Dattri: A Library for Efficient Data Attribution

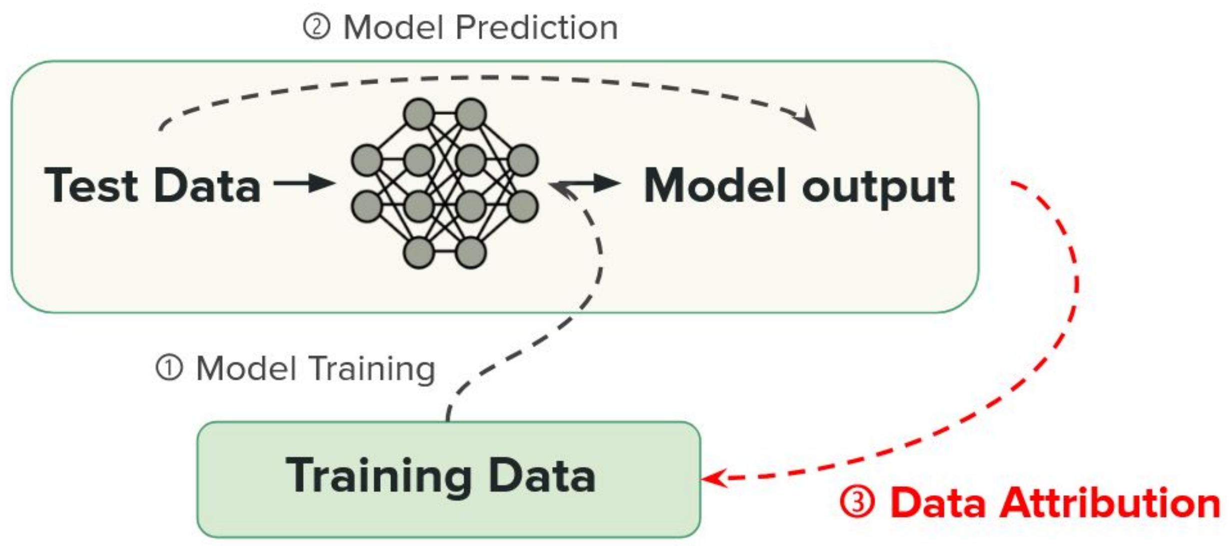
Junwei Deng*[1], Ting-Wei Li*[1], Shiyuan Zhang[1], Shixuan Liu[2], Yijun Pan[2], Hao Huang[1], Xinhe Wang[2], Pingbang Hu[1], Xingjian Zhang[2], Jiaqi W. Ma[1]

[1]University of Illinois Urbana-Champaign [2]University of Michigan-Ann Arbor

ILLINOIS · UNIVERSITY OF MICHIGAN · NEURAL INFORMATION PROCESSING SYSTEMS

Introduction

Data Attribution



Data attribution quantitatively estimates the influence of each training sample on the model output.

Efficient Data Attribution Methods

Prioritizing efficient data attribution methods applicable to large neural network models, we have implemented most of the state-of-the-art efficient data attribution methods.

Family	Algorithms	LOO		LDS		AUC	
		Linear	Non-linear	Linear	Non-linear	Linear	Non-linear
IF	Explicit [22]	++	-	++	-	++	-
	CG [26]	++	-	++	+	++	+
	LiSSA [1]	++	-	++	+	++	+
	Arnoldi [30]	+	-	+	-	++	+
TracIn	TracInCP [29]	+	-	+	+	++	+
	Grad-Dot [5]	+	-	+	+	++	+
	Grad-Cos [5]	+	-	+	+	-	-
RPS	RPS-L2 [36]	+	-	+	-	++	+
TRAK	TRAK [27]	++	-	++	++	++	++

A summary of existing libraries and our library dattri

Motivation

- **Minimal code invasion**
  - Many existing implementations of data attribution methods are heavily invasive to the model training pipeline, i.e., the data attribution process is significantly entangled with the model training code, making it challenging for users to adapt the code to other models or applications.
- **Low-level utility functions**
  - Different data attribution methods can share common sub-routines in their algorithms. These sub-routines can be reused by developers to develop new methods
- **Comprehensive benchmark suite**
  - Evaluation metrics, pre-trained model checkpoints and ground truths and reference benchmark results.

