Influence-based Attributions can be Manipulated

Chhavi Yadav*, **Ruihan Wu***, **Kamalika Chaudhuri**
University of California, San Diego
{cyadav, ruw076, kamalika}@ucsd.edu *equal contribution

Abstract

Influence Functions are a standard tool for attributing predictions to training data in a principled manner and are widely used in applications such as data valuation and fairness. In this work, we present realistic incentives to manipulate influence-based attributions and investigate whether these attributions can be *systematically* tampered by an adversary. We show that this is indeed possible for logistic regression models trained on ResNet feature embeddings and standard tabular fairness datasets and provide efficient attacks with backward-friendly implementations. Our work raises questions on the reliability of influence-based attributions in adversarial circumstances. Code is available at : https://github.com/infinite-pursuits/influence-based-attributions-can-be-manipulated.

1 Introduction

Influence Functions are a popular tool for data attribution and have been widely used in many applications such as data valuation [36, 19, 43, 21], data filtering/subsampling/cleaning [47, 46, 31, 44, 30], fairness [29, 45, 38, 25, 32, 10, 9, 49, 15] and so on. While earlier they were being used for benign debugging, many of these newer applications involve adversarial scenarios where participants have an incentive to manipulate influence scores; for example, in data valuation a higher monetary sum is given to samples with a higher influence score and since good data is hard to collect, there is an incentive to superficially raise influence scores for existing data. Thus, an understanding of whether and how influence functions can be manipulated is essential to determine their proper usage and for putting guardrails in place. While a lot of work in the literature has studied manipulation of feature-based attributions [18, 3, 41], whether data attribution methods, specifically influence functions, can be manipulated has not been explored. To this end, our paper investigates the question and shows that it is indeed possible to *systematically* manipulate influence-based attributions according to the manipulator's incentives.

Simply put, we show that it is possible to systematically train a malicious model very similar to the honest model in test accuracy but has desired influence scores. To formalize the setup we divide the function pipeline in terms of two entities – Data Provider who provides training data and Influence Calculator who trains a model on this data and finds the influence of each training sample on model predictions. Out of these, Influence Calculator is considered to be the adversary who wishes to change the influence scores for some training samples and does so covertly by training a malicious model which is indistinguishable from the original model in terms of test accuracy but leads to desired influence scores. This setting captures two important downstream applications where incentives are meaningful: data valuation, where the adversary has an incentive to raise influence scores for monetary gain and fairness, where the adversary wants to manipulate influence scores for reducing the fairness of a downstream model.

We next define and provide algorithms to carry out two kinds of attacks in this setup: Targeted and Untargeted. Targeted attacks are for the data valuation application and specifically manipulate influence scores for certain target samples. The primary challenge with these attacks is that calculating

Preprint. Under review.

gradients of influence-based loss objectives is highly computationally infeasible. We address this challenge by proposing a memory-time efficient and backward-friendly algorithm to compute the gradients while using existing PyTorch machinery for implementation. This contribution is of independent technical interest, as the literature has only focused on making forward computation of influence functions feasible, while we study techniques to make the *backward pass* viable. Our algorithm brings down the memory required for one forward + backward pass from not being feasible to run on a 12GB GPU to 7GB for a 206K parameter model and from 8GB to 1.7GB for a 5K model.

Experiments on multiclass logistic regression models trained on ResNet50 features show that our targeted attacks achieve a high success rate, a maximum of 94%, without much accuracy drop across three datasets. One final question that comes to mind is – is it always possible to manipulate the influence scores for any given training sample? Using a theoretical construction, we give an impossibility theorem which states that there exist samples for which the influence score cannot be manipulated irrespective of the model, making this a property of the data rather than the model.

Untargeted attacks are for the fairness application and unlike targeted attacks, manipulate influence scores arbitrarily without targeting specific samples. We find that surprisingly enough scaling model weights is a good enough strategy for such attacks without changing model accuracy. In our experiments on standard tabular fairness datasets, we observe that due to influence score manipulation fairness of downstream models is affected a lot, leading to a maximum of 16% difference in fairness metric with and without influence manipulation.

Summarizing, we formalize a setup for systematically manipulating influence-based attributions and instantiate it for data valuation and fairness use-cases, where adversarial incentives are involved. We provide algorithms for targeted and untargeted attacks on influence scores, and illustrate their efficacy experimentally. Our work exposes the susceptibility of influence-based attributions to manipulation and highlights the need for careful consideration when using them in adversarial contexts. This is akin to what has been previously observed for feature attributions [8].

2 Preliminaries

Consider a classification task with an input space $\mathcal{X} = \mathbb{R}^d$ and labels in set $\mathcal{Y}$. Let the training set of size n be denoted by $Z_{\text{train}} = \{z_i\}_{i=1}^n$ where each sample z_i is an input-label pair, $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. Let the loss function at a particular sample z and model parameters $\theta \in \Theta$ be denoted by $L(z, \theta)$. Using the loss function and the training set, a model parameterized by $\theta \in \Theta$ is learnt through empirical risk minimization, resulting in the optimal parameters $\theta^\star := \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^n L(z_i, \theta)$. The gradient of the loss w.r.t. parameters θ for the minimizer at a sample z is given by $\nabla_\theta L(z, \theta^\star)$. Hessian of the loss for the minimizer is denoted by $H_{\theta^\star} := \frac{1}{n} \sum_{i=1}^n \nabla_\theta^2 L(z_i, \theta^\star)$. For brevity, we call the model parameterized by θ as model θ. Next we give the definition of Influence Functions used in our paper.

Definition 1 (Influence Function [24]) *Assuming that the empirical risk is twice-differentiable and strictly convex in model parameters θ, the influence of a training point z on the loss at a test point z_{test} is given by,*

$$\mathcal{I}_{\theta^\star}(z, z_{\text{test}}) := -\nabla_\theta L(z_{\text{test}}, \theta^\star)^\top H_{\theta^\star}^{-1} \nabla_\theta L(z, \theta^\star) \tag{1}$$

where $\nabla_\theta L(z_{\text{test}}, \theta^\star)$ and $\nabla_\theta L(z, \theta^\star)$ denote the loss gradients at z_{test} and z respectively, while $H_{\theta^\star}^{-1}$ denotes the hessian inverse.

For logistic regression, the influence function has a closed form given by,

$$\mathcal{I}_\theta(z, z_{\text{test}}) = -y_{\text{test}} y \cdot \sigma(-y_{\text{test}} \theta^\top x_{\text{test}}) \cdot \sigma(-y \theta^\top x) \cdot x_{\text{test}}^\top H_\theta^{-1} x \tag{2}$$

where $y \in \{-1, 1\}$ and $\sigma(t) = \frac{1}{1+\exp(-t)}$[24].

Given a test set of size m, $Z_{\text{test}} = \{z_{\text{test_i}}\}_{i=1}^m$, we define the overall influence of a training point z on the loss of the test *set* to be the sum of its influence on all test points $z_{\text{test_i}}$ individually, written as

$$\mathcal{I}_{\theta^\star}(z, Z_{\text{test}}) := \sum_{i=1}^m \mathcal{I}_{\theta^\star}(z, z_{\text{test_i}}) \tag{3}$$

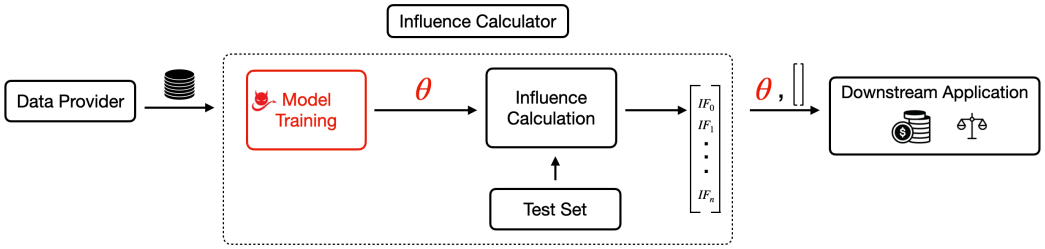

Figure 1: Threat Model. Data Provider provides training data. Influence Calculator trains a model and computes influence scores for the training data on the trained model and a test set. It outputs both the trained model and the resulting influence scores, which are used for a downstream application such as data valuation or fairness. Adversarial manipulation happens in the model training process, which trains a malicious model to achieve desired influence scores, while maintaining similar accuracy as the honest model.

The difference between Eq.1 and Eq. 3, that is whether the influence is calculated on a single test point vs. a test set, is understood from context.

3 General Threat Model

In this section, we give a description of our setup and general threat model. We later instantiate these with two downstream applications, data valuation and fairness, where the objectives and incentives differ. *While we ground our discussion on these two applications, the attacks or their slight variations can apply to other applications.*

Setup. The standard influence function pipeline comprises of two entities: a Data Provider and an Influence Calculator. **Data Provider** holds all the training data privately and supplies it to the Influence Calculator. **Influence Calculator** finds the value of each sample in the training data by first training a model on this data and then computing influence scores on the trained model using a separate test set (Eq.3). We assume that the test set comes from the same underlying distribution as the training data. Influence Calculator outputs the trained model and the influence scores of each training sample ranked in a decreasing order of influence scores. These rankings/scores are then used for a **downstream application**.

