Machine Unlearning in Semi Supervised Decision Tree Based Models

by

Nurassyl Nurmaganbet

Submitted to the Department of Electrical and Computer Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical and Computer Engineering

at the

NAZARBAYEV UNIVERSITY

Apr 2025

© Nazarbayev University 2025. All rights reserved.

Author .....
Department of Electrical and Computer Engineering
Apr 13, 2025

Certified by .....
Amin Zollanvari
Associate Professor
Thesis Supervisor

Certified by .....
Berdakh Abibullaev
Associate Professor
Thesis Co-Supervisor

Accepted by .....
Mehdi Bagheri
Associate Professor, School of Engineering and Digital Sciences

Machine Unlearning in Semi Supervised Decision Tree Based Models

by

Nurassyl Nurmaganbet

Submitted to the Department of Electrical and Computer Engineering
on Apr 13, 2025, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical and Computer Engineering

Abstract

Under strict data privacy regulations such as the General Data Protection Regulation, it is important not only to delete data but also to mitigate its impact on machine learning models. These regulations present a challenging problem because of the memorization capabilities of the machine learning models. While retraining models from scratch ensures the removal of the impact of data from the model, their computational cost underscores the need for efficient removal techniques. Existing methods that modify the dataset or target simpler models are less effective for models with hierarchical structures. In these models, even one data point removal can affect split decisions, leading to complete retraining. Moreover, existing methods are not well suited to a semi-supervised setting. This setting involves using both labeled and unlabeled data to shape the model architecture. This thesis addresses these gaps by proposing a novel unlearning method for semi-supervised tree-based models, aiming to improve computational efficiency while retaining the model's predictive performance. This work first analyzes supervised unlearning methods. After that, it adapts these methods to semi-supervised settings. Next, we evaluate the unlearning performance across multiple datasets. The goal is to enable efficient data removal, which avoids full model retraining while preserving the model's predictive performance.

Thesis Supervisor: Amin Zollanvari
Title: Associate Professor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the era of big data and machine learning, it has become increasingly important to preserve users privacy. In addition to preserving privacy, organizations must comply with regulations such as the General Data Protection Regulation (GDPR) [1]. One of the important aspects of GDPR is the *right to be forgotten* rule. This rule grants the user the right to request the deletion of personal data concerning them from the records of any organization [2]. While removing the data from the dataset is a simple task, removing its impact from the machine learning models trained on that data is far more complex. Removed data may still be reflected in a model's parameters or learned structure, as models tend to memorize patterns from the training data [3].

One of the straightforward approaches to solve this task is retraining the model. This solution involves rebuilding the model from scratch, excluding the removal data points. However, because of its high computational cost, this approach is often considered impractical. Depending on dataset size and model complexity, retraining can take anywhere from a few seconds to hundreds of hours, making it inefficient for frequent single data removal requests. This inefficiency underscores the need for *machine unlearning*—methods that selectively remove data influence without full retraining. Machine unlearning ensures compliance with privacy regulations by completely erasing the impact of removed data [4].

Earlier approaches to data removal involved altering datasets [5], [6], and primarily focused on models with relatively simple parameter structures, such as k-means [7].

Even though data-centric methods provide the unlearning with a simple implementation, they lead to reduced predictive performance when removing large data subsets. Moreover, these methods may not be suitable for models with hierarchical structures, such as decision trees. Decision tree models rely on splits that partition the dataset, meaning the removal of even a single data point can alter the entire tree's structure. As a result, data removal in decision trees often requires retraining.

A further complication arises in a semi-supervised setting. In this setting, models are trained on datasets with limited labeled data and abundant unlabeled data. Semi-supervised learning uses both labeled and unlabeled data to improve model robustness and predictive performance [8], thereby offering advantages over traditional supervised models. However, these advantages of semi-supervised learning complicates the unlearning process. In semi-supervised models, data removal must account not only for labeled data but also for the unlabeled data because both data types are used in shaping the model's architecture. Removing just a few training instances, whether labeled or unlabeled, can cause cascading effects throughout the entire tree structure.

These challenges highlight why efficiently unlearning in semi-supervised tree-based models remains an open research problem. Existing unlearning methods are suitable only for supervised settings. To bridge this gap, this thesis aims to analyze and develop an unlearning method tailored to semi-supervised tree-based models. This thesis has two primary goals: first, to propose an unlearning method that can remove the influence of specific data from a semi-supervised tree model while preserving its predictive performance; and second, to design an unlearning method that is computationally more efficient than retraining the entire semi-supervised tree.

The organization of this thesis is as follows: In Chapter 2, the thesis explores previously proposed unlearning algorithms for supervised decision trees to assess their effectiveness in data removal. Informed by insights from previous methods, Chapter 3 covers how to implement unlearning in semi-supervised decision tree-based models. Chapter 4 presents the experimental setup, including detailed information about the datasets. Chapter 5 discusses the results, while Chapter 6 concludes the thesis.

1.1 Literature Review

Machine unlearning algorithms have become increasingly important with the rise of privacy-preserving challenges in machine learning models [9]. Traditional machine learning models often memorize the data samples used during training, making data removal particularly difficult [10]. While retraining models from scratch after data deletion ensures data removal, this approach is computationally expensive, underscoring the critical need for efficient unlearning techniques. This issue is particularly important in the context of data privacy and compliance with regulations, such as GDPR, which enforces strict data privacy and the right to be forgotten principle. To deal with this issue, this literature review explores several unlearning techniques and the methods available today.

The very first efficient unlearning algorithm was proposed by Cao and Yang [11] and was implemented using the summation-based approach. The key point here is that the authors used a *statistical query (SQ)* learning method—a framework where algorithms learn models by querying aggregate statistics over the dataset rather than accessing individual data points directly [12]. The SQ learning method is used to transform the learning process into summation-based forms so that the learning algorithms depend on the summation points, not the individual data points. Upon receiving the unlearning request, the system simply subtracts the relevant summation points and subsequently retrains the model. This method significantly reduces data removal time compared to the time required for retraining the entire model, as demonstrated by achieving a six-fold speedup in the LensKit recommendation system. However, this approach is limited to models convertible to summation forms, excluding many common models, such as decision trees. Because the decision tree models with n nodes and c classes have a statistical query dimension of $n^c logn$, which means the complexity for a class function is too high for an efficient SQ model [13].

To address broader model compatibility, the "Sharded, Isolated, Sliced, and Aggregated" (SISA) algorithm was proposed by Bourtoule *et al.* [5]. However, instead of changing the training structure, SISA restructures the dataset itself. The SISA

algorithm is shown in Fig. 1.1.1.

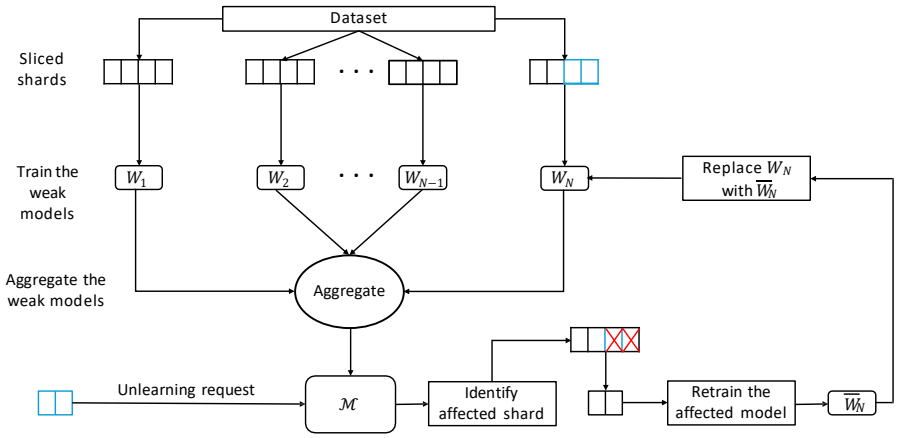


Figure 1.1.1: SISA method: the dataset is sharded and sliced for isolated training of weak models. Upon an unlearning request, only the affected model is retrained and replaced

As shown in Fig. 1.1.1, the dataset is partitioned into smaller, disjoint subsets called *shards*. After that, these shards are used to train models separately from each other so that the effect of each data point is isolated within a single model. When an unlearning request arrives, only the model trained on the shard containing the requested data is retrained. Since the shards are smaller in size, the retraining time of the affected model is lower compared to retraining the entire model as a whole. To further reduce unlearning time, each shard is then divided into data *slices*, which are then used for incremental training. The idea behind slicing the shards involves saving the model parameters after introducing a new slice to the model. This allows the retraining process to start from the last known model parameter before the requested data was introduced. This slicing strategy reduces the shard retraining time at the expense of additional storage. At the end, every model that is trained on the shards is aggregated together to give the final output. This algorithm offers an effective removal technique, but it lacks robustness. Imagine the scenario where large portions of the data to be deleted are located in a single shard. Removing such a significant amount of data could result in a weakened model with reduced learning capacity. Additionally, slicing the shards and incorporating incremental learning may

lead to catastrophic forgetting [14], where models lose previously learned information, degrading performance over time.

Aldaghri *et al.* [15] extended partitioning strategies with coded unlearning, combining dataset sharding with linear coding. In this method, the authors first partition the dataset into s small disjoint *uncoded shards* and encode them using a random binary coding matrix $G \in \{0,1\}^{s\times r}$. This coding matrix linearly combines the uncoded shards into r *coded shards* as follows:

$$\widetilde{D}_j = \sum_{i=1}^{s} g_{ij} D_i, \qquad j = 1, \ldots, r, \tag{1.1}$$

where the D_i denotes the i-th uncoded shard, $\widetilde{D}_j$ represents the j-th coded shard, and g_{ij} are elements of the G.

From the Eq. (1.1) it can be seen that each coded shard is a linear combination of the original uncoded shards. Specifically, each coded shard $\widetilde{D}_j$ is constructed by summing a subset of the uncoded shards D_i, weighted by the binary coefficients g_{ij} from the coding matrix G. These coded shards are then used to train independent weak learners, and the resulting weakly learned models are aggregated into a final model. Since the coded shards are the linear combinations of the original data points, each coded sample contains multiple uncoded data points. Therefore, when an unlearning request for data x_{un} arrives, only the coded shards that have the removal data are updated. To update the coded shard, first the uncoded shard D_i is identified, and using the random binary coding matrix G, the relevant data from the coded shards are located. Given that the coded shards are linear combinations of the original data points, subtracting the removal data effectively eliminates its contribution.

$$\widetilde{D}_j^{new} = \widetilde{D}_j - x_{un}, \quad \forall g_{ij} = 1. \tag{1.2}$$

While this approach reduces retraining costs, it introduces a fundamental trade-off governed by the *coding rate*, which is defined as $\tau = \frac{s}{r}$. The higher the coding rate, the lower the computational overhead, but also the lower the accuracy. In contrast, a lower coding rate improves accuracy, though it requires more computation.

While the unlearning methods discussed above are fast and efficient, their application is limited to models that are trained incrementally or to ensemble models. This incompatibility becomes critical for decision tree-based models. These models fundamentally differ in how they learn. In contrast to models that learn incrementally, decision trees use patterns drawn from all data points simultaneously rather than updating the model's structure step by step. As a result, eliminating even a single data point can invalidate splits across the tree, requiring structural adjustments. To address this, specific unlearning techniques for decision trees, such as *subtree retraining*, are required. This approach retrains only the subtrees affected by removal data, striking a balance between efficiency and the necessity to maintain the tree's integrity.

One of the very first decision tree-based unlearning methods was implemented by Schelter *et al.* [16]. The authors proposed an unlearning method for Extremely Randomized Trees (ERT), where the tree-splitting features are chosen randomly. Despite this randomness in selecting split features, the predictive performance of ERT is high enough to be comparable to that of models like Random Forests [17]. To unlearn, the method constructs *robust decision* nodes. A decision node is considered robust if splits remain stable even after simulating the removal of up to $\epsilon|D|$ training samples, where ϵ is the *unlearning budget*. This budget specifies the maximum number of data points that may be deleted from the dataset D. If the decision node is non-robust for data removal, then several alternative nodes will be pre-computed and saved together with the model. To balance robustness and efficiency, the algorithm limits the search for robust splits to B trials per node. If no robust split is found after B attempts, the node is marked as non-robust, and alternative nodes are generated. When a data removal request arrives, robust nodes only require updates to dataset information (such as decrementing class frequency counts) rather than split criterion revisions, since these splits remain valid even after removing up to $\epsilon|D|$ samples. In contrast, for non-robust nodes, the algorithm re-evaluates all pre-computed nodes by recalculating their split score as if the deleted sample was never present. If one of the nodes yields a higher split score than the original node, then the original node will be replaced with the node that has the higher split score. This ensures the tree effectively

forgets the deleted data by using a split that matches what the model would have learned if that data was never included in the training. Despite its efficiency, this approach has several limitations. In case of high ϵ, the number of non-robust nodes increases, as splits must tolerate more removals. Similarly, when the parameter B is set too small, the fewer robust splits are identified, again increasing the proportion of non-robust nodes. These conditions could result in a significant computational and memory overhead. This is primarily because the model contains a large number of non-robust nodes. Moreover, the method's reliance on the structure of Extremely Randomized Trees limits its applicability to other machine learning models, hence constraining its generalizability.

A similar approach can be adapted to traditional supervised decision tree-based models for unlearning, as demonstrated by Brophy and Lowd in the Data Removal Enabled-RF (DaRE-RF) method [18]. In this approach, the authors achieved the unlearning by updating the nodes directly impacted by data removal, as shown in Fig. 1.1.2.

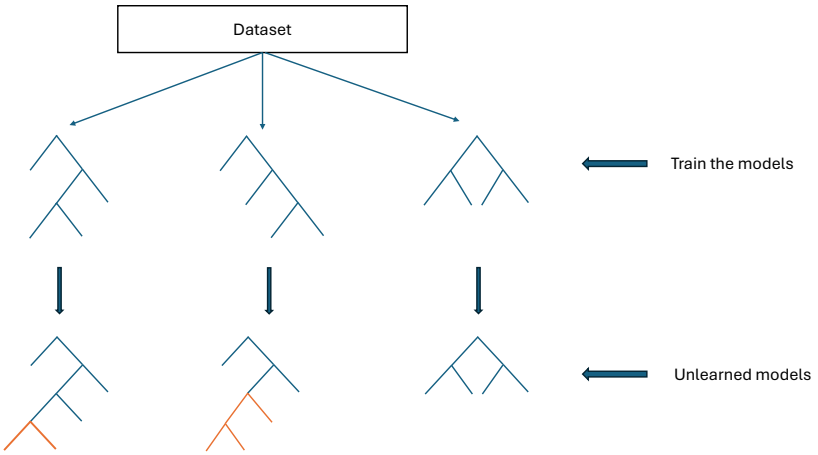


Figure 1.1.2: Overview of the DaRE-RF unlearning process. The dataset is used to train a random forest, after which individual data removal requests trigger selective updates. Orange lines indicate the nodes and subtrees that are retrained during the unlearning process, while unaffected parts of the model remain unchanged to preserve efficiency.

To unlearn the model, the split scores only for affected models are re-evaluated.

If the recalculated split score of the node remains unchanged or the new split score is lower than the current one, only the node's dataset information is updated. This is because the current split remains the optimal partition for the updated data, as no recalculated splits outperform it. If the new score exceeds the original score, then the subtree rooted at that node is retrained. Retraining ensures the model is identical to one built from scratch on the updated data. To further optimize the method, the authors introduced *random node layers* at the top of the tree. Random node layers use random split thresholds to partition data early, thereby reducing the subtree's dataset size so that the overall retraining time is decreased. Additionally, random nodes are considered robust, as they require retraining only if one of the child nodes is emptied during unlearning. To boost performance even more, the authors implemented *random threshold sampling*, which limits the number of candidate splits evaluated during training and unlearning. However, these optimizations come at a cost. Randomly sampled splits are considered suboptimal compared to data-driven splits, as they may overlook the optimal splits and thus degrade predictive performance.

Lin *et al.* [19] extended the subtree retraining strategy to Gradient Boosted Decision Trees (GBDT). As in the previously described unlearning methods, the algorithm evaluates split gains (scores) to determine whether subtrees require retraining. In GBDT, derivatives are used to compute split gains, which determine how nodes partition data. The first-order derivative Δ_i and second-order derivative ω_i for the data point i are calculated as:

$$\Delta_i = r_{i,k} - p_{i,k}, \quad \omega_i = p_{i,k}(1 - p_{i,k}), \tag{1.3}$$

where $r_{i,k}$ is the true label of the data instance i (1 if instance i belongs to class k, 0 otherwise), and $p_{i,k}$ is the probability of instance i belonging to class k.

