Understanding Model Predictions via Influence Functions

Yundong Liu [1 *] **Yuze Liu** [2 *] **Zhengyang Qi** [3 *] **Ze Yang** [4 *]

Abstract

In this paper, we reproduced (Koh & Liang, 2017), using influence functions to trace a models prediction back to the training data. We carried out efficient implementations with `Tensorflow`, and demonstrated its performance & applications on real-world datasets.

1. Introduction

Consider a supervised learning problem with random input variable $Z = (X, Y) \in \mathcal{X} \times \mathcal{Y}$, following the joint CDF $F_{X,Y}$. $\{\boldsymbol{z}_i\}_{i=1}^n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ are the realizations of the input variables, i.e. the training points. Assume we are estimating a hypothesis parametrized by $\boldsymbol{\theta} \in \Theta$. Let $\mathcal{L}(\boldsymbol{z}, \boldsymbol{\theta})$[1] be the loss for a point z. The desired parameter $\boldsymbol{\theta}(F_Z) = \text{argmin}_{\boldsymbol{\theta}\in\Theta}\mathbb{E}\left[\mathcal{L}(\boldsymbol{z}, \boldsymbol{\theta})\right]$ minimizes the expected prediction error.

$\boldsymbol{\theta}$, as well as many other $\mathcal{M}$-estimators, falls in to a broader concept of *statistical functional* (Wassermann, 2006), which is a mapping from the space of population cumulative distribution to some field. Given the training set, the empirical cumulative distribution $\widehat{F}_Z$ is usually our best estimate to the true underlying population distribution. $\widehat{\boldsymbol{\theta}} = \boldsymbol{\theta}(\widehat{F}_Z)$, the plug-in estimator of the statistical functional, is obtained by solving empirical risk minimization problem.

$$\begin{aligned}\boldsymbol{\theta}(\widehat{F}_Z) &= \underset{\boldsymbol{\theta}\in\Theta}{\text{argmin}} \int \mathcal{L}(\boldsymbol{z}, \boldsymbol{\theta}) d\widehat{F}_Z \\ &= \underset{\boldsymbol{\theta}\in\Theta}{\text{argmin}} \int \mathcal{L}(\boldsymbol{z}, \boldsymbol{\theta}) \frac{\partial}{\partial \boldsymbol{z}} \left(\frac{1}{n} \sum_{i=1}^n H(\boldsymbol{z} - \boldsymbol{z}_i) \right) d\boldsymbol{z} \\ &= \underset{\boldsymbol{\theta}\in\Theta}{\text{argmin}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\boldsymbol{z}_i, \boldsymbol{\theta}) \end{aligned} \tag{1}$$

Where $H(\boldsymbol{z} - \boldsymbol{z}_i)$ is the Heaviside step function centered at the i^{th} observation $\boldsymbol{z}_i$, and it's distribution derivative is the Dirac delta function centered at $\boldsymbol{z}_i$.

*Equal contribution [1][yundongl@andrew.cmu.edu] [2][yuzel@andrew.cmu.edu] [3][zhengyaq@andrew.cmu.edu] [4][zey@andrew.cmu.edu].

[1]We fold in any regularization terms in to $\mathcal{L}$.

Solving the empirical risk optimization problem is the first phase of supervised learning. The second phase is usually assess quality of the fitted model's prediction power on a validation set. The validation loss on a test point $\boldsymbol{z}_{te}$ (with true label $\boldsymbol{y}_{te}$ and predicted label $f(\boldsymbol{x}_{te}, \boldsymbol{\theta})$) is $\mathcal{L}(\boldsymbol{z}_{te}, \boldsymbol{\theta}(F_Z))$, also a statistical functional.

Our goal is to understand the effect of the training points on a models fitted parameters and its predictions. Now that we have established the concept of empirical risk minimizer and validation loss as plug-in estimators of the statistical functionals $\boldsymbol{\theta}(F_Z)$ and $\mathcal{L}(\boldsymbol{z}_{te}, \boldsymbol{\theta}(F_Z))$, respectively; it is natural to ask: How to measure the "sensitivity" of the plug-in estimators with respect to the empirical distribution? In the language of machine learning, we are asking how much would the fitted parameters and the validation losses change, if we perturb the training data by a little bit. This idea can be mathematically formalized by the *influence function*.

1.1. Measuring the Influence to Parameters

The Gâteaux Derivative generalizes the idea of directional derivative to functions between locally convex topological vector spaces such as Banach spaces. The Gâteaux derivative of a statistical functional $\boldsymbol{\theta}(F_Z)$ at F_Z^* in the direction G is defined as:

$$\begin{aligned}\mathcal{I}_{\boldsymbol{\theta}}(G) &:= \nabla_G \boldsymbol{\theta}(F_Z)\Big|_{F_Z = F_Z^*} \\ &= \lim_{h\to 0} \frac{\boldsymbol{\theta}((1-h)F_Z^* + hG) - \boldsymbol{\theta}(F_Z^*)}{h} \end{aligned} \tag{2}$$

