Understanding Black-Box predictions via Influence Functions [1]

Marko Milanovic Everseen/MATF

Belgrade, February 26, 2020

[1][Wei Koh, Liang 2017 ICML]

Introduction

Model interpretability: why does a model predict what it predicts?

- Make better decisions
- Improve the model
- Provide end-user with explanations

Introduction

Many works has been done on model interpretability:

- Which part of the input is most responsible for the model's prediction?
- What inputs activate neurons the most?
- Can the model be replaced with a simpler interpretable model?

Introduction

Many works has been done on model interpretability:

- Which part of the input is most responsible for the model's prediction?
- What inputs activate neurons the most?
- Can the model be replaced with a simpler interpretable model?

All these methods consider models as fixed!

Introduction

Influence functions enable us to study models as a function of their training data

Influence functions is a technique from robust statistics that helps us estimate the effect of removing particular training point from dataset

Introduction

Influence functions enable us to study models as a function of their training data

Influence functions is a technique from robust statistics that helps us estimate the effect of removing particular training point from dataset

They haven't been used in ML because:

- Require expensive second derivative calculation
- Assume differentiability
- Assume convexity

Influence functions

$\mathcal{X}$ - input space
$\mathcal{Y}$ - output space
$z_1, ..., z_n$ - training points, where $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
$L(z_i, \theta)$ - loss for a training point z and parameters $\theta \in \Theta$
$R(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta)$ - the empirical risk.
$\hat{\theta} \overset{def}{=} argmin_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta)$ - the empirical risk minimizer.

It is also assumed that empirical risk is twice-differentiable and strictly convex in θ

Influence functions

The goal is to understand the effect of training points to model's predictions. How would the model's predictions change if didn't have particular training point?

Influence functions

The goal is to understand the effect of training points to model's predictions. How would the model's predictions change if didn't have particular training point?

Let's study the change in model parameters due to removing a point z from training set:

$$\hat{\theta}_{-z} \overset{def}{=} argmin_{\theta \in \Theta} \frac{1}{n} \sum_{z_i \neq z} L(z_i, \theta)$$

Than, the change is given by:

$$\hat{\theta}_{-z} - \hat{\theta}$$

Not feasible!

Influence functions

Influence functions give us an efficient approximation:

Influence functions

Influence functions give us an efficient approximation:

$$\hat{\theta}_{\epsilon,z} \overset{def}{=} argmin_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z, \theta) \tag{1}$$

Influence functions

Influence functions give us an efficient approximation:

$$\hat{\theta}_{\epsilon,z} \overset{def}{=} argmin_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z, \theta) \tag{1}$$

Influence of upweighting z on the parameters $\hat{\theta}$ is given by:

$$\mathcal{I}_{up,params}(z) \overset{def}{=} \left. \frac{d\hat{\theta}_{(\epsilon,z)}}{d\epsilon} \right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1} \nabla_{\theta} L(z, \hat{\theta}) \tag{2}$$

Influence on parameters - proof

The Hessian matrix ($H_{\hat{\theta}}^{-1}$ exists by assumptions):

$$H_{\hat{\theta}} \overset{def}{=} \nabla^2 R(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L(z, \hat{\theta}) \tag{3}$$

The perturbed parameters $\hat{\theta}_{\epsilon,z}$ can be written as:

$$\hat{\theta}_{\epsilon,z} = \underset{\theta\in\Theta}{argmin}\{R(\theta) + \epsilon L(z, \theta)\} \tag{4}$$

Influence on parameters - proof

The Hessian matrix ($H_{\hat{\theta}}^{-1}$ exists by assumptions):

$$H_{\hat{\theta}} \overset{def}{=} \nabla^2 R(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta}^2 L(z, \hat{\theta}) \tag{3}$$

The perturbed parameters $\hat{\theta}_{\epsilon,z}$ can be written as:

$$\hat{\theta}_{\epsilon,z} = \underset{\theta \in \Theta}{argmin} \{R(\theta) + \epsilon L(z, \theta)\} \tag{4}$$

