Approximate Data Deletion from Machine Learning Models: Algorithms and Evaluations

Zachary Izzo [*1] Mary Anne Smart [*2] Kamalika Chaudhuri [2] James Zou [3]

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 For example, we may want to remove the influence of training points that might be out of date or outliers.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. Moreover there is no known efficient algorithm that exactly deletes data from most ML models. In this work, we evaluate several approaches for approximate data deletion from trained models. For the case of linear regression, we propose a new method with linear dependence on the feature dimension d, a significant gain over all existing methods which all have superlinear time dependence on the dimension. We also provide a new test for evaluating data deletion from linear models.

1. Introduction

Given a machine learning (ML) model that is trained on a dataset, there are many settings now 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. 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 machine learning models trained on the deleted data. These models may still 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.

Beyond Right to be Forgotten, data removal from ML is also useful when certain training points become outdated over time or are identified to be mistakes after training. The ability to remove the impact of such points can greatly improve the responsiveness and reliability of the ML model. Despite the fundamental importance of ML data deletion, this question is very much under-explored. The naive approach is to simply retrain the whole model on the remaining data. This can be prohibitively expensive, especially when deletion requests occur online and regularly (e.g. deletion on Facebook). Computationally efficient methods that can quickly post-process a trained model—without retraining from scratch—and remove some or most of the effects of the requested data would be tremendously useful both to comply with the regulations and for other ML applications.

A recent work formalized the data deletion problem and proposed an approach to efficiently delete data for K-means clustering (Ginart et al., 2019). While an important first step, this work did not address the question of how to efficiently delete data in a supervised model, and the technique developed there is tailor-made for clustering. In this paper, we investigate data deletion in supervised learning. We compare and evaluate several approaches for removing the impact of specified training data from a trained predictor. The deletion methods we investigate leverage recently developed approaches, such as influence functions, as well as a new approach we propose here based on carefully generating synthetic data. Given the complexity of this challenge, no single approach always works the best. Therefore, we characterize the computational and effectiveness tradeoffs

*Equal contribution. [1]Department of Mathematics, Stanford University, Stanford, CA, USA [2]Department of Computer Science and Engineering, UC San Diego, La Jolla, CA, USA [3]Department of Biomedical Data Science, Stanford University, Stanford, CA, USA. Correspondence to: James Zou <jamesz@stanford.edu>, Kamalika Chaudhuri <kamalika@cs.ucsd.edu>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML).

of the different deletion methods through both theoretical analysis and empirical experiments. We show that these more efficient removal methods, while not able to completely delete data, are several orders of magnitude faster than retraining the model from scratch.

Our contributions. Our first contribution is the development of a novel method for approximately removing datapoints from a trained model, dubbed the *projective residual update* (PRU), which computes the projection of the exact parameter update vector onto a particular low-dimensional subspace. Unlike existing approximate and exact deletion methods, the projective residual update has a runtime which scales only linearly in the dimension of the data; other methods have quadratic or higher dependence on the dimension.

Our second contribution is a rigorous evaluation of the advantages of several approaches to data deletion. There is very little published work on this topic and such a multi-approach evaluation fills an important gap. We conduct experiments on several synthetic and real datasets which showcase the benefits and shortcomings of the available methods. We also introduce a new metric for evaluating data removal from models—the feature injection test—which captures how well we can remove the model's "knowledge" of a sensitive, highly predictive feature present in the data.

2. Notation and 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. We will always assume that $n \gg d \gg k$. We will primarily work in the linear regression setting; in practice, this can be applied to neural networks by freezing all but the last layer, as is done in (Koh & Liang, 2017). Due to space constraints, the proofs of most lemmas and theorems are deferred to the appendices.

- $\theta \in \mathbb{R}^d$ 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.
- $\hat{y}_i = \theta^{\text{full}\intercal} x_i$ is the prediction of the full model on the i-th datapoint in the linear regression setting.
- $\hat{y}_i^{\setminus k} = \theta^{\setminus k \intercal} x_i$ 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.

3. Methods

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

Analytic formula from scratch 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 runtime of $O(nd^2)$.

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 more general settings, forming and inverting the new Hessian may take up to $O(d^3)$ 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 & 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:

$$\theta(w) \equiv \operatorname*{argmin}_\theta \sum_{i=1}^n w_i \ell(x_i, y_i; \theta). \quad (2)$$

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 & Liang, 2017) show that the linear approximation is given by

$$\theta^{\text{inf}} = \theta^{\text{full}} - [\nabla_\theta^2 L^{\text{full}}(\theta^{\text{full}})]^{-1} \nabla_\theta L^{\setminus k}(\theta^{\text{full}}). \quad (3)$$

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

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

EXACT	INFLUENCE	PROJECTIVE RESIDUAL
$O(kd^2)$	$O(d^2)$	$O(k^2 d)$

We summarize the asymptotic runtimes in table 1 alongside the runtime of our novel method, the projective residual update. (Since the Newton step with Sherman-Morrison formula is exact and has a strictly faster runtime than the naive method of retraining from scratch, we do not include the naive method in the table.)

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; the derivation can be found in the section 4.1. Critically, as in the influence function setting, 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 computational cost which will not be included in the runtime 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

$$\nabla_\theta L^{\{(x_i, \hat{y}_i^{\setminus k})\}_{i=1}^k}(\theta^{\text{full}}) = \left(\sum_{i=1}^k x_i x_i^\intercal\right)(\theta^{\text{full}} - \theta^{\setminus k}). \quad (4)$$

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. We give details on the LKO method in

Algorithm 1 The projective residual update

```
1: procedure ResidualUpdate($X, Y, H, \theta^{\text{full}}, k$)
2:     $\hat{y}'_1, \ldots, \hat{y}'_k \leftarrow$ LKO($X, Y, H, \theta^{\text{full}}, k$)
3:     $S^{-1} \leftarrow$ PseudoInv($\sum_{i=1}^{k} x_i x_i^\intercal$)
4:     $\nabla L \leftarrow \sum_{i=1}^{k} (\theta^{\text{full}\intercal} x_i - \hat{y}'_i) x_i$   ▷ Loss gradient on
5:                                                   ▷ synthetic points
6:     return $\theta^{\text{full}} -$ FastMult($S^{-1}, \nabla L$)
7: end procedure
```

section 4.1, as well as a brief discussion of the subroutines PseudoInv, and FastMult. A full discussion can be found in the appendix.

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

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

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, the projective residual update is optimal among gradient-based updates, regardless of choice of step size or number of gradient steps, when we restrict ourselves to only look at the gradient on points of the form $(x_i, \tilde{y}_i)$, $i = 1, \ldots, k$.

As we will see later, this characterization of the projective residual update guarantees that it performs well for a particular deletion metric (the feature injection test, introduced in section 5.2) under certain sparsity conditions on the data. We discuss this further in section 6.2.

4.1. The LKO subroutine

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 what follows, $X_{1:k}$ 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.

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

For simplicity, we prove Theorem 2 for the case of ordinary least squares. We generalize the result to weighted, regularized least squares in the appendix. We also note that by extending to weighted linear regression, the PRU can be applied to logistic regression.

Algorithm 2 Leave-k-out predictions

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

Proof of Theorem 2. We make use of the analytic form of the parameters for least squares linear regression. Given a dataset $\{(x_i, y_i)\}_{i=1}^{n}$, we have $\theta^{\text{full}} = (X^\intercal X)^{-1} XY$, with X, Y defined as in section 2. The predictions for the fitted model on the dataset are then given by

$$\hat{Y} = X\theta^{\text{full}} = \underbrace{(X(X^\intercal X)^{-1}X^\intercal)}_{H} Y, \tag{5}$$

where $H = X(X^\intercal X)^{-1}X^\intercal$ is the so-called hat matrix. As previously mentioned, we assume that we already have access to H after the model has been trained on the full dataset.

Next, observe that

$$\theta^{\setminus k} = \underset{\theta}{\text{argmin}} \left[\sum_{i=1}^{k} (\theta^\intercal x_i - \hat{y}_i^{\setminus k})^2 + \sum_{i=k+1}^{n} (\theta^\intercal x_i - y_i)^2 \right]$$

since $\theta^{\setminus k}$ minimizes both sums individually. It follows from equation (5) that $HY' = \hat{Y}_{\setminus k}$, where $Y' = (\hat{y}_1^{\setminus k}, \ldots, \hat{y}_k^{\setminus k}, y_{k+1}, \ldots, y_n)^\intercal$ and $\hat{Y}_{\setminus k} = (\hat{y}_1^{\setminus k}, \ldots, \hat{y}_n^{\setminus k})^\intercal$.