In particular, if F_Z^* is the true population distribution, and the direction G is the Heaviside step function $H(\boldsymbol{z} - \boldsymbol{z}^*)$ centered at $\boldsymbol{z}^*$, the Gâteaux derivative is called the *Influence Function*. It is essentially the sensitivity of the statistical functional $\boldsymbol{\theta}(F_Z)$ with respect to adding a little probability mass at $\boldsymbol{z} = \boldsymbol{z}^*$. By convention, the Heaviside function in this context is usually denoted as $\delta_{\boldsymbol{z}^*}$, which should not be confused with the Dirac delta function. By this definition, we can write down the Gâteaux derivative of the empirical risk minimizer with respect to upweighting a training point $\boldsymbol{z}_{tr}$:

$$\mathcal{I}_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{z}_{tr}) = \lim_{h\to 0} \frac{\boldsymbol{\theta}((1-h)\widehat{F}_Z + h\delta_{\boldsymbol{z}_{tr}}) - \boldsymbol{\theta}(\widehat{F}_Z)}{h} \quad (3)$$

which is usually called the empirical influence function. It is worth mentioning that if we take $h = -\frac{1}{n}$, then the contaminated distribution in the numerator, $(1+\frac{1}{n})\widehat{F}_Z - \frac{1}{n}H(\boldsymbol{z}-\boldsymbol{z}_{tr})$ is actually the empirical distribution after *removing* $\boldsymbol{z}_{tr}$ from the training set[2].

1.2. Measuring the Influence to Testing Loss

The fitted model parameters is a bridge that connects the training points and the prediction on unseen testing points. Ultimately we care about the training points' effect on the prediction. For this purpose we introduce one more influence function to measure the sensitivity of the loss of a given *validation point* $\boldsymbol{z}_{te}$ with respect to adding a little probability mass at a particular training point $\boldsymbol{z}_{tr}$.

$$\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te}) = \lim_{h\to 0} \frac{\mathcal{L}(\boldsymbol{z}_{te}, \boldsymbol{\theta}(\widehat{F}_{-\boldsymbol{z}_{tr}})) - \mathcal{L}(\boldsymbol{z}_{te}, \boldsymbol{\theta}(\widehat{F}_Z))}{h}$$
$$F_{-\boldsymbol{z}_{tr}} = (1-h)\widehat{F}_Z + h\delta_{\boldsymbol{z}_{tr}} \quad (4)$$

2. Approach

The naive approach to compute $\mathcal{I}_{\hat{\theta}}$ and $\mathcal{I}_{\mathcal{L}}$ is very straightforward. Recall that $\widehat{\boldsymbol{\theta}}_{-i} := \boldsymbol{\theta}((1+\frac{1}{n})\widehat{F}_Z - \frac{1}{n}\delta_{\boldsymbol{z}_i})$ is the model parameter estimated from the leave-one-out training set; (3) can be reformulated as $\mathcal{I}_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{z}_i) = \lim_{n\to\infty} n(\widehat{\boldsymbol{\theta}}_{-i} - \widehat{\boldsymbol{\theta}})$. This approach, however, requires n-runs of leave-one-out retraing if we want to evaluate the influence for all training points, which is prohibitively slow in most cases. Fortunately, under certain assumptions we can approximate $\widehat{\boldsymbol{\theta}}_{-i} - \widehat{\boldsymbol{\theta}}$ very well without explicitly solving another empirical risk optimization problem for $\widehat{\boldsymbol{\theta}}_{-i}$.

To make that possible we assume for now that the empirical risk is twice differentiable, and strictly convex in $\boldsymbol{\theta}$. That implies the hessian of the empirical risk is positive definite. We will relax these assumptions in subsequent sections.

Consider the empirical risk optimization problem for the contaminated distribution which *downweights* the training point $\boldsymbol{z}_i$:

$$\widehat{\boldsymbol{\theta}}_{-h,\boldsymbol{z}_i} = \boldsymbol{\theta}((1+h)\widehat{F}_Z - h\delta_{\boldsymbol{z}_i}) \quad (5)$$
$$= \operatorname*{argmin}_{\theta\in\Theta} \frac{1+h}{n}\sum_{j=1}^{n}\mathcal{L}(\boldsymbol{z}_j, \boldsymbol{\theta}) - h\mathcal{L}(\boldsymbol{z}_i, \theta)$$

Since the empirical risk $R(\boldsymbol{\theta}) := \frac{1}{n}\sum_{j=1}^{n}\mathcal{L}(\boldsymbol{z}_j, \boldsymbol{\theta})$ is strictly convex, the optimality condition is:

$$(1+h)\nabla_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}}_{-h,\boldsymbol{z}_i}) - h\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_i, \widehat{\boldsymbol{\theta}}_{-h,\boldsymbol{z}_i}) = 0 \quad (6)$$

[2]It evaluates to $(1+\frac{1}{n})\widehat{F}_{-i} + o(\frac{1}{n})$, we can drop the higher order term.

We assume the perturbation is small in the sense that $h \to 0$, so the difference in parameter estimate is also small: $\widehat{\boldsymbol{\theta}}_{-h,\boldsymbol{z}_i} - \widehat{\boldsymbol{\theta}} = \boldsymbol{\Delta}_{\boldsymbol{\theta}} \to \mathbf{0}$. We perform a Taylor expansion of the left-hand side of (6) at $\widehat{\boldsymbol{\theta}}$:

$$(1+h)\nabla_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}}) - h\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_i, \widehat{\boldsymbol{\theta}}) + \quad (7)$$
$$\left[(1+h)\nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}}) - h\nabla^2_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_i, \widehat{\boldsymbol{\theta}})\right]\boldsymbol{\Delta}_{\boldsymbol{\theta}} + o(\|\boldsymbol{\Delta}_{\boldsymbol{\theta}}\|) = 0$$

