the geometric mean over the 11 datasets). By adding randomness, R-DART achieves even larger speedups, from 1,500x up to nearly 6,000x, depending on the performance tolerance (0.1% to 1.0%). A more sophisticated worst-of-1000 adversary that always deletes the worst instance out of a random set of 1,000 can force more frequent retraining. In this case, D-DART is 60x faster than naive retraining, and R-DART ranges from 180x to 1,700x depending on the tolerance. The CEDR baseline was much less effective, averaging only a 15x and 8x speedup over naive retraining for the random and worst-of-1000 adversaries, respectively; more detailed results and analyses of the CEDR baseline are in Section B.3 of the Appendix.

Effect of $topd$ on Deletion Efficiency Figure 2 shows a detailed analysis of the effect $topd$ has on deletion efficiency under each adversary for the Credit Card datastet. As expected, we see that deletion efficiency increases as $topd$ increases. Test error also increases as $topd$ increases, but initially degrades gracefully, maintaining a low increase in relative test error even as the top ten layers of each tree are replaced with random nodes.

Figure 2 also shows the number of retrains that occur at each depth in each tree, across all trees in the model. We immediately notice the increase in retrainings 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 samples; each sample 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.

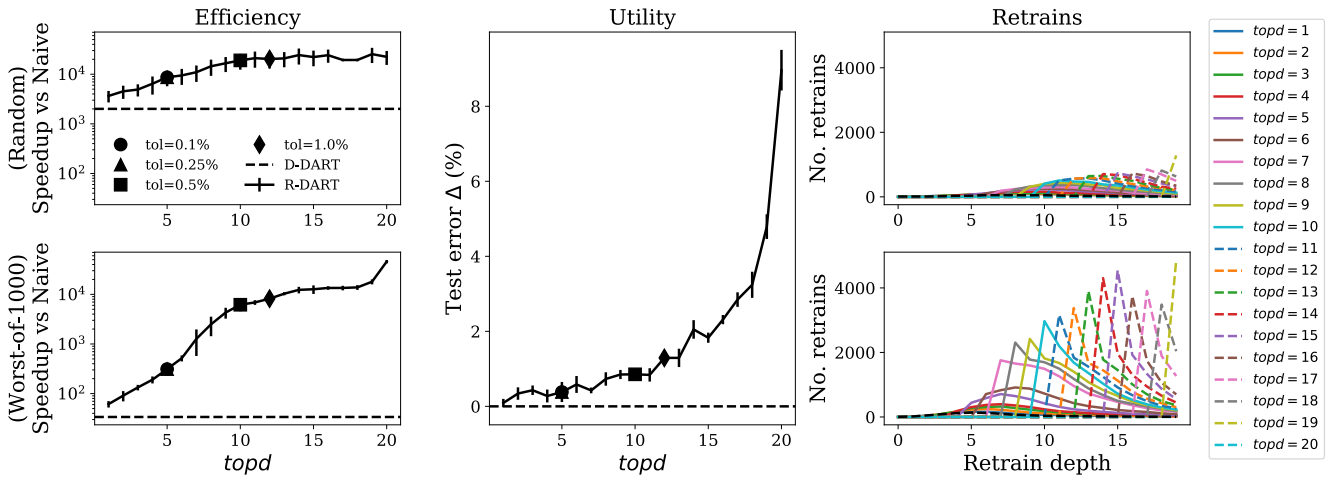

Figure 2: Effect of *topd* on deletion efficiency (left), utility (middle), and retrain depth (right) using the random (top) and worst-of-1000 (bottom) adversaries on the Credit Card dataset. Utility is independent of the adversary, as predictive performance is measured before deletion.

