Approximate Data Deletion from Machine Learning Models

Zachary Izzo
Dept. of Mathematics
Stanford University
zizzo@stanford.edu

Mary Anne Smart
Department of CS&E
UC San Diego
msmart@eng.ucsd.edu

Kamalika Chaudhuri
Department of CS&E
UC San Diego
kamalika@cs.ucsd.edu

James Zou
Deptartment of BDS
Stanford University
jamesz@stanford.edu

Abstract

Deleting data from a trained machine learning (ML) model is a critical task in many applications. For example, we may want to remove the influence of training points that might be out of date or outliers. Regulations such as EU's General Data Protection Regulation also stipulate that individuals can request to have their data deleted. The naive approach to data deletion is to retrain the ML model on the remaining data, but this is too time consuming. In this work, we propose a new approximate deletion method for linear and logistic models whose computational cost is linear in the the feature dimension d and independent of the number of training data n. This is a significant gain over all existing methods, which all have superlinear time dependence on the dimension. We also develop a new feature-injection test to evaluate the thoroughness of data deletion from ML models.

1 Introduction

Given a trained machine learning (ML) model, there are many settings where we would like to *delete* specific training points from this trained model. Deletion here means that we need to post-process the model to remove the effect of the specified training point(s). One example of the need for deletion is the Right to be Forgotten requirement which is a part of many policies including the EU's General Data Protection Regulation and the recent California Consumer Privacy Act. The Right to be Forgotten stipulates that individuals can request to have their personal data be deleted and cease to be used by organizations and companies such as Google, Facebook, etc. The challenge here is that even after an organization deletes the data associated with a given individual, information about that individual may persist in predictions made by machine learning models trained on the deleted data. These predictions may in turn leak information, impeding the individual's ability to truly be "forgotten." For example, recent works show how one can reconstruct training data by attacking vision and NLP models (Zhang et al., 2019). Therefore there is a great need for approaches to remove an individual's data from the trained ML model as much as possible.

We propose a computational model inspired by this problem. After allowing a reasonable amount of precomputation, the model designer will receive a request to delete a batch of k points from the model. Our goal is to accomplish this task as efficiently and accurately as possible.

A first plausible solution is exact data deletion, where the goal is to exactly reproduce the model that would have been output had the deleted points been omitted from training. However, in general this is computationally demanding: except for a few limited scenarios (e.g. (Ginart et al., 2019) for K-means clustering), it will require retraining the model from scratch. Even in the simple case of training a logistic regression model via SGD, this will take time $O(ndP)$, where n is the size of the dataset, d is the dimensionality of the data, and P is the number of passes over the data. When deletion requests need to be fulfilled promptly and in an online setting, retraining the model completely is infeasible. This motivates our study of approximate data deletion: by relaxing the requirements for removing data from the model, we hope to make the problem computationally tractable.

Approximate data deletion has two main challenges – algorithmic (i.e. how to delete points effectively and quickly) and evaluation (i.e. how to quantify the quality of our approximate deletion). In this paper, we make

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

progress in both of these areas.

Existing approaches to approximate deletion include the use of influence functions (Koh and Liang, 2017) and Newton's method. Both of these methods have computational costs which scale as $\Omega(d^2)$, where d is the dimensionality of the data. We develop the first approximate deletion method with $O(d)$ computational cost, dubbed the *projective residual update*, which computes the projection of the exact parameter update vector onto a particular low-dimensional subspace. The dependence of the computational cost on d matches the trivial lower bound $\Omega(d)$ required to fully specify all of the entries of the model parameters (which we also assume to be d-dimensional), and is independent of the number of training data n. We additionally show that the PRU is optimal in terms of deletion accuracy within a certain class of gradient-based deletion methods.

Additionally motivated by privacy concerns, we propose a new evaluation criterion, dubbed the *feature injection test*, which captures a deletion method's ability to remove the model's knowledge of sensitive attributes of the deleted points. The test works by adding a synthetic feature to only the deleted points which is perfectly correlated with the label, then measuring the amount by which the deletion method removes the weight on this artificial feature. All of our theoretical findings are corroborated with experiments on both real and synthetic datasets.

Summary of contributions.

- We introduce a novel approximate data deletion method, the *projective residual update* (PRU), which has a time complexity that is *linear* in the dimension of the deleted data and is independent of the size of the dataset. We show that this method is optimal among a certain class of gradient-based updates in terms of deletion accuracy.
- We propose a new metric for evaluating data removal from models—the *feature injection test* (FIT)—which captures how well we can remove the model's "knowledge" of a sensitive, highly predictive feature present in the data.
- Experiments support our theoretical findings.

2 Notation and Problem Setup

For the reader's convenience, we collect key notation and background here. Throughout the paper, n denotes the total number of training points, d denotes the data dimension, and k denotes the number of data points to be deleted from the model. The k points to be deleted will be supplied as a batch request—that is, the k points should be deleted simultaneously, rather than one-by-one. We may think of this either as a request from a group of individuals, or a request to delete all of the data for one individual who has k datapoints associated to her in the database. We will always assume that $n \gg d \gg k$.