Drop the $o(\|\boldsymbol{\Delta}_{\boldsymbol{\theta}}\|)$ terms, and use the fact that $\nabla_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}}) = \mathbf{0}$:

$$\boldsymbol{\Delta}_{\boldsymbol{\theta}} \approx -\left[(1+h)\nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}}) - h\nabla^2_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_i, \widehat{\boldsymbol{\theta}})\right]^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_i, \widehat{\boldsymbol{\theta}})h$$
$$\approx -\nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_i, \widehat{\boldsymbol{\theta}})h \quad (8)$$

If n is large enough, we take $h = -\frac{1}{n} \to 0$, and thus

$$\mathcal{I}_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{z}_i) = \lim_{n\to\infty} \frac{\boldsymbol{\theta}((1+1/n)\widehat{F}_Z - \delta_{\boldsymbol{z}_i}/n) - \boldsymbol{\theta}(\widehat{F}_Z)}{-1/n}$$
$$= \lim_{n\to\infty} -n(\widehat{\boldsymbol{\theta}}_{-\frac{1}{n},\boldsymbol{z}_i} - \widehat{\boldsymbol{\theta}}) = \lim_{n\to\infty} -n\boldsymbol{\Delta}_{\boldsymbol{\theta}}$$
$$\approx -\nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_i, \widehat{\boldsymbol{\theta}}) \quad (9)$$

A classic theory (Cook & Weisberg, 1982) shows that this approximation equals a single step of Newton's method using $\widehat{\boldsymbol{\theta}}$ as the starting value to minimize the quadratic approximation to the leave-one-out empirical risk.

Next, we can apply the chain rule to derive the closed-form expression of the influence of downweighting a training sample on a function of $\widehat{\boldsymbol{\theta}}$. The function we care about is the validation loss $\mathcal{L}(\boldsymbol{z}_{te}, \boldsymbol{\theta}(\widehat{F}_Z))$. We have:

$$\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te}) = \nabla_{\delta_{\boldsymbol{z}_{tr}}}\mathcal{L}(\boldsymbol{z}_{te}, \boldsymbol{\theta}(F_Z))\Big|_{F_Z=\widehat{F}_Z} \quad (10)$$
$$= \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}})^\top \lim_{h\to 0}\frac{\boldsymbol{\theta}(\widehat{F}_{-\boldsymbol{z}_{tr}}) - \boldsymbol{\theta}(\widehat{F}_Z)}{h}$$
$$\approx -\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}})^\top \nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_{tr}, \widehat{\boldsymbol{\theta}})$$

3. Implementations

For n observations and $\boldsymbol{\theta} \in \mathbb{R}^p$, the exact calculation using (10) enables us to get the influence of all training points on validation loss without retraining the model. For a particular testing point $\boldsymbol{z}_{te}$, the quantity $\boldsymbol{p}_{te} = \nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}})$ is the same for all training points, so it need to be evaluated only once. Then one can compuate the inner product $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_i, \boldsymbol{z}_{te}) = -\boldsymbol{p}_{te}^\top\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{z}_i, \widehat{\boldsymbol{\theta}})$ for all training points $\{\boldsymbol{z}_i\}_{i=1}^n$, which only costs $O(np)$.

The real bottleneck of this exact method is computing $\boldsymbol{p}_{te}$. It involves forming the $p \times p$ Hessian matrix for each of the n terms in the empirical risk, and inverting this matrix. This requires $O(np^2 + p^3)$ operations -

very expensive for high-dimensional problems with a large number of parameters.

The idea to overcome this bottleneck is to avoid explicitly forming and inverting the hessian $\widehat{\boldsymbol{H}} := \nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})$. Fortunately, the only thing we really need is $\boldsymbol{p}_{te}$, which has the form of $\widehat{\boldsymbol{H}}\boldsymbol{v}$ - the product of the $p \times p$ hessian and a $p \times 1$ vector. (Pearlmutter, 1994) showed that the Hessian-vector product (HVP) equals the directional derivative of the gradient in the direction of $\boldsymbol{v}$,

$$\widehat{\boldsymbol{H}}\boldsymbol{v} = \lim_{r \to 0} \frac{\nabla_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}} + r\boldsymbol{v}) - \nabla_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})}{r} \tag{11}$$

This method can be implemented in auto-gradients systems, or using finite-difference approximation, such that computing $\widehat{\boldsymbol{H}}\boldsymbol{v}$ takes $O(np)$ time for arbitrary $\boldsymbol{v}$. Two techniques are discussed in (Koh & Liang, 2017) to compute $\boldsymbol{p}_{te}$ by only evaluating HVPs, and we implemented both of them.

3.1. Conjugate Gradients

Under our assumptions, $\boldsymbol{p}_{te}$ is the solution to the positive-definite linear system $\nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})\boldsymbol{p}_{te} = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}})$. There is a very effective iterative algorithm for solving such linear systems with a symmetric positive-definite matrix, and only requires the evaluation of matrix-vector products: the linear conjugate gradients method (CG). Its implementation details and empirical performance on large datasets are well-studied in (Martens, 2010).