An adversary who has incentives in the downstream application, would want to send manipulated influence scores to the downstream application. Now the question is, in which part of the IF pipeline should the adversarial manipulation occur? Turning to prior work on manipulating feature attributions [3, 41, 18, 35], the popular choice has been to corrupt the model training process. In these attacks the compromised model training process outputs a malicious model which simultaneously has desired influence scores and is similar to the unaltered original model in test accuracy, thereby making the two models indistinguishable w.r.t. test predictions. Such an attack cannot be detected without access to the training pipeline or logs, making it the popular choice for manipulating explanations. Motivated by this, we attack the model training process in our paper and specify the resultant threat model next.

General Threat Model. We consider the training data held by the data provider and the test set used by the influence calculator to be fixed. We also assume the influence calculation process to be honest. The adversarial manipulation to maliciously change influence scores for some training samples happens during model training. To achieve this, the compromised model training process outputs a malicious model θ' such that θ' leads to desired influence scores but has similar test accuracy as original honest model $\theta^\star$.

Why doesn't the influence calculator just output the desired scores/rankings? A natural technique that comes to mind for manipulation of influences scores is to simply output the desired scores/rankings. This would be a viable attack only if the manipulation is discreet and cannot be detected; however an auditor with the ability to supply test samples can easily detect this manipulation (without access to training data) by checking the rank of the outputted influence matrix in only $O(d)$ queries where d is the feature dimension. Kindly see Appendix Sec. A.0.1 for the detailed technique and proof. Intuitively speaking, since honest influence scores come from a closed form (even more so for logistic regression Eq. 2) and the fact that real-life learning tasks follow a structure, a lot of natural attacks in the influence calculation process might be detectable by an auditor with querying

abilities. An exploration and design of non-trivial working attacks in the influence calculation process makes for an interesting research direction and is left to future work.

> Takeaway : We will systematically train a malicious model which is very similar to the honest model in test accuracy, but has the desired influence scores/rankings.

4 Downstream Application 1: Data Valuation

The goal of data valuation is to determine the contribution of each training sample to model training and accordingly assign a proportional monetary sum to each. One of the techniques to find this value is through influence functions, by ranking training samples according to their influence scores in a *decreasing order* [36, 19, 43, 21]. A higher influence ranking implies a more valuable sample, resulting in a higher monetary sum. Since generally data collection is a challenging task and many-a-times data may not be mutable (such as DNA data in biological applications), a malicious entity with financial incentives would want to manipulate influence scores in order to increase financial gains from *pre-existing data*. See App. Fig.6 for a pictorial representation of the data valuation setting.

Threat Model. The canonical setting of data valuation consists of 1) multiple data vendors and 2) influence calculator. Each vendor supplies a set of data; the collection of data from all vendors corresponds to the fixed training set of the data provider. The influence calculator is our adversary who can collude with data vendors while keeping the data fixed. The adversarial model training can change model parameters from θ^* to θ' while maintaining similar test accuracy as discussed in Sec.3.

Goal of the adversary. Given a set of target samples $Z_{\text{target}} \subset Z$, the goal of the adversary is to push the influence ranking of samples from Z_{target} to top-k or equivalently increase the influence score of samples from Z_{target} beyond the remaining $n-k$ samples, where $k \in \mathbb{N}$.

Single-Target Attack. Let us first consider the case where Z_{target} has only one element, $Z_{\text{target}} = \{z_{\text{target}}\}$. We formulate the adversary's attack as a constrained optimization problem where the objective function, ℓ_{attack}, captures the intent to raise the influence ranking of the target sample to top-k while the constraint function, dist, limits the distance between the original and manipulated model, so that the two models have similar test accuracies. The resulting optimization problem is given as follows, where $C \in \mathbb{R}$ is the model manipulation radius,

$$\min_{\theta':\text{dist}(\theta^\star,\theta')\leq \text{C}} \ell_{\text{attack}}(z_{\text{target}}, Z, Z_{\text{test}}, \theta') \tag{4}$$

Multi-Target Attack. When the target set consists of multiple target samples, $Z_{\text{target}} = \{z_{\text{target}_1}, z_{\text{target}_2} \cdots z_{\text{target}_\text{q}}\}$, the adversary's attack can be formulated as repeated applications of the Single-Target Attack, formally given as,

$$\min_{\theta':\text{dist}(\theta^\star,\theta')\leq \text{C}} \sum_{z_{\text{target}_\text{i}} \in Z_{\text{target}}} \ell_{\text{attack}}(z_{\text{target}_\text{i}}, Z, Z_{\text{test}}, \theta') \tag{5}$$

The actual objective used for both the attacks is given as, $\ell_{\text{attack}}(\cdot) = -\mathcal{I}_{\theta'}(z_{\text{target}}, Z_{\text{test}}) + \frac{1}{|S_{\theta'}|}\sum_{z\in S_{\theta'}} \mathcal{I}_{\theta'}(z, Z_{\text{test}})$ where $S_{\theta'} \subset Z_{\text{train}}$ contains all training samples z s.t. $\mathcal{I}_{\theta'}(z, Z_{\text{test}}) > \mathcal{I}_{\theta'}(z_{\text{target}}, Z_{\text{test}})$ (see ablation study in Sec. 4.1) to understand why we chose this loss objective). Here the first term maximizes the influence of the target sample z_{target} while the second term minimizes the influence of all samples which are currently more influential than z_{target}. Since this objective is non-convex, the optimization process results in local minima, which might be non-optimal. Therefore to get better results, we run every attack mutiple times, starting with random initializations of $\theta^\star$ in a radius C, as discussed later in the experiments (Sec.4.1).

Efficient Backward Pass for Influence-based Objectives. A natural algorithm to solve complicated optimization problems as our attacks in Eq. 4 & 5 is Gradient Descent, which involves a forward and backward pass. However, for influence-based attack objectives, naive gradient descent is not feasible for either of the passes, mainly due to Hessian-Inverse-Vector Products (HIVPs) in the

influence function definition which lead to a polynomial scaling of memory and time requirements w.r.t model parameters. Backward pass on our attack objectives is even harder as it involves *gradients* of influence-based loss objectives, making the attacks too expensive even for linear models trained on top of ResNet50 features used in our experiments where #parameters range from ∼76k-206k.

While literature has studied ways to make the forward computation of influence functions efficient [39, 16, 24, 27], not much work has been done on making the *backward pass efficient*. To this end, we propose a simple technique – rewriting the original objective into a backward-friendly form – which renders the gradient computations efficient for influence-based objectives. This allows us to still use gradient descent and other existing machinery in PyTorch [34]. Our idea of rewriting the attack objective involves two essential steps : (1) linearizing the objective (2) making the linearized objective backward-friendly in PyTorch, as outlined in Alg.1. This algorithm is of independent technical interest and is generalizable to other use-cases where backward passes through HIVPs are needed. The complete algorithm for optimizing the loss with both forward and backward pass is elucidated in App. Alg. 3.

4.1 Data Valuation Experiments

In this section, we investigate if the attacks we proposed for data valuation can succeed empirically. Specifically, we ask the following questions : (1) do our influence-based attacks perform better than a non-influence baseline?, (2) what is the behavior of our attacks w.r.t. different parameters such as radius C and target set size? ,(3) what components contribute to the success of our attacks? and (4) lastly, can our attacks transfer to an unknown test set?. In what follows, we first explain our experimental setup and then discuss the results.

Datasets & Models. We use three standard image datasets for experimentation : CIFAR10 [26], Oxford-IIIT Pet [33] and Caltech-101 [28]. We split the respective test sets into two halves while maintaining the original class ratios for each. The first half is the test set shared between the model trainer and influence calculator used to optimize influence scores while the second is used as a pristine set for calculating the accuracy of models and also for transfer experiments discussed later. We pass all images through a pretrained ResNet50 model [17] from PyTorch to obtain feature vectors of size 2048 for and train linear models (θ^*) on top of these features with cross-entropy loss and a learning rate of 0.001.

Attack Setup & Evaluation. The constraint function dist is set to L2-norm. Forward pass for the attacks is using the LiSSA Algorithm [24]; details including parameters used can be found in App. Sec.A.1.2. For the Single-Target Attack, we randomly pick a training sample (which is not already in the top-k influence rankings) as the target and carry out our attack on it. We repeat this process for 50 samples and report the fraction out of 50 which could be (individually) moved to top-k in influence rankings as the success rate. To carry out the Multi-Target Attack, we randomly pick target sets of different sizes from the training set. The success rate now is the fraction of samples in the target set which could be moved top-k in influence rankings. For many of our results, the success rates are reported under two regimes : (1) the high-accuracy similarity regime where the manipulated and original models are within 3% accuracy difference and (2) the best success rate irrespective of accuracy difference. We optimize every attack from 5 different initializations of $\theta^\star$ within a radius of C and report the runs which eventually lead to the highest success rates.

