Influenciæ: A library for tracing the influence back to the data-points

Agustin Martin Picard, Lucas Hervier, Thomas Fel, David Vigouroux

Influenciæ
A library for tracing the influence back to the data-points

Agustin Picard[*1,2], **Lucas Hervier**[*1,2], **Thomas Fel**[2,3,4], **David Vigouroux**[1,2]
[1]Institut de Recherche Technologique Saint-Exupery
[2]Artificial and Natural Intelligence Toulouse Institute
[3]Carney Institute for Brain Science, Brown University
[4]Innovation & Research Division, SNCF

Abstract

In today's AI-driven world, understanding model behavior is becoming more important than ever. While libraries abound for doing so via traditional XAI methods, the domain of influence-based techniques for data-centric explanations remains mostly underserved. To fill this void, we introduce **Influenciæ**, an open-source library that implements the state-of-the-art methods for estimating the influence of training points on the model, with a focus on efficiency and scalability to fit the needs and the recent trends in the field. Finally, we have thoroughly documented and included plenty of tutorials to make the library reachable to the public, in the hopes that it will bring these methods back into the spotlight.

1 Introduction

In the rapidly evolving landscape of artificial intelligence, eXplainable AI (XAI) has emerged as a critical domain, bridging the gap between the opacity of complex models and the need for transparency and accountability. The importance of XAI cannot be overstated, particularly as AI systems become increasingly integrated into various facets of our daily lives, from healthcare to finance and beyond. To ensure the widespread adoption of AI in these critical domains, it is imperative to not only build highly accurate models but also to gauge their reliability and interpretability. In this pursuit, traditional XAI techniques offer a plethora of post-hoc methods for understanding predictions [16, 7, 40, 10, 21, 41, 32, 38, 34, 28, 33, 25, 12, 24, 35, 26, 36]. What if we could directly trace model behavior back to the training data itself?

The first reference to using a notion of the influence of training data in the model's implemented function as a means to explain the behavior of deep neural networks dates back to [18], where Koh & Liang proposed a technique to approximately compute influence functions without needing to re-train the model – one of the major setbacks of (approximate) leave-k-out techniques. This allowed them to attribute some of the model's predictions to the presence of certain "influential" data-points in the training dataset. However, as any approximation, it comes with some limitations [6, 3, 29], and other alternatives have since been released with the promise of improving on their results. Namely, in [39], the authors propose to use the *representer point theorem* for kernels to re-write the function implemented by the neural network as a kernel decomposed in a sum of the effects of each of the training data-points, and use the coefficient associated to each of them as a notion of their influence on the actual model. Nevertheless, this procedure requires the training of a surrogate model at the last layer, thus potentially limiting the faithfulness of the method. As a response to this, an improvement was devised in [37], where this was no longer required.

Preprint.

