PUMA: Performance Unchanged Model Augmentation for Training Data Removal

Ga Wu, Masoud Hashemi, Christopher Srinivasa

Borealis AI

{ga.wu, masoud.hashemi, christopher.srinivasa}@borealisai.com

Abstract

Preserving the performance of a trained model while removing unique characteristics of marked training data points is challenging. Recent research usually suggests retraining a model from scratch with remaining training data or refining the model by reverting the model optimization on the marked data points. Unfortunately, aside from their computational inefficiency, those approaches inevitably hurt the resulting model's generalization ability since they remove not only unique characteristics but also discard shared (and possibly contributive) information. To address the performance degradation problem, this paper presents a novel approach called Performance Unchanged Model Augmentation (PUMA). The proposed PUMA framework explicitly models the influence of each training data point on the model's generalization ability with respect to various performance criteria. It then complements the negative impact of removing marked data by reweighting the remaining data optimally. To demonstrate the effectiveness of the PUMA framework, we compared it with multiple state-of-the-art data removal techniques in the experiments, where we show the PUMA can effectively and efficiently remove the unique characteristics of marked training data without retraining the model that can 1) fool a membership attack, and 2) resist performance degradation. In addition, as PUMA estimates the data importance during its operation, we show it could serve to debug mislabelled data points more efficiently than existing approaches.

Introduction

As many countries and territories become increasingly concerned with personal data protection, the corresponding protection regulations[1] entitle individuals to revoke their authorization of using their data for data analysis and machine learning (ML) model training. While retraining ML models by removing marked data points is a feasible solution, frequent data removal requests inevitably put enormous computational pressure on the infrastructures responsible for real-time ML services. Furthermore, cumulative data loss results in quick performance degradation. Hence, effectively eliminating data's unique characteristics while preserving model performance is a critical and challenging research question.

[1]CCPA in California, GDPR in Europe, PIPEDA in Canada, LGPD in Brazil, and NDBS in Australia.

In the literature, a few initial works attempted to address the data removal challenge. For example, (Ginart et al. 2019) devised a general notion of *removal efficiency* and proposed two model-specific data removal algorithms (for k-means clustering models). Similarly, (Guo et al. 2020) introduced a notion of *Certified Removal* and verified the effectiveness of their data removal approach on linear classifiers. However, those methods usually focus on specific ML algorithms and are hard to generalize to deep neural networks that dominate the latest ML research and applications. (Bourtoule et al. 2019), alternatively, proposed a data removal-friendly model by ensembling multiple ML models trained on disjoint data partitions. As such, the data removal operation would only involve a sub-model. (Graves, Nagisetty, and Ganesh 2020) proposed a more generalized single-model solution by explicitly estimating the contribution (gradients) of each training data point as an additive function. Unfortunately, such approaches require high costs; maintaining many sub-models and tracking the model training process are barely feasible for real-world applications. In addition, existing data removal works merely pay attention to the performance degradation problem when removing marked data points. While (Ginart et al. 2019)'s criterion includes a constraint such as performance of the resulting model should not be worse than that of a model trained from scratch with remaining data, it does not intend to preserve the performance of the original model.

In this paper, we propose a novel approach, Performance Unchanged Model Augmentation (PUMA), to efficiently erase the unique characteristics of marked data points from a trained model without causing performance degradation. In particular, the proposed PUMA framework explicitly models the influence of each training data point on the model with respect to various performance criteria (that are not necessarily the model training objectives). It then complements the negative impact of removing marked data by reweighting the remaining data points sparsely and optimally through a constrained optimization. Consequently, PUMA can preserve model performance by linearly patching the original model via reweighting operation while eliminating unique characteristics of marked data points. In the experiments, we compare PUMA with existing data removal approaches and show that PUMA has two desired properties: 1) It can successfully fool a membership attack (Shokri et al. 2017), 2) It can resist performance degradation.

Preliminary and Related Works

Before proceeding, we review existing related data removal approaches which inspired this work. We also briefly describe the influence function to facilitate our description in the main content. Finally, we list several information leaking attack approaches that can be used to test the effectiveness of data removal in the existing literature.

Data Removal Approaches

Removing training data from models has a long research history that can be tracked back to the era of support vector machines. (Cauwenberghs and Poggio 2000) proposed a decremented unlearning approach, called Leave-One-Out (LOO), to gradually remove marked training data points from trained SVM model. By examining the margin of the data points, LOO could significantly reduce the computational effort of data removal. Later, (Karasuyama and Takeuchi 2009) extended the decremental unlearning approach to support simultaneous addition and/or removal of multiple data points through multi-parametric programming. Following the same line of research, (Tsai, Lin, and Lin 2014) proposed a warm-up based unlearning approach that is effective on multiple linear machine learning models. Lastly, (Ginart et al. 2019) payed attention to unsupervised learning tasks where it presented two model-specific data removal algorithms for k-means clustering models.

Recent research (Graves, Nagisetty, and Ganesh 2020) stated that the previously mentioned approaches are not suitable to work on deep network models where the contribution of individual training data points are intractable to compute exactly and analytically. To mitigate the computational cost of retraining a new model from scratch, (Bourtoule et al. 2019) suggested training multiple models on disjoint data partitions so that retraining is limited to small groups of sub-models. Alternatively, (Graves, Nagisetty, and Ganesh 2020) presented *Amnesiac training* which tracks contribution of each training batch (a set of data points) during the model training. When a batch is marked as to be removed, the operation is simply a subtraction between model parameters and data contribution.

While the existing approaches show remarkable achievement on improving efficiency of removing data points from a trained model, we note that they underestimated two critical criteria of data removal tasks: 1) The data removal approach should maintain model stability and protect against performance degradation. 2) The data removal approach should minimize the overall computational cost instead of only looking at the cost of the data removal operation. More specifically, training multiple models or tracking gradients of every training epoch is undesired in practice. All of the above observations motivated our work on proposing Performance Unchanged Model Augmentation (PUMA) in this paper.

Influence Function for Prediction Explanation

An influence function is a limit equation which estimates the prediction changes of a model when its inputs are perturbed. In statistics, the influence function is similar to the Gâteaux derivative, but it can exist even when the Gâteaux derivative does not exist for a particular model.

Recently, the influence function was used to explain the prediction of complex machine learning models as it can reveal the impact of training data point $(\mathbf{x}_k, y_k)$ on the test example $(\mathbf{x}_j, y_j)$'s predictions (Koh and Liang 2017) such that

$$\begin{aligned}\mathcal{I}_{up,loss}\left((\mathbf{x}_k, y_k), (\mathbf{x}_j, y_j)\right) &\overset{\text{def}}{=} \frac{d\mathcal{L}(\mathbf{x}_j, y_j, \theta)}{d\epsilon}\Big|_{\epsilon=0} \\ &= -\nabla_\theta \mathcal{L}(\mathbf{x}_j, y_j, \theta)\left(\frac{1}{m}\sum_{i=1}^{m}\nabla^2_\theta \mathcal{L}(\mathbf{x}_i, y_i, \theta)\right)^{-1}\nabla_\theta \mathcal{L}(\mathbf{x}_k, y_k, \theta),\end{aligned} \tag{1}$$

where $\mathcal{L}$ denotes the loss function for the individual data point, and ϵ denotes the degree of perturbation on the data k. By computing Equation 1 for all training data points k, we can summarize a training data importance rank for a particular test sample j.

Naturally, if we can explain the model prediction based on its training data points, we can also refine the model prediction by perturbing those data points. Based on this idea, (Guo et al. 2020) proposed a data removal approach that leverages the Newton method and influence function. However, their solution is defined for a linear model, making it hard to verify its performance on complex models.

In this work, we will also leverage the influence function. The critical difference between our work and (Guo et al. 2020) is two-folds: First, our objective is to let the modified model preserve the original model's performance after data removal rather than passively monitoring whether the modified model can produce near identical predictions against a model trained on the remaining data from scratch. When a huge number of data points are requested to remove, the difference between these two objectives is significant; training new model from scratch with insufficient data points may not reach a desirable performance. Second, the proposed approach modifies all trainable parameters of the model while (Guo et al. 2020) only adjusts the linear decision making layer which does not eliminate unique characteristics of the removed data points (since the representations are learned with the knowledge of the removed data points).

Data Privacy Protection and Membership Attacks

In terms of evaluating the effectiveness of data removal approaches, previous research (Graves, Nagisetty, and Ganesh 2020) suggested leveraging information leaking attacks (Homer et al. 2008; Dwork et al. 2015; Fredrikson, Jha, and Ristenpart 2015a; Yeom et al. 2018) to check if the data characteristics are indeed removed from a trained model. Specifically, it is suggested that the *membership attack* (Homer et al. 2008) could reveal whether a particular data point is present in training a model, which is an ideal reference to see the difference of attacks before and after the data removal operation. In the literature, there are various membership attack algorithms (Shokri et al. 2017; Nasr, Shokri, and Houmansadr 2018; Yeom et al. 2018) since the concept was introduced by (Homer et al. 2008).

In this paper, we will follow the track of previous works and conduct membership attack experiments to show the effectiveness of our model in the experiments.

Performance Unchanged Model Augmentation

Given a machine learning model $f_{\theta_{org}}$ learned on training data set D_{tn}, we aim to remove the unique characteristics of marked data points $D_{mk} \subset D_{tn}$ from the model by updating model parameters $\theta_{org} \rightarrow \theta_{mod}$ without seriously hurting its prediction performance with respect to various performance criteria $\mathcal{C}$ (or $\mathcal{L}_c$ for an individual sample) such that

$$\Big|\underbrace{\frac{1}{|D_{tn}|}\sum_{i=1}^{|D_{tn}|}\mathcal{L}_c(\mathbf{x}_i, y_i, \theta_{mod})}_{\mathcal{C}(\theta_{mod})} - \underbrace{\frac{1}{|D_{tn}|}\sum_{i=1}^{|D_{tn}|}\mathcal{L}_c(\mathbf{x}_i, y_i, \theta_{org})}_{\mathcal{C}(\theta_{org})}\Big| \leq \delta, \tag{2}$$

where δ is a small change in performance. In particular, we are interested in preserving overall performance rather than being concerned with a shift in an individual prediction.

Influence of Training Data

To tackle the data removal task defined above, we first need to reveal the underlining causal relation between training data perturbation and model performance variation. Specifically, in this section, we clarify two aspects of this connection: 1) How the training data changes would impact model parameters, and 2) How the parameter changes would impact the model performance with respect to specific criteria $\mathcal{C}$.