- $\theta \in \mathbb{R}^d$ denotes the model parameters.
- $D^{\text{full}} = \{(x_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times \mathbb{R}$ is the full set of training data. Throughout the paper, we assume that the feature vectors x_i are in general position, i.e. that any collection of at most d x_is may be assumed to be linearly independent. This assumption holds with probability 1 when the x_i are drawn i.i.d. from any distribution arising from a probability density on $\mathbb{R}^d$ (i.e. a probability distribution on $\mathbb{R}^d$ which is absolutely continuous with respect to the Lebesgue measure), for instance a non-degenerate Gaussian.
- $X = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}^\intercal \in \mathbb{R}^{n \times d}$ is the data matrix for D^{full}; its rows are the feature vectors $x_i^\intercal$. Note that since we have assumed that the x_i are in general position and that $n \gg d$, X is implicitly assumed to have full column rank.
- $Y = (y_1, \ldots, y_n)^\intercal \in \mathbb{R}^n$ is the response vector for D^{full}.
- $D^{\setminus k} = \{(x_i, y_i)\}_{i=k+1}^n$ is the dataset with the k desired points removed. We assume WLOG that that these are the first k points, and we will frequently refer to this as the leave-k-out (LKO) dataset.
- $L^{\text{full}}(\theta) = \sum_{i=1}^n \ell(x_i, y_i; \theta) + \frac{\lambda}{2}\|\theta\|_2^2$ is the (ridge-regularized) loss on the full dataset. The "single-point" loss function ℓ will be the quadratic loss for linear regression ($\frac{1}{2}(\theta^\intercal x_i - y_i)^2$). Note that this includes the unregularized setting by simply taking the regularization strength $\lambda = 0$.
- $L^{\setminus k}(\theta) = \sum_{i=k+1}^n \ell(x_i, y_i; \theta) + \frac{\lambda}{2}\|\theta\|_2^2$ is the loss on the LKO dataset. We require that the regularization strength be fixed independent of the number of samples.
- $\theta^{\text{full}} = \operatorname{argmin}_\theta L^{\text{full}}(\theta)$ are the model parameters when fitted to the full dataset. We will refer to the model with these parameters as the full model. In the case of linear regression, θ^{full} has the explicit form $\theta^{\text{full}} = (X^\intercal X + \lambda I)^{-1} XY$. (This is derived by setting the gradient of the loss to zero.)
- $\theta^{\setminus k} = \operatorname{argmin}_\theta L^{\setminus k}(\theta)$ are the model paramteres when fitted to the LKO dataset. We will refer to the model with these parameters as the LKO model.

- $\hat{y}_i^{\setminus k} = \theta^{\setminus k\intercal} x_i$ is the prediction of the LKO model on the i-th datapoint.

We remark that although the number of data points k to be deleted from the model is small compared to the dimension d, we make no more assumptions. In particular, we do not assume that the removed points need to be in any way "similar" (e.g. i.i.d.) to the rest of the data and specifically consider cases where large outliers are removed. As a result, the removal of these points can still have a large impact on the model parameters.

For our discussions of computational cost, we are interested in updating and quickly redeploying the model after a deletion request. Thus, we consider only the "just-in-time" (i.e. at the time of the deletion request) computational cost of each method. A reasonable amount of precomputation (that is, computations which can be done without knowledge of the k points to be deleted) will be permitted without being included in the computational cost. Here, "reasonable" is simply meant to exclude trivial but prohibitively expensive methods such as training a model on each subset of the training data, then returning the model parameters corresponding to the dataset with the appropriate points removed at deletion time.

We emphasize that we will focus on fulfulling a *single* such batch deletion request. While simple, this framework captures the key essence of the data deletion challenge. Extending our methods to work in a fully online setting, where we may receive several batch deletion requests and the precomputation required between each request becomes significant, is an important next step towards practical approximate deletion methods.

We obtain results for both linear and logistic regression models. The results for logistic regression are an extension of the results for linear regression, so we choose to focus primarily on linear regression in the main body of the paper and defer a more complete discussion of logistic regression to the appendix.

Finally, we note that while we focus on linear models for the sake of theoretical clarity, these two scenarios capture most of the difficulty for nonlinear models as well. When retraining e.g. deep neural networks, it is often sufficient to consider all but the final layer as a fixed feature map on top of which we perform either linear or logistic regression (for regression and classification tasks, respectively). Retraining only the last layer is then sufficient and reduces to the two cases we consider in this paper. This method can be seen in (Koh and Liang, 2017), in which the authors retrain their model to determine which training images are most influential for an image classification task; and in (Ghorbani and Zou, 2019), where the authors retrain their model in order to compute data Shapley values, a measure of how much each data point contributes to the model's overall accuracy. In both cases, retraining only the last layer of the model is sufficient to give accurate results, and our results here can be similarly applied to the last layer.

3 Methods

We give a brief overview of approximate deletion methods for parametric models from the literature.

Exact retraining The most straightforward way to remove data is by retraining the model completely. For the case of linear regression, we can naively compute $\theta^{\setminus k}$ using the analytic formula $\theta^{\setminus k} = (X^{\setminus k\intercal} X^{\setminus k} + \lambda I)^{-1} X^{\setminus k\intercal} Y^{\setminus k}$. ($X^{\setminus k}$ and $Y^{\setminus k}$ are the data matrix and response vector for $D^{\setminus k}$, respectively.) The bottleneck is in forming the new Hessian $X^{\setminus k\intercal} X^{\setminus k}$, giving an overall computational cost of $O(nd^2)$. Alternatively, we could retrain via iterative methods like SGD. This will take time $O(ndP)$, where P is the number of passes over the dataset.