This relation $HY' = \hat{Y}_{\setminus k}$ allows us to derive a system of linear equations between $\hat{y}_i^{\setminus k}$ for $i = 1, \ldots, k$. Namely, if we define $r_i = y_i - \hat{y}_i$, $r = (r_1, \ldots, r_k)^\intercal$, $r_i^{\setminus k} = y_i - \hat{y}_i^{\setminus k}$, and $r^{\setminus k} = (r_1^{\setminus k}, \ldots, r_k^{\setminus k})^\intercal$, we have

$$r_i^{\setminus k} = \frac{r_i + \sum_{j \neq i} h_{ij} r_j^{\setminus k}}{1 - h_{ii}}, \tag{6}$$

where h_{ij} are the entries of H. Vectorizing equation (6) and solving yields

$$r^{\setminus k} = (I - T)^{-1} \left(\tfrac{r_1}{1-h_{11}}, \;\ldots,\; \tfrac{r_k}{1-h_{kk}}\right)^\intercal, \tag{7}$$

where $T_{ij} = \mathbf{1}\{i \neq j\}\frac{h_{ij}}{1-h_{jj}}$. Since this is a system of k linear equations in k unknowns, we can solve it in time $O(k^3)$ via simple Gaussian elimination. The values $\hat{y}_i^{\setminus k}$ can then be easily recovered in an additional $O(k)$ time by noting that $\hat{y}_i^{\setminus k} = y_i - r_i^{\setminus k}$. □

4.2. The PSEUDOINV and FASTMULT subroutines

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 & 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. The pseudoinverse has the form $\sum_{i=1}^{k} \lambda_i^{-1} v_i v_i^\intercal$. Rather than computing the full matrix, then multiplying by it, we can quickly compute a matrix-vector product via careful parenthesizing: $\left(\sum_{i=1}^{k} \lambda_i^{-1} v_i v_i^\intercal\right) u = \sum_{i=1}^{k} \lambda_i^{-1} (v_i^\intercal u) v_i$. For a more detailed explanation, refer to the appendix.

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

Theorem 3 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 $X^\intercal X + \lambda^2 x_i x_i^\intercal$ deviates more and more from the LOO Hessian $X^\intercal X$, causing this drop in performance. Since the PRU is guaranteed to make some improvement, for a large enough outlier the PRU will perform better than the influence method in terms of parameter distance.

5. Evaluation Metrics

5.1. 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.

However, L^2 distance alone does not tell us how effectively individual information was removed. If the model is a deep neural network with a non-convex loss function, different settings of θ, potentially with large L^2 distance, may express the same function. Even for convex loss functions, L^2 distance does not provide an intuitive understanding of how much information has been removed from the model. Therefore, in addition to the L^2 distance, we develop a second metric: the feature injection test.

5.2. Feature injection test

The rationale behind this 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. 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 that is 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.

6. Experiments

We now verify our theoretical guarantees and compare the accuracy and speed of the methods experimentally.

6.1. Experimental Methodology

6.1.1. DATASETS

Synthetic We first describe the general procedure for constructing the dataset. We modify this procedure slightly between experiments, and we will discuss the modifications after the general procedure.

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 our experiments, we set the noise level $\sigma^2 = 1$; for reasonable values of σ^2 this parameter does not play a large role in the outcome of the experiments. For all of the synthetic datasets, we take $n = 10d$.

Table 2. Mean runtimes for each method as a fraction of full retraining runtime. (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.

	$d = 1000$	$d = 1500$	$d = 2000$	$d = 2500$	$d = 3000$
$k = 1$ (INF)	0.0085	0.0053	0.0041	0.0036	0.0028
$k = 1$ (PRU)	**0.0062**	**0.0017**	**0.0008**	**0.0004**	**0.0003**
$k = 2$ (INF)	0.0088	0.0053	0.0042	0.0035	0.0034
$k = 2$ (PRU)	**0.0079**	**0.0023**	**0.0012**	**0.0006**	**0.0004**
$k = 3$ (INF)	0.0089	0.0052	0.0042	0.0032	0.0029
$k = 3$ (PRU)	0.0089	**0.0027**	**0.0014**	**0.0008**	**0.0005**
$k = 4$ (INF)	**0.0090**	0.0053	0.0042	0.0033	0.0028
$k = 4$ (PRU)	0.0096	**0.0029**	**0.0015**	**0.0008**	**0.0005**
$k = 5$ (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
$k = 100$ (INF)	**0.0141**	**0.0080**	**0.0056**	**0.0041**	**0.0032**
$k = 100$ (PRU)	0.2005	0.0716	0.0400	0.0231	0.0157

For the runtime experiment, no modifications are made to the general setup. We vary the dimension d between $d = 1000$ and $d = 3000$.

For the L^2 experiment, we simulate the setting described in section 4.3 by scaling up the points to be deleted by a factor λ. We fix the dimension $d = 1500$ and vary the multiplier λ between $\lambda = 1$ (no outlier) and $\lambda = 300$ (large outlier).

The modifications for the feature injection test are slightly more involved. We construct sparse data with three key properties: (1) only the deleted feature vectors x_i, $i = 1, \dots, k$ have nonzero d-th entry (this is the "special feature"); (2) the deleted feature vectors all lie on the same low-dimensional subspace; (3) the response for the deleted points is perfectly correlated with the special feature. While assumption (2) may seem restrictive, it is not unnatural. We observe a similar phenomenon in the Yelp dataset, since a single user is likely to reuse the same words; in this case, the user's feature vectors will have shared sparse support. We again fix the dimension $d = 1500$ and vary the sparsity p between $p = 0.5$ and $p = 0.05$, and the group deletion size k between $k = 1$ and $k = 150$.

More details for the construction of all of the synthetic datasets are given in the appendix.

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.

CIFAR-100 We briefly note that we have experimental results on a subset of CIFAR-100 (Krizhevsky, 2009). A detailed discussion of the experimental setup and results can be found in the appendix.

6.1.2. Baselines

We focus on comparing the following four methods: no removal, the influence method (influence), the projective residual update (PRU), and exact retraining. No retraining will give us a baseline for measuring L^2 distance and the unique feature weight; and exact retraining will give us a baseline for runtimes.

6.1.3. Setup

On the Yelp dataset, we run linear regression with labels in $\{1, \text{-}1\}$, using an L^2 regularization parameter of 10^{-4}. 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. Below we describe the deletion procedures for our experiments.

L^2 Parameter Distance For the synthetic dataset, we select outlier datapoints (points whose feature vector has large norm) for deletion in order to illustrate the results of section 4.3. For the Yelp dataset, we select points at random for deletion.

Feature Injection Test The process for the feature injection test is similar. For the Yelp dataset, we randomly select points from the positive class for deletion. In each round of the experiment, as we increase the size of group to be deleted from k to $k + 5$, we select five additional points at random, rather than re-selecting all $k + 5$ points. Then we set the injected feature equal to one for the selected points before we delete them. If the new model after deletion assigns a weight of zero to the injected feature, this indicates that the injected signal has been effectively removed.

6.2. Results

Synthetic 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 will 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 feature injection test, 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. We refer to the cases indicated with two asterisks in table 3. For $k = 50, p = 0.05$, influence had a standard error of 1.35; for $k = 100, p = 0.5$, influence had a standard error of 3.28. We observe similarly unstable behavior on the Yelp dataset, discussed below.

For the L^2 metric, refer to table 4. The influence method outperforms PRU for deleting "typical" points (when $\lambda = 1$,

Table 3. Feature injection test on synthetic data. The special weight is given as fraction of baseline weight. Results are for $d = 1500$ for various group sizes (k) and sparsity values (p). See below for a discussion of the standard errors of the methods, as well as the notable values (indicated by asterisks). The baseline weights to which we compare can be found in the appendix.

