
Efficient Optimization Algorithms for Linear Adversarial Training

Antônio H. Ribeiro
Uppsala University

Thomas B. Schön
Uppsala University

Dave Zachariah
Uppsala University

Francis Bach
INRIA, PSL Research Univ.

Abstract

Adversarial training can be used to learn models that are robust against perturbations. For linear models, it can be formulated as a convex optimization problem. Compared to methods proposed in the context of deep learning, leveraging the optimization structure allows significantly faster convergence rates. Still, the use of generic convex solvers can be inefficient for large-scale problems. Here, we propose tailored optimization algorithms for the adversarial training of linear models, which render large-scale regression and classification problems more tractable. For regression problems, we propose a family of solvers based on iterative ridge regression and, for classification, a family of solvers based on projected gradient descent. The methods are based on extended variable reformulations of the original problem. We illustrate their efficiency in numerical examples.

1 INTRODUCTION

Adversarial training can be used to estimate models that are robust against perturbations (Madry et al., 2018). It considers training samples that have been modified by an adversary, with the goal of obtaining a model that will be more robust when faced with newly perturbed samples. The training procedure is formulated as a min-max optimization problem, searching for the best model given the worst-case perturbations.

In linear models, adversarial training can be seen both as a type of regularization and as a type of robust regression. This method is promising for estimating linear models: For the ℓ_2 -norm, adversarial training has properties similar to ridge regression, and for the ℓ_∞ -norm, it is similar to Lasso and produces sparse

solutions. Unlike the hyperparameters set in these conventional methods, however, the adversarial radius δ has a clear interpretation and—as proved by Ribeiro et al. (2023b) and Xie et al. (2024)—this parameter can be set independently of the noise levels in the problem. Indeed, for adversarial training, there exists a default choice of δ values that yields zero coefficients for random outputs and near-oracle prediction performance. Hence, it could be a powerful method in applications where cross-validation cannot be effectively used to choose the regularization parameter. We illustrate this in Fig. 1 (*left panel*), where *adversarial training with the default adversarial radius δ* achieves a performance that is close to that of *cross-validated Lasso* in diverse benchmarks.

Consider a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ consisting of n training datapoints of dimension $\mathbb{R}^p \times \mathbb{R}$. Adversarial training in linear models finds the parameter vector $\boldsymbol{\beta} \in \mathbb{R}^p$ by solving

$$\min_{\boldsymbol{\beta}} \left(\frac{1}{n} \sum_{i=1}^n \max_{\|\mathbf{d}_i\| \leq \delta} \ell(y_i, (\mathbf{x}_i + \mathbf{d}_i)^\top \boldsymbol{\beta}) \right), \quad (1)$$

where $\ell(\cdot, \cdot)$ is the loss and the adversarial radius δ gives the maximum magnitude of the adversarial perturbation. In this paper, we consider both classification problems and regression problems—with the logistic and squared error loss. In both cases, the function being minimized in (1) is convex.

The two most direct options for solving (1) have drawbacks. The *first option* is to use *solvers proposed in the context of deep learning*, e.g., Madry et al. (2018). These approaches are scalable but suffer from *slow convergence*, and therefore inaccurate solutions, since they do not exploit the structure of the optimization problem. The *second option* is to use off-the-shelf convex solvers (Diamond et al., 2016), which in contrast, usually produce accurate solutions due to their fast convergence. On the other hand, they offer limited scalability and are prohibitively slow for problems with many variables, as in the motivating example below. In this paper *we propose tailored solvers that are accurate and efficient for high-dimensional problems, enabling new applications*.

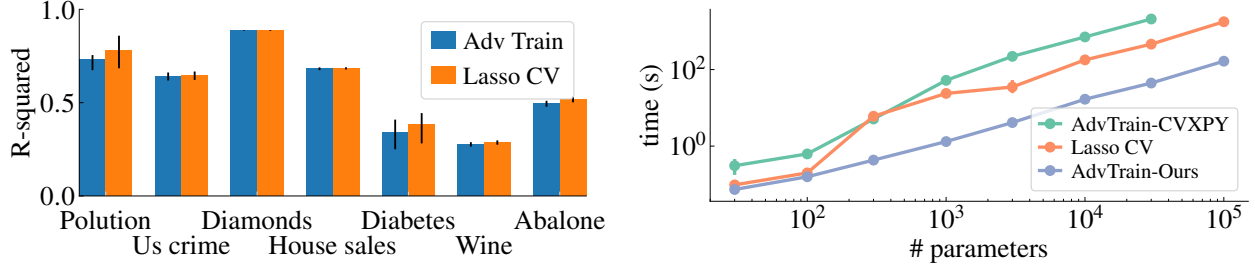


Figure 1: *Adversarial training in linear regression.* Left: we compare linear adversarial training (ℓ_∞ -norm bounded attacks) using the *default adversarial radius*—see Ribeiro et al. (2023b) and the description in Appendix A.2—and Lasso with parameters selected through *cross-validation*. Right: the running times for pruned MAGIC dataset of *adversarial training with our tailored solver*, with CVXPY, and of *cross-validation Lasso*, optimized as in Friedman et al. (2007). The error bars in (a) show the interquartile range obtained using bootstrap. In (b), we plot the median of 5 repetitions.

Motivating example. Consider phenotype prediction from genotype in the Diverse MAGIC wheat dataset (Scott et al., 2021) from the National Institute for Applied Botany. The dataset contains the whole genome sequence data and multiple phenotypes for a population of 504 wheat lines. In this problem, our tailored solver converges in less than 5 minutes while the CVXPY solver (Diamond et al., 2016) did not manage to converge in one day. Fig. 1(right panel) shows running times for subsampled inputs with increasing sizes. Here the coefficient of determination R^2 is used to evaluate goodness-of-fit (higher is better): ℓ_∞ -adversarial training with default parameter has a performance $R^2 = 0.35$ (IQR 0.27-0.46) while cross-validated lasso leads to $R^2 = 0.38$ (IQR 0.27-0.41).¹

There are important reasons to be *interested directly in linear models*. High-dimensional sparse problems, where the number of features far exceeds the number of observations, are common in fields like computational biology and genetics. High variance and overfitting are major concerns, and *simple but highly regularized approaches are often preferred*. For instance, Lasso is very frequently used for polygenic risk score (Torkamani et al., 2018), proteomics analysis (Safo et al., 2023), and multivariate mendelian randomization (Burgess et al., 2020). However, Lasso has limitations: while cross-validation can be effective for prediction, it often selects too many variables (Bühlmann et al., 2011), especially in small datasets where cross-validation itself may be unreliable. Consequently, methods that bypass cross-validation, such as those implemented in polygenic risk score analysis packages like `lassosum` and `LDpred` (Mak et al., 2017; Vilhjálmsdóttir et al., 2015), are often pursued. In light of these challenges, we propose that linear adversarial training could be a promising alternative, with the development of effective solution methods being critical to its success.

¹We use IQR for bootstrapped interquartile range.

Contributions. The contribution of this paper is to *reformulate the problem and propose tailored efficient solvers for the minimization of Eq. (1)*. We propose: (A) An *augmented* variable formulation that renders a smooth optimization problem in adversarially trained *logistic regression*. We propose an efficient family of first-order algorithms based on this reformulation, allowing us to efficiently solve (1) when $\ell(y, \hat{y}) = \log(1 + e^{-y\hat{y}})$. (B) A *reformulation* of adversarially trained *linear regression* that can be solved using iteratively reweighted least-squares problems. This reformulation allow us to efficiently solve (1) when $\ell(y, \hat{y}) = (y - \hat{y})^2$. We validate these algorithms with real and simulated data. The implementation is available in github.com/antonior92/advtrain-linear.

2 RELATED WORK

Adversarial attacks can significantly degrade the performance of state-of-the-art models and adversarial training has emerged as one of the most effective strategies. Relevant related work is discussed next.

Adversarial training for deep learning models. Traditional methods for generating *adversarial attacks* involve solving an optimization problem. One approach is to maintain the model’s prediction while minimizing the disturbance to the input. Examples include L-BFGS (Bruna et al., 2014), DeepFool (Moosavi-Dezfooli et al., 2016) and the C&W method (Carlini et al., 2017). Another approach is to maximize the error with limited attack size, such as in the Fast Gradient Size Method (Goodfellow et al., 2015), and PGD type of attacks (Madry et al., 2018). *Adversarial training* builds on the latter approach, resulting in a min-max optimization problem, as in (1), that can be solved backpropagating through the inner loop (Madry et al., 2018). Most improvements proposed in the context of deep learning try to better deal with the inner maximization problem. And, for the linear case, they do

not provide improvements over the Fast Gradient Size Method (FGSM), since the inner problem can already be solved exactly. As we discuss in Section 3.4, *just adapting FGSM to the linear case has limitations in that it is hard to get accurate solutions due to its slow convergence rate.*

Adversarial training in linear models. Adversarial attacks in linear models have been intensely studied over the past 5 years. Linear adversarial training provides an alternative to Lasso and other sparsity inducing methods (Ribeiro et al., 2023b; Xie et al., 2024). It also provides a simplified analysis for understanding the properties of deep neural networks: indeed, linear models have been used to explore the trade-off between robustness and performance in neural networks (Ilyas et al., 2019; Tsipras et al., 2019) and to examine how overparameterization impacts robustness (Ribeiro et al., 2023a). Adversarial training has been studied through asymptotics in both binary classification (Taheri et al., 2022) and linear regression (Javanmard et al., 2020), as well as in classification (Javanmard et al., 2022) and random feature regressions (Hassani et al., 2022). Gaussian classification problems have also been analyzed (Dan et al., 2020; Dobriban et al., 2022), alongside the effects of dataset size on adversarial performance (Min et al., 2021) and ℓ_∞ -attacks on linear classifiers (Yin et al., 2019). *While the literature above focuses on its generalization properties of adversarial training in linear models, we propose new optimization algorithms to solve the problem.*

Robust regression and square-root Lasso. Lasso (Tibshirani, 1996) usage for sparse recovery depends on the knowledge of the noise variance. Square-root Lasso avoids this dependence (Belloni et al., 2011). Ribeiro et al. (2023b) and Xie et al., 2024 show that ℓ_∞ -adversarial settings also have the same desirable properties. Both methods can be framed within the robust regression framework (Ribeiro et al., 2023b). Still, *having tailored solvers that work in high-dimensional settings yields a practical advantage to the adversarial training framework over square-root Lasso.*

Sparsity-inducing penalties and convex optimization. Our solutions are inspired by methods used in the optimization of other sparsity-inducing penalties. For instance, we use the “ η -trick” (a.k.a. iteratively reweighted least-squares) to solve adversarial-trained linear regression (Bach, 2011, 2019; Daubechies et al., 2010). Our convergence rates analysis uses tools from convex optimization Bubeck (2015). We also use conjugate gradient (Golub et al., 2012; Nocedal et al., 2006, Chap. 5) as a solver to linear systems that arise as subproblems. Finally, we implement variance reduction methods to improve our solver (Gower et al., 2020). *We reformulate adversarial training in order to leverage*

these tools.

3 CLASSIFICATION

In this section, we focus on adversarial training for classification. More precisely, in solving (1) when $y \in \{-1, +1\}$ and $\ell(y, \hat{y}) = h(y\hat{y})$ for h non-increasing, smooth and convex:

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^n \max_{\|\mathbf{d}_i\| \leq \delta} h(y_i(\mathbf{x}_i + \mathbf{d}_i)^\top \beta)$$

We focus on $h(z) = \log(1 + e^{-z})$, but other alternatives are possible, i.e., $h(z) = (1 - z)_+^2$. We first introduce a reformulation of this problem that turns it into a constrained smooth optimization problem (Section 3.1) and propose the use of projected gradient descent to solve it. In Section 3.2 we give more details on how to solve the projection step efficiently. In Section 3.4 we give a comparison with methods used in the context of deep learning. Finally, in Section 3.3 we propose improvements of this base algorithm.

Notation. We use $\|\cdot\|$ to denote an arbitrary norm and $\|\beta\|_* = \sup\{\beta^\top \mathbf{x} : \|\mathbf{x}\| \leq 1\}$ its dual norm. The relevant pairs of dual norms for this papers are $(\|\cdot\|_2, \|\cdot\|_2)$ and $(\|\cdot\|_1, \|\cdot\|_\infty)$.

3.1 Smooth minimization formulation

The following result allows us to solve a smooth constrained optimization problem rather than the original min-max optimization.

Proposition 1. Let $\rho > 0$ and $\ell(y, \hat{y}) = h(y\hat{y})$ for h non-increasing, 1-smooth and convex. The optimization of (1) is equivalent to

$$\min_{(\beta, t)} \mathcal{R}(\beta, t) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n h(y_i \mathbf{x}_i^\top \beta - \rho t), \quad (2)$$

subject to: $\rho t \geq \delta \|\beta\|_*$.

Moreover, the function \mathcal{R} is L -smooth and jointly convex. And, $L \leq \frac{1}{2} \lambda_{\max}(\frac{1}{n} \sum_i \mathbf{x}_i \mathbf{x}_i^\top)$ if $\rho^2 \leq \lambda_{\max}(\frac{1}{n} \sum_i \mathbf{x}_i \mathbf{x}_i^\top)$.

This reformulation allows us to solve the problem efficiently using projected gradient descent (GD).

Algorithm 1 Smooth min. with projected GD

Choose step size γ

$\mathbf{w}^{(0)} \leftarrow (\beta^{(0)}, t^{(0)})$

for $k = 1, 2, \dots$:

$\mathbf{w}^{(k)} \leftarrow \text{Proj}_C(\mathbf{w}^{(k-1)} - \gamma \nabla \mathcal{R}(\mathbf{w}^{(k-1)}))$

The cost function \mathcal{R} optimized in Proposition 1 is L -smooth (i.e., its gradient is L -Lipshitz continuous, that is, $\|\nabla \mathcal{R}(\mathbf{w}) - \nabla \mathcal{R}(\mathbf{z})\|_2 \leq L \|\mathbf{w} - \mathbf{z}\|_2$). Let k denote the

number of iterations, projected GD (Algorithm 1) has a convergence rate of $O(L/k)$ for a step size $\gamma = 1/L$, see Bubeck (2015, Theorem 3.7). Let \mathbf{w} denote the extended variable in \mathbb{R}^{p+1} , C the set of feasible points, i.e.,

$$C = \{\mathbf{w} = (\boldsymbol{\beta}, t) : \rho t \geq \delta \|\boldsymbol{\beta}\|_*\} \subset \mathbb{R}^{p+1},$$

and Proj_C , the projection of a point into this set.