Design Highlights

HIGHLIGHT 1: A unified and user-friendly API

`dattri` is carefully designed to provide a unified API that can be applied to the most common PyTorch model training pipeline with minimal code invasion.

User prepare

Train loader · Test loader · Model ckpts · Traget func · Loss func

```
dataset_train, dataset_test = create_mnist_dataset()
train_loader = torch.utils.data.DataLoader(dataset_train)
test_loader = torch.utils.data.DataLoader(dataset_test)

model = train_mnist_lr(train_loader)

def loss_func(params, data):
    x, y = data
    loss = nn.CrossEntropyLoss()
    yhat = torch.func.functional_call(model, params, x)
    return loss(yhat, y)
```

①: target and loss func could be the same
②: dattri is based on torch.func, so model(...) needs to be replaced as functional_call

Dattri API

AttributionTask · Attributor

```
task = AttributionTask(loss_func=loss_func,
                       model=model,
                       checkpoints=model.state_dict())

attributor = IFAttributorCG(task=task)

attributor.cache(train_loader)
score = attributor.attribute(train_loader, test_loader)
```

A Demo showing an example of applying IF methods on a PyTorch model.

HIGHLIGHT 2: Modularized low-level utility functions

Different data attribution methods can share common sub-routines in their algorithms. In `dattri`, we modularize such sub-routines through low-level utility functions so that they can be reused in the development of new methods.

- Hessian, HVP, and IHVP
- Fisher Information Matrix (FIM) / IFVP
- Random projection
- Dropout ensemble

```
from dattri.func.hessian import ihvp_cg, ihvp_at_x_cg

def f(x, param):
    return torch.sin(x / param).sum()

x = torch.randn(2)
param = torch.randn(1)
v = torch.randn(5, 2)

ihvp_func = ihvp_cg(f, argnums=0, max_iter=2)
ihvp_result_1 = ihvp_func((x, param), v)

ihvp_at_x_func = ihvp_at_x_cg(f, x, param, argnums=0, max_iter=2)
ihvp_result_2 = ihvp_at_x_func(v)

assert torch.allclose(ihvp_result_1, ihvp_result_2)
```

Example usage of the CG implementation of the IHVP function.

HIGHLIGHT 3: A comprehensive benchmark suite

- The first type of metrics treats the change of model outputs after removing certain data points and retraining the model as a gold standard
- The second type of metrics evaluates data attribution methods through downstream applications, where the most common ones are noisy label detection and data selection

Retraining (if needed)

```
from dattri.model_util.retrain import retrain_lds, retrain_loo

def train_mnist_lr(train_loader, **kwargs):
    ...
    return model

retrain_loo(train_mnist_lr, train_loader,
            path="./mnist_lr_loo", **kwargs)
retrain_lds(train_mnist_lr, train_loader,
            path="./mnist_lr_lds", **kwargs)
```

Groundtruth calculation

```
from dattri.metric import calculate_lds_ground_truth, calculate_loo_ground_truth

gt_loo = calculate_loo_ground_truth(target_func, "./mnist_lr_loo", test_loader)
gt_lds = calculate_lds_ground_truth(target_func, "./mnist_lr_lds", test_loader)
```

Evaluate the metrics

```
from dattri.metric import loo_corr, lds

loo_result = loo_corr(score, gt_loo)
lds_result = lds(score, gt_lds)
auc_result = mislabel_detection_auc(score, noise_index)
```