	$p = 0.5$	$p = 0.25$	$p = 0.1$	$p = 0.05$
$k = 1$ (INF)	**0.98**	0.99	0.99	0.98
$k = 1$ (PRU)	0.99	0.99	0.99	0.98
$k = 5$ (INF)	1.08	1.09	0.99	1.01
$k = 5$ (PRU)	**0.99**	**0.98**	**0.96**	**0.93**
$k = 50$ (INF)	**0.87**	**0.84**	0.97	2.32^{**}
$k = 50$ (PRU)	0.93	0.86	**0.67**	**0.35**
$k = 100$ (INF)	3.65^{**}	0.76	0.92	0.98
$k = 100$ (PRU)	**0.90**	**0.72**	**0.32**	**0.00***
$k = 150$ (INF)	0.85	0.95	0.98	0.99
$k = 150$ (PRU)	**0.79**	**0.59**	**0.02**	**0.00***

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.

Yelp Since the Yelp dataset does not have large outliers, the influence method outperforms the projective residual update in terms of L^2 distance; yet the projective residual update actually performs better on the feature injection test, at least for large groups; the influence method fails to remove the injected signal.

6.3. Discussion

Deletion effectiveness On the L^2 distance metric, the influence method outperforms the projective residual update on the Yelp dataset. On the feature injection test, however, the PRU does much better than the influence method, especially for large group sizes. The fact that the influence method performs well in terms of L^2 distance and yet poorly on the feature injection test 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.

Practical guidelines The experimental and theoretic results offer insight into when to use one approximate method over the other. If one is interested in minimizing the distance to the exact parameters and the data do not contain large outliers, then the influence method tends to perform better. When removing outlier points or attempting to remove the effect of a specific feature, the PRU should be used as it is both more effective and more stable, especially when the

Table 4. L^2 test on synthetic data. 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 3).

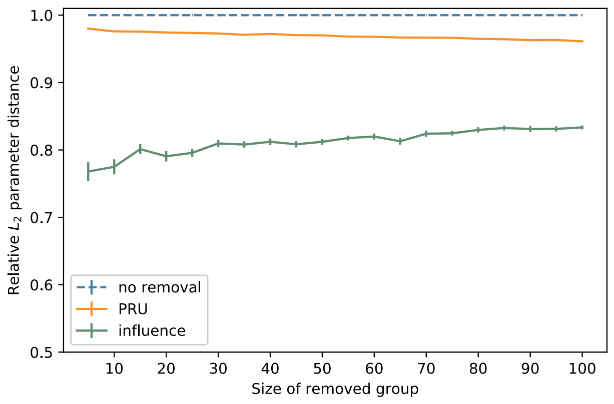
	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$
$k = 1$ (INF)	**0.73**	**0.96**	0.99
$k = 1$ (PRU)	0.97	0.97	**0.97**
$k = 5$ (INF)	**0.38**	0.93	0.99
$k = 5$ (PRU)	0.92	**0.92**	**0.92**
$k = 10$ (INF)	**0.27**	0.92	0.99
$k = 10$ (PRU)	0.89	**0.90**	**0.90**
$k = 50$ (INF)	**0.16**	0.91	0.99
$k = 50$ (PRU)	0.88	**0.88**	**0.88**
$k = 100$ (INF)	**0.14**	0.90	0.99
$k = 100$ (PRU)	0.88	**0.88**	**0.88**

removed data have a low-dimensional structure and strong dependence on the specific feature.

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 & Poggio, 2000). Cao et al. consider a more general class of models and propose a solution using the statistical query framework for the problem they dub ”machine unlearning”; their proposed method for adaptive SQ learning algorithms, such as gradient descent, is analogous to the aforementioned warm start method (Cao & 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).

Since this work is partially motivated by privacy concerns, it is worth discussing the relationship between efficient data removal and differential privacy. Differential privacy 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, (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.



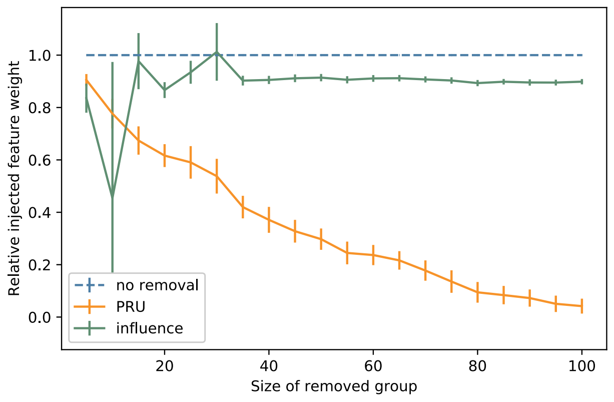


Figure 1. **Top:** Yelp L^2 experiment, **Bottom:** Yelp special feature experiment.

8. Conclusion

We consider the problem of approximate data deletion from machine learning models, with a particular focus on linear regression. We develop a novel algorithm—the projective residual update (PRU)—with a runtime 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. We conduct extensive results on both real and synthetic data, verifying our theoretical results empirically and showcasing the advantages and shortcomings of various methods. Our experiments also highlight the efficacy of the PRU with respect to the feature injection test. We leave the problem of extending these algorithms and establishing theoretical guarantees for the case of deep neural networks to future work.

The code for replicating our experiments can be found at https://github.com/zleizzo/datadeletion.

9. Acknowledgements

J.Z. is supported by NSF CCF 1763191, NIH R21 MD012867-01, NIH P30AG059307, and grants from the Silicon Valley Foundation and the Chan-Zuckerberg Initiative. K.C. and M.S. thank ONR under N00014-16- 1-261 and UC Lab Fees under LFR 18-548554 for support; M.S. was also supported by a Qualcomm Fellowship.

References

Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C., Jia, H., Travers, A., Zhang, B., Lie, D., and Papernot, N. Machine unlearning, 2019.

Brunet, M.-E., Alkalay-Houlihan, C., Anderson, A., and Zemel, R. 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*, pp. 803–811, 2019.

Cao, Y. and Yang, J. Towards making systems forget with machine unlearning. In *IEEE Symposium on Security and Privacy*, pp. 463–480, 2015.

Carlini, N., Liu, C., Erlingsson, Ú., Kos, J., and Song, D. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium*, pp. 267–284, 2019.

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

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

Dua, D. and Graff, C. UCI machine learning repository, 2017.

Dwork, C. Differential Privacy: A Survey of Results. In Agrawal, M., Du, D., Duan, Z., and Li, A. (eds.), *Theory and Applications of Models of Computation*, pp. 1–19, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-79228-4.

Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., and Ristenpart, T. Privacy in pharmacogenetics: An end-to-end case study of personalized Warfarin dosing. In *USENIX Security*, pp. 17–32, 2014.

Ginart, A., Guan, M., Valiant, G., and Zou, J. Making AI forget you: Data deletion in machine learning. *arXiv:1907.05012*, 2019.

Giordano, R., Stephenson, W., Liu, R., Jordan, M. I., and Broderick, T. A Swiss Army Infinitesimal Jackknife. *arXiv:1806.00550 [stat]*, June 2018. URL http://arxiv.org/abs/1806.00550. arXiv: 1806.00550.

Guo, C., Goldstein, T., Hannun, A., and van der Maaten, L. Certified data removal from machine learning models, 2019.

Koh, P. W. and Liang, P. 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*, pp. 1885–1894, 2017.

Koh, P. W., Ang, K.-S., Teo, H. H. K., and Liang, P. On the accuracy of influence functions for measuring group effects. *arXiv:1905.13289*, 2019.

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

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

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

Pan, V. Y. and Chen, Z. Q. The complexity of the matrix eigenproblem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, pp. 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.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS Autodiff Workshop*, 2017.

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

Salem, A., Zhang, Y., Humbert, M., Berrang, P., Fritz, M., and Backes, M. 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.

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

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

Song, C. and Shmatikov, V. 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.

Tsai, C.-H., Lin, C.-Y., and Lin, C.-J. Incremental and decremental training for linear classification. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD, pp. 343–352, 2014.

Zhang, Y., Jia, R., Pei, H., Wang, W., Li, B., and Song, D. The secret revealer: Generative model-inversion attacks against deep neural networks. *arXiv preprint arXiv:1911.07135*, 2019.