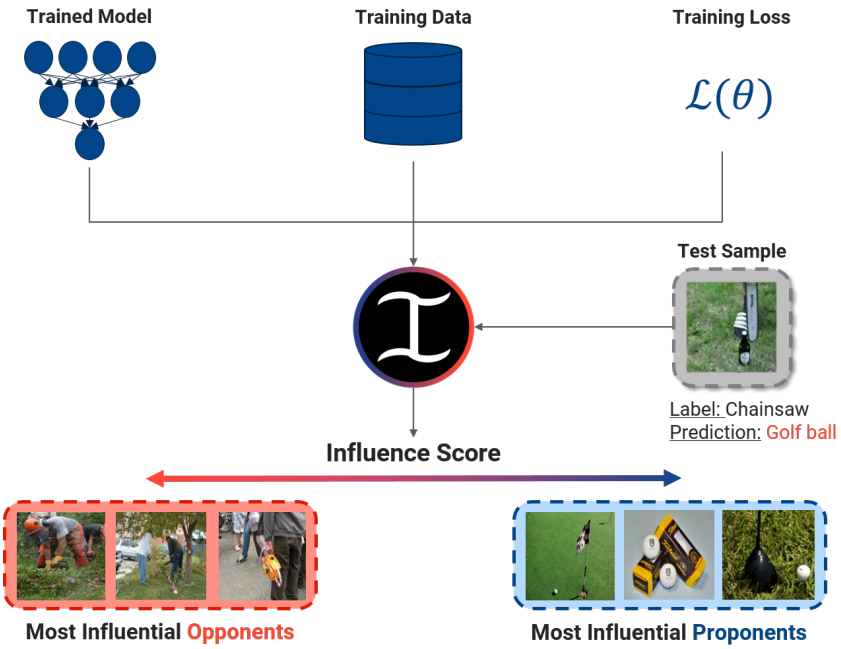

Figure 1: **Influenciæ.** Our library implements a plethora of influence-based methods for attributing model behavior to training data (see Table 1). We showcase how we can find the most influential points for a wrong prediction from the Imagenette subset of the ILSVRC [9] dataset, and thus, better understand why the model got it wrong. These examples can be separated in two classes: **Proponents** whose presence in the training set helps lower the loss on the sample under study; and **Opponents** that increase the model's loss – *i.e.* confuse the model. For a complicated test sample labeled as *Chainsaw* but predicted as *Golf ball*, we observe that the main opponents are images of chainsaws where the blade isn't quite visible, while the proponents contain mostly images of golf balls in different settings.

In yet another vein, [15] introduced another approach to approximating leave-one-out (LOO) and used it as a way of extracting insights from the training dataset. In particular, they propose to train a linear model to predict a quantity related to an set of neural networks based on the presence of a point (or group of points) on the choice of the subsample of the training dataset.

Nonetheless, all the aforementioned methods suffer from a high computational and memory costs, and this can be exacerbated if their implementation is not correctly optimized. For instance, there are several ways of approximately computing the product of the inverse of a given matrix and a vector without needing to materialize the matrix in memory [23, 2, 30] that demand some knowledge on how auto-differentiation frameworks work to obtain the most optimized implementations. Additionally, dealing with very large datasets can be quite cumbersome, as the amount of elements that need to be stored in memory can be quite important, and thus why lazy and batched computations can help lessen memory requirements.

In summary, our contribution addresses the pressing need for readily available and open-source, reliable AI models and methods to measure their reliability. From the point of view of post-hoc, attribution, feature visualization and concept-based XAI, this has already been (at least, partially) addressed via some well-known libraries [11, 20, 17, 22], but, to the best of our knowledge, this has not been the case for influence-based techniques. Thus, we introduce **Influenciæ**, an open-source Python package optimized for TensorFlow models that implements most of the prominent methods for computing data influence in the literature, complemented with documentation and a set of user-friendly tutorials that walk users through the process of getting started with our library. The essence of our library is schematized in Fig. 1. We hope that this library will make data-attribution methods more accessible to everyone in the community.