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

Model	Min	Max	Geo. Mean
Random Adversary			
CEDR ($\epsilon = 10$)	1x	21,465x	15x
D-DART	109x	44,228x	677x
R-DART (tol=0.1%)	159x	41,108x	1,500x
R-DART (tol=0.25%)	424x	52,136x	2,713x
R-DART (tol=0.5%)	472x	71,710x	4,037x
R-DART (tol=1.0%)	593x	65,101x	5,979x
Worst-of-1000 Adversary			
CEDR ($\epsilon = 10$)	1x	1,642x	8x
D-DART	10x	2,131x	61x
R-DART (tol=0.1%)	16x	2,446x	184x
R-DART (tol=0.25%)	28x	3,051x	386x
R-DART (tol=0.5%)	32x	6,174x	778x
R-DART (tol=1.0%)	43x	8,185x	1,704x

5.2 Adding Data

To test data addition with R-DART, we perform the same set of experiments as the data removal case, now *adding* new training samples instead of deleting them. The random adversary chooses each training sample to add by sampling from the existing set of training instances uniformly at random, adding a copy of that random instance to the training data. The worst-of-1000 adversary chooses each sample to add by selecting each sample from a pool of 1,000 candidate samples (randomly chosen from the set of existing training instances) that causes the greatest number of samples to be retrained as a result of *adding* that sample. We find that our results follow the same trends as in the deletion case. On averge, D-DART is 1-2 orders of magnitude faster than the naive retraining approach and R-DART is 2-3 orders of magnitude faster than naive depending on the dataset and chosen tolerance, all while incurring small increases in test error. More details on the data addition results can be found in Section B.4 of the appendix.

6 Related Work

Exact Unlearning There have been a number of works that support decremental learning of SVMs (Cauwenberghs and Poggio 2001; Chen et al. 2019; Duan et al. 2007; Karasuyama and Takeuchi 2009; Romero, Barrio, and Belanche 2007; Tveit, Hetland, and Engum 2003), aimed at accelerating leave-one-out cross-validation (Shao 1993). More recently, Cao and Yang (2015) first introduced the concept of model unlearning, recognizing that a certain class of models (e.g. naive Bayes) fall under the umbrella of SQ-learning (Kearns 1998), where data deletion is efficient and exact; Schelter (2020) has also developed decremental update procedures for similar classes of models. Ginart et al. (2019) developed a quantized variant of the k-means (Lloyd 1982) algorithm called Q-k-means that supports exact data deletion; their method relies on the notion of model *stability*, with the assumption that the deletion of a small enough proportion of data should not cause a large change in the learned model.

Unlearning for Stochastically-Trained Models Bourtoule et al. (2021) propose SISA (sharded, isolated, sliced, and aggregated) training. Their method breaks the data into shards and then further into slices, from which they train an ensemble of models and save snapshots of each model for every slice. The biggest drawbacks of their approach are the large storage costs and applicability only to iterative learning algorithms. Golatkar, Achille, and Soatto (2020a,b) propose a scrubbing mechanism for deep neural networks that does not require any retraining; however, the computational com-

plexity of their approach is currently quite high. Guo et al. (2020), Izzo et al. (2020), and Wu, Dobriban, and Davidson (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.

Mitigation Baumhauer, Schöttle, and Zeppelzauer (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, Yao et al. (2019) and Du et al. (2019) perform similar unlearning techniques to mitigate backdoor attacks (Gu, Dolan-Gavitt, and Garg 2017) on image classifiers and anomaly detectors, respectively. Wang et al. do this by fine-tuning their model on corrected versions of the poisoned training instances; Du et al. update their model by minimizing the probability of false positives in the training data. Although both approaches show promising empirical performance, they provide no guarantees about the extent to which these samples are removed from the model (Sommer et al. 2020). Tople, Brockschmidt et al. (2019) analyze privacy leakage in language model snapshots before and after they are updated.