These derivatives are used to calculate the split gains defined as:

$$\text{Gain}(s) = \frac{\left[\sum_{i=1}^{t}(r_{i,k} - p_{i,k})\right]^2}{\sum_{i=1}^{t} p_{i,k}(1 - p_{i,k})} + \frac{\left[\sum_{i=t+1}^{N}(r_{i,k} - p_{i,k})\right]^2}{\sum_{i=t+1}^{N} p_{i,k}(1 - p_{i,k})} - \frac{\left[\sum_{i=1}^{N}(r_{i,k} - p_{i,k})\right]^2}{\sum_{i=1}^{N} p_{i,k}(1 - p_{i,k})}, \tag{1.4}$$

where the t denotes the index of a candidate split threshold, sampled from the sorted dataset such that $1 < t < |D|$, and $|D|$ is the total number of data instances considered for the split.

To unlearn the model, as in the previous tree unlearning methods, the split gain (score) of the impacted nodes is re-evaluated. If the optimal split score of the node changes, indicating that the removed data invalidates the current split, the affected subtree is retrained. However, if the split threshold does not change, the algorithm propagates the removal request to affected child nodes for further evaluation. To improve unlearning efficiency, the authors introduced an optimization method called *lazy updates*, where derivatives are updated lazily—using approximated derivative values rather than recalculating at every iteration, reducing the number of updates required. Since split gains depend on the sums of the derivatives, as can be seen from Eq. (1.4), during the training part, sum of first and second derivatives are stored together with the model:

$$S_\Delta = \sum_{i \in D} \Delta_i \ , \quad S_\omega = \sum_{i \in D} \omega_i, \tag{1.5}$$

where S_Δ is the sum of the first order derivatives and S_ω is the sum of the second order derivatives.

When the unlearning request arrives, only the sums of the derivatives of the requested data are calculated as follows:

$$\delta_\Delta = \sum_{i \in D_{un}} \Delta_i, \quad \delta_\omega = \sum_{i \in D_{un}} \omega_i, \tag{1.6}$$

where the δ_Δ denotes the total first-order derivatives of the data samples in the unlearning set D_{un} and δ_ω represents the total second-order derivatives of those same samples.

The approximated sums of the derivatives are then calculated as follows:

$$\bar{S}_\Delta = S_\Delta - \delta_\Delta, \quad \bar{S}_\omega = S_\omega - \delta_\omega, \tag{1.7}$$

where $\bar{S}_\Delta$ $\bar{S}_\omega$ are the lazily updated sums of the first- and second-order derivatives, respectively.

These approximated sums of the derivatives are then used to calculate the split gains. However, delaying the derivative updates could harm the predictive performance of the model. This is because it may overlook the optimal split, as only the approximated values of the derivatives are used. As more data is removed over time, these approximations may significantly differ from the correct derivative sums. In order to preserve model accuracy, the algorithm updates the sums every λ iterations, where λ controls the frequency of these corrective adjustments.

While these methods address unlearning in supervised decision trees, they fall short in semi-supervised settings, where models use both labeled and unlabeled data. Semi-supervised learning is particularly valuable because labeled data is often costly to obtain. In contrast, unlabeled data is abundant and accessible, enabling cost-effective model training [20]. However, applying semi-supervised techniques to decision trees remains challenging. In a supervised setting, decision trees rely heavily on hierarchical splits derived only from labeled data. In contrast, semi-supervised trees incorporate patterns inferred from unlabeled data to determine the optimal split. Methods such as self-training and pseudo-labeling [21] attempt to bridge this gap, but they blur the distinction between labeled and unlabeled data contributions during training. Consequently, because of overlap between labeled and unlabeled data, unlearning in semi-supervised decision trees becomes challenging. Unlearning methods must remove specific labeled samples and also remove the indirect influence of unlabeled data, which is harder without clear boundaries for how unlabeled data was incorporated.

This thesis aims to address these challenges by exploring new unlearning techniques suited for semi-supervised decision trees. The key innovation lies in developing an unlearning algorithm capable of efficiently removing the influence of both labeled and unlabeled data, which, to the best of our knowledge, has not been implemented previously.

Chapter 2

Implementation and Evaluation of Subtree Retraining Method for Supervised Decision Tree Models

This chapter presents a comprehensive implementation and analysis of the subtree retraining method for supervised decision tree-based models, which is used in algorithms like HEDGE-CUT, DaRE-RF, and an unlearning method for GBDT. The chapter begins by reviewing the structure and principles of supervised decision trees, focusing on impurity metrics. The chapter then explores structural modifications such as random node layers and threshold sampling, evaluating their impact on training time, unlearning efficiency, and predictive performance. After that, the subtree retraining method is implemented and tested for single and subset data removal scenarios, with extensive experimental results illustrating trade-offs between computational efficiency and model accuracy that provided by the modifications. Finally, the limitations of these approaches are discussed, providing the foundation for developing an unlearning method for semi-supervised decision tree models.

2.1 Supervised Decision Trees Structure

Before implementing the unlearning method for supervised decision trees, let us briefly review the decision tree models. Decision tree models are the machine learning models that recursively partition the dataset into subsets based on the different values (thresholds) of the different features until the stopping criteria are met [22]. From the definition of the decision tree, it is clear that the model has a hierarchical structure. As shown in Fig. 2.1.1, each decision tree consists of a root node, decision or internal nodes, and leaf nodes. Each non-leaf node contains the condition that partitions the dataset. In order to find the best split condition, each possible condition is evaluated and scored. The metric used to evaluate these conditions can be either entropy or the Gini score, both of which are based on the impurity of the node.

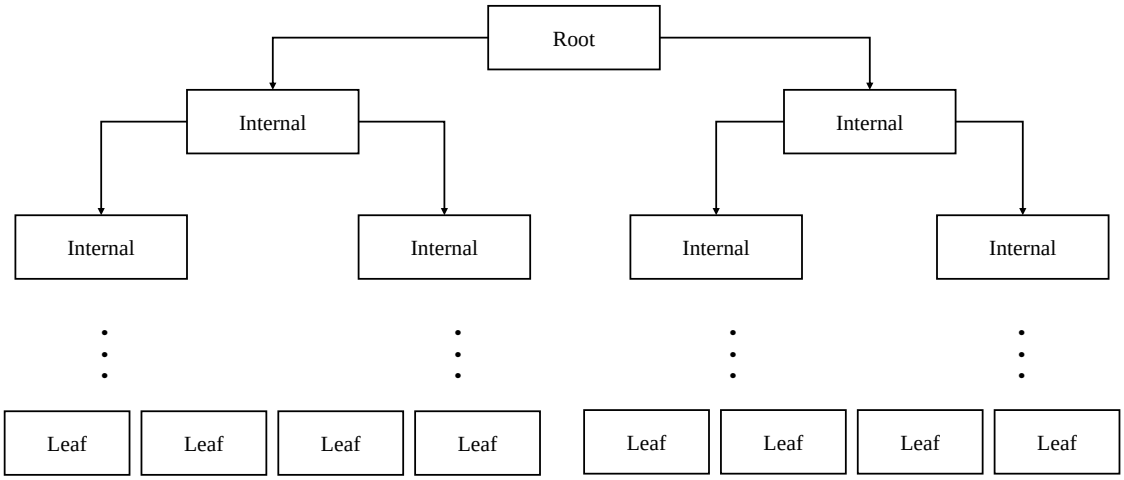


Figure 2.1.1: Structure of a decision tree model. It consists of a root node, multiple internal (decision) nodes, and leaf nodes that represent class labels.

After recursively partitioning the dataset into subsets based on split conditions and meeting the stopping criteria, the leaf nodes are reached. Each leaf node corresponds to a specific class label, indicating the outcome for the samples that fall into that node.

Impurity Metrics for Node Splitting

As discussed earlier, splits are evaluated using impurity metrics to quantify the quality of partitions. Here, *impurity*— refers to the term that is used to describe the split conditions that show how much the partitioned dataset is impure with respect to target variables. A partitioned subset is considered homogeneous (pure) if all samples belong to the same class; otherwise, heterogeneous (impure). One of the common measures of impurity is the Gini score. If we denote the impurity measure of a node with dataset D as $i(D)$, then the impurity calculated via the Gini score is defined as:

$$i(D) = \sum_{k=0}^{C} p_k(1 - p_k), \tag{2.1}$$

where C is the number of classes and p_k denotes the probability of picking a data point with class k. A lower Gini score indicates that the node is more homogeneous or less mixed with other classes.

Entropy is another metric that is used to evaluate the impurity of the node. It measures the level of uncertainties of the partitioned dataset. High entropy means that the split is partitioning the dataset into mixed classes, while a low entropy value signifies homogeneity. Entropy of the node is defined as:

$$i(D) = -\sum_{k=0}^{C} p_k \log_2(p_k), \tag{2.2}$$

To find the best split for the node, a threshold must be chosen to partition the dataset such that it minimizes the node's impurity while maximizing the impurity reduction between the parent node and its child nodes. Assuming that each split either lowers the impurity metrics or keeps them the same as in parent node, the splits should maximize the *impurity drop* [23].

Formally, for a split $s \in S$ where S is the set of all possible splits that divides the dataset D into two subsets D_l (left child node) and D_r (right child node), the impurity drop Δ_s is then defined as follows:

$$\Delta_s = i(D) - \frac{|D_l|}{|D|} i(D_l) - \frac{|D_r|}{|D|} i(D_r), \tag{2.3}$$

where the $|D_l|$ and $|D_r|$ are the numbers of data samples in the left and right nodes.

Therefore, to find the best split s^* that partitions the dataset into less impure subsets, the impurity drop should be maximized:

$$s^* = \underset{s \in S}{\operatorname{argmax}} \Delta_s \tag{2.4}$$

2.2 Optimizing Training and Unlearning: Modifications to Decision Trees

While impurity and entropy metrics guide the evaluation of splits, structural modifications are still necessary to optimize the algorithm. Examples of such modifications include random node layers and threshold sampling. These modifications offer a trade-off: by reducing exhaustive search processes, they accelerate training and unlearning processes, but they also risk overlooking the optimal splits that may degrade predictive accuracy. In this section, the modifications represented by random node layers and threshold sampling that alter supervised decision tree structure will be discussed, and their impact on model performance, training time, and unlearning efficiency will be evaluated.

Random Node Layers

Random node layers are decision nodes located at the top of the tree, where split conditions are chosen randomly. Introducing random node layers can improve both training and unlearning efficiency. In a traditional decision tree, evaluating all possible split thresholds for a node at depth d_i takes $|F|\frac{|D|}{2^{d_i}}$ impurity computations, where $|F|$ is the number of features and $|D|$ is the number of data samples. If we assume that the tree is balanced, early partitions caused by the random nodes should reduce the optimal split computation to $|F|\frac{|D|}{2^{d_{rn}+d_i}}$, where d_{rn} is the number of random node

layers [19].

For example, multiple decision trees were trained on a synthetic dataset of 1,000,000 samples with 10 features. One of the models was trained without random node layers (blue bar), while the others were trained with $d_{rn} = \{x \mid 1 \leq x \leq 6\}$. The training times of all models are presented in Fig. 2.2.1. As shown in the Fig. 2.2.1, the decision trees with random node layers train significantly faster than the model without random nodes.

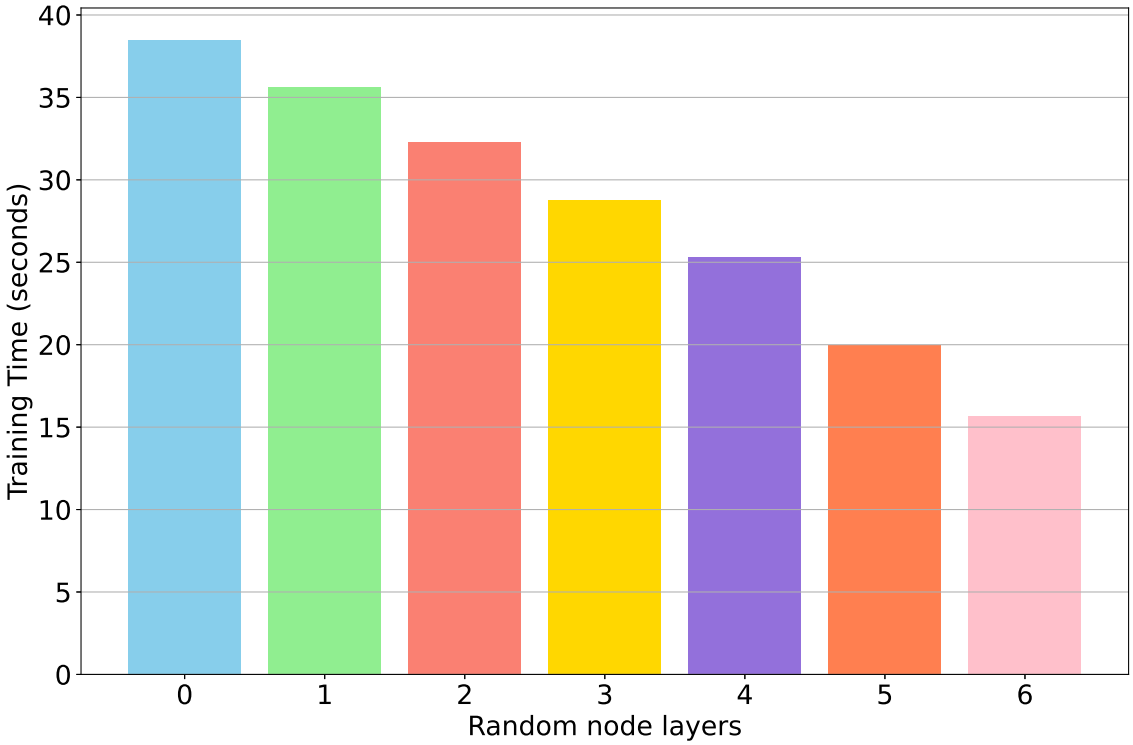


Figure 2.2.1: Training time of the decision tree without random node layers (blue) and with varying numbers of random node layers

However, as illustrated in Fig. 2.2.2, adding random node layers degrades the model's prediction performance. Randomly selected splits are not always optimal, leading to less effective partitions and, consequently, lower accuracy. Despite this trade-off between accuracy and training time, random node layers look appealing for scenarios where time efficiency is prioritized over insignificant accuracy losses.

Beyond training efficiency, the random nodes also accelerate the unlearning process, as they are comparatively more robust than standard decision nodes. This is

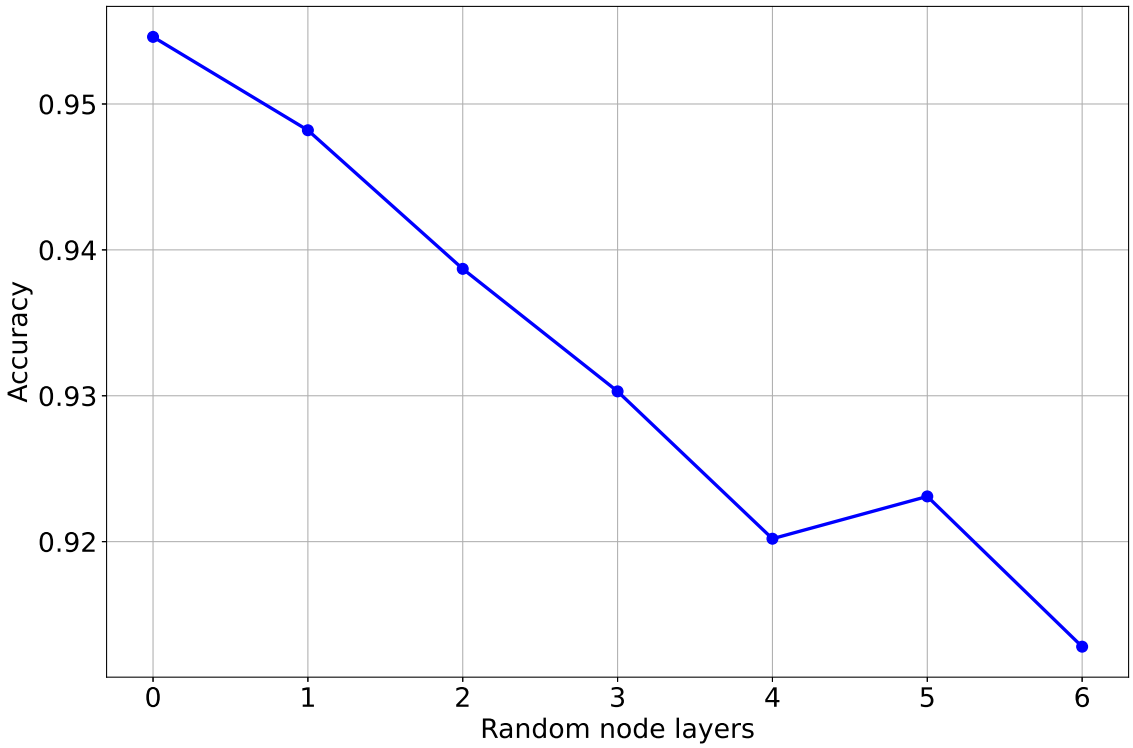


Figure 2.2.2: Accuracy of the model with different random node layers

because random nodes are retrained only when one of their child nodes becomes empty, which means they are rarely updated. A detailed analysis of the effect of random node layers on unlearning performance is presented in Section 2.3.