Newton's method Recent work (Guo et al., 2019) has attempted approximate retraining by taking a single step of Newton's method. This amounts to forming a quadratic approximation to the LKO loss $L^{\setminus k}$ and moving to the minimizer of the approximation. This can be done in closed form, yielding the update

$$\theta_{\text{Newton}} = \theta^{\text{full}} - [\nabla_\theta^2 L^{\setminus k}(\theta^{\text{full}})]^{-1} \nabla_\theta L^{\setminus k}(\theta^{\text{full}}). \quad (1)$$

When the loss function is quadratic in θ (as is the case in least squares linear regression), the approximation to $L^{\setminus k}$ is just $L^{\setminus k}$ itself and so Newton's method gives the exact solution. That is, in the case of linear regression, Newton's method reduces to the trivial "approximate" retraining method of retraining the model exactly.

Since the full Hessian can be computed without knowing which points need to be deleted, we can consider it an offline cost. For linear regression, the new Hessian matrix is a rank k update of the full Hessian, which can be computed via the Sherman-Morrison-Woodbury formula in $O(kd^2)$ time. In general, forming and inverting the new Hessian may take up to $O(nd^2)$ time.

Influence method Recent works studied how to estimate the influence of a particular training point on the model's predictions (Giordano et al., 2018; Koh and Liang, 2017). While the original methods were developed for different applications—e.g. interpretation and cross-validation—they can be adapted to perform approximate data deletion. Under suitable assumptions on the loss function ℓ, we can view the

model parameters θ as a function of weights on the data: $\theta(w) \equiv \text{argmin}_\theta \sum_{i=1}^n w_i \ell(x_i, y_i; \theta)$. In this setting, $\theta^{\text{full}} = \theta(\mathbf{1})$ where $\mathbf{1}$ is the all 1s vector and $\theta^{\setminus k} = \theta((\underbrace{0, \ldots,}_{k} \underbrace{1, \ldots}_{n-k})^\intercal)$. The influence function approach (henceforth referred to as the influence method) uses the linear approximation to $\theta(w)$ about $w = \mathbf{1}$ to estimate $\theta^{\setminus k}$. (Giordano et al., 2018; Koh and Liang, 2017) show that the linear approximation is given by

$$\theta^{\text{inf}} = \theta^{\text{full}} - [\nabla_\theta^2 L^{\text{full}}(\theta^{\text{full}})]^{-1} \nabla_\theta L^{\setminus k}(\theta^{\text{full}}). \quad (2)$$

Assuming that we already have access to the inverse of the Hessian, the bottleneck for this method is the Hessian-gradient product. This gives a $O(d^2)$ computational cost.

We summarize the asymptotic online computational costs in Table 1 alongside the computational cost of our novel method, the projective residual update. (Since the Newton step with Sherman-Morrison formula is exact and has a strictly lower computational cost than the naive method of retraining from scratch, we do not include the naive method in the table.) The precomputation costs for each of the methods (Newton, influence, and PRU) are approximately the same; they are dominated by forming and inverting the full Hessian, which takes time $O(nd^2)$.

Table 1: Asymptotic computational costs for each approximate retraining method. The projective residual update is the only method with linear dependence on d.

Exact	Influence	Projective residual
$O(kd^2)$	$O(d^2)$	$O(k^2 d)$

4 The Projective Residual Update

We now introduce our proposed approximate update. We leverage *synthetic data*, a term we use to refer to artificial datapoints which we construct and whose properties form the basis of the intuition for our method. We combine gradient methods with synthetic data to achieve an approximate parameter update which is fast for deleting small groups of points. The intuition is as follows: if we can calculate the values $\hat{y}_i^{\setminus k} = \theta^{\setminus k\intercal} x_i$ that the model would predict on each of the removed x_is *without knowing* $\theta^{\setminus k}$, then minimize the loss of the model on the synthetic points $(x_i, \hat{y}_i^{\setminus k})$ for $i = 1, \ldots, k$, we should expect our parameters to move closer to $\theta^{\setminus k}$ since $\theta^{\setminus k}$ achieves the minimum loss on the points $(x_i, \hat{y}_i^{\setminus k})$. We will minimize the loss on these synthetic points by taking a (slightly modified) gradient step.

It may be surprising that we can calculate the values $\hat{y}_i^{\setminus k}$ without needing to know $\theta^{\setminus k}$. We accomplish this by generalizing a well-known technique from statistics for computing leave-one-out residuals for linear models. As in the influence function applications, we incur an upfront cost of forming the so-called "hat matrix" $H \equiv X(X^\intercal X + \lambda I)^{-1} X^\intercal$ for the full linear regression. Since we can compute this matrix without needing to know which points will be deleted, it is reasonable to consider it as an offline computation which will not be included in the computational cost of the update itself.

We formalize the intuition for the update as follows. Assume that we can compute $\hat{y}_i^{\setminus k}$ efficiently, without needing to know $\theta^{\setminus k}$. The gradient of the loss on the synthetic points $(x_i, \hat{y}_i^{\setminus k})$ is $\nabla_\theta L^{\{(x_i, \hat{y}_i^{\setminus k})\}_{i=1}^k}(\theta) = \sum_{i=1}^k (\theta^\intercal x_i - \hat{y}_i^{\setminus k}) x_i$. Substituting $\theta^{\setminus k\intercal} x_i$ for $\hat{y}_i^{\setminus k}$ and rearranging, then setting $\theta = \theta^{\text{full}}$ shows that $\nabla_\theta L^{\{(x_i, \hat{y}_i^{\setminus k})\}_{i=1}^k}(\theta^{\text{full}}) = \left(\sum_{i=1}^k x_i x_i^\intercal\right)(\theta^{\text{full}} - \theta^{\setminus k})$. We show that the form of the matrix $\sum_{i=1}^k x_i x_i^\intercal$ allows us to efficiently compute a pseudoinverse. We summarize these steps in Algorithm 1.

