Data Attribution

REFORM reading group

John Cherian

Overview

- Divides data attribution into three categories

1. **Corroborative** ⤳ e.g., citation generation
2. **Game-theoretic** ⤳ e.g., Data Shapley
3. **Predictive** ⤳ e.g., influence functions, datamodeling

Overview

- Divides data attribution into three categories

1. ~~**Corroborative** $\rightsquigarrow$ e.g., citation generation~~
2. **Game-theoretic** $\rightsquigarrow$ e.g., Data Shapley
3. **Predictive** $\rightsquigarrow$ e.g., influence functions, datamodeling

Data Shapley
[GZ19, JDW+19]

- Given some performance score $V(\cdot)$ (e.g., test accuracy), want data attribution ϕ_i satisfying the following properties

  1. If $V(S) = V(S \cup \{i\})$ for all subsets S, then $\phi_i = 0$
  2. If $V(S \cup \{i\}) = V(S \cup \{j\})$ for all i, j then $\phi_i = \phi_j$
  3. If $V(\cdot) = V_1(\cdot) + V_2(\cdot)$, then $\phi_i^V = \phi_i^{V_1} + \phi_i^{V_2}$

Characterization

ϕ_i must be of the form:

$$\phi_i = C \cdot \sum_{S \subseteq D - \{i\}} \frac{V(S \cup \{i\}) - V(S)}{\binom{n-1}{|S|}}$$

Predictive attribution

- Non-axiomatic approach $\rightsquigarrow$ how does fitting to point i affect the prediction?

- **Leave-one-out approach**: how does the fit change if we drop point i from the model?

- **Data-modeling**: can we fit a predictive model for data $\mapsto$ prediction

LOO / influence function

$\hat{\theta}_{-j}$ = model parameters if we remove the j-th data point

Sometimes this is easy to compute:

OLS:

$$\hat{\theta} - \hat{\theta}_{-j} = \frac{(x_j^\top \hat{\theta} - y_j)(\sum_{i=1}^n x_i x_i^\top)^{-1} x_j}{1 - x_j^\top (\sum_{i=1}^n x_i x_i^\top)^{-1} x_j}$$

LOO / influence function

Not OLS?

- For *generalized* linear models $\{\sigma(\theta^\top x) \mid \theta \in \mathbb{R}^d\}$, we can take a Newton step on the leave-one-out loss

$$\hat{\theta}_{-j} \approx \hat{\theta} - H^{-1}_{\hat{\theta},-j} \nabla_\theta \mathscr{L}_{-j}(\hat{\theta}) = \frac{H^{-1}_{\hat{\theta}} \cdot (\mathscr{L}'_j(\hat{\theta}^\top x_j) \cdot x_j)}{1 - \mathscr{L}''_j(\hat{\theta}^\top x_j) \cdot x_j^\top H^{-1}_{\hat{\theta}} x_j}$$

- Possible because the inverse Hessian for the leave-one-out loss can be updated efficiently via Sherman-Morrison-Woodbury

Aside: Sherman-Morrison-Woodbury

- If we have the inverse of some matrix H, it is very easy to compute the inverse of H + a low-rank update:

$$(H + uv^\top)^{-1} = H^{-1} - \frac{H^{-1}uv^\top H^{-1}}{1 + v^\top H^{-1}u}$$

No $O(n^3)$ matrix inversion required!

Influence functions

Beyond (G)LMs

- When our model class is more flexible (e.g., NNs), we cannot SMW our way to success

- Previous approach relies on second-order Taylor expansion of *leave-one-out-loss* around $\hat{\theta}$

- New approach: second-order Taylor expansion of *full loss* around $\hat{\theta}$

Influence functions
(cont.)

Write $\mathscr{L}(\hat{\theta}) = \sum_{i=1}^{n} w_i \ell(\hat{\theta}; x_i, y_i)$

Using the second-order Taylor expansion of $\mathscr{L}(\hat{\theta})$ around $\hat{\theta}$, we compute $\frac{\partial \hat{\theta}}{\partial w_j}$

$$\hat{\theta}_{-j} \approx \hat{\theta} - \frac{\partial \hat{\theta}}{\partial w_j} = \hat{\theta} + H_{\hat{\theta}}^{-1} \ell_j'(\hat{\theta}; x_j, y_j)$$

Commentary

Comparing the two approaches

1. Influence function extrapolates from local *perturbations* of the full loss quadratic approximation

2. Approx. LOO runs a Newton step on the leave-one-out quadratic approximation

- Both approaches require a leap of faith (maybe formally, some form of leave-one-out stability?)