Threshold Sampling

The second modification applied to supervised decision trees is the threshold sampling method. This modification involves selecting and evaluating a subset of potential thresholds per feature instead of exhaustively evaluating all possible values. The main idea behind this sampling is to reduce training and unlearning time by limiting the search space for optimal splits. Finding the optimal split among a smaller subset is faster than searching through the entire set of potential thresholds. However, similar to random node layers, threshold sampling introduces a trade-off between accuracy and time efficiency. The larger the size of the sampled threshold, the higher the prediction performance tends to be, but also the higher the training time.

To quantify this trade-off, several models using the threshold sampling method were trained on the 1,000,000 samples with 10 features, and results are shown in Figs. 2.2.3 and 2.2.4. As illustrated in Fig. 2.2.3 as the size of the sampled thresholds increases, the model becomes more robust. The figure demonstrates that beyond 60 sampled thresholds, the prediction performance stabilizes around 95.33%.

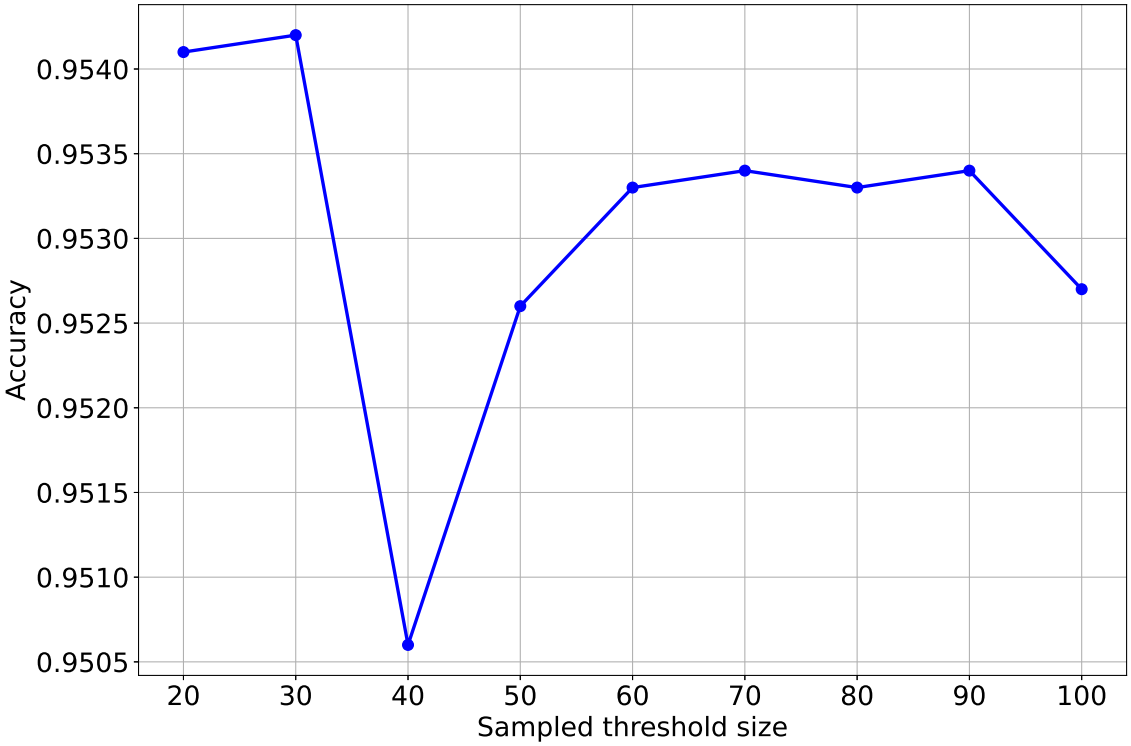


Figure 2.2.3: Accuracy of the model with different size of sampled threshold

In contrast, Fig. 2.2.4 reveals an increase in training time with larger threshold subsets. Together, these results validate the previously mentioned statement, highlighting the trade-off between training time and the size of sampled thresholds.

Despite these benefits, threshold sampling has several limitations. Sometimes sampling a subset of thresholds from the feature set might overlook the best split threshold, consequently decreasing the predictive performance of the model. Additionally, the randomness of the threshold sampling method introduces variability between different training runs, since the sampled set of thresholds may differ each run. This variability makes it difficult to reproduce the result, as repeated experiments may result in different outcomes. Furthermore, for small datasets, sampling

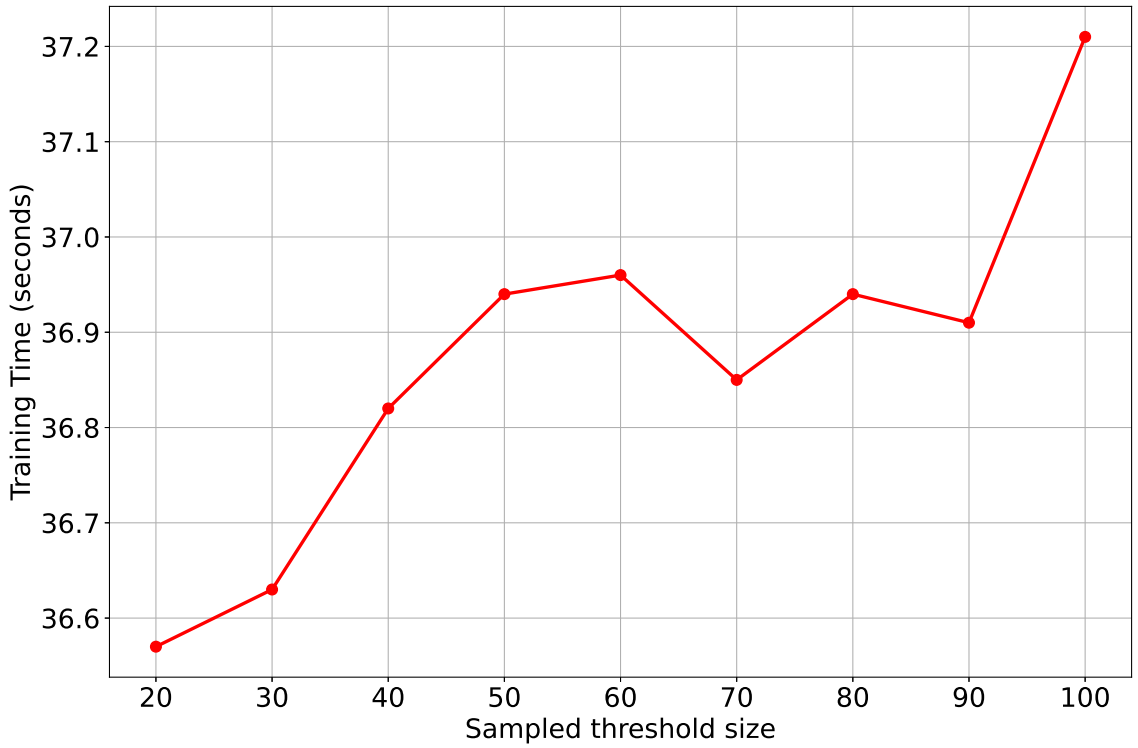


Figure 2.2.4: Training time of the model with different sampled thresholds

the thresholds could significantly degrade the overall prediction performance while giving a small boost in training time.

2.3 Implementation of Subtree Retraining Method

When a removal request arrives, machine unlearning aims to update the model such that it behaves as if the removed data were never part of the training process. For decision tree-based models, this is achieved through the subtree retraining approach, a method commonly used in supervised decision tree unlearning as in HEDGECUT and DaRE-RF. This method minimizes computational cost by finding and updating the subtrees of the tree directly influenced by the removal data.

The subtree retraining approach could be described by the following three steps:

1. Identify the affected nodes. Identify the path from the root node to the leaf node where the removal data were classified. This step is crucial because it

helps in isolating the specific nodes of the tree that may require retraining.

2. Update the node statistics. For each node along this path, update the statistical information about the class distribution (class counts) and re-evaluate the split conditions to ensure they remain optimal.

3. Selective retraining. If the split condition is no longer optimal, the subtree growing from that node is retrained, and the process terminates. If the split remains optimal, retain it and proceed to the next node in the path.

This method relies on cached statistics at each node, offering efficient updates of nodes without retraining the entire tree. Retraining solely the affected nodes offers several advantages:

- Reduced computational complexity. Method updates only affected nodes, not the entire tree.

- Consistency. By selective retraining, the method preserves unaffected regions of the tree.

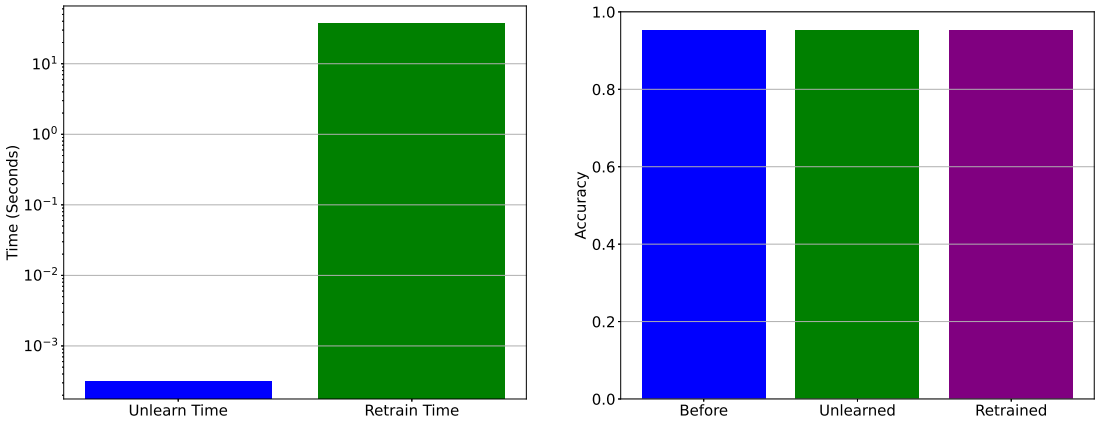
An example of the subtree retraining approach applied in practice is shown in Fig. 2.3.1. The figure compares the computational time and prediction performance of the unlearning and retraining processes implemented in a decision tree model trained without any modification (without random nodes and random sampled thresholds) on 1,000,000 data samples with 10 features.

Fig. 2.3.1(a) shows the comparison between the time required to unlearn a single data point using subtree retraining and the time needed to retrain the entire model from scratch. The results demonstrate that unlearning the single data point is computationally more efficient than retraining the entire model, as retraining took 38 seconds while the unlearning took 0.002 seconds.

Turning to prediction performance, Fig. 2.3.1(b) provides information about the prediction performance of the models at different stages: before the data removal (blue), after unlearning the data using the subtree retraining method (green), and

after retraining the entire model. The results indicate that subtree retraining achieves similar accuracy to full retraining in a single data removal case.

The results confirm that efficiency gains through efficient unlearning methods do not compromise performance in a single data removal case.



(a) (b)

Figure 2.3.1: Single data removal using subtree unlearning method and retrain method. (a) Unlearning and Retraining time of data removal. (b) Accuracy of the model: before (blue), after the unlearning (green) and after the retraining (purple)

A similar experiment was applied to modified decision tree models with varying numbers of sampled thresholds and random node layers, as shown in Fig. 2.3.2, using 1,000,000 data samples with 10 features. The modified decision trees behave in the same way as the unmodified supervised model. Fig. 2.3.2 is divided into four subplots, where the top two plots provide information about how the sampled threshold affects unlearning and retraining, while the bottom two plots provide information about how random node layers affect the unlearning and retraining processes.

Subfigures 2.3.2(a) and 2.3.2(c), clearly show that data removal through full retraining (red) is computationally expensive compared to the unlearning through subtree retraining (blue) method. The results indicate that the subtree retraining method provides an efficient approach to unlearning, maintaining low unlearning time across all sampled thresholds and random node layer configurations.

Additionally, subfigures 2.3.2(b) and 2.3.2(d) show the comparison of prediction performance of the subtree retraining and full retraining methods. The unlearned

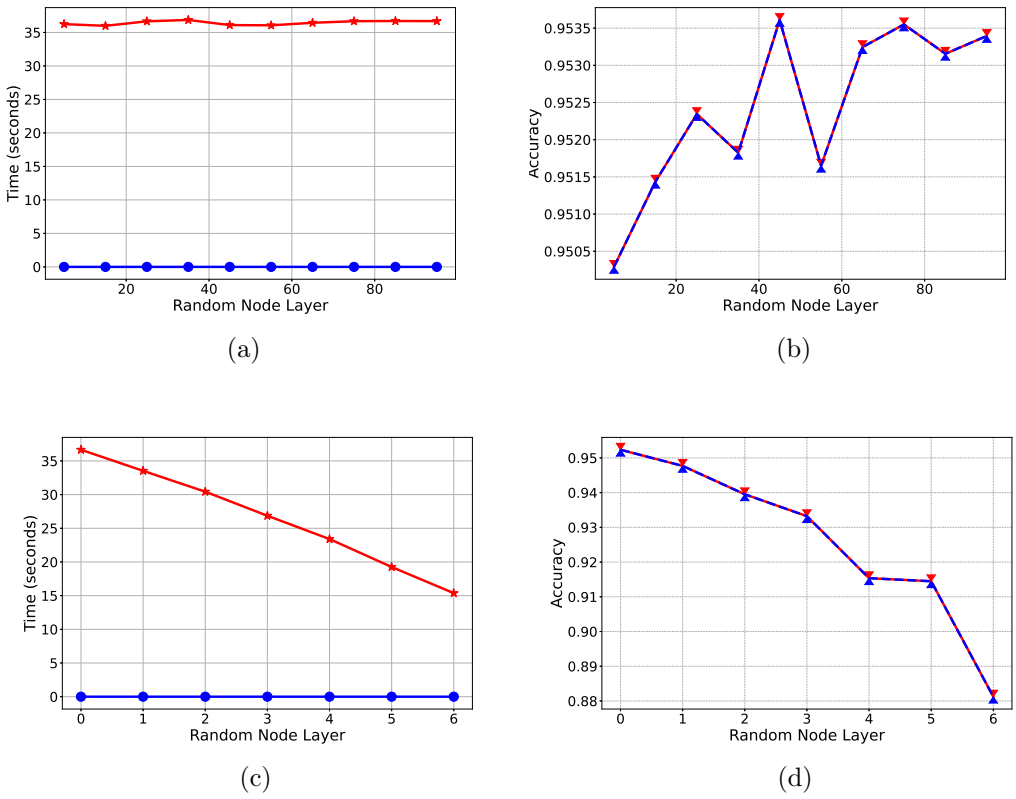
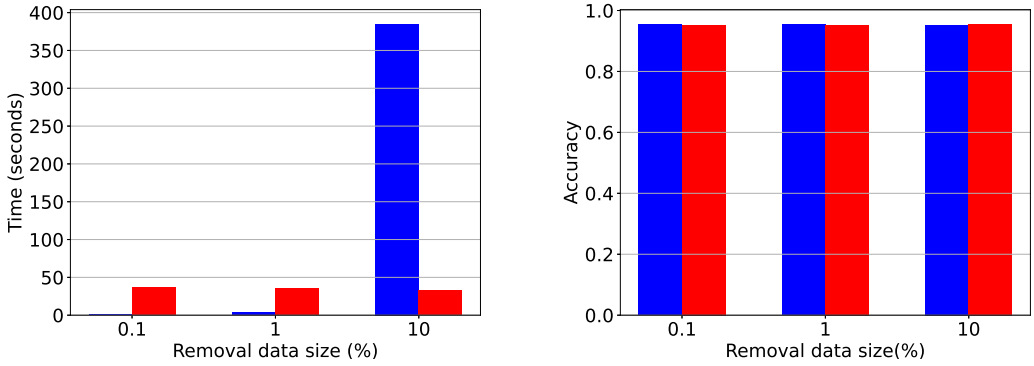


Figure 2.3.2: Effect of the sampled threshold size and random node layers (unlearning: blue; retraining: red). (a) Time vs. random node layers. (b) Accuracy vs. random node layers (c) Time vs. sampled threshold size. (d) Accuracy vs. sampled threshold size.

model's accuracy remains stable and matches the retrained model's performance, indicating that the efficiency gains of subtree retraining do not come at the cost of prediction accuracy.