7 Discussion

Protecting private user data continues to be increasingly important (Shah 2020); for example, Google has implemented a system that removes potentially damaging URLs from their search engine upon request (Bertram et al. 2019). Thus, data deletion from learned models has significant privacy implications, especially as many machine learning models are vulnerable to different types of "model inversion" and "membership inference" attacks (Carlini, Liu et al. 2018; Shokri et al. 2017; Tople, Brockschmidt et al. 2019; Veale, Binns, and Edwards 2018; Yeom, Giacomelli et al. 2018) which can extract private data from a learned model. DART models guarantee safeguards against these types of classical membership inference attacks and reduce the need for deletion-verification methods (Shintre and Dhaliwal 2019; Sommer et al. 2020); however, if an adversary has access to the updated model *and* the original model (the model trained on all the data including the data to be added/deleted), there is evidence that suggests some level of privacy leakage, even using the gold standard of retraining from scratch (Chen, Zhang et al. 2020).

We have explored the robustness of our method to a small extent by using a simple but effective adversary, but we encourage researchers to develop stronger adversaries, especially ones that emulate plausible scenarios from possible real-world deployments of an addition/deletion-enabled model. In addition to providing strong privacy guarantees, DART (and efficient data deletion in general) can impact a number of different machine learning areas.

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

Dataset Cleaning User privacy may not be the only reason for deleting samples; it may also be beneficial to efficiently remove outliers identified by outlier detectors (Rahmani and Li 2019; Dong, Hopkins, and Li 2019). One may also want to remove noisy / problematic training instances caused by randomness or dataset poisoning attacks (Mozaffari-Kermani, Sur-Kolay et al. 2014; Steinhardt, Koh, and Liang 2017) which may be more efficiently identified using instance-attribution explanations (Koh and Liang 2017; Sharchilev et al. 2018; Yeh, Kim et al. 2018).

Continual Learning In addition to deleting noisy training instances, our approach can also efficiently add training samples, making it particularly appealing for any online learning setting with streaming data in which continuous updating is desirable (Chrysakis and Moens 2020; Knoblauch, Husain, and Diethe 2020). However, one must be aware that a different set of hyperparameters may be become optimal as a distributional shift of the training data may occur as a result of adding/deleting more and more samples.

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. We believe our work not only has positive implications for user privacy, but also opens up new possibilities for continuous model learning while having a positive impact on the environment (Gupta, Lanteigne, and Kingsley 2020).

8 Conclusion

In this work, we developed DART, a random forest variant that supports efficient updating in response to repeated additions and deletions of training examples. We find that, on average, DART models are 1-2 orders of magnitude faster than the naive retraining approach with no loss in accuracy, and models 2-3 orders of magnitude faster if slightly worse predictive performance is tolerated. Certified removal methods based on differential privacy have been effective in linear models, but we found them to be much less effective in trees.

For future work, there are many exciting opportunities and applications of DART, from maintaining user privacy to building interpretable models to cleaning data, all without retraining from scratch. One could even investigate the possibility of extending DART to boosted trees (Chen and Guestrin 2016; Ke, Meng et al. 2017; Prokhorenkova, Gusev et al. 2018). At its best, DART was 50,000 times faster than naive retraining, so it has the potential to enable new applications of model updating that were previously intractable.

References

Abadi, M.; Chu, A.; Goodfellow, I.; et al. 2016. Deep Learning with Differential Privacy. In *CCS*.

Baldi, P.; Sadowski, P.; and Whiteson, D. 2014. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Communications* .

Baumhauer, T.; Schöttle, P.; and Zeppelzauer, M. 2020. Machine Unlearning: Linear Filtration for Logit-based Classifiers. *arXiv preprint arXiv:2002.02730* .

Bertram, T.; Bursztein, E.; Caro, S.; et al. 2019. Five Years of the Right to be Forgotten. In *CCS*.

Bhatt, R. B.; Sharma, G.; et al. 2009. Efficient Skin Region Segmentation Using Low Complexity Fuzzy Decision Tree Model. In *IEEE India Conference*.

Biau, G.; Scornet, E.; and Welbl, J. 2019. Neural Random Forests. *Sankhya A* .