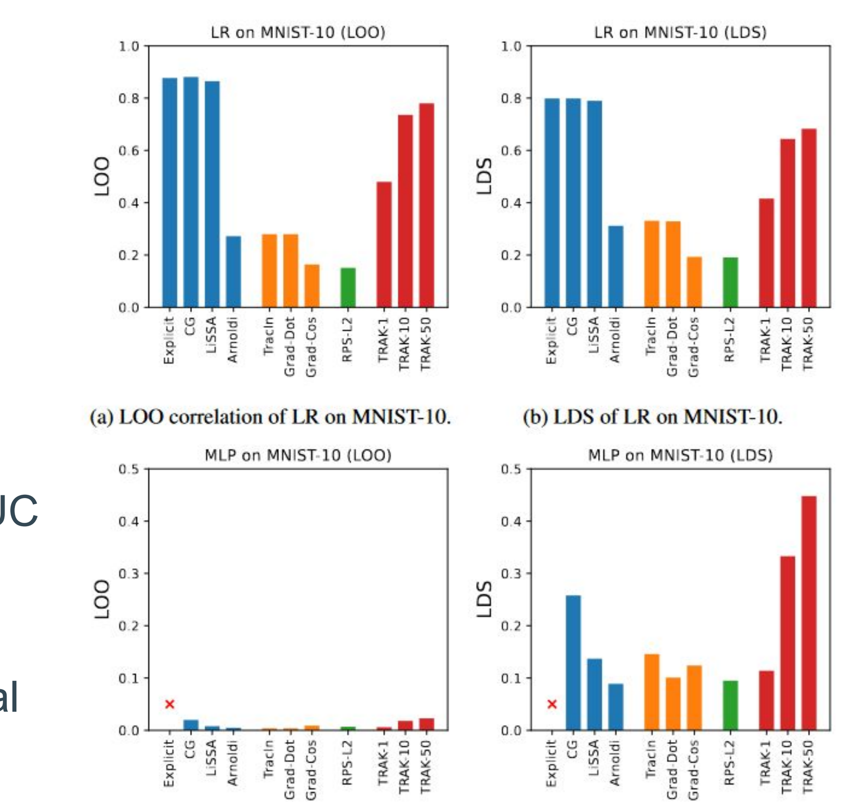
Example usage of the evaluation metrics and their util functions.

Benchmark Results

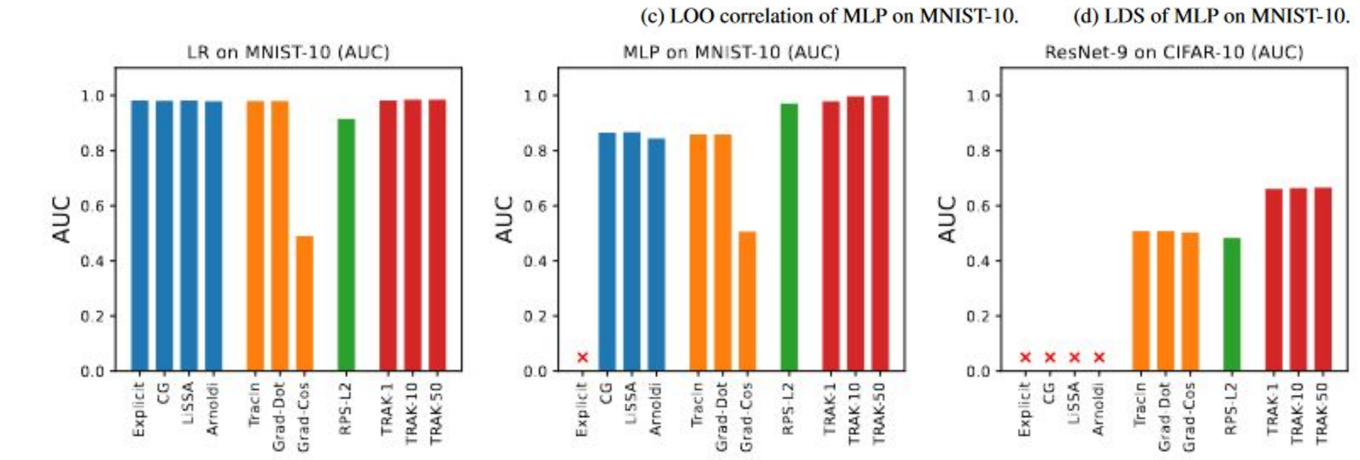
Table 3: The full experimental setting for data attribution benchmark.