3.2. Stochastic Taylor Approximation

If the Jordan normal form of $(\boldsymbol{I} - \widehat{\boldsymbol{H}})$ is $\boldsymbol{J}$, there exists an invertible matrix $\boldsymbol{P}$ such that $(\boldsymbol{I} - \widehat{\boldsymbol{H}}) = \boldsymbol{P}\boldsymbol{J}\boldsymbol{P}^{-1}$; this yields $(\boldsymbol{I} - \widehat{\boldsymbol{H}})^m = \boldsymbol{P}\boldsymbol{J}^m\boldsymbol{P}^{-1} \to 0$ as $m \to \infty$ if and only if $|\lambda| < 1$ for all the eigenvalues of $(\boldsymbol{I} - \widehat{\boldsymbol{H}})$. We assume this to be true, then power series approximation can be applied: $(\boldsymbol{I} - (\boldsymbol{I} - \widehat{\boldsymbol{H}}))^{-1} = \widehat{\boldsymbol{H}}^{-1} = \sum_{i=0}^{\infty} (\boldsymbol{I} - \widehat{\boldsymbol{H}})^i$. Let the partial sum be $\widehat{\boldsymbol{H}}_j^{-1} = \sum_{i=0}^{j} (\boldsymbol{I} - \widehat{\boldsymbol{H}})^i$, we have the following recursive formula for the inverse hessian vector product $\boldsymbol{p}_m := \widehat{\boldsymbol{H}}_m^{-1}\boldsymbol{v}$:

$$\begin{aligned} \widehat{\boldsymbol{H}}_j^{-1}\boldsymbol{v} &= \left[\boldsymbol{I} + (\boldsymbol{I} - \widehat{\boldsymbol{H}})\widehat{\boldsymbol{H}}_{j-1}^{-1}\right]\boldsymbol{v} \\ \boldsymbol{p}_j &= \boldsymbol{v} + \boldsymbol{p}_{j-1} - \widehat{\boldsymbol{H}}\boldsymbol{p}_{j-1} \end{aligned} \tag{12}$$

So that $\boldsymbol{p}_m = \widehat{\boldsymbol{H}}_m^{-1}\boldsymbol{v} \to \widehat{\boldsymbol{H}}^{-1}\boldsymbol{v}$. (Agarwal et al., 2017) showed that one can replace the full hessian $\widehat{\boldsymbol{H}}$ in (12) with the hessian of a single element of loss $\nabla^2_{\theta} \mathcal{L}(\boldsymbol{z}_j, \widehat{\boldsymbol{\theta}})$, and $\boldsymbol{z}_j$ uniformed random sampled from the whole training set for each iteration, and still retains a relatively high accuracy. By doing this, one only need to evaluate the HVP: $\nabla^2_{\theta} \mathcal{L}(\boldsymbol{z}_j, \widehat{\boldsymbol{\theta}})\boldsymbol{p}_{j-1}$ for each iteration, which costs $O(p)$. The following pseudocode describes how to implement this procedure in a `Tensorflow`-like auto-gradients system.

Algorithm 1
Linear time Stochastic 2^{nd} Order Algorithm (LiSSA)

Inputs: (data $\{\boldsymbol{z}_i\}$, vector $\boldsymbol{v}$, integer m,
function $\texttt{hvp}(\boldsymbol{z}, \boldsymbol{v})$ to evaluate $\nabla^2_{\theta} \mathcal{L}(\boldsymbol{z}, \widehat{\boldsymbol{\theta}})\boldsymbol{v}$)
Initialize $\texttt{curEstimate} \leftarrow \boldsymbol{v}$
for $i = 0$ **to** m **do**
  $\texttt{z} \leftarrow$ Draw 1 point from $\{\boldsymbol{z}_i\}_{i=1}^n$
  $\texttt{h} \leftarrow \texttt{hvp}(\texttt{z}, \texttt{curEstimate})$
  Scale $\texttt{h}$ such that the power series converges.
  $\texttt{curEstimate} \leftarrow \boldsymbol{v} + \texttt{curEstimate} + \texttt{h}$
end for

The algorithm above takes $O(mp)$ time to run. We can repeat it for s times and take the average to further reduce the variance - which takes $O(smp)$ in total. Therefore, evaluating $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ takes $O(np + smp)$ time. Our experiments in next section show empirically that on a large dataset, using $sm = O(n)$ gives good accuracy.

4. Experiments and Extensions

Recall (9), $\mathcal{I}_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{z}_i) = \lim_{n \to \infty} -n(\widehat{\boldsymbol{\theta}}_{-\frac{1}{n}, \boldsymbol{z}_i} - \widehat{\boldsymbol{\theta}})$ implies that the influence function is an asymptotic approximation of the quantity $-n(\widehat{\boldsymbol{\theta}}_{-\frac{1}{n}, \boldsymbol{z}_i} - \widehat{\boldsymbol{\theta}})$, the difference between leave-one-out refitted parameter and the full empirical risk minimizer, scaled by the sample size. Same argument holds for $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$. In this section, we show with experiments and different models that the influence matches the LOO retraining difference under the assumptions that $R(\widehat{\boldsymbol{\theta}})$ is strictly convex and twice differentiable. Moreover, they remain highly correlated even if the assumptions are violated.

4.1. Ridge Regression

We can derive a closed-form expression of $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ for the ridge regression empirical risk:

$$R_{\text{ridge}}(\widehat{\boldsymbol{\theta}}) = \frac{1}{n} \sum_{i=1}^{n} \left((y_i - \boldsymbol{x}_i^\top \widehat{\boldsymbol{\theta}})^2 + \frac{\lambda}{n} \|\widehat{\boldsymbol{\theta}}\|_2^2 \right) \tag{13}$$