Algorithm 1 The projective residual update

procedure PRU($X, Y, H, \theta^{\text{full}}, k$)
  $\hat{y}_1', \ldots, \hat{y}_k' \leftarrow \text{LKO}(X, Y, H, k)$
  $S^{-1} \leftarrow \text{PseudoInv}(\sum_{i=1}^k x_i x_i^\intercal)$
  $\nabla L \leftarrow \sum_{i=1}^k (\theta^{\text{full}\intercal} x_i - \hat{y}_i') x_i$
  return $\theta^{\text{full}} - \text{FastMult}(S^{-1}, \nabla L)$
end procedure

Algorithm 2 Leave-k-out predictions

procedure LKO($X, Y, H, \theta^{\text{full}}, k$)
  $R \leftarrow Y_{1:k} - X_{1:k}\theta^{\text{full}}$
  $D \leftarrow \text{diag}(\{(1 - H_{ii})^{-1}\}_{i=1}^k)$
  $T_{ij} \leftarrow \mathbf{1}\{i \neq j\} \frac{H_{ij}}{1 - H_{jj}}$
  $T \leftarrow (T_{ij})_{i,j=1}^k$
  $\hat{Y}^{\setminus k} \leftarrow Y_{1:k} - (I - T)^{-1} DR$
  return $\hat{Y}^{\setminus k}$
end procedure

The results of running the residual update are described by Theorem 1, our main theorem.

Theorem 1. *Algorithm 1 computes* $\theta^{\text{res}} = \theta^{\text{full}} + \text{proj}_{\text{span}(x_1, \ldots, x_k)}(\theta^{\setminus k} - \theta^{\text{full}})$ *with computational cost* $O(k^2 d)$.

The result of Theorem 1 is striking. It says that the projective residual update makes the *most improvement possible* for any parameter update which is a linear

combination of the removed x_is. As a direct result of this, we have the following corollary.

Definition 2. For any dataset $D = \{(\bar{x}_i, \bar{y}_i)\}_{i=1}^N$ (not necessarily the same as the original dataset D^{full}), define $L^D(\theta) = \sum_{i=1}^N \frac{1}{2}(\theta^\intercal \bar{x}_i - \bar{y}_i)^2$. Define a *gradient-based update* of the model parameters θ^{full} as any update θ^{approx} which can be computed by the following procedure: set $\theta_0 = \theta^{\text{full}}$, then define

$$\theta_{t+1} = \theta_t - \alpha_t \nabla_\theta L^{D_t}(\theta_t)$$

for some sequence of datasets D_t. Finally, let $\theta^{\text{approx}} = \theta_T$ for some T.

Corollary 3. *Let $\mathcal{S} = \{x_i\}_{i=1}^k \times \mathbb{R}$ be the set of all datapoints whose feature vectors belong to the set of points to be deleted from the original dataset D^{full}. If θ^{approx} is any gradient-based update of θ^{full} with $D_t \subseteq \mathcal{S}$ for all t, then we have*

$$\|\theta^{\setminus k} - \theta^{\text{res}}\| \leq \|\theta^{\setminus k} - \theta^{\text{approx}}\|.$$

Proof. This follows immediately from the fact that the gradient of the square loss on any point (x_i, y) is a scalar multiple of x_i, and therefore the update $\theta^{\text{full}} - \theta^{\text{approx}} \in \text{span}(x_1, \ldots, x_k)$. □

As we will see later, Theorem 1 guarantees that PRU performs well for deleting the model's knowledge of sensitive attributes under data sparsity conditions.

The LKO, PseudoInv, and FastMult subroutines The ability to efficiently calculate $\hat{y}_i^{\setminus k}$, $i = 1, \ldots k$ is crucial to our method. Algorithm 2 accomplishes this by generalizing a well-known result from statistics which allows one to compute the leave-one-out residuals $\hat{y}_i^{\setminus 1} - y_i$. In Algorithm 2, $X_{1:k}$ denotes the first k rows of X, $Y_{1:k}$ the first k entries of Y, and H the hat matrix for the data, defined below. Since the residuals $r_i = y_i - x_i^\intercal \theta^{\text{full}}$ and the hat matrix H can be computed before the time of the deletion request, these steps can be excluded from the total computational cost of Algorithm 2.

Theorem 4. *Algorithm 2 computes the LKO predictions $\hat{y}_i^{\setminus k}$, $i = 1, \ldots, k$ in $O(k^3)$ time.*

The proof of Theorem 4 can be found in Appendix C. The low-rank structure of $A \equiv \sum_{i=1}^k x_i x_i^\intercal$ allows us to quickly compute its pseudoinverse. We do this by finding the eigendecomposition of an associated $k \times k$ matrix (which can again be done quickly when k is small, see e.g. (Pan and Chen, 1999)), which we then leverage to find the eigendecomposition of A. Computing the pseudoinverse in this way also allows us to multiply by it quickly. For a more detailed explanation, refer to the appendix.