Even though subtree retraining is an efficient method for unlearning a single data point, removing large portions of the training dataset can become computationally heavy. At a certain point, data removal size could become large that retraining the entire model will be computationally efficient rather than unlearning each data point individually. This statement is supported by Fig. 2.3.3, which illustrates the impact of removing a subset in terms of unlearning time and prediction performance. The experiments are conducted using a dataset of 1,000,000 samples with 10 features.

Subfigure 2.3.3(a) shows the time taken to unlearn (blue) and retrain (red) the



(a) (b)

Figure 2.3.3: Time taken by unlearned and retrained models to remove a subset of the training data (a), and their prediction performance after the removal (b).

model to remove the training subsets of different sizes. As it can be observed, as the subset size increases, the unlearning time grows exponentially, while the retraining time decreases. The result indicates that the subtree retraining method performs poorly in terms of unlearning the big portions of data. Despite the problems with the computation time, the overall performance of the model is still comparable with the retrained model as shown in the subfigure 2.3.3(b). This evidence indicates that while subtree retraining may be inefficient for large-scale deletions, it does not significantly degrade model accuracy. Collectively, these results demonstrate that while subtree retraining is well-suited for small-scale unlearning, large-scale data removal scenarios should be approached carefully by finding the optimal boundary between unlearning and retraining, after which retraining the model is cheaper and more efficient than unlearning the model.

2.4 Limitations of the Previously Implemented Algorithms

While previously implemented unlearning approaches have demonstrated effectiveness in single data removal scenarios, there are some limitations that impact their efficiency and scalability.

One of the key limitations lies in the use of threshold sampling to boost both training and unlearning processes. As it can be seen in Fig. 2.3.2, while threshold sampling does provide some improvements in computational efficiency, the performance gain is relatively small when compared to the enhancements achieved through the random node layers. Moreover, the use of threshold sampling leads to the generation of suboptimal split conditions instead of optimal ones. As a result, it leads to a reduction of the prediction performance of the model.

Another limitation of the previously provided methods is that the large-scale unlearning scenarios are not handled well, as can be seen from Fig. 2.3.3. Provided methods mainly focus on the single data removal, which is ineffective when it comes to unlearning a large portion of the dataset. As a result, retraining the entire model provides better computational efficiency. This gap in unlearning techniques underscores the necessity of developing a more scalable and adaptive approach for handling subset unlearning.

Understanding these limitations is key for designing the unlearning method for the semi-supervised tree models. The proposed unlearning algorithm aims to cover all the strengths of subtree retraining while extending the method for the semi-supervised trees.

Chapter 3

Implementing the Unlearning in Semi-Supervised Decision Tree Models

The chapter starts by providing a comprehensive explanation of the semi-supervised decision tree, detailing the novel impurity metric employed to balance contributions from labeled and unlabeled data. It further provides information about algorithmic modifications essential to enable efficient unlearning, specifically highlighting enhancements to node structures and the integration of random node layers. The proposed semi-supervised decision tree is then expanded into a semi-supervised random forest model. Subsequently, the chapter presents an effective unlearning method tailored for semi-supervised trees and forests, reducing computational overhead by selectively retraining impacted subtrees. The chapter concludes with an analysis of the algorithm's computational complexity, providing insights into its performance under various practical conditions.

3.1 Semi-supervised Decision Tree Structure

To construct a semi-supervised decision tree with explicit separation between labeled and unlabeled data, Levatić *et al.* [24] introduced a novel impurity metric. This

metric enables precise tracking of individual data contributions and integrates both labeled and unlabeled data for split evaluation. The impurity calculation is defined as:

$$i_{SSL} = \omega \times i_l(D_l, Y) + \frac{1-\omega}{|F|} \times \sum_{i=1}^{|F|} i_{ul}(D, f_i), \tag{3.1}$$

where D is the entire dataset and D_l refers to the labeled portion of the dataset, Y is the target classes, $|F|$ is the number of features, f_i is the i^{th} feature, and ω is a weight parameter controlling the contribution of labeled and unlabeled data.

As shown in Eq. (3.1), the impurity calculation for a semi-supervised decision tree consists of two parts. The first part, $i_l(D_l, Y)$ is calculated using only the labeled portion of the dataset. The second part, $i_{ul}(D, f_i)$ is computed using only the unlabeled portion of the dataset. A weight parameter ω controls the relative contribution of these components, enabling control over the learning mode. When ω is set to 1, the model is fully supervised, which means only labeled data will contribute to the splits. When ω is set to 0, the model's impurity calculations will be based on the unlabeled data only. The model functions in a semi-supervised learning mode for values between 0 and 1.

The impurity score of a node using only labeled data is computed based on the Gini score, with some modifications. Formally, the impurity of the node based on the labeled data is defined as:

$$i_l(D_l, Y) = \frac{i(D_l, Y)}{i(D_l^{train}, Y)}, \tag{3.2}$$

where the $i(D, Y)$ and $i(D_l^{train}, Y)$ are calculated same as in Eq. (2.1). D_l^{train} denotes the entire available labeled training dataset.

To calculate the impurity of the node based on the unlabeled data, the following approach will be used:

$$i_{ul}(D, X_i) = \begin{cases} \frac{i(D,f_i)}{i(D^{train},f_i)}, & \text{if } f_i \text{ is nominal,} \\ \frac{\sigma(D,f_i)}{\sigma(D^{train},f_i)}, & \text{if } f_i \text{ is continuous,} \end{cases} \tag{3.3}$$

where the $\sigma(D, f_i)$ is the variance calculated using the:

$$\sigma(D, f_i) = \frac{\sum_{j=1}^{|D|}(x_{f_i}^j)^2 - \frac{1}{|D|}(\sum_{j=1}^{|D|} x_{f_i}^j)^2}{|D|}, \tag{3.4}$$

where $|D|$ is the number of examples and x_i^j denotes the value of the feature f_i for the j-th example.

From Eq. (3.3), it can be observed that there are two methods for calculating the split score using unlabeled data. In the first case, when the unlabeled feature data consists of nominal values, the Gini score, as shown in Eq. (2.1), is used. In the second case, when the feature data is continuous, the variance of the feature is applied as shown in Eq. (3.4).

As can be seen from Eqs. (3.2) and (3.3), both contain denominator parts, which serves as a form of normalization. This ensures comparability between labeled and unlabeled contributions, preventing one component from dominating the impurity score due to scale differences.

3.2 Algorithm Modifications to Enable Unlearning

To unlearn this semi-supervised model, several modifications to the model are made. First, each decision node is changed so that it stores more than just the split threshold and feature. It also stores statistical information, such as the number of labeled and unlabeled examples. This additional information helps node re-evaluation during the unlearning process. Second, each leaf node is modified to store the number of labeled and unlabeled examples as well as the class assigned to the leaf node. This is done to enable accurate updates to class distributions following data removal. Additionally, random node layers are introduced at the top of the tree to improve the training and unlearning processes by randomly selecting the split features and thresholds. These random nodes help reduce the dataset size used for computing split scores and. As a result, it reduces overall computational overhead. Moreover, random node layers are generally considered robust. Random nodes require retraining only if one of the child

nodes becomes empty after the unlearning process.

Algorithm 1 Semi-Supervised Decision Tree Algorithm

Require: Labeled dataset D_l, Unlabeled dataset D_{ul}, Feature set F, Target variable Y, Number of random node layers d_{rn}, Maximum depth d_{max}, weight parameter ω

Ensure: Decision Tree T

Node Creation:
Each node stores the following information:
    - Labeled dataset used to create the node: D_l
    - Unlabeled dataset used to create the node: D_{ul}
    - If the node is a decision node:
        - Best split score achieved using f^* and t^*
    - Assigned class of the node if the node is a leaf
SSL Tree Construction:
if all instances in D_l belong to the same class c **then** ▷ $c \in Y$
    return a leaf node labeled with class c ▷ Stop if all labeled instances are of the same class
end if
if Current depth d equals d_{max} **then** ▷ Stop if maximum depth is reached
    return a leaf node labeled with the majority class in D_l
end if
if Current depth d is less than d_{rn} **then** ▷ Use random node layers for early splits
    Randomly select a splitting feature f and threshold t
else
    Select the best feature $f^* \in F$ and threshold t^* that maximize the split score ▷ Use Equation (3.1)
end if
Create a root node N_0 labeled with f^* and t^*
Partition $D_l \cup D_{ul}$ into subsets D_1, D_2 based on f^* and t^*
for each subset D_i **do**
    Recursively construct the subtree rooted at N_0 using D_i and F ▷ Build the tree recursively
end for
return the decision tree T rooted at N_0 ▷ Return the constructed tree

Algorithm 1 presents the training procedure for the semi-supervised decision tree. To fit the semi-supervised decision tree model, it is necessary to provide labeled D_l and unlabeled D_{ul} data, a feature set F, target classes Y, a weight parameter ω, and the maximum depth of the tree d_{max}. Specifying the depth of the tree is necessary to prevent overfitting.

The algorithm begins by constructing the root node N_0 at depth d_0. If the parameter d_{rn} (number of random node layers) is specified, the first d_{rn} layers of the tree are generated using randomly selected features and thresholds. After the d_{rn} layer, the algorithm picks the best feature and threshold using the Eq. (3.1). The algorithm then partitions the dataset into subsets based on the chosen feature and threshold. This process is repeated recursively for each subset, returning the decision tree that was made. After the root node is constructed, the algorithm at each node starts determining whether the dataset D_l contains only one class or whether the current depth equals the specified maximum. When either condition is met, the algorithm returns a leaf node that contains the labeled and unlabeled datasets that were used to create this node along with its assigned class label.

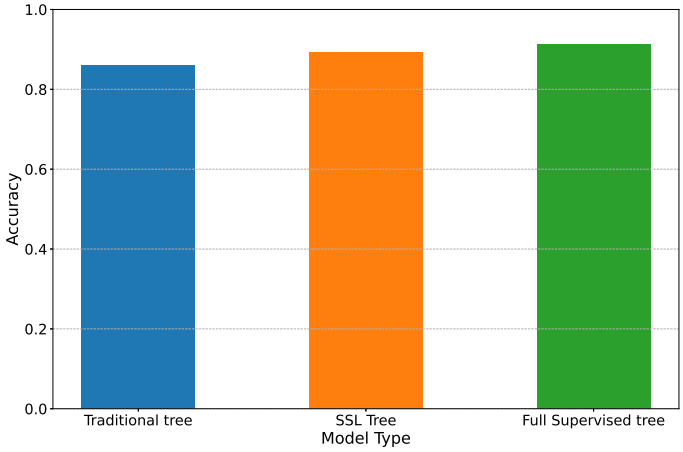


Figure 3.2.1: Supervised (blue) vs. Semi-supervised (orange) tree prediction performance using 100 labeled and 9,900 unlabeled data samples. The fully supervised model using the entire dataset (green) is set as a benchmark.

Fig. 3.2.1 shows the performance of the semi-supervised decision tree built using the Algorithm 1 compared to the traditional supervised tree. To construct the trees, a synthetic dataset containing 10,000 samples and 10 features was generated. To construct the semi-supervised tree, 100 labeled examples and 9,900 unlabeled examples were used, whereas the supervised decision tree model was trained using only the same 100 labeled samples as the semi-supervised decision tree. Additionally, a fully supervised tree trained on all 10,000 labeled samples was included as a benchmark to

demonstrate the maximum possible accuracy. From the figure, we can also observe that the semi-supervised tree outperforms the supervised tree, achieving an accuracy of 89% compared to 86% for the supervised tree. Meanwhile, the fully supervised decision tree trained on the entire dataset achieved an accuracy of 91%.

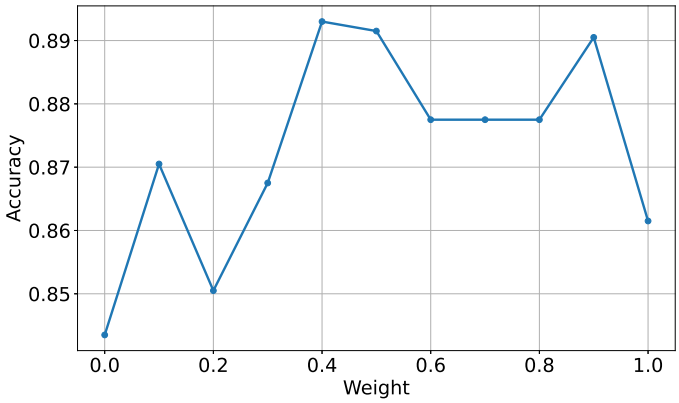


Figure 3.2.2: Effect of the weight parameter on the prediction performance of the semi- supervised model

As it was mentioned earlier, a weight parameter ω has a significant effect on the model's prediction performance. Fig. 3.2.2 shows the effect of the weight parameter on the prediction performance of the semi-supervised decision tree. From Fig. 3.2.2, it can be observed that when the weight parameter is between 0 and 1, the prediction accuracy is higher than when the splits depend solely on either the labeled or the unlabeled data.

3.3 Extending the Algorithm to Semi-Supervised Random Forest

The previously discussed semi-supervised decision tree algorithm can be extended to the semi-supervised random forest algorithm. A random forest [25] is an ensemble of multiple decision tree models trained using bootstrap samples drawn from the training dataset. *Bootstrapping* is a sampling method in machine learning that selects data points from the original dataset "with replacement". In other words, the same data point can be selected multiple times from the dataset. Bootstrapping trains each

tree in the random forest on a slightly different version of the data. This diversity caused by bootstrapping helps improve the overall predictive performance and makes the model robust. Additionally, in a random forest model, each tree considers only a subset of the feature set when determining splits at each node. The size of this subset can be calculated using various methods, such as the logarithmic approach [24], square root rule [25], and fixed fraction [26]. In this work, the logarithmic approach is adopted, defined as:

$$m = \lfloor \log_2(|F|) + 1 \rfloor \tag{3.5}$$

where the $|F|$ represents the number of features and m denotes the size of the selected feature subset.

Since the model is the ensemble model, the final prediction is obtained by aggregating the tree predictions from the forest. There are several ways to combine the predictions, like probability voting, weighted voting, and averaging predictions. However, this thesis employs the majority voting method. In this method, predictions from all trees are aggregated, and the class that appears most frequently is selected as the final output.

Algorithm 2 Semi-Supervised Random Forest Algorithm

Require: Labeled dataset D_l, Unlabeled dataset D_{ul}, Feature set F, Target variable Y, Number of trees T, Maximum depth d_{max} Number of features to consider at each split m, Number of random node layers d_{rn}

Ensure: Random Forest model RF

Initialize an empty forest $RF \leftarrow \emptyset$ ▷ Start with an empty forest
for $t = 1$ to T **do** ▷ Build T decision trees
    Sample bootstrap datasets D_{t_l} and $D_{t_{ul}}$ from $(D_l \cup D_{ul})$ with replacement
    Build a decision tree T_t using the Decision Tree Algorithm (Algorithm 1) with:
        - Training data: D_{t_l} and $D_{t_{ul}}$
        - Feature set: Randomly sample m features from F at each split ▷ Eq.(3.5)
        - Target variable: Y
        - Maximum depth: d_{max}
        - Number of random node layers: d_{rn}
    Add T_t to the forest: $RF \leftarrow RF \cup \{T_t\}$ ▷ Add the tree to the forest
end for
return the Random Forest model RF ▷ Return the final random forest model

The random forest fitting process is shown in the Algorithm 2. The algorithm

starts by initializing an empty forest that will store the trees. Next, T trees are constructed iteratively: for each tree $T_t \in T$, a bootstrap sample is drawn from the original dataset. At each split within a tree, a random subset of m features is selected from the feature set F. Each tree is then trained using Algorithm 1. Additionally, if the random node layer d_{rn} is specified, then the first d_{rn} levels of the tree employ randomly chosen features and thresholds rather than optimized splits. This process continues recursively until the tree reaches maximum depth, after which the tree is added to the forest. The algorithm repeats these procedures until the desired number of trees T has been generated, thereby completing the forest construction.

3.4 Unlearning Method For Semi-Supervised Decision Trees

Unlearning in semi-supervised decision trees builds on the subtree retraining method used in supervised settings. This approach ensures efficient data removal by retraining only the subtrees that directly affected by the unlearned data, significantly reducing computational cost compared to full model retraining. The unlearning process can be carried out in several steps:

1. Re-evaluate Split Scores: To unlearn a data point, start from the root of the tree and re-evaluate the split scores of the affected nodes using Eq. (3.1), excluding the data marked for removal.

2. Retrain if Necessary: If the new split score improves over the current split score, retrain the subtree affected by the deletion of the data.

3. Retain Otherwise: If the new split score is similar to or worse than the current split score, simply update the count of labeled examples with their corresponding labels (for labeled data unlearning) or the count of unlabeled examples (for unlabeled data unlearning).