Note that we fold the regularization term within the loss function, and scale it with n so that the regularization in total loss is $\lambda \|\widehat{\boldsymbol{\theta}}\|_2^2$. For a certain validation point $\boldsymbol{z}_{te}$, the loss is $\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}}) = (y_{te} - \boldsymbol{x}_{te}^\top \widehat{\boldsymbol{\theta}})^2 + \frac{\lambda}{n} \|\widehat{\boldsymbol{\theta}}\|_2^2$. Therefore, the influence of a training point $\boldsymbol{z}_{tr}$ on that validation loss is:

$$\begin{aligned} \mathcal{I}_{\mathcal{L}_{\text{ridge}}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te}) = \frac{n}{2} \nabla_{\boldsymbol{\theta}}^\top \mathcal{L}(\boldsymbol{z}_{tr}, \widehat{\boldsymbol{\theta}}) \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_p \right)^{-1} \cdot \\ \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}}) \\ \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{z}, \widehat{\boldsymbol{\theta}}) = -2\boldsymbol{x}^\top (y - \boldsymbol{x}^\top \widehat{\boldsymbol{\theta}}) + \frac{2\lambda}{n} \widehat{\boldsymbol{\theta}} \end{aligned} \tag{14}$$

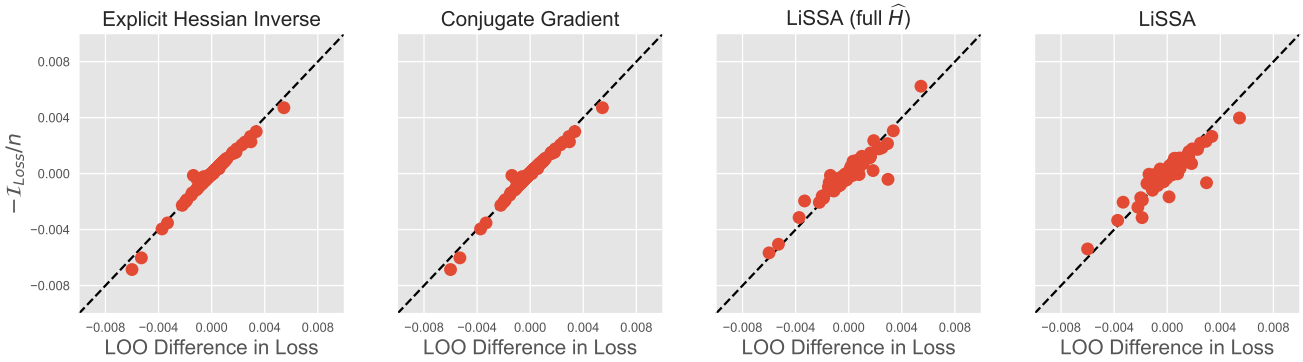

Figure 1. **Influence function $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ matches leave-one-out difference.** We randomly picked a validation point $\boldsymbol{z}_{te}$ and calculated the influence on validation loss for all training points $\boldsymbol{z}_i$. In this illustration $\boldsymbol{z}_{te}$ is the 42$^{\text{rd}}$ instance in the dataset, and $\mathcal{I}_{\mathcal{L}}$ was evaluated for the rest 516 points. **(Panel-1):** We plotted the LOO difference $\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}}_{-\frac{1}{n},\boldsymbol{z}_i}) - \mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}})$ against $-\frac{1}{n}\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_i, \boldsymbol{z}_{te})$ for each $\boldsymbol{z}_i$. The influence is calucated by explicitly forming and inverting hessian matrix $\nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})$. **(Panel-2):** The influence is evaluated by solving the linear system with conjugate gradients, only hessian-vector products were evaluated. **(Panel-3):** The influence was evaluated with the original Taylor approximation. The recursion step is *exactly* the same as (12), i.e. the HVPs were evaluated with the *full* empirical risk $R(\widehat{\boldsymbol{\theta}})$. We set the recursion depth to be $m = 2000$, and we divided the HVPs by a scaling constant $C = 400$ to ensure the convergence of the power series. **(Panel-4):** The influence was evaluated with the LiSSA algorithm: for each recursion step, the algorithm only look at one training point $\boldsymbol{z}_j$, and the HVP was evaluated with that *single* element of loss $\mathcal{L}(\boldsymbol{z}_j, \widehat{\boldsymbol{\theta}})$. We used the same recursion depth and scaling constant.

We fit a ridge regression with $\lambda = 1.0$ on the `ForestFires` dataset, a regression task with 517 instances and 13 features. The aim is to predict the burned area of forest fires. We randomly draw a validation point and use the rest instances to fit the model.

After training the model, we computed $\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}}_{-\frac{1}{n},\boldsymbol{z}_i}) - \mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}})$, the difference between the validation loss after doing actural LOO retaining and the original loss with $\widehat{\boldsymbol{\theta}}$, for all training points. Then, we plot the actual LOO difference against its asymptotic approximation, $-\frac{1}{n}\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$, to assess the accuracy of our influence functions with all three methods that we discussed: explicit hessian inverse, conjugate gradients, and stochastic Taylor approximation (LiSSA). The results are presented in Figure 1.

The influence and the leave-one-out difference matches chosely. For explicit hessian inverse and CG, there was an almost exact match. For Taylor approximation methods, there was a little variance; furthermore, we didn't lose much accuracy when switching from full HVP to LiSSA algorithm, but increased the speed by n times.