Parameter as Linear Function of Data Contributions We start by analyzing how perturbing the training dataset would impact the model parameter changes via the influence function.

Let us assume the model parameter θ_{org} is the optimal solution of the (original) training objective $\mathcal{J}_{org}$

$$\theta_{org} = \underset{\theta}{\operatorname{argmin}}\ \mathcal{J}_{org}(\theta) = \underset{\theta}{\operatorname{argmin}}\frac{1}{|D_{tn}|}\sum_{i=1}^{|D_{tn}|}\mathcal{L}_t(\mathbf{x}_i, y_i, \theta) \tag{3}$$

and θ_{mod} is the optimal solution of a modified objective $\mathcal{J}_{mod}$

$$\theta_{mod} = \underset{\theta}{\operatorname{argmin}}\ \mathcal{J}_{mod}(\theta) = \underset{\theta}{\operatorname{argmin}}\ \underbrace{\frac{1}{|D_{tn}|}\sum_{i=1}^{|D_{tn}|}\mathcal{L}_t(\mathbf{x}_i, y_i, \theta)}_{\mathcal{J}_{org}(\theta)} + \underbrace{\frac{1}{|D_{up}|}\sum_{j=1}^{|D_{up}|}\lambda_j\mathcal{L}_t(\mathbf{x}_j, y_j, \theta)}_{\mathcal{J}_{add}(\theta)} \tag{4}$$

that optimizes an additional weighted objective $\mathcal{J}_{add}$ on a subset of training data points $D_{up} \subseteq D_{tn}$, where $\mathcal{L}_t$ denotes individual prediction loss[2] and $\boldsymbol{\lambda} \in \mathbb{R}^{|D_{up}|}$ denotes the weight vector of upweighted data points.

When the values of weights $\boldsymbol{\lambda}$ are negligibly small, the derivative of the modified objective $\mathcal{J}_{mod}$ with respect to its optimal parameters θ_{mod} could be Taylor expanded at the local anchor θ_{org} such that

$$\begin{aligned}\underbrace{\nabla\mathcal{J}_{mod}(\theta_{mod})}_{\approx 0} &\approx \nabla\mathcal{J}_{mod}(\theta_{org}) + \nabla^2\mathcal{J}_{mod}(\theta_{org})(\theta_{mod} - \theta_{org}) \\ &\approx \underbrace{\nabla\mathcal{J}_{org}(\theta_{org})}_{\approx 0} + \nabla\mathcal{J}_{add}(\theta_{org}) + \nabla^2\mathcal{J}_{mod}(\theta_{org})(\theta_{mod} - \theta_{org}).\end{aligned} \tag{5}$$

Since the both θ_{mod} and θ_{org} are optimal solutions with respect to their corresponding objective functions $\nabla\mathcal{J}_{mod}(\theta)$ and $\nabla\mathcal{J}_{org}(\theta)$ (whose derivatives are 0s), the Equation 5 yields a difference between the two optimal solution θ_{mod} and θ_{org} such that

$$\theta_{mod} - \theta_{org} \overset{\text{def}}{=} -\left(\nabla^2\mathcal{J}_{org}(\theta_{org})\right)^{-1}\nabla\mathcal{J}_{add}(\theta_{org}), \tag{6}$$

where we relaxed the Hessian matrix $\nabla^2\mathcal{J}_{mod}(\theta_{org})$ to $\nabla^2\mathcal{J}_{org}(\theta_{org})$. There are multiple justifications for such relaxation. First, since the $\boldsymbol{\lambda}$s are set to be small values, such a setting makes the difference of these second order derivatives insignificant. Second, in practice, computing the Hessian matrix (or Hessian Vector Product described later) is usually an iterative and stochastic process which introduces larger noise than the relaxation we introduced here. It is worth to mention that the expression in Equation 6 aligns with previous influence function work (Koh and Liang 2017) when $\boldsymbol{\lambda}$ is restricted as a one-hot vector (that only upweights a single data point). In our implementation, we compute HVP approximation in the same way as described in (Koh and Liang 2017).

By expanding the derivative of the additive perturbation term $\nabla\mathcal{J}_{add}(\theta_{org})$, we can convert the Equation 6 to a linear function of the perturbation weight $\boldsymbol{\lambda}$ as follows:

$$\theta_{mod} - \theta_{org} = -\sum_{j=1}^{|D_{up}|}\lambda_j\left(\nabla^2\mathcal{J}_{org}(\theta_{org})\right)^{-1}\nabla\mathcal{L}_t(\mathbf{x}_j, y_j, \theta_{org}). \tag{7}$$

Indeed, with trained model whose parameter θ_{org} is fixed, both the Hessian matrix $\nabla^2\mathcal{J}_{org}(\theta_{org})$ and gradient vector $\nabla\mathcal{L}(\mathbf{x}_j, y_j, \theta_{org})$ are constant for the fixed set of upweighted data points D_{up}.

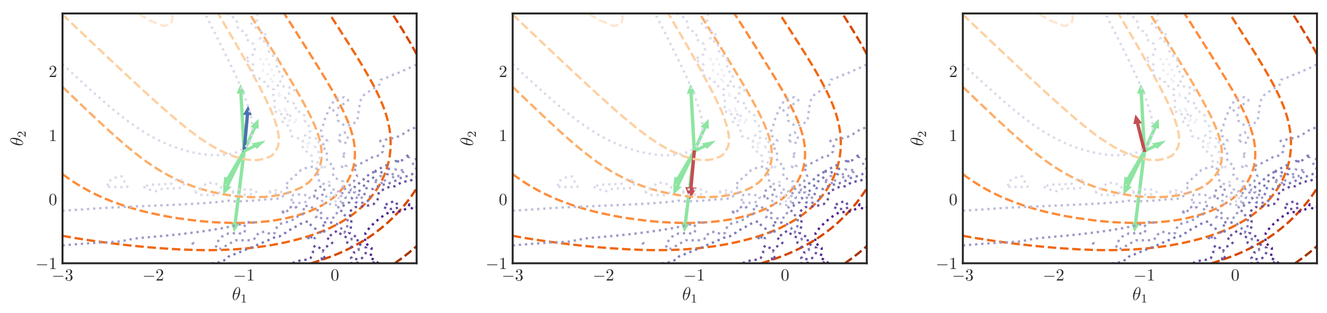
Performance Gap as Taylor Approximation of Parameter Changes When the difference between two sets of parameters is reasonably small, the performance gap between the two corresponding models could be approximated through Taylor expansion such that

$$\begin{aligned}\mathcal{C}(\theta_{mod}) - \mathcal{C}(\theta_{org}) &= \nabla\mathcal{C}(\theta_{org})(\theta_{mod} - \theta_{org}) + \epsilon \\ &\approx -\sum_{j=1}^{|D_{up}|}\lambda_j\nabla\mathcal{C}(\theta_{org})\left(\nabla^2\mathcal{J}_{org}(\theta_{org})\right)^{-1}\nabla\mathcal{L}(\mathbf{x}_j, y_j, \theta_{org}),\end{aligned} \tag{8}$$

which is a linear function of the additive data perturbation $\boldsymbol{\lambda}$, where ϵ represents the higher order Taylor expansion that is exponentially smaller than the first term. Intuitively, term

$$\psi(\mathbf{x}_j, y_j) = \underbrace{\nabla\mathcal{C}(\theta_{org})\left(\nabla^2\mathcal{J}_{org}(\theta_{org})\right)^{-1}}_{\text{Hessian Vector Product (HVP)}}\nabla\mathcal{L}(\mathbf{x}_j, y_j, \theta_{org}) \tag{9}$$

[2]Training loss $\mathcal{L}_t$ is not necessarily identical to the performance criterion loss $\mathcal{L}_c$ defined in Equation 2.



(a) Before Removal (b) Naive Influence Removal (c) PUMA

Figure 1: **Projection Direction Comparison between Naive Influence Removal and PUMA.** (a) The projection direction of each data point (green arrow as shown Equation 12). Blue arrow shows the one marked to remove. (b) The overall projection direction (red arrow) is toward high loss area after naive data contribution removal. (c) The overall projection direction (red arrow) is toward low loss area after PUMA data removal. Orange contour plot shows the loss surface of training objective $\mathcal{J}$. Purple contour plot shows the loss surface of performance criterion $\mathcal{C}$. For both contour plots, lighter color shows lower loss.

is a scalar that serves as the individual contribution score of data $(\mathbf{x}_j, y_j)$ to the performance degradation. By adjusting the weights $\boldsymbol{\lambda}$, one can control the performance gap effortlessly. Hence, at this point, we established the causal relation between data perturbation and model performance changes.

Performance Preserved Data Removal through Gradient Re-weighting

By combining Equation 2 and Equation 8, we note they form an implicit constraint on the data up-scaling factors $\boldsymbol{\lambda}$ such that any changes on a subset factor $\lambda_\mathbf{j}$ would encourage the changes of remaining $\boldsymbol{\lambda}_{/\mathbf{j}}$ as complement to maintain the performance gap smaller than δ.

Based the above notion, we describe how we remove the influence of some marked data points $D_{mk} \subseteq D_{up}$ from a target model $f_{\theta_{org}}$ without hurting the model performance.

According to the Equation 4, removing the contribution of a marked data point $(\mathbf{x}_k, y_k)$ is equivalent to setting its perturbation factor λ_k to -1. Correspondingly, to maintain the model performance while removing data points D_{mk}, we propose optimizing the assignment of the perturbation factor $\boldsymbol{\lambda}$ for the remaining training data points (or randomly sampled subset $D_{up\backslash mk}$) to complement model criterion degradation. Concretely, we propose solving the following linear optimization task

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmin}} \left\| \sum_{j \notin D_{mk}}^{|D_{up}|} \lambda_j \psi(\mathbf{x}_j, y_j) - \sum_{k=1}^{|D_{mk}|} \psi(\mathbf{x}_k, y_k) \right\|^2 + \Omega(\boldsymbol{\lambda}), \tag{10}$$

where Ω denotes the regularization term which encourages both sparsity (l_1 norm) and small changes of $\boldsymbol{\lambda}$ (l_2 norm). In terms of computational efficiency, since the $\psi(\mathbf{x}, y)$s are scalar values, the optimization is simple convex optimization. While estimating individual contribution $\psi(\mathbf{x}_j, y_j)$ looks expensive, the estimation is no more than a dot product between individual gradient and pre-cached Hessian Vector Product (HVP) term.