4.1 Outlier deletion

To illustrate the usefulness of the residual update, we consider its performance compared to the influence method on the dataset $D^{\text{full}} = \{(\lambda x_1, \lambda y_1)\} \cup D^{\setminus 1}$, where $D^{\setminus 1} = \{(x_i, y_i)\}_{i=2}^{n+1}$ and we are attempting to remove the first datapoint from D^{full} so that we are left with $D^{\setminus 1}$. In particular, we examine the difference in performance between the residual and influence updates as the parameter $\lambda \to \infty$.

Theorem 5. *Let $D^{\text{full}} = \{(\lambda x_1, \lambda y_1)\} \cup D^{\setminus 1}$. Then $\theta^{\text{inf}} \to \theta^{\text{full}}$ as $\lambda \to \infty$.*

Theorem 5 says that when we try to delete points with large norm, the influence method will barely update the parameters at all, with the update shrinking as the size of the removed features increases. This makes intuitive sense. The performance of the influence method relies on the Hessian of the full loss being a good approximation of the Hessian of the leave-one-out loss. As the scaling factor λ grows, the full Hessian $X^\intercal X + \lambda^2 x_1 x_1^\intercal$ deviates more and more from the LOO Hessian $X^\intercal X$, causing this drop in performance. On the other hand, as the size of the outlier grows, the exact parameter update vector $\theta^{\text{full}} - \theta^{\setminus 1}$ approaches a well-defined, finite limit. The PRU computes the projection of this update onto the subspace spanned by the deleted points, and therefore in general its improvement will remain bounded away from 0 even as the outlier grows. It follows that the PRU will outperform the influence method for the deletion of large enough outliers. For a complete proof of this fact, see Proposition 7 in Appendix B.

4.2 Extension to logistic regression

The generalization of the PRU to logistic regression relies on the fact that a logistic model can be trained by iteratively reweighted least squares; indeed, a Newton step for logistic regression reduces to the solution of a weighted least squares problem (Murphy, 2012). We leverage this fact along with the generalization of Theorem 4 from Appendix D to compute a fast approximation to a Newton step. The method is given by Algorithm 3. (Note: $H_{\lambda,w}$ denotes the Hessian for a weighted linear least squares problem with weights w and regularization λ.)

Theorem 6. *Algorithm 3 computes the update $\theta^{\text{res}} = \theta^{\text{full}} + proj_{span(x_1,\ldots,x_k)}(\Delta_{Newton})$ in $O(k^2 d)$ time.*

Refer to Appendix E for an explanation of the algorithm and the proof.

Algorithm 3 The PRU for logistic regression

procedure LOGISTICPRU($X, Y, \theta^{\text{full}}, k$)
  for $i = 1, \ldots, n$ **do**
    $w_i \leftarrow h_{\theta^{\text{full}}}(x_i)(1 - h_{\theta^{\text{full}}}(x_i))$
  end for
  $S_{\theta^{\text{full}}} \leftarrow \text{diag}(w)$
  $Z \leftarrow X\theta^{\text{full}} + S^{-1}_{\theta^{\text{full}}}(Y - h_{\theta^{\text{full}}})$
  return RESIDUALUPDATE($X, Z, H_{\lambda,w}, \theta^{\text{full}}, k$)
end procedure

5 Evaluation Metrics

L^2 distance A natural way to measure the effectiveness of an approximate data deletion method is to consider the L^2 distance between the estimated parameters and the parameters obtained via retraining from scratch. If the approximately retrained parameters have a small L^2 distance to the exactly retrained parameters, then when the models depend continuously on their parameters (such as in linear regression), the models are guaranteed to make similar predictions.

In addition to the general similarity between two models captured by the L^2 distance, we are also interested in studying a more fine-grained metric: how well can an approximate deletion method remove specific sensitive attributes from the retrained model? This motivates a new metric that we propose: the *feature injection test*.

Feature injection test The rationale behind this new test is as follows. If a user's data belongs to some small minority group within a dataset, that user may be concerned about what the data collector will be able to learn about her and this small group. When she requests that her data be deleted from a model, she will want any of these localized correlations that the model learned to be forgotten.

This thought experiment motivates a new test for evaluating data deletion, which we call the feature injection test (FIT). We inject a strong signal into our dataset which we expect the model to learn. Specifically, we append an extra feature to the data which is equal to zero for all but a small subset of the datapoints, and which is perfectly correlated with the label we wish to predict. In the case of a linear classifier, we expect the model to learn a weight for this special feature with absolute value significantly greater than zero. After this special subset is deleted, however, any strictly positive regularization will force the weight on this feature to be 0 in the exactly retrained model. We can plot the value of the model's learned weight for this special feature before and after deletion and use this as a measure of the effectiveness of the approximate deletion method.

Below we give a general description the FIT for logistic regression. Let $D^{\text{full}} = \{(x_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times \{0, 1\}$ be the full dataset and assume WLOG that we wish to delete points $i = 1, \ldots, k$. We require that the deleted points all belong to the positive class, i.e. $y_1 = \cdots = y_k = 1$.

1. Set $\tilde{x}_i = [x_i^\intercal,\ 1]^\intercal$ for $1 \leq i \leq k$ and $\tilde{x}_i = [x_i^\intercal,\ 0]^\intercal$ for $k < i \leq n$. The last entry of each $\tilde{x}_i$ is the injected feature; each deleted point has an injected feature with value 1, while the non-deleted points have injected feature value 0.
2. Train a logistic classifier on $\{(\tilde{x}_i, y_i)\}_{i=1}^n$ (using ridge-regularized cross-entropy loss and strictly positive regularization strength) and let $\theta^{\text{full}} \in \mathbb{R}^{d+1}$ be the weights of the resulting model. Define $w_* = \theta^{\text{full}}[d+1]$ to be the $d+1$-th entry of θ^{full}, i.e. the weight corresponding to the injected feature.
3. Given the output θ^{approx} of an approximate retraining method, its FIT metric is defined as $\theta^{\text{approx}}[d+1]/w_*$. The closer the FIT metric is to 0, the better the approximate deletion method is at removing the injected sensitive feature from the model.