4.2. Logistic Regression

Next we switch to classification tasks. A closed-form expression of $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ is also easy to derive for the logistic regression empirical risk. Let $\sigma(t) = 1/(1 + e^{-t})$, label $y \in \{0, 1\}$, we have

$$R_{\text{logit}}(\widehat{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{i=1}^{n}\left(-y_i\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_i + \log(1 + e^{\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_i})\right) \quad (15)$$

For a certain validation point, the loss is $\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}}) = -y\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{te} + \log(1 + e^{\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{te}})$. Hence the influence of a training point $\boldsymbol{z}_{tr}$ on that validation loss is:

$$\mathcal{I}_{\mathcal{L}_{logit}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te}) = -\boldsymbol{x}_{tr}^\top \nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})^{-1} \boldsymbol{x}_{te} \cdot (\sigma(\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{tr}) - y_{tr})(\sigma(\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{te}) - y_{te}) \quad (16)$$

$$\nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{i=1}^{n} \sigma(\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_i)\sigma(-\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_i)\boldsymbol{x}_i \boldsymbol{x}_i^\top$$

If there is an L_2-regularization, we just need to fold the $\frac{\lambda}{n}\|\widehat{\boldsymbol{\theta}}\|_2^2$ into the loss function like we did for the Ridge regression.

(Koh & Liang, 2017) discussed an alternative formulation of logistic loss. Instead of using 0-1 labels, consider $y \in \{1, -1\}$. The loss function under this ± 1 formulation is the loss is $\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}}) = \log(1 + e^{-y_{te}\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{te}})$. The influence of a training point on that ± 1 validation loss is:

$$\mathcal{I}'_{\mathcal{L}_{logit}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te}) = -\boldsymbol{x}_{tr}^\top \nabla^2_{\boldsymbol{\theta}} R(\widehat{\boldsymbol{\theta}})^{-1} \boldsymbol{x}_{te} \cdot y_{tr}\sigma(-y_{tr}\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{tr}) \cdot y_{te}\sigma(-y_{te}\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{te}) \quad (17)$$

One can examine that (16) and (17) are equivalent.

We fit an L_2-regularized logistic regression model with $\lambda = 1000$ on the `MNIST` dataset. For simplicity we

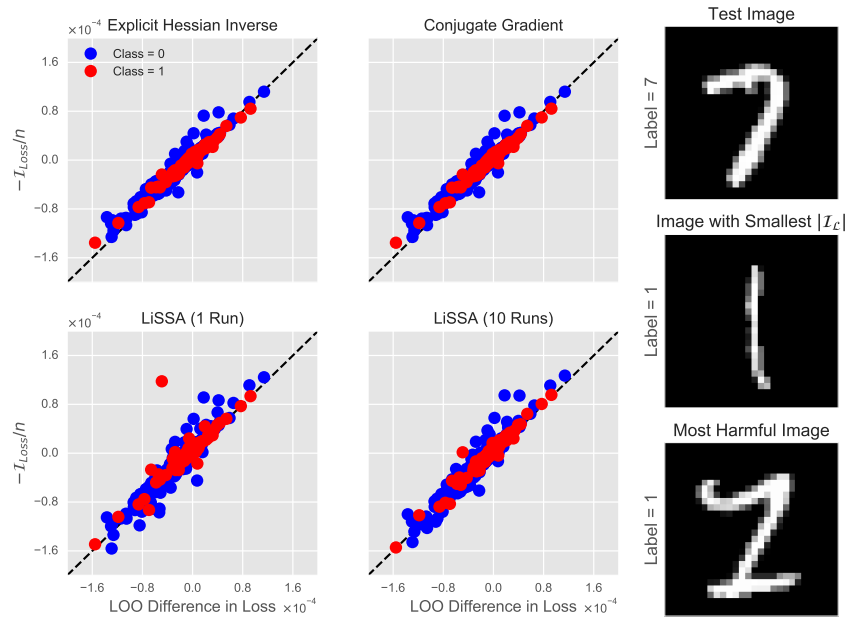

Figure 2. **Influence for handwritten digits classification.** In this illustration $\boldsymbol{z}_{te}$ is the 7^{th} instance in the dataset. The red dots stand for positive points, blue for negative points. **(Left):** We plotted the LOO difference $\mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}}_{-\frac{1}{n},\boldsymbol{z}_i}) - \mathcal{L}(\boldsymbol{z}_{te}, \widehat{\boldsymbol{\theta}})$ against $-\frac{1}{n}\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_i, \boldsymbol{z}_{te})$ for all the three methods. The explicit hessian inverse and the CG method are the same as we used in the Ridge experiment. For the LiSSA method we sampled 1 training point in each recursion step, and we set the recursion depth as $m = 10000$; we divided the HVPs by a scaling constant $C = 10^3$. An outlier can be observed in the Southwest panel where we ran the LiSSA algorithm only once. In the Southeast panel, we ran the algorithm 10 times, and took average to obtain the final estimate. The variance is reduced and the outlier has gone. **(Right):** The upper panel shows the validation point. The middle panel is the training point with the smallest influence, i.e. the smallest $|\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_i, \boldsymbol{z}_{te})|$. It looks like an "easy" positive point. The bottom panel is the training point with the largest negative influence, which looks like a "hard" positive point. It can be thought as the most harmful to predicting $\boldsymbol{z}_{te}$ in the sense that removing this point will cause reduce the prediction loss.

only include digits 1 and 7, and we let the label $y = 1$ stand for digit 1 and $y = 0$ for digit 7. The resulting binary MNIST dataset has 9084 instances; each instance is a 28×28 image, represented by grayscale matrix. We first standardize the data before training the model, it turns out that standardizing helps control the numerical scale of losses, and stabilizes the influence calculation. Then, we randomly choose a testing point and calculate the influence of the rest training points; we also compute the actual LOO difference using `sklean`'s LBFGS solver. The results are presented in Figure 2. Note that there are points from both classes that have positive or negaive influences, indicating that $\text{sgn}(\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_i, \boldsymbol{z}_{te}))$ is *not* determined by whether $y_{te} = y_{tr}$ or not.