With the optimized contribution factor $\boldsymbol{\lambda}^*$, we can then update the model parameters by a simple patching such that

$$\theta_{mod} = \theta_{org} + \eta \left[\sum_{k=1}^{|D_{mk}|} \phi(\mathbf{x}_k, y_k) - \sum_{j \notin D_{mk}}^{|D_{up}|} \lambda_j^* \phi(\mathbf{x}_j, y_j) \right], \tag{11}$$

where the individual projection of each data point is

$$\phi(\mathbf{x}, y) = \left(\nabla^2 \mathcal{J}_{org}(\theta_{org})\right)^{-1} \nabla \mathcal{L}(\mathbf{x}, y, \theta_{org}) \tag{12}$$

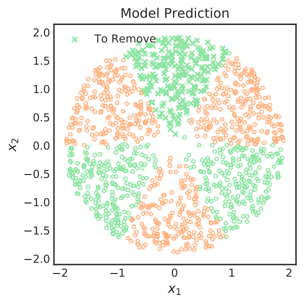
and projection rate $\eta \ll 1$ is a hyper-parameter which keeps patching effective while holding our previous assumptions such that data upweighting is reasonably small.

Figure 1 shows a simple example of PUMA data removal. When a data point is marked for removal (blue arrow), PUMA optimizes Equation 10 and applies the optimal factor $\boldsymbol{\lambda}$ to the projection formula (Equation 12) to adjust model parameters such that model performance with respect to the performance criterion (purple contour) is preserved. In contrast, if we naively remove the local influence of the marked data point, the model would result in performance degradation. In this particular example, performance criterion is measured through Expected Calibration Error (ECE) (Guo et al. 2017). The example model is a linear model with two parameters trained on a binary classification task.

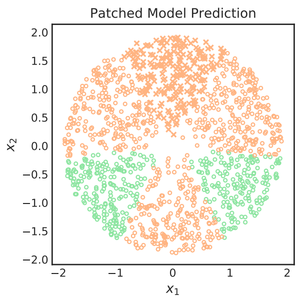
Experiments and Evaluations

In this section, we conduct various experiments to answer the following research questions:

- **RQ1:** Is the proposed approach able to preserve model performance while removing data points?
- **RQ2:** Is the removal successful in terms of causing membership attack failure?
- **RQ3:** How efficient is the proposed approach compared to other state-of-the-art candidates?
- **RQ4:** How sensitive is PUMA with respect to its hyper-parameters?
- **RQ5:** Can the proposed approach conduct mislabeling debugging as it estimates the influence of training data point?



(a) Before Removal



(b) After Removal

Figure 2: Removing the training points marked by crosses from the model. As demonstrated in the right plot, PUMA successfully removed the information of all marked points. 'x' in the plot shows the data intended to remove. Colors show the class labels.

Experimental Settings

Candidate Data Removal Algorithms In data removal experiments, we compare PUMA against the following state-of-the-art data removal approaches.

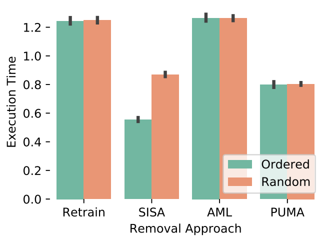
- **Retrain Model:** Retrain model from scratch with remaining data points after picking out marked data points.
- **Retrain Sub-model:** Retrain sub-model that is trained on marked data points. This is also called Sharded, Isolated, Sliced, and Aggregated training (SISA).
- **Amnesiac Machine Learning:** Track gradient information of each training batch during training phase. Subtract the gradients when the batch is marked for removal.

Mislabelling Debugging Algorithms In mislabelled data debugging experiments, we compare PUMA against the following well-known debugging approaches including Influence Function (Koh and Liang 2017), Representor Point Selection (Yeh et al. 2018), and Data Sharply Value (Ghorbani and Zou 2019).

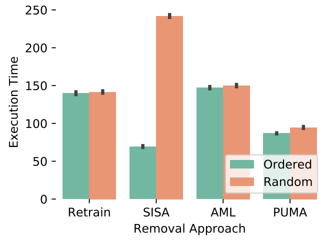
Datasets We conducted our experiments on two synthetic datasets, two tabular datasets from UCI data group (Dua and Graff 2017), and the MNIST dataset (LeCun and Cortes 2010). Full description of the data used in this paper is given in Appendix A.

Dessert: Preliminary Data Removal Check

Before starting quantitative evaluation, we first run a preliminary check on a simple binary classification task to show the effect of PUMA data removal. Specifically, we first train a classifier on a synthetic dataset that contains three observation clusters for each class as shown in Figure 2 (a). The trained classifier is a perfect estimator of data distribution (with 100% prediction accuracy). We then mark all data in one cluster for removal (denoted by 'x' in the plots). Intuitively, if the marked data points are never used for training the classifier, we can imagine that their predictions should align with the predictions of data points surrounding them. Indeed, the model obtained after the PUMA data removal operation reflects our intuition as shown in Figure 2 (b), where all removed data points are now predicted as members of the orange class.



(a) UCI German Credit



(b) MNIST

Figure 3: Execution time comparison among the data removal approaches. Statistics come from 50 times run, and error bar shows the standard deviation. Lower is better.

Effectiveness of Preserving Model Performance

In this section, we quantitatively evaluate how the data removal approaches preserve model performance after data removal. In particular, we gradually remove training data points with percentages $[20\%, 40\%, 60\%, 80\%]$ and aim to show the performance degradation after data removal. To simplify the experimental setting, here we assume the training objective $\mathcal{J}$ and performance criterion $\mathcal{C}$ are identical (both of them are cross entropy loss of prediction). Considering that both Amnesiac ML and SISA models may show better performance when the data marked to be removed belong to same training batch, we conduct experiments in two scenarios. In the first scenario (Ordered), we intentionally group all data points marked to be removed into small set of training batches such that the removal operation would not impact other training batches (and sub-models for SISA). In the second scenario (Random), we simulate a more realistic setting where removal may apply to any data points irrespective of training batches.

Table 1 shows performance preservation comparison between our proposed approach (PUMA) and various baselines. In the table, we make the following observations:

- Among all candidate data removal approaches, PUMA shows the best performance preservation ability. And, in some cases, the model obtained after the PUMA operation even shows better performance than the original model.
- Amnesiac ML often completely destroys the model with its data removal operation when the removal is applied to more than 20% of training data. This observation aligns with the original results described in the Amnesiac ML paper (Graves, Nagisetty, and Ganesh 2020) where refined training is required after the removal operation.
- While Amnesiac ML and SISA show reasonably satisfactory performance preservation ability in one of the two scenarios, they tend to fail in another scenario. Amnesiac ML fails in the setting where data may be required to be removed from random batches. In contrast, SISA does not perform well when the number of sub-models is reduced, as a consequence of removing all training data points of the sub-models.

Table 1: Comparison of Model Performance Preservation among Candidate Removal Approaches. Value shows accuracy. Higher is better after data removal. We omit to present statistics in the main paper for clearness. The full table with statistics is presented in Appendix D for further reference.

Data Group	Dataset	Ordered						Random					
		Original	Approach	20%	40%	60%	80%	Original	Approach	20%	40%	60%	80%
Synthetic	Radial	95.04	Retrain Model	**93.64**	**91.60**	**84.15**	66.24	95.89	Retrain Model	**93.97**	**90.94**	**82.58**	66.51
		80.88	SISA	67.35	63.57	61.93	51.91	75.62	SISA	64.71	64.35	54.80	54.77
		95.04	Amnesiac ML	56.38	54.75	53.53	50.54	95.88	Amnesiac ML	49.08	48.95	48.95	48.95
		94.97	PUMA	68.97	69.60	67.99	**70.77**	95.82	PUMA	72.44	73.22	71.82	**76.02**
	Rectangular	62.00	Retrain Model	**61.20**	**60.35**	**55.80**	54.25	65.00	Retrain Model	**64.70**	**64.50**	62.30	58.65
		55.60	SISA	55.90	48.30	30.10	29.55	56.50	SISA	56.50	56.50	56.55	56.90
		62.00	Amnesiac ML	46.60	43.85	43.45	39.15	65.00	Amnesiac ML	35.40	35.40	35.40	35.40
		61.85	PUMA	55.25	56.30	53.85	**61.70**	64.95	PUMA	59.90	62.05	**62.55**	**64.80**
Tabular (UCI)	German	71.52	Retrain Model	**70.56**	70.12	70.11	70.00	75.16	Retrain Model	**74.88**	**73.24**	72.47	70.00
		70.00	SISA	70.00	70.00	68.96	66.16	70.00	SISA	70.00	70.00	70.00	70.00
		71.52	Amnesiac ML	68.52	64.40	66.24	64.03	75.16	Amnesiac ML	36.24	36.28	35.72	35.72
		71.47	PUMA	69.08	**70.72**	**70.64**	**70.72**	75.12	PUMA	70.96	**73.24**	**74.44**	**74.28**
	Breast Cancer	96.45	Retrain Model	**96.62**	**96.11**	96.00	94.85	96.00	Retrain Model	**95.82**	**95.54**	**95.65**	95.20
		91.31	SISA	89.20	88.91	80.68	52.62	92.28	SISA	91.60	88.05	88.22	87.88
		96.45	Amnesiac ML	96.05	95.82	95.25	82.28	96.00	Amnesiac ML	35.20	30.51	30.51	30.51
		96.39	PUMA	96.17	95.88	**96.22**	**96.62**	96.00	PUMA	95.08	94.91	95.25	**95.54**
Image	MNIST	97.58	Retrain Model	**97.28**	**96.72**	95.76	93.48	97.99	Retrain Model	**97.72**	97.16	96.60	93.98
		95.89	SISA	95.80	95.67	94.78	89.86	95.66	SISA	93.47	90.63	78.06	59.83
		97.44	Amnesiac ML	9.44	9.84	9.56	9.36	98.06	Amnesiac ML	10.39	10.39	10.39	10.39
		97.60	PUMA	96.70	96.66	**97.17**	**97.16**	97.97	PUMA	97.42	**97.58**	**97.60**	**97.61**

Table 2: Comparison of Membership Attack after Data Removal Operation. Value shows percentage of removed data that is identified as training data. Lower values in the table show better performance of removal.