A. Detailed algorithm description and proof of Theorem 1

Algorithm 3 Compute the pseudoinverse of $\sum_{i=1}^{k} x_i x_i^\intercal$

```
1: procedure PSEUDOINV($x_1, \dots, x_k$)
2:     $c_1, \dots, c_k, u_1, \dots, u_k \leftarrow$ GRAM-SCHMIDT($x_1, \dots, x_k$)    ▷ $x_i = \sum_{j=1}^{k} c_{ij} u_j$.
3:     $C \leftarrow \sum_{i=1}^{k} c_i c_i^\intercal$
4:     $\lambda_1, a_1, \dots, \lambda_k, a_k \leftarrow$ EIGENDECOMPOSE($C$)    ▷ Eigenvalue $\lambda_i$ has corresponding eignevector $a_i$
5:     for $i = 1, \dots, k$ do
6:         $v_i \leftarrow \sum_{j=1}^{k} a_{ij} u_j$
7:     end for
8:     return $\lambda_1^{-1}, v_1, \dots, \lambda_k^{-1}, v_k$
9: end procedure
```

Algorithm 4 Fast multiplication by pseudoinverse

```
1: procedure FASTMULT($S^{-1}, \nabla L$)    ▷ $S^{-1}$ must be given in its low-rank form $S^{-1} = \sum_{i=1}^{k} \lambda_i^{-1} v_i v_i^\intercal$
2:     return $\sum_{i=1}^{k} (\lambda_i^{-1} (v_i^\intercal \nabla L)) v_i$    ▷ Compute according to the specified parenthesization
3: end procedure
```

We take a gradient step in the direction specified by the synthetic LKO points $(x_i, \hat{y}_i^{\setminus k})$, $i = 1, \dots, k$. That is, we update

$$\theta^{\text{res}} = \theta^{\text{full}} - \alpha \sum_{i=1}^{k} (\theta^{\text{full}\intercal} x_i - \hat{y}_i^{\setminus k}) x_i. \tag{8}$$

Ordinarily, the parameter α is a scalar which specifies the step size. For our purposes, we will replace α with a "step matrix" A.

Proof of Theorem 1. Recalling that $\hat{y}_i^{\setminus k} = \theta^{\setminus k \intercal} x_i$, we can rewrite equation (8):

$$\begin{aligned}\theta^{\text{full}} - A \sum_{i=1}^{k} (\theta^{\text{full}\intercal} x_i - \hat{y}_i^{\setminus k}) x_i &= \theta^{\text{full}} - A \sum_{i=1}^{k} (\theta^{\text{full}\intercal} x_i - \theta^{\setminus k \intercal} x_i) x_i \\ &= \theta^{\text{full}} - A \left(\sum_{i=1}^{k} x_i x_i^\intercal \right) (\theta^{\text{full}} - \theta^{\setminus k}).\end{aligned} \tag{9}$$

Let $B = \sum_{i=1}^{k} x_i x_i^\intercal$. Note that range$(B) =$ span$\{x_1, \dots, x_k\} \triangleq V_k$. Due to the form that B has, we can efficiently compute its eigendecomposition $B = V \Lambda V^\intercal$, where $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_k, 0, \dots, 0)$, $\lambda_1, \dots, \lambda_k$ are the nonzero eigenvalues of B, and $V V^\intercal = I$. We then define

$$A = V \Lambda^\dagger V^\intercal, \qquad \Lambda^\dagger = \text{diag}(\lambda_1^{-1}, \dots, \lambda_k^{-1}, 0, \dots, 0). \tag{10}$$

This choice of A gives us $AB = \sum_{i=1}^{k} v_i v_i^\intercal = \text{proj}_{V_k}$, and therefore the update (9) is equivalent to

$$\theta^{\text{full}} + \text{proj}_{V_k} (\theta^{\setminus k} - \theta^{\text{full}}). \tag{11}$$

This establishes the first claim in Theorem 1. It remains to perform the runtime calculation. We analyze the runtime of the algorithm by breaking it down into several submodules.

Step 1: Computing $\hat{y}_i^{\setminus k}$, $i = 1, \dots, k$

By Theorem 2, this step can be accomplished in $O(k^3)$ time.

Step 2: Finding the eigendecomposition of $\sum_{i=1}^{k} x_i x_i^\intercal$

We will show that this step can be completed in $O(k^2 d)$ time. We compute the eigendecomposition of $B \equiv \sum_{i=1}^{k} x_i x_i^\intercal$ as follows.

I. Perform Gram-Schmidt on $x_1, \dots, x_k$ to recover $u_1, \dots, u_k$ and coefficients c_{ij}. Runtime: $O(k^2 d)$.

(a) In the i-th step, we set $w_i = x_i - ((x_i^\intercal u_1)u_1 + \cdots + (x_i^\intercal u_{i-1})u_{i-1})$, followed by $u_i = w_i / \|w_i\|$. Naively computing the dot products, scalar-vector products, and vector sums for step i takes $O(id)$ time. Summing over the steps, the total time to perform Gram-Schmidt is $\sum_{i=1}^k O(id) = O(k^2 d)$.

(b) From the i-th step of Gram-Schmidt, we see that

$$x_i = (x_i^\intercal u_1)u_1 + \cdots + (x_i^\intercal u_{i-1})u_{i-1} + \|w_i\| u_i$$

$$\therefore c_{ij} = \begin{cases} x_i^\intercal u_j, & 1 \le j < i \\ \|w_i\|, & j = i \\ 0, & j > i \end{cases}$$

We can store these coefficients as we compute them during the Gram-Schmidt procedure without increasing the asymptotic time complexity of this step.

II. Eigendecompose the $k \times k$ matrix $C = \sum_{i=1}^k c_i c_i^\intercal$ and recover the eigendecomposition of B. Runtime: $O(k^2 d)$.

(a) We claim that the first k eigenvalues of B are identical to the eigenvalues of C, and that the eigenvectors of B can easily be recovered from the eigenvectors of C. In particular, if $a_1, \dots, a_k \in \mathbb{R}^k$ are the eigenvectors of C, then $v_i = a_{i1}u_1 + \ldots + a_{ik}u_k$ is the i-th eigenvector of B.
To see this, note that $R(B) = \text{span}\{x_1, \dots, x_k\}$, so any eigenvector for a nonzero eigenvalue of B must be in the span of the x_i. Since $u_1, \dots, u_k$ have the same span as the x_i, if v is an eigenvector for B with nonzero eigenvalue λ, we can write $v = b_1 u_1 + \cdots + b_k u_k$. Let $b = (b_1, \dots, b_k)^\intercal \in \mathbb{R}^k$. We can also rewrite $B = \sum_{i=1}^k x_i x_i^\intercal = \sum_{i,j,\ell=1}^k c_{ij} c_{i\ell} u_j u_\ell^\intercal$. Combining these facts yields

$$\begin{aligned} Bv &= \sum_{i,j,\ell=1}^k b_\ell c_{i\ell} c_{ij} u_j \\ &= \sum_{i,j=1}^k (c_i^\intercal b) c_{ij} u_j \\ &= \lambda b_1 u_1 + \cdots + \lambda b_k u_k. \end{aligned}$$

Since the u_js are linearly independent, we can equate coefficients. Doing so shows that $\lambda b_j = \sum_{i=1}^k (c_i^\intercal b) c_{ij}$ for all $j = 1, \dots, k$. Vectorizing these equations, we have that

$$Cb = \sum_{i=1}^k c_i (c_i^\intercal b) = \lambda b.$$

This chain of equalities holds in reverse order as well, so we conclude that v is an eigenvector for B with nonzero eigenvalue λ iff b is an eigenvector for C with eigenvalue λ. Since we know that the remaining eigenvalues of B are 0, it suffices to find an eigendecomposition of C. Forming C takes $O(k^3)$ time, and finding its eigendecomposition can be done (approximately) in $O(k^3)$ time, see (Pan & Chen, 1999). Finally, converting each eigenvector a_i for C into an eigenvector for B takes $O(kd)$ time (we set $v_i = a_{i1}u_1 + \cdots + a_{ik}u_k$), so converting all k of them takes $O(k^2 d)$ time.