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

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

Bull, P.; Slavitt, I.; and Lipstein, G. 2016. Harnessing the Power of the Crowd to Increase Capacity for Data Science in the Social Sector. In *ICML #Data4Good Workshop*.

California. 2018. California Consumer Privacy Act. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375. [Online; accessed 16-April-2020].

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

Cao, Y.; and Yang, J. 2015. Towards Making Systems Forget with Machine Unlearning. In *IEEE Symposium on Security and Privacy*.

Carlini, N.; Liu, C.; et al. 2018. The Secret Sharer: Measuring Unintended Neural Network Memorization & Extracting Secrets. *arXiv preprint arXiv:1802.08232* .

Cauwenberghs, G.; and Poggio, T. 2001. Incremental and Decremental Support Vector Machine Learning. In *NeurIPS*.

Chaudhuri, K.; Monteleoni, C.; and Sarwate, A. D. 2011. Differentially Private Empirical Risk Minimization. *JMLR* .

Chen, M.; Zhang, Z.; et al. 2020. When Machine Unlearning Jeopardizes Privacy. *arXiv preprint arXiv:2005.02205* .

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*.

Chen, Y.; Xiong, J.; Xu, W.; and Zuo, J. 2019. A Novel Online Incremental and Decremental Learning Algorithm Based on Variable Support Vector Machine. *Cluster Computing* .

Chrysakis, A.; and Moens, M.-F. 2020. Online Continual Learning from Imbalanced Data. In *ICML*.

Denil, M.; Matheson, D.; and De Freitas, N. 2014. Narrowing the Gap: Random Forests in Theory and in Practice. In *ICML*.

Dong, Y.; Hopkins, S.; and Li, J. 2019. Quantum Entropy Scoring for Fast Robust Mean Estimation and Improved Outlier Detection. In *NeurIPS*.

DrivenData. 2019. Flu Shot Learning: Predict H1N1 and Seasonal Flu Vaccines. https://www.drivendata.org/competitions/66/flu-shot-learning/data/. [Online; accessed 12-August-2020].

Du, M.; Chen, Z.; Liu, C.; Oak, R.; and Song, D. 2019. Lifelong Anomaly Detection Through Unlearning. In *CCS*.

Dua, D.; and Graff, C. 2019. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml.

Duan, H.; Li, H.; He, G.; and Zeng, Q. 2007. Decremental Learning Algorithms for Nonlinear Langrangian and Least Squares Support Vector Machines. In *OSB*.

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

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

Fletcher, S.; and Islam, M. Z. 2015. A Differentially Private Decision Forest. In *AusDM*.

Fletcher, S.; and Islam, M. Z. 2017. Differentially Private Random Decision Forests Using Smooth Sensitivity. *Expert Systems with Applications* .

Fletcher, S.; and Islam, M. Z. 2019. Decision Tree Classification with Differential Privacy: A Survey. *ACM Computing Surveys* .

Friedman, J. H. 2001. Greedy Function Approximation: a Gradient Boosting Machine. *Annals of Statistics* .

Garg, S.; Goldwasser, S.; and Vasudevan, P. N. 2020. Formalizing Data Deletion in the Context of the Right to be Forgotten. *arXiv preprint arXiv:2002.10635* .

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

Geurts, P.; Ernst, D.; and Wehenkel, L. 2006. Extremely Randomized Trees. *Machine learning* .

Ginart, A.; Guan, M.; Valiant, G.; and Zou, J. Y. 2019. Making AI Forget You: Data Deletion in Machine Learning. In *NeurIPS*.

Golatkar, A.; Achille, A.; and Soatto, S. 2020a. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. In *CVPR*.

Golatkar, A.; Achille, A.; and Soatto, S. 2020b. Forgetting Outside the Box: Scrubbing Deep Networks of Information Accessible from Input-Output Observations. *arXiv preprint arXiv:2003.02960* .