On `MNIST` dataset, we performed another analysis on the closed-form expression (17): we examined the effect of the $\sigma(-y_{tr}\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{tr})$ factor and the hessian matrix on the influence. The analysis is shown in Figure 3. In particular, we replicated the approach in the original paper by plotting $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ calculated by (17) against its variants that misses certain factor(s).

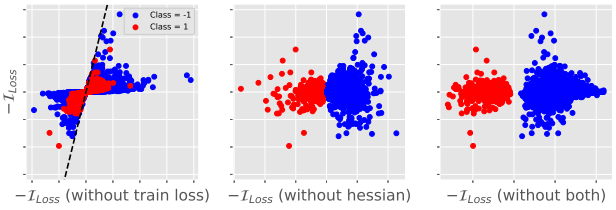

Figure 3. **Components of influence.** The correct influence is plotted on y-axis, while its variants that misses certain factor(s) are plotted on x-axis. Blue dots are the training points with *same label* as $\boldsymbol{z}_{te}$; red dots are the opposites. **(Left):** The black-dashed line has slope = 1; one can observe that many points lie to the right of the $45°$ line in the first quadrant, suggesting that we overestimate the influence of these training points if the factor $\sigma(-y_{tr}\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{tr})$ was excluded. **(Mid):** Without the hessian inverse, $\text{sgn}(\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_i, \boldsymbol{z}_{te})) = \mathbf{1}(y_{te} = y_{tr})$. This is due to the fact that we used the unscaled data, where all features are positive grayscales. Clearly, this is incorrect since we have illustrated in Fugure 2 that points from both classes can be helpful or harmful. **(Right):** Without both factors, the influence degenerates into $\boldsymbol{x}_{tr}^\top \boldsymbol{x}_{te} \cdot [y_{tr} y_{te} \sigma(-y_{te}\widehat{\boldsymbol{\theta}}^\top \boldsymbol{x}_{te})] = \boldsymbol{x}_{tr}^\top \boldsymbol{x}_{te} c$, which is just the inner product scaled by a constant.

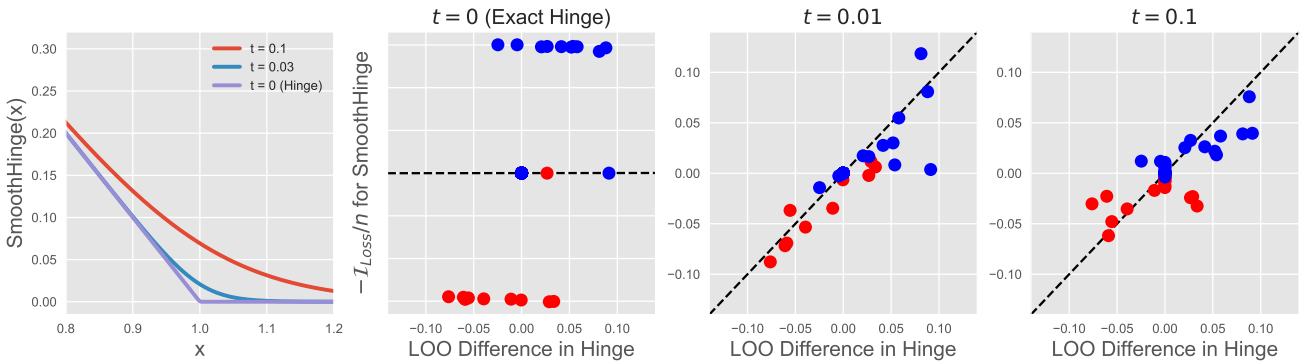

Figure 4. **Smooth approximation to the non-differentiable loss.** In this illustration, the red dots stand for $y = 1$ points, blue for $y = -1$ points. The influence calculated with *smoothed* hinge loss is plotted on y-axis, while the LOO difference in exact hinge loss ($t = 0$) is plotted on x-axis. The black-dashed line has slope 1. **(Panel-1):** By varying t, we can approximate the hinge loss with arbitrary accuracy. **(Panel-2):** We force the system to compute $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ for the $t = 0$ non-differentiable case. The influence function is not a good estimate of LOO difference, overestimating it disastrously. The is due to the fact that $\nabla^2 R_{\text{sv}}(\widehat{\boldsymbol{\theta}})$ is close to zero in this setting, providing no information to the quadratic approximation of $\mathcal{L}(\boldsymbol{z}, \widehat{\boldsymbol{\theta}})$. As a result, the quadratic approximation is linear, leading to overestimating the true influence. **(Panel-3):** The influence is computed for the smoothed empirical risk with $t = 0.01$. The correlation coefficient between $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ and hinge LOO difference is 0.8602. **(Panel-4):** $t = 0.1$, correlation is 0.7700. With t too large, the smoothed loss is no longer a good approximation of the original hinge loss, so the correlation decreases.

4.3. Support Vector Machine

Until now, we have assumed that the loss function is twice differentiable. However, non-differentiable losses, such as hinge loss and L_1 regularization, are common in statistical learning. The idea to make things work is to construct a smooth function that converges to the non-differentiable components of the loss function. Then, we minimize the smoothed version of the empirical risk, and compute $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$. In this section, we examplify this general idea with the linear support vector classifier. Besides, we show that the influence computed for the smoothed linear SVC is still a good approximation to the LOO difference.