For a description of the FIT for linear regression, see Appendix F.

6 Empirical Validation

We now verify our theoretical guarantees and compare the accuracy and speed of the various retraining methods experimentally. We emphasize that these experiments are intended to confirm the theory rather than demonstrate practical usage. Deploying and testing large-scale data deletion methods is an important direction of future work. Our analysis and methods provide an important initial step towards this goal. Code for reproducing our experiments can be found at `https://github.com/zleizzo/datadeletion`.

6.1 Linear regression

Synthetic datasets The synthetic datasets are constructed so that the linear regression model is well-specified. That is, given the data matrix X, the response vector Y is given by $Y = X\theta^* + \varepsilon$, where $\varepsilon \sim N(0, \sigma^2 I_n)$ is the error vector. For all of the synthetic datasets, we take $n = 10d$. Slight modifications are made to this general setup for each experiment. For outlier removal tests, we scale a subset of the full dataset to create outliers, then delete these points. For the sparse data setting, we generate sparse feature vectors rather than drawing from a Gaussian. For full details on dataset construction, refer to the appendix.

Table 2: Mean runtimes for each method as a fraction of full retraining runtime (100 trials). (INF stands for influence method.) In all instances, the standard error was not within the significant digits of the mean (all standard errors were of order 10^{-4} or smaller) so for clarity we do not include the errors. The absolute runtimes of the exact method to which we compare is in the appendix. The results match our theory that PRU's runtime is especially advantageous for high dimensions and relatively small k.

	$d = 1000$	$d = 1500$	$d = 2000$	$d = 2500$	$d = 3000$
$k = 1$ (INF)	0.0085	0.0053	0.0041	0.0036	0.0028
$k = 1$ (PRU)	**0.0062**	**0.0017**	**0.0008**	**0.0004**	**0.0003**
$k = 5$ (INF)	**0.0092**	0.0052	0.0043	0.0033	0.0028
$k = 5$ (PRU)	0.0112	**0.0035**	**0.0019**	**0.0011**	**0.0007**
$k = 10$ (INF)	**0.0098**	0.0054	0.0045	0.0033	0.0031
$k = 10$ (PRU)	0.0155	**0.0049**	**0.0025**	**0.0015**	**0.0010**
$k = 25$ (INF)	**0.0105**	**0.0058**	**0.0050**	**0.0035**	0.0032
$k = 25$ (PRU)	0.0365	0.0121	0.0067	0.0037	**0.0026**
$k = 50$ (INF)	**0.0122**	**0.0065**	**0.0051**	**0.0036**	**0.0033**
$k = 50$ (PRU)	0.0794	0.0273	0.0151	0.0085	0.0059

Yelp We select 200 users from the Yelp dataset and use their reviews (2100 reviews in total). We use a separate sample of reviews from the dataset to construct a vocabulary of the 1500 most common words; then we represent each review in our dataset as a vector of counts denoting how many times each word in the vocabulary appeared in the given review. Four and five star reviews are considered positive, and the rest are negative. To turn the regression model's predictions into a binary classifier, we threshold scores at zero– a predicted value that is greater than zero becomes a prediction of the positive class while a predicted value that is less than zero becomes a prediction of the negative class.

Results - Synthetic data The experimental results closely match the theory in all respects. For the runtime experiments, refer to Table 2. Both the influence method and the projective residual update are significantly faster than exact model retraining. In the extreme case of $d = 3000$ and a removal group of size $k = 1$, the projective residual update is more than 3000 times faster than exact retraining. The relative speed of the PRU and influence method are also as we expect: PRU is faster than influence for small group sizes, and the size of the largest group that we can delete while maintaining this speed advantage increases as d increases. For 3000-dimensional data, PRU has the speed advantage for deleting groups as large as 25.

For the FIT, refer to Table 3. As the data matrix becomes more sparse, the span of the removed points become more likely to contain the d-th standard basis vector e_d (or a vector very close to it), allowing the residual update to completely remove the special weight. We observe this phenomenon in several of the cases we tested (denoted by an asterisk in table 3). All of the standard errors for the PRU were well below 5% of the mean. In contrast, the influence method performs poorly compared to the PRU in most scenarios, in addition to exhibiting much less numerical stability.

Table 3: Mean results for the FIT on synthetic data (50 trials). The special weight is given as fraction of baseline weight (lower the better). Results are for $d = 1500$ for various group sizes (k) and sparsity values (p). See text for discussion of the standard errors and the notable values (indicated by asterisks). The baseline weights to which we compare can be found in the appendix. These results match our theory that PRU performs especially well in the sparse regime.

	$p = 0.25$	0.1	0.05
$k = 5$ (INF)	1.09	0.99	1.01
$k = 5$ (PRU)	**0.98**	**0.96**	**0.93**
$k = 50$ (INF)	**0.84**	0.97	2.32^{**}
$k = 50$ (PRU)	0.86	**0.67**	**0.35**
$k = 100$ (INF)	0.76	0.92	0.98
$k = 100$ (PRU)	**0.72**	**0.32**	$\mathbf{0.00}^{*}$

For the L^2 metric, refer to Table 4. The influence method outperforms PRU for deleting "typical" points (when $\lambda = 1$, the deleted points are i.i.d. with the rest of the data rather than being outliers). As the size of the deleted points grows, however, we see a steep drop in the performance of the influence method, while PRU remains almost completely unaffected.