Gu, T.; Dolan-Gavitt, B.; and Garg, S. 2017. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. In *Machine Learning and Computer Security Workshop*.

Guo, C.; Goldstein, T.; Hannun, A.; and van der Maaten, L. 2020. Certified Data Removal from Machine Learning Models. In *ICML*.

Gupta, A.; Lanteigne, C.; and Kingsley, S. 2020. SECure: A Social and Environmental Certificate for AI Systems. In *ICML Deploying and Monitoring Machine Learning Systems Workshop*.

Hanley, J.; and McNeil, B. 1982. The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve. *Radiology* .

Haupt, J.; and Nowak, R. 2006. Signal Reconstruction from Noisy Random Projections. *IEEE Transactions on Information Theory* .

Huerta, R.; Mosqueiro, T.; et al. 2016. Online Decorrelation of Humidity and Temperature in Chemical Sensors for Continuous Monitoring. *Chemometrics and Intelligent Laboratory Systems* .

Izzo, Z.; Smart, M. A.; Chaudhuri, K.; and Zou, J. 2020. Approximate Data Deletion from Machine Learning Models: Algorithms and Evaluations. *arXiv preprint arXiv:2002.10077* .

Kaggle. 2018a. 120 Years of Olympic History: Athletes and Events. https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results. [Online; accessed 28-July-2020].

Kaggle. 2018b. Credit Card Fraud Detection. https://www.kaggle.com/mlg-ulb/creditcardfraud/. [Online; accessed 27-July-2020].

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

Karasuyama, M.; and Takeuchi, I. 2009. Multiple Incremental Decremental Learning of Support Vector Machines. In *NeurIPS*.

Ke, G.; Meng, Q.; et al. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *NeurIPS*.

Kearns, M. 1998. Efficient Noise-Tolerant Learning from Statistical Queries. *Journal of the ACM (JACM)* .

Knoblauch, J.; Husain, H.; and Diethe, T. 2020. Optimal Continual Learning has Perfect Memory and is NP-hard. In *ICML*.

Kocev, D.; Vens, C.; et al. 2013. Tree Ensembles for Predicting Structured Outputs. *Pattern Recognition* .

Koh, P. W.; and Liang, P. 2017. Understanding Black-Box Predictions via Influence Functions. In *ICML*.

Kwak, C.; Lee, J.; et al. 2017. Let Machines Unlearn–Machine Unlearning and the Right to be Forgotten. *SIGSEC* .

Linero, A. R.; and Yang, Y. 2018. Bayesian Regression Tree Ensembles that Adapt to Smoothness and Sparsity. *Journal of the Royal Statistical Society* .

Lloyd, S. 1982. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory* .

Lundberg, S. M.; Erion, G. G.; and Lee, S.-I. 2018. Consistent Individualized Feature Attribution for Tree Ensembles. *arXiv preprint arXiv:1802.03888* .

Mentch, L.; and Hooker, G. 2016. Quantifying Uncertainty in Random Forests via Confidence Intervals and Hypothesis Tests. *Journal of Machine Learning Research* .

Moro, S.; Cortez, P.; et al. 2014. A Data-Driven Approach to Predict the Success of Bank Telemarketing. *Decision Support Systems* .

Mozaffari-Kermani, M.; Sur-Kolay, S.; et al. 2014. Systematic Poisoning Attacks on and Defenses for Machine Learning in Healthcare. *Journal of Biomedical and Health Informatics* .

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; et al. 2011. Scikit-Learn: Machine Learning in Python. *JMLR* .

Prokhorenkova, L.; Gusev, G.; et al. 2018. CatBoost: Unbiased Boosting with Categorical Features. In *NeurIPS*.

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

Rahmani, M.; and Li, P. 2019. Outlier Detection and Robust PCA Using a Convex Measure of Innovation. In *NeurIPS*.