Define the parameter change:

$$\Delta_{\epsilon} = \hat{\theta}_{\epsilon,z} - \hat{\theta} \tag{5}$$

The quantity we're interested into:

$$\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon} = \frac{d\Delta_{\epsilon}}{d\epsilon} \tag{6}$$

Influence on parameters - proof

$\hat{\theta}_{\epsilon,z}$ is the minimizer of the empirical risk (4):

$$0 = \nabla R(\hat{\theta}_{\epsilon,z}) + \epsilon \nabla L(z, \hat{\theta}_{\epsilon,z}) \tag{7}$$

Since $\hat{\theta}_{\epsilon,z} \to \hat{\theta}$ as $\epsilon \to 0$ we get by Taylor expansion:

$$\begin{aligned} 0 \approx & \left[\nabla R(\hat{\theta}) + \epsilon \nabla L(z, \hat{\theta})\right] \\ & + \left[\nabla^2 R(\hat{\theta}) + \epsilon \nabla^2 L(z, \hat{\theta})\right] \Delta_\epsilon \end{aligned} \tag{8}$$

Solving for Δ_ϵ:

$$\begin{aligned} \Delta_\epsilon \approx & - \left[\nabla^2 R(\hat{\theta}) + \epsilon \nabla^2 L(z, \hat{\theta})\right]^{-1} \\ & \left[\nabla R(\hat{\theta}) + \epsilon \nabla L(z, \hat{\theta})\right] \end{aligned} \tag{9}$$

Influence on parameters - proof

Since $\hat{\theta}$ minimizes R and keeping only $O(\epsilon)$ we get:

$$\Delta_\epsilon \approx -\nabla^2 R(\hat{\theta})^{-1} \nabla L(z, \hat{\theta})\epsilon \tag{10}$$

Influence on test examples

The influence of upweighting z on the loss at a test point z_{test}

$$\mathcal{I}_{up,loss}(z) \overset{def}{=} \left. \frac{dL(z_{test}, \hat{\theta}_{\epsilon,z})}{d\epsilon} \right|_{\epsilon=0} \tag{11}$$

Influence on test examples

The influence of upweighting z on the loss at a test point z_{test}

$$\begin{aligned}\mathcal{I}_{up,loss}(z) &\overset{def}{=} \left.\frac{dL(z_{test}, \hat{\theta}_{\epsilon,z})}{d\epsilon}\right|_{\epsilon=0} \\ &= \nabla_\theta L(z_{test}, \hat{\theta})^\top \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} \\ &= -\nabla_\theta L(z_{test}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})\end{aligned} \tag{11}$$

Relation to Euclidean distance

To find a training point most relevant to a test point it is natural to observe the closest point in Euclidean space (equivalent to $x_{test} \cdot x$)

Relation to Euclidean distance

To find a training point most relevant to a test point it is natural to observe the closest point in Euclidean space (equivalent to $x_{test} \cdot x$)

Logistic regression:

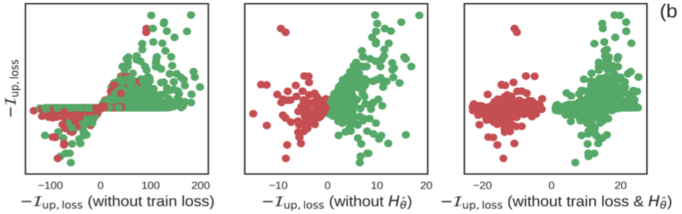
- $p(y|x) = \sigma(y\theta^\top x)$ where $y \in (-1, 1)$ and $\sigma(t) = \frac{1}{1+exp(-t)}$
- $L(z, \theta) = log(1 + exp(-y\theta^\top x))$
- $\nabla_\theta L(z, \theta) = -\sigma(-y\theta^\top x)yx$
- $H_\theta = \frac{1}{n}\sum_{i=1}^{n} \sigma(\theta^\top x_i)\sigma(-\theta^\top x_i)x_i x_i^\top$