Proof of Proposition 1. Eq. (1) is equivalent to

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^n h(y_i \mathbf{x}_i^\top \boldsymbol{\beta} - \delta \|\boldsymbol{\beta}\|_*), \quad (3)$$

for $\ell(y, \hat{y}) = h(y\hat{y})$ and h non-increasing, smooth and convex. See Appendix A.1 for a proof. In turn, the minimization of (3) is equivalent to that of $\mathcal{R}(\boldsymbol{\beta}, t)$ subject to the constraint $\delta t \geq \rho \|\boldsymbol{\beta}\|$. And since $\ell(\cdot)$ is convex and smooth, so is $\mathcal{R}(\cdot, \cdot)$. Finally, the Hessian of \mathcal{R} is:

$$\nabla^2 \mathcal{R}(\mathbf{w}) \preceq \frac{1}{2} \begin{bmatrix} \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top & 0 \\ 0 & \rho^2 \end{bmatrix}. \quad (4)$$

And, since $L \leq \sup_{\mathbf{w}} \|\nabla^2 \mathcal{R}(\mathbf{w})\|_2$, we have

$$L \leq \frac{1}{2} \max \left(\left\| \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top \right\|_2, \rho^2 \right) \leq \lambda_{\max} \left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top \right),$$

where the last inequality follows from the condition on ρ and that $\|\cdot\|_2 = \lambda_{\max}(\cdot)$ for matrices. \square

3.2 Projection step

The set C is convex and we can define the following projection operator:

$$\text{Proj}_C(\tilde{\boldsymbol{\beta}}, \tilde{t}) = \arg \min_{\rho t \geq \delta \|\boldsymbol{\beta}\|_*} \frac{1}{2} \|\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}\|_2^2 + \frac{1}{2} (t - \tilde{t})^2.$$

For ℓ_2 -adversarial and ℓ_∞ -adversarial attacks the following propositions can be used for computing the projection. The proofs are given in Appendix B.1.

Proposition 2 (Projection step). If $(\tilde{\boldsymbol{\beta}}, \tilde{t}) \in C$, the projection is the identity. On the other hand, if $(\tilde{\boldsymbol{\beta}}, \tilde{t}) \notin C$, let us denote $\text{Proj}_C(\tilde{\boldsymbol{\beta}}, \tilde{t}) = (\boldsymbol{\beta}^{\text{Proj}}, t^{\text{Proj}})$. We have that:

1. For $C = \{(\boldsymbol{\beta}, t) : \rho t \geq \delta \|\boldsymbol{\beta}\|_2\}$, then $t^{\text{Proj}} = \frac{\delta \rho \|\tilde{\boldsymbol{\beta}}\|_2 + \delta \tilde{t}}{\delta^2 + \rho^2}$ and $\boldsymbol{\beta}^{\text{Proj}} = \frac{\rho t^{\text{Proj}}}{\delta \|\tilde{\boldsymbol{\beta}}\|_2} \tilde{\boldsymbol{\beta}}$.

2. Now, for $C = \{(\boldsymbol{\beta}, t) : \rho t \geq \delta \|\boldsymbol{\beta}\|_1\}$. Let $(\cdot)_+$ denote the operator $\max(\cdot, 0)$ and define λ as the solution to

$$\sum_{i=1}^p (|\tilde{\beta}_i| - \delta \lambda)_+ = \tilde{t} + \rho \lambda, \quad (5)$$

which exists and is unique for $\tilde{t} \geq 0$. Then, $t^{\text{Proj}} = \tilde{t} + \rho \lambda$ and $\beta_i^{\text{Proj}} = \text{sign}(\tilde{\beta}_i) (|\tilde{\beta}_i| - \delta \lambda)_+$.

Here the projection step for ℓ_∞ -adversarial attacks needs to solve the piecewise linear equation (5). This

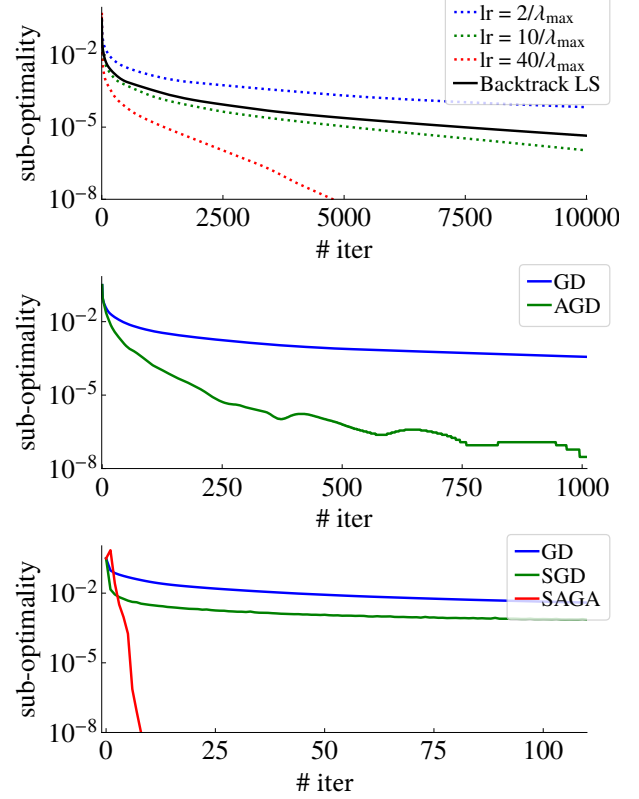


Figure 2: *Convergence of Projected GD.* Sub-optimality *vs* number of iterations in the MNIST dataset. *Top:* we show the results for different step sizes. *Middle:* we show the effect of momentum (with line search implemented for both methods). *Bottom:* we compare GD, SGD and SAGA. Results for ℓ_∞ -adv. training with $\delta = 0.01$. Here one epoch is a full pass throughout the dataset.

equation can be solved by sorting the elements of the sum and has the overall computational cost of $O(p \log p)$.² For ℓ_2 -adversarial attacks the projection has the computational cost of $O(p)$.

3.3 Practical considerations and improvements on the basic algorithm

In what follows, we describe practical details and improvements. These variations that we proposed, implemented and evaluated are described in Table 1. We also report their convergence rates and costs per interaction. Pseudo-algorithms are provided in Appendix B.3.

(a) Choosing step size. Proposition 1 yields practical

²Our problem is similar to projecting into the ℓ_1 ball. See Condat (2016) for a description of different algorithms for it. There are alternatives with an *expected* complexity of $O(p)$ that might be faster in practice but have worst-case complexity $O(p^2)$. Similar algorithms might apply here but they will not be considered in our developments.

Table 1: *Proposed algorithms for classification.* Variance reduction is abbreviated as *VR*, Acceleration as *Acc.* we consider the computational cost of the projection step $O(p \log p)$.

	Algorithm	Line search	Acc.	VR	Cost/iter ($\nabla + \text{Proj}$)	Convergence Rate
Determin.	FGSM with GD	×	×	-	$O(pn)$	$O(R/\sqrt{k})$
	GD (Ours)	✓	×	-	$O(pn) + O(p \log p)$	$O(L/k)$
	AGD (Ours)	✓	✓	-	$O(pn) + O(p \log p)$	$O(L/k^2)$
Stochastic	FGSM with SGD	×	×	-	$O(p)$	$O(R/\sqrt{k})$
	SGD (Ours)	×	×	×	$O(p) + O(p \log p)$	$O((L+n)/\sqrt{k})$
	SAGA (Ours)	×	×	✓	$O(p) + O(p \log p)$	$O((L+n)/k)$

Notes: L and R are the smoothness and Lipschitz constant of \mathcal{R} in (2). For FGSM with GD it is not possible to use efficient line search methods that are based on curvature conditions. The proof of the convergence rate of GD, AGD, SGD is given in (Bubeck, 2015), Chapters 3 and 6. The proof for SAGA is given in (Defazio et al., 2014).

choices of γ . Notice that $2L = \lambda_{\max}(\frac{1}{n} \sum \mathbf{x}_i \mathbf{x}_i^\top)$ yielding $\gamma = 1/L = 2/\lambda_{\max}(\frac{1}{n} \sum \mathbf{x}_i \mathbf{x}_i^\top)$. In our implementation, we approximately compute the maximum eigenvalue using 10 iterations of the power method (Golub et al., 2012, Section 7.3.1). Motivated by the proof of Proposition 1, we also choose $\rho = \frac{1}{\sqrt{n}} \sum_i \|\mathbf{x}_i\|_2$. This guarantees the conditions of the theorem are satisfied. It also guarantees that the trace of the matrix in the left-hand-side of Equation (4) is the same as the trace of the empirical covariance matrix $\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top$. As an alternative for a fixed step size, we propose the use of backtracking line search (LS), which we will describe next. Notice, that LS will not work for stochastic methods in (d) and (e), hence, the considerations here are still useful.

(b) Backtracking line search. This method is used to determine the step size. The method involves starting with a relatively large estimate of the step size for movement along the line search direction, and iteratively shrinking the step size (i.e., “backtracking”) until a decrease of the objective function is observed that adequately corresponds to the amount of decrease that is expected (Nocedal et al., 2006, Chapter 3). More precisely we use the same backtracking line search used by Beck et al. (2009), it considers the following quadratic approximation around a point \mathbf{w} : $\tilde{\mathcal{R}}_{\mathbf{w}}(\mathbf{w}^+; \gamma) = \mathcal{R}(\mathbf{w}) + (\mathbf{w}^+ - \mathbf{w})^\top \nabla \mathcal{R}(\mathbf{w}) + \frac{1}{2\gamma} \|\mathbf{w}^+ - \mathbf{w}\|_2^2$. accepting the next point \mathbf{w}^+ only if $\mathcal{R}(\mathbf{w}^+) \leq \tilde{\mathcal{R}}_{\mathbf{w}}(\mathbf{w}^+; \gamma)$. If a point is rejected, the learning used in the gradient step is divided by 2, consecutively, until the condition is satisfied.

(c) Accelerated projected gradient descent. Accelerated gradient descent is a popular improvement of the standard gradient descent. It uses information from the previous iteration to improve the convergence of gradient descent. The cost function \mathcal{R} optimized in Proposition 1 is L -smooth and step size $\gamma = 1/L$, the projected gradient descent has a worst-case complexity of $O(L/k)$. The projected accelerated gradient descent (AGD) considers the addition of momentum for

$\gamma = 1/L$ and has a worst-case complexity of $O(L/k^2)$. The analysis of the algorithm follows similar steps as that of FISTA (Beck et al., 2009). The effect of using accelerated projected gradient in practice is illustrated in Figure 2 (*middle panel*).

(d) Stochastic gradient descent. The stochastic gradient descent (SGD) method avoids the heavy cost per iteration of GD by using, in each update, one randomly selected gradient instead of the full gradient. That is, let us define $f_i(\mathbf{w}) = \ell(y_i \mathbf{x}_i \beta - \rho t)$, we can write $\mathcal{R}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{w})$ and its gradient $\nabla \mathcal{R}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{w})$. SGD use one randomly selected ∇f_{i_k} at each iteration instead of the summation of all terms. *Batch implementations* were also implemented and tested. See Figure S.3 in the Appendix.

(e) Variance reduction. Variance reduction methods keep an estimate g_k of the gradient, with the estimate having a variance that converges to zero (see, e.g., Gower et al., 2020, and references therein). For gradient descent methods we have $\mathbb{E}[\nabla f_i(\mathbf{w})] = \nabla \mathcal{R}(\mathbf{w})$. Having an unbiased estimate, however, is not sufficient to guarantee the convergence of SGD with fixed step size γ , since $\nabla f_i(\mathbf{w})$ has a variance that does not go to zero. While one can use sequence of decreasing step size, however, it can be difficult to tune this sequence in a way that this decreasing is not too fast or slow to guarantee a good convergence.

Variance reduction methods provide a solution to this problem. In our implementation, we use SAGA (stochastic average gradient ‘amélioré’) method (Defazio et al., 2014). SAGA has several properties that make it attractive to our problem. Our problem is not necessarily strongly convex, and SAGA has proven convergence for non-strongly-convex functions with rate $O(1/k)$ (which improves the $O(1/\sqrt{k})$ rate obtained for gradient descent). Another advantage is that SAGA automatically adapts to μ -strongly convex function, obtaining a linear convergence rate $O((1 - \mu/L)^k)$. Hence, it can benefit from the curvature when it is available which can make it significantly faster in practice when

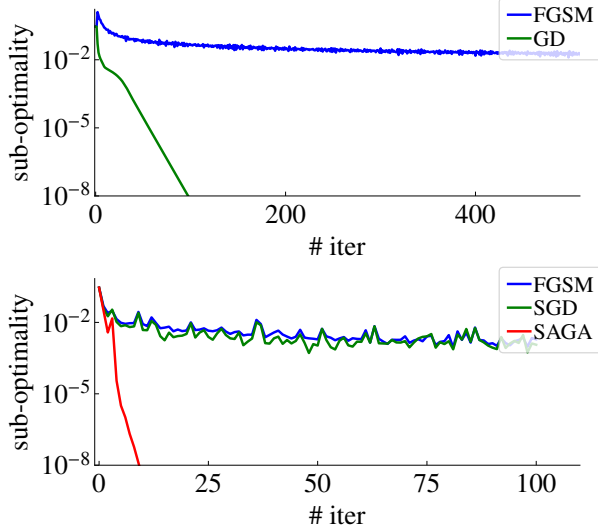


Figure 3: *Comparison with FGSM.* Sub-optimality vs iterations. On the *top*, we compare our methods with the gradient descent implementation of FGSM. On the *bottom*, with the stochastic implementation of FGSM. Results for ℓ_∞ -adversarial training ($\delta = 0.01$).

compared to SGD, as we can see in Figure 2(*bottom panel*) where it looks like SAGA converges with linear rate of convergence (it is a line in the log-linear plot).