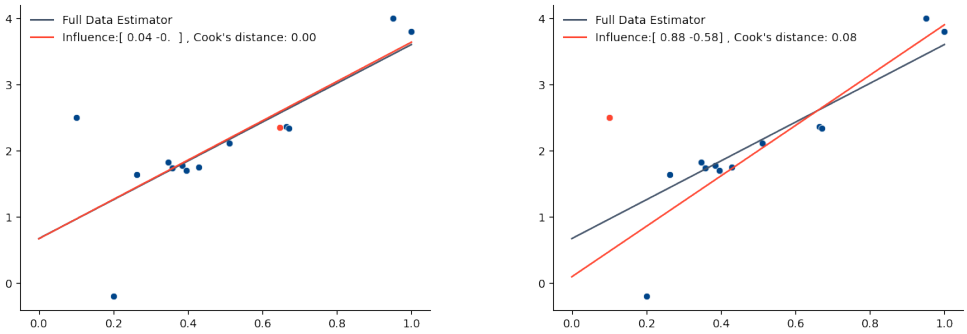

Figure 2: **Illustrating Influence functions.** For a linear regression problem defined by the dataset constituted by the blue data-points and an eventual held-out red data-point, we can compute the influence function for two different cases of held-out points, yielding two different perturbed models. For the red point on the left, due to its position (outlier), its influence will be much higher than that of a data-point that's close to other points, and thus, provide a model that's much more different than in the other case. This can be quantified by the Cook's distance – *i.e.* the influence values – and estimate the importance of each of these red data-points in the model.

2 Attributing model behavior through data influence

The idea behind influence is that provided with a data point, a model, and all the other data used in constructing the model, the Influence Function quantifies the extent to which the data contributed positively or negatively to the current state or behavior of the model. To illustrate this, let's consider the entirety of data points presented in Figure 2 and the resulting estimator, symbolized by the blue line – a common outcome for both scenarios. However, if one chooses to retain the distinctive red data point from the training dataset, the estimator takes on a new form—the red curve. This stark contrast becomes evident when observing the two graphs (right and left). Depending on the specific data point held out during the model's training phase, the resulting machine learning model can exhibit significantly divergent behavior compared to when trained on the complete dataset. The influence vector associated with the excluded data point provides valuable insights into the disparity between the perturbed model and the original one. Additionally, it offers guidance by indicating the "direction" in which the perturbed model moved with respect to the original model when the data-point was held out from the training dataset.

In this section, we aim to comprehensively cover **all the methods that we implemented in the toolkit**, introducing each one with technical details and highlighting their advantages and drawbacks. For any end-user familiar with the different techniques at stake, a summary is provided in Table 1.

2.1 Notation

Throughout the paper, each one of the methods will explain a machine learning model $h : \mathcal{X} \to \mathcal{Y}$, with $\mathcal{X}$ and $\mathcal{Y}$ being respectively the input and output domain. In particular, this model is parameterized by the weights $\theta \in \Theta \subseteq \mathbb{R}^d$. If not specified otherwise, h is trained on a training dataset $\mathcal{D}_{train} \subset (\mathcal{X} \times \mathcal{Y})$ of size n to minimize a loss function $\ell : (\mathcal{X}, \mathcal{Y}, \Theta) \to \mathbb{R}$. We denote a sample by the tuple $z = (x, y) | \quad x \in \mathcal{X}, y \in \mathcal{Y}$. When an index subscript as i or j is added, *e.g.* z_i, it is assumed that z_i belongs to the training dataset. If the subscript "test" is added, z_{test}, the sample does not belong to the training data. When there is no subscript, the sample can either belong to the training data or not. Finally, the empirical risk function is denoted as $\mathcal{L}(\theta) := \frac{1}{n} \sum_{(x,y) \in \mathcal{D}_{train}} \ell(x, y, \theta) = \frac{1}{n} \sum_{z_j \in \mathcal{D}_{train}} \ell(z_j, \theta)$, the parameters that minimized this empirical risk as $\theta^* := \arg\min_\theta \mathcal{L}(\theta)$ and an estimator of θ^* is denoted $\hat{\theta}$.

Influence Calculator		
Paper	Method Name	Object name in Influenciae
[18]	First Order Influence Function (Eq. 2)	`FirstOrderInfluenceCalculator` [C, T]
[4]	RelatIF (Eq. 3)	`FirstOrderInfluenceCalculator` [C, T]
[19]	First Order Group Influence	`FirstOrderInfluenceCalculator` [C, T]
[6]	Second Order Group Influence	`SecondOrderInfluenceCalculator` [C, T]
[39]	RPS-L2 (Eq. 4)	`RepresenterPointL2` [C, T]
[37]	RPS-LJE (Eq. 5)	`RepresenterPointLJE` [C, T]
[27]	TracIn (Eq. 6)	`TracIn` [C, T]
[30], [2]	Arnoldi Iteration Influence Functions	`ArnoldiInfluenceCalculator` [C, T]
Inverse Hessian Vector Product		
Paper	Method Name	Object name in Influenciae
N/A	Exact computation	`ExactIHVP` [C]
[1]	LiSSA	`LissaIHVP` [C]
[23]	Conjugate Gradient Descent	`ConjugateGradientIHVP` [C]