Data Group	Dataset	Ordered								Random							
		Retrain Model		SISA		Amnesiac ML		PUMA		Retrain Model		SISA		Amnesiac ML		PUMA	
		Before	After	Before	After	Before	After	Before	After	Before	After	Before	After	Before	After	Before	After
Synthetic	Radial	100.00	100.00	100.00	100.00	100.00	**0.00**	100.00	5.31	100.00	52.36	100.00	37.00	100.00	50.00	100.00	**1.18**
	Rectangular	100.00	91.65	83.18	83.18	100.00	**33.33**	100.00	36.66	100.00	67.07	98.50	94.00	100.00	86.20	100.00	**20.00**
Tabular	German	100.00	77.12	100.00	100.00	100.00	0.00	100.00	**3.42**	94.44	84.44	100.00	98.81	94.44	93.33	85.18	**2.22**
	Breast Cancer	100.00	100.00	87.50	87.50	100.00	100.00	100.00	**56.25**	100.00	100.00	90.00	73.75	100.00	87.50	100.00	**71.25**
Image	MNIST	100.00	100.00	100.00	100.00	100.00	100.00	100.00	**0.00**	100.00	100.00	100.00	100.00	100.00	100.00	100.00	**72.00**

Effectiveness of Data Removal

Now, we show how well the proposed approach works in removing the influence of data points from the model. To quantitatively evaluate the performance, we conduct a membership attack on the model after data removal. Ideally, if the influence of a data point is successfully removed, then the membership attack would predict that the given data point does not belong to the training data set. Hence, a lower value for data removal shows better removal effectiveness.

Table 2 shows a comparison of the effectiveness of the data removal approaches. In the table, we observe follows:

- In most cases, PUMA shows better data removal performance compared to the other baseline models. While Amnesiac ML occasionally outperforms PUMA, we realize that it could be due to a complete model degradation, as previously observed in Table 1.
- In multiple experiments, we observed that the data removal operations could not reduce the success rate of membership attack to zero. This is due to the existence of similar training examples to the marked data points that are not marked for removal. Since well-train ML models can generalize well on previously unseen data points, these remaining data points can also fool the membership attack classifier when the prediction confidence is high enough.

Efficiency of Data Removal

As efficiency is the one of most important reason of running the data removal operation, we compare the execution time of different data removal approaches in the previously described experimental settings. Here, we only show the two most representative plots as the general trendy is similar.

Figure 3 shows the execution time comparison on UCI German Credit and MNIST datasets. Specifically:

- PUMA shows the best efficiency compared to the other candidates when the data removal happens to be random (i.e. the more practical scenario).
- SISA's efficiency depends on how many sub-models are involved in retraining. In the ordered data removal setting, SISA shows competitive efficiency. However, when the data removal happens to involve more sub-models, its efficiency is dramatically reduced.
- In general, data removal approaches are more efficient

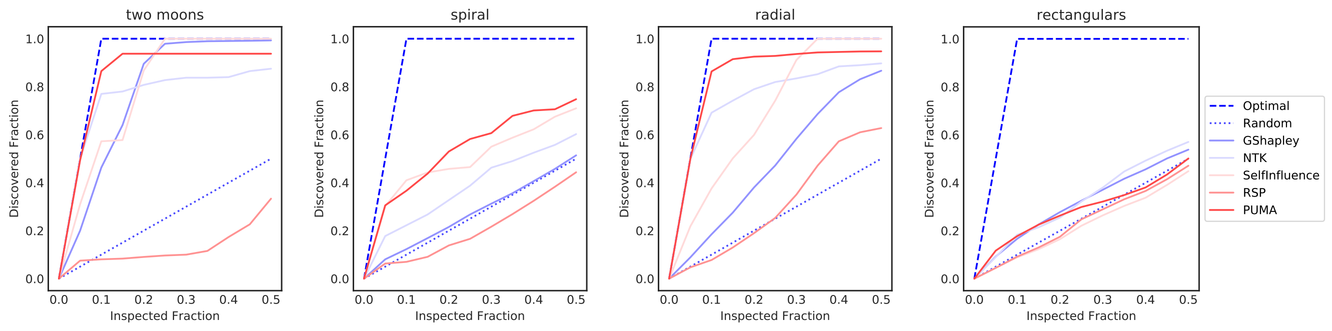
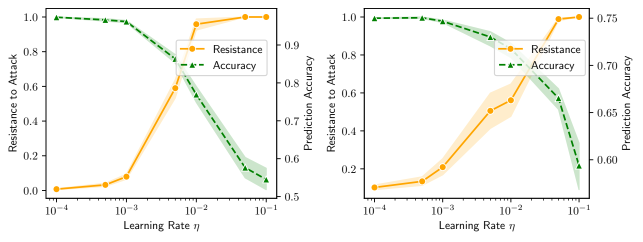


Figure 4: Mislabelling debugging comparison between PUMA and state-of-the-art debugging algorithms. We corrupted datasets by randomly flipping 10% of the data labels. The goal of the candidate approaches is to identify and correct the mislabelled data as early as possible. PUMA shows significant advantage when only 20% of data are processed during debugging.



(a) Synthetic Radial (b) UCI German Credit

Figure 5: Effects of hyper-parameter tuning. Large projection rate η shows better resistant to the membership attack while suffers from severe performance degradation.

than training a model from scratch. However, for the small dataset (UCI-German Credit), there is no significant advantage of using a data removal operation. In particular, the Amnesiac ML approach does not show better efficiency compared to retraining a model from scratch.

Insight of Hyper-parameter Tuning

As introduced in Equation 11, PUMA has one important hyper-parameter η which controls the projection step of parameter augmentation. Indeed, a huge projection step η would seriously violate the Taylor approximation assumption that PUMA approach relies on. Hence, in this experiment, we aim to demonstrate the importance of tuning this hyper-parameter.

Figure 5 shows the trend of tuning η on two representative datasets (UCI German Credit and MNIST (LeCun and Cortes 2010)). Overall, there is a trade-off between the effectiveness of removing data and the ability of preserving model generalization. Keeping the projection rate in the range of $\eta \in [10^{-2}, 10^{-1}]$ often show satisfactory removal performance while maintaining the model's generalization ability.

Corrupted Sample Discovery

As PUMA explicitly states the contribution of individual data points to the performance criterion (see Equation 9), a side functionality of PUMA is to debug mislabelled data in the same fashion as Influence Function (Koh and Liang 2017), Representer Point Selection (Yeh et al. 2018), and Data Shapley (Ghorbani and Zou 2019). We also have included a simplified version of the Influence function by removing the inverse Hessian matrix from the influence function formulation to accelerate the computation, denoted by Neural Tangent Kernel (NTK), due to its similarity to the NTK formulation (Jacot, Gabriel, and Hongler 2018). Figure 4 shows the overall performance of mislabel debugging. In this experiment, we randomly flip the label of 10% of the training data samples and calculate the data values using the aforementioned algorithms. PUMA outperforms other algorithms by discovering more corrupted training data points while reviewing fewer data fractions. Tabel 3 shows the corresponding execution time for the debugging test, where we observe that PUMA is significantly more efficient than the other approaches.

Table 3: Comparison of Running Time (in Seconds). Lower values in the table show better performance to the mislabelled data debugging. We omit the statistic in this table for saving space. Please refer to Appendix D for statistics.

Data	Approach				
	GShapley	NTK	SelfInfluence	RSP	PUMA
Two Moons	90.37	22.65	1562.93	17.14	**7.61**
Spiral	78.47	19.91	1464.01	16.51	**7.44**
Radial	82.99	21.60	1563.53	17.76	**7.76**
Rectangulars	78.04	20.23	1480.12	16.81	**7.29**

Conclusion

This paper presents a novel data removal approach, PUMA, which removes unique characteristics of marked training data points from a trained ML model while preserving the model's performance with respect to certain performance criterion. Compared to existing approaches which require access to the model training process, PUMA shows a significant advantage as it does not restrict how the model is trained. From various experiments, we note PUMA also demonstrates better performance compared to the baseline approaches in multiple aspects, including effectiveness, efficiency and performance preservation ability.

References

Ateniese, G.; Mancini, L. V.; Spognardi, A.; Villani, A.; Vitali, D.; and Felici, G. 2015. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, 10(3): 137–150.

Bourtoule, L.; Chandrasekaran, V.; Choquette-Choo, C. A.; Jia, H.; Travers, A.; Zhang, B.; Lie, D.; and Papernot, N. 2019. Machine Unlearning. *CoRR*, abs/1912.03817.

Cauwenberghs, G.; and Poggio, T. A. 2000. Incremental and Decremental Support Vector Machine Learning. In Leen, T. K.; Dietterich, T. G.; and Tresp, V., eds., *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, 409–415. MIT Press.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository.

Dwork, C.; Smith, A. D.; Steinke, T.; Ullman, J. R.; and Vadhan, S. P. 2015. Robust Traceability from Trace Amounts. In Guruswami, V., ed., *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, 650–669. IEEE Computer Society.

Fredrikson, M.; Jha, S.; and Ristenpart, T. 2015a. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In Ray, I.; Li, N.; and Kruegel, C., eds., *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, 1322–1333. ACM.

Fredrikson, M.; Jha, S.; and Ristenpart, T. 2015b. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 1322–1333.

Ghorbani, A.; and Zou, J. 2019. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, 2242–2251. PMLR.

Ginart, A.; Guan, M. Y.; Valiant, G.; and Zou, J. 2019. Making AI Forget You: Data Deletion in Machine Learning. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 3513–3526.

Graves, L.; Nagisetty, V.; and Ganesh, V. 2020. Amnesiac Machine Learning. *CoRR*, abs/2010.10981.

Guo, C.; Goldstein, T.; Hannun, A. Y.; and van der Maaten, L. 2020. Certified Data Removal from Machine Learning Models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, 3832–3842. PMLR.

Guo, C.; Pleiss, G.; Sun, Y.; and Weinberger, K. Q. 2017. On calibration of modern neural networks. In *International Conference on Machine Learning*, 1321–1330. PMLR.

Hitaj, B.; Ateniese, G.; and Perez-Cruz, F. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 603–618.

Homer, N.; Szelinger, S.; Redman, M.; Duggan, D.; Tembe, W.; Muehling, J.; Pearson, J. V.; Stephan, D. A.; Nelson, S. F.; and Craig, D. W. 2008. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genet*, 4(8): e1000167.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Jacot, A.; Gabriel, F.; and Hongler, C. 2018. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*.

Karasuyama, M.; and Takeuchi, I. 2009. Multiple incremental decremental learning of support vector machines. *Advances in neural information processing systems*, 22: 907–915.

Koh, P. W.; and Liang, P. 2017. Understanding Black-box Predictions via Influence Functions. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, 1885–1894. PMLR.