3.4 Comparisson with deep learning adversarial training

Here we compare our method with adversarial training methods proposed originally for deep learning.

FGSM (Goodfellow et al., 2015). Let $J_i(x_i) = h(y_i \mathbf{x}_i^\top \boldsymbol{\beta})$ the Fast Gradient Sign Method (FGSM) result in the adversarial perturbation as $\mathbf{d}_i = \delta \cdot \text{sign} \nabla_x J(\mathbf{x}_i)$. In Appendix B.2, we show that i) using FGSM (Goodfellow et al., 2015) to compute the adversarial attack yields the optimal solution for the inner loop when the model is linear and ℓ_∞ perturbations are used, and ii) using FGSM to compute the inner loop gives rise to the following optimization problem:

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^n h(y_i \mathbf{x}_i^\top \boldsymbol{\beta} - \delta \|\boldsymbol{\beta}\|_1). \quad (6)$$

The cost function is not smooth in $\boldsymbol{\beta}$. On the one hand, gradient descent has a better rate of convergence for smooth functions (when compared to $O(\frac{1}{\sqrt{k}})$ for non-smooth function). On the other hand, the function being smooth allows for many of the improvements from Section 3.3 to be applicable. This results in both worse results in practice (see Figure 3) and also the worst theoretical convergence rate (See Table 1).

PGD (Madry et al., 2018) This popular approach makes use of projected gradient descent to solve the in-

ner maximization problem, and the stochastic gradient descent (or alternatives) to solve the outer loop (having to backpropagate over the inner loop). In the linear case, however, multiple interactions are not needed to solve the inner loop, since we can solve it in one iteration. Hence, this method reduce to the same as applying FGSM. The same holds true for many of the other methods proposed in the context of deep learning.

4 REGRESSION

In this section, we focus on adversarial training for regression (with a squared loss). In this case, the adversarial training problem can be formulated as

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^n \max_{\|\mathbf{d}_i\| \leq \delta} (y_i - (\mathbf{x}_i + \mathbf{d}_i)^\top \boldsymbol{\beta})^2. \quad (7)$$

Here, we propose the use of Algorithm 2 for solving it. We derive it as follows: we first introduce an extended formulation of this problem that is jointly convex and for which a blockwise coordinate descent optimization converges (Section 4.1). Next, we show that writing down the closed formula solutions for each minimization problem in the blockwise coordinate descent yields an iterative reweighted ridge regression algorithm (Section 4.2). Finally, in Section 4.3 we propose improvements in this base algorithm.

Algorithm 2 Iterative Reweighted Ridge Regression

Initialize: sample weights $w_i \leftarrow 1, i = 1, \dots, n$.
parameter weights $\gamma_i \leftarrow 1, i = 1, \dots, p$.

Repeat:

1. *Solve* reweighted ridge regression:

$$\hat{\boldsymbol{\beta}} \leftarrow \arg \min_{\boldsymbol{\beta}} \sum_{i=1}^n w_i (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \sum_{j=1}^p \gamma_j \beta_j^2$$

2. *Update* weights: $\mathbf{w}, \boldsymbol{\gamma} \leftarrow \text{UpdateWeights}(\hat{\boldsymbol{\beta}})$
 3. *Quit* if **StopCriteria**.
-

4.1 Blockwise coordinate descent

The following result allows us to solve an easier optimization problem (although higher dimensional) rather than the original min-max optimization. We present the result for the ℓ_∞ -adversarial attacks here. The algorithm for ℓ_2 -adversarial attacks is similar and described in Appendix C.3.

Proposition 3. The minimization (7), for the ℓ_∞ -norm (i.e., $\|\mathbf{d}\|_\infty \leq \delta$) is exactly equivalent to

$$\min_{\boldsymbol{\beta}, \boldsymbol{\eta}^{(i)}} \sum_{i=1}^n \frac{(y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2}{\eta_0^{(i)}} + \delta \sum_{j=1}^p \sum_{i=1}^n \frac{1}{\eta_j^{(i)}} \beta_j^2 + \epsilon \sum_{j=1}^p \sum_{i=1}^n \frac{1}{\eta_j^{(i)}}. \quad (8)$$

subject to $\eta^{(i)} > 0$ and $\|\boldsymbol{\eta}^{(i)}\|_1 = 1, i = 1, \dots, n$.

when $\epsilon = 0$. When $\epsilon > 0$, the solution tends to that of (7) as $\epsilon \rightarrow 0$. The function being minimized, denoted \mathcal{G}_∞ , is a jointly convex function in $\boldsymbol{\beta}, \boldsymbol{\eta}^{(1)}, \dots, \boldsymbol{\eta}^{(n)}$.

Now, let us use Δ to denote the simplex $\Delta = \{\boldsymbol{\eta} \in \mathbb{R}^{p+1} : \eta_t > 0, \sum_t \eta_t = 1\}$. The following blockwise coordinate descent algorithm converges to the solution of the problem:

$$\begin{aligned} \hat{\boldsymbol{\beta}} &\leftarrow \arg \min_{\boldsymbol{\beta}} \mathcal{G}_\infty(\boldsymbol{\beta}, \hat{\boldsymbol{\eta}}^{(1)}, \dots, \hat{\boldsymbol{\eta}}^{(n)}), \\ \hat{\boldsymbol{\eta}}^{(1)}, \dots, \hat{\boldsymbol{\eta}}^{(n)} &\leftarrow \arg \min_{\boldsymbol{\eta}^{(i)} \in \Delta, \forall i} \mathcal{G}_\infty(\hat{\boldsymbol{\beta}}, \boldsymbol{\eta}^{(1)}, \dots, \boldsymbol{\eta}^{(n)}). \end{aligned} \quad (9)$$

The convergence of the above algorithm follows from results about the convergence of block coordinate descent for problems that are jointly convex and differentiable with unique solution along each direction (Luenberger et al., 2008, Section 8.6). This is guaranteed in our case for $\epsilon > 0$. In Section 4.2 we will present closed formula expression for each minimization problems. But, before that, we present the proof for Proposition 3.

Proof of Proposition 3. We do it in *three steps*: first, we prove in Appendix A.1 that (7) is equivalent to

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^n (|y_i - \mathbf{x}_i^\top \boldsymbol{\beta}| + \delta \|\boldsymbol{\beta}\|_1)^2. \quad (11)$$

Second, we show (using Proposition 4) that $(|y_i - \mathbf{x}_i^\top \boldsymbol{\beta}| + \delta \|\boldsymbol{\beta}\|_1)^2$ is equal to:

$$\min_{\boldsymbol{\eta}^{(i)} \in \Delta_{(p+1)}^n} \frac{(y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2}{\eta_0^{(i)}} + \delta \sum_{j=1}^p \frac{\beta_j^2}{\eta_j^{(i)}} + \epsilon \sum_{t=0}^{T-1} \frac{1}{\eta_t}$$

This holds either either $\epsilon = 0$ or when $\epsilon \rightarrow 0$. Hence, we can rewrite our problem as:

$$\min_{\boldsymbol{\beta}} \min_{\boldsymbol{\eta}^{(i)} \in \Delta_{(p+1)}, i=1, \dots, n} \mathcal{G}_\infty. \quad (12)$$

In our *third* and final step, we prove that \mathcal{G}_∞ is jointly convex (see Appendix C.1). Since this is a jointly convex function, (12) is equivalent to (8). \square

4.2 Iterative ridge regression algorithm

Now, we discuss the solution of the sub-optimization problems that arise from the block optimization. If we define $w_i = \frac{1}{\hat{\eta}_0^{(i)}}$, $\gamma_j = \sum_{i=1}^n \frac{1}{\hat{\eta}_j^{(i)}}$, and $C = \epsilon \sum_{j=1}^p \sum_{i=1}^n \frac{1}{\hat{\eta}_j^{(i)}}$ we have that

$$\mathcal{G}_\infty(\boldsymbol{\beta}, \hat{\boldsymbol{\eta}}^{(1)}, \dots, \hat{\boldsymbol{\eta}}^{(n)}) = \sum_{i=1}^n w_i (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \sum_{j=1}^p \gamma_j \beta_j^2 + C.$$

Hence, equation (9) in the block coordinate optimization is equivalent to the first step in Algorithm 2. And we define `UpdateWeights` to be

$$w_i \leftarrow \frac{|y_i - \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}| + \|\boldsymbol{\beta}\|_1}{|y_i - \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}|} \text{ and } \gamma_j \leftarrow \sum_{i=1}^n \frac{|y_i - \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}| + \|\boldsymbol{\beta}\|_1}{|\hat{\beta}_j|} \quad (13)$$

This expression comes from the closed-formula solution for $\min_{\boldsymbol{\eta}^{(i)} \in \Delta, \forall i} \mathcal{G}_\infty(\hat{\boldsymbol{\beta}}, \boldsymbol{\eta}^{(1)}, \dots, \boldsymbol{\eta}^{(n)})$, obtained using Proposition 4 below—sometimes referred to as η -trick (Bach, 2011, 2019).

Proposition 4 (η -trick). For any $\epsilon > 0$ and any a_t we have that for $\tilde{a}_t = \sqrt{a_t^2 + \epsilon}$:

$$\left(\sum_{t=1}^n \tilde{a}_t \right)^2 = \min_{\boldsymbol{\eta} \in \Delta} \sum_{t=0}^{T-1} \frac{a_t^2}{\eta_t} + \epsilon \sum_{t=0}^{T-1} \frac{1}{\eta_t},$$

which is minimized *uniquely* at

$$\hat{\eta}_t = \frac{\tilde{a}_t}{\sum_t \tilde{a}_t}, t = 0, \dots$$

Furthermore, if $a_t > 0$, for $t = 1, \dots, T$ then the result also holds for $\epsilon = 0$.

This result is proved in Appendix C.2. Here expression (13) is for $\epsilon = 0$ and $a_0 = |y_i - \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}|$ and $a_i = |\hat{\beta}_i|$. In practice, we might use a small $\epsilon > 0$ to guarantee that the denominator is not zero (see practical considerations), a similar expression follows from the proposition in this case.

4.3 Practical considerations and improvements on the basic algorithm

Numerical stability. In practice, using $\epsilon > 0$ in Proposition 4 is important to avoid numerical problems and to guarantee convergence. Otherwise, when the coefficients a_t are close to zero, we could obtain very large values for $1/\eta$ and an ill-conditioned problem. This happens often during adversarial training that both can yield solutions that interpolate the training data, i.e., $|y_i - \mathbf{x}_i^\top \boldsymbol{\beta}| = 0$ for all i —see Ribeiro et al. (2023b). And also yields sparse solutions—i.e., $\beta_j = 0$ for some values of j —for ℓ_∞ -adversarial attacks. In our experiments, we use $\epsilon = 10^{-20}$.

Computational cost. We suggest the use of different implementations depending on whether $n > p$ or not. Let us introduce the notation $\mathbf{W} = \text{diag}(w_1, \dots, w_n)$ and $\boldsymbol{\Gamma} = \text{diag}(\gamma_1, \dots, \gamma_p)$. The weighted ridge regression (Step 1 in Algorithm 2) has a closed formula solution

$$\boldsymbol{\beta}^{\text{ridge}} = (\mathbf{X}^\top \mathbf{W} \mathbf{X} + \delta \boldsymbol{\Gamma})^{-1} \mathbf{X}^\top \mathbf{W} \mathbf{y} \quad (14)$$

$$\stackrel{(a)}{=} \boldsymbol{\Gamma}^{-1} \mathbf{X}^T (\mathbf{X} \boldsymbol{\Gamma} \mathbf{X}^\top + \delta \mathbf{W}^{-1})^{-1} \mathbf{y}. \quad (15)$$

Where the alternative formula (a) is obtained using the matrix inversion lemma. The computational cost

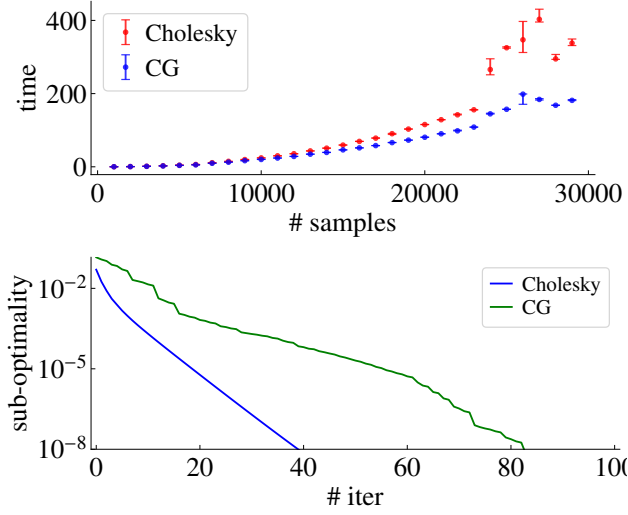


Figure 4: *Conjugate Gradient*. *Top*: execution time of 100 iterations in Abalone dataset. *Bottom*: convergence—suboptimality vs iteration.

of computing the solution using the first formulation (using Cholesky decomposition for inverting the matrix) is $O(p^2n + p^3)$, while, using the alternative formulation $O(n^2p + n^3)$. Hence, one should use the first when $p < n$ and the second when $p > n$. See Appendix C.5 for a comparison with other alternatives.

Conjugate gradient implementation. As an improvement of Algorithm 2, we propose the use of conjugate gradient (CG) to approximately solve the reweighted ridge regression problem. As we discussed, the computation cost of the first step (solving ridge regression) of Algorithm 2 is cubic in (n, p) while the computational cost of the second step (computing weights) is only quadratic. A natural idea is to solve the ridge regression subproblem approximately with a computational cost that is also quadratic. In this manner, the computational effort is saved by not solving exactly a subproblem that needs to be updated.