Table 1: **Summary of the methods available in the Influenciæ library.** The techniques for computing influence – i.e. `InfluenceCalculators` and inverse-hessian-vector products (IHVPs) that we have implemented in the library thus far. The C and T captions following the object name are clickable link to respectively the source code and the tutorial (if available). We showcase some examples in Fig. 3.

2.2 Influence functions

Influence functions originated from the field of robust statistics in the early 70s. In essence, they evaluate the change of a model's parameters as one up-weights a training sample by an infinitesimal amount ϵ [13]: $\hat{\theta}_{\epsilon,z_j} := \arg\min_\theta \mathcal{L}(\theta) + \epsilon\ell(z_j, \theta)$. One way to estimate the change in a model's parameters of a single training sample would be to perform leave-one-out (LOO) retraining, that is, to train the model again with the sample of interest being held out of the training dataset. However, repeatedly re-training the model to retrieve the parameters' changes exactly can be computationally prohibitive, especially when the dataset size and/or the number of parameters grows. As removing a sample z_j can be linearly approximated by up-weighting it by $\epsilon = -\frac{1}{n}$, computing influence helps to estimate the change of a model's parameters if a specific training point was removed. Thus, by making the assumption that the empirical risk $\mathcal{L}$ is twice-differentiable and strictly convex with respect to the model's parameters θ – making the Hessian $H_{\hat{\theta}} := \frac{1}{n}\sum_{z_i \in \mathcal{D}_{train}} \nabla^2_\theta \ell(z_i, \hat{\theta})$ positive definite –, Cook & Weisberg [8] proposed to compute the influence of z_j on the parameters $\hat{\theta}$ as:

$$\mathcal{I}(z_j) := -H^{-1}_{\hat{\theta}} \nabla_\theta \ell(z_j, \hat{\theta}) \tag{1}$$

Later, Koh and Liang [18] popularized influence functions in the machine learning community as they took advantage of auto-differentiation frameworks to efficiently compute the hessian for Deep Neural Networks (DNNs) and derived Eq. 1 to formulate the influence of up-weighting a training sample z_j on the loss at a test point z_{test}:

$$\text{IF}(z_j, z_{test}) := -\nabla_\theta \ell(z_{test}, \hat{\theta})^T H^{-1}_{\hat{\theta}} \nabla_\theta \ell(z_j, \hat{\theta}) \tag{2}$$

[18] in Influenciae

This approach is the one implemented in the Influenciae library as the FirstOrderInfluenceCalculator object. One can discover more in the Source Code and the dedicated Tutorial.

This formulation opens its way into example-based XAI as it compares to the study of finding the nearest neighbors of z_{test} in the training dataset – *i.e.* the most similar examples – albeit with two major differences: i) points with high training loss are given more influence *revealing that outliers*

can dominate the model's parameters [18], and ii) $H_{\hat{\theta}}^{-1}$ measures what Koh & Liang called: *the resistance of the other training points to the removal of* z_j [18].

2.2.1 Inverse hessian vector product

It should be noted that hessian computation remains a significant challenge. Although one could perform the exact computation of the hessian and then invert it, it can computationally prohibitive as it requires second order derivatives to be calculated for all the target weights, and the inversion is typically $\mathcal{O}(n^3)$. Thus, the library includes several methods to estimate the inverse-hessian-vector product (IHVP) in Eq. 2. As suggested in [18], the stochastic estimation via [1] (LiSSA: Linear time Stochastic Second-order Algorithm) or through conjugate gradient descent [23] are implemented. Furthermore, it also includes the Arnoldi algorithm [2] as it is performed by Schioppa et al. in [30] (in which they also compare the three approaches).