LeCun, Y.; and Cortes, C. 2010. MNIST handwritten digit database.

Lyu, L.; and Chen, C. 2021. A Novel Attribute Reconstruction Attack in Federated Learning. *arXiv preprint arXiv:2108.06910*.

Nasr, M.; Shokri, R.; and Houmansadr, A. 2018. Comprehensive Privacy Analysis of Deep Learning: Stand-alone and Federated Learning under Passive and Active White-box Inference Attacks. *CoRR*, abs/1812.00910.

Shokri, R.; Stronati, M.; Song, C.; and Shmatikov, V. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, 3–18. IEEE.

Tsai, C.; Lin, C.; and Lin, C. 2014. Incremental and decremental training for linear classification. In Macskassy, S. A.; Perlich, C.; Leskovec, J.; Wang, W.; and Ghani, R., eds., *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, 343–352. ACM.

Yang, F.; Zhong, Z.; Liu, H.; Wang, Z.; Luo, Z.; Li, S.; Sebe, N.; and Satoh, S. 2021. Learning to Attack Real-World Models for Person Re-identification via Virtual-Guided Meta-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 3128–3135.

Yeh, C.-K.; Kim, J. S.; Yen, I. E.; and Ravikumar, P. 2018. Representer point selection for explaining deep neural networks. *arXiv preprint arXiv:1811.09720*.

Yeom, S.; Giacomelli, I.; Fredrikson, M.; and Jha, S. 2018. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, 268–282. IEEE Computer Society.

Appendix A: Detailed Experiment Setup

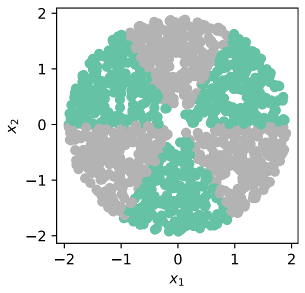
Dataset

The experiments in this paper involves three groups of data sources, including synthetically generated data, UCI data (Dua and Graff 2017), and the MNIST dataset (LeCun and Cortes 2010). Table 4 summarizes the datasets used in this paper.

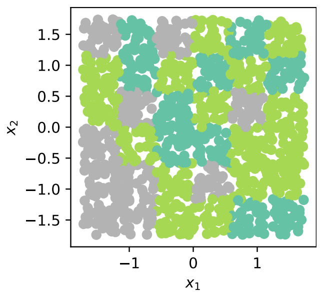
Table 4: Dataset used in the experiments.

Dataset	Size	Num of Features	Num of Labels	Model Architecture
Radial	600	2	2	2-layer FCN
Rectangular	800	2	3	2-layer FCN
Breast Cancer	699	9	2	2-layer FCN
German Credit	1000	20	2	2-layer FCN
MNIST	70000	784	10	DenseNet

Synthetic Data We used multiple synthetic datasets to conduct the proof of concept experiments in the paper as the results are easy to visualize and interpret. Figure 6 shows the synthetic data we produced for our experiments.



(a) Radial



(b) Rectangular

Figure 6: Synthetic data used in our experiments. Radial dataset has 2 classes with 6 clusters, while rectangular dataset has 3 classes with 16 clusters.

UCI datasets The UCI data is a group of classic tabular datasets well suited to justify the quantitative performance of a ML model in the literature. Among the hundreds of datasets, we chose to use German Credit and Breast Cancer dataset in our experiments since both of the datasets are closely related to two important machine learning application fields: finance and medicine.

MNIST We include the MNIST dataset along with the Convolution Network (DenseNet) architecture in our experiments to show that the proposed approach can work on more complex model architectures than simple Fully Connected Networks (FNC). In addition to the quantitative evaluations of the main paper, we also use it to visualize the effects of data removal in Appendix C.

Predictive Models

In the experiments, we train the classifiers with two neural network architectures. Specifically, for the synthetic and the tabular (UCI) data, we train fully connected neural networks that have two hidden layers with dimension of 64 and 32 respectively. For the MNIST dataset, we train a DenseNet model (Huang et al. 2017) to reach a better prediction performance.

The following pytorch code shows the actual architecture we used in the experiments.

```python
class FlexibleFCN(nn.Module):
    def __init__(self, neurons, activation):
        super(FlexibleFCN, self).__init__()
        self.layers = nn.ModuleList()
        for i in range(len(neurons) - 1):
            self.layers.append(torch.nn.Linear(neurons[i],
      neurons[i + 1]))
        if activation == "relu":
            self.activation = torch.nn.ReLU()
        else:
            self.activation = torch.nn.Sigmoid()

    def forward(self, x):
        for i in range(len(self.layers) - 1):
            x = self.layers[i](x)
            x = self.activation(x)
        x = self.layers[-1](x)
        return x
```

Listing 1: Fully Connect Network

```python
class DenseNet(nn.Module):
    def __init__(
        self,
        block,
        nblocks,
        growth_rate=12,
        reduction=0.5,
        num_classes=10,
        channel=3,
        last_pool_size=4,
    ):
        super(DenseNet, self).__init__()
        self.growth_rate = growth_rate

        num_planes = 2 * growth_rate
        self.conv1 = nn.Conv2d(
            channel, num_planes, kernel_size=3,
            padding=1, bias=False
        )

        self.dense1 = self._make_dense_layers(block,
                                              num_planes,
                                              nblocks[0])
        num_planes += nblocks[0] * growth_rate
        out_planes = int(math.floor(num_planes * reduction))
        self.trans1 = Transition(num_planes, out_planes)
        num_planes = out_planes

        self.dense2 = self._make_dense_layers(block,
                                              num_planes,
                                              nblocks[1])
        num_planes += nblocks[1] * growth_rate
        out_planes = int(math.floor(num_planes * reduction))
        self.trans2 = Transition(num_planes, out_planes)
        num_planes = out_planes

        self.dense3 = self._make_dense_layers(block,
                                              num_planes,
                                              nblocks[2])
        num_planes += nblocks[2] * growth_rate
        out_planes = int(math.floor(num_planes * reduction))
        self.trans3 = Transition(num_planes, out_planes)
        num_planes = out_planes

        self.dense4 = self._make_dense_layers(block,
                                              num_planes,
                                              nblocks[3])
        num_planes += nblocks[3] * growth_rate

        self.bn = nn.BatchNorm2d(num_planes)
        self.linear = nn.Linear(num_planes, num_classes)
        self.softmax = torch.nn.Softmax(dim=1)
        self.num_classes = num_classes

        self.last_pool_size = last_pool_size

    def _make_dense_layers(self, block, in_planes, nblock):
        layers = []
        for i in range(nblock):
            layers.append(block(in_planes, self.growth_rate))
            in_planes += self.growth_rate
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.trans1(self.dense1(out))
        out = self.trans2(self.dense2(out))
        out = self.trans3(self.dense3(out))
        out = self.dense4(out)
        out = F.avg_pool2d(F.relu(self.bn(out)),
                           self.last_pool_size)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

Listing 2: CNN DenseNet

Membership Attack Setup

We implemented the attack model based on the description in the original membership attack paper (Shokri et al. 2017). Since, in our experimental setting, the training data is fully observable, we use the training data to train the shadow models. Here, the shadow models hold the same architecture as the predictive models described previously.

We train a membership attack model for each of the datasets using five shadow models, where each shadow model is trained on a subset of data points (less than 10% of training set) randomly sampled from the training set. The number of training epochs for the shadow models is set to 50.

After training the shadow models, a neural network based binary classifier is trained on the output of the shadow models to predict whether a data point has been a member of the training set or not. The architecture of this classifier is a two-hidden layer neural network with latent dimensions of 128 and 64. This architecture aligns with the description in the previous data removal literature (Graves, Nagisetty, and Ganesh 2020). The binary classifier is then used to attack the predictive models before and after data removal to demonstrate the effectiveness of the data removal algorithms.

Data Marking for Removal Experiment

As mentioned in the main paper, when many similar data points exist in a dataset, removing one or a small set of them does not help to demonstrate the data removal performance (via membership attack) since the ML model would generalize to the removed data points with high prediction confidence.

Hence, we mark the data through a clustering based approach. Specifically, we conduct the following steps to mark the data for removal experiments:

1. For each class y in the training set, run a k-means clustering algorithm on the features $\mathbf{X}$.
2. Choose all data points in a cluster from the k-means clusters as the marked data points.
3. Run step 1-2 for all classes.

The following python code shows how we did it in our implementation.

```
@staticmethod
def mark_data(data_loader, label_interested, clustering=True):
    with torch.no_grad():
        tensor_X, tensor_y = data_loader.dataset.tensors
        label_index = torch.where(tensor_y == label_interested)[0].cpu().numpy()
        if not clustering:
            data_index_to_remove = label_index
        else:
            x = tensor_X.cpu().numpy()[label_index]
            kmeans = KMeans(n_clusters=8, random_state=0).fit(
                np.array(x).reshape(len(x), -1)
            )
            values, counts = np.unique(kmeans.labels_, return_counts=True)
            value_pri = values[counts.argsort()[0]]
            data_index_to_remove = label_index[np.where(kmeans.labels_ == value_pri)]
        return data_index_to_remove
```

Listing 3: Mark Data to Remove

How does Amnesiac ML and SISA Handle Single/Batch Data Removal?

Amnesiac ML does not support single data removal operation since it requires storing the gradient information for each point, which is barely possible in practice. Hence, to remove contribution of a single data point, we need to remove the entire training batch containing the point in question.

Indeed, in the Experiment and Evaluation section, we mentioned that we have two experiment scenarios, namely Ordered and Random. In the Ordered scenario, we assume all data points that are marked to be removed belong to the same data batch. In this particular setting, Amnesiac ML could work as it was presented in its original paper (Graves, Nagisetty, and Ganesh 2020). We introduce this scenario for the purpose of fair comparison. For a more realistic scenario (Random), since the marked data points may be spread across many batches, we have no choice but to remove the batches that contain the marked data points. Hence, the updated model could face severe performance degradation, as shown in Table 1.

SISA naturally supports removing single a data point since it retrains the sub-model that has been trained on the marked data point. However, SISA faces a computational challenge when we remove multiple data points requiring to retrain multiple involved sub-models simultaneously. Hence, SISA's efficiency depends on the distribution of marked data points in the sub-model training datasets. This intuition could be justified in our experiment results (Figure 3), where we show SISA's run time varies when we switch scenarios (Random vs. Ordered).

Appendix B: Algorithm Implementation

In this section, we present the implementation details of the PUMA algorithm in the form of pseudo-code.

PUMA Removal Pseudo Code

Algorithm 1 shows the pseudo-code of the PUMA data removal procedure. Here, we use D_{up} instead of $D_{up/mk}$ to represent the reweight data points for simplicity.

Algorithm 1: PUMA Data Characteristics Removal