Fig. 3.4.1 shows the workflow of the subtree unlearning approach for a depth-3 decision

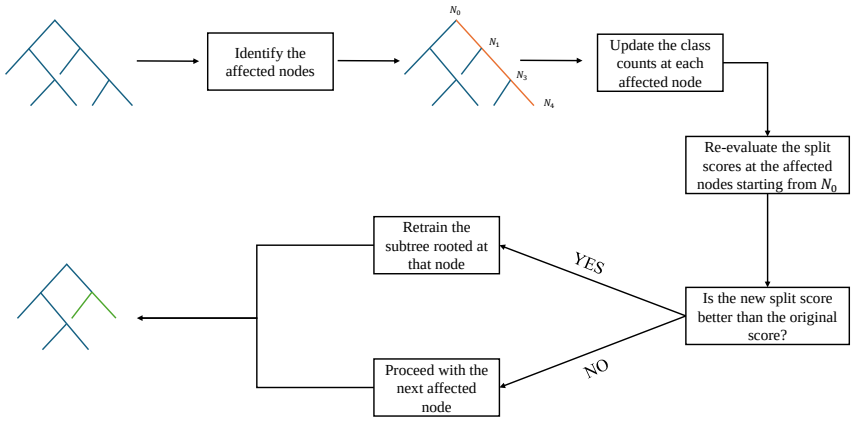


Figure 3.4.1: Workflow of the subtree unlearning approach for a depth-3 decision tree: (1) Identify nodes affected by removal (e.g., N_0 to N_4), (2) Update class counts and re-evaluate splits for affected nodes, (3) Retrain subtrees (e.g., at N_2) if splits improve.

tree. The unlearning process begins by identifying all nodes affected by data removal, as illustrated by nodes N_o through N_4. For each affected node, the method updates the class distribution information and re-evaluates split criteria while excluding the removed data points. A critical decision point occurs when comparing the new split scores against the original values; if improvements are found, the affected subtrees (such as the one rooted at N_2) undergo retraining, while nodes with unchanged or worse splits retain their original structure.

Overall, the unlearning procedure is shown in Algorithm 3. The provided algorithm removes the impact of the training instance (x, y) from the semi-supervised decision tree efficiently without retraining the entire tree.

The unlearning procedure begins by traversing the tree from the root node to the leaf node where the removal instance is classified. Each node encountered during the traversal is updated. First, from each decision node's dataset, exclude removal data. If the removal instance is labeled, remove the instance (x, y) from D_l and Y, otherwise, remove the instance (x) from D_{ul} if the instance is unlabeled. After that, the new split score at the node is calculated using the Eq. (3.1) excluding the removal data. The subtree rooted at the node is retrained only if the new split score improves

Algorithm 3 Efficient Semi-Supervised Decision Tree Update for Unlearning

Require: Decision Tree T, Training instance (x, y) to delete
Ensure: Updated Decision Tree $\widetilde{T}$

Re-Evaluate Split Scores:
Start from the root node N_0 of the tree T
for each decision node N_{d_i} on the path from N_0 to the leaf node L where (x, y) would be classified **do**
    Update statistics at N_{d_i} by excluding (x, y):
    if (x, y) is a labeled example **then**
        Remove (x, y) from D_l and update the count of labeled examples
    else
        Remove (x) from D_{ul} and update the count of unlabeled examples
    end if
    Recompute the split score at N_{d_i} using Equation (3.1) ▷ Exclude (x, y) from the calculation
    Retrain if Necessary:
    if The new split score is better than the current split score **then**
        Retrain the subtree rooted at N_{d_i}:
            Rebuild the subtree using the Decision Tree Algorithm (Algorithm 1)
    end if
    Update Otherwise:
    if The new split score is similar to or worse than the current split score **then**
        Simply update the counts of labeled or unlabeled examples at N_{d_i}
        Proceed to the affected child node of N_{d_i} ▷ Move to the next node in the path
    end if
end for
return the updated Decision Tree $\widetilde{T}$ ▷ Return the updated tree

partitioning (better split score). However, if the computed split score is worse than the previous split score, then only the statistics(class counts) of the tree are updated without any structural changes. This approach helps to minimize the unnecessary retraining while adapting the tree to the new dataset without (x, y).

To scale this method for subset removal (removal of $D_s \subseteq D$), recalculate split scores at affected nodes while excluding all instances in D_s. This ensures efficient propagation of large-scale removals without full-model retraining.

Additionally, this approach can be applied to unlearn random forest models. This is demonstrated in Algorithm 4, which outlines the unlearning process for a semi-supervised random forest model. As it can be seen from the unlearning procedure,

each tree in the forest is unlearned using Algorithm 3.

Algorithm 4 Efficient Semi-Supervised Random Forest Update for Large Subset Removal

Require: Random Forest RF, training data instance (x, y)
Ensure: Updated Random Forest RF
1: **for** For each tree T_t in the forest RF **do**
2: **Update tree** T_t **using the Algorithm 3**
3: **end for**
4: **return** the updated Random Forest RF ▷ Return the updated forest

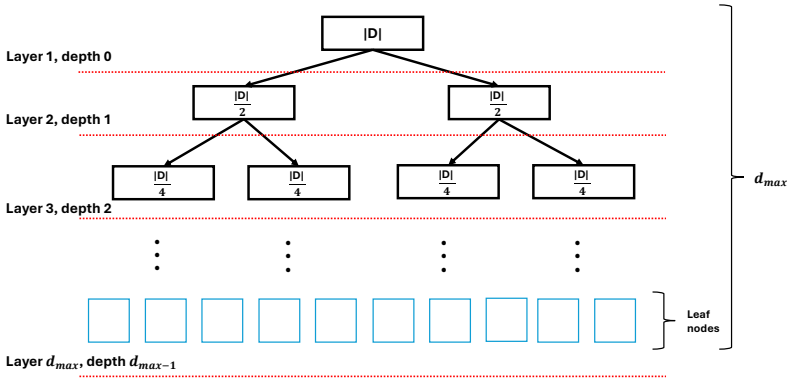
3.4.1 Time Complexity

Understanding the algorithm's time complexity is crucial for evaluating its efficiency, scalability, and robustness when applied to large datasets. In this section, the time complexity of the provided algorithm is analyzed for both tree construction and unlearning, covering best, average, and worst cases. The time required to construct and unlearn a decision tree depends on several factors, including the number of data points $|D|$, the number of features $|F|$, the maximum depth of the tree d_{max}, the number of random node layers d_{rn}, and the tree's structure—whether balanced (Fig. 3.4.2(a)) or imbalanced (Fig. 3.4.2(b)).

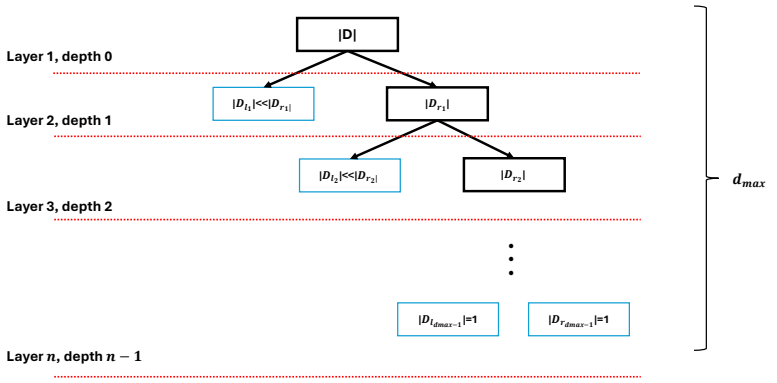
Tree construction time complexity. The time complexity of the training process for the proposed algorithm can vary based on whether the tree is balanced or imbalanced.

For the balanced case where each split partitions the dataset into equal-sized subsets as shown in Fig. 3.4.2(a), the overall time complexity is $\mathcal{O}(|F| \cdot |D| \cdot d_{max})$. However, if we assume that the random node layers are specified, then overall computation complexity reduces to $\mathcal{O}(|F| \cdot |D| \cdot (d_{max} - d_{rn}))$.

The most computationally heavy part of the tree construction is the split evaluation. For a node in a balanced tree at depth d_i, it takes $|F| \cdot \frac{|D|}{2^{d_i}}$ computations for split evaluation. This follows from the assumption that each node's split criterion partitions the dataset into two roughly equal subsets. Thus, summing across all nodes at depth d_i there will be $|F| \cdot |D|$ split evaluations for each layer. Given that the tree



(a)



(b)

Figure 3.4.2: Illustration of decision tree structures. (a) Balanced tree, where each split evenly partitions the dataset D with a specified maximum depth d_{max}. (b) Imbalanced tree, where splits generate highly unequal partitions, leading to deeper trees with unspecified maximum depth.

is balanced and assuming the maximum depth is d_{max}, then there are $|F| \cdot |D| \cdot d_{max}$ overall evaluations. If the number of random node layers is specified, then overall evaluation is reduced to $|F| \cdot |D| \cdot (d_{max} - d_{rn})$, because the random selection of the threshold takes a constant amount of time.

However, if the tree is imbalanced as shown in Fig. 3.4.2(b) where split partitions the dataset into two highly unequal-size subsets, the overall time complexity increases to $\mathcal{O}(|F| \cdot |D|^2)$. This quadratic complexity arises from inefficient splits, where each partition produces one leaf node with a small subset and one decision node that retains

nearly the entire dataset. Consequently, each decision node must process nearly the entire dataset. If the maximum depth of the tree, d_{max}, is not specified, then tree splitting continues until each leaf node contains one data point, which means the resulting tree depth approaches $|D|$. Since the number of evaluations per layer takes $|F| \cdot |D|$, the overall computation time increases to $\mathcal{O}(|F| \cdot |D|^2)$.

Unlearning time complexity. The time complexity of the unlearning process in a semi-supervised decision tree depends on the structure and depth where retraining occurs.

In the worst-case scenario, retraining starts at the root node, meaning the entire tree must be reconstructed. In this case the unlearning time is similar to the tree construction time complexity, which is either $\mathcal{O}(|F| \cdot |D|^2)$ if the tree is imbalanced or $\mathcal{O}(|F| \cdot |D| \cdot d_{max})$ if the tree is balanced. This occurs because retraining from the root node requires re- evaluating all splits and rebuilding the tree from the very beginning, making it computationally expensive.

The best-case scenario happens when there is no retraining taking place in a tree, and the tree is assumed to be balanced. In this scenario, the unlearning algorithm only re-evaluates the nodes directly affected by the removal of data. Since only one node per layer is affected, the time complexity is $\mathcal{O}(\sum_{i=0}^{d_{max}} |F|\frac{|D|}{2^{d_i}})$. This is because at each node at depth d_i, the split evaluation takes $\mathcal{O}(|F|\frac{|D|}{2^{d_i}})$ time, and since only one node is affected per layer, the total cost is the sum of these evaluations across all layers.

An average case occurs when the retraining takes place between the root node and leaf nodes. If the d_{ul} is the intermediate depth where the retraining takes place, then the time complexity for unlearning is $\mathcal{O}(\sum_{i=0}^{d_{ul}} |F|\frac{|D|}{2^{d_i}} + |F|\frac{|D|}{2^{d_{ul}}}(d_{max} - d_{ul}))$. As it can be seen, the time complexity for an average case consists of two parts. The first term $\sum_{i=0}^{d_{ul}} |F|\frac{|D|}{2^{d_i}}$ accounts for re-evaluating nodes from the root down to depth d_{ul} where no structural retraining is required. The second part $|F|\frac{|D|}{2^{d_{ul}}}(d_{max} - d_{ul})$ reflects the cost of rebuilding the subtree below d_{ul}, which involves processing of $\frac{|D|}{2^{d_{ul}}}$ data points at each of the remaining $(d_{max} - d_{ul})$ layers.

Chapter 4

Experimental Setup

This chapter discusses the experimental setup used to construct and unlearn the semi-supervised decision tree and random forest models. The experiments are designed to evaluate unlearning efficiency and the accuracy of the unlearned model compared to the fully retrained counterpart. The main goal of this setup is to establish the effectiveness of the unlearning algorithm in handling the data removal in the semi-supervised decision tree model and to analyze the impact of unlearning and retraining on model performance.

All experiments are carried out using Python version 3.12.2 on a system with an Intel Core i7-7990X processor, 32 GB of RAM, and Windows 10 OS. To provide a complete evaluation, tests were conducted on several datasets with different conditions, including different amounts of labeled data.

The following subsections provide detailed information about the datasets, evaluation metrics, and implementation specifics used in the experiments.

4.1 Datasets

To test the unlearning ability of the proposed method, 10 distinct datasets were used. These datasets include both binary and multiclass classification tasks to ensure the adaptability of the unlearning algorithm. Additionally, the datasets can be categorized into three groups based on their size: small, moderate, and large datasets. Using

different dataset sizes allows for a more profound understanding of the behavior of the proposed method across the different scales.

Table 4.1: Datasets

Dataset ID	**Dataset Name**	**Data Points**	**Features**	**Classes**
1	EEG Eyestate [27]	14,980	14	2
2	BNG TIC-TAC-TOE [28]	39,366	10	2
3	Loan [29]	45,000	13	2
4	Adults [30]	48,842	14	2
5	Sepsis [31]	110,341	3	2
6	Skin Segmentation [32]	245,057	3	2
7	BNG Wine [28]	1,000,000	14	3
8	Synthetic	1,000,000	10	2
9	HAR70 [33]	2,259,597	6	8
10	HARTH [34]	6,461,328	6	12

Table 4.1 presents the 10 datasets that are used during the experiment. The datasets are further categorized into three size groups: first four datasets are considered small, then the following two datasets are considered moderate, and the last four are considered large-sized. This categorization allows us to evaluate the proposed method across a wide range of data scales and complexities.

The small-sized dataset consists of EEG Eyestate [27], Tic-Tac-Toe [28], Loan [29], and Adults [30] datasets. As can be seen from the table, all four datasets were used for the binary classification tasks. The EEG Eyestate dataset consists of 14,980 data points and is used to predict whether an individual's eyes are open or closed based on EEG signal data. Similarly, the BNG TIC-TAC-TOE dataset, with 39,366 data samples, was used to classify whether an individual won the game or not. The loan dataset, which has 45,000 instances, and the Adults dataset, consisting of 48,842 data samples, are used in credit risk assessment to predict loan approval based on applicant information.

The moderate category includes the Sepsis [31] and Skin Segmentation [32] datasets. Both datasets have more than 100,000 data points; however, they have relatively small feature sets, making them suitable for evaluating the algorithm's performance on moderately large datasets with limited feature dimensions.

The last four datasets are considered to be the large datasets with more than 1,000,000 data instances. Furthermore, the datasets BNG Wine [28], HAR70 [33], and HARTH [34] are used for the multiclass classification tasks.

4.2 Testing Methods

To evaluate the overall performance of the proposed unlearning method, tests compare the unlearned model with the retrained model in terms of unlearning time, retraining time, and predictive performance. The quality of unlearning will be assessed by applying a backdoor poisoning method. This method will help us to analyze how well the proposed unlearning algorithm can eliminate the effect of poisoned data while maintaining the model's accuracy. These approaches ensure a comprehensive assessment of both efficiency and effectiveness in unlearning compared to conventional retraining.

4.2.1 Unlearning Time and Prediction Performance

To measure time and accuracy, the following steps should be taken. First, several semi-supervised tree models should be trained using varying amounts of labeled samples. Second, a randomly selected data point is removed from each model using the unlearning method and retraining process. Last, the unlearned and retrained models' prediction accuracies and their retraining and unlearning times are compared. Furthermore, larger subsets of training datasets are unlearned to assess how the unlearning method performs when large portions of data are requested for removal. The same metrics—accuracy, unlearning time, and retraining time—are tracked to evaluate performance under bulk removal requests. These metrics will help us to evaluate the robustness of the unlearning method and its efficiency relative to complete retraining.

4.2.2 Backdoor poisoning

To evaluate the overall unlearning quality, the backdoor poisoning method is used. *Backdoor poisoning* is an adversarial attack where an adversary modifies the training dataset to introduce vulnerability in a model [35]. This vulnerability enables adversaries to alter the model's behavior in response to specific inputs. This behavior causes instability in the model. This type of attack is particularly concerning because it is difficult to detect. During standard evaluations, the model performs as expected on clean, non-poisoned data. However, when specific trigger inputs are faced, the model might misclassify the output [36]. This characteristic creates a false sense of security during model evaluation and testing phases. Security audits that rely on standard accuracy metrics might completely miss these embedded vulnerabilities, allowing compromised models to be deployed in critical systems [37]. Furthermore, backdoor attacks represent a persistent threat that continues to impact systems long after the initial poisoning event. Once a backdoor is successfully implanted in a model, it remains effective for the entire lifecycle of that model, potentially causing security breaches months or years after deployment [38].