IHVP in Influenciae

The library includes all the aforementioned IHVP: ExactIHVP, ConjugateGradientDescentIHVP [23] and LissaIHVP [1]. Those three objects can be used as a parameter for InfluenceCalculator needing to perform IHVP. The Arnoldi algorithm [2] is also implemented but it is directly embedded in its own InfluenceCalculator: ArnoldiInfluenceCalculator [30]. One can discover more concerning the IHVP operators in the Source Code.

2.2.2 RelatIF

As highlighted by Barshan et al. [4], the influential instances yielded by Eq. 2 tend to overlap for vastly different test samples due to the overpowering effect of the influence of the most influential training samples over the test samples. To overcome this issue, they suggest putting constraints on the re-weighting of influential instances by normalizing the formulation in 2:

$$\text{RelatIF}(z, z_{test}) := \frac{\mathcal{I}_{up,loss}(z, z_{test})}{||H_{\hat{\theta}}^{-1} \nabla_\theta \ell(z, \hat{\theta})||} \tag{3}$$

[4] in Influenciae

By setting the parameter normalize of the FirstOrderInfluenceCalculator to true at initialization, one is actually using RelatIF. One can discover more in the Source Code and the dedicated Tutorial.

2.2.3 Group influence

Oftentimes, we are not only interested in the influence of individual instances but rather in the influence of a group of training samples (*e.g.* mini-batch effect, multi-source data, etc., as per [19].). Koh et al. [19] propose to utilize the sum of individual influences, and demonstrate that this constitutes a reliable proxy for ranking groups based on their influence.

[19] in Influenciae

As this approach is derived from [18], you can compute the influence of a group of data points using the same FirstOrderInfluenceCalculator object but using a method dedicated to group influence. One can discover more in the Source Code and the dedicated Tutorial.

However, this approximation leads to large absolute and relative errors. In addition, Basu et al. [6] point out that such approximations ignore possible cross-correlations between samples in the group. Thus, they suggest studying second-order approximations instead "*to capture model changes when a potentially large group of training samples is up-weighted*".

[6] in Influenciae

This work is encapsulated in the SecondOrderInfluenceCalculator object of the library. One can discover more in the Source Code and the dedicated Tutorial.

However, it should be highlighted that their formulation can be computationally demanding, and they specify that it holds for linear prediction models where the underlying optimization is convex, but could eventually break in the case of DNNs. In a later work, Basu et al. [5] further investigated the most appropriate settings for reliably computing such influence scores from a theoretical standpoint.

2.3 Kernel-based influence

Besides influence functions, other paradigms exist for finding the training examples that are the most responsible for a given set of predictions. For instance, Yeh et al. [39] suggest an approach that leverages kernels and the representer point theorem [31] "*which loosely states that under certain conditions, the minimizer of a loss functional over a reproducing kernel Hilbert space (RKHS) can be expressed as a linear combination of kernel evaluations at training points*". Therefore, they decide to focus on explaining only the *pre-activation prediction layer* of a neural network $\Phi(x_i, \theta) := \theta_1 \mathrm{f}_i$ with θ_1 the parameters of the last classification layer and f_i the last intermediate layer feature for input x_i. Considering these last notations, they posit that if $\tilde{\theta}$ constitutes a stationary point of the optimization problem: $\arg\min_\theta \{\mathcal{L}(\theta) + \lambda ||\theta_1||^2\}$ for some $\lambda > 0$, then it is possible to compute the representer value for z_{test} given z as follows:

$$\mathrm{RPSL2}(z, z_{test}) := \frac{1}{-2\lambda n} \frac{\partial \ell(z, \theta)}{\partial \Phi(x, \theta)} * \mathrm{f}_z^T \mathrm{f}_{z_{test}} = \alpha(z) * \kappa(z, z_{test}) \,, \tag{4}$$