(b) Since we know B is rank k, the remaining eigenvalues are 0 and any orthonormal extension of the orthonormal eigenvectors $v_1, \dots, v_k$ computed in step 2 will suffice to complete an orthonormal basis of eigenvectors for $\mathbb{R}^d$. Let $v_{k+1}, \dots, v_d$ be any such extension. This gives us a complete orthonormal basis of eigenvectors $v_1, \dots, v_d$ for $\mathbb{R}^d$ with associated eigenvalues $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_k > \lambda_{k+1} = \ldots = \lambda_d = 0$. We now define A as in equation (10). Observe that since $(\Lambda^\dagger)_{ii} = 0$ for $i > k$, we can compute A without needing to know the values of $v_{k+1}, \dots, v_d$:

$$A = \sum_{i=1}^k \lambda_i^{-1} v_i v_i^\intercal. \tag{12}$$

Step 3: Performing the update

We will show that this step can be completed in $O(kd)$ time. Recall the form of the projective residual update:

$$\theta^{\text{res}} = \theta^{\text{full}} - A \sum_{i=1}^{k} (\theta^{\text{full}\intercal} x_i - \hat{y}_i') x_i.$$

I. Form the vector $\nabla L = \sum_{i=1}^{k} (\theta^{\text{full}\intercal} x_i - \hat{y}_i') x_i$. (This is the gradient of the loss on the synthetic datapoints.) Runtime: $O(kd)$.

II. Compute the step $A\nabla L$. Runtime: $O(kd)$.

   (a) Rather than performing the computationally expensive operations of forming the matrix A, then doing a $d \times d$ matrix-vector multiplication, we use the special form of A. Namely, we have

$$\begin{aligned} A\nabla L &= \left(\sum_{i=1}^{k} \lambda_i^{-1} v_i v_i^\intercal \right) s \\ &= \sum_{i=1}^{k} (\lambda_i^{-1} (v_i^\intercal s)) v_i. \end{aligned} \tag{13}$$

   (b) Each term in the summand (13) can be computed in $O(d)$ time, so we can compute the entire sum in $O(kd)$ time.

III. Update $\theta^{\text{res}} = \theta^{\text{full}} - A\nabla L$. Runtime: $O(d)$.

Since we have assumed $k \le d$, the total runtime of the algorithm is therefore $O(k^3) + O(k^2 d) + O(kd) = O(k^2 d)$ as desired. □

Note that the crucial step of computing the exact leave-k-out predicted y-values may vary depending on the the specific instance of least squares we found ourselves in (e.g. with or without regularization or weighting, see Appendix D), but the rest of the algorithm remains exactly the same.

B. Performance analysis for outlier removal

In this section we prove Theorem 3. We also quantify the behavior of the true step $\theta^{\text{full}} - \theta^{\setminus 1}$ as the outlier size λ grows.

Proposition 4. *Let D^{full} be as in Theorem 3. As $\lambda \to \infty$, $\theta^{full} - \theta^{\setminus 1} \to C\hat{\Sigma}^{-1} x_1$, where $\hat{\Sigma}$ is the empirical covariance matrix for the dataset $D^{\setminus 1}$ and C is a (data-dependent) scalar constant.*

Proof. Departing slightly from the notation in section 2, let X and Y denote the feature matrix and response vector, respectively, for the dataset $D^{\setminus 1}$. The exact values of θ^{full} and $\theta^{\setminus 1}$ are then given by

$$\begin{aligned} \theta^{\setminus 1} &= (X^\intercal X)^{-1} XY \\ \theta^{\text{full}} &= (X^\intercal X + \lambda^2 x_1 x_x^\intercal)^{-1} (XY + \lambda^2 y_1 x_1). \end{aligned}$$

We can expand the expression for θ^{full} with the Sherman-Morrison formula:

$$\theta^{\text{full}} = \left[(X^\intercal X)^{-1} - \frac{\lambda^2 (X^\intercal X)^{-1} x_1 x_1^\intercal (X^\intercal X)^{-1}}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1} \right] \cdot (XY + \lambda^2 y_1 x_1). \tag{14}$$

From equation (14), we see that the actual step is

$$\begin{aligned} \theta^{\text{full}} - \theta^{\setminus 1} &= \lambda^2 y_1 (X^\intercal X)^{-1} x_1 - \frac{\lambda^2 (X^\intercal X)^{-1} x_1 x_1^\intercal (X^\intercal X)^{-1}}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1} (XY + \lambda^2 y_1 x_1) \\ &= (X^\intercal X)^{-1} \left(y_1 \underbrace{\lambda^2 \left[I - \frac{\lambda^2 x_1 x_1^\intercal (X^\intercal X)^{-1}}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1} \right] x_1}_{\text{(I)}} - \underbrace{\frac{\lambda^2 x_1 x_1^\intercal (X^\intercal X)^{-1}}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1} XY}_{\text{(II)}} \right). \end{aligned} \tag{15}$$

Let us analyze the behavior of the terms (I) and (II) in equation (15) as $\lambda \to \infty$. Term (II) is straightforward: the λ^2 terms dominate both the numerator and the denominator, so we have

$$\text{(II)} \longrightarrow \frac{x_1 x_1^\intercal (X^\intercal X)^{-1} XY}{x_1^\intercal (X^\intercal X)^{-1} x_1} = \frac{x_1^\intercal (X^\intercal X)^{-1} XY}{x_1^\intercal (X^\intercal X)^{-1} x_1} x_1 \qquad \text{as } \lambda \to \infty.$$

The term (I) is slightly more delicate, since the first-order behavior of (I) without multiplication by λ^2 tends to 0; however, the multiplication by λ^2 means that this term does not vanish. Observing that (I) can be rewritten as

$$\lambda^2 \left[1 - \frac{\lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1} \right] x_1,$$

we have reduced our analysis of (I) to determining the leading order behavior of a function of the form

$$f(\lambda) \equiv \lambda^2 \left[1 - \frac{c\lambda^2}{1 + c\lambda^2} \right]. \tag{16}$$

(In our case, $c = x_1^\intercal (X^\intercal X)^{-1} x_1$.) A Taylor expansion of (16) shows that $f(\lambda) = c^{-1} + O(\lambda^{-2})$, and thus we have

$$\text{(I)} \longrightarrow \frac{x_1}{x_1^\intercal (X^\intercal X)^{-1} x_1} \qquad \text{as } \lambda \to \infty.$$

Substituting the limits of (I) and (II) into equation (15), we see that

$$\begin{aligned} \theta^{\text{full}} - \theta^{\setminus 1} &\to (X^\intercal X)^{-1} \left[\frac{y_1}{x_1^\intercal (X^\intercal X)^{-1} x_1} x_1 - \frac{x_1^\intercal (X^\intercal X)^{-1} XY}{x_1^\intercal (X^\intercal X)^{-1} x_1} x_1 \right] \\ &= \underbrace{\frac{y_1 - x_1^\intercal (X^\intercal X)^{-1} XY}{x_1^\intercal (X^\intercal X)^{-1} x_1}}_{C'} (X^\intercal X)^{-1} x_1. \end{aligned} \tag{17}$$

The result follows by multiplying and dividing (17) by a factor of n (so $C = C'/n$ and the other factor of n gets pulled into $(X^\intercal X)^{-1}$ to yield $\hat{\Sigma}^{-1}$). □

Proof of Theorem 3. We will analyze $\theta^{\text{inf}} - \theta^{\setminus 1}$ and show that the limit of this difference is the same as that of $\theta^{\text{full}} - \theta^{\setminus 1}$ as $\lambda \to \infty$; it immediately follows that $\theta^{\text{inf}} \to \theta^{\text{full}}$. By the exactness of the Newton update for linear regression, we have

$$\theta^{\setminus 1} = \theta^{\text{full}} + (X^\intercal X)^{-1} \lambda^2 (\theta^{\text{full}\intercal} x_1 - y_1) x_1.$$

By definition, the influence parameters are given by

$$\theta^{\text{inf}} = \theta^{\text{full}} + (X^\intercal X + \lambda^2 x_1 x_1^\intercal)^{-1} \lambda^2 (\theta^{\text{full}\intercal} x_1 - y_1) x_1.$$

Subtracting these two expressions yields

$$\theta^{\text{inf}} - \theta^{\setminus 1} = \lambda^2 (\theta^{\text{full}\intercal} x_1 - y_1) \cdot [(X^\intercal X + \lambda^2 x_1 x_1^\intercal)^{-1} - (X^\intercal X)^{-1}] x_1. \tag{18}$$

We analyze the terms in the RHS of (18) separately.