The backdoor poisoning method is a highly effective approach for testing unlearning. The focus on backdoor poisoning in this study directly addresses the real-world risks. This type of attack involves injecting specific triggers into the dataset, causing the trained model to alter its behavior when it encounters those triggers. The purpose of the unlearning procedure is to revert the effect of those triggers without complete model retraining while maintaining model prediction capability. In other words, unlearning should eliminate the impact of injected triggers from the model, restoring it to a secure and reliable state. By demonstrating that we provide a pathway to mitigate supply-chain attacks and maintain compliance in dynamic, adversarial environments. This is particularly relevant for applications where retraining is prohibitively expensive.

Section 5.3 provides additional details regarding the implementation and evaluation of backdoor poisoning attacks and their impact on model performance.

4.3 Optimization

Python is one of the most widely used programming languages for machine learning and data science because of its large library support. However, Python's execution speed is comparatively slower than compiled programming languages like C or C++. This problem arises due to the various structural and design choices.

One of the main bottlenecks in Python is function call overhead. Each function call allocates a stack frame in the memory, allocates the memory for the variables, and handles execution flow back to the caller after the function completes. Since Python runs on a virtual machine, not directly on hardware, the function call becomes computationally expensive. As is well known, a decision tree is constructed using recursive function calls, which will slow the training and unlearning processes if this function overhead is not handled properly. Moreover, Python is considered a dynamically typed programming language, which means variable types are determined at runtime. Every time the function is called, Python looks up the function reference, checks and resolves the types of input arguments, and performs runtime binding.

Additionally, Python does not support Tail Call Optimization (TCO), which allows the reuse of the current stack frame for recursive calls and limits excessive memory usage. This means deep recursive calls, as in the decision tree construction, could lead to inefficient memory allocation, computational overhead, and stack overflow errors.

To address these issues, Just-in-Time (JIT) or Ahead-of-Time (AOT) compilers could be used. Both compilers are intended to gain performance by compiling higher-level languages into native machine code, reducing the need for repeated interpretation [39]. Furthermore, JIT and AOT compilers speed up numerical computations by optimizing loops and reducing function call overhead [40], which is useful for evaluating the split conditions.

The main difference between the AOT and JIT compilers lies in when the compilation occurs. The AOT compiler converts the code into machine code during compilation time, which is before execution. In contrast, a JIT compiler converts

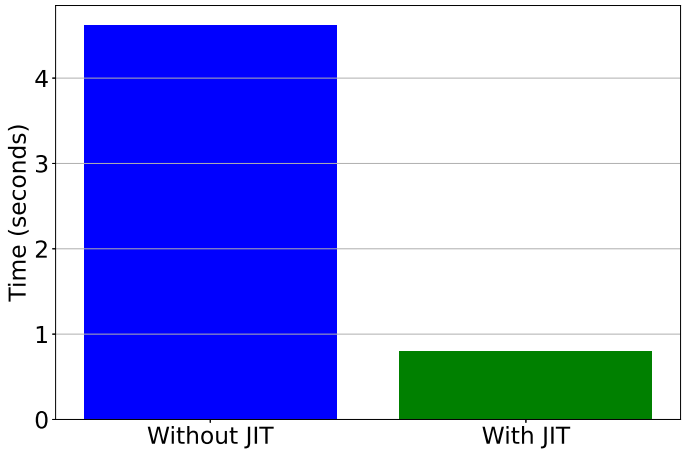


Figure 4.3.1: Tree construction time without and with JIT compiler

higher-level language code into native machine code during runtime.

Fig. 4.3.1 compares the training times of the decision tree models with 10,000 data instances with 10 features, using JIT compilation (green) and without it (blue). As shown in the figure, JIT-compiled models provide significantly faster execution than the traditional Python interpreter. This improvement in training time shows that the JIT compiler optimizes function calls, reduces interpretation overhead, and accelerates overall numerical computations. This result supports the usage of the JIT compiler to improve overall training and unlearning time because decision tree construction involves several recursive function calls with intensive computations.

Chapter 5

Results

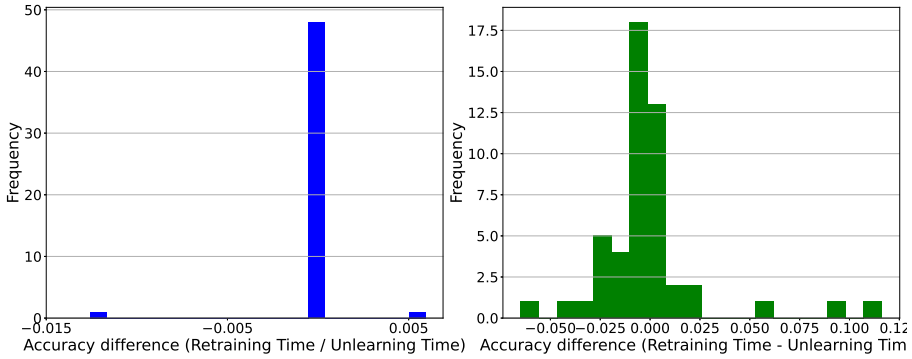
This chapter presents the results derived from the experiments. By systematically analyzing the results, this chapter provides an understanding of how the proposed method compares to traditional retraining approaches in terms of both time efficiency and predictive performance.

To evaluate the time efficiency of unlearning, the *efficiency ratio*, defined as the retraining time divided by unlearning time, is used. This ratio provides a measure of the computational gain provided by the unlearning method compared to retraining. Additionally, to evaluate the impact of unlearning on prediction performance, *accuracy difference*, defined as the difference in accuracy between the retrained and unlearned models, is used. The accuracy difference reflects how closely unlearning approximates retraining results. These metrics allow for a comprehensive comparison of both computational efficiency and model performance.

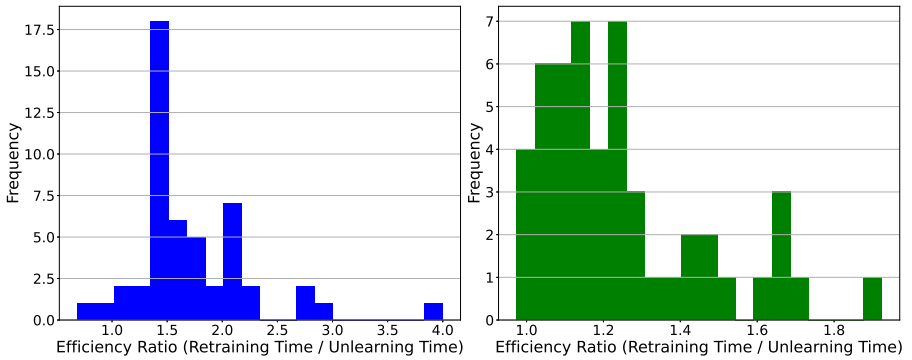
The first section focuses on the time speedups that are gained by unlearning a single data point. The second section presents the results of the method's capability in removing subsets of the dataset, assessing its efficiency in large-scale unlearning scenarios. Finally, in the third section, the quality of the unlearning process is evaluated using backdoor poisoning attacks.

5.1 Unlearning the Single Data

This section evaluates the efficiency of the proposed unlearning method by comparing the time required to unlearn a single data point with full model retraining. To properly evaluate the unlearning, several semi-supervised models with different labeled sample sizes are constructed, and a randomly selected single data point is unlearned. For comparison, models are retrained from scratch after removing the same data point.



(a)



(b)

Figure 5.1.1: Single data unlearning for the semi supervised decision tree (blue) and random forest (green) (a) Distribution of accuracy differences (b) Distribution of efficiency ratio

The results of the single data removal case are shown in Fig. 5.1.1. The histograms reveal distinct trends for decision trees and random forests. For decision trees (blue histograms), the accuracy differences are concentrated at zero, indicating that the unlearned model is exactly the same as the retrained model. The efficiency ratio

peaks at 1.5, showing that unlearning is 50% faster than retraining on average. In contrast, random forests (green histograms) show broader distributions for both metrics. While accuracy differences remain centered near zero, the efficiency ratio varies significantly. This is primarily because of bootstrapped samples, as the removal data point may appear multiple times in the training subsets of different trees. In this case, instead of removing the single data point, several data points may require simultaneous removal within a single tree. When multiple instances are removed, the likelihood that removing these data points will affect not just leaf nodes but potentially nodes higher up in the tree increases. If the retraining starts at or near the root node due to multiple removals, it forces tree reconstruction, which significantly increases computational complexity and unlearning time.

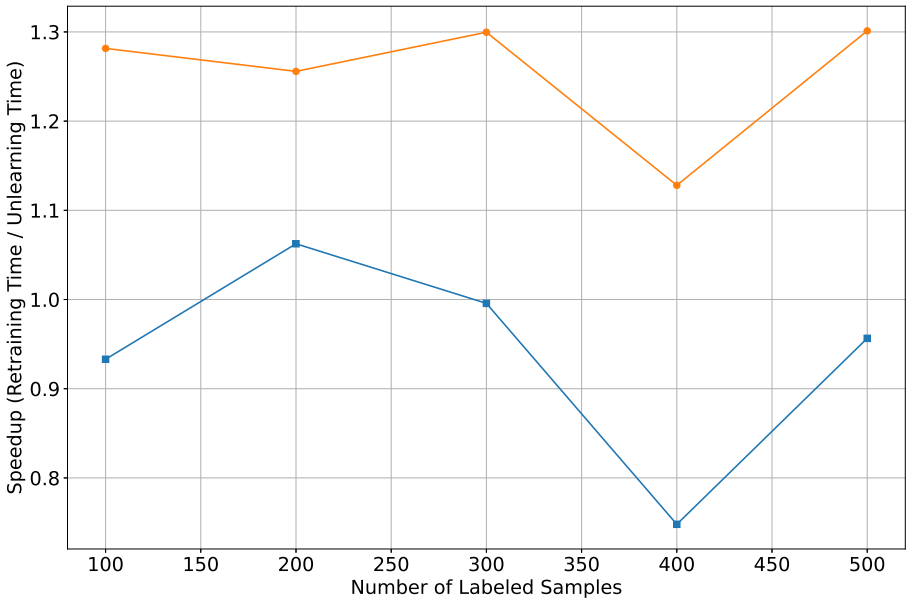


Figure 5.1.2: Unlearning speed up in Random Forest model with (blue) and without (orange) bootstrapping

As shown in Fig. 5.1.2, two different random forest models were unlearned: one trained with bootstrapping (blue) and another without bootstrapping (orange). From the figure, the no-bootstrapping model typically shows higher speedup values—often staying around 1.3, suggesting that unlearning clearly outperforms retraining in those scenarios. In contrast, the bootstrapping model fluctuates more and sometimes dips below 1.0, indicating that unlearning is struggling to remove a single data point.

Overall, these findings highlight how the use of bootstrapping significantly affects unlearning efficiency.

5.2 Unlearning the Subset

This section extends the unlearning evaluation to larger data subsets. It analyzes how the removal of labeled and unlabeled data impacts model performance and efficiency. As in the previous experiment, semi-supervised decision trees and random forest models are trained with varying labeled sample sizes. The key distinction here is the focus on subset removal: instead of removing a single data point, we unlearn labeled and unlabeled subsets independently, progressively increasing the size of the removed data to test scalability. This experiment allows for an in-depth analysis of how the removal of labeled and unlabeled data impacts the overall performance of the model.

5.2.1 Unlearning the Labeled Subset

Figs. 5.2.1 and 5.2.2 illustrate the accuracy-difference and efficiency-ratio histograms for the unlearning of labeled subsets. Each subfigure corresponds to a specific percentage of labeled data removed from the dataset: 0.1%, 1%, 10%, and 20%. We analyze data removal for both the decision tree and random forest models.

For decision trees (blue), removing 0.1% of labeled data (Fig. 5.2.1(a)) results in a narrow accuracy difference distribution centered at zero, indicating that the model's performance remains unaffected by the removal of such a small portion of labeled data. At 1% removal (Fig. 5.2.1(b)), the distribution widens slightly. By 10% and 20% removal (Figs. 5.2.1(c) and 5.2.1(d)), the spread increases further. However, even as the accuracy difference distribution broadens, it is still centered at 0, which indicates that, on average, the retraining and unlearning models are similar in terms of prediction performance.

The Random Forest model (green), however, shows a wider spread distribution in each case, indicating that accuracy differences are more varied. Unlike a single Decision Tree, which uses deterministic splits based on the full feature set, a Ran-

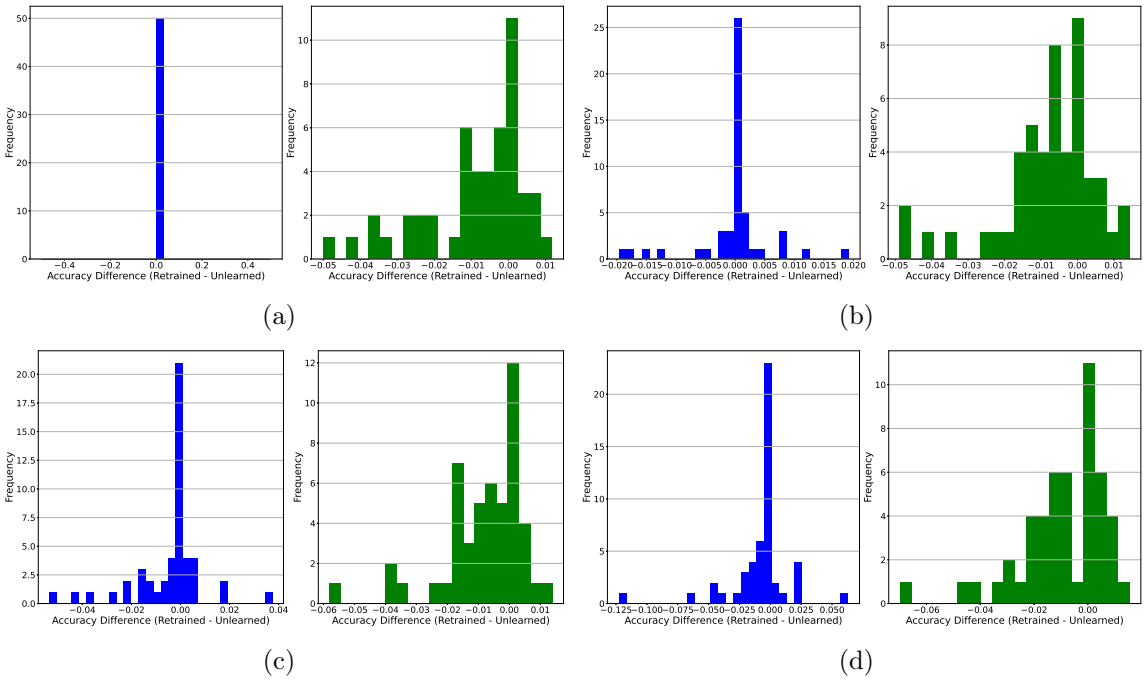


Figure 5.2.1: Distribution of accuracy differences (Retrained - Unlearned) across varying proportions of labeled data in a semi-supervised setting. Subfigures represent removed labeled data subsets: (a) 0.1%, (b) 1%, (c) 10%, (d) 20%. Decision Tree (blue) and Random Forest (green).

dom Forest trains multiple trees with random subsets of features at each node. This randomness results in variability of how splits are optimized during training. When unlearning is applied, the retrained and unlearned models may diverge because retraining reconstructs trees by randomly selecting features for node splits, which may differ from the original feature subsets preserved in the unlearned model. These differences in feature selection propagate through the tree structure, altering hierarchical prediction pathways. Nevertheless, even if there is a spread in the distribution of accuracy differences, it is small and mostly concentrated around zero, which means the prediction performance of the unlearned model is similar to the retrained model.