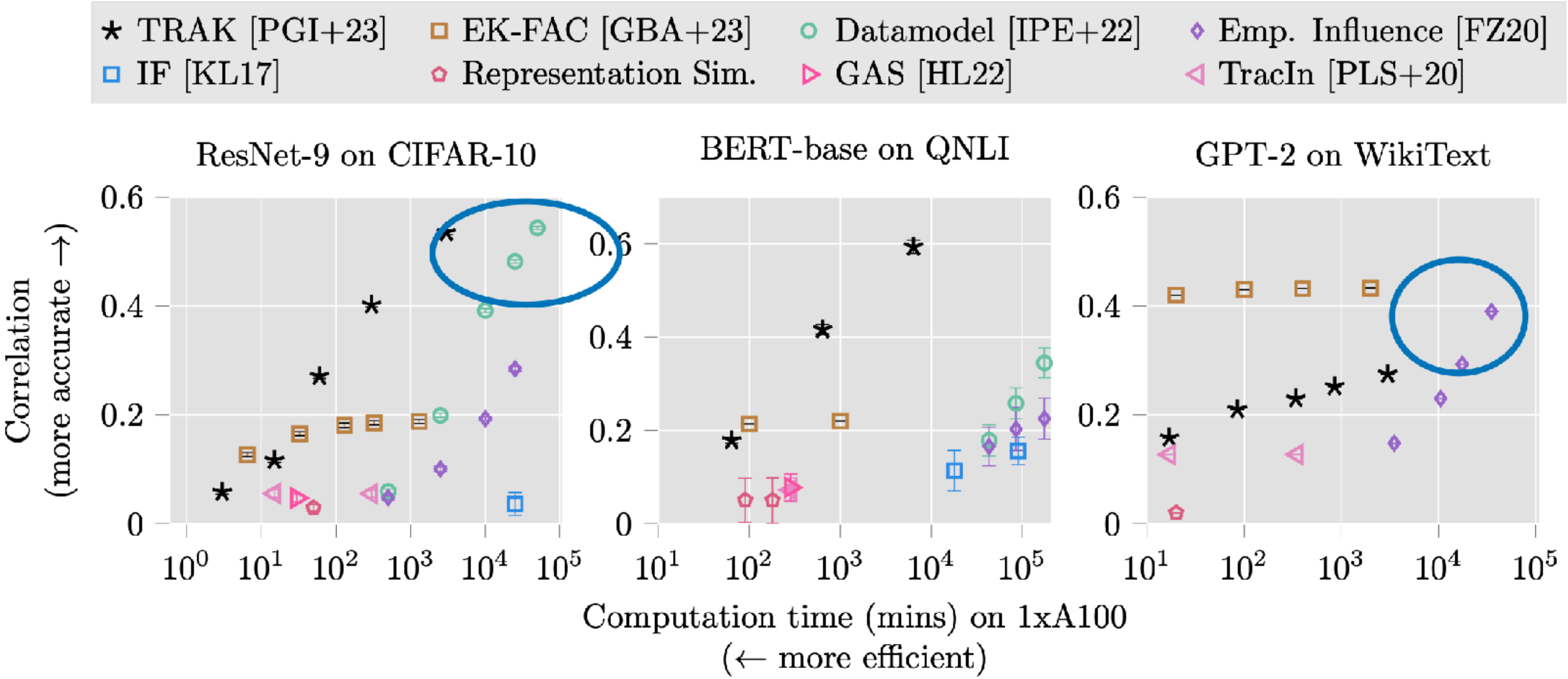
Influence functions

In practice

- Smart Hessian approximations (e.g., via structural approximation, Gauss-Newton-Hessian approx.)
- (Approximately) unrolling gradient descent
- Replacing the NN with a surrogate model (e.g., TRAK)

- But there’s no really clear picture of what is best…

Evaluating the landscape



Data modeling

DsDm: Model-aware data selection with Datamodels

$$S^* = \text{argmin}_{S \subset \mathcal{S}, |S|=k} \mathcal{L}_{\mathcal{D}_{targ}}(S)$$

$$\text{where } \mathcal{L}_{\mathcal{D}}(S) := \mathbb{E}_{\mathcal{D}}[\ell(x; \mathcal{A}(S))]$$

- How do we select a training set of size k that ensures good performance on a target population?