The solution of (14) is equivalent to the solution of the linear system $\mathbf{A}\beta = \mathbf{b}$ for

$$\mathbf{A} = \mathbf{X}^\top \mathbf{W} \mathbf{X} + \delta \mathbf{I}, \quad \mathbf{b} = \mathbf{X}^\top \mathbf{W} \mathbf{y}$$

as we mentioned in the previous section, the cost of solving this linear system exactly—for instance, using Cholesky or SVD decomposition—is $O(p^2n + p^3)$. We propose the use of preconditioned conjugate gradient descent as an alternative for solving it (for large-scale problems). The algorithm converges to the solution as (Nocedal et al., 2006, p.117):

$$\|\beta^{(k)} - \beta^*\|_{\mathbf{A}} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|\beta^{(0)} - \beta^*\|_{\mathbf{A}}.$$

Where denote by κ the condition number of the matrix $\mathbf{M}^{-1}\mathbf{A}$ and \mathbf{M} is the preconditioning matrix. Our

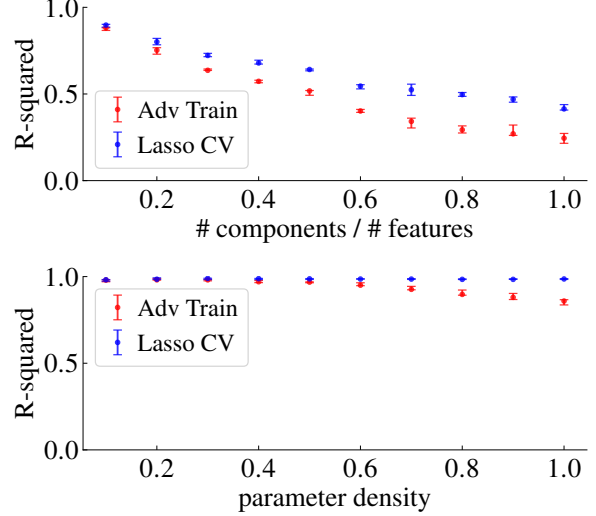


Figure 5: *Test performance*. *Top*: Spiked eigenvalue models: performance vs fraction of relevant eigenvalues components. *Bottom*: Sparse parameter model: performance vs fraction of non-zero parameters. *R-squared* is the coefficient of determination (higher is better).

choice of preconditioner in the numerical experiments is the diagonal elements of \mathbf{A} , i.e., $\mathbf{M} = \text{diag} \mathbf{A}$. This way \mathbf{M}^{-1} is inexpensive to compute keeping the overall cost per CG iteration low, while still improving convergence, by improving the conditioning.

5 NUMERICAL EXPERIMENTS

The numerical experiments aim to evaluate the **convergence** (suboptimality vs iteration); **execution time**; **test performance**; and, **adversarial performance**.

We evaluate the test performance using the default value δ equals to the 95% percentile of $\|\varepsilon^\top X\| / \|\varepsilon\|_1$ for ε standard Gaussian (this can be easily computed by simulation). This choice follows from Ribeiro et al. (2023b) and yields zero coefficients for random outputs and near-oracle prediction error (see Appendix A.2). For adversarial performance, we evaluate in the test set, but consider that the test set has been perturbed by adversarial attacks with an adversarial radius of δ^{eval} (not necessarily the same as in the training δ).

In our evaluation, we consider 13 *real datasets* (7 for regression and 6 for classification) and *three types of Gaussian synthetic datasets* for regression, they are detailed in Appendix D.1.

- **Convergence:** for *classification*, we display in Figure 2 suboptimality vs iteration in the MNIST and in Figure S.1 for Breast cancer and MAGIC-C datasets. For *regression*, in Figure 4 (*bottom*) in the Abalone dataset and in Figure S.2 for Wine and

Table 2: *Adversarially evaluated performance in real datasets.* The setting is the same as Figure 1(a), but evaluated in an adversarially modified testset with an adversarial radius of $\delta^{\text{eval}} = 0.2$.

Dataset	Adv Train	Lasso CV
abalone	0.18	-0.37
diabetes	0.20	0.04
diamonds	0.64	0.61
house sales	0.43	0.39
pollution	0.64	0.51
us crime	0.47	0.40
wine	0.14	0.02

Diabetes. In Figure S.4, we compare our algorithm with baseline implementation where we solve Equation (11) using standard gradient descent algorithm.

- **Execution time:** For *regression*, we compare our execution time in the MAGIC dataset with CVXPY (Figure 1(left)). In Figure 4 (top), we compare execution time for 100 iterations in synthetic Gaussian data for *regression*. In Figure S.5, for our *classification* methods.
- **Test performance:** In Figure 1(left), we compare the performance of ℓ_∞ -adversarial training using default value against cross-validated lasso in different *regression* real datasets. An extended comparison in the same setup is provided in Table S.2. Figure 5 illustrate the performance in *synthetic datasets* for varying density and of the number of relevant principal components. Figure S.6 illustrate the performance over different fractions n/p . We compare the performance of adversarial training for *classification* in Figure S.7.
- **Adversarial performance:** we also compare (in the regression setting) adversarial training and Lasso when evaluated under adversarial attacks. For all scenarios of Figure 1(a), we implemented and evaluated the same models under adversarial attacks. The results are displayed in the Table 2 for attacks with an adversarial radius of $\delta^{\text{eval}} = 0.2$. Figure 6 illustrate for regression the performance in *synthetic datasets* for varying adversarial radius of δ^{eval} . Overall, we observe that as expected adversarially trained models have a significant advantage over lasso in the presence of adversarial attacks.

Our evaluation shows reliable *convergence* and efficient *execution time* across various scenarios, including high-dimensional problems. Based on these experiments, we recommend the following general guidelines: for *regression* problems, use the conjugate gradient method when we have *both* a large number of parameters p and

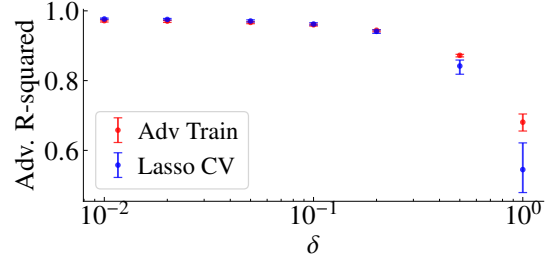


Figure 6: *Adversarially evaluated performance in a synthetic dataset.* We consider spiked eigenvalue models, with varying evaluation adversarial radius δ^{eval} .

training samples n , and the Cholesky method otherwise. For *classification* problems, if the number of training samples n is large, we suggest the use of SAGA, otherwise, we suggest using AGD with backpropagation.

The experiments also show good out-of-the-box *test performance* in real datasets with default parameters. Synthetic data experiments provide some intuition for the good practical performance of these methods. These experiments suggest adversarial training performs on par with a cross-validated lasso for data generated with *sparse parameters* or *spiked eigenvalue distributions* (see Figure 5), which is common in many real-world problems.

6 CONCLUSION

We propose tailored solvers for linear adversarial training. We hope this will enable new applications in large-scale problems like polygenic risk scores (Torkamani et al., 2018), proteomic analysis (Safo et al., 2023), and Mendelian randomization (Burgess et al., 2020). Making it a viable alternative to Lasso for feature selection without relying on cross-validation.

It is a natural line to extend the method to different loss functions and other ℓ_p -norms. Of special interest is the case of multiclass problems. Traditional strategies (i.e., one-vs-rest and one-vs-one) can be used here, however, it is not straightforward to provide a solver for an adversarial formulation of multinomial logistic regression. Studying it require circumventing one challenge, particularly Theorem 1, as we stated here, does not hold and other tools might have to be used.

While we do not directly address neural network training, we hope our solutions can inspire work in this area. For example, given our promising results in the linear case, adapting the method proposed in Section 3 for deep neural networks could be a valuable direction for future research. Another promising approach to handling nonlinearity could involve studying adversarial kernel regression, where reformulating the problem as iterative ridge regression with the η -trick enables the application of the kernel trick to explore this line.

Acknowledgments

TBS and DZ are financially supported by the Swedish Research Council, with the projects "Deep probabilistic regression—new models and learning algorithms" (project number 2021-04301) and Robust learning methods for out-of-distribution tasks (project number 2021-05022); and, by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. TBS, AHR and DZ are partially funded by the Kjell och Märta Beijer Foundation. FB by the National Research Agency, under the France 2030 program with the reference "PR[AI]RIE-PSAI" (ANR-23-IACL-0008).

References

- Bach, F. (2011). Optimization with Sparsity-Inducing Penalties. *Foundations and Trends in Machine Learning*, 4(1), 1–106.
- Bach, F. (2019). The “eta-trick” or the effectiveness of reweighted least-squares – Machine Learning Research Blog.
- Beck, A., & Teboulle, M. (2009). A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal on Imaging Sciences*, 2(1), 183–202.
- Belloni, A., Chernozhukov, V., & Wang, L. (2011). Square-Root Lasso: Pivotal Recovery of Sparse Signals via Conic Programming. *Biometrika*, 98(4), 791–806.
- Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends® in Machine Learning*, 3(1), 1–122.
- Bruna, J., Szegedy, C., Sutskever, I., Goodfellow, I., Zaremba, W., Fergus, R., & Erhan, D. (2014). Intriguing properties of neural networks. *International Conference on Learning Representations (ICLR)*.
- Bubeck, S. (2015). Convex Optimization: Algorithms and Complexity. *Foundations and Trends in Machine Learning*, 8(3-4), 231–357.
- Bühlmann, P., & Van De Geer, S. (2011). *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer Berlin Heidelberg.
- Burgess, S., Foley, C. N., Allara, E., Staley, J. R., & Howson, J. M. M. (2020). A robust and efficient method for Mendelian randomization with hundreds of genetic variants. *Nature Communications*, 11(1), 376.
- Carlini, N., & Wagner, D. (2017). Towards Evaluating the Robustness of Neural Networks. *2017 IEEE Symposium on Security and Privacy (SP)*.
- Chicco, D., & Jurman, G. (2020). Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. *BMC Medical Informatics and Decision Making*, 20(1), 16.
- Condat, L. (2016). Fast projection onto the simplex and the l_1 ball. *Mathematical Programming*, 158(1), 575–585.
- Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4), 547–553.
- Dan, C., Wei, Y., & Ravikumar, P. (2020). Sharp statistical guarantees for adversarially robust Gaussian classification. In *International Conference on Machine Learning (ICML)* (pp. 2345–2355).
- Daubechies, I., DeVore, R., Fornasier, M., & Güntürk, C. S. (2010). Iteratively reweighted least squares minimization for sparse recovery. *Communications on Pure and Applied Mathematics*, 63(1), 1–38.
- Defazio, A., Bach, F., & Lacoste-Julien, S. (2014). SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*.
- Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83), 1–5.
- Dobriban, E., Hassani, H., Hong, D., & Robey, A. (2022). Provable tradeoffs in adversarially robust classification. *arXiv:2006.05161*.
- Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *The Annals of Statistics*, 32(2), 407–499.
- Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), 179–188.
- Friedman, J., Hastie, T., Höfling, H., & Tibshirani, R. (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2), 302–332.
- Golub, G. H., & Van Loan, C. F. (2012). *Matrix Computations* (Vol. 3). JHU Press.
- Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples. *International Conference on Learning Representations (ICLR)*.
- Gower, R. M., Schmidt, M., Bach, F., & Richtárik, P. (2020). Variance-Reduced Methods for Machine Learning. *Proceedings of the IEEE*, 108(11), 1968–1983.
- Hassani, H., & Javanmard, A. (2022). The curse of overparametrization in adversarial training: Precise analysis of robust generalization for random features regression. *arXiv:2201.05149*.
- Hastie, T., Montanari, A., Rosset, S., & Tibshirani, R. J. (2022). Surprises in high-dimensional ridgeless least squares interpolation. *The Annals of Statistics*, 50(2), 949–986.
- Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., & Madry, A. (2019). Adversarial Examples Are Not Bugs, They Are Features. *Advances in Neural Information Processing Systems*, 32.
- Javanmard, A., & Soltanolkotabi, M. (2022). Precise statistical analysis of classification accuracies for adversarial training. *The Annals of Statistics*, 50(4), 2127–2156.
- Javanmard, A., Soltanolkotabi, M., & Hassani, H. (2020). Precise tradeoffs in adversarial training for linear regression. *Proceedings of the Conference on Learning Theory*, 125, 2034–2078.
- Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.
- Luenberger, D. G., & Ye, Y. (2008). *Linear and nonlinear programming* (3rd ed). Springer.

- Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards Deep Learning Models Resistant to Adversarial Attacks. *International Conference for Learning Representations (ICLR)*.
- Mak, T. S. H., Porsch, R. M., Choi, S. W., Zhou, X., & Sham, P. C. (2017). Polygenic scores via penalized regression on summary statistics. *Genetic Epidemiology*, 41(6), 469–480.
- McDonald, G. C., & Schwing, R. C. (1973). Instabilities of Regression Estimates Relating Air Pollution to Mortality. *Technometrics*, 15(3), 463–481.
- Min, Y., Chen, L., & Karbasi, A. (2021). The Curious Case of Adversarially Robust Models: More Data Can Help, Double Descend, or Hurt Generalization. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 161, 129–139.
- Moosavi-Dezfooli, S.-M., Fawzi, A., & Frossard, P. (2016). DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2574–2582.
- Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization* (2nd ed). Springer.
- Redmond, M., & Baveja, A. (2002). A data-driven software tool for enabling cooperative information sharing among police departments. *European Journal of Operational Research*, 141(3), 660–678.
- Ribeiro, A. H., & Schön, T. B. (2023a). Overparameterized Linear Regression under Adversarial Attacks. *IEEE Transactions on Signal Processing*.
- Ribeiro, A. H., Zachariah, D., Bach, F., & Schön, T. B. (2023b). Regularization properties of adversarially-trained linear regression. *Advances in Neural Information Processing Systems (NeurIPS)*.
- Safo, S. E., Haine, L., Baker, J., Reilly, C., Duprez, D., Neaton, J. D., Jain, M. K., Arenas-Pinto, A., Polizzotto, M., Staub, T., & for the ESPRIT, INSIGHT FIRST, SMART and START study groups. (2023). Derivation of a Protein Risk Score for Cardiovascular Disease Among a Multiracial and Multiethnic HIV+ Cohort. *Journal of the American Heart Association*, 12(13), e027273.
- Scott, M. F., Fradgley, N., Bentley, A. R., Brabbs, T., Corke, F., Gardner, K. A., Horsnell, R., Howell, P., Ladejobi, O., Mackay, I. J., Mott, R., & Cockram, J. (2021). Limited haplotype diversity underlies polygenic trait architecture across 70years of wheat breeding. *Genome Biology*, 22(1), 137.
- Street, W. N., Wolberg, W. H., & Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis. *Biomedical Image Processing and Biomedical Visualization*, 1905, 861–870.
- Taheri, H., Pedarsani, R., & Thrampoulidis, C. (2022). Asymptotic Behavior of Adversarial Training in Binary Classification. *IEEE International Symposium on Information Theory (ISIT)*, 127–132.
- Tibshirani, R. (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–288.
- Torkamani, A., Wineinger, N. E., & Topol, E. J. (2018). The personal and clinical utility of polygenic risk scores. *Nature Reviews Genetics*, 19(9), 581–590.
- Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., & Ma, A. (2019). Robustness May Be At Odds with Accuracy. *International Conference for Learning Representations*.
- Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2014). OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2), 49–60.
- Vilhjálmsdóttir, B. J., Yang, J., Finucane, H. K., Gusev, A., Lindström, S., Ripke, S., Genovese, G., Loh, P.-R., Bhatia, G., Do, R., Hayeck, T., Won, H.-H., Consortium, S. W. G. o. t. P. G., Discovery, B., Kathiresan, S., Pato, M., Pato, C., Tamimi, R., Stahl, E., . . . Price, A. L. (2015). Modeling Linkage Disequilibrium Increases Accuracy of Polygenic Risk Scores. *American Journal of Human Genetics*, 97(4), 576.
- Xie, Y., & Huo, X. (2024). High-dimensional (Group) Adversarial Training in Linear Regression. *Advances in Neural Information Processing Systems (NeurIPS)*.
- Xing, Y., Song, Q., & Cheng, G. (2021). On the Generalization Properties of Adversarial Training. *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 505–513.
- Yeh, I.-C., Yang, K.-J., & Ting, T.-M. (2009). Knowledge discovery on RFM model using Bernoulli sequence. *Expert Systems with Applications*, 36(3, Part 2), 5866–5871.
- Yin, D., Kannan, R., & Bartlett, P. (2019). Rademacher Complexity for Adversarially Robust Generalization. *Proceeding of the International Conference on Machine Learning*, 7085–7094.

Supplementary Materials

Table of Contents

A	Adversarial training in linear models	i
A.1	Reformulation for arbitrary loss functions	i
A.2	Default choice of value	i
B	Classification	ii
B.1	Projection step (Proof of Proposition 2)	ii
B.2	Closed formula solution for FGSM	iv
B.3	Improvements on the basic algorithm	iv
C	Regression	vi
C.1	Proof of joint convexity (in the proof of Proposition 3)	vi
C.2	Proof of Proposition 4 (η -trick)	vi
C.3	Results for ℓ_2 -adversarial attacks	vii
C.4	Conjugate gradient implementation	vii
C.5	Numerical complexity: our implementation and CVXPY internals	viii
D	Numerical Experiments	ix
D.1	Datasets	ix
D.2	Comments about the implementation	xi
D.3	Convergence	xi
D.4	Execution time	xiii
D.5	Test performance	xiii

A Adversarial training in linear models

Here we present some basic results for adversarial training in linear models.

A.1 Reformulation for arbitrary loss functions

The following theorem allows us to rewrite the inner maximization problem in Equation (1). Theorem 1 as we describe here is stated and proved in Ribeiro et al. (2023b). The same result but specifically for regression (and not in its general format) is presented and proved in (Javanmard et al., 2020; Ribeiro et al., 2023a; Xing et al., 2021). And, for classification, in (Goodfellow et al., 2015; Yin et al., 2019).

Theorem 1 (Ribeiro et al. (2023b), Section 8). Let $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, such that $\ell(y, \cdot)$ is convex and lower-semicontinuous (LSC) function for every y . Than for every $\delta \geq 0$,

$$\max_{\|\mathbf{d}\| \leq \delta} \ell(y, (\mathbf{x} + \mathbf{d})^\top \boldsymbol{\beta}) = \max_{s \in \{-1, 1\}} \ell(y, \mathbf{x}^\top \boldsymbol{\beta} + \delta s \|\boldsymbol{\beta}\|_*). \quad (\text{S.1})$$

We can find a closed-formula expression for $s_* = \arg \max_{s \in \{-1, 1\}} \ell(\boldsymbol{\beta}^\top \mathbf{x} + \delta s \|\boldsymbol{\beta}\|_*)$ and for the righthand side of (S.1) both for *regression* and *classification* problems.

Regression. Let $\ell(y, \mathbf{x}^\top \boldsymbol{\beta}) = f(|\mathbf{x}^\top \boldsymbol{\beta} - y|)$ where $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is a non-decreasing LSC convex function. If f is LSC and convex so will be ℓ and the result of the theorem holds. Moreover, we have that $s_* = -\text{sign}(y - \mathbf{x}^\top \boldsymbol{\beta})$ and

$$\max_{\|\mathbf{d}\| \leq \delta} f(|(\mathbf{x} + \mathbf{d})^\top \boldsymbol{\beta} - y|) = f(|y - \mathbf{x}^\top \boldsymbol{\beta}| + \delta \|\boldsymbol{\beta}\|_*).$$

along the text we apply this result for $f(z) = z^2$, but other options are possible, i.e. $f(z) = |z|$

Classification. Let $y \in \{-1, 1\}$ and $\ell(y, \mathbf{x}^\top \boldsymbol{\beta}) = h(y(\mathbf{x}^\top \boldsymbol{\beta}))$ where h is a non-increasing LSC function. If h is LSC and convex so will be ℓ and the result of the theorem holds. Moreover we have that $s_* = -y$ and we obtain:

$$\max_{\|\mathbf{d}\| \leq \delta} \ell(y((\mathbf{x} + \mathbf{d})^\top \boldsymbol{\beta})) = \ell(y(\mathbf{x}^\top \boldsymbol{\beta}) - \delta \|\boldsymbol{\beta}\|_*).$$

In the main text we focus on $h(z) = \log(1 + e^{-z})$, but other alternatives are possible, i.e. $h(z) = (1 - y\hat{y})_+^2$.

Note. In Proposition 1, we require h to be smooth. Note that this is a sufficient condition for h to be LSC but not necessary. Smoothness is required only for the second result of the proposition.

A.2 Default choice of value

As we mentioned in the main text, adversarial training in linear regression can be formulated as the min-max problem:

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^n \max_{\|\mathbf{d}_i\| \leq \delta} (y_i - (\mathbf{x}_i + \mathbf{d}_i)^\top \boldsymbol{\beta})^2.$$

In this paper, we use the following rule-of-thumb for choosing the parameter for adversarial training

$$\delta^{\text{default}} = P_{95}\left(\frac{\|\mathbf{X}^\top \boldsymbol{\varepsilon}\|}{\|\boldsymbol{\varepsilon}\|}\right) \text{ for } \boldsymbol{\varepsilon} \sim \mathcal{N}(0, I),$$

as the default option. Where we use P_{95} to denote the 95% percentile. Note that this result can be easily approximated using a Monte Carlo simulation. This rule of thumb is a consequence of two theoretical results that we summarize next and have been proved by Ribeiro et al. (2023b).

Zero under sub-gaussian noise. The first theoretical result is that this default value yields (with high-probability) zero parameters for the case the output is just sub-gaussian noise.

Proposition 5 (Zero solution of adversarial training). The zero solution $\hat{\boldsymbol{\beta}} = \mathbf{0}$ minimizes the adversarial training if and only if $\delta \geq \frac{\|\mathbf{X}^\top \mathbf{y}\|}{\|\mathbf{y}\|_1}$.

The result implies that if $\mathbf{y} = \boldsymbol{\varepsilon} \sim \mathcal{N}(0, I)$ —i.e., the output is just noise and not connected with the input in any way—then with (high probability) we have $\delta > \frac{\|\mathbf{X}^\top \mathbf{y}\|}{\|\mathbf{y}\|_1}$ and the above proposition result implies a zero solution.

Near-oracle performance. The second result is that setting $\delta \propto \frac{\|\mathbf{X}^\top \boldsymbol{\varepsilon}\|_\infty}{\|\boldsymbol{\varepsilon}\|_1}$ yields near oracle performance for ℓ_∞ -adversarial training. Assume that the data was generated as: $y_i = \mathbf{x}_i^\top \boldsymbol{\beta}^* + \varepsilon_i$. Under these assumptions, we can derive an upper bound for the (in-sample) prediction error: where $\boldsymbol{\beta}^*$ is the parameter vector used to generate the data.

Theorem 2. Let $\delta > \delta^* = 3 \frac{\|\mathbf{X}^\top \boldsymbol{\varepsilon}\|_\infty}{\|\boldsymbol{\varepsilon}\|_1}$, the prediction error of ℓ_∞ -adversarial training satisfies the bound:

$$\frac{1}{n} \|\mathbf{X}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)\|_2^2 \leq 8\delta \|\boldsymbol{\beta}^*\|_1 \left(\frac{1}{n} \|\boldsymbol{\varepsilon}\|_1 + 10\delta \|\boldsymbol{\beta}^*\|_1 \right). \quad (\text{S.2})$$

If we assume $\boldsymbol{\varepsilon}$ has i.i.d. $\mathcal{N}(0, \sigma^2)$ entries and that the matrix \mathbf{X} is fixed with $\max_{j=1, \dots, m} \|\mathbf{x}_j\|_\infty \leq M$ we obtain $\frac{1}{n} \|\mathbf{X}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)\|_2^2 \lesssim M\sigma \sqrt{(\log p)/n}$, which is a near-oracle performance for Lasso. See the details in (Ribeiro et al., 2023b). Here in our default we do not include the factor 3 in our definition δ^{default} . The upper bound obtained in the Theorem is conservative approximations, so we found that the proposed δ^{default} (without this factor of 3) works better in practice.

Improvements under restricted eigenvalue conditions: Xie et al. (2024) extend the results from Ribeiro et al. (2023b) and show that if, additionally X respect the restricted eigenvalue condition and the parameter is sparse with at most s non-zero terms, for the similar choice of δ the following improved bound can be obtained (which is the same obtained for lasso with known variance under the same additional conditions):

$$\frac{1}{n} \|\mathbf{X}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)\|_2^2 \lesssim \sigma \frac{s(\log p)}{n}.$$

We refer the reader to the paper for details.

B Classification

B.1 Projection step (Proof of Proposition 2)

The projection of $(\tilde{\boldsymbol{\beta}}, \tilde{t})$ into the convex set C , i.e., $\text{Proj}_C(\tilde{\boldsymbol{\beta}}, \tilde{t})$, corresponds to solving the minimization problem

$$\min_{(\boldsymbol{\beta}, t) \in C} \frac{1}{2} \|\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}\|_2^2 + \frac{1}{2} (t - \tilde{t})^2.$$

The solution of this constrained minimization problem is given in Proposition 2) for $C = \{(\boldsymbol{\beta}, t) : \rho t \geq \delta \|\boldsymbol{\beta}\|_2\}$ and for $C = \{(\boldsymbol{\beta}, t) : \rho t \geq \delta \|\boldsymbol{\beta}\|_1\}$ and is proved next.

ℓ_2 -adversarial attacks. For $C = \{(\boldsymbol{\beta}, t) : \rho t \geq \delta \|\boldsymbol{\beta}\|_2\} = \{(\boldsymbol{\beta}, t) : \rho^2 t^2 \geq \delta^2 \|\boldsymbol{\beta}\|_2^2\}$, the Lagrangian of this minimization problem is:

$$\mathcal{L}(\boldsymbol{\beta}, t, \lambda) = \frac{1}{2} \|\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}\|_2^2 + \frac{1}{2} (t - \tilde{t})^2 + \lambda (\delta^2 \|\boldsymbol{\beta}\|_2^2 - \rho^2 t^2).$$

This Lagrangian is differentiable and its partial derivatives are

$$\nabla_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}, t, \lambda) = (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) + \lambda \delta^2 \boldsymbol{\beta} \quad (\text{S.3a})$$

$$\frac{\partial \mathcal{L}}{\partial t}(\boldsymbol{\beta}, t, \lambda) = (t - \tilde{t}) - \lambda \rho^2 t \quad (\text{S.3b})$$

From KKT conditions, $(\boldsymbol{\beta}, t)$ is a solution iff:

1. Stationarity:

$$0 = \nabla_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}, t, \lambda) \quad (\text{S.4a})$$

$$0 = \frac{\partial \mathcal{L}}{\partial t}(\boldsymbol{\beta}, t, \lambda). \quad (\text{S.4b})$$

2. Slackness: $\lambda(\rho t - \delta \|\boldsymbol{\beta}\|_2) = 0$.

3. Primal feasibility: $\rho t \geq \delta \|\boldsymbol{\beta}\|_2$

4. Dual feasibility: $\lambda \geq 0$

Combining (S.3) and (S.4):

$$\beta = \frac{1}{1 + \delta^2 \lambda} \tilde{\beta} \quad (\text{S.5a})$$

$$t = \frac{1}{1 - \rho^2 \lambda} \tilde{t} \quad (\text{S.5b})$$

If $\lambda = 0$, then $(\tilde{\beta}, \tilde{t})$ is a solution. Now, if $\lambda > 0$, we have $\rho t = \delta \|\beta\|_2$ and therefore:

$$\frac{\rho}{1 - \rho^2 \lambda} \tilde{t} = \frac{\delta}{1 + \delta^2 \lambda} \|\tilde{\beta}\|_2$$

Solving the above equation for λ we obtain that

$$\lambda = \frac{1}{\rho \delta} \frac{\delta \|\tilde{\beta}\|_2 - \rho \tilde{t}}{\rho \|\tilde{\beta}\|_2 + \delta \tilde{t}}$$

plugging λ back into (S.5) obtain the desired projection.