where $\alpha(z) = \frac{1}{-2\lambda n} \frac{\partial \ell(z,\theta)}{\partial \Phi(x,\theta)}$ and $\kappa(z, z_{test}) = \mathrm{f}_z^T \mathrm{f}_{z_{test}}$. This representer value is the quantity that will define as the influence of an instance, with its sign providing additional insights: positive representer values are excitatory while negative ones are inhibitory to the prediction at the given test point. Note that $\alpha(z)$ could be used as an importance measure of the training sample z on θ_1.

[39] in Influenciae

This approach can be reproduced with Influenciae by using the RepresenterPointL2 object. One can discover more in the Source Code and the dedicated Tutorial.

On one hand, this formulation has the significant advantage of being less memory intensive to compute as it does not require the computation of hessian matrices. On the other hand, it suffers on crucial drawback: it works only for models that perform a linear matrix multiplication before the final activation and one needs to introduce a heavy $L2$ regularization on the last layer of the model during the optimization phase, thus one cannot explain the behavior of pre-trained models. One workaround for the latter can be to retrain a regularized model while imposing some closeness between the retrained model's outputs and the original one's.

However, Sui et al. [37] state that, despite the similarity between these two models' outputs, disagreements between the two can still exist. Moreover, they point out that the previous approach tends to yield a static ranking of training samples for test points in the same class providing more of a *class-level* explanation rather than an *instance-level* explanation. In order to overcome those issues they suggest a derivation for Representer Point Selection (RPS) based on a Local Jacobian Taylor expansion (LJE). In practice, they suppose that they have a model $\hat{\theta}$ such that: $\sum_{i=1}^{n} \frac{\partial \ell(z_i, \hat{\theta})}{\partial \theta_1}|_{\theta_1 = \hat{\theta_1}} \simeq 0$. By taking a one-step gradient ascent from the trained model they obtain θ_1^*, which is supposed to be close to the original model's parameters $\hat{\theta}_1$, but with a small shift in the loss's landscape. With the help of a first-order Taylor expansion they propose to rewrite Eq. 4 as:

$$\mathrm{RPSLJE}(z, z_{test}) := \theta_1^* \frac{1}{\mathrm{f}_z n} - \frac{1}{n} H_{\theta_1^*}^{-1} \frac{\partial \ell(z, \theta^*)}{\partial \Phi(x, \theta^*)} * \kappa(z, z_{test}) \tag{5}$$

Figure 3: An example of all the methods available in Influenciae in application. For each test sample, we applied 2 different techniques and plotted the 3 most influential **Proponent** samples – whose presence in the training set helps lower the loss on the sample under study; and the 3 most influential **Opponent** samples – whose presence confuses the model.

[37] in Influenciae

One can leverage this derivation through the RepresenterPointLJE class. One can discover more in the Source Code and the dedicated Tutorial.

While they need the same assumptions as [18] – *i.e.* that ℓ be twice differentiable and strictly convex – they only require them with respect to the last linear layer's parameters θ_1. In addition, their approach only requires one step of gradient ascent, compared to a possible retraining with RPS-L2, but they need stronger assumptions mainly because of the hessian.

2.4 Tracing influence throughout the training process

Another popular approach for identifying *influential instances* involves leveraging training dynamics – which in practice is done by replaying the training process with model checkpoints in a post-hoc

fashion. Mainly, such approaches rely neither on being near optimality nor being strongly convex, which is more realistic considering the reality of DNNs.