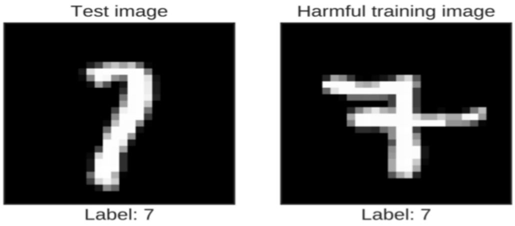
Relation to Euclidean distance

$$I_{up,loss}(z, z_{test}) = \\ y_{test} y \cdot \sigma(-y_{test}\theta^\top x_{test}) \cdot \sigma(-y\theta^\top x) \cdot x_{test}^\top H_{\hat{\theta}}^{-1} x$$

Relation to Euclidean distance

$$I_{up,loss}(z, z_{test}) = \\ y_{test} y \cdot \sigma(-y_{test}\theta^\top x_{test}) \cdot \sigma(-y\theta^\top x) \cdot x_{test}^\top H_{\hat{\theta}}^{-1} x$$





Efficiently Computing Influence

There are two computational challenges to get $\mathcal{I}_{up,loss}(z, z_{test})$:

- forming and inverting Hessian matrix $H_{\hat{\theta}}$ which requires $O(np^2 + p^3)$ operations
- we often want to calculate the influence $\mathcal{I}_{up,loss}(z, z_{test})$ across all training points z_i

Efficiently Computing Influence

There are two computational challenges to get $\mathcal{I}_{up,loss}(z, z_{test})$:

- forming and inverting Hessian matrix $H_{\hat{\theta}}$ which requires $O(np^2 + p^3)$ operations
- we often want to calculate the influence $\mathcal{I}_{up,loss}(z, z_{test})$ across all training points z_i

Idea is to avoid explicitly computing $H_{\hat{\theta}}^{-1}$; instead Hessian-vector products (HPVs) are used to efficiently approximate

$$s_{test} = H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{test}, \hat{\theta})$$

and than compute

$$\mathcal{I}_{up,loss}(z, z_{test}) = -s_{test} \cdot \nabla_\theta L(z, \hat{\theta})$$

Efficiently Computing Influence

Conjugate Gradients

Transform matrix inversion problem into an optimization problem:

$$H_{\hat{\theta}}^{-1} v \equiv argmin_t \{ \frac{1}{2} t^\top H_{\hat{\theta}} t - v^\top t \}$$

This can be solved with CG that only require evaluation of $H_{\hat{\theta}} t$ which takes $O(np)$ time

Exact solution takes p iterations, while in practice good approximation can be obtained after fever iterations

Efficiently Computing Influence

Stochastic Estimation[2]

With large datasets CG can be still very slow (each iteration goes through all training points) ...

$$H_j^{-1} \overset{def}{=} = \sum_{i=0}^{j} (I - H)^i$$

Rewrite recursively:

$$H_j^{-1} = I + (I - H) H_{j-1}^{-1}$$

The key is that at each iteration H can be replaced with any unbiased estimator of H. Concretely, z_i can be sampled at random and $\nabla_\theta^2 L(z_i, \hat{\theta})$ can be used as unbiased estimator.

[2] [Agarwal et al., 2016]

Efficiently Computing Influence

Stochastic Estimation - procedure

1. Uniformly sample t training points $z_{s_1}, \ldots, z_{s_t}$
2. Define $\tilde{H}_0^{-1} v = v$
3. Recursively compute $\tilde{H}_j^{-1} v = v + (I - \nabla_\theta^2 L(z_i, \hat{\theta}))\tilde{H}_{j-1}^{-1} v$
4. Repeat steps 1-3. r times and average results

With this procedure we can compute influence $\mathcal{I}_{up,loss}(z, z_{test})$ in $O(np + rtp)$ time. It is empirically shown that setting $rt = O(n)$ gives accurate results

Validations and Extensions

Influence Functions vs. leave-one-out retraining

To asses the accuracy of influence functions $-\frac{1}{n}\mathcal{I}_{up,loss}(z, z_{test})$ is compared with $L(z_{test}, \hat{\theta}_{-z}) - L(z_{test}, \hat{\theta})$