First, note that by the Sherman-Morrison formula, we have

$$\begin{aligned} [(X^\intercal X + \lambda^2 x_1 x_1^\intercal)^{-1} - (X^\intercal X)^{-1}] x_1 &= \frac{-\lambda^2 (X^\intercal X)^{-1} x_1 x_1^\intercal (X^\intercal X)^{-1}}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1} x_1 \\ &= \frac{-\lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1} (X^\intercal X)^{-1} x_1 \qquad (19) \\ &\to -(X^\intercal X)^{-1} x_1. \qquad (20) \end{aligned}$$

Equation (20) follows since the numerator and denominator of (19) have the same leading order behavior in λ.

Next, we analyze the term $\theta^{\text{full}\intercal} x_1 - y_1$. We begin by substituting the expression for θ^{full} and once more applying the Sherman-Morrison formula:

$$\begin{aligned}
x_1^\intercal \theta^{\text{full}} - y_1 &= x_1^\intercal (X^\intercal X + \lambda^2 x_1 x_1^\intercal)^{-1}(XY + \lambda^2 x_1 y_1) - y_1 \\
&= x_1^\intercal \left[(X^\intercal X)^{-1} - \frac{\lambda^2 (X^\intercal X)^{-1} x_1 x_1^\intercal (X^\intercal X)^{-1}}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1}\right](XY + \lambda^2 x_1 y_1) - y_1 \\
&= x_1^\intercal (X^\intercal X)^{-1} \left[\underbrace{\left(I - \frac{\lambda^2 x_1 x_1^\intercal (X^\intercal X)^{-1}}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1}\right) XY}_{(i)} + y_1 \underbrace{\left(\lambda^2 \left[I - \frac{\lambda^2 x_1 x_1^\intercal (X^\intercal X)^{-1}}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1}\right] x_1\right)}_{(ii)}\right] - y_1
\end{aligned} \tag{21}$$

We rearrange (i) and (ii) and then Taylor expand:

$$\begin{aligned}
(i) &= XY - \frac{\lambda^2 x_1^\intercal (X^\intercal X)^{-1} XY}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1} x_1 \\
&= XY - \frac{x_1^\intercal (X^\intercal X)^{-1} XY}{x_1^\intercal (X^\intercal X)^{-1} x_1} x_1 - \frac{x_1^\intercal (X^\intercal X)^{-1} XY}{(x_1^\intercal (X^\intercal X)^{-1} x_1)^2} \lambda^{-2} x_1 + O(\lambda^{-4}) \\
(ii) &= \lambda^2 \left[1 - \frac{\lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1}{1 + \lambda^2 x_1^\intercal (X^\intercal X)^{-1} x_1}\right] x_1 \\
&= \frac{x_1}{x_1^\intercal (X^\intercal X)^{-1} x_1} - \frac{\lambda^{-2} x_1}{(x_1^\intercal (X^\intercal X)^{-1} x_1)^2} + O(\lambda^{-4})
\end{aligned}$$

Substituting these equations into equation (21) yields

$$\begin{aligned}
x_1^\intercal \theta^{\text{full}} - y_1 = x_1^\intercal (X^\intercal X)^{-1} \Bigg[& XY - \frac{x_1^\intercal (X^\intercal X)^{-1} XY}{x_1^\intercal (X^\intercal X)^{-1} x_1} x_1 + \frac{y_1 x_1}{x_1^\intercal (X^\intercal X)^{-1} x_1} \\
& - \frac{x_1^\intercal (X^\intercal X)^{-1} XY}{(x_1^\intercal (X^\intercal X)^{-1} x_1)^2} \lambda^{-2} x_1 - \frac{\lambda^{-2} y_1 x_1}{(x_1^\intercal (X^\intercal X)^{-1} x_1)^2} \Bigg] - y_1 + O(\lambda^{-4}) \\
= & \left(x_1^\intercal (X^\intercal X)^{-1} XY - \frac{x_1^\intercal (X^\intercal X)^{-1} XY}{x_1^\intercal (X^\intercal X)^{-1} x_1} x_1^\intercal (X^\intercal X)^{-1} x_1 + \frac{y_1 x_1^\intercal (X^\intercal X)^{-1} x_1}{x_1^\intercal (X^\intercal X)^{-1} x_1} - y_1 \right) \\
& + \left(\frac{x_1^\intercal (X^\intercal X)^{-1} XY - y_1}{x_1^\intercal (X^\intercal X)^{-1} x_1} \right) \lambda^{-2} + O(\lambda^{-4}) \\
= & \frac{x_1^\intercal (X^\intercal X)^{-1} XY - y_1}{x_1^\intercal (X^\intercal X)^{-1} x_1} \lambda^{-2} + O(\lambda^{-4}).
\end{aligned} \tag{22}$$

Finally, we substitute the expressions from equations (20) and (22) into (18) to obtain

$$\begin{aligned}
\theta^{\text{inf}} - \theta^{\setminus 1} &= \left(-\frac{(X^\intercal X)^{-1} x_1 x_1^\intercal (X^\intercal X)^{-1}}{x_1^\intercal (X^\intercal X)^{-1} x_1} + O(\lambda^{-2}) \right) \lambda^2 \left(\frac{x_1^\intercal (X^\intercal X)^{-1} XY - y_1}{x_1^\intercal (X^\intercal X)^{-1} x_1} \lambda^{-2} + O(\lambda^{-4}) \right) x_1 \\
&= \frac{y_1 - x_1^\intercal (X^\intercal X)^{-1} XY}{x_1^\intercal (X^\intercal X)^{-1} x_1} (X^\intercal X)^{-1} x_1 + O(\lambda^{-2}).
\end{aligned}$$

Note that this has the same limiting value as $\theta^{\text{full}} - \theta^{\setminus 1}$ as $\lambda \to \infty$ (see equation (17)) and we are done. $\square$

C. Generalization of Theorem 2 to weighted, ridge regularized least squares

We can generalize our method for computing the predictions of the LKO model to weighted least squares with ridge regularization. Let $w \succeq 0 \in \mathbb{R}^n$ denote a (fixed) weight vector and $\lambda \geq 0$ be the regularization strength, which we require to be fixed independent of the number of samples. The weighted, regularized loss is given by

$$\begin{aligned} L^{\text{full}}(\theta) &= \frac{1}{2}\left(\sum_{i=1}^{n} w_i(\theta^\intercal x_i - y_i)^2 + \lambda\|\theta\|^2\right) \\ &= \frac{1}{2}[(X\theta - Y)^\intercal W(X\theta - Y) + \lambda\|\theta\|^2]. \end{aligned}$$

The gradient is therefore

$$\nabla L^{\text{full}}(\theta) = X^\intercal WX\theta - X^\intercal WY + \lambda\theta \tag{23}$$

Using equation (23), we see that $\nabla L^{\text{full}} = 0$ when

$$\theta^{\text{full}} = (X^\intercal WX + \lambda I)^{-1}X^\intercal WY.$$

Predictions are therefore given by

$$X\theta^{\text{full}} = \underbrace{X(X^\intercal WX + \lambda I)^{-1}X^\intercal W}_{H_{\lambda,w}}Y.$$

If we replace H in equation (5) with $H_{\lambda,w}$, the same logic carries through. Note that the regularization strength needs to be fixed for us to use the same trick, i.e. to write

$$\theta^{\setminus k} = \operatorname*{argmin}_{\theta} \sum_{i=1}^{k} w_i(\theta^\intercal x_i - \hat{y}_i^{\setminus k})^2 + \sum_{i=k+1}^{n} w_i(\theta^\intercal x_i - y_i)^2 + \lambda\|\theta\|^2$$

with $\hat{y}_i^{\setminus k} = \theta^{\setminus k\intercal}x_i$ the predicted y-value for the LKO model. In this case, we can compute the LKO prediction values efficiently ($O(k^3)$ time when we precompute $H_{\lambda,w}$). Theorem 2 therefore holds in this more general setting as well.

D. Generalization of the PRU 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 2 from the previous section to compute a fast approximation to a Newton step.