Pruthi et al. [27] propose to save only model checkpoints θ_{t_i}, and with that information, they decompose the difference between the loss of the data-point under study at the end of training compared to at the beginning of training along the path taken by the training process. Supposing that one has k checkpoints $\theta^{[t_1]}, ..., \theta^{[t_k]}$ corresponding to iterations $t_1, ..., t_k$ and that the step size η_i is kept constant between checkpoints $i-1$ and i we can use the following as an influence measure:

$$TracInCP(z, z_{test}) = \sum_{i=1}^{k} \eta_i \nabla_\theta l(z, \theta^{[t_i]}).\nabla_\theta l(z_{test}, \theta^{[t_i]}) \tag{6}$$

[27] in Influenciae

Influenciae provides a simple TracIn object to allow one to replicate this method. One can discover more in the Source Code and the dedicated Tutorial.

It should be noted that this formulation is also derived for the SGD optimization process in mini-batches, but it can be adjusted to work for other optimization techniques and training processes. The main advantage of this technique relies on its simplicity (compared to [14]) while being empirically demonstrated as efficient as RPSL2 (Eq. 4) [39] or IF (Eq. 2) [18]. Nonetheless, this kind of approach requires handling the training procedure to save the different checkpoints, potentially numerous, which in practice is not always feasible.

3 API

With the ever growing size of datasets and models, the most memory-efficient way of computing influence-related quantities is possibly to take advantage of lazy and batched computations as implemented in the `tensorflow.data.Dataset` module. This is why we have built Influenciæin such a way as to operate entirely via Tensorflow Datasets. Starting from the observation that the influence score/vector of a sample has meaning only by comparing it with other samples of a dataset, all core functions of the Influenciælibrary take as an argument a dataset and return a dataset that the user can iterate over to extract the influence score or the influence vector. All methods implemented in the library inherit from the `BaseInfluenceCalculator` class which contains all the core functionality.

```
from deel.influenciae.common import InfluenceModel, ExactIHVP
from deel.influenciae.influence import FirstOrderInfluenceCalculator

# Extract the end of the model defined by the target layer
influence_model = InfluenceModel(model, target_layer, loss_function)

# Create the influence calculator from the model with the exact ihvp
ihvp_calculator = ExactIHVP(influence_model, train_dataset)
influence_calculator = FirstOrderInfluenceCalculator(influence_model,
    train_dataset, ihvp_calculator)

# Return a dataset containing the self influence score for each point
# of the training dataset
data_and_influence_dataset = influence_calculator.
    compute_influence_values(train_dataset)
```

Oftentimes, the samples associated with the highest or smallest values of influence score can reveal outliers of the distribution or mislabeled samples, thus abnormal model behaviour. In addition, one could be interested in retrieving the training samples most responsible for the prediction of a given test sample. For all these cases, the API provides a `top_k` function to retrieve the most "relevant" samples.

```
# For a given dataset to explain, this function will return a dataset
# containing the top_k closest samples of the training set
# of each sample to explain
explanation_ds = influence_calculator.top_k(samples_to_explain,
    train_ds.batch(32), k=5, order=ORDER.DESCENDING)
```

Finally, the API provides `save` and `load` arguments to avoid recomputing each time the influence's values.

```
# The argument save_top_k_ds_path allows to save the result of top_k
explanation_ds = influence_calculator.top_k(samples_to_explain,
    train_ds.batch(32), k=5, order=ORDER.DESCENDING,
    save_top_k_ds_path="./my_path/")
```

4 Conclusion

The year 2017 brought along a wave of interest in influence functions and data-centric approaches for understanding model behavior. This interest peaked a little later, but, due to their computational cost, these techniques have never really gotten the opportunity to catch up to other types of explanations in terms of popularity and use by the general public. We believe that by providing an efficient implementation in a popular auto-differentiation framework, we can facilitate their usage, and thus, hopefully reignite the initial flame of data-centric explanations.

5 Acknowledgements

This work has benefited from the AI Interdisciplinary Institute ANITI, which is funded by the French "Investing for the Future – PIA3" program under the Grant agreement ANR-19-P3IA-0004. The authors gratefully acknowledge the support of the DEEL[1] project.