```
Require: Original model $\theta_{org}$, Training objective $\mathcal{J}_{org}$, Performance criterion $\mathcal{C}$, Data marked to remove $D_{mk}$, Whole training set $D_{tn}$, Learning rate $\eta$
    Optional: Upweight data $D_{up}$
Ensure: $\mathcal{C}$ is differentiable, $D_{mk} \subset D_{tn}$

  $\nabla\mathcal{C} \leftarrow Gradient(\mathcal{C}, D_{tn}, \theta_{org})$
  $\nabla\mathcal{C}\left(\nabla^2\mathcal{J}_{org}\right)^{-1} \leftarrow HVP(\mathcal{J}_{org}, D_{tn}, \theta_{org}, \nabla\mathcal{C})$        ▷ Cache
  $D_{up} \leftarrow Sample(D_{tn}, excluding = D_{mk})$        ▷ Optional

  for $(\mathbf{x}_k, y_k) \in D_{mk}$ do        ▷ Get Data Influence
      $\psi(\mathbf{x}_k, y_k) \leftarrow \nabla\mathcal{C}\left(\nabla^2\mathcal{J}_{org}\right)^{-1}\nabla\mathcal{L}(\mathbf{x}_k, y_k, \theta_{org})$

  for $(\mathbf{x}_j, y_j) \in D_{up}$ do        ▷ Get Data Influence
      $\psi(\mathbf{x}_j, y_j) \leftarrow \underbrace{\nabla\mathcal{C}\left(\nabla^2\mathcal{J}_{org}\right)^{-1}}_{Cached}\nabla\mathcal{L}(\mathbf{x}_j, y_j, \theta_{org})$

  $\boldsymbol{\lambda}^* \leftarrow \underset{\boldsymbol{\lambda}}{\operatorname{argmin}} \left\|\sum_j \lambda_j\psi(\mathbf{x}_j, y_j) - \sum_k \psi(\mathbf{x}_k, y_k)\right\|^2 + \Omega(\boldsymbol{\lambda})$
  ▷ SLSQP Constraint Optimization

  $V_{mk} \leftarrow 0$        ▷ Get Parameter Projection Direction
  for $(\mathbf{x}_k, y_k) \in D_{mk}$ do
      $V_{mk} \leftarrow V_{mk} + \nabla\mathcal{L}(\mathbf{x}_k, y_k, \theta_{org})$
  $\Phi_{mk} \leftarrow HVP(\mathcal{J}_{org}, D_{tn}, \theta_{org}, V_{mk})$

  $V_{up} \leftarrow 0$        ▷ Get Parameter Projection Direction
  for $(\mathbf{x}_j, y_j) \in D_{mk}$ do
      $V_{up} \leftarrow V_{up} + \lambda_j\nabla\mathcal{L}(\mathbf{x}_j, y_j, \theta_{org})$
  $\Phi_{up} \leftarrow HVP(\mathcal{J}_{org}, D_{tn}, \theta_{org}, V_{up})$

  $\theta_{mod} \leftarrow \theta_{org} + \eta\,[\Phi_{mk} - \Phi_{up}]$        ▷ Update Model
```

Overall, there are five fundamental steps in the procedure.

1. Compute and cache the Hessian Vector Product $\nabla\mathcal{C}\left(\nabla^2\mathcal{J}_{org}\right)^{-1}$ for both the training objective $\mathcal{J}$ and performance criterion $\mathcal{C}$ on training data D_{tn}.
2. Estimate the influence value of the data points marked for removal D_{mk} and the reweighted data points D_{up}.
3. Optimize the reweighting weights $\boldsymbol{\lambda}$ with constrained optimization algorithms. E.g. SLSQP or L-BFGS.
4. Estimate the weighted parameter projection directions for both D_{mk} and D_{up}
5. Update the model parameters with learning rate η

While computing the inverse of the Hessian matrix is possible for simple linear models, it is generally infeasible for more complicated setups, since there is no guarantee on the Hessian matrix to be positive definite, as noticed in the previous literature (Koh and Liang 2017). Hence, in our implementation, we directly compute the Hessian Vector Product approximation to avoid the potential numerical issue.

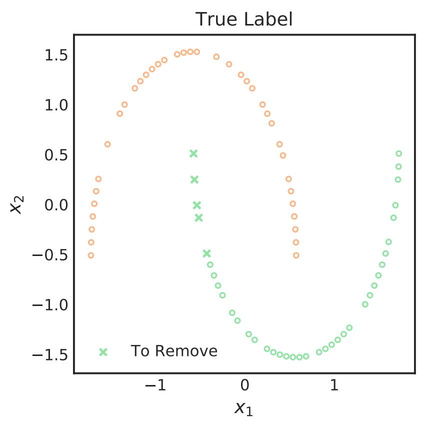
PUMA Data Debugging Pesudo Code

While PUMA was originally not designed for debugging mislabeled data, we note it shows reasonably good performance (with significant advantage in efficiency) compared with the state-of-the-art approaches.

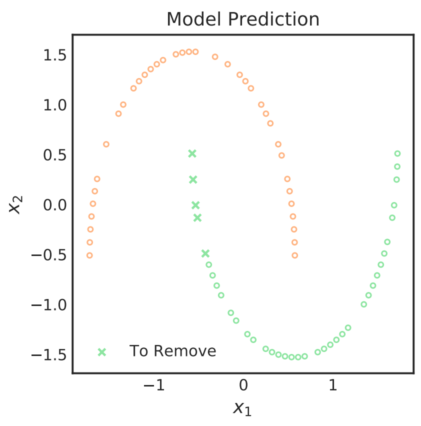
As mentioned in the main paper, since PUMA computes the influence of each data point on the overall model performance (see Equation 9), we can use this information to rank the data points and identify the mislabelled data points that usually have large negative influences. In addition, since the data points close to the decision boundary are often noisy, we filter the candidate data points with this condition to avoid large false positive predictions. Algorithm 2 shows the algorithm for debugging mislabelled data. Compared to the self-influence method (Koh and Liang 2017), our proposed approach does not require computing the Hessian Vector Product (HVP) for each training data. Instead, it caches the HVP for the entire training set and computes individual data point influence through simple gradient estimation and dot-product (with the cached HVP). While using the entire training data's HVP may hurt our approach's debugging performance, we empirically show that the performance degradation is negligible.

Algorithm 2: PUMA Problematic Data Debugging

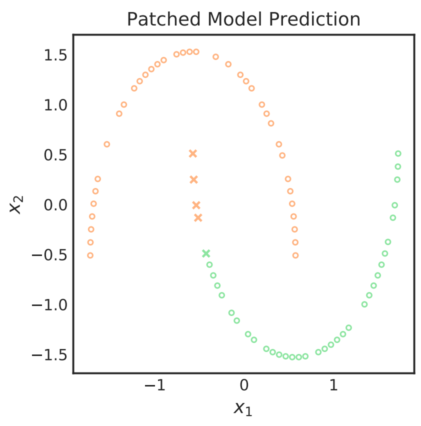
```
Require: Original model $\theta_{org}$, Training objective $\mathcal{J}_{org}$, Performance criterion $\mathcal{C}$, Whole training set $D_{tn}$, Number of problematic data to return $k$
Ensure: $\mathcal{C}$ is differentiable

  $\nabla\mathcal{C} \leftarrow Gradient(\mathcal{C}, D_{tn}, \theta_{org})$
  $\nabla\mathcal{C}\left(\nabla^2\mathcal{J}_{org}\right)^{-1} \leftarrow HVP(\mathcal{J}_{org}, D_{tn}, \theta_{org}, \nabla\mathcal{C})$        ▷ Cache

  $Confidence \leftarrow EmptyList()$
  $Influence \leftarrow EmptyList()$
  for $(\mathbf{x}_i, y_i) \in D_{tn}$ do
      $Confidence[i] \leftarrow p(y_i|\mathbf{x}_i; \theta_{org})$        ▷ Prediction
      $Influence[i] \leftarrow \underbrace{\nabla\mathcal{C}\left(\nabla^2\mathcal{J}_{org}\right)^{-1}}_{Cached}\nabla\mathcal{L}(\mathbf{x}_i, y_i, \theta_{org})$

  $L_{if} \leftarrow argsort(Influence, top = k)$        ▷ Top low influences
  $L_{cf} \leftarrow argsort(Confidence, top = k)$
  $H_{cf} \leftarrow argsort(Confidence, bottom = k)$

  $S \leftarrow L_{if} \setminus (L_{cf} \cup H_{cf})$        ▷ Candidate mislabelled data