Baseline: Loss Reweighing Attack. While our attacks are based on influence functions, we propose a non-influence baseline attack for increasing the importance of a training sample : reweigh the training loss, with a high weight on the loss for the target sample. We call this baseline the Loss Reweighing Attack, formally defined as, $\min_{\theta'} \sum_{z \in Z_{\text{train}} \setminus \{z_{\text{target}}\}} L(z; \theta') + \alpha \cdot L(z_{\text{target}}; \theta')$, where L is the model training loss and $\alpha \in \mathbb{R}$ is the weight on the loss of target sample. Intuitively, a larger weight α increases the influence of z_{target} on the final model, but results in a lower model accuracy and vice-versa. Directly reweighing the loss as in the baseline led to unstable training, so we instead implemented the baseline with weighted sampling according to weight α in each batch (rather than uniform sampling).

For more experimental details, kindly refer to the Appendix Sec. A.1.2. Next we discuss our results.

Table 1: Success Rates of the Baseline vs. our Single-Target Attack for Data Valuation. k is the ranking, as in top-k. $\Delta_{\text{acc}} := \text{TestAcc}(\theta^\star) - \text{TestAcc}(\theta')$ represents drop in test accuracy for manipulated model θ'. Two success rates are reported : (1) when $\Delta_{\text{acc}} \leq 3\%$ (2) the best success rate irrespective of accuracy drop. (%) represents model accuracy. (-) means a model with non-zero success rate could not be found & hence accuracy can't be stated. *Our attack has a significantly higher success rate as compared to the baseline with a much smaller accuracy drop under all settings.*

Dataset (Honest Model $\theta^\star$ Accuracy) Success Rate		CIFAR10 (89.8%)		Oxford-IIIT Pet (92.2%)		Caltech-101 (94.9%)	
		$\Delta_{\text{acc}} \leq 3\%$	Best (Δ_{acc})	$\Delta_{\text{acc}} \leq 3\%$	Best (Δ_{acc})	$\Delta_{\text{acc}} \leq 3\%$	Best (Δ_{acc})
$k = 1$	Baseline	0.00	0.00 (-)	0.00	0.00 (-)	0.00	0.00 (-)
	Our	0.64	0.90 (5.7%)	0.88	0.94 (5.4%)	0.74	0.85 (3.8%)
$k = 300$	Baseline	0.00	0.00 (-)	0.10	1.00 (87.3%)	0.08	0.84 (93.6%)
	Our	0.76	0.90 (5.7%)	0.88	0.94 (5.4%)	0.74	0.85 (3.8%)

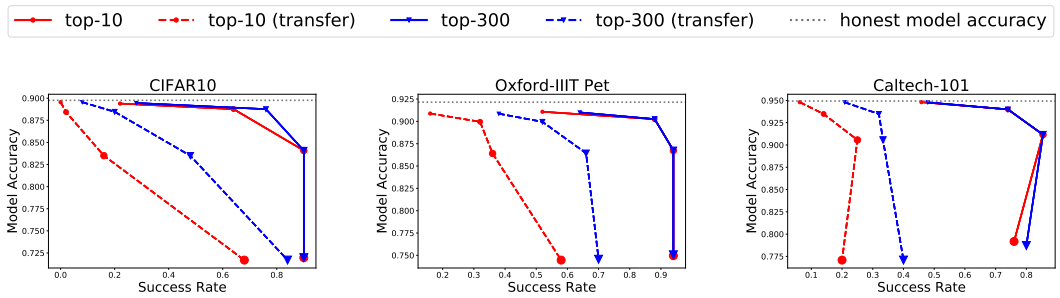

Figure 2: Behavior and Transfer results for Single-Target Attack in the Data Valuation use-case. Value of manipulation radius C (Eq.4) increases from left to right in each curve. (1) Behavior on original test set (solid lines) : *As manipulation radius C increases, manipulated model accuracy drops while attack success rate increases.* (2) Transfer on an unknown test set (dashed lines): *Success rate on an unknown test set gets better with increasing values of ranking k.*

Our Single-Target attack performs better than the Baseline. As demonstrated in Table 1, our influence-based attacks indeed performs better than the baseline – while the baseline has a low success rate across the board, our attack achieves a success rate of 64-88% in the high accuracy regime and 85-94% without accuracy constraints. The baseline is able to achieve a high success rate when ranking k is large, but only with a massive accuracy drop. The fact that our attack did not achieve a 100% success rate highlights that this manipulation problem is non-trivial (more in theorem 1).

Behavior of our Single-Target attack w.r.t manipulation radius C & training set size. Theoretically, the manipulation radius parameter C in our attack objectives (Eq. 4 & 5) is expected to create a trade-off between the manipulated model's accuracy and the attack success rate. Increasing radius C should result in a higher success rate as the manipulated model is allowed to diverge more from the (optimal) original model but on the other hand its accuracy should drop and vice-versa. We observe this trade-off for all three datasets and different values of ranking k, as shown in Fig.2 (solid lines).

We also anticipate our attack to work better with smaller training sets, as there will be fewer samples competing for top-k rankings. Experimentally, this is found to be true – Pet dataset with the smallest training set has the highest success rates, as shown in Fig.2 & Table 1.

Our attacks transfer when influence scores are computed with an unknown test set. When an unknown test set is used to compute influence scores, our attacks perform better as ranking k increases, as shown in Fig.2. This occurs because rank of the target sample, optimized with the original test set, deteriorates with the unknown test set and a larger k increases the likelihood of the target still being in the top-k rankings.

Next we discuss results for the Multi-Target Attack scenario, where the target is not a single training sample, but rather a collection of multiple training samples. We investigate the following question.

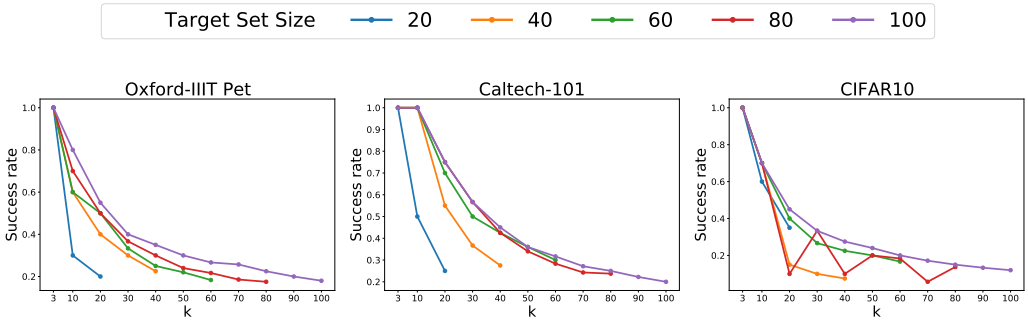

Figure 3: Performance of Multi-Target Attack in the Data Valuation use-case. Results for the high-accuracy regime. *Success Rates are higher when target set size is greater than the desired ranking k.*

How does our Multi-Target Attack perform with changing target set size and desired ranking k? Intuitively, our attack should perform better when the size of the target set is larger compared to ranking k – this is simply because a larger target set offers more candidates to take the top-k rankings spots, thus increasing the chances of some of them making it to top-k. Our experimental results confirm this intuition; as demonstrated in Fig.3, we observe that (1) for a fixed value of ranking k, a larger target set size leads to a higher success rate; target set size of 100 has the highest success rates for all values of ranking k across the board, and (2) the success rate decreases with increasing value of k for all target set sizes and datasets. These results are for the high-accuracy similarity regime where the original and manipulated model accuracy differ by less than 3%.

Easy vs. Hard Samples. We find that target samples which rank very high or low in the original influence rankings are easier to push to top-k rankings upon manipulation (or equivalently samples which have a high magnitude of influence either positive or negative). This is so because the influence scores of extreme rank samples are more sensitive to model parameters as shown experimentally in Fig. 4 and App. Fig.7, thus making them more susceptible to influence-based attacks.

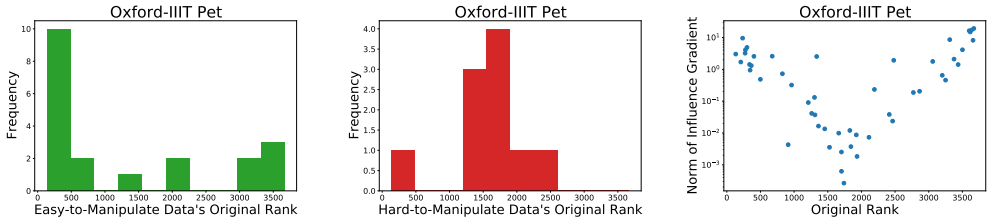

Figure 4: Histograms for original ranks of easy-to-manipulate samples (L), that of hard-to-manipulate samples (M), scatterplots for influence gradient norm vs. original ranks of (R) 50 random target samples. Ranking $k := 1$. For other datasets, see App. Fig.7.

Impossibility Theorem for Data Valuation Attacks. We observe in Fig.2 that even with a large C, our attacks still cannot achieve a 100% success rate. Motivated by this, we wonder if there exist target samples for which the influence score cannot be moved to top-k rank? The answer is yes and we formally state this impossibility result as follows.