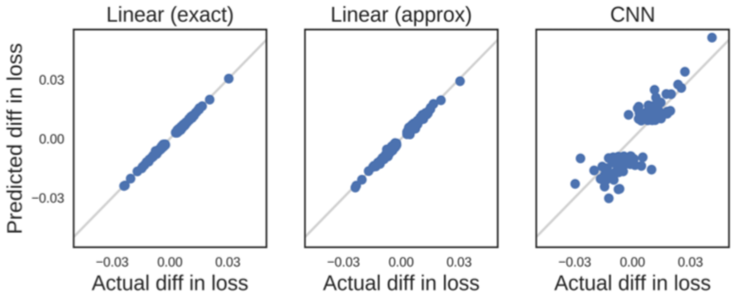
Logistic regression model is trained on 10-class MNIST dataset

Validations and Extensions

Influence Functions vs. leave-one-out retraining

To asses the accuracy of influence functions $-\frac{1}{n}\mathcal{I}_{up,loss}(z, z_{test})$ is compared with $L(z_{test}, \hat{\theta}_{-z}) - L(z_{test}, \hat{\theta})$

Logistic regression model is trained on 10-class MNIST dataset



Validations and Extensions

Non-convexity and non-convergence

What if $\tilde{\theta}$ are obtained by running SGD with early stopping on non-convex objectives?

Clearly $\tilde{\theta} \neq \hat{\theta}$ and as a result $H_{\tilde{\theta}}$ could have negative eigenvalues.

Validations and Extensions

Non-convexity and non-convergence

What if $\tilde{\theta}$ are obtained by running SGD with early stopping on non-convex objectives?

Clearly $\tilde{\theta} \neq \hat{\theta}$ and as a result $H_{\tilde{\theta}}$ could have negative eigenvalues.

Convex quadratic approximation of the loss:

$$\tilde{L}(z, \theta) = L(z, \tilde{\theta}) + \nabla L(z, \tilde{\theta})^\top (\theta - \tilde{\theta}) + \frac{1}{2}(\theta - \tilde{\theta})^\top (H_{\tilde{\theta}} + \lambda I)(\theta - \tilde{\theta})$$

Influence functions $\mathcal{I}_{up,loss}$ are computed using $\tilde{L}$

Validations and Extensions
Non-differentiable losses

What if $\nabla_\theta L$ or $\nabla^2_\theta L$ do not exist?

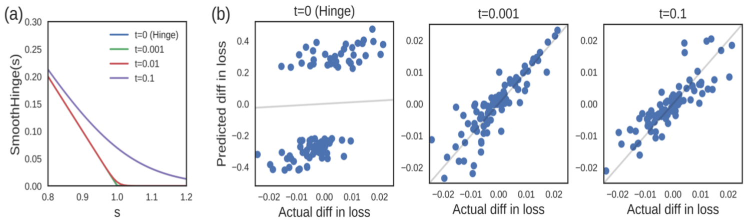
Linear SVM model is trained on MNIST 1s vs. 7s classification

Loss function: $Hinge(s) = max(0, 1-s)$

Differentiable loss function:
$SmoothHinge(s,t) = tlog(1 + exp(\frac{1-s}{t}))$

Validations and Extensions

Non-differentiable losses



Correlation remained high over a wide range of t:

- $t = 0.001$, Pearson's $R = 0.95$
- $t = 0.1$, Pearson's $R = 0.91$

Applications

1.Understanding model behavior

Influence functions reveal insights about how models rely on the training data.