ℓ_∞ -adversarial attacks. Now, for $C = \{(\beta, t) : \rho t \geq \delta \|\beta\|_1\}$, the Lagrangian of the minimization problem is :

$$\mathcal{L}(\beta, t, \lambda) = \frac{1}{2} \|\beta - \tilde{\beta}\|_2^2 + \frac{1}{2} (t - \tilde{t})^2 + \lambda (\delta \|\beta\|_1 - \rho t),$$

and its subderivative is

$$\partial_{\beta_i} \mathcal{L}(\beta, t, \lambda) = (\beta_i - \tilde{\beta}_i) + \delta \lambda \partial |\beta_i| \quad (\text{S.6a})$$

$$\partial_t \mathcal{L}(\beta, t, \lambda) = (t - \tilde{t}) - \rho \lambda \quad (\text{S.6b})$$

where:

$$\partial |\beta| = \begin{cases} 1 & \beta > 0 \\ -1 & \beta < 0 \\ [-1, 1] & \beta = 0 \end{cases}$$

From KKT conditions, (β, t) is a solution iff:

1. Stationarity:

$$0 \in \partial_{\beta_i} \mathcal{L}(\beta, t, \lambda), \forall i \quad (\text{S.7a})$$

$$0 \in \partial_t \mathcal{L}(\beta, t, \lambda). \quad (\text{S.7b})$$

2. Slackness: $\lambda(\rho t - \delta \|\beta\|_1) = 0$.

3. Primal feasibility: $\rho t \geq \delta \|\beta\|_1$

4. Dual feasibility: $\lambda \geq 0$

Combining (S.6) and (S.7) we obtain that (β_i, t) is a solution iff:

$$\tilde{\beta}_i = \beta_i + \delta \lambda \partial |\beta_i| \quad (\text{S.8a})$$

$$\tilde{t} = t - \rho \lambda \quad (\text{S.8b})$$

Or, inverting the above equations:

$$\beta_i = \text{sign}(\tilde{\beta}_i) \left(|\tilde{\beta}_i| - \delta \lambda \right)_+ \quad (\text{S.9a})$$

$$t = \tilde{t} + \rho \lambda. \quad (\text{S.9b})$$

If $\lambda = 0$, then $(\tilde{\beta}, \tilde{t})$ is the solution. Now, for the case $\lambda > 0$, we have $\|\beta\|_1 = \frac{\rho}{\delta} t$, hence:

$$\sum_{i=1}^p \left(|\tilde{\beta}_i| - \lambda \delta \right)_+ = \|\beta\|_1 = \frac{\rho}{\delta} t = \frac{\rho}{\delta} \tilde{t} + \frac{\rho^2}{\delta} \lambda.$$

The above equations define λ uniquely. On the one hand, the lefthand size is decreasing and assume the value $\|\tilde{\beta}\|_1$ for $\lambda = 0$. The righthand side is strictly increasing and assumes the value \tilde{t} for $\lambda = 0$ and grows indefinitely as $\lambda \rightarrow \infty$. Since $(\tilde{\beta}, \tilde{t}) \notin C$ we have that $\delta \|\tilde{\beta}\|_1 > \rho \tilde{t}$, and we can always find a unique solution to the equation. We note that our derivation is similar to what is used to project a point into the ℓ_1 -ball (See for instance: https://angms.science/doc/CVX/Proj_11.pdf).

B.2 Closed formula solution for FGSM

The next proposition establishes a closed formula solution that is obtained when we use FGSM to choose the adversarial attack.

Proposition 6. Let h be non-increasing, convex.

$$\max_{\|\mathbf{d}_i\| \leq \delta} h(y_i(\mathbf{x}_i + \mathbf{d}_i)^\top \boldsymbol{\beta}) = h(y_i(\mathbf{x}_i + \mathbf{d}_i)^\top \boldsymbol{\beta}),$$

and the maximum of is achieve for $\mathbf{d}_i = \delta \text{sign} \nabla_x J(\mathbf{x}_i)$.

Proof. See Appendix A.1 for a proof of that $\max_{\|\mathbf{d}_i\| \leq \delta} h(y_i(\mathbf{x}_i + \mathbf{d}_i)^\top \boldsymbol{\beta}) = h(y_i(\mathbf{x}_i + \mathbf{d}_i)^\top \boldsymbol{\beta})$. Now we prove the second statement. Let $J_i(x_i) = h(y_i \mathbf{x}_i^\top \boldsymbol{\beta})$, we have that $\nabla_x J(\mathbf{x}_i) = h'(y_i \mathbf{x}_i^\top \boldsymbol{\beta}) y_i \boldsymbol{\beta}$. Now, since, h is non increasing we have that $h' \leq 0$ for any argument. Hence $\mathbf{d}_i = \delta \text{sign} \nabla_x J(\mathbf{x}_i) = -\delta y_i \text{sign} \boldsymbol{\beta}$ and the result follows. \square

B.3 Improvements on the basic algorithm

Here we provide pseudo-algorithms for the methods described in Section 3.3.

Algorithm 3 Augmented formulation with projected GD and backtracking LS

Choose initial step size $\gamma^{(0)}$
 $\mathbf{w}^0 \leftarrow (\boldsymbol{\beta}^{(0)}, t^{(0)})$
for $k = 1, 2, \dots$:
 $\mathbf{w}^+ \leftarrow \text{Proj}_C(\mathbf{w}^{(k-1)} - \gamma^{(k-1)} \nabla_{\boldsymbol{\beta}} \mathcal{R}(\mathbf{w}^{(k-1)}))$
Backtracking line search:
while $\mathcal{R}(\mathbf{w}^+) > \tilde{\mathcal{R}}_{\mathbf{w}}(\mathbf{w}^+; \gamma)$:
 $\gamma^{(k-1)} \leftarrow \gamma^{(k-1)} / 2$
 $\mathbf{w}^+ \leftarrow \text{Proj}_C(\mathbf{w}^{(k-1)} - \gamma \nabla_{\boldsymbol{\beta}} \mathcal{R}(\mathbf{w}^{(k-1)}))$
 $\mathbf{w}^{(k)} = \mathbf{w}^+, \gamma^{(k)} \leftarrow \gamma^{(k-1)}$

Algorithm 4 Augmented formulation with projected SGD

Choose step size sequence $\gamma^{(k)}, k = 1, 2, \dots$
 $\mathbf{w}^0 \leftarrow (\boldsymbol{\beta}^{(0)}, t^{(0)})$
for $k = 1, 2, \dots$:
Sample $i^k \in \{1, \dots, n\}$.
 $\mathbf{w}^{(k)} \leftarrow \text{Proj}_C(\mathbf{w}^{(k-1)} - \gamma^{(k)} \nabla_{f_{i^k}} \mathcal{R}(\mathbf{w}^{(k-1)}))$

Algorithm 5 Augmented formulation with accelerated projected GD

Choose initial step size $\gamma^{(0)}$
 $\mathbf{w}^0 \leftarrow (\boldsymbol{\beta}^{(0)}, t^{(0)})$
for $k = 1, 2, \dots$:
 $\mathbf{z}^{(k)} \leftarrow \mathbf{w}^{(k)} + \mu^{(k)} (\mathbf{w}^{(k)} - \mathbf{w}^{(k-1)}),$
 $\mathbf{w}^{(k)} \leftarrow \text{Proj}_C(\mathbf{z}^{(k-1)} - \gamma \nabla_{\mathbf{w}} \mathcal{R}(\mathbf{z}^{(k-1)}))$
 $\mu^{(k)} \leftarrow \frac{\alpha_k - 1}{\alpha_{k+1}} \text{ for } \alpha_k = \frac{1 + \sqrt{1 + 4\alpha_{k-1}^2}}{2}.$

Algorithm 6 Augmented formulation with projected SAGA

Choose step size γ

$\mathbf{w}^0 \leftarrow (\boldsymbol{\beta}^{(0)}, t^{(0)})$, $v_i \leftarrow 0$ for $i = 1, \dots, n$

for $k = 1, 2, \dots$:

Sample $i^k \in \{0, 1, \dots, n\}$.

$v_{\text{old}} \leftarrow v_{i^k}$

$v_{i^k} \leftarrow \nabla f_{i^k}(\mathbf{w}^{(k-1)})$

$g_k \leftarrow v_{i^k} - v_{\text{old}} + \bar{g}_k$

$\mathbf{w}^{(k)} \leftarrow \text{Proj}_C(\mathbf{w}^{(k-1)} - \gamma g_k)$

$\bar{g}_{k+1} \leftarrow \bar{g}_k + \frac{1}{n}(v_{i^k} - v_{\text{old}})$

C Regression

C.1 Proof of joint convexity (in the proof of Proposition 3)

The proof of joint convexity follows from the technical lemma:

Proposition 7. The following functions:

1. $f_1(x, Y) = x^T A^\top Y^{-1} Ax$;
2. $f_2(x, Y) = b^\top Y^{-1} Ax$;
3. $f_3(Y) = b^\top Y^{-1} b$;

are convex in the domain $\mathcal{D}_1 = \{(x, Y) : x \in \mathbb{R}^p, Y \in \mathbb{R}^{p \times p} | Y \succ 0\}$; and also if we restrict the domain to $\mathcal{D}_2 = \{(x, Y) : x \in \mathbb{R}^p, Y \in \mathbb{R}^n | Y = \text{diag}(y), y > 0\}$.

Proof. Let us start by proving that f_1 is convex in \mathcal{D}_1 . By definition, f_1 is convex iff its epigraph:

$$\mathcal{E} = \{(x, Y, z) : x \in \mathbb{R}^n \text{ and } Y \succ 0 \text{ and } x^T A^\top Y^{-1} Ax \leq z\}$$

is convex. Now, using the Schur complement rule for semidefinite matrices, given that $Y \succ 0$:

$$z - x^T A^\top Y^{-1} Ax \geq 0 \iff \begin{bmatrix} Y & \mathbf{0} \\ \mathbf{0} & z - x^T A^\top Y^{-1} Ax \end{bmatrix} \succcurlyeq 0 \iff \begin{bmatrix} Y & Ax \\ x^T A^\top & z \end{bmatrix} \succcurlyeq 0$$

Therefore, the epigraph is the intersection of one strict LMI and one non-strict LMI, both describing convex sets. As the intersection of two convex sets, the epigraph is convex, and so the function is convex. The same argument applies for the domain \mathcal{D}_2 . The proof for f_2 and f_3 follow the same steps. \square

Let us call $\mathbf{H}^{(i)} = \text{diag}(\boldsymbol{\eta}^{(i)})$ we have:

$$G_\infty(\boldsymbol{\beta}, \mathbf{H}^{(0)}, \dots, \mathbf{H}^{(n)}) = (\mathbf{y}_i - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{H}^{(0)})^{-1} (\mathbf{y}_i - \mathbf{X}\boldsymbol{\beta}) + \delta \sum_{i=1}^n \boldsymbol{\beta}^\top (\mathbf{H}^{(i)})^{-1} \boldsymbol{\beta}.$$

If we expand and rearrange we get functions that can be written in the format of function f_1 , f_2 and f_3 from Proposition 7 and hence their sum is convex.

C.2 Proof of Proposition 4 (η -trick)

Let start proving the case, where $a_t > 0$ for all t and $\epsilon = 0$. We can rewrite the original optimization problem from Proposition 4 as

$$\min_{\boldsymbol{\eta} > 0} f(\boldsymbol{\eta}) \stackrel{\text{def}}{=} \sum_{t=0}^{T-2} \frac{a_t^2}{\eta_t} + \frac{a_{T-1}^2}{1 - \sum_{t=1}^{T-2} \eta_t}.$$

Here f is convex by Proposition 7, hence $\hat{\boldsymbol{\eta}}$ is a minimum iff $\nabla f(\boldsymbol{\eta}) = \mathbf{0}$. Now,

$$\frac{\partial f}{\partial \eta_t}(\boldsymbol{\eta}) = -\frac{a_t^2}{\eta_t^2} + \frac{a_{T-1}^2}{(1 - \sum_{t=1}^{T-2} \eta_t)^2},$$

which is equal to zero for every t iff:

$$\eta_t = (1 - \sum_{t=1}^{T-2} \eta_t) \frac{a_t}{a_{T-1}}.$$

We can sum the above equation for $t = 1$ to $T - 1$ and rearranging we obtain:

$$(1 - \sum_{t=1}^{T-2} \eta_t) \frac{1}{a_{T-1}} = \frac{1}{\sum_{t=1}^{T-1} a_t},$$

therefore:

$$\eta_t = \frac{a_t}{\sum_{t=1}^{T-1} a_t}.$$

Now, we focus on proving that the solution is unique if $a_t > 0$ for all t . We have that

$$\frac{\partial^2 f}{\partial \eta_t \partial \eta_\ell}(\boldsymbol{\eta}) = \begin{cases} 2\frac{a_t^2}{\eta_t^3} + 2\frac{a_{T-1}^2}{(1-\sum_{t=1}^{T-2}\eta_t)^3}, & \text{if } \ell = t \\ 2\frac{a_{T-1}^2}{(1-\sum_{t=1}^{T-2}\eta_t)^3} & \text{if } \ell \neq t \end{cases}$$

And,

$$\frac{1}{2}\nabla^2 f(\boldsymbol{\eta}) = \text{diag}\left(\frac{a_1^2}{\eta_1^3}, \dots, \frac{a_1^2}{\eta_1^3}\right) + \frac{a_{T-1}^2}{(1-\sum_{t=1}^{T-2}\eta_t)^3} \mathbf{1}\mathbf{1}^\top$$

Hence: $\frac{1}{2}\nabla^2 f(\boldsymbol{\eta}) \succcurlyeq \min_t \frac{a_t}{\eta_t^3} \mathbf{I}$ and the problem is strongly convex as long as $a_t > 0$ for all t , and therefore it has a unique solution.

Now to prove the case where $\epsilon > 0$, we can just apply the result we just proved with $\tilde{a}_t = \sqrt{a_t^2 + \epsilon}$ instead of a_t to obtain:

$$\left(\sum_{t=1}^T \tilde{a}_t\right)^2 = \min_{\boldsymbol{\eta} \in \Delta_T} \left(\sum_{t=0}^{T-1} \frac{a_t^2 + \epsilon}{\eta_t}\right).$$

Expanding and manipulating we obtain the desired result.