- Key idea: build ***datamodel*** that maps dataset composition to target loss

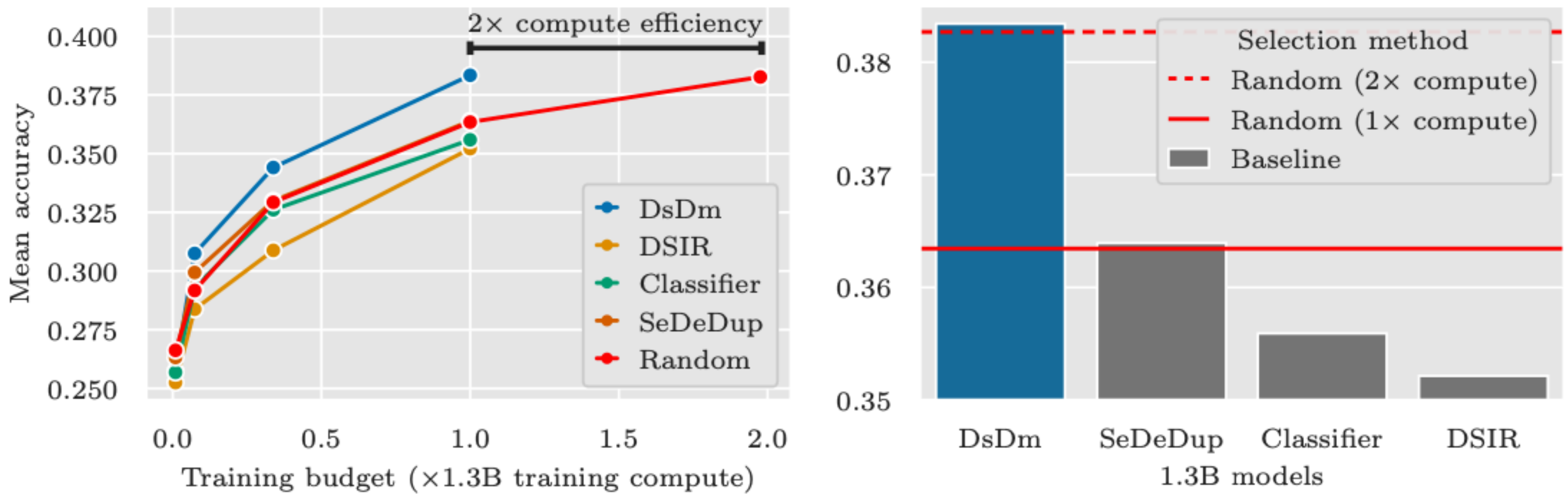
Datamodels are linear

$$\hat{\mathscr{L}}_{target}(S) = \theta_x^\top \mathbf{1}_S$$

each data point has a separable and additive effect on the final loss

- Select the k data points corresponding to the smallest values of θ_x
- Success?

Results


2× compute efficiency
0.400
0.375
0.350
0.325
0.300
0.275
0.250
Mean accuracy
DsDm
DSIR
Classifier
SeDeDup
Random
0.0
0.5
1.0
1.5
2.0
Training budget (×1.3B training compute)
0.38
0.37
0.36
0.35
Selection method
Random (2× compute)
Random (1× compute)
Baseline
DsDm
SeDeDup
Classifier
DSIR
1.3B models

But how do datamodels work?

Original approach (Ilyas et al. 2022)

- Collect a large "meta"-dataset consisting of resampled training sets and models fit to those training sets