For a binary classification problem with label $y \in \{1, -1\}$, the linear SVC hypothesis is parametrized by weights and intercept: $\boldsymbol{\theta} = (\boldsymbol{\beta}, \beta_0)$, and the empirical risk is, by (Hastie et al., 2001):

$$R_{\text{sv}}(\widehat{\boldsymbol{\theta}}) = \frac{1}{n} \sum_{i=1}^{n} \left[h\left(y_i (\boldsymbol{x}_i^\top \widehat{\boldsymbol{\beta}} + \widehat{\beta}_0) \right) + \frac{\lambda}{2n} \|\boldsymbol{\beta}\|_2^2 \right] \quad (18)$$

Where $h(x) = \max\{0, 1 - x\}$ is the hinge loss. (Koh & Liang, 2017) constructed a smooth approximation of h. Let $t > 0$ be a constant hyperparameter, define $h_t(x) := t \log(1 + \exp(\frac{1-x}{t}))$, one can show that h_t converges to h as $t \to 0$. (See Figure 4 Panel-1). Therefore, we replace the hinge function in (18) with h_t, and examine the influence induced by the hessian matrix of this smoothed empirical risk.

We fit a smoothed linear SVC with $\lambda = 2.0$ on the classic `Iris` dataset. For simplicity we only include two classes: $y = 1$ for *Versicolor*, and $y = -1$ for *Virginica*. The resulting binary Iris dataset has 100 instances and 4 features. We start by randomly choosing 10 testing points, fit the smoothed linear SVC, and calculate the influence of the rest training points with (18). Then, we compute the actual LOO difference using the original hinge loss ($t = 0$). The results are presented in Figure 4.

5. Applications

The influence function can be applied in many real-life cases, and can help us get a better understanding upon how black-box models rely on and extrapolate from the training data. (Koh & Liang, 2017) discussed different use cases of influence functions, including adversarial training (training set attacks), debugging domain mismatch, and fixing mislabeled examples.

In this section, we perform two additional experiments to exemplify the applications of influence functions. The first is to uncover the difference of two hyperplane classifiers and to understand their behaviors; the second is to use influence function to identify mislabeled training examples.

5.1. Understanding Model Behavior

Two models can make the same correct predictions and achieve similar level of performance, but get there in very different ways. Binary logistic regression and

linear support vector classifier are very simple models. They can be both regarded as a separating hyperplane that tries to cut the high-dimensional data into two classes. Nevertheless, they have very different behaviors and interpretations. This difference is, indeed, a direct result from the different mathematical form of the loss functions. Therefore, looking at $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ of both models for the same dataset can give us an intuitive picture in terms of how the two mathematical form of $\mathcal{L}(\boldsymbol{z}, \widehat{\boldsymbol{\theta}})$ trace back to the same training data in different ways.

For our purpose, we train the logistic regression and linear SVC on the same `Iris` dataset, and compute $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_{tr}, \boldsymbol{z}_{te})$ for *the same* testing point. By plotting the influence against the Euclidean distance to the decision boundary (see Figure 5), we found the two models behave differently in terms of how the training data are used for prediction: Both models put more leverage on the points that are closer to the separating hyperplane. However, logistic regression looks at more points; while linear SVC *does not* look at points that are too far away from the boundary: those points have zero influence on testing loss.

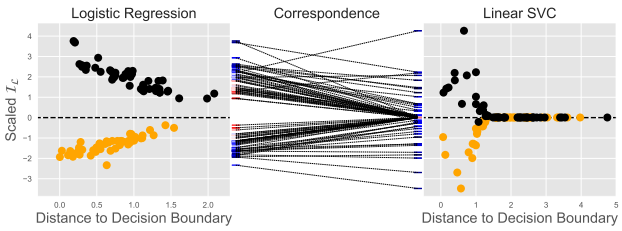

Figure 5. **Logistic regression V.S. Linear SVC.** We compare the behavior of logistic regression and linear SVC for the same testing point. For each training point $\boldsymbol{z}_i$, $\mathcal{I}_{\mathcal{L}}(\boldsymbol{z}_i, \boldsymbol{z}_{te})$ is plotted on y-axis; while $(\boldsymbol{x}_i^\top \widehat{\boldsymbol{\beta}} + \beta_0)/\left\|\widehat{\boldsymbol{\beta}}\right\|$, the Euclidean distance to the decision boundary is plotted on x-axis. **(Left):** For logistic regression, the absolute value of influence decreases slowly for points that are further away from the boundary, and it's always non-zero. **(Mid):** The thin dashed line connects the same training point in the two settings; the ticks on y-axis mark the influence level of that particular training point; the color of ticks stand for distance to the boundary: blueish for close, redish for faraway. **(Right):** For linear SVC, the influence decrease much faster, and those training points that are far beyond the margin have literally zero influence.

5.2. Fixing Mislabeled Examples

TODO

6. Software Package

We built a generic empirical risk optimizing framework with `Tensorflow` to compute influence functions for various models, and we also implemented CG and LiSSA algorithm to scale up to large datasets. With the help of auto-gradients system, the required gradients and hessian-vector products are automatically kept track of. The user only need to define the empirical risk funtion and its parametrization. All of our code, data, and reproducible experiments are available in our Github repository.

References

Agarwal, N., Bullins, B., and Hazan, E. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 18:116:1–116:40, 2017.

Cook, R.D. and Weisberg, S. *Residuals and Influence in Regression*. Monographs on statistics and applied probability. Chapman & Hall, 1982.

Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer New York Inc., 2001.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*, 2017.

Martens, J. Deep learning via hessian-free optimization. In *International Conference on Machine Learning (ICML)*, 2010.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160, 1994.

Wassermann, L. *All of nonparametric statistics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.