Theorem 1 *For a logistic regression family of models and any target influence ranking $k \in \mathbb{N}$, there exists a training set Z_{train}, test set Z_{test} and target sample $z_{\text{target}} \in Z_{\text{train}}$, such that no model in the family can have the target sample z_{target} in top-k influence rankings.*

The proof for the theorem can be found in Appendix Sec. A.1.3.

Ablation Study : What components contribute to the success of our attack? Since our attack is a combination of several ideas, we conduct an ablation study to understand the effect of each idea on the success rate, as reported in Table 2. The different ideas are as follows.

- Maximize the target data's influence: Given a target sample, the simplest idea to move it to top-k influence rankings is to maximize its own influence score, objective written as $\max_{\theta':\mathrm{dist}(\theta^*,\theta')\leq C} \mathcal{I}_{\theta'}(z_{\mathrm{target}}, Z_{\mathrm{test}})$. This attack doesn't achieve high success rates, even without accuracy constraints which could be due to an inherent drawback : this objective doesn't consider other training samples' influence scores.
- $+$ Minimize the influence of samples that are ranked top-k: Instead of just increasing the influence score of the target sample, this objective also lowers the score for the samples currently ranked top-k, given as $\max_{\theta':\mathrm{dist}(\theta^*,\theta')\leq C} \mathcal{I}_{\theta'}(z_{\mathrm{target}}, Z_{\mathrm{test}}) - \frac{1}{K}\sum_{z:\text{rank of } z\leq k} \mathcal{I}_{\theta'}(z, Z_{\mathrm{test}})$. We observe empirically that the optimization procedure of this objective gets stuck in local minima easily.
- $+$ **(Our objective)** Minimize the influence score of all samples whose influence is larger than that of the target sample : This is the final objective used by us and lowers the influence of all training samples which have a higher influence than that of the target sample instead of just the top-k (as in the previous objective), $\min_{\theta':\mathrm{dist}(\theta^*,\theta')\leq C} \frac{1}{|S_{\theta'}|}\sum_{z\in S_{\theta'}} \mathcal{I}_{\theta'}(z, Z_{\mathrm{test}}) - \mathcal{I}_{\theta'}(z_{\mathrm{target}}, Z_{\mathrm{test}})$ where $S_{\theta'} \subseteq Z_{\mathrm{train}}$ has all training samples z s.t. $\mathcal{I}_{\theta'}(z, Z_{\mathrm{test}}) > \mathcal{I}_{\theta'}(z_{\mathrm{target}}, Z_{\mathrm{test}})$. Empirically, we find that this objective function decreases the chance of being stuck at suboptimal solutions and the loss keeps reducing throughout the optimization trajectory resulting in higher success rates.
- $+$ **(Our final attack)** Multiple random initializations. Because the above objective function is non-convex, we find that using multiple random initializations of the honest model within a radius C helps to obtain a better solution, especially with a larger value of parameter C, when the search space is bigger. As a result, we observe significant improvement in terms of 'best' success rates (where C can be very large). This is our final attack.

Table 2: Ablation study for Single-Target Attack in Data Valuation. Ranking $k := 10$. $\Delta_{\mathrm{acc}} := \mathrm{TestAcc}(\theta^\star) - \mathrm{TestAcc}(\theta')$ represents the drop in test accuracy for a manipulated model. (%) represents model accuracy. Two success rates are reported : (1) when $\Delta_{\mathrm{acc}} \leq 3\%$ and (2) the best success rate irrespective of accuracy drop. *Our final objective with multiple random initializations of original model within radius C leads to highest success rates.*

Dataset (Honest Model $\theta^\star$ Accuracy)	CIFAR10 (89.8%)		Oxford-IIIT Pet (92.2%)		Caltech-101 (94.9%)	
Success Rate	$\Delta_{\mathrm{acc}} \leq 3\%$	Best (Δ_{acc})	$\Delta_{\mathrm{acc}} \leq 3\%$	Best (Δ_{acc})	$\Delta_{\mathrm{acc}} \leq 3\%$	Best (Δ_{acc})
Max. Target Inf.	0.44	0.64 (15.9%)	0.64	0.70 (15.2%)	0.52	0.72 (11.7%)
+ Min. Top-k Inf.	0.60	0.72 (20.0%)	0.74	0.74 (1.6%)	0.68	0.68 (3.6%)
+ Min. Higher-Rank Inf.	0.60	0.80 (6.0%)	0.88	0.88 (2.1%)	0.74	0.77 (8.3%)
+ Multiple Rand Init. (**Ours**)	**0.64**	**0.90** (5.5%)	**0.88**	**0.94** (5.3%)	**0.74**	**0.85** (7.1%)

5 Downstream Application 2: Fairness

Recently, a lot of studies have used influence functions in different ways to achieve fair models [29, 45, 38, 25, 32, 10, 9, 49, 15]. In our paper, we focus on the study by [29] as they use the same definition of influence functions as us. The suggested approach in [29] to achieve a fair model is by *reweighing training data* based on influence scores for a *base model* and then using this reweighed data to train a new downstream model from scratch. This downstream model is expected to have high fairness as a result of the reweighing. For a pictorial representation of this process, see App. Fig. 8. The weights for training data are found by solving an optimization problem which places influence functions in the constraints. See Appendix Sec.A.2 for more details on the optimization problem.

Ultimately, weights of the training data determine the fairness of the downstream model. Since these weights are derived from influence scores, manipulating the influence scores can alter the fairness of the downstream model. As a result, a malicious entity who wants to spread unfairness is incentivized to manipulate influence scores.

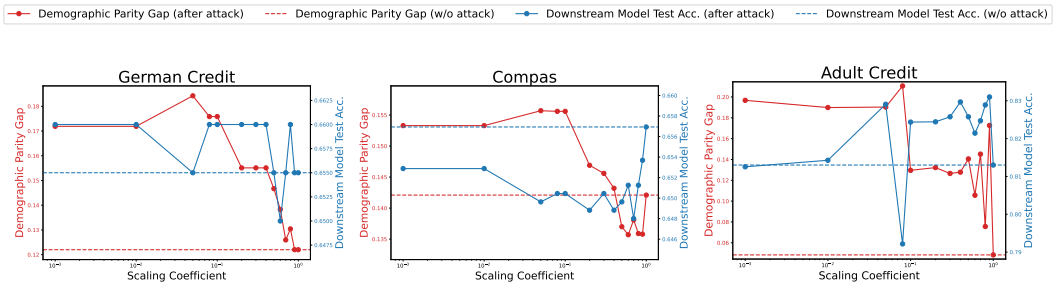

Figure 5: Scaling attack for the Fairness use-case. *Demographic Parity Gap of post-attack downstream models is higher than that of those w/o attack while test accuracies are comparable. This implies that post-attack downstream models are less fair than those w/o attacks.* Scaling coefficients in log scale.

Threat Model. Similar to the general setup, training and test set are fixed, influence calculator is assumed to be the adversary. Model trained by the influence calculator is now the *base model used by the reweighing pipeline.* The adversarial model training can tamper the *base model* parameters from θ^* to θ' to manipulate the influence scores while maintaining similar test accuracy.

Goal of the adversary. Fairness of the final downstream model is measured with a fairness metric. The concrete goal of the adversary is to make the value of this fairness metric worse for the downstream model than what could have been achieved without adversarial manipulation.

Attack. Since the goal of the adversary in this case is not tied to specific target samples, we propose an untargeted attack for the adversary. Our attack is deceptively simple – scale the base model $\theta^\star$ by a constant $\lambda > 0$. The malicious base model output by the model trainer is now $\theta' = \lambda \cdot \theta^\star$, instead of $\theta^\star$. Note that for logistic regression the malicious and original base model are indistinguishable since scaling with a positive constant maintains the sign of the predictions, leading to the same accuracy.

5.1 Fairness Manipulation Experiments

We investigate experimentally if our scaling attack can be successful in reducing fairness of the final downstream model. We use three standard fairness benchmarks : Adult Credit [7], German Credit [20] and Compas [4]. For details about these datasets, see Appendix Table 3. We train L2-regularized logistic regression models for these datasets which act as base models for the training data reweighing pipeline. To carry out our fairness manipulation attack, we scale the models weights with different scaling coefficients and feed these scaled models to the reweighing pipeline. The reweighed training data is then used to train a final downstream model, which is expected to be fair. Parameter values for L2 regularization and the reweighing are same as in [29] and can be found in Appendix Tables 4 and 3.

Evaluation. We evaluate the fairness of the final downstream model with a popular fairness metric, called demographic parity (DP) (See Appendix Def. 2 for definition). A higher value of DP gap implies less fairness and vice-versa. We call the attack a success if the final downstream model post-attack is less fair (higher DP gap) than the final model achieved without attacking. Additionally, we expect the test accuracies of the downstream model with and without attack to be similar to keep the manipulation discreet. Note that this is different from our threat model's indistinguishability constraint, which applies to the base model, not to the downstream model.