  if $mean(Influence, S) > mean(Influence, L_{if} \cap L_{cf})$ then
      $S \leftarrow \emptyset$
  return S
```



(a) Data

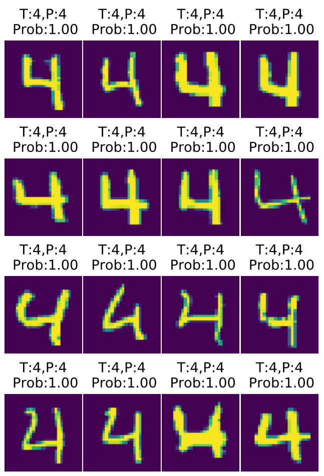


(b) Model prediction before removal

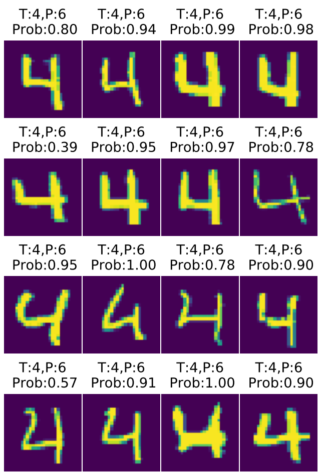


(c) Model prediction after removal

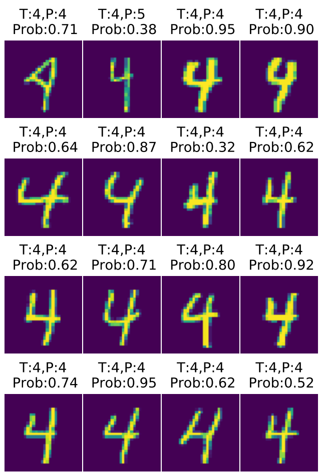
Figure 7: Removing the training points marked by crosses from the model. As demonstrated in the right plot, PUMA successfully removed the information of all marked points. ‘x’ in the plot shows the data intended to remove. Colors show the class labels.



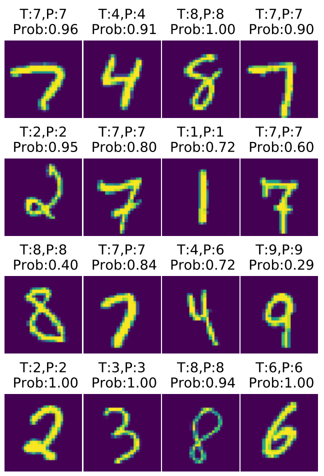
(a) Data marked to remove
(Predictions of original model)



(b) Data marked to remove
(Predictions of augmented model)



(c) Other data in the same class
(Predictions of augmented model)



(d) Randomly sampled data
(Predictions of augmented model)

Figure 8: Removing marked data points that belong to a particular style of digit 4. On top of each digit image, we show the ground truth label (denoted as T), predicted label (denoted as P), and the prediction confidence (denoted as Prob).

Appendix C: More Demo Cases

In this section, we show two demonstrative examples which visualize the data removal effect of PUMA.

Figure 7(a) shows a binary classification task, where data points are distributed as moon shapes. Since there is no noise introduced in the data generation process, the model can make perfect predictions as shown in Figure 7(b). When we remove the data points marked as ‘x’ from the train model (see Figure 7(c)), the top four data points are classified as belonging to the orange class and the bottom one is still classified as the green class. More importantly, the predictions for the other data points remain unchanged. This observation reflects our intuition, where PUMA would leverage remaining data points to stabilize the model’s generalization ability by complementing the loss of the removed data points.

Figure 8 shows a more complex removal case, where we try to remove some data points belonging to digit 4 with a certain writing style. Figure 8 (a) shows that, before the removal operation, the model predicts the marked data points quite well; all of the predictions are correct with high confidence ($= 1.0$). After the removal operation, we note the marked data points are misclassified with various prediction confidences, as shown in Figure 8 (b). In contrast, the predictions on other data points of the same class are not seriously affected by the removal operation as their predictions are still correct, as shown in Figure 8 (c). This demonstrates that the PUMA data removal operation does not false-fully generalize to the other data points in the same class, even if they look similar. Finally, Figure 8 (d) shows that the removal operation also preserved the prediction accuracy for other inputs.

Table 5: Comparison of Model Performance Preservation among Candidate Removal Approaches with Statistics. Value shows accuracy. Higher is better after data removal. Results are collected from 20 times independent experiments.

Data Group	Dataset	Ordered					
		Original	Approach	20%	40%	60%	80%
Synthetic	Radial	95.04 ± 0.59	Retrain Model	**93.64 ± 0.87**	**91.60 ± 1.22**	**84.15 ± 3.00**	66.24 ± 6.42
		80.88 ± 3.62	SISA	67.35 ± 8.16	61.93 ± 7.86	63.57 ± 9.53	51.91 ± 4.57
		95.04 ± 0.59	Amnesiac ML	56.38 ± 4.85	54.75 ± 4.46	53.53 ± 4.68	50.54 ± 1.60
		94.97 ± 0.64	PUMA	68.97 ± 9.47	69.60 ± 4.88	67.99 ± 5.62	**70.77 ± 7.61**
	Rectangular	62.00 ± 0.81	Retrain Model	**61.20 ± 1.08**	**60.35 ± 1.10**	**55.80 ± 3.75**	54.25 ± 3.86
		55.60 ± 5.04	SISA	55.90 ± 4.97	48.30 ± 12.39	30.10 ± 7.78	29.55 ± 5.71
		62.00 ± 0.81	Amnesiac ML	46.60 ± 9.34	43.85 ± 8.37	43.45 ± 9.73	39.15 ± 4.29
		61.85 ± 0.74	PUMA	55.25 ± 8.28	56.30 ± 6.11	53.85 ± 9.64	**61.70 ± 6.80**
Tabular (UCI)	German	71.52 ± 0.49	Retrain Model	**70.56 ± 0.86**	70.12 ± 0.80	70.11 ± 0.73	70.00 ± 0.00
		70.00 ± 0.00	SISA	70.00 ± 0.00	70.00 ± 0.00	68.96 ± 3.28	66.16 ± 5.23
		71.52 ± 0.49	Amnesiac ML	68.52 ± 5.26	64.40 ± 10.80	66.24 ± 8.60	64.03 ± 12.27
		71.47 ± 0.50	PUMA	69.08 ± 3.19	**70.72 ± 0.77**	**70.64 ± 0.73**	**70.72 ± 1.07**
	Breast Cancer	96.45 ± 0.45	Retrain Model	**96.62 ± 0.42**	**96.11 ± 0.59**	96.00 ± 0.54	94.85 ± 1.92
		91.31 ± 1.84	SISA	89.20 ± 1.67	88.91 ± 5.61	80.68 ± 12.00	52.62 ± 26.43
		96.45 ± 0.45	Amnesiac ML	96.05 ± 1.71	95.82 ± 1.54	95.25 ± 2.85	82.28 ± 11.89
		96.39 ± 0.47	PUMA	96.17 ± 1.42	95.88 ± 1.67	**96.22 ± 0.61**	**96.62 ± 0.62**
Image	MNIST	97.58 ± 0.48	Retrain Model	**97.28 ± 0.16**	**96.72 ± 0.56**	95.76 ± 0.73	93.48 ± 0.47
		95.89 ± 0.24	SISA	95.80 ± 0.36	95.67 ± 0.57	94.78 ± 0.46	89.86 ± 0.39
		97.44 ± 0.43	Amnesiac ML	9.44 ± 1.12	9.84 ± 1.42	9.56 ± 1.25	9.36 ± 1.24
		97.60 ± 0.37	PUMA	96.70 ± 0.99	96.66 ± 0.48	**97.17 ± 0.33**	**97.16 ± 0.26**
Data Group	Dataset	Random					
		Original	Approach	20%	40%	60%	80%
Synthetic	Radial	95.89 ± 0.88	Retrain Model	**93.97 ± 1.01**	**90.94 ± 1.25**	**82.58 ± 4.87**	66.51 ± 6.61
		75.62 ± 5.55	SISA	64.71 ± 7.29	64.35 ± 8.78	54.80 ± 3.65	54.77 ± 5.70
		95.88 ± 0.88	Amnesiac ML	49.08 ± 2.31	48.95 ± 2.20	48.95 ± 2.20	48.95 ± 2.20
		95.82 ± 0.86	PUMA	72.44 ± 7.05	73.22 ± 7.82	71.82 ± 7.39	**76.02 ± 8.43**
	Rectangular	65.00 ± 0.23	Retrain Model	**64.70 ± 1.00**	**64.50 ± 1.54**	62.30 ± 2.49	58.65 ± 3.55
		56.50 ± 0.00	SISA	56.50 ± 0.00	56.50 ± 0.00	56.55 ± 0.15	56.90 ± 1.44
		65.00 ± 0.23	Amnesiac ML	35.40 ± 19.08	35.40 ± 19.08	35.40 ± 19.08	35.40 ± 19.08
		64.95 ± 0.15	PUMA	59.90 ± 4.53	62.05 ± 2.74	**62.55 ± 2.49**	**64.80 ± 1.60**
Tabular (UCI)	German	75.16 ± 0.66	Retrain Model	**74.88 ± 0.95**	**73.24 ± 0.66**	72.47 ± 0.88	70.00 ± 0.18
		70.00 ± 0.00	SISA	70.00 ± 0.00	70.00 ± 0.00	70.00 ± 0.00	70.00 ± 0.00
		75.16 ± 0.66	Amnesiac ML	36.24 ± 7.50	36.28 ± 7.46	35.72 ± 7.69	35.72 ± 7.69
		75.12 ± 0.61	PUMA	70.96 ± 3.62	**73.24 ± 1.39**	**74.44 ± 0.66**	**74.28 ± 1.29**
	Breast Cancer	96.00 ± 0.00	Retrain Model	**95.82 ± 0.27**	**95.54 ± 0.36**	**95.65 ± 0.39**	95.20 ± 0.67
		92.28 ± 2.24	SISA	91.60 ± 2.95	88.05 ± 4.57	88.22 ± 4.20	87.88 ± 5.23
		96.00 ± 0.00	Amnesiac ML	35.20 ± 20.33	30.51 ± 18.97	30.51 ± 18.97	30.51 ± 18.97
		96.00 ± 0.00	PUMA	95.08 ± 1.94	94.91 ± 2.29	95.25 ± 1.76	**95.54 ± 0.24**
Image	MNIST	97.99 ± 0.36	Retrain Model	**97.72 ± 0.29**	97.16 ± 0.42	96.60 ± 0.74	93.98 ± 0.72
		95.66 ± 0.41	SISA	93.47 ± 0.68	90.63 ± 1.51	78.06 ± 3.28	59.83 ± 10.02
		98.06 ± 0.37	Amnesiac ML	10.39 ± 1.32	10.39 ± 1.32	10.39 ± 1.32	10.39 ± 1.32
		97.97 ± 0.34	PUMA	97.42 ± 0.66	**97.58 ± 0.36**	**97.60 ± 0.58**	**97.61 ± 0.44**

Appendix D: Results with Statistics

Here we show the full table of our experiment results with statistics that were omitted in the main paper.

Performance Preservation with Statistics

Table 5 shows the performance preservation table. The mean values are exactly the same as presented in the main paper. Statistics show standard deviation of 10 runs. Here, we note that Amnesiac ML often shows a large variance compared to the other approaches, even when the marked data points are intentionally organized into a small number of batches (see results in Ordered scenario). In addition, we also observe that, on the small datasets, training multiple sub-models (as SISA does) often results in bad performance compared with single model approaches (Retrain, Amnesiac ML, and PUMA).

Table 6: Comparison of Membership Attack after Data Removal Operation. Value shows percentage of removed data that is identified as training data. Lower values in the table show better performance of removal.

Data Group	Dataset	Ordered							
		Retrain Model		SISA		Amnesiac ML		PUMA	
		Before	After	Before	After	Before	After	Before	After
Synthetic	Radial	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	$\mathbf{0.00 \pm 0.00}$	100.00 ± 0.00	5.31 ± 10.38
	Rectangular	100.00 ± 0.00	91.65 ± 6.20	83.18 ± 19.76	83.18 ± 19.76	100.00 ± 0.00	$\mathbf{33.33 \pm 0.00}$	100.00 ± 0.00	36.66 ± 29.18
Tabular	German	100.00 ± 0.00	77.12 ± 1.38	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	0.00 ± 0.00	100.00 ± 0.00	$\mathbf{3.42 \pm 6.90}$
	Breast Cancer	100.00 ± 0.00	100.00 ± 0.00	87.50 ± 8.83	87.50 ± 8.83	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	$\mathbf{56.25 \pm 13.50}$
Image	MNIST	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	$\mathbf{0.00 \pm 0.00}$

Data Group	Dataset	Random							
		Retrain Model		SISA		Amnesiac ML		PUMA	
		Before	After	Before	After	Before	After	Before	After
Synthetic	Radial	100.00 ± 0.00	52.36 ± 3.86	100.00 ± 0.00	37.00 ± 17.88	100.00 ± 0.00	50.00 ± 0.00	100.00 ± 0.00	$\mathbf{1.18 \pm 1.96}$
	Rectangular	100.00 ± 0.00	67.07 ± 5.29	98.50 ± 3.37	94.00 ± 12.86	100.00 ± 0.00	86.20 ± 0.00	100.00 ± 0.00	$\mathbf{20.00 \pm 23.30}$
Tabular	German	94.44 ± 0.00	84.44 ± 2.34	100.00 ± 0.00	98.81 ± 8.38	94.44 ± 0.00	93.33 ± 3.51	85.18 ± 2.22	$\mathbf{2.22 \pm 7.13}$
	Breast Cancer	100.00 ± 0.00	100.00 ± 0.00	90.00 ± 7.90	73.75 ± 27.29	100.00 ± 0.00	87.50 ± 0.00	100.00 ± 0.00	$\mathbf{71.25 \pm 25.71}$
Image	MNIST	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	$\mathbf{72.00 \pm 13.03}$

Table 7: Comparison of Running Time (in Seconds). Lower values in the table show better performance to the mislabelled data debugging. We omit the statistic in this table for saving space. Please refer to appendix for statistics.

Data	Approach				
	GShapley	NTK	SelfInfluence	RSP	PUMA
Two Moons	90.37 ± 2.50	22.65 ± 0.62	1562.93 ± 113.78	17.14 ± 0.17	$\mathbf{7.61 \pm 0.08}$
Spiral	78.47 ± 0.62	19.91 ± 0.23	1464.01 ± 5.32	16.51 ± 0.13	$\mathbf{7.44 \pm 0.11}$
Radial	82.99 ± 0.29	21.60 ± 0.42	1563.53 ± 2.54	17.76 ± 0.27	$\mathbf{7.76 \pm 0.07}$
Rectangulars	78.04 ± 0.79	20.23 ± 0.38	1480.12 ± 9.33	16.81 ± 0.26	$\mathbf{7.29 \pm 0.08}$

Removal Performance with Statistics

Table 6 shows the data removal performance table with statistics. As the data points marked to be removed are randomly selected in each experiment, we observe large variances in the table. However, since the mean values reported in the table are dramatically different, we don't observe an overlap among the statistics of the results.

Execution Time for Debugging

Table 7 shows the execution time for mislabelled data debugging with statistics.

Appendix E: Model Calibration Application

As described in the main paper, PUMA preserves model performance while removing and reweighting the contributions of the training data points. The performance can be measured by any differentiable performance criterion $\mathcal{C}$. In this example, we define the performance criterion $\mathcal{C}$ as an Expected Calibration Error (ECE) (Guo et al. 2017).

In the following context, we first use PUMA's data debugging ability to identify the problematic data points causing mis-calibration. We then use PUMA's data removal functionality to update the model parameters to get a better calibrated model.

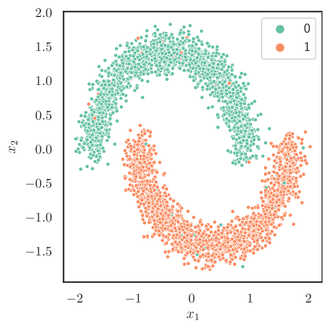
Identifying Problematic Data Points

To identify the problematic data points causing mis-calibration, we need to modify the PUMA debugging algorithm to filter the problematic data points into three categories, namely over-confidence, over-uncertain, and other-noise. Specifically, we make the following adjustment on the Algorithm 2 to meet the requirement.

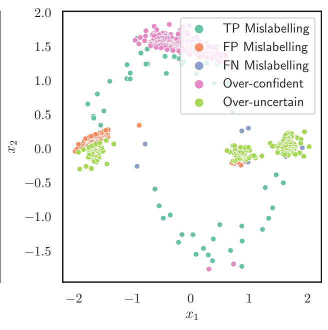
Algorithm 3: PUMA Problematic Data Debugging