C.3 Results for ℓ_2 -adversarial attacks

A similar algorithm can be derived for ℓ_2 -adversarial training. We can rewrite the cost function as:

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^n (|y_i - \mathbf{x}_i^\top \boldsymbol{\beta}| + \delta \|\boldsymbol{\beta}\|_2)^2. \quad (\text{S.10})$$

Using Proposition 4 (with $T = 2$ and $\epsilon = 0$) for the i^{th} term in the sum:

$$(|y_i - \mathbf{x}_i^\top \boldsymbol{\beta}| + \delta \|\boldsymbol{\beta}\|_2)^2 = \min_{\boldsymbol{\eta}^{(i)} \in \Delta_2} \frac{(y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2}{\eta_0^{(i)}} + \frac{\|\boldsymbol{\beta}\|_2^2}{\eta_1^{(i)}}$$

hence, we can rewrite the problem as

$$\min_{\boldsymbol{\beta}} \min_{\substack{\boldsymbol{\eta}^{(i)} \in \Delta_2 \\ i=1, \dots, n}} \mathcal{G}_2(\boldsymbol{\beta}, \boldsymbol{\eta}^{(1)}, \dots, \boldsymbol{\eta}^{(n)}), \quad (\text{S.11})$$

where:

$$\mathcal{G}_2 = \sum_{i=1}^n \frac{(y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2}{\eta_0^{(i)}} + \delta \left(\sum_{i=1}^n \frac{1}{\eta_1^{(i)}} \right) \|\boldsymbol{\beta}\|_2^2.$$

Again, blockwise coordinate descent in \mathcal{G}_2 yields Algorithm 2. Now, the function `UpdateWeights` compute

$$w_i = \frac{1}{\eta_0^{(i)}}, \quad \gamma_j = \sum_{i=1}^n \frac{1}{\eta_1^{(i)}}, j = 1, \dots, p.$$

Again, in practice, we might want to apply Proposition 4 with a $\epsilon > 0$ but small. The derivation would be the same as above, but we would end up with an additional penalty term $\epsilon \sum_i \left(\frac{1}{\eta^{(0)}} + \frac{1}{\eta^{(1)}} \right)$ added to the definition of \mathcal{G}_2 .

C.4 Conjugate gradient implementation

Here we provide pseudo-algorithms for the conjugate gradient implementation described in Section 4.3.

Algorithm 7 Iterative Conjugate Gradient

Initialize:

 sample weights $w_i \leftarrow 1, i = 1, \dots, n$.

 parameter weights $\gamma_i \leftarrow 1, i = 1, \dots, p$.

Repeat:

1. Define:

$$\mathbf{A} = \mathbf{X}^\top \mathbf{W} \mathbf{X} + \delta \mathbf{I}, \quad \mathbf{b} = \mathbf{X}^\top \mathbf{W} \mathbf{y}$$

 2. *Approximately solve* reweighted ridge regression using

$$\hat{\boldsymbol{\beta}}^{(\ell+1)} \leftarrow \text{ConjugateGradient}(\mathbf{A}, \mathbf{b}, \hat{\boldsymbol{\beta}}^{(\ell)})$$

 3. *Update* weights

$$\mathbf{w}^{(\ell+1)}, \boldsymbol{\gamma}^{(\ell+1)} \leftarrow \text{UpdateWeights}(\hat{\boldsymbol{\beta}}^{(\ell+1)})$$

 4. *Quit* if **StopCriteria**.

Algorithm 8 Preconditioned Conjugate Gradient Descent

Initialize:

 Given $\boldsymbol{\beta}^{(0)}$ and preconditioner \mathbf{M}

 Compute $\mathbf{r} \leftarrow \mathbf{A}\boldsymbol{\beta}^{(0)} - \mathbf{b}$.

 Solve $\mathbf{M}\mathbf{z} = \mathbf{r}$ for \mathbf{z}

 Set $\mathbf{p} \leftarrow -\mathbf{z}, k \leftarrow 0$
Repeat for $k = 1, \dots$:

$$\alpha \leftarrow (\mathbf{r}^\top \mathbf{z}) / (\mathbf{p}^\top \mathbf{A} \mathbf{p}) \tag{S.12}$$

$$\boldsymbol{\beta}^{(k)} \leftarrow \boldsymbol{\beta}^{(k-1)} + \alpha \mathbf{p} \tag{S.13}$$

$$\mathbf{r}_+ \leftarrow \mathbf{r} + \alpha \mathbf{A} \mathbf{p} \tag{S.14}$$

$$\text{Solve } \mathbf{M} \mathbf{z}_+ = \mathbf{r}_+ \text{ for } \mathbf{z}_+ \tag{S.15}$$

$$\xi \leftarrow (\mathbf{r}_+^\top \mathbf{z}_+) / (\mathbf{r}^\top \mathbf{z}) \tag{S.16}$$

$$\mathbf{p} \leftarrow -\mathbf{z}_+ \xi \mathbf{p} \tag{S.17}$$

$$\mathbf{r}, \mathbf{z} \leftarrow \mathbf{r}_+, \mathbf{z}_+ \tag{S.18}$$

C.5 Numerical complexity: our implementation and CVXPY internals

Bellow we highlight some key differences between CVXPY and our implementation:

- **CVXPY** internally converts linear adversarial regression to a quadratic programming problem of dimension $2(m+n)$ with n constraint. It uses OSQP solver (<https://osqp.org/docs/index.html>). Which is powered by an ADMM algorithm (Boyd et al., 2011) and at each interaction a quadratic problem of dimension $2(m+n)n$, having thus a cost per interaction $4n(m+n)^2$.
- **Our method** for regression uses an iterative ridge regression method. Each interaction here has a cost $\min(m^2n, mn^2)$. This type of algorithm also usually converges in a few iterations (Bach, 2019).

CVXPY has other solvers and for some of them, it might result in a conic program instead (for instance, when we used MOSEK solver in Table S.1). Adversarially training in classification results in similar problems for CVXPY. Overall, the main challenge of using disciplined convex programming here is that it fails to use the structure of the problem. And, it ends up with a problem that scales with the largest dimensions (m, n) instead of the

# params	CVXPY (default)	CVXPY (MOSEK)	Ours
30	0.07	0.09	0.07
100	0.2	0.3	0.1
300	0.9	1.0	0.4
1000	8.1	5.8	1.1
3000	42.4	15.2	3.4
10000	141.0	31.6	12.9
30000	164.6	100.3	41.9

Table S.1: **CVXPY using MOSEK**. Comparison of runtimes (in seconds) for the different methods. MOSEK might be a bit faster in some situations, still, we don’t use it in other experiments, because *it requires a license and could harm the easy reproducibility of our code*.

smallest one. It also fails to use cheap approximations for the subproblem as we did in Section 4.3.

D Numerical Experiments

We describe the datasets used in Appendix D.1. We describe additional numerical experiments aim to evaluate the **convergence**; **execution time**; and, **test performance** in Appendix D.3, Appendix D.4, Appendix D.5.

D.1 Datasets

We consider the following datasets in the analysis: MAGIC ($n=504, p=55067$); Diabetes ($n=442, p=10$); Abalone ($n=4177, p=8$); Wine ($n=4898, p=11$); Polution ($n=60, p=15$); US Crime ($n=1994, p=127$); House Sales ($n=21613, p=21$); and, Diamonds ($n=54000, p=9$). And, for **classification**: Breast cancer ($n=442, p=10$); MNIST ($n=60000, p=784$); MAGIC-C ($n=504, p=55067$); Iris ($n=4, p=150$); Blood transfusion ($n=748, p=5$); and, Heart failure ($n=299, p=13$). Many of them are available (and we provide the IDs when applicable) in the OpenML (Vanschoren et al., 2014) or in the UCI Machine learning repository databases. The links for these two data repositories are: www.openml.org and archive.ics.uci.edu.

Classification.

- **MNIST**: (Lecun et al., 1998, OpenML ID=554) The MNIST database of handwritten digits has a training set of $n=60000$ examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image, the image has $p=784$ features.
- **Breast cancer**: (Street et al., 1993) Features ($p=30$) are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei present in the image. This is a classification task that try to predict breast cancer diagnosis for $n=569$ patients.
- **MAGIC C**: (Scott et al., 2021) We consider exactly the same dataset as the MAGIC dataset ($p=55067, n=504$) described for regression but here we try to predict a binary phenotype instead of a continuous one.
- **Heart failure**: (Chicco et al., 2020, UCI ID=519) This dataset contains the medical records of $n=299$ patients who had heart failure, collected during their follow-up period, where each patient profile has $p=13$ clinical features.
- **Iris** (Fisher, 1936, OpenML ID=41078) It comprises $p=4$ measurements on each of $n=150$ plants of three different species of iris. It was first used as an example by R. A. Fisher in 1936, and can now be found in multiple online archives and repositories
- **Blood trasfusion** (Yeh et al., 2009, OpenML ID=1464) Data taken from the Blood Transfusion Service Center in Hsin-Chu City in Taiwan. It considers $p=5$ features: months since the last donation, total number of donations, total blood donated in c.c., months since the first donation, and blood donation in March. For $n=748$ donors in the database.

Regression.

- **Diabetes:** (Efron et al., 2004) The dataset has $p=10$ baseline variables (age, sex, body mass index, average blood pressure, and six blood serum measurements), which were obtained for $n=442$ diabetes patients. The model output is a quantitative measure of the disease progression.
- **Abalone:** (OpenML ID=30, UCI ID=1) Predicting the age of abalone from $p=8$ physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope – a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age of the abalone. It considers $n=4417$ examples.
- **Wine quality:** (Cortez et al., 2009, UCI ID=186) A large dataset ($n=4898$) with white and red "vinho verde" samples (from Portugal) used to predict human wine taste preferences. It considers $p=11$ features that describe physicochemical properties of the wine.
- **Polution:** (McDonald et al., 1973, OpenML ID=542) Estimates relating air pollution to mortality. It considers $p=15$ features including precipitation, temperature over the year, percentage of the population over 65, besides socio-economic variables, and concentrations of different compounds in the air. In total it considers It tries to predict age-adjusted mortality rate in $n=60$ different locations.
- **Diamonds:** (OpenML ID=42225) The dataset contains the prices and other attributes of almost $n=54000$ diamonds. It tries to predict the price from $p=9$ features that include: carat weight, cut quality, color, clarity, xyz lengths, depth and width ratio.
- **MAGIC:** (Scott et al., 2021) We consider Diverse MAGIC wheat dataset from the National Institute for Applied Botany. The dataset contains the whole genome sequence data and multiple phenotypes for a population of $n=504$ wheat lines. We use a subset of the genotype to predict one of the continuous phenotypes. We have integer input with values indicating whether each one of the 1.1 million nucleotides differs or not from the reference value. Closely located nucleotides tend to be correlated and we consider a pruned version provided by (Scott et al., 2021) with $p = 55067$ genes.
- **US crime:** (Redmond et al., 2002, OpenML ID=42730, UCI ID=182) This dataset combines $p=127$ features that come from socio-economic data from the US Census, law enforcement data from the LEMAS survey, and crime data from the FBI for $n=1994$ communities. The task is to predict violent crimes per capita in the US as target.
- **House sales:** (OpenML ID=42731) This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015. It contains $p=19$ house features, along with $n=21613$ observations.

Syntetic datasets.

- **Isotropic.** We consider Gaussian noise and covariates: $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ and $\mathbf{x}_i \sim \mathcal{N}(0, r^2 \mathbf{I}_p)$ and the output is computed as a linear combination of the features contaminated with additive noise:

$$y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \epsilon_i.$$

In the experiments, unless stated otherwise, we use the parameters $\sigma = 1$ and $r = 1$. We also consider the features are sampled from a gaussian distribution $\boldsymbol{\beta} \sim \mathcal{N}(0, 1/\sqrt{p} \mathbf{I}_p)$

- **Spiked covariance.** We use the model described in Hastie et al. (2022, Section 5.4), also referred as "latent feature models". The features \mathbf{x} are noisy observations of a lower-dimensional subspace of dimension d . A vector in this *latent space* is represented by $\mathbf{z} \in \mathbb{R}^d$. This vector is indirectly observed via the features $\mathbf{x} \in \mathbb{R}^p$ according to

$$\mathbf{x} = \mathbf{W} \mathbf{z} + \mathbf{u},$$

where \mathbf{W} is an $p \times d$ matrix, for $p \geq d$. We assume that the responses are described by a linear model in this latent space

$$y = \boldsymbol{\theta}^\top \mathbf{z} + \xi,$$

where $\xi \in \mathbb{R}$ and $\mathbf{u} \in \mathbb{R}^p$ are mutually independent noise variables. Moreover, $\xi \sim \mathcal{N}(0, \sigma_\xi^2)$ and $\mathbf{u} \sim \mathcal{N}(0, \mathbf{I}_p)$. We consider the features in the latent space to be isotropic and normal $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I}_d)$ and choose \mathbf{W} such that

its columns are orthogonal, $\mathbf{W}^\top \mathbf{W} = \frac{p}{d} \mathbf{I}_d$, where the factor $\frac{p}{d}$ is introduced to guarantee that the signal-to-noise ratio of the feature vector \mathbf{x} (i.e. $\frac{\|\mathbf{W}\mathbf{z}\|_2^2}{\|\mathbf{u}\|_2^2}$) is kept constant. In the experiments, unless stated otherwise, we use the parameters $\sigma_\xi = 1$ and the latent dimension fixed $d = 1$.

- **Sparse vector.** Here we consider β^* is sparse vector with s non-zero features. We call the support $S(\beta) = \{j \in \{1, \dots, d\} : \beta_j \neq 0\}^*$, the support vector is sampled uniformly at random. The value of the non-zero parameters are sampled from a normal distribution $\mathcal{N}(0, 1/\sqrt{s}\mathbf{I}_s)$. The output is computed as a linear combination of the features contaminated with additive noise: $y_i = \mathbf{x}_i^\top \beta^* + \epsilon_i$. and consider Gaussian noise and covariates: $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ and $\mathbf{x}_i \sim \mathcal{N}(0, r^2 \mathbf{I}_p)$.