Algorithm 5 The PRU for logistic regression

```
1: procedure LogisticPRU($X, Y, \theta^{\text{full}}, k$)
2:     for $i = 1, \ldots, n$ do
3:         $w_i \leftarrow h_{\theta^{\text{full}}}(x_i)(1 - h_{\theta^{\text{full}}}(x_i))$
4:     end for
5:     $S_{\theta^{\text{full}}} \leftarrow \text{diag}(w)$
6:     $Z \leftarrow X\theta^{\text{full}} + S_{\theta^{\text{full}}}^{-1}(Y - h_{\theta^{\text{full}}})$        ▷ $w, S_{\theta^{\text{full}}}$, and $Z$ are all computed offline
7:     return ResidualUpdate($X, Z, H_{\lambda,w}, \theta^{\text{full}}, k$)
8: end procedure
```

Theorem 5. *Algorithm 5 computes the update $\theta^{res} = \theta^{full} + proj_{span(x_1,\ldots,x_k)}(\Delta_{Newton})$ in $O(k^2 d)$ time.*

Proof. For logistic regression, we use the loss function

$$L(\theta) = \sum_{i=1}^{n}[y_i \log h_\theta(x_i) + (1 - y_i)\log(1 - h_\theta(x_i))] + \frac{1}{2}\lambda\|\theta\|^2,$$

where $(x_i, y_i) \in \mathbb{R}^d \times \{0,1\}$ are the data, and the classifier $h_\theta(x)$ is given by

$$h_\theta(x) = \frac{1}{1+\exp\{-\theta^\intercal x\}}.$$

We compute the gradient and Hessian of the loss:

$$\nabla L(\theta) = \sum_{i=1}^{n} (h_\theta(x_i) - y_i)x_i + \lambda\theta = \bar{X}^\intercal(\bar{h}_\theta - \bar{Y}) + \lambda\theta \tag{24}$$

$$\nabla^2 L(\theta) = \sum_{i=1}^{n} h_\theta(x_i)(1-h_\theta(x_i))x_i x_i^\intercal + \lambda I = \bar{X}^\intercal \bar{S}_\theta \bar{X} + \lambda I, \tag{25}$$

where $\bar{X} \in \mathbb{R}^{n\times d}$ is the data matrix whose rows are $x_i^\intercal$, $\bar{h}_\theta \in \mathbb{R}^n$ is the vector of model predictions, $\bar{Y} \in \mathbb{R}^n$ is the vector of labels, and $\bar{S}_\theta = \text{diag}(\{h_\theta(x_i)(1-h_\theta(x_i)\}_{i=1}^n) \in \mathbb{R}^{n\times n}$. Using these formulas, we can compute a Newton step for the LKO loss when we start at the minimizer θ^{full} for the full loss.

Now let $X = (x_{k+1} \ \cdots x_n)^\intercal \in \mathbb{R}^{(n-k)\times d}$, $Y = (y_{k+1}, \ldots, y_n)^\intercal \in \mathbb{R}^{n-k}$, $h_{\theta^{\text{full}}} = (h_{\theta^{\text{full}}}(x_{k+1}), \ldots, h_{\theta^{\text{full}}}(x_n))^\intercal \in \mathbb{R}^{n-k}$, $S_{\theta^{\text{full}}} = \text{diag}(\{h_{\theta^{\text{full}}}(x_i)(1-h_{\theta^{\text{full}}}(x_i)\}_{i=k+1}^n)$ be the LKO quantities corresponding to the terms defined above. By definition, we have

$$\begin{aligned}
\theta^{\text{Newton}} &= \theta^{\text{full}} - [\nabla^2 L^{\setminus k}(\theta^{\text{full}})]^{-1}\nabla L_{LKO}(\theta^{\text{full}}) \\
&= \theta^{\text{full}} + (X^\intercal S_{\theta^{\text{full}}} X + \lambda I)^{-1}(X^\intercal(Y - h_{\theta^{\text{full}}}) - \lambda\theta^{\text{full}}) \\
&= (X^\intercal S_{\theta^{\text{full}}} X + \lambda I)^{-1} X^\intercal S_{\theta^{\text{full}}}(X\theta^{\text{full}} + S_{\theta^{\text{full}}}^{-1}(Y - h_{\theta^{\text{full}}})) \\
&= (X^\intercal S_{\theta^{\text{full}}} X + \lambda I)^{-1} X^\intercal S_{\theta^{\text{full}}} Z,
\end{aligned} \tag{26}$$

where $Z \equiv X\theta^{\text{full}} + S_{\theta^{\text{full}}}^{-1}(Y - h_{\theta^{\text{full}}})$. Observe that equation (26) is the solution to the LKO weighted least squares problem

$$\min_\theta \sum_{i=k+1}^{n} h_{\theta^{\text{full}}}(x_i)(1-h_{\theta^{\text{full}}}(x_i))(\theta^\intercal x_i - z_i)^2 + \lambda\|\theta\|^2, \tag{27}$$

where z_i is the i-th component of $\bar{Z} \equiv \bar{X}\theta^{\text{full}} + \bar{S}_{\theta^{\text{full}}}^{-1}(\bar{Y} - \bar{h}_{\theta^{\text{full}}})$. By adapting the PRU to this situation, we can compute a fast approximation to the Newton step.

We can compute the vector $\bar{Z} \equiv \bar{X}\theta^{\text{full}} + \bar{S}_{\theta^{\text{full}}}^{-1}(\bar{Y} - \bar{h}_{\theta^{\text{full}}})$, as well as the matrix $H_{\lambda,\bar{h}_{\theta^{\text{full}}}} \equiv \bar{X}(\bar{X}^\intercal \bar{S}_{\theta^{\text{full}}}\bar{X} + \lambda I)^{-1}\bar{X}^\intercal \bar{S}_{\theta^{\text{full}}}$, offline. Observe that $H_{\lambda,\bar{h}_{\theta^{\text{full}}}}$ is the hat matrix for the "full" least squares problem

$$\min_\theta \sum_{i=1}^{n} h_{\theta^{\text{full}}}(x_i)(1-h_{\theta^{\text{full}}}(x_i))(\theta^\intercal x_i - z_i)^2 + \lambda\|\theta\|^2. \tag{28}$$

For consistency with the rest of the paper, let $\theta^{\setminus k}$ be the exact solution to (27) (so $\theta^{\setminus k} = \theta^{\text{Newton}}$). By the result of Theorem 2, we can compute the LKO model predictions $\hat{z}_i^{\setminus k} \equiv \theta^{\setminus k\intercal} x_i$, $i = 1, \ldots, k$ in $O(k^3)$ time. Observe that the gradient of the (unregularized, unweighted, quadratic) loss on the synthetic points $(x_i, \hat{z}_i^{\setminus k})$ is

$$\sum_{i=1}^{k} (\theta^{\text{full}}{}^{x}_{i} - \hat{z}^{\setminus k})x_i = \left(\sum_{i=1}^{k} x_i x_i^\intercal\right)(\theta^{\text{full}} - \theta^{\setminus k}). \tag{29}$$

We are now in a setting exactly analogous to equation (9), even though θ^{full} was the minimizer for the original *cross-entropy* objective rather than (28). By mimicking the proof of Theorem 1 from this point, we can derive the exact same results. Namely, the step taken by the projective residual update is equal to $\text{proj}_{\text{span}(x_1,\ldots,x_k)}(\theta^{\setminus k} - \theta^{\text{full}})$. By definition of $\theta^{\setminus k}$ and of the Newton step, it follows that $\theta^{\setminus k} - \theta^{\text{full}} = \Delta_{\text{Newton}}$. Combining these two facts yields the statement of Theorem 5. The runtime calculation is identical to the calculation in Theorem 1. □

E. Synthetic data construction

We first generate a matrix of n d-dimensional covariates $X \in \mathbb{R}^{n \times d}$; we do this by drawing the rows $x_i^\intercal$ of X according to $x_i \overset{\text{i.i.d.}}{\sim} N(0, \Sigma)$, where Σ is randomly selected via `sklearn.datasets.make_spd_matrix` (Pedregosa et al., 2011). Once X is generated, the response vector $Y \in \mathbb{R}^n$ is generated by randomly selecting a (fixed) "true" underlying parameter $\theta^* \in \mathbb{R}^d$, and setting $Y = X\theta^* + \varepsilon$, where $\varepsilon \sim N(0, \sigma^2 I_n)$ is the error vector. For our experiments, we set the noise level $\sigma^2 = 1$; for reasonable values of σ^2 this parameter does not play a large role in the outcome of the experiments. For all of the synthetic experiments, when deleting a group of size k, we always assume that it is the first k datapoints which are being deleted. (That is, we delete the datapoints specified by the first k rows of X and the first k entries of Y.) For all of the synthetic datasets, we take $n = 10d$.