```
Require: Original model $\theta_{org}$, Training objective $\mathcal{J}_{org}$, Performance criterion $\mathcal{C}$, Whole training set $D_{tn}$, Number of problematic data to return $k$
Ensure: $\mathcal{C}$ is differentiable
    ...
    $L_{if} \leftarrow argsort(Influence, top = k)$      ▷ Top low influences
    $L_{cf} \leftarrow argsort(Confidence, top = k)$
    $H_{cf} \leftarrow argsort(Confidence, bottom = k)$

    $S_1 \leftarrow L_{if} \cap H_{cf}$                    ▷ Over-confident
    $S_2 \leftarrow L_{if} \cap L_{cf}$                    ▷ Over-uncertain
    $S_3 \leftarrow L_{if} \setminus (L_{cf} \cup H_{cf})$            ▷ Other-noise

    return $S_1$, $S_2$, $S_3$
```

When the performance criterion is ECE, running the above algorithm, we can identify the three categories of data which negatively influence the predictive uncertainty estimation of the model (increasing ECE).

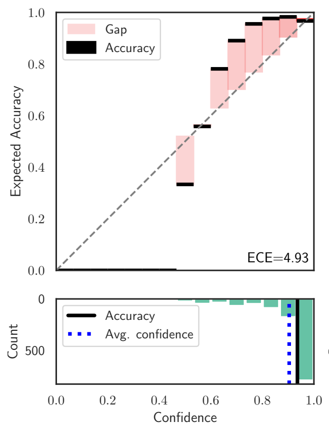


(a) Noisy Data

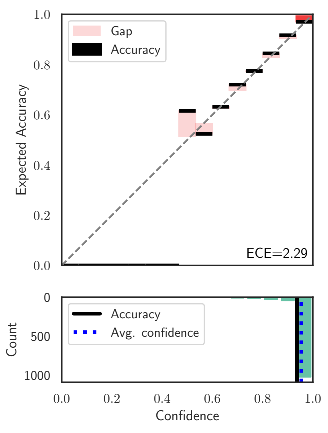


(b) Problem Highlight

Figure 9: Problematic data points discovered by PUMA. The problems are categorized into three categories. For the prediction of mislabelled data points, we show the prediction's True Positive (TP), False Positive (FP), and False Negative (FN).



(a) ECE Before Patching



(b) ECE After Patching

Figure 10: Expected Calibration Error (ECE) before and after PUMA model patching/augmentation. For most of prediction confidence bins, we observe the better calibration after PUMA model augmentation.

Figure 9 shows our experiment results on the synthetic data. The data is heavily corrupted two-moon synthetic data with 5000 data points. Among the data points, 100 data points' labels are randomly flipped. We run PUMA with ECE loss as the performance metric $\mathcal{C}$ to debug the trained model on the corrupted data. Results show that PUMA can fairly accurately identify the various problems in the model (i.e. the three data categories). Note, unlike the mislabelled data points, the identified over-confident and over-uncertain predictions are part of the model problems, not a training data quality issue. Hence, in Figure 9(b), the over-confident predictions are not symmetric in the two classes for this particular model.

Model Patching with PUMA

Now, if we set $D_{mk} \overset{\text{def}}{=} S_1 \cup S_3$ and $D_{up} \overset{\text{def}}{=} S_2$ and run the PUMA data removal algorithm (see Algorithm 1) with small learning rate η (E.g. $\eta \leq 1e-4$), we expect to see that the ECE loss of the model is reduced after updating the model with PUMA.

Figure 10 shows the model augmentation results. The results reflect our expectation, since the ECE loss of the updated model is dramatically reduced.

Appendix F: Discussion

Purging Data vs. Data Characteristic Removal

Complete removal of data points from a trained model is barely possible since the model training procedure is usually complex and it mixes information from all training data points with mutual dependence. In particular, training process of the modern deep learning often uses momentum-enabled optimization algorithms to avoid getting stuck in a local minimum. This makes the decomposition of the contribution of each training data point in learned model parameters hard. In addition, approaches such as batch normalization and weight decay could make the data contribution estimation even more intractable. Hence, in this paper, we do not intend to completely purge the effects of the training data points. Indeed, the only possible solution for removing data from a model is probably retraining the model from scratch with the remaining data points.

The main goal of this work is to remove the identifiable characteristics of the marked data points such that 1) their negative impact on the model can be mitigated and 2) the marked data points cannot be retrieved or identified by the adversaries. The fundamental difference between our goal and removing/purging data is that we want to keep the general properties of all data points (including the marked ones) that are positively influential to the model such that the model's performance is preserved. This technique is particularly useful when the models under service are required to respond quickly to data removal requests, so that a full model retraining can be conducted offline within certain period of time. Indeed, the immediate response to data removal requests in online service is increasingly important.

Privacy Protection vs Data Characteristic Removal

Privacy protection is a very important research field with many well defined criteria. However, PUMA is not designed to satisfy those criteria (e.g. information leakage (Hitaj, Ateniese, and Perez-Cruz 2017; Ateniese et al. 2015), re-identification attack (Yang et al. 2021), reconstruction attack (Lyu and Chen 2021), tracing attack (Homer et al. 2008), model inversion (Fredrikson, Jha, and Ristenpart 2015b)). Indeed, this paper does not aim to solve privacy preservation problems. While we use the performance of the membership attack as a metric in the paper, it is meant to show the effectiveness of removing data characteristics only.

Actually, we note that PUMA could be used in various application scenarios such as improving the prediction calibration of a model. As shown in previous examples in Appendix E, by removing data which causes over-confidence and upweighting over-uncertain data points optimally, PUMA can adjust a model's prediction uncertainty to fit calibration requirements in certain applications. This application does not involve privacy protection but is very useful for cascade models or multi-stage models, where outputs of upstream models are expected to be well calibrated.