Results. As can be seen from our results in Fig.5, the scaling attack works surprisingly well across all datasets – downstream models achieved after our attack are considerably less fair (higher DP gap) than the models without attack, achieving a maximum difference of 16% in the DP gap. Simultaneously, downstream models post-attack maintain similar test accuracies to downstream models without attack. Since the process to achieve the downstream model involves a lot of steps, including solving a non-convex optimization problem to find training data weights and then retraining a model, we sometimes do not see a smooth monotonic trend in fairness metric values w.r.t. scaling coefficients. However, this does not matter much from the attacker's perspective as all the attacker needs is *one* scaling coefficient which meets the attack success criteria.

6 Discussion on Susceptibility and Defense

The susceptibility of influence functions to our attacks can come from the fact that there can exist models that behave very similarly (Rashomon Effect [37]) but have different influential samples up to an extent. Equivalently, changing the influence for many samples does not affect the model accuracy much, as is shown by our experiments (though there exist some samples for which the influence can't be manipulated, from theorem 1). Some plausible ways to defend against the attacks are (1) providing cryptographic proofs of honest model training using Zero-Knowledge Proofs [42, 1, 48] and, (2) to check if the model is atleast a local minima or not, since IFs assume that the model is an optimal solution to the optimization.

7 Related Work

Fragility of Influence Functions. Influence functions proposed in [24] are an approximation to the effect of upweighting a training sample on the loss at a test point. This approximation error can be large as shown by [6, 5, 13], making influence functions fragile especially for deep learning models. Our work is *orthogonal* to this line of work as we study the robustness of influence functions w.r.t. *model parameters* instead of approximation error of influence functions w.r.t. the true influence.

Model Manipulation in the Threat Model. Manipulating models to execute attacks is a prevalent theme in the literature. [41] use model manipulations to corrupt feature attributions in tabular data while [18, 3] do so in vision models. [35] corrupt attention-based explanations for language models while maintaining model accuracy. [40] show that it is possible to corrupt a fairness metric by manipulating an interpretable surrogate of a black-box model while maintaining empirical performance of the surrogate. Similar to these, our threat model also allows the adversary to manipulate models while maintaining the test accuracy. However, our adversarial goal is to corrupt influence-based attributions.

Data Manipulation Attack on Explanations. [14, 2, 50, 11, 22] have studied how data can be manipulated to corrupt feature attributions. On the contrary, firstly, we keep data fixed and manipulate the model and secondly, we work with data attributions rather than feature attributions.

8 Conclusion & Future Work

While past work has mostly focused on feature attributions, in this paper we exhibit realistic incentives to manipulate data attributions. Motivated by the incentives, we propose attacks to manipulate outputs from a popular data attribution tool – Influence Functions. We demonstrate the success of our attacks experimentally on multiclass logistic regression models on ResNet features and standard tabular fairness datasets. Our work lays bare the vulnerablility of influence-based attributions to manipulation and serves as a cautionary tale when using them in adversarial circumstances.

While logistic regression is a good starting point for formulating and solving a new problem and these models are still relevant in many domains, we do think attacking influence functions for large models is an interesting avenue for future research. Some other future directions include exploring different threat models, additional use-cases and manipulating other kinds of data attribution tools.

Acknowledgements

This work was supported by grants NSF CIF-2402817, NSF CNS-1804829, SaTC-2241100, CCF-2217058, ARO-MURI W911NF2110317 and ONR under N00014-24-1-2304.

References

[1] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. *Cryptology ePrint Archive*, 2024.

[2] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. *Advances in neural information processing systems*, 31, 2018.

[3] Christopher Anders, Plamen Pasliev, Ann-Kathrin Dombrowski, Klaus-Robert Müller, and Pan Kessel. Fairwashing explanations with off-manifold detergent. In *International Conference on Machine Learning*, pages 314–323. PMLR, 2020.

[4] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. In *Ethics of data and analytics*, pages 254–264. Auerbach Publications, 2022.

[5] Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger B Grosse. If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*, 35:17953–17967, 2022.

[6] Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020.

[7] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[8] Sebastian Bordt, Michèle Finck, Eric Raidl, and Ulrike von Luxburg. Post-hoc explanations fail to achieve their purpose in adversarial contexts. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 891–905, 2022.

[9] Ruizhe Chen, Jianfei Yang, Huimin Xiong, Jianhong Bai, Tianxiang Hu, Jin Hao, Yang Feng, Joey Tianyi Zhou, Jian Wu, and Zuozhu Liu. Fast model debias with machine unlearning. *Advances in Neural Information Processing Systems*, 36, 2024.

[10] Anshuman Chhabra, Peizhao Li, Prasant Mohapatra, and Hongfu Liu. "what data benefits my classifier?" enhancing model performance and interpretability through influence-based data selection. 2023.

[11] Ann-Kathrin Dombrowski, Maximillian Alber, Christopher Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. Explanations can be manipulated and geometry is to blame. *Advances in neural information processing systems*, 32, 2019.

[12] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 214–226, 2012.

[13] Jacob R Epifano, Ravi P Ramachandran, Aaron J Masino, and Ghulam Rasool. Revisiting the fragility of influence functions. *Neural Networks*, 162:581–588, 2023.

[14] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3681–3688, 2019.

[15] Bishwamittra Ghosh, Debabrota Basu, and Kuldeep S Meel. "how biased are your features?": Computing fairness influence functions with global sensitivity analysis. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pages 138–148, 2023.

[16] Han Guo, Nazneen Fatema Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. Fastif: Scalable influence functions for efficient model interpretation and debugging. *arXiv preprint arXiv:2012.15781*, 2020.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Juyeon Heo, Sunghwan Joo, and Taesup Moon. Fooling neural network interpretations via adversarial model manipulation. *Advances in neural information processing systems*, 32, 2019.

[19] Joshua Hesse, Davide Boldini, and Stephan Sieber. Data valuation: A novel approach for analyzing high throughput screen data using machine learning. 2023.

[20] Hans Hofmann. Statlog (German Credit Data). UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C5NC77.

[21] Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nick Hynes, Nezihe Merve Gürel, Bo Li, Ce Zhang, Dawn Song, and Costas J Spanos. Towards efficient data valuation based on the shapley value. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1167–1176. PMLR, 2019.

[22] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. *Explainable AI: Interpreting, explaining and visualizing deep learning*, pages 267–280, 2019.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

[25] Shuming Kong, Yanyan Shen, and Linpeng Huang. Resolving training biases via influence-based data relabeling. In *International Conference on Learning Representations*, 2021.

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[27] Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. *arXiv preprint arXiv:2310.00902*, 2023.

[28] Fei-Fei Li, Marco Andreeto, Marc'Aurelio Ranzato, and Pietro Perona. Caltech 101, Apr 2022.

[29] Peizhao Li and Hongfu Liu. Achieving fairness at no utility cost via data reweighing with influence. In *International conference on machine learning*, pages 12917–12930. PMLR, 2022.

[30] Xianjia Meng, Yong Yang, Ximeng Liu, and Nan Jiang. Active forgetting via influence estimation for neural networks. *International Journal of Intelligent Systems*, 37(11):9080–9107, 2022.

[31] Xiaoye Miao, Yangyang Wu, Lu Chen, Yunjun Gao, Jun Wang, and Jianwei Yin. Efficient and effective data imputation with influence functions. *Proceedings of the VLDB Endowment*, 15 (3):624–632, 2021.

[32] Jinlong Pang, Jialu Wang, Zhaowei Zhu, Yuanshun Yao, Chen Qian, and Yang Liu. Fair classifiers without fair training: An influence-guided data sampling approach. *arXiv preprint arXiv:2402.12789*, 2024.

[33] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012.

[34] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[35] Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C Lipton. Learning to deceive with attention-based explanations. *arXiv preprint arXiv:1909.07913*, 2019.

[36] Adam Richardson, Aris Filos-Ratsikas, and Boi Faltings. Rewarding high-quality data via influence functions. *arXiv preprint arXiv:1908.11598*, 2019.

[37] Cynthia Rudin, Chudi Zhong, Lesia Semenova, Margo Seltzer, Ronald Parr, Jiachang Liu, Srikar Katta, Jon Donnelly, Harry Chen, and Zachery Boner. Amazing things come from having many good models. *arXiv preprint arXiv:2407.04846*, 2024.

[38] Prasanna Sattigeri, Soumya Ghosh, Inkit Padhi, Pierre Dognin, and Kush R Varshney. Fair infinitesimal jackknife: Mitigating the influence of biased training data points without refitting. *Advances in Neural Information Processing Systems*, 35:35894–35906, 2022.

[39] Andrea Schioppa, Polina Zablotskaia, David Vilar, and Artem Sokolov. Scaling up influence functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8179–8186, 2022.

[40] Ali Shahin Shamsabadi, Mohammad Yaghini, Natalie Dullerud, Sierra Wyllie, Ulrich Aïvodji, Aisha Alaagib, Sébastien Gambs, and Nicolas Papernot. Washing the unwashable: On the (im) possibility of fairwashing detection. *Advances in Neural Information Processing Systems*, 35: 14170–14182, 2022.

[41] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020.

[42] Haochen Sun, Tonghe Bai, Jason Li, and Hongyang Zhang. Zkdl: Efficient zero-knowledge proofs of deep learning training. *Cryptology ePrint Archive*, 2023.