For the runtime experiment, no modifications are made to the general setup. We vary the dimension d between $d = 1000$ and $d = 3000$ and the group size k between $k = 1$ and $k = 100$.

For the L^2 experiment, we first construct $\tilde{X}$ and $\tilde{Y}$ according to the general procedure above. We then obtain the data X, Y by multiplying the first k rows of $\tilde{X}$ and the first k entries of $\tilde{Y}$ (that is, the points which will eventually be deleted) by a factor λ to demonstrate the effectiveness of each method at removing outlier datapoints. This is the setting described in section 4.3. We fix the dimension $d = 1500$ and vary the multiplier λ between $\lambda = 1$ (no outlier) and $\lambda = 100$ (large outlier), and the group size k between $k = 1$ and $k = 150$.

The modifications for the special feature experiment are slightly more involved. We construct sparse data with three key properties: (1) only the deleted feature vectors x_i, $i = 1, \dots, k$ have nonzero d-th entry (this is the "special feature"); (2) the deleted feature vectors all lie on the same low-dimensional subspace (this exaggerates the characteristics we observe on the Yelp dataset); (3) the response for the deleted points is perfectly correlated with the special feature. The exact steps for this procedure are as follows: (Here $x_i[d]$ denotes the d-th entry of x_i.)

1. Construct $\tilde{X}$ according to the general procedure. (Pick a random covariance Σ and draw the rows $x_i^\intercal$ of $\tilde{X}$ according to $x_i \overset{\text{i.i.d.}}{\sim} N(0, \Sigma)$.)
2. The "special feature" will be the last (d-th) entry of each vector x_i. Since only the group being removed has the special feature, we set the last entry in rows $k+1$ to n of $\tilde{X}$ equal to 0; the first k rows keep their original final entry.
3. Sparsify $\tilde{X}$ so that it has a fraction of approximately p nonzero entries. Let $\tilde{X}[i, j]$ denote the (i, j)-th entry of $\tilde{X}$.
   (a) Sparsify the first k rows of $\tilde{X}$ simultaneously: for each $j = 1, \dots, d-1$, set $\tilde{X}[i, j] = 0$ for all $i = 1, \dots, k$ with probability $1 - p$.
   (b) Sparsify the remaining entries of $\tilde{X}$: for each $i = k+1, \dots, n$ and $j = 1, \dots, d-1$, set $X[i, j] = 0$ with probability $1 - p$.
4. Let X be the matrix resulting from performing operations 1-3 on $\tilde{X}$. Set $\tilde{Y}$ according to the general procedure: $\tilde{Y} = X\theta^* + E$.
5. Let $\tilde{Y}[i]$ denote the i-th entry of $\tilde{Y}$. For $i = 1, \dots, k$, set $\tilde{Y}[i] = w_* \cdot X[i, d]$, where w_* is the pre-specified "true" special weight of the special feature.

We again fix the dimension $d = 1500$ and vary the sparsity p between $p = 0.5$ and $p = 0.05$, and the group deletion size k between $k = 1$ and $k = 150$.

F. Baseline values for synthetic experiments

All of the experimental results in the main body of the paper are given relative to an absolute baseline value. In Figures 5, 6, and 7, we report the medians of the absolute baseline values to which we are comparing for each of the three synthetic experiments (runtime, L^2, and feature injection, respectively). The baseline values follow the trends we would expect. In particular, the runtimes increase sharply as the dimension increases and slowly as the group size increases (Table 5); the unique feature weight originally learned by the model is close to its true value, 10 (Table 6); and the distance between θ^{full} and $\theta^{\setminus k}$ increases with the number of points removed, as well as the dissimilarity of these points to the rest of the dataset as measured by the multiplier λ (Table 7).

Table 5. Median exact retraining runtimes in seconds. Method used was Newton with Sherman-Morrison formula.

	$d = 1000$	$d = 1500$	$d = 2000$	$d = 2500$	$d = 3000$
$k = 1$	0.08	0.27	0.67	1.19	2.25
$k = 2$	0.08	0.27	0.67	1.19	1.54
$k = 3$	0.08	0.30	0.67	1.33	2.20
$k = 4$	0.08	0.30	0.67	1.33	2.27
$k = 5$	0.08	0.31	0.67	1.33	2.22
$k = 10$	0.08	0.31	0.63	1.33	2.04
$k = 25$	0.08	0.32	0.62	1.36	2.06
$k = 50$	0.09	0.33	0.64	1.40	2.11
$k = 75$	0.10	0.35	0.66	1.44	2.16
$k = 100$	0.10	0.36	0.69	1.48	2.47
$k = 125$	0.11	0.38	0.71	1.52	2.49
$k = 150$	0.12	0.39	0.73	1.56	1.76

Table 6. Median baseline special weights.

	$p = 0.5$	$p = 0.25$	$p = 0.1$	$p = 0.05$
$k = 1$	11.01	9.03	10.33	9.53
$k = 5$	9.57	8.97	10.61	11.03
$k = 10$	11.25	9.90	10.90	10.43
$k = 50$	10.36	10.23	9.73	10.10
$k = 100$	9.73	9.51	9.99	10.01
$k = 150$	9.73	10.09	9.88	10.04

Table 7. Median baseline L^2 parameter distance.

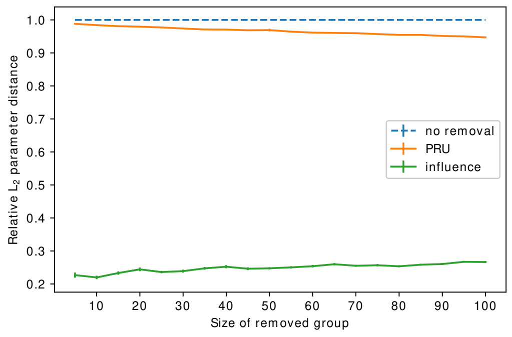
	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$
$k = 1$	0.009	0.081	0.089
$k = 5$	0.018	0.175	0.192
$k = 10$	0.025	0.234	0.254
$k = 50$	0.057	0.523	0.572
$k = 100$	0.082	0.761	0.842
$k = 150$	0.097	0.921	1.015

G. CIFAR-100 Experiment

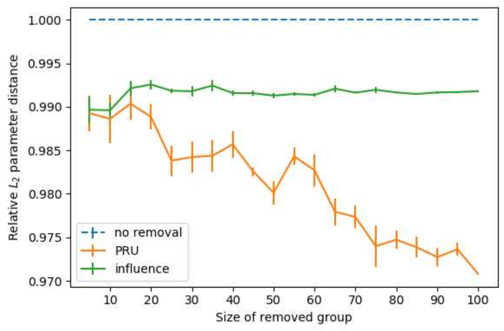
Dataset We focus on two classes in the CIFAR-100 image dataset: fish and aquatic mammals (Krizhevsky, 2009). We then use a pre-trained version of AlexNet to extract 1000 features from the 32x32 pixel images (Krizhevsky, 2014). The resulting dataset contains 5000 examples, 2500 of each class. We want to consider the problem of deletion of outliers for this dataset, so we select 100 points from the positive class and multiply each feature by 25 to artificially create outliers. For the experiments on this dataset, we select random points from the set of 100 outliers. Neither method was very successful at removing the injected feature for this dataset, though we did see the predictable behavior that the influence method removed the special weight slightly better than the PRU in the non-outlier setting, and the PRU removed the special weight slightly more effectively with the outliers added. The L^2 results show a clearer picture, so we focus on the L^2 test for this experiment.

Discussion of results When the dataset contains no outlier points, the full Hessian and LKO Hessian are still relatively similar, and the influence method performs well as it gives an accurate approximation of the Newton step. However, when we again consider removal of outlier points, the performance of the influence method drops below that of the PRU. Note that while the PRU outperforms the influence method for outlier removal, both methods are still closer to the full parameters than the exact LKO parameters.

Figure 2. Figures for the CIFAR-100 experiment. When removing "normal" points from the dataset, the influence method gives a close approximation to Newton's method and therefore performs well. When we remove outliers, however, the approximation is no longer valid and the performance of the influence method drops.



(a) CIFAR L^2 experiment, no outlier



(b) CIFAR L^2 experiment, $25\times$ outlier