Dataset	Model	Task	Sample size (train,test)	Parameter size	Metrics	Data Source
MNIST-10	LR	Image Classification	(5000,500)	7840	LOO/LDS/AUC	[8]
MNIST-10	MLP	Image Classification	(5000,500)	0.11M	LOO/LDS/AUC	[8]
CIFAR-2	ResNet-9	Image Classification	(5000,500)	4.83M	LDS	[24]
CIFAR-10	ResNet-9	Image Classification	(5000,500)	4.83M	AUC	[24]
MAESTRO	Music Transformer	Music Generation	(5000,178)	13.3M	LDS	[12]
Shakespeare	NanoGPT	Text Generation	(3921,435)	10.7M	LDS	[20]

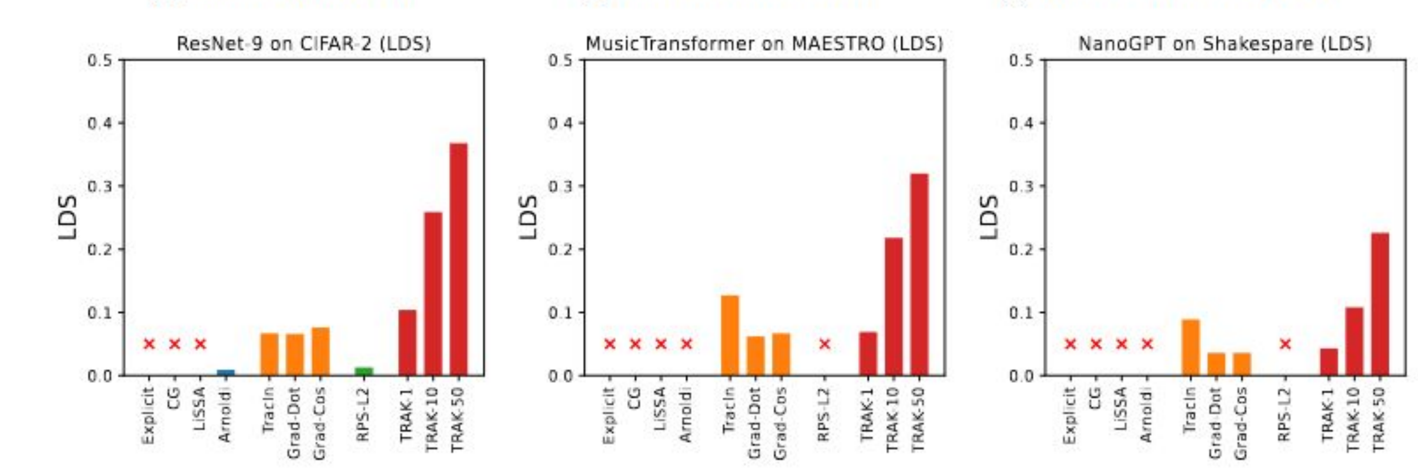
- IF performs well on Linear model
- TRAK performs generally well on both LR and MLP
- LOO is not a good evaluation metric when the model gets non-linear.
- Most algorithms do well on AUC noisy label detection to MNIST-10.
- Only TRAK could get non-trvial result on complex models



(a) LOO correlation of LR on MNIST-10. (b) LDS of LR on MNIST-10. (c) LOO correlation of MLP on MNIST-10. (d) LDS of MLP on MNIST-10.



(a) LR on MNIST-10. (b) MLP on MNIST-10. (c) ResNet-9 on CIFAR-10.



(a) ResNet-9 on CIFAR-2. (b) Music Transformer on MAESTRO. (c) NanoGPT on Shakespeare.

Installation & Github

- `dattri` has been released on PyPI

```
pip install dattri
```

- More than 10 examples and benchmark scripts in Github Repos
- More methods / benchmark settings / examples to come
- **Any feedbacks from users are important!**

Stars