[43] Mukund Sundararajan and Walid Krichene. Inflow, outflow, and reciprocity in machine learning. In *International Conference on Machine Learning*, pages 33195–33208. PMLR, 2023.

[44] Stefano Teso, Andrea Bontempelli, Fausto Giunchiglia, and Andrea Passerini. Interactive label cleaning with example-based explanations. *Advances in Neural Information Processing Systems*, 34:12966–12977, 2021.

[45] Haonan Wang, Ziwei Wu, and Jingrui He. Fairif: Boosting fairness in deep learning via influence functions with validation set sensitive attributes. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 721–730, 2024.

[46] Zifeng Wang, Hong Zhu, Zhenhua Dong, Xiuqiang He, and Shao-Lun Huang. Less is better: Unweighted data subsampling via influence function. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6340–6347, 2020.

[47] Ga Wu, Masoud Hashemi, and Christopher Srinivasa. Puma: Performance unchanged model augmentation for training data removal. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 8675–8682, 2022.

[48] Chhavi Yadav, Amrita Roy Chowdhury, Dan Boneh, and Kamalika Chaudhuri. Fairproof: Confidential and certifiable fairness for neural networks. *arXiv preprint arXiv:2402.12572*, 2024.

[49] Yuanshun Yao and Yang Liu. Understanding unfairness via training concept influence. *arXiv preprint arXiv:2306.17828*, 2023.

[50] Xinyang Zhang, Ningfei Wang, Hua Shen, Shouling Ji, Xiapu Luo, and Ting Wang. Interpretable deep learning under fire. In *29th {USENIX} security symposium ({USENIX} security 20)*, 2020.

A Appendix

A.0.1 Auditing the Influence Calculator by Supplying Test Data

We provide an auditing algorithm that can detect if the influence calculator outputs arbitrary numbers as influence scores.

We first collect a sequence of test points $z_{\text{test_1}}, \cdots, z_{\text{test_d}}$ such that the rank of G_{test} is d, where the ith row in $G_{\text{test}} \in \mathbb{R}^{d \times d}$ is $\nabla_\theta L(z_{\text{test}}, \theta^*)$ and d is the size of model parameters θ^*. This can be done because θ^* is publicly known by the auditor. We then query the influence calculator by feeding $z_{\text{test_i}}$ one by one and collect the returned influence scores as $I \in \mathbb{R}^{m \times n}$. We compute the matrix $C := G_{\text{test}}^{-1} I$. Then we feed a new sequence of test points $z_{\text{test_d+1}}, \cdots, z_{\text{test_2d}}$ and suppose $I_i \in \mathbb{R}^m$ $(i = d+1, \cdots, 2d)$ are the returned influence scores. If there is any i s.t. $I_i \neq \nabla_\theta L(z_{\text{test}}, \theta^*)^\top C$, we return *True*, i.e. state this influence calculator is malicious; otherwise, we return *False*.

We prove that if the influence calculator is honest, this algorithm will return False. In this case, $I = -G_{\text{test}} H_{\theta^*} G_{\text{train}}^\top$. Then the returned score $I_i = -\nabla_\theta L(z_{\text{test}}, \theta^*)^\top H_{\theta^*} G_{\text{train}}^\top = \nabla_\theta L(z_{\text{test}}, \theta^*)^\top G_{\text{test}}^{-1} \left(-G_{\text{test}} H_{\theta^*} G_{\text{train}}^\top\right) = -\nabla_\theta L(z_{\text{test}}, \theta^*)^\top G_{\text{test}}^{-1} I = \nabla_\theta L(z_{\text{test}}, \theta^*)^\top C$ should pass the auditing. In contrast, if the malicious influence calculator returns arbitrary scores, it will be captured by this auditing algorithm.

A.1 Data Manipulation Attack Details

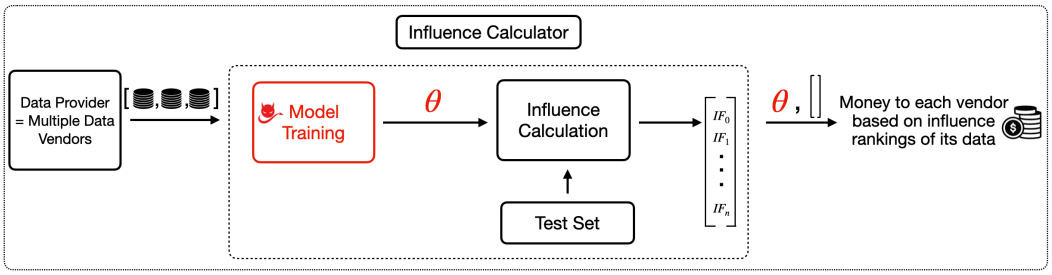

Figure 6: Data Valuation Setup and Threat Model.

A.1.1 Efficient Backward Pass Algorithm

Our idea of rewriting the attack objective involves two essential steps : (1) linearizing the objective (2) making the linearized objective backward-friendly in PyTorch.

Linearize the attack objective: Generally the attack objective can be a non-linear combination of influence functions over different training samples, which makes the backward pass inefficient. Therefore we first transform the given objective into a linear combination of influence functions, $\hat{\ell}_{\text{attack}}((\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z)) := u^\top (\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z)$ for some vector $u \in \mathbb{R}^n$, such that, the objective and gradient values are the same, $\hat{\ell}_{\text{attack}}(\cdot) = \ell_{\text{attack}}(\cdot)$ and $\nabla_\theta \hat{\ell}_{\text{attack}}(\cdot) = \nabla_\theta \ell_{\text{attack}}(\cdot)$. Observe that from chain rule $\nabla_\theta \ell_{\text{attack}}(\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z) = \sum_{z \in Z} \frac{\partial \ell_{\text{attack}}}{\partial \mathcal{I}_\theta(z, Z_{\text{test}})} \cdot \nabla_\theta \mathcal{I}_\theta(z, Z_{\text{test}})$. Therefore, we can set u as $\left(\frac{\partial \ell_{\text{attack}}}{\partial \mathcal{I}_\theta(z, Z_{\text{test}})} : z \in Z\right)$ while meeting the equal objective and gradient value requirement. Influence scores in this vector are computed using an efficient forward pass algorithm, given in Alg. 2.

Get a PyTorch backward-friendly attack objective: Simply expanding our linearized attack objective gives, $\hat{\ell}_{\text{attack}}(\cdot) = v_{\theta,1}^\top H_\theta^{-1} v_{\theta,2}$ where $v_{\theta,1} = \left(-\nabla_\theta \sum_{i=1}^m L\left(z_{\text{test_i}}, \theta\right)\right)$ and $v_{\theta,2} = \left(\nabla_\theta \sum_{z \in Z} u_z \cdot L(z, \theta)\right)$. Gradient computations for this objective will have to go through HIVPs, which is highly inefficient. Therefore we next convert the linearized objective into one which does not involve HIVPs, again such that the objective and gradient values are same as the original objective.

Using chain rule, the gradient of the expanded linearized attack objective can be written as $\nabla_\theta \hat{\ell}_{\text{attack}}(\cdot) = (\nabla_\theta v_{\theta,1})^\top u_2 + u_1^\top (\nabla_\theta v_{\theta,2}) - u_1^\top (\nabla_\theta H_\theta) u_2$ where $u_1 = H_\theta^{-1} v_{\theta,1}$ and $u_2 = H_\theta^{-1} v_{\theta,2}$. PyTorch supports the gradient computation for functions of gradient, making $v_{\theta,1}$ and $v_{\theta,2}$ backward-friendly. PyTorch also calculates gradients for functions of hessian vector products *implicitly*, which leads to efficiency. Additionally, we can precompute u_1, u_2 and freeze them.

As a result, our final backward-friendly objective function is efficient and backward-friendly with PyTorch and is given as, $\bar{\ell}_{\text{attack}}(\theta) = v_{\theta,1}^\top u_2 + u_1^\top v_{\theta,2} - u_1^\top H_\theta u_2$. The algorithm for computing our backward-friendly objective $\bar{\ell}_{\text{attack}}$ is elucidated in Alg. 1.

Derivation for expanding the linearized objective:

$$
\begin{aligned}
\hat{\ell}_{\text{attack}}((\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z)) &= u^\top (\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z) \\
&= \sum_{z \in Z} u_z \cdot \mathcal{I}_\theta(z, Z_{\text{test}}) \\
&= \sum_{z \in Z} u_z \cdot \sum_{i=1}^{m} \mathcal{I}_\theta(z_{\text{test_i}}, z) \\
&= \sum_{z \in Z} u_z \cdot \sum_{i=1}^{m} -\nabla_\theta L\left(z_{\text{test,i}}, \theta\right)^\top H_\theta^{-1} \nabla_\theta L(z, \theta) \\
&= \left(-\nabla_\theta \sum_{i=1}^{m} L\left(z_{\text{test,i}}, \theta\right)\right)^\top H_\theta^{-1} \left(\nabla_\theta \sum_{z \in Z} u_z \cdot L(z, \theta)\right) \\
&= v_{\theta,1}^\top H_\theta^{-1} v_{\theta,2}
\end{aligned}
$$

where $v_{\theta,1} = (-\nabla_\theta \sum_{i=1}^{m} L\left(z_{\text{test,i}}, \theta\right))$ and $v_{\theta,2} = \left(\nabla_\theta \sum_{z \in Z} u_z \cdot L(z, \theta)\right)$.