1. Sample a new training data set S'

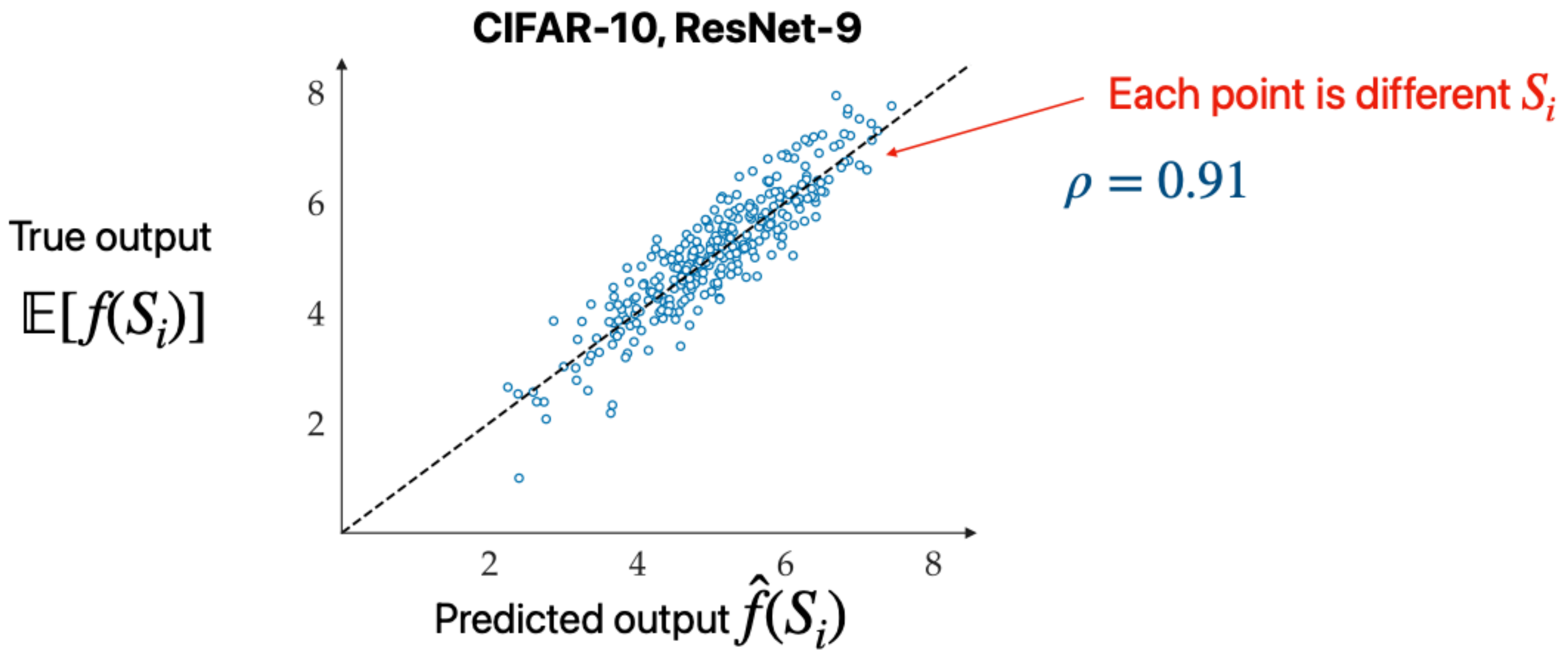
2. Fit model by running $\mathscr{A}(S')$

3. Estimate test accuracy of fitted model: $\mathscr{L}_{target}(S')$

- Fit a **linear** model on $\mathbf{1}_S$ that predicts the test accuracy of each fitted model

Data regression works!

Sample **new** random *subsets* S_i, compare predictions and ground-truth



Practical implementation

TRAK

- Refitting the model many times is completely infeasible
- Idea: replace $\mathscr{A}(S)$ with some simpler algorithm $\mathscr{A}'(S)$ that we can easily recompute for changes to the sample
- What sorts of algorithms are easy to recompute?

Recomputing $\mathscr{A}'(S)$

- Paper overloads the term “influence function” - here it refers to the approx. LOO approach:

$$\tau_\theta(S) = \mathrm{IF}(z)^\top \mathbb{1}_S + f(z; \mathcal{A}_{\mathrm{Log}}(\mathcal{S})) - \sum_{k=1}^{n} \mathrm{IF}(z)_k$$

IF(z) arises from performing a Newton step from logistic model parameters for S to minimize loss on S \ z_i.

Where do we get a logistic regression from?

- First, they linearize the predictor

$$\hat{f}(z;\theta) = f(z;\theta^*) + \nabla_\theta f(z;\theta^*)^\top (\theta - \theta^*).$$

- Second, they plug that into a logistic loss function

$$\mathcal{A}'(S) = \arg\min_{\theta} \sum_{z_i \in S} \log\left(1 + \exp\left(-y_i \cdot \left(\theta^\top \nabla_\theta f(z_i;\theta^*) + f(z_i;\theta^*) - \nabla_\theta f(z_i;\theta^*)^\top \theta^*\right)\right)\right).$$

TRAK subtleties

- It's still completely impractical to compute the influence function

$$\mathrm{IF}(z)_i := \frac{x^\top (X^\top R X)^{-1} x_i}{1 - x_i^\top (X^\top R X)^{-1} \cdot p_i^*(1-p_i^*)}(1-p_i^*)$$

- Random projection of gradient reduces the dimensionality of the matrix inverse while preserving inner products (J-L)

Commentary

- There are *a lot* of approximations inside of the datamodeling application here

1. Linear data model
2. Influence functions in place of data regression
3. Kernel approximation to neural network
4. Various term ablations + random projections of feature vectors