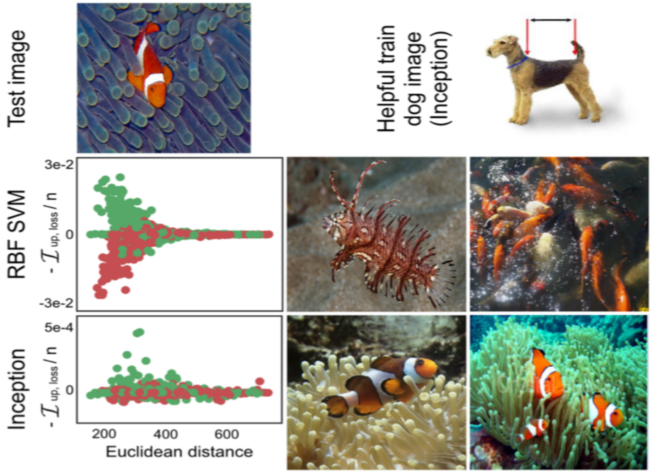
Problem: dog vs. fish classification (900 images from both classes were taken from ImageNet)

Two models are compared:

- Inception V3 model with all but the top layer frozen
- SVM with RBF kernel

Applications

1. Understanding model behavior



Applications

2. Debugging domain mismatch

Influence functions can identify the training examples most responsible for the errors, helping model developers identify domain mismatch.

Case study: predict whether a patient will be readmitted.
Data set: 20K diabetic patients from 100+ US hospitals each represented by 127 features (3 out of 24 children under the age of 10 were readmitted)
Domain mismatch: 20 out of 21 healthy children were filtered out

Applications

2. Debugging domain mismatch

What caused the model to make those mistakes?

Observing model parameters didn't help (14 out of 127 parameters were bigger than one indicating a child)

For a random z_{test} that model made a mistake $-\mathcal{I}_{up.loss}(z_i, z_{test})$ is calculated for each training point z_i. This clearly highlighted 4 training children (absolute value of the influence is 30-40 times higher than the next most influential point)

Applications

3. Fixing mislabeled examples

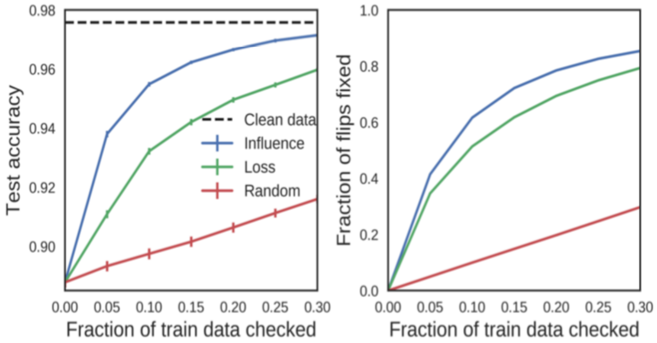
Key idea is to flag the training points that exert the most influence on the model.

Since test set is not available influence of z_i is measured by $\mathcal{I}_{up,loss}(z_i, z_i)$

Case study: email spam classification
labels of 10% of the training data were randomly flipped

Applications

3. Fixing mislabeled examples



Prioritizing checking training examples using influence functions outperforms baseline method that selects the points based on train loss

Applications

4. Adversarial training examples

How would the model's predictions change if a training input were modified?

Applications

4. Adversarial training examples

How would the model's predictions change if a training input were modified?

$z_\delta \overset{def}{=} (x + \delta, y)$

$\hat{\theta}_{z_\delta, -z}$ - empirical risk minimizer on the training points with z_δ in place of z

$\hat{\theta}_{\epsilon, z_\delta, -z} \overset{def}{=} argmin_\theta \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z_\delta, \theta) - \epsilon L(z, \theta)$

Applications

4. Adversarial training examples

An analogous calculation for $\mathcal{I}_{up,params}$ yields:

$$\begin{aligned} \left.\frac{d\hat{\theta}_{\epsilon,z_\delta,-z}}{d\epsilon}\right|_{\epsilon=0} &= \mathcal{I}_{up,params}(z_\delta) - \mathcal{I}_{up,params}(z) \\ &= -H_{\hat{\theta}}^{-1}(\nabla_\theta L(z_\delta,\hat{\theta}) - \nabla_\theta L(z,\hat{\theta})) \end{aligned} \tag{12}$$