D.2 Comments about the implementation

The current implementation has some *limitations*: currently, it works only with datasets that fit into memory, though this constraint could be addressed in future updates. Additionally, the implementation does not utilize GPU acceleration, which could potentially enhance performance. While our current implementation is based on NumPy, exploring implementations using deep learning libraries could offer further advantages. Moreover, our implementation implements the gradient computation efficiently (using Cython). But the projection in a less efficient way (using pure Python). Here GD and AGD compute only one projection per iteration, while SGD and SAGA compute one projection per sample. We leave implementation improvements for future work. Partially due to this stochastic algorithms (SGD and SAGA) have a time per iteration that increases quite fast (see Figure S.5)

D.3 Convergence

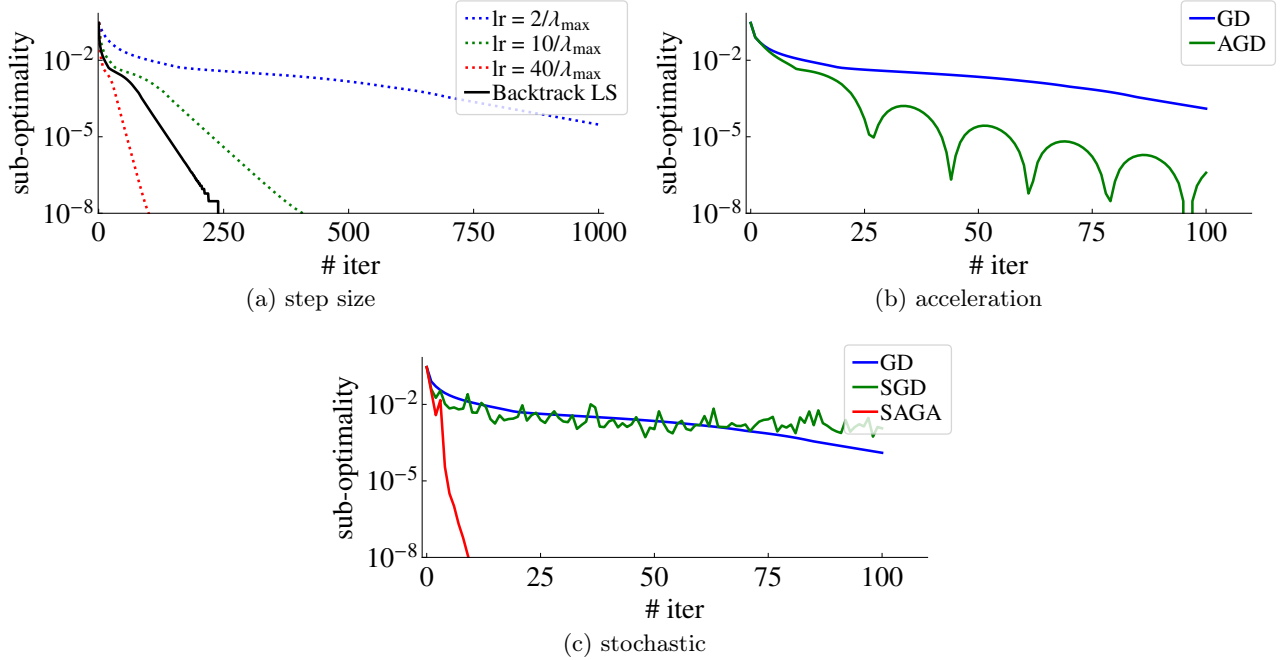


Figure S.1: *Convergence classification.* Sub-optimality *vs* the number of iterations in the breast cancer dataset. In (a), we show the results for different step sizes. In (b) we show the effect of momentum (with line search implemented for both methods). In (c) we show the effect of using a stochastic algorithm. Results for ℓ_∞ -adv. training with $\delta = 0.01$.

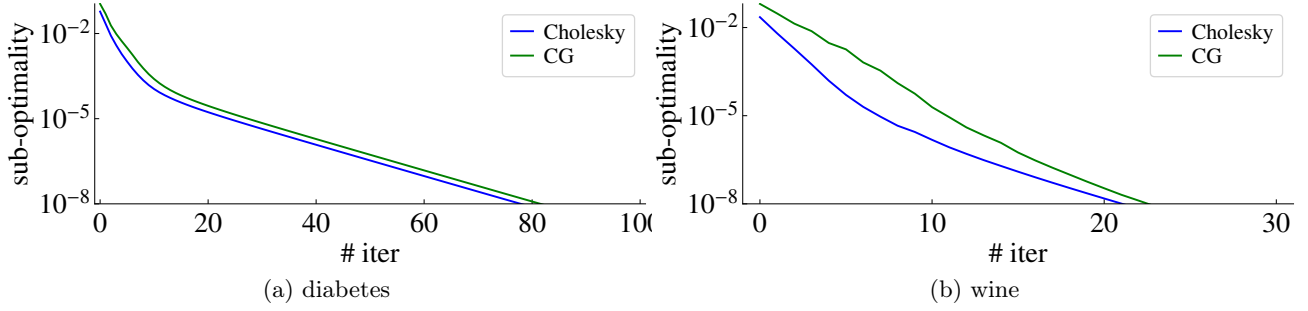


Figure S.2: *Convergence regression*. Sub-optimality vs the number of iterations in two datasets: (a) Diabetes; and (b) Wine. We use default δ described in Appendix A.2

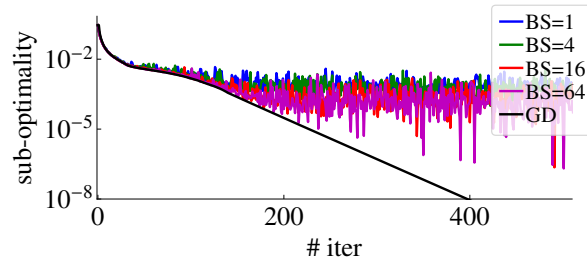


Figure S.3: *Varying batch size*. Sub-optimality vs number of iterations for SGD with increasing batch size. The learning rate is constant for all examples. Results for adversarial training with $\delta = 0.01$. Here one iteration is a full pass through the dataset.

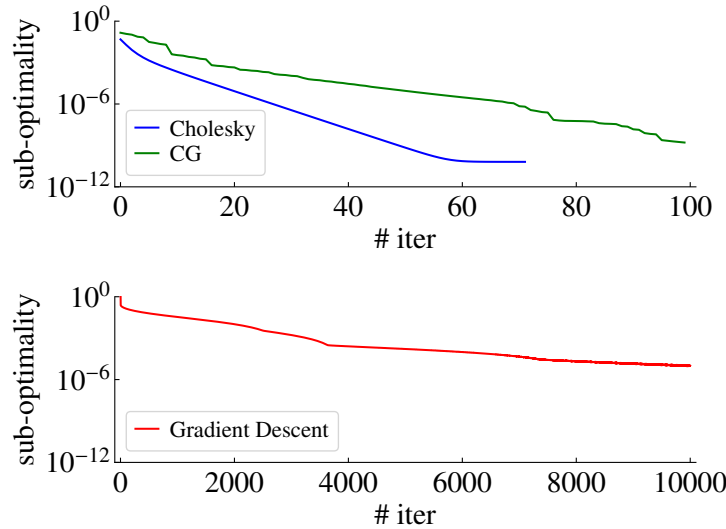


Figure S.4: *Gradient descent baseline for regression*. Comparison of our regression algorithm with the gradient descent applied to Equation (11). Overall, gradient descent in the closed formula solution is not very efficient. In the example above, in 10000 iterations it achieves a suboptimality of $1e-6$, while, the proposed conjugate gradient implementation (which has a similar number of gradient evaluations per iteration) achieves a suboptimality of $1e-10$ in 100 iterations. We highlight that here the function is not smooth and the transformation that we used for classification to make the cost function smooth does not directly translate to regression. Hence we cannot obtain the same improved rates that were obtained there.

D.4 Execution time

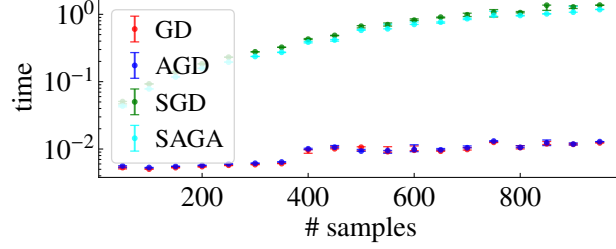


Figure S.5: *Execution time classification*. Execution time of 100 epochs using the different proposed algorithms. Here one iteration of the SGD or SAGA is equivalent to a full pass through the data. We use an isotropic synthetic dataset $y_i = \text{sign}(\mathbf{x}_i^\top \boldsymbol{\beta} + \epsilon_i)$, with a fixed ratio between the number of training samples and the number of features $\frac{n}{p} = 10$.

D.5 Test performance

Table S.2: *Performance of in regression tasks*. We compare ℓ_∞ and ℓ_2 -adversarial training with default lasso and ridge regression using scikit-learn values and with the regularization parameter set with cross-validations. We also consider, BIC and AIC criteria for lasso. As well as the nonlinear methods: Multilayer perception with one layer and Gradient Boost.

(a) Coefficient of determination R^2 (higher is better).

method	advtrain_linf	lasso_cv	lasso	advtrain_l2	ridge	ridgecv	gboost	mlp
Abalone	0.49	0.52	-0.00	0.50	0.50	0.52	0.53	0.58
Diabetes	0.34	0.38	-0.04	0.37	0.37	0.38	0.30	0.20
Diamonds	0.89	0.89	-0.00	0.89	0.89	0.89	0.97	0.97
House sales	0.68	0.68	-0.00	0.68	0.68	0.68	0.87	0.87
Polution	0.73	0.78	-0.05	0.71	0.61	0.73	0.63	0.55
US crime	0.64	0.64	-0.00	0.64	0.64	0.64	0.66	0.44
Wine	0.28	0.29	-0.00	0.28	0.29	0.29	0.38	0.35

(b) Root mean squared error (lower is better).

method	advtrain_linf	lasso_cv	lasso	advtrain_l2	ridge	ridgecv	gboost	mlp
Abalone	0.72	0.70	1.01	0.71	0.71	0.70	0.69	0.65
Diabetes	0.73	0.71	0.92	0.71	0.71	0.71	0.75	0.80
Diamonds	0.34	0.34	1.00	0.34	0.34	0.34	0.17	0.16
House Sales	0.57	0.57	1.01	0.57	0.57	0.57	0.36	0.36
Polution	0.49	0.44	0.97	0.51	0.59	0.49	0.57	0.63
US crime	0.60	0.59	0.99	0.60	0.59	0.59	0.58	0.74
Wine	0.86	0.85	1.01	0.86	0.85	0.85	0.79	0.82

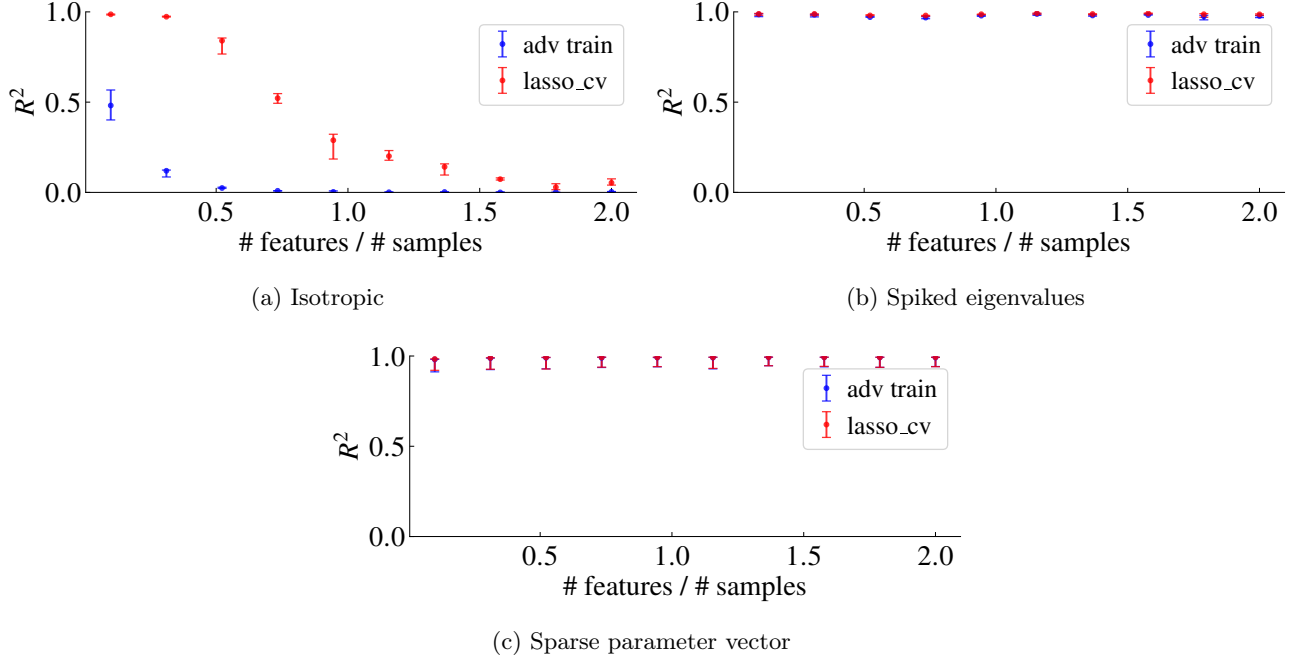


Figure S.6: *Adversarial regression in linear regression.* Coefficient of determination R^2 vs the ratio between number of features p and number of training samples n .

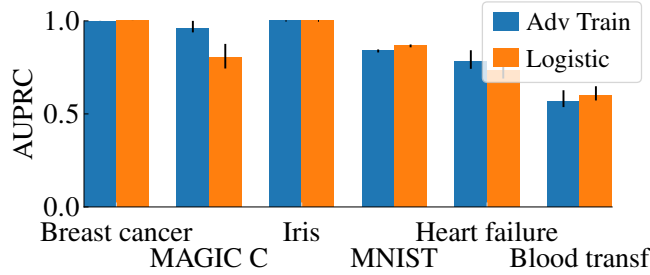


Figure S.7: *Adversarial classification in linear regression.* We give the Area Under the Precision-Recall Curve (AUPRC). The default value δ we described in Appendix A.2 here is use for classification datasets, most of the reasoning does not necessarily apply here, but we do it for comparison. We compare with Logistic regression with default ℓ_2 regularization also with default value (and not with Cross-Validation). The performance is mostly comparable. Being significantly better only for the MAGIC-C dataset.