[1] `https://www.deel.ai/`

References

[1] Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 2017.

[2] Walter E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 1951.

[3] Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger B Grosse. If influence functions are the answer, then what is the question? *NeurIPS*, 2022.

[4] Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. Relatif: Identifying explanatory training samples via relative influence. In *AISTATS*, 2020.

[5] S Basu, P Pope, and S Feizi. Influence functions in deep learning are fragile. In *ICLR*, 2021.

[6] Samyadeep Basu, Xuchen You, and Soheil Feizi. On second-order group influence functions for black-box predictions. In *ICML*, 2020.

[7] Julien Colin, Thomas Fel, Rémi Cadène, and Thomas Serre. What i cannot predict, i do not understand: A human-centered evaluation framework for explainability methods. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[8] R Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[10] Thomas Fel, Thibaut Boissin, Victor Boutin, Agustin Picard, Paul Novello, Julien Colin, Drew Linsley, Tom Rousseau, Rémi Cadène, Laurent Gardes, et al. Unlocking feature visualization for deeper networks with magnitude constrained optimization. *arXiv preprint arXiv:2306.06805*, 2023.

[11] Thomas Fel, Lucas Hervier, David Vigouroux, Antonin Poche, Justin Plakoo, Remi Cadene, Mathieu Chalvidal, Julien Colin, Thibaut Boissin, Louis Béthune, Agustin Picard, Claire Nicodeme, Laurent Gardes, Gregory Flandin, and Thomas Serre. Xplique: A deep learning explainability toolbox. 2022.

[12] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE international conference on computer vision*, pages 3429–3437, 2017.

[13] Frank R Hampel. The influence curve and its role in robust estimation. *JASA*, 1974.

[14] Satoshi Hara, Atsushi Nitanda, and Takanori Maehara. Data cleansing for models trained with sgd. *NeurIPS*, 2019.

[15] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. In *ICML*, 2022.

[16] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pages 2668–2677. PMLR, 2018.

[17] Janis Klaise, Arnaud Van Looveren, Giovanni Vacanti, and Alexandru Coca. Alibi explain: Algorithms for explaining machine learning models. *Journal of Machine Learning Research*, 22(181):1–7, 2021.

[18] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *NeurIPS*, 2017.

[19] Pang Wei W Koh, Kai-Siang Ang, Hubert Teo, and Percy S Liang. On the accuracy of influence functions for measuring group effects. *NeurIPS*, 2019.

[20] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for pytorch, 2020.

[21] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[22] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[23] James Martens. Deep learning via hessian-free optimization. In *ICML*, 2010.

[24] Paul Novello, Thomas Fel, and David Vigouroux. Making sense of dependence: Efficient black-box explanations using dependence measure. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[25] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[26] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

[27] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *NeurIPS*, 2020.

[28] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[29] Andrea Schioppa, Katja Filippova, Ivan Titov, and Polina Zablotskaia. Theoretical and practical perspectives on what influence functions do. *arXiv preprint arXiv:2305.16971*, 2023.

[30] Andrea Schioppa, Polina Zablotskaia, David Vilar, and Artem Sokolov. Scaling up influence functions. In *AAAI*, 2022.

[31] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *EuroCOLT*, 2001.

[32] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[33] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *In Workshop at International Conference on Learning Representations*. Citeseer, 2014.

[34] 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.

[35] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smooth-grad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[36] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[37] Yi Sui, Ga Wu, and Scott Sanner. Representer point selection via local jacobian expansion for post-hoc classifier explanation of deep neural networks and ensemble models. In *NeurIPS*, 2021.

[38] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[39] Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. *NeurIPS*, 2018.

[40] Ruihan Zhang, Prashan Madumal, Tim Miller, Krista A Ehinger, and Benjamin IP Rubinstein. Invertible concept-based explanations for cnn models with non-negative concept activation vectors. *arXiv preprint arXiv:2006.15417*, 2020.

[41] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.