Chain Rule for gradient of expanded linearized objective:

$$
\begin{aligned}
\nabla_\theta \hat{\ell}_{\text{attack}}((\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z)) &= \nabla_\theta v_{\theta,1}^\top H_\theta^{-1} v_{\theta,2} \\
&= (\nabla_\theta v_{\theta,1})^\top \cdot H_\theta^{-1} v_{\theta,2} + v_{\theta,1}^\top H_\theta^{-1} (\nabla_\theta v_{\theta,2}) - v_{\theta,1}^\top H_\theta^{-1} (\nabla_\theta H_\theta) H_\theta^{-1} v_{\theta,2} \\
&= (\nabla_\theta v_{\theta,1})^\top u_2 + u_1^\top (\nabla_\theta v_{\theta,2}) - u_1^\top (\nabla_\theta H_\theta) u_2
\end{aligned}
$$

where $u_1 = H_\theta^{-1} v_{\theta,1}$ and $u_2 = H_\theta^{-1} v_{\theta,2}$.

Algorithm 1 Get_Backward_Friendly_Attack_Objective

Input: Model Parameters θ, Train Set Z, Test Set Z_{test}, Loss L, Original Attack Objective ℓ_{attack}
Output: Backward-Friendly Attack Objective $\bar{\ell}_{\text{attack}}(\theta)$

1: Compute $(\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z)$ from Appendix Alg. 2
2: Compute $u := \left(\frac{\partial \ell_{\text{attack}}}{\partial \mathcal{I}_\theta(z, Z_{\text{test}})} : z \in Z\right)$
3: Compute $v_{\theta,1} := (-\nabla_\theta \sum_{i=1}^{m} L\left(z_{\text{test_i}}, \theta\right))$ and $v_{\theta,2} := \left(\nabla_\theta \sum_{z \in Z} u_z \cdot L(z, \theta)\right)$
4: Compute and freeze $u_1 := H_\theta^{-1} v_{\theta,1}$ and $u_2 := H_\theta^{-1} v_{\theta,2}$
5: Compute $\bar{\ell}_{\text{attack}}(\theta) := v_{\theta,1}^\top u_1 + u_2^\top v_{\theta,2} - u_1^\top H_\theta u_2$
6: **Return:** $\bar{\ell}_{\text{attack}}(\theta)$

When using the original influence-based objective naively, it was not possible to even do one backward pass due to memory constraints. This algorithm brings down the memory required for one forward + backward pass from not being feasible to run on a 12GB GPU to 7GB for a 206K parameter model and from 8GB to 1.7GB for a 5K model.

Algorithm 2 ForwardOnlyInf

Input: Parameters θ, train set Z, test set Z_{test}, loss L
Output:$(\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z)$

1: Compute $L\left(Z_{\text{test}}, \theta\right) := \nabla_\theta \sum_{i=1}^{m} L\left(z_{\text{test_i}}, \theta\right)$
2: Compute $s_{\text{test}} := H_\theta^{-1} L\left(Z_{\text{test}}, \theta\right)$ by the hessian-inverse-vector product in [24].
3: $\forall z \in Z$, compute $\mathcal{I}_\theta(z, Z_{\text{test}}) := s_{\text{test}}^\top \nabla_\theta L\left(z, \theta\right)$
4: **Return:** $(\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z)$

Algorithm 3 Gradient-based optimization for Attack Loss ℓ_{attack}

Input: Parameters $\theta^\star$, Radius C, train set Z, test set Z_{test}, loss L, attack objective ℓ_{attack}, gradient-based optimizer Opt, distance function dist
Output: θ'

1: Set θ_0 := Randomly chosen model parameters from a ball of radius C centered around $\theta^\star$, $\mathcal{B}(\theta^\star, C)$ where distance is calculated using dist
2: **for** $t = 1$ **to** T **do**
3: $\bar{\ell}_{\text{attack}}(\theta_{t-1}) =$ Get_Backward_Friendly_Attack_Objective(θ_{t-1}, Z, Z_{test}, L, ℓ_{attack})
4: Compute $\nabla_\theta \bar{\ell}_{\text{attack}}(\theta_{t-1})$ through $\bar{\ell}_{\text{attack}}(\theta_{t-1})$.Backward() in PyTorch
5: Update $\theta_{t-1} \to \theta_t$ by the given gradient-based optimizer Opt
6: If $\theta_t \notin \mathcal{B}(\theta^\star, C)$, clip θ_t to lie within $\mathcal{B}(\theta^\star, C)$ where distance is calculated using dist
7: **end for**
8: **Return:** $\theta' := \theta_T$

A.1.2 Experimental details

Dataset Details. CIFAR10 [26] has a training/test set size of 50000/10000 with 10 output classes, Oxford-IIIT Pet [33] has a training/test set size of 3680/3669 with 37 output classes and Caltech-101 [28] has a training/test set size of 6941/1736 with 101 output classes.

Forward pass Details. We use the LiSSA implementation given at `https://github.com/nimarb/pytorch_influence_functions` with recursion depth=1e6, scale=25 and damping factor= 0.01.

Attack Details. To optimize our attack objective, we use algorithm Alg. 3 for computing gradients with Adam as the optimizer [23]. We use two learning rates $\{0.01, 0.1\}$ and 100 steps of updates. We optimize every attack from 5 different initializations. We run our attacks with multiple values of the constraint radius $C = \{0.05, 0.1, 0.2, 0.5\}$. For each regime, the reported number is the highest we could obtain with different values of constants C or α.

Baseline Details. For training a model under the baseline attack we choose Adam as the optimizer, set the batch size as 256 and update for 1400 steps. We ran the baseline for different weights α with a logarithmic scaling from 10 to $1\text{e}18$. We use weighted sampling instead of uniform to account for the weight on the target sample.

A.1.3 Proof of Impossibility Theorem (Theorem 1)

Proof of Theorem 1]:
We first introduce a construction of 2-class classification dataset $Z_{\text{train}}, Z_{\text{test}} \subseteq \mathbb{R}^d \times \{1, -1\}$ and show under this construction and a logistic regression model θ, there exists a target data $z_{\text{target}} \in Z_{\text{train}}$ such that no matter how we manipulate this linear model θ, i.e. $\forall \theta \in \mathbb{R}^d$, the rank of $\mathcal{I}_\theta(z_{\text{target}}, Z_{\text{test}})$ among $(\mathcal{I}_\theta(z, Z_{\text{test}}) : z \in Z_{\text{train}})$ *cannot* reach top-1.

Denote $z_i \in \mathbb{R}^d$ as a one-hot vector which has all zeros except 1 at the ith dimension. We construct Z_{test} as $\{(\mathbf{e}_1, 1)\}$, and construct $Z_{\text{train}} = \{(z_i, y_i) | i = 2, \cdots, d\} \cup \{(\mathbf{e}_1, 1), (-\mathbf{e}_1, 1)\}$ where y_i can be arbitrarily selected from $\{1, -1\}$. By choosing $z_{\text{target}} = (\mathbf{e}_1, 1) \in Z_{\text{train}}$ and $z_{\text{bar}} = (-\mathbf{e}_1, 1) \in Z_{\text{train}}$, next we are going to prove $\mathcal{I}_\theta(z_{\text{target}}, Z_{\text{test}}) < \mathcal{I}_\theta(z_{\text{bar}}, Z_{\text{test}})$ for any $\theta \in \mathbb{R}^d$, which indicates $\mathcal{I}_\theta(z_{\text{target}}, Z_{\text{test}})$ among $(\mathcal{I}_\theta(z_{\text{target}}, Z_{\text{test}}) : z \in Z_{\text{train}})$ *cannot* reach top-1.

We adopt the computation of influence function on logistic regression from the original influence function paper [24]. Denote $r(z_i, \theta) = \sigma(z_i^\top \theta) \cdot \sigma(-z_i^\top \theta)$ where $\sigma(t) = \frac{1}{1+\exp(-t)} > 0$.

Then the Hessian H_θ of the logistic regression training loss $\frac{1}{d+1}\sum_{z\in Z_{\text{train}}} L(z,\theta)$ is $H_\theta = \frac{1}{d+1}\left(2r(\mathbf{e}_1,\theta)\mathbf{e}_1\mathbf{e}_1^\top + \sum_{i=2}^{d} r(z_i,\theta)z_i z_i^\top\right)$. Then we can calculate the influence function for z_{target} and z_{bar}:

$$\mathcal{I}_\theta(z_{\text{target}}, Z_{\text{test}}) = -\sigma(-\theta^\top \mathbf{e}_1)\cdot\sigma(\theta^\top \mathbf{e}_1)\cdot \mathbf{e}_1^\top H_\theta^{-1}\mathbf{e}_1 = -\sigma(-\theta^\top \mathbf{e}_1)\cdot\sigma(\theta^\top \mathbf{e}_1)\cdot\frac{d+1}{r(\mathbf{e}_1,\theta)} < 0,$$

$$\mathcal{I}_\theta(z_{\text{bar}}, Z_{\text{test}}) = -\sigma(-\theta^\top \mathbf{e}_1)\cdot\sigma(\theta^\top \mathbf{e}_1)\mathbf{e}_1^\top H_\theta^{-1}(-\mathbf{e}_1) = \sigma(-\theta^\top \mathbf{e}_1)\cdot\sigma(\theta^\top \mathbf{e}_1)\cdot\frac{d+1}{r(\mathbf{e}_1,\theta)} > 0.$$

This complete the proof: $\forall\theta, \mathcal{I}_\theta(z_{\text{target}}, Z_{\text{test}}) < 0 < \mathcal{I}_\theta(z_{\text{bar}}, Z_{\text{test}})$ and therefore z_{target} can never achieve top-1.