Results - Yelp Since the Yelp dataset does not have large outliers, the influence method outperforms the

Table 4: Mean results for the L^2 test on synthetic data (50 trials). The L^2 distance is given as fraction of baseline distance ($\|\theta^{\text{full}} - \theta^{\setminus k}\|$; the values of the starting distance can be found in the appendix). Results are for $d = 1500$ for various group sizes (k) and outlier sizes (λ, see Theorem 5).

	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$
$k = 5$ (INF)	**0.38**	0.93	0.99
$k = 5$ (PRU)	0.92	**0.92**	**0.92**
$k = 50$ (INF)	**0.16**	0.91	0.99
$k = 50$ (PRU)	0.88	**0.88**	**0.88**
$k = 100$ (INF)	**0.14**	0.90	0.99
$k = 100$ (PRU)	0.88	**0.88**	**0.88**

projective residual update in terms of L^2 distance. For larger groups, however, the PRU's performance on the FIT is superior to the influence method, which fails to remove the injected signal. These results are summarized in Figure 1. The fact that the influence method performs well in terms of L^2 distance and yet poorly on the FIT for the same dataset highlights the fact that L^2 distance alone is not a sufficient metric to consider, especially if the main concern is privacy. Due to space constraints, the results for the L^2 test can be found in the appendix.

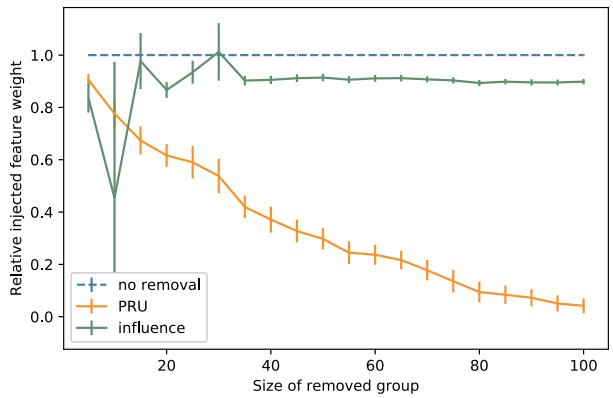

Figure 1: Yelp FIT experiment. We plot the mean of each metric ± the standard error of the mean. The PRU deletes the injected feature much more effectively and exhibits greater stability.

6.2 Logistic regression

We test the PRU on a synthetic data logistic regression experiment. The data $(x, y) \in \mathbb{R}^d \times \{0, 1\}$ were generated so that the logistic model is well-specified, i.e. there exists some parameter θ^* such that $\mathbb{P}(y = 1|x) = \sigma(x^\intercal \theta^*)$, where $\sigma(z) = 1/(1 + e^{-z})$.

For this experiment, we generate $n = 5000$ datapoints of dimension $d = 1000$. We compare the influence method to the PRU and leave analysis of the Newton step to the appendix. Refer to Tables 5 and 6. Consistent with linear regression and our theory, in the sparse data regime, the PRU performs very well in terms of both the L^2 metric and the FIT.

Table 5: Median FIT results for logistic regression over 10 trials. Due to space constraints, we report these figures with the IQR in the the appendix; the variation across trials was generally very small. For larger group sizes and sparse data, the PRU is able to completely remove the injected feature.

	$p = 0.5$	0.1	0.05
$k = 25$ (INF)	**0.82**	0.77	0.78
$k = 25$ (PRU)	0.86	**0.69**	**0.44**
$k = 50$ (INF)	0.81	0.82	0.82
$k = 50$ (PRU)	0.81	**0.48**	**0.02**
$k = 100$ (INF)	0.82	0.85	0.84
$k = 100$ (PRU)	**0.71**	**0.00**	**0.00**

Table 6: Median L^2 results for logistic regression over 10 trials. See the appendix for IQR. We examine the performance of each method for different group deletion sizes (k) and different levels of data sparsity (p). The results closely match the theory. For larger group and sparse data, PRU outperforms the influence method.

	$p = 0.5$	0.1	0.05
$k = 25$ (INF)	**0.85**	**0.77**	0.78
$k = 25$ (PRU)	0.86	0.80	**0.65**
$k = 50$ (INF)	0.85	0.83	0.82
$k = 50$ (PRU)	0.85	**0.69**	**0.20**
$k = 100$ (INF)	0.85	0.86	0.84
$k = 100$ (PRU)	**0.80**	**0.24**	**0.13**

7 Related Work

Most previous work on this topic has focused on specific classes of models. For example, Ginart et al. examined the problem of data deletion for clustering algorithms (Ginart et al., 2019). Tsai et al. use retraining with warm starts as a data deletion method for logistic regression, although they refer to the problem as decremental training (Tsai et al., 2014). Others such as Cauwenberghs et al. have studied the problem of decremental training for SVM models (Cauwenberghs and Poggio, 2000). Cao et al. consider a more general class of models and propose a solution using the statistical query framework for the problem of data deletion (which they refer to as machine unlearning);

their proposed method for adaptive SQ learning algorithms, such as gradient descent, is analogous to the aforementioned warm start method (Cao and Yang, 2015). Bourtoule et al. introduce a method called SISA (Sharded, Isolated, Sliced, and Aggregated) training, that minimizes the computational cost of retraining by taking advantage of sharding and caching operations during training (Bourtoule et al., 2019). Other previous approaches for machine unlearning are very closely related to the influence and Newton's methods. The method introduced by Monari and Dreyfus in (Monari and Dreyfus, 2000) is the same as the influence method with a different update step size. In the earlier work of Hansen and Larsen (Hansen and Larsen, 1996), their proposed update is simply a Newton step.