The efficiency ratio histograms are shown in Fig. 5.2.2. From the histograms it is evident that the larger the removal subset size, the smaller the efficiency ratio, indicating diminishing time savings as more data is removed. For example, the efficiency ratio for the decision tree model at 0.1% removal is concentrated around 2, meaning unlearning is twice as fast as retraining. However, as the removal size increases to 1%,

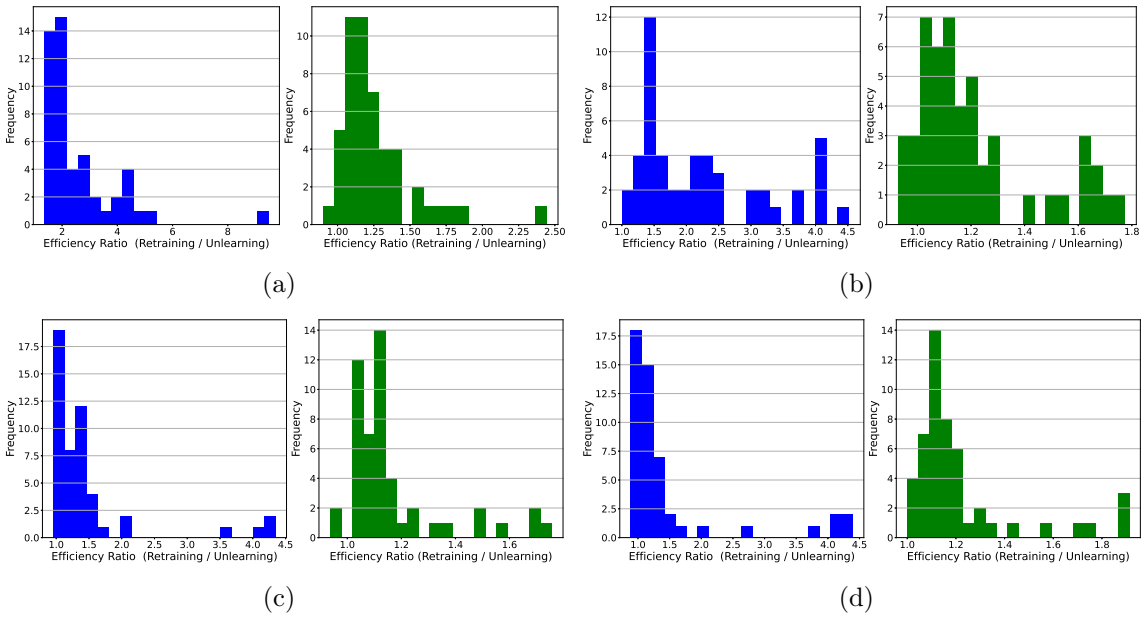


Figure 5.2.2: Unlearning time efficiency (Retraining Time / Unlearning Time) across varying proportions of labeled data in a semi-supervised setting. Subfigures correspond to removed labeled data subsets: (a) 0.1%, (b) 1%, (c) 10%, (d) 20%. Decision Tree (blue) and Random Forest (green).

the ratio drops to 1.5. At 10% and 20% removal, the ratio approaches 1, suggesting that unlearning takes almost the same amount of time as the retraining.

Random forests exhibit an intriguing pattern. At 0.1% removal, the efficiency ratio distribution is wide but centered at 1.2. As the removal size increases to 1–20%, the distribution narrows and shifts closer to 1.1. While the accuracy difference histograms provided pretty satisfactory results, the efficiency ratio distribution histogram indicates that unlearning the large subset of the random forest model requires the same amount of time as retraining it.

5.2.2 Unlearning the Unlabeled Subset

Fig. 5.2.3 provides information about the accuracy difference for semi-supervised decision trees (blue) and random forests (green) across varying proportions of unlabeled data removal (0.1%, 1%, 10%, and 20%). Fig. 5.2.3 provides a detailed analysis of how the accuracy difference metric behaves under different unlabeled data removal scenarios in a semi-supervised learning framework.

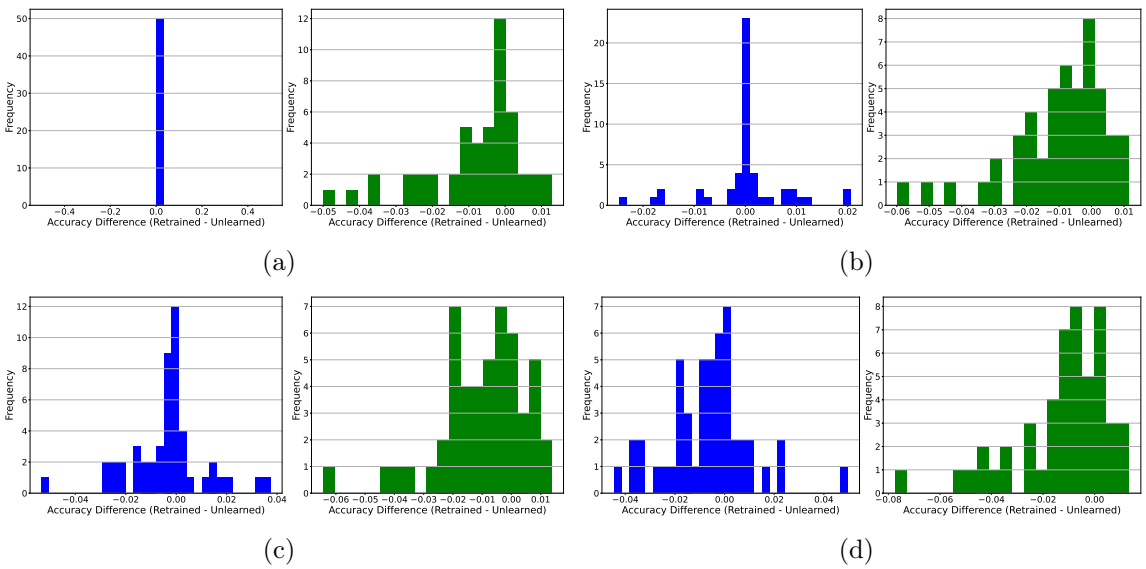


Figure 5.2.3: Distribution of accuracy differences (Retrained - Unlearned) across varying proportions of unlabeled data in a semi-supervised setting. Subfigures represent removed unlabeled data subset: (a) 0.1%, (b) 1%, (c) 10%, (d) 20%. Decision Tree (blue) and Random Forest (green)

The trend in the distribution of accuracy differences for the decision tree model (blue) in Figs. 5.2.3(a) and 5.2.3(b) is similar to the earlier observations. The distribution peaks at zero, indicating minimal difference between retrained and unlearned models. However, in Figs. 5.2.3(c) and 5.2.3(d), the accuracy difference distribution of the decision tree started broadening. Despite this broadening, the accuracy difference concentrated near 0. This indicates that the difference in prediction performance between the retrained and unlearned models is minimal. The same conclusion could be applied to the random forest models (green): even at 10%–20% unlabeled data removal, the accuracy differences remain distributed around zero, reflecting the similarity of the unlearning and retraining models.

Fig. 5.2.4 illustrates histograms comparing the efficiency ratios across varying proportions of unlabeled data removed in a semi-supervised learning setting. The results reveal unexpected trends in computational efficiency.

When a 0.1% subset of the dataset was removed, the efficiency ratio of the decision tree was concentrated around 1.5, indicating that unlearning is 50% faster than retraining. Additionally, the random forest's efficiency ratio centered around 1.1, in-

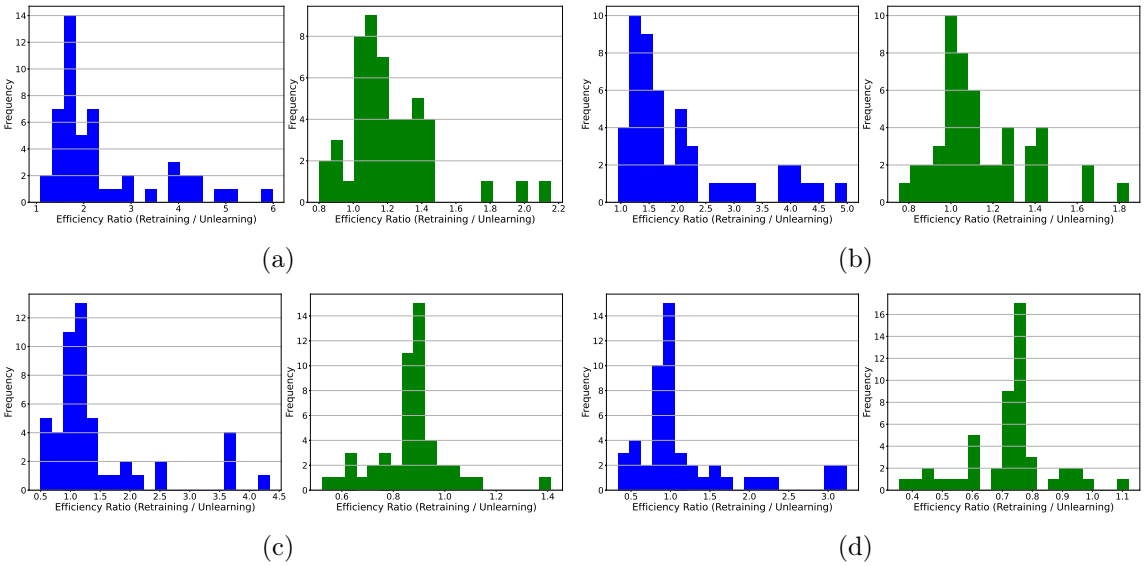


Figure 5.2.4: Unlearning time efficiency (Retraining Time / Unlearning Time) across varying proportions of unlabeled data in a semi-supervised setting. Subfigures represent removed unlabeled data subset: (a) 0.1%, (b) 1%, (c) 10%, (d) 20%. Decision Tree (blue) and Random Forest (green).

dicating that time savings are minimal. As expected, as the subset size increased, efficiency ratio decreased, and the efficiency ratio started shifting leftward, meaning that retraining the model was more efficient than unlearning it. Furthermore, at 10% and 20% subset sizes, the efficiency ratio for the random forest model became less than 1, indicating that unlearning large portions of the unlabeled data is inefficient.

5.3 Backdoor Poisoning

As outlined in the section 4.2.2, the backdoor poisoning attack is an adversarial attack used to assess the quality of unlearning algorithms. Adversaries manipulate training data by inserting specific triggers that alter the model's behavior without affecting its prediction performance on clean data. To evaluate the unlearning quality, we take the following steps:

1. Poison the dataset. Randomly select one feature from the feature set and modify its values in 30% of the data samples by replacing them with an uncommon value. Additionally, for these data points, the same class labels are attached.

The overall poisoning workflow is shown in Fig. 5.3.1. As shown in the figure, the attackers first get access to the training dataset. After that, they generate specific triggers that alter the model's behavior. These triggers are then embedded into the training dataset, and this dataset is published to the public.

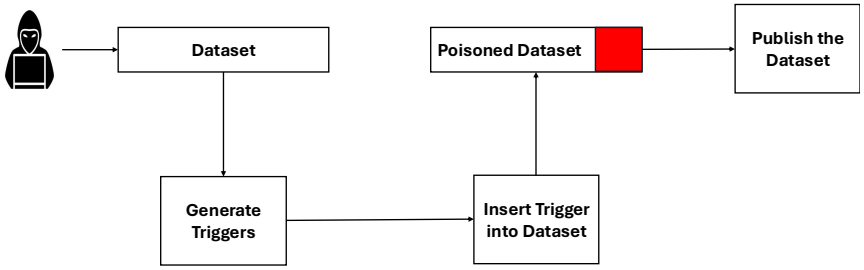


Figure 5.3.1: Poisoning workflow: (1) Select a feature, (2) Generate triggers (uncommon values), (3) Insert triggers into the dataset, and (4) Publish the poisoned dataset.

2. Train models. Construct the semi-supervised decision tree and random forest models with the poisoned dataset. Test the model's predictive performance using both clean and poisoned (contain triggers) test sets. The overall workflow of the second step is shown in Fig. 5.3.2.

3. Unlearn poisoned data. Remove the poisoned data from the models using the unlearning method. Next, evaluate the model's performance on both the clean and poisoned test sets to assess how well the unlearning method handled removing the poisoned data. The overall workflow of the third step is shown in Fig. 5.3.3.

The results of the backdoor poisoning test are shown in Fig. 5.3.4. These results show how well the proposed unlearning method removes the impact of poisoned data on the semi-supervised decision tree and random forest models. The presented figure

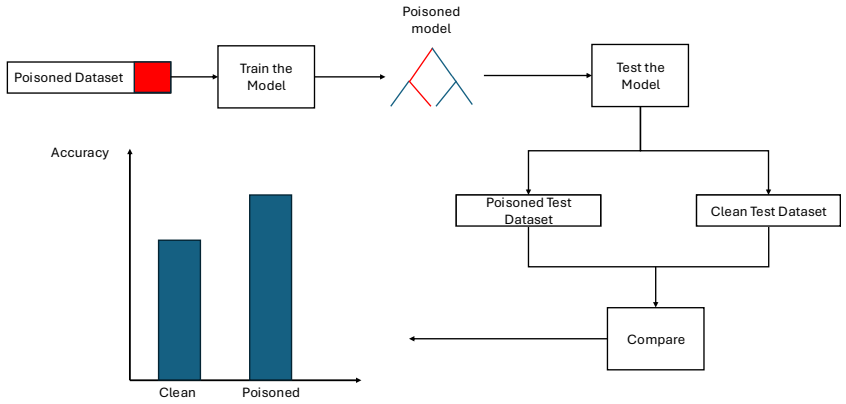


Figure 5.3.2: Model training and evaluation workflow: (1) Train models on poisoned dataset, (2) Test performance on clean and poisoned test sets.

consists of several subfigures, and each subfigure consists of two subplots. The left subplot corresponds to the backdoor poisoning testing for the decision tree model, while the right subplot corresponds to the random forest model.

For each dataset, accuracy metrics are plotted against varying labeled sample sizes. The blue solid line represents the accuracy of the model on the clean test set before unlearning, and the green solid line indicates the accuracy of the model on the poisoned test set before unlearning. The orange dashed line shows how accurate the model was on clean test data after unlearning, and the red dashed line indicates how accurate the model was on poisoned test data after unlearning, indicating whether the model retains the poisoned data effect.

From Fig. 5.3.4, the following observations can be made. Across all datasets, the poisoned test set accuracy (red dashed line) shows a significant drop compared to the poisoned accuracy before unlearning (green solid line). Additionally, the impact of the unlearning on the overall model's performance is negligible because the accuracy of the clean test data after unlearning (the orange dashed line) closely follows the accuracy of the clean test data before unlearning (the blue solid line). This result suggests that the algorithm selectively removes poisoned data samples without compromising predictive performance.

Based on these results, it can be concluded that the proposed unlearning method

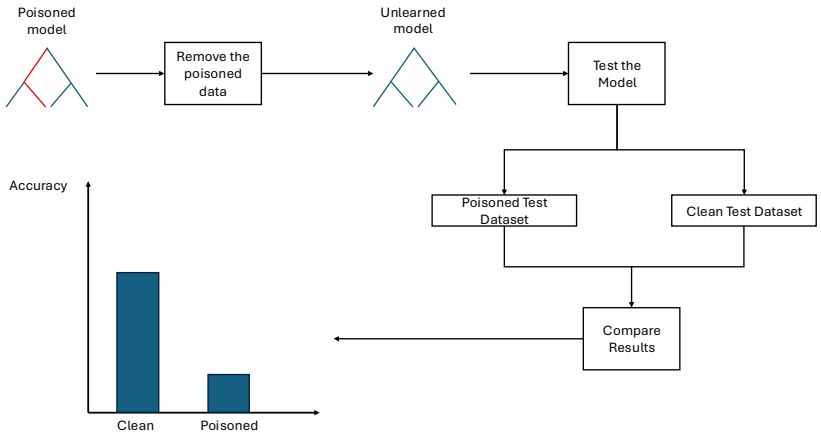


Figure 5.3.3: Unlearning validation workflow: (1) Remove poisoned data from model, (2) Test unlearned model on clean and poisoned datasets, (3) Compare accuracy improvements to verify backdoor removal.

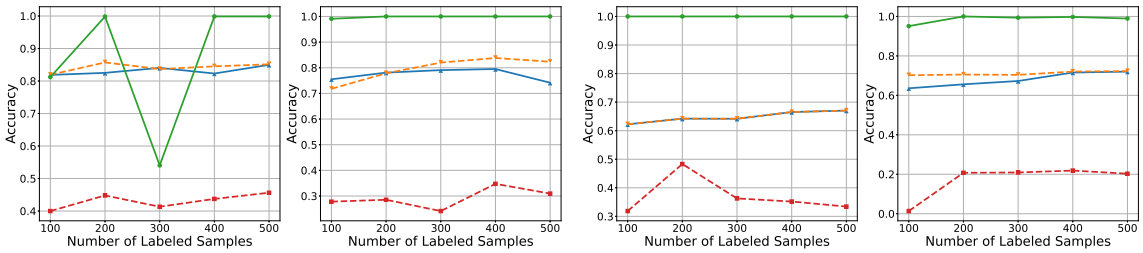
can effectively mitigate the backdoor effects, making the model behave as if it had never encountered the poisoned samples.

5.4 Discussion

The experimental results demonstrated that the proposed unlearning algorithm for semi- supervised decision tree based models can effectively remove influence of data points from the model without retraining. The analysis was conducted in three main areas: (1) single data unlearning; (2) subset unlearning; and (3) quality of the unlearning. Each of these areas provided intuition into the computational efficiency and quality of the proposed unlearning method compared to full model retraining.