The above construction can be generalized to top-K for any $K > 1$: we can construct Z_{test} as $\{(\mathbf{e}_1, 1)\}$, and construct $Z_{\text{train}} = \{(z_i, y_i) | i = 2, \cdots, d\} \cup \{(\mathbf{e}_1, 1), K \text{ duplicated } (-\mathbf{e}_1, 1)\}$ where y_i can be arbitrarily selected from $\{1, -1\}$. Then similar to the proof above, by choosing $z_{\text{target}} = (\mathbf{e}_1, 1) \in Z_{\text{train}}$ and K duplicated training point in the training set $z_{\text{bar}}^k = (-\mathbf{e}_1, 1), k \in [K]$, we can prove $\forall\theta, \mathcal{I}_\theta(z_{\text{target}}, Z_{\text{test}}) < 0$ and $\forall\theta, k \in [K], \mathcal{I}_\theta(z_{\text{bar}}^k, Z_{\text{test}}) > 0$. Consequently, $\forall\theta$, z_{target} will not achieve top-K.

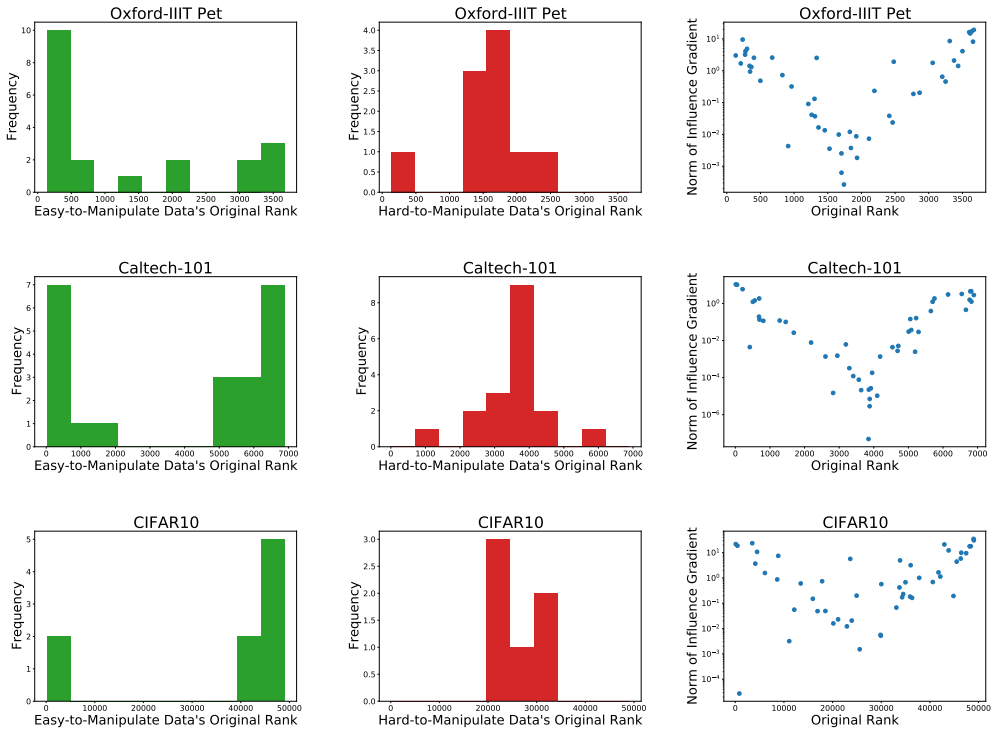

Figure 7: Histograms for original ranks of easy-to-manipulate samples, Histograms for original ranks of hard-to-manipulate samples, Scatterplots for influence gradient norm vs. original ranks of the 50 random target samples. Ranking $k := 1$. *Easy-to-manipulate samples have extreme original influence ranks (large positive or negative) as the samples with the extreme rankings also have higher influence gradient norms, where the gradient is taken w.r.t. model parameters.*

A.2 Fairness Manipulation Attack Details

Optimization Problem for Reweighing Training Data as proposed by Li and Liu [29] is given as follows,

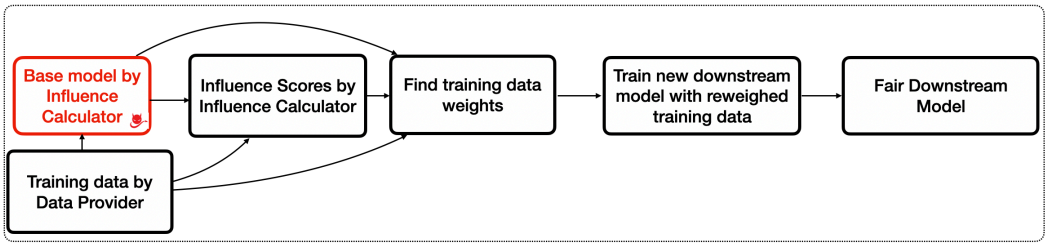

Figure 8: Overview of the process to achieve a fair model, as proposed by [29]. Our adversary, the model trainer manipulates the base model used to calculate influence scores.

$$\begin{array}{ll} \text{minimize} & \sum_i w_i \\ \text{subject to} & \sum_i w_i \mathcal{I}_{\text{fair}}\ (z_i) = -f_{\text{fair}}^{\mathcal{V}} \\ & \sum_i w_i \mathcal{I}_{\text{util}}\ (z_i) \leq 0 \\ & w_i \in [0,1] \end{array} \tag{6}$$

where w_i refers to the weight of the ith training sample, z_i refers to the ith training sample, $\mathcal{I}_{\text{util}}$ refers to our influence function, $\mathcal{I}_{\text{fair}}$ refers to some fairness influence function and $f_{\text{fair}}^{\mathcal{V}}$ corresponds to a differentiable fairness metric. In our threat model, the adversary manipulates the base model, which changes the influence scores $\mathcal{I}_{\text{util}}$.

An advanced version of the above optimization problem using additional parameters (β, γ) which lead to various tradeoffs is given as,

$$\begin{array}{ll} \text{minimize} & \sum_i w_i \\ \text{subject to} & \sum_i w_i \mathcal{I}_{\text{fair}}\ (z_i) \leq -(1-\beta)\ell_{\text{fair}}^{\nu}\,, \\ & \sum_i w_i \mathcal{I}_{\text{util}}\ (z_i) \leq \gamma\left(\min_{\mathbf{v}} \sum_i v_i \mathcal{I}_{\text{util}}\ (z_i)\right), \\ & w_i \in [0,1]. \end{array} \tag{7}$$

Fairness Metric. We define the fairness metric used in our paper, Demographic Parity.

Definition 2 (Demographic Parity Gap (DP) [12]) *Given a data distribution $\mathcal{D}$ over $\bar{\mathcal{X}} \times \{0,1\}$ from which features $x^{\setminus a}$ and sensitive attribute $x^a \in \{0,1\}$ are jointly drawn from, Demographic Parity gap for a model f_θ is defined to be the difference in the rate of positive predictions between the two groups, $|\Pr(\hat{y} \mid x^a = 0) - \Pr(\hat{y} \mid x^a = 1)|$ where $\hat{y}$ is the prediction $f_\theta(x^{\setminus a}, x^a)$.*

Dataset	Predict	Train / Val. / Test Split	#Dim.	Sensitive Attribute	Group Pos. Rate
Adult	if annual income >= 50k	22622/7540/15060	102	Gender - Male / Female	0.312/0.113
Compas	if defendant rearrested in 2 yrs.	3700/1234/1233	433	Race - White / Non-white	0.609/0.518
German Credit	good/bad credit risk	600/200/200	56	Age - 30	0.742/0.643

Table 3: Details for datasets used in our Fairness Manipulation attack experiments. In our setting, the Validation set is the test set shared between the model trainer and influence calculator. The Test set is an untouched set which is used to assess the performance and fairness of the final model achieved after the training with the reweighed training set. All our results are reported on this untouched Test set. Table details borrowed from [29].

Dataset	Model ℓ_2 reg.	DP (β, γ)
Adult	2.26	(0.8,0.3)
Compas	37.00	(0.3,0.1)
German Credit	5.85	(0.5,0.0)

Table 4: Details for the Fairness Manipulation attack experiments. We use the same values of parameters as used by [29].