Rana, S.; Gupta, S. K.; and Venkatesh, S. 2015. Differentially Private Random Forest with High Utility. In *ICDM*.

Research; and Administration, I. T. 2019. 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. [Online; accessed 16-April-2020].

Romero, E.; Barrio, I.; and Belanche, L. 2007. Incremental and Decremental Learning for Linear Support Vector Machines. In *International Conference on Artificial Neural Networks*.

Schelter, S. 2020. "Amnesia" - Machine Learning Models That Can Forget User Data Very Fast. In *CIDR*.

Sedhai, S.; and Sun, A. 2015. Hspam14: A Collection of 14 Million Tweets for Hashtag-Oriented Spam Research. In *SIGIR*.

Shah, V. K. 2020. *User Acceptance of Online Tracking if "Forgetting" was an Option*. Ph.D. thesis, Carleton University.

Shao, J. 1993. Linear Model Selection by Cross-Validation. *Journal of the American Statistical Association* .

Sharchilev, B.; Ustinovskiy, Y.; Serdyukov, P.; and de Rijke, M. 2018. Finding Influential Training Samples for Gradient Boosted Decision Trees. In *ICML*.

Shintre, S.; and Dhaliwal, J. 2019. Verifying That the Influence of a User Data Point Has Been Removed from a Machine Learning Classifier.

Shintre, S.; Roundy, K. A.; and Dhaliwal, J. 2019. Making Machine Learning Forget. In *Annual Privacy Forum*.

Shoeybi, M.; Patwary, M.; et al. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using GPU Model Parallelism. *arXiv preprint arXiv:1909.08053* .

Shokri, R.; Stronati, M.; Song, C.; and Shmatikov, V. 2017. Membership Inference Attacks Against Machine Learning Models. In *IEEE Symposium on Security and Privacy*.

Sommer, D. M.; Song, L.; Wagh, S.; and Mittal, P. 2020. Towards Probabilistic Verification of Machine Unlearning. *arXiv preprint arXiv:2003.04247* .

Steinhardt, J.; Koh, P. W. W.; and Liang, P. S. 2017. Certified Defenses for Data Poisoning Attacks. In *NeurIPS*.

Strack, B.; DeShazo, J. P.; et al. 2014. Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records. *BioMed research international* 2014.

Tople, S.; Brockschmidt, M.; et al. 2019. Analyzing Privacy Loss in Updates of Natural Language Models. *arXiv preprint arXiv:1912.07942* .

Tveit, A.; Hetland, M. L.; and Engum, H. 2003. Incremental and Decremental Proximal Support Vector Classification Using Decay Coefficients. In *DaWaK*.

Veale, M.; Binns, R.; and Edwards, L. 2018. Algorithms that Remember: Model Inversion Attacks and Data Protection Law. *Philosophical Transactions of the Royal Society A* .

Villaronga, E. F.; Kieseberg, P.; and Li, T. 2018. Humans Forget, Machines Remember: Artificial Intelligence and the Right to be Forgotten. *Computer Law & Security Review* .

Wager, S.; and Athey, S. 2018. Estimation and Inference of Heterogeneous Treatment Effects Using Random Forests. *Journal of the American Statistical Association* .

Wang, B.; Yao, Y.; et al. 2019. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *IEEE Symposium on Security and Privacy*.

Wu, Y.; Dobriban, E.; and Davidson, S. B. 2020. DeltaGrad: Rapid Retraining of Machine Learning Models. In *ICML*.

Yeh, C.-K.; Kim, J.; et al. 2018. Representer Point Selection for Explaining Deep Neural Networks. In *NeurIPS*.

Yeom, S.; Giacomelli, I.; et al. 2018. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In *CSF*.

Zaman, F.; and Hirose, H. 2009. Effect of Subsampling Rate on Subbagging and Related Ensembles of Stable Classifiers. In *PReMI*.

Zhu, M. 2004. Recall, Precision and Precision, Average. *University of Waterloo, Waterloo* .