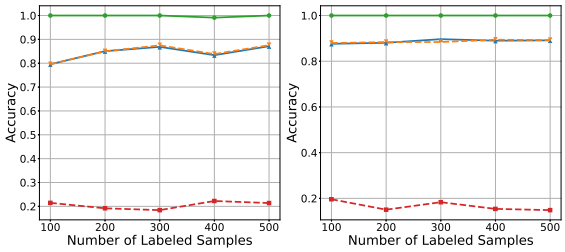
5.4.1 Unlearning Efficiency

The most significant finding from the conducted experiments is that the proposed unlearning method is computationally cheaper than retraining the entire model from scratch. The experiments demonstrated that for the single data removal case, unlearning achieved at least one and a half times faster results for semi-supervised decision trees. Moreover, in subset unlearning cases based on the removal subset size, the

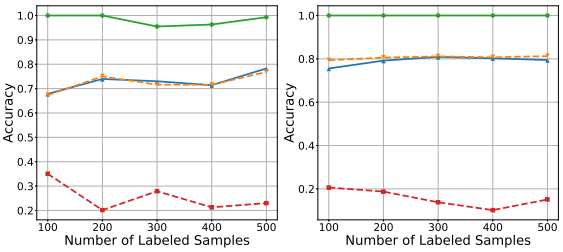


(a) Eyestate

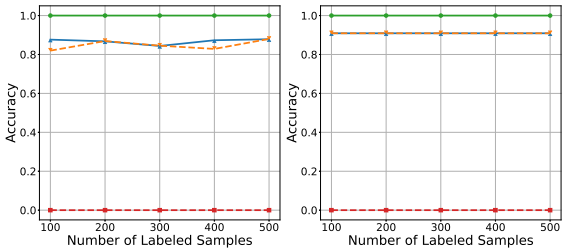
(b) Tic-Tac-Toe



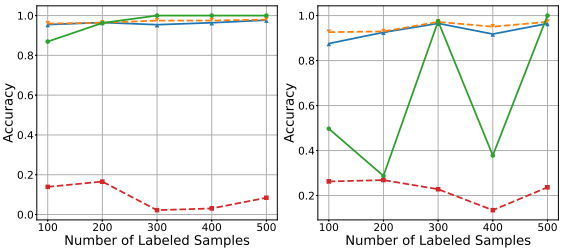
(c) Loan



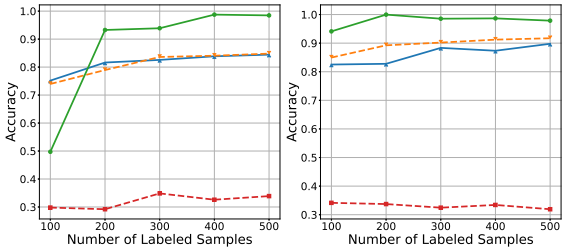
(d) Adults



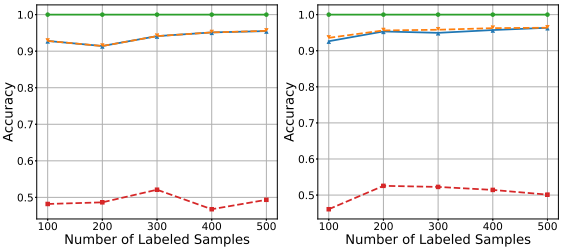
(e) Sepsis



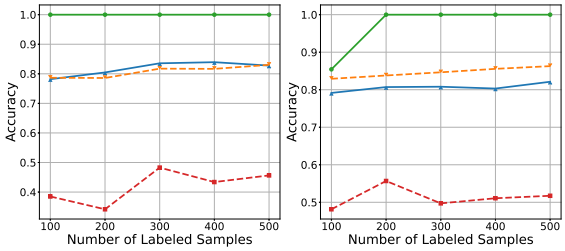
(f) Skin Segmentation



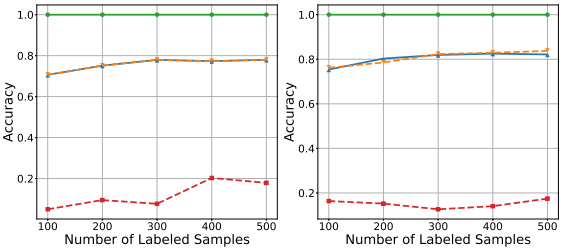
(g) BNG Wine



(h) Synthetic



(i) HAR70



(j) HARTH

Figure 5.3.4: Backdoor poisoning testing showing accuracy on clean data before the unlearning (blue solid line), accuracy on the poisoned data (green solid line), accuracy on the clean data after the unlearning (orange dashed line), and accuracy on the poisoned test set (red dashed line).

unlearning provided a boost in computational speed of a maximum of between two and four times faster results. However, the results for the semi-supervised random forest models were not as satisfactory as for the semi-supervised decision trees. The random forest model achieves an average speedup of 1.3 when removing an individual data point. The worst-case scenario occurred in the subset of unlearning cases, where the unlearning time was similar to the retraining time. However, as previously mentioned, the bootstrapped samples were responsible for all these outcomes. In a random forest, bootstrapping causes the same data point to appear multiple times in a single tree, unlike in a single decision tree, where it appears only once. As a consequence, when unlearning is performed, the same data point must be removed from multiple trees multiple times, often leading to deeper retraining in several trees.

5.4.2 Accuracy Retention and Model Performance

One of the critical aspects of the unlearning is how well the model maintains its predictive performance after data removal. For semi-supervised decision trees, retrained and unlearned models exhibit nearly identical accuracies in single-data removal cases. However, in the case of the semi-supervised random forest, small differences exist between the retrained and unlearned models. This is likely due to randomized feature subsets. As previously discussed, a random forest trains multiple trees with random subsets of features at each node. This randomness results in variability in how splits are optimized during training. When unlearning is applied, the retrained and unlearned models may diverge. This is because retraining reconstructs trees by randomly selecting new features for node splits. These selections may differ from the original feature subsets preserved in the unlearned model. However, these differences are small, and they can be neglected. Similarly, subset unlearning introduces only minor fluctuations, which remain insignificant, as evidenced by histograms showing accuracy differences concentrated near zero.

5.4.3 Quality of the Unlearning

The quality of the proposed unlearning method was tested by the backdoor poisoning method. Initially, models trained on poisoned data performed differently when tested on clean versus poisoned test sets. Specifically, while the model performed well on standard clean test data, it showed significantly high accuracy when backdoor-triggered inputs were presented. After the unlearning, the prediction performance of the model on a clean test set remains similar; however, the accuracy of the model to the backdoor-triggered inputs significantly dropped. This evidence indicates that the proposed unlearning method was able to remove the influence of poisoned data and heal the model.

5.4.4 Limitations and Future Considerations

While the results proved the efficiency and effectiveness of the proposed unlearning algorithm for the semi-supervised decision tree models, there are certain limitations that require further investigation:

1. Managing unlearning for bootstrapped samples. As previously mentioned, using bootstrapped samples reduces the efficiency of unlearning. To handle the unlearning of bootstrapped samples more efficiently and computationally cheaper, further investigation is necessary.

2. Defining a clear boundary for deciding when to retrain the model instead of unlearning is an important aspect that requires further exploration. Future research should focus on developing adaptive strategies that determine whether unlearning or full model retraining is more optimal based on the number of removal samples, the tree depth, and the structure of the decision nodes.

Further research is required to optimize the unlearning of bootstrapped samples, as their presence significantly affects computational efficiency. Handling these samples more effectively will allow for a more cost-effective unlearning process, particularly

in ensemble models where bootstrapped training data complicates data removal. Additionally, an important area of investigation involves defining a clear boundary for deciding when full retraining is more efficient than unlearning. Subset removal experiments have shown that, in some cases, retraining can be more cost-effective than the unlearning method. Factors such as the size of the removal data, the depth of the tree, and the structural complexity of the model should be considered when developing a boundary between retraining and unlearning. These challenges highlight the need for continued research on unlearning methods to improve their efficiency.

Chapter 6

Conclusion

This thesis examined the development and implementation of an effective machine unlearning technique for semi-supervised decision tree models. The proposed method addresses the challenges of unlearning in models that use both labeled and unlabeled data. It ensures compliance with data privacy regulations while preserving computational efficiency and predictive performance.

Extensive experimentation demonstrated that the proposed unlearning algorithm significantly reduces the computational time taken for data removal in contrast to entire model retraining. The unlearning algorithm that uses the subtree retraining method proved to be highly effective for single data removal, reducing the frequency of retraining operations while maintaining the integrity of the decision tree structure. However, extending the algorithm to random forest models did not yield the same efficiency as it did for the decision tree. This effect was particularly evident when bootstrapped samples in the random forest model resulted in excessive retraining at the root node.

This study also revealed the effectiveness of unlearning in preserving model accuracy. In most of the scenarios, the predictive performance of the unlearning model was comparable with the retrained model. Backdoor poisoning tests also provided a comprehensive assessment of unlearning quality. The findings demonstrated that the proposed unlearning technique entirely eliminated the influence of contaminated data, causing the model to function as if it had never seen the deleted data.

Despite these promising results, several limitations remain. The efficiency of unlearning the bootstrapped samples in random forests still requires further research. Additionally, there is a need to define a clear boundary for deciding when full retraining is more appropriate than applying unlearning methods. It is important to ensure that the model retains its overall generality.

Bibliography

[1] P. Voigt and A. Bussche, *EU General Data Protection Regulation (GDPR): A practical guide*. SPRINGER INTERNATIONAL PU, 2018.

[2] GDPR.eu. [Online]. Available: `https://gdpr.eu/article-17-right-to-be-forgotten/`.

[3] T. T. Nguyen, T. T. Huynh, Z. Ren, *et al.*, "A survey of machine unlearning," 2024. arXiv: `2209.02299 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2209.02299`.

[4] Z. Liu, H. Ye, C. Chen, and K.-Y. Lam, "Threats, attacks, and defenses in machine unlearning: A survey," *ArXiv*, vol. abs/2403.13682, 2024. [Online]. Available: `https://api.semanticscholar.org/CorpusID:268536826`.

[5] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, *et al.*, "Machine unlearning," 2020. arXiv: `1912.03817 [cs.CR]`. [Online]. Available: `https://arxiv.org/abs/1912.03817`.

[6] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 463–480. DOI: `10.1109/SP.2015.35`.

[7] A. Ginart, M. Y. Guan, G. Valiant, and J. Zou, "Making ai forget you: Data deletion in machine learning," 2019. arXiv: `1907.05012 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1907.05012`.

[8] Y. C A Padmanabha Reddy, P. Viswanath, and B. Eswara Reddy, "Semi-supervised learning: A brief review," *International Journal of Engineering & Technology*, vol. 7, no. 1.8, p. 81, Feb. 2018. DOI: 10.14419/ijet.v7i1.8.9977.

[9] H. C. Tanuwidjaja, R. Choi, S. Baek, and K. Kim, "Privacy-preserving deep learning on machine learning as a service—a comprehensive survey," *IEEE Access*, vol. 8, pp. 167 425–167 447, 2020. DOI: 10.1109/access.2020.3023084.

[10] Z. Izzo, M. Anne Smart, K. Chaudhuri, and J. Zou, "Approximate data deletion from machine learning models," in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, A. Banerjee and K. Fukumizu, Eds., ser. Proceedings of Machine Learning Research, vol. 130, PMLR, Apr. 2021, pp. 2008–2016. [Online]. Available: https://proceedings.mlr.press/v130/izzo21a.html.

[11] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 463–480. DOI: 10.1109/SP.2015.35.

[12] M. Kearns, "Efficient noise-tolerant learning from statistical queries," *J. ACM*, vol. 45, no. 6, pp. 983–1006, Nov. 1998, ISSN: 0004-5411. DOI: 10.1145/293347.293351. [Online]. Available: https://doi.org/10.1145/293347.293351.

[13] L. Reyzin, "Statistical queries and statistical algorithms: Foundations and applications," 2020. arXiv: 2004.00557 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2004.00557.

[14] E. L. Aleixo, J. G. Colonna, M. Cristo, and E. Fernandes, "Catastrophic forgetting in deep learning: A comprehensive taxonomy," 2023. arXiv: 2312.10549 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2312.10549.

[15] N. Aldaghri, H. Mahdavifar, and A. Beirami, "Coded machine unlearning," *IEEE Access*, vol. 9, pp. 88 137–88 150, 2021. DOI: 10.1109/ACCESS.2021.3090019.

[16] S. Schelter, S. Grafberger, and T. Dunning, "Hedgecut: Maintaining randomised trees for low-latency machine unlearning," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD '21, New York, NY, USA: Association for Computing Machinery, 2021, pp. 1545–1557, ISBN: 9781450383431. DOI: 10.1145/3448016.3457239. [Online]. Available: https://doi.org/10.1145/3448016.3457239.

[17] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, Mar. 2006. DOI: 10.1007/s10994-006-6226-1.

[18] J. Brophy and D. Lowd, "Machine unlearning for random forests," 2021. arXiv: 2009.05567 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2009.05567.

[19] H. Lin, J. W. Chung, Y. Lao, and W. Zhao, "Machine unlearning in gradient boosting decision trees," in *KDD*, 2023, pp. 1374–1383. [Online]. Available: https://doi.org/10.1145/3580305.3599420.

[20] X. Zhu and A. Goldberg, *Introduction to semi-supervised learning*. SPRINGER, 2009.

[21] J. E. van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, no. 2, pp. 373–440, Nov. 2019. DOI: 10.1007/s10994-019-05855-6.

[22] I. D. Mienye and N. Jere, "A survey of decision trees: Concepts, algorithms, and applications," *IEEE Access*, vol. 12, pp. 86 716–86 727, 2024. DOI: 10.1109/ACCESS.2024.3416838.

[23] A. Zollanvari, *Machine learning with Python : theory and implementation*, eng. Cham: Springer, 2023 - 2023, ISBN: 9783031333415.

[24] J. Levatić, M. Ceci, D. Kocev, and S. Džeroski, "Semi-supervised classification trees," *Journal of Intelligent Information Systems*, vol. 49, no. 3, pp. 461–486, Mar. 2017. DOI: 10.1007/s10844-017-0457-4.

[25] L. Breiman, "Random forest," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/a:1010933404324.

[26] J. Rodriguez, L. Kuncheva, and C. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1619–1630, 2006. DOI: 10.1109/TPAMI.2006.211.

[27] O. Roesler, *"EEG Eye State"*, UCI Machine Learning Repository, 2013. DOI: 10.24432/C57G7J.

[28] OpenML. [Online]. Available: https://www.openml.org/.

[29] T. Lo, *"Loan Approval Classification Dataset"*, 2019. [Online]. Available: https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data.

[30] B. Becker and R. Kohavi, *"Adult"*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C5XW20, 1996.

[31] D. Chicco and G. Jurman, "Survival prediction of patients with sepsis from age, sex, and septic episode number alone," *Scientific Reports*, vol. 10, no. 1, Oct. 2020. DOI: 10.1038/s41598-020-73558-3.

[32] R. Bhatt and A. Dhall, *"Skin Segmentation"*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C5T30C, 2009.

[33] A. Ustad, A. Logacjov, S. Ø. Trollebø, *et al.*, "Validation of an activity type recognition model classifying daily physical behavior in older adults: The har70+ model," *Sensors*, vol. 23, no. 5, p. 2368, Feb. 2023. DOI: 10.3390/s23052368.

[34] A. Logacjov, K. Bach, A. Kongsvold, H. B. Bårdstu, and P. J. Mork, "Harth: A human activity recognition dataset for machine learning," *Sensors*, vol. 21, no. 23, 2021, ISSN: 1424-8220. DOI: 10.3390/s21237853. [Online]. Available: https://www.mdpi.com/1424-8220/21/23/7853.

[35] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 1, pp. 5–22, 2024. DOI: 10.1109/TNNLS.2022.3182979.

[36] G. Wang, X. Xian, J. Srinivasa, *et al.*, "Demystifying poisoning backdoor attacks from a statistical perspective," 2023. arXiv: 2310.10780 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2310.10780.

[37] T. Dao, C. C. Le, K. D. Doan, and K.-S. Wong, "Towards clean-label backdoor attacks in the physical world," 2024. arXiv: 2407.19203 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2407.19203.

[38] G. Severi, J. Meyer, S. Coull, and A. Oprea, "Explanation-Guided backdoor poisoning attacks against malware classifiers," in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Aug. 2021, pp. 1487–1504, ISBN: 978-1-939133-24-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/severi.

[39] Q. Zhang, L. Xu, and B. Xu, "Python meets jit compilers: A simple implementation and a comparative evaluation," *Software: Practice and Experience*, vol. 54, no. 2, pp. 225–256, Sep. 2023. DOI: 10.1002/spe.3267.

[40] S. Ramesh, B. N. Sukanth, S. S. Jaswanth, V. Sharma, and M. Belwal, "Thrive-jit: Dynamic just-in-time compilation for efficient execution of arithmetic expressions," in *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 2024, pp. 1–9. DOI: 10.1109/ICCCNT61001.2024.10725306.