Applications

4. Adversarial training examples

An analogous calculation for $\mathcal{I}_{up,params}$ yields:

$$\left.\frac{d\hat{\theta}_{\epsilon,z_\delta,-z}}{d\epsilon}\right|_{\epsilon=0} = \mathcal{I}_{up,params}(z_\delta) - \mathcal{I}_{up,params}(z) \\ = -H_{\hat{\theta}}^{-1}(\nabla_\theta L(z_\delta,\hat{\theta}) - \nabla_\theta L(z,\hat{\theta})) \qquad (12)$$

If we assume that L is differentiable in θ and x then as $||\delta|| \to 0$ we have

$$\nabla_\theta L(z_\delta,\hat{\theta}) - \nabla_\theta L(z,\hat{\theta}) \approx [\nabla_x \nabla_\theta L(z,\hat{\theta})]\delta$$

Thus:

$$\left.\frac{d\hat{\theta}_{\epsilon,z_\delta,-z}}{d\epsilon}\right|_{\epsilon=0} \approx -H_{\hat{\theta}}^{-1}[\nabla_x \nabla_\theta L(z,\hat{\theta})]\delta$$

Applications

4. Adversarial training examples

Finally, differentiating w.r.t δ gives us

$$\begin{aligned}\mathcal{I}_{pert,loss}(z, z_{test})^\top &\overset{def}{=} \nabla_\delta L(z_{test}, \hat{\theta}_{z_\delta,-z})^\top \Big|_{\delta=0} \\ &= -\nabla_\theta L(z_{test}, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_x \nabla_\theta L(z, \hat{\theta})\end{aligned} \tag{13}$$

Applications

4. Adversarial training examples

Creating adversarial training examples:

- Initialize $\tilde{z}_i = z_i$
- Iterate $\tilde{z}_i := \prod(\tilde{z}_i + \alpha \cdot sign(\mathcal{I}_{pert,loss}(\tilde{z}_i, z_{test})))$
- Retrain the model after each iteration

$\prod$ - denotes projection onto the set of valid images that share the same 8-bit representation with z_i

Applications

4. Adversarial training examples

Experiment: dog vs. fish classification using Inception network with all but the top layer frozen. Originally the model correctly classified 591/600 images

Results:

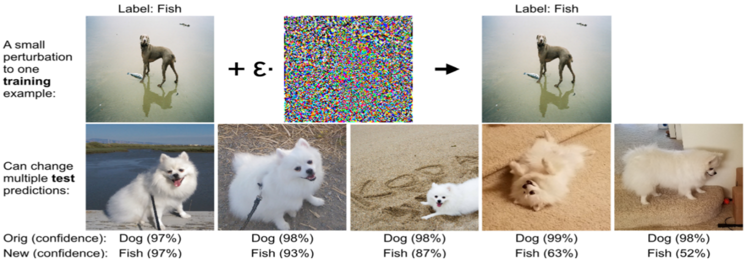
- 335/591 (57%) images were flipped by perturbing 1 training example
- 455/591 (77%) images were flipped by perturbing 2 training examples
- 590/591 (99%) images were flipped by perturbing 10 training examples

Applications

4. Adversarial training examples

Observations:

- Even though the change in pixel values is small, the change in final NN layer is significantly larger
- More training examples reduce chances for training attacks
- Ambiguous or mislabeled examples are effective points to attack


Label: Fish
Label: Fish
A small perturbation to one **training** example:
+ ε·
Can change multiple **test** predictions:
Orig (confidence): Dog (97%) Dog (98%) Dog (98%) Dog (99%) Dog (98%)
New (confidence): Fish (97%) Fish (93%) Fish (87%) Fish (63%) Fish (52%)

Implementations

- Tensorflow (author's implementation): https://github.com/kohpangwei/influence-release/
- PyTorch: https://github.com/nimarb/pytorch_influence_functions

Thank you!

Questions?

References

Koh, P. W., & Liang, P. (2017) Understanding Black-box Predictions via Influence Functions

Agarwal, N. et al. Second-Order Stochastic Optimization for Machine Learning in Linear Time