While our work has applications to privacy, it is distinct from previous research focusing on privacy. The differential privacy framework, for instance, provides a way to minimize the risks associated with belonging to a model's training set. However, the strong privacy guarantees offered by differential privacy often come at the cost of significantly reduced accuracy. In a setting where most users are not overly concerned about privacy and are willing to share data, the option to use a non-private model while allowing users to opt-out if they change their minds provides a useful middle ground. Drawing on the definition of differential privacy, the authors of (Guo et al., 2019) define a notion of ϵ-certified removal from machine learning models. They propose a modification of Newton's method for data deletion from linear models to satisfy this definition.

Our method's key advantage over previous work is that it is the first deletion algorithm for parametric models with a runtime linear in the data dimension and independent of the dataset size. This is a crucial development for modern high-dimensional ML.

8 Conclusion

We consider the problem of approximate data deletion from ML models, with a particular focus on linear and logistic regression. We develop a novel algorithm—the projective residual update (PRU)—with a computational cost which is linear in the dimension of the data, a substantial improvement over existing methods with quadratic dimension dependence. We also introduce a new metric for evaluating data removal from models—the feature injection test—a measure of the removal of the model's knowledge of a sensitive, highly predictive feature present in the data. Experiments on both real and synthetic data corroborate the theory. With any approximate deletion method, the accuracy of the approximation will decay as more deletion requests are processed. Extending our ideas to address this challenge is an important direction for future work.

Acknowledgements

JZ is supported by NSF CCF 1763191, NSF CAREER 1942926, NIH P30AG059307, NIH U01MH098953 and grants from the Silicon Valley Foundation and the Chan-Zuckerberg Initiative. MS was supported by a Qualcomm Fellowship. KC thanks ONR under N00014-20-1-2334 and a Google Faculty Fellowship for research support. We also thank the anonymous reviewers for their insightful comments.

References

Lucas Bourtoule, Varun Chandrasekaran, Christopher Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning, 2019.

Marc-Etienne Brunet, Colleen Alkalay-Houlihan, Ashton Anderson, and Richard Zemel. Understanding the origins of bias in word embeddings. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 803–811, 2019.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *IEEE Symposium on Security and Privacy*, pages 463–480, 2015.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium*, pages 267–284, 2019.

Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, 2000.

R. Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

Cynthia Dwork. Differential Privacy: A Survey of Results. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, pages 1–19, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-79228-4.

Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized Warfarin dosing. In *USENIX Security*, pages 17–32, 2014.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. volume 97 of *Proceedings of Machine Learning Research*, pages 2242–2251, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/ghorbani19c.html.

Antonio Ginart, Melody Guan, Gregory Valiant, and James Zou. Making AI forget you: Data deletion in machine learning. *arXiv:1907.05012*, 2019.

Ryan Giordano, Will Stephenson, Runjing Liu, Michael I. Jordan, and Tamara Broderick. A Swiss Army Infinitesimal Jackknife. *arXiv:1806.00550 [stat]*, June 2018. URL http://arxiv.org/abs/1806.00550. arXiv: 1806.00550.

Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens van der Maaten. Certified data removal from machine learning models, 2019.

Lars Kai Hansen and Jan Larsen. Linear unlearning for cross-validation. *Advances in Computational Mathematics*, 5(1):269–280, Dec 1996. ISSN 1572-9044. doi: 10.1007/BF02124747.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894, 2017.

Pang Wei Koh, Kai-Siang Ang, Hubert H. K. Teo, and Percy Liang. On the accuracy of influence functions for measuring group effects. *arXiv:1905.13289*, 2019.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014. URL http://arxiv.org/abs/1404.5997.

Gaétan Monari and Gérard Dreyfus. Withdrawing an example from the training set: An analytic estimation of its effect on a non-linear parameterised model. *Neurocomputing*, 35(1):195–201, Nov 2000. ISSN 0925-2312. doi: 10.1016/S0925-2312(00)00325-8.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Victor Y. Pan and Zhao Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, page 507–516, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130678. doi: 10.1145/301250.301389. URL https://doi.org/10.1145/301250.301389.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Autodiff Workshop*, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. ML-Leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *Proceedings of the 2019 Network and Distributed System Security Symposium (NDSS)*, 2019.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy*, pages 3–18, 2017.

Congzheng Song and Vitaly Shmatikov. Auditing data provenance in text-generation models. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD, pages 196–206, 2019a.

Congzheng Song and Vitaly Shmatikov. Overlearning reveals sensitive attributes. *arXiv:1905.11742*, 2019b.

Symantec Corporation. (2019). *U.S. Patent No. 10225277*. Verifying that the influence of a user data point has been removed from a machine learning classifier.

Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. Incremental and decremental training for linear classification. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD, pages 343–352, 2014.

Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. The secret revealer: Generative model-inversion attacks against deep neural networks. *arXiv preprint arXiv:1911.07135*, 2019.