Lecture 22: Explainability

Trustworthy Machine Learning

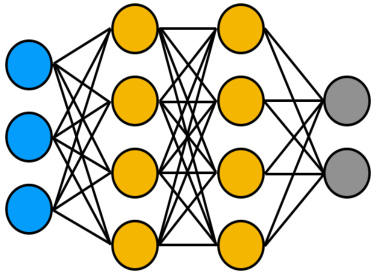
Spring 2024

Explainability

- Recap: Feature Attribution Methods
  - LIME (Local Interpretable Model-agnostic Explanations) algorithm
  - SHAP methods based on cooperative game theory
  - Saliency Maps (different versions)
  - Formal guarantees for feature attribution methods
  - Counterfactuals
  - Representation-based explanations

- Today's agenda: Data attribution methods

Dat Attribution



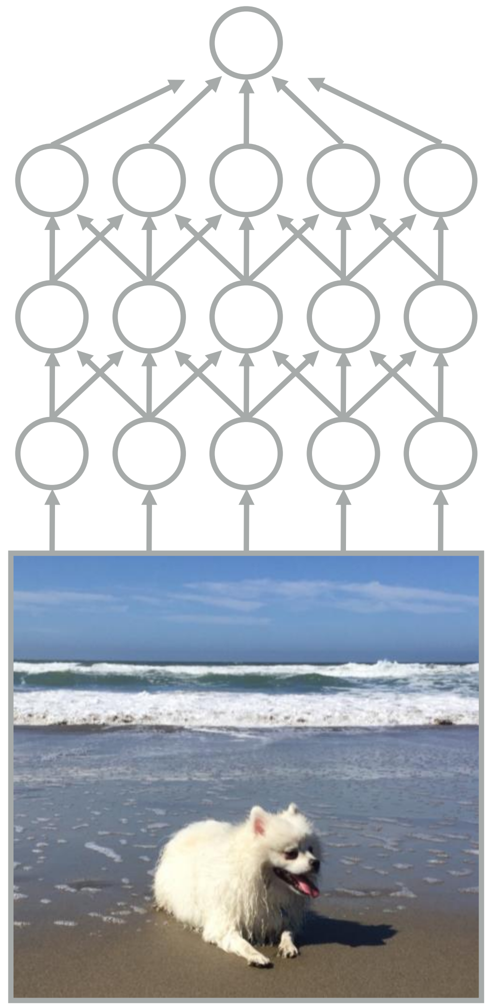
Malignant

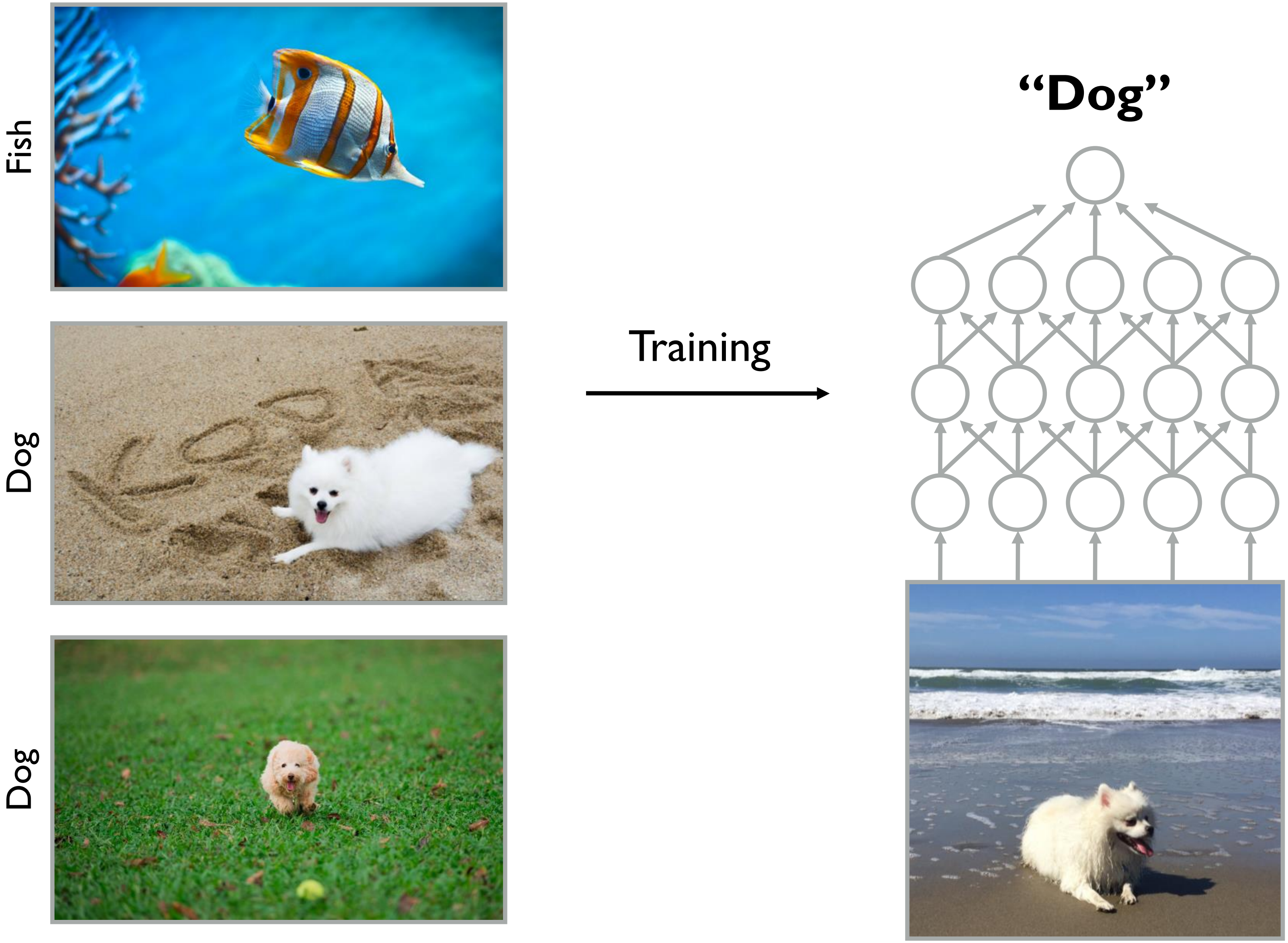
Understand how the choice of training data influences the model's prediction

Agenda

- Today:
  - Influence Functions
  - Datamodels
- Resources:
  - Understanding black-box predictions via influence functions; Koh et al.; ICML 2017
  - Datamodels: Predicting predictions from training data; Ilyas et al.; ICML 2022
  - TRAK: Attributing model behavior at scale; Park et al.; ICML 2023
  - Credit: Talk slides for above papers by the authors

“Dog”





Training data

Why did the model make this prediction?

Why did the model make this prediction?

Which training points were most responsible for this prediction?

The influence of individual training points

Koh & Liang, Understanding Black-box Predictions via Influence Functions, *ICML* 2017

Fish

Dog

Dog

Training data $z_1, z_2, \ldots, z_n$

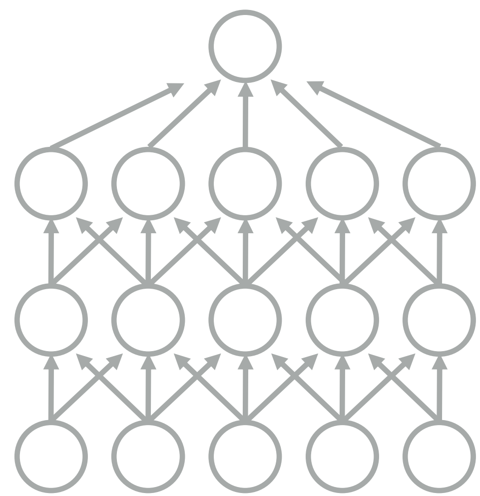
Fish

Dog

Dog

Training data $z_1, z_2, \ldots, z_n$

"Dog"



$\hat{\theta}$

Pick $\hat{\theta}$ to minimize $\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$

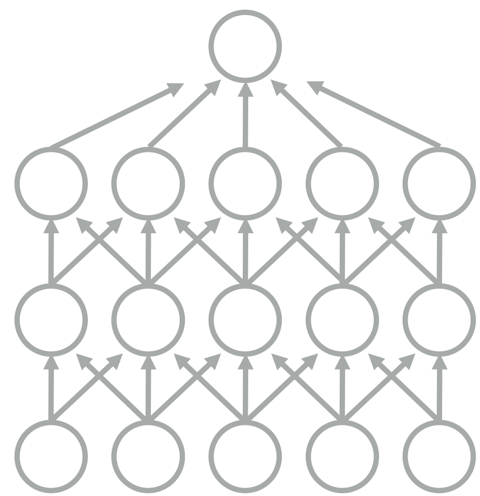
Fish

Dog

Dog

Training data $z_1, z_2, \ldots, z_n$

"Dog"



$\hat{\theta}$

Fish

Dog

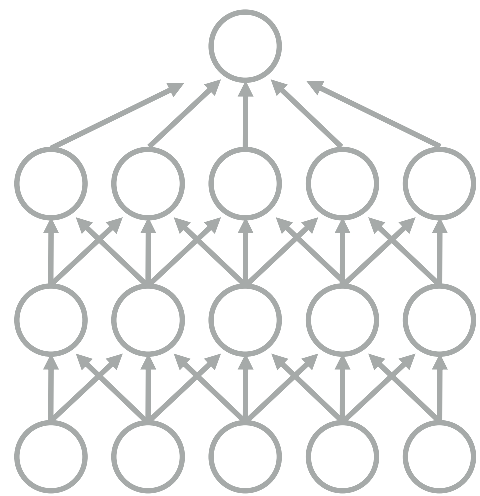
z_{train}

Dog

Training data $z_1, z_2, \dots, z_n$

Pick $\hat{\theta}$ to minimize $\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$

“Dog”



$\hat{\theta}$

Pick $\hat{\theta}$ to minimize $\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$

Fish

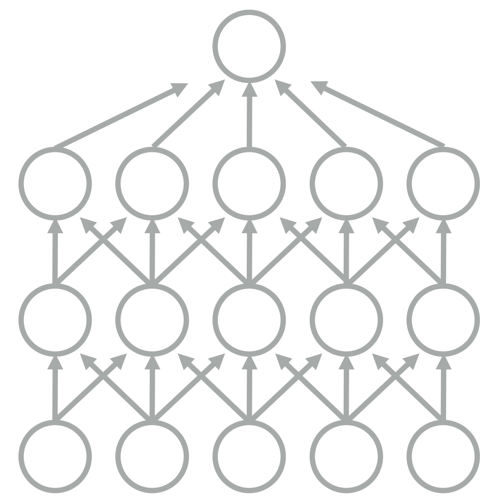
Dog

z_{train}

Dog

Training data $z_1, z_2, \ldots, z_n$

"Dog"



$\hat{\theta}$

Fish

Dog

z_{train}

Dog

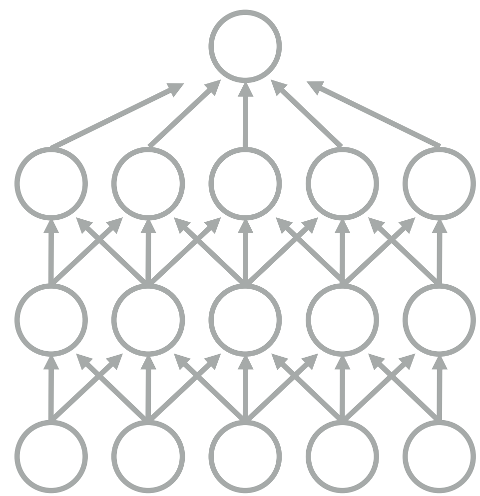
Training data $z_1, z_2, \ldots, z_n$

Pick $\hat{\theta}$ to minimize $\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$

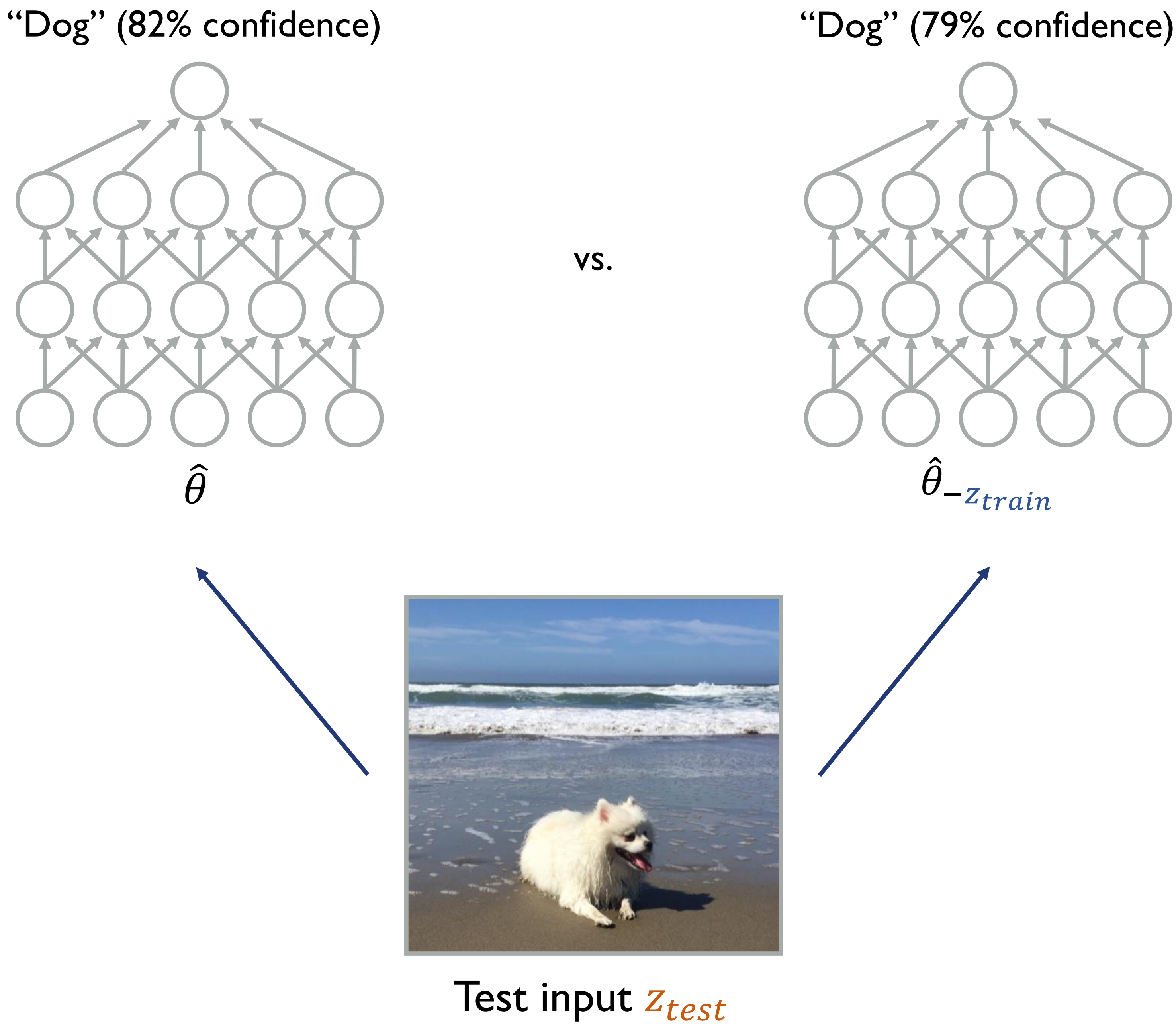
Pick $\hat{\theta}_{-z_{train}}$ to minimize

$$\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta) - \frac{1}{n} L(z_{train}, \theta)$$

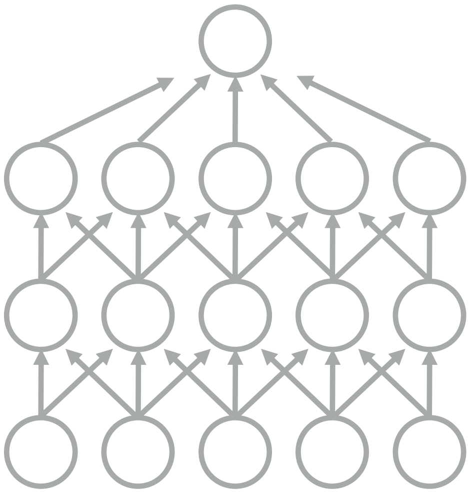
"Dog"



$\hat{\theta}_{-z_{train}}$


“Dog” (82% confidence)
“Dog” (79% confidence)
vs.
$\hat{\theta}$
$\hat{\theta}_{-z_{train}}$
Test input z_{test}

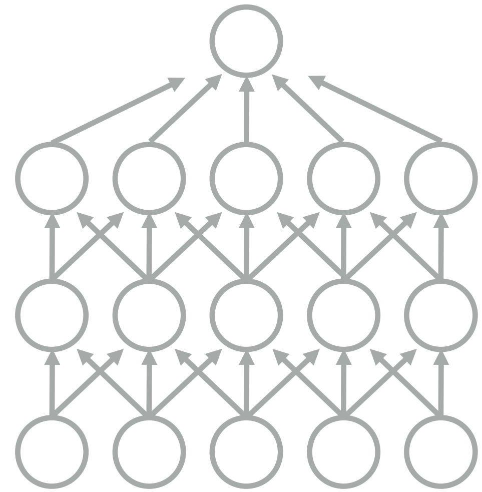
"Dog" (82% confidence)



$\hat{\theta}$

vs.

"Dog" (79% confidence)



$\hat{\theta}_{-z_{train}}$

What is $L\left(z_{test}, \hat{\theta}_{-z_{train}}\right) - L\left(z_{test}, \hat{\theta}\right)$?

Why did the model make this prediction?

↓

Which training points were most responsible for this prediction?

↓

How would the prediction change if we removed a training point?

Problem

Repeatedly removing a training point and retraining the model is too slow

Problem	Repeatedly removing a training point and retraining the model is too slow
Solution	Approximation via influence functions (a classical technique from the 1970s)

Influence functions

- Goal: Compute $L\left(z_{test}, \hat{\theta}_{-z_{train}}\right) - L\left(z_{test}, \hat{\theta}\right)$

$$\hat{\theta}_{-z_{train}} \stackrel{\text{def}}{=} \text{argmin}_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) - \frac{1}{n} L(z_{train}, \theta)$$

Influence functions

- Goal: Compute $L\left(z_{test}, \hat{\theta}_{-z_{train}}\right) - L\left(z_{test}, \hat{\theta}\right)$

$$\hat{\theta}_{-z_{train}} \stackrel{\text{def}}{=} \text{argmin}_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) - \frac{1}{n} L(z_{train}, \theta)$$

- Equivalent to removing $\frac{1}{n}$ weight from z_{train} in the empirical distribution, then renormalizing

Influence functions

- Goal: Compute $L\left(z_{test}, \hat{\theta}_{-z_{train}}\right) - L\left(z_{test}, \hat{\theta}\right)$

$$\hat{\theta}_{-z_{train}} \stackrel{\text{def}}{=} \text{argmin}_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) - \frac{1}{n} L(z_{train}, \theta)$$

- Idea:
  - Assume $\frac{1}{n}$ is small
  - Use calculus to compute effect of removing ϵ weight from z_{train}
  - Linearly extrapolate to removing $\frac{1}{n}$ weight

Influence functions

- Goal: Compute $L\left(z_{test}, \hat{\theta}_{-z_{train}}\right) - L\left(z_{test}, \hat{\theta}\right)$

$$\hat{\theta}_{-z_{train}} \stackrel{\text{def}}{=} \operatorname{argmin}_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta) - \frac{1}{n} L(z_{train}, \theta)$$

- Specifically, compute gradient of

$$\hat{\theta}_{\epsilon, z_{train}} \stackrel{\text{def}}{=} \operatorname{argmin}_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z_{train}, \theta)$$

w.r.t. ϵ.

Influence functions

- $\hat{\theta}_{\epsilon, z_{train}} \stackrel{\text{def}}{=} \text{argmin}_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z_{train}, \theta)$
- Under smoothness assumptions,

$$I_{up,loss}(z_{train}, z_{test}) \stackrel{\text{def}}{=} \left. \frac{dL\left(z_{test}, \hat{\theta}_{\epsilon, z_{train}}\right)}{d\epsilon} \right|_{\epsilon=0}$$

Influence functions

- $\hat{\theta}_{\epsilon,z_{train}} \stackrel{\text{def}}{=} \operatorname{argmin}_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L(z_{train}, \theta)$
- Under smoothness assumptions,

$$I_{up,loss}(z_{train}, z_{test}) \stackrel{\text{def}}{=} \left. \frac{dL\left(z_{test}, \hat{\theta}_{\epsilon,z_{train}}\right)}{d\epsilon} \right|_{\epsilon=0}$$

$$= -\nabla_{\theta} L\left(z_{test}, \hat{\theta}\right)^{\top} H_{\hat{\theta}}^{-1} \nabla_{\theta} L\left(z_{train}, \hat{\theta}\right)^{\top}$$

where $H_{\hat{\theta}} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta}^{2} L\left(z_i, \hat{\theta}\right)$.

Influence functions

- $\hat{\theta}_{\epsilon,z_{train}} \stackrel{\text{def}}{=} \text{argmin}_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta) + \epsilon L(z_{train},\theta)$
- Under smoothness assumptions,

$$I_{up,loss}(z_{train}, z_{test}) \stackrel{\text{def}}{=} \left.\frac{dL\left(z_{test}, \hat{\theta}_{\epsilon,z_{train}}\right)}{d\epsilon}\right|_{\epsilon=0}$$

$$= -\nabla_\theta L\left(z_{test}, \hat{\theta}\right)^\top H_{\hat{\theta}}^{-1} \nabla_\theta L\left(z_{train}, \hat{\theta}\right)^\top$$

where $H_{\hat{\theta}} \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L\left(z_i, \hat{\theta}\right)$.

- $L\left(z_{test}, \hat{\theta}_{-z_{train}}\right) - L\left(z_{test}, \hat{\theta}\right) = -\frac{1}{n} I_{up,loss}(z_{train}, z_{test})$

Debugging Models with Influence Functions

- Task: Image Classification
- Model 1: Support Vector Machine (SVM) with Radical Basis Function (RBF) kernel
- Model 2: Inception v3 network from CNN family
- Training dataset: ImageNet

Debugging Models with Influence Functions

- Task: Image Classification
- Model 1: Support Vector Machine (SVM) with Radical Basis Function (RBF) kernel
- Model 2: Inception v3 network from CNN family
- Training dataset: ImageNet
- Sample correct prediction by both models: Fish

- Question: Which training images were most influential in the model's prediction?

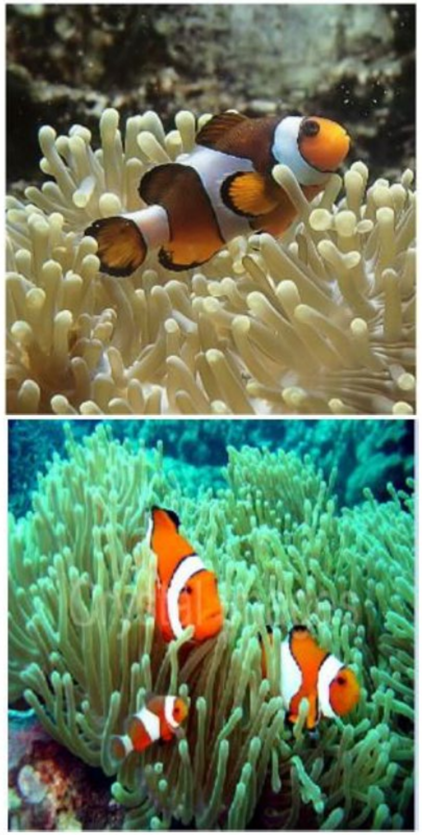
Debugging Models with Influence Functions

RBF SVM

Debugging Models with Influence Functions

RBF SVM

Inception



Applications of Influence Functions

- Understanding model predictions
- Adversarial training examples
- Debugging domain mismatch (i.e. distribution shift in test-data vs. training-data)
- Fixing mislabeled examples

The influence of groups of training points

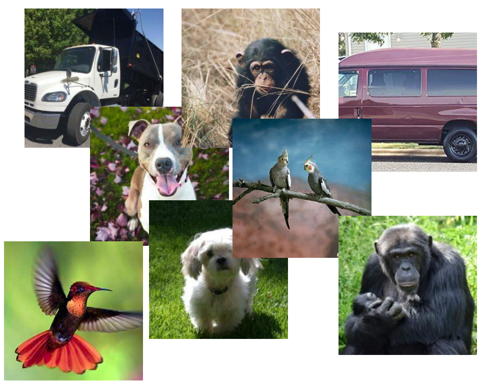
Koh*, Ang*, Teo*, & Liang, On the Accuracy of Influence Functions for Measuring Group Effects [under review]

Datamodels

- Datamodels: Predicting predictions from training data; Ilyas et al.; ICML 2022
- TRAK: Attributing model behavior at scale; Park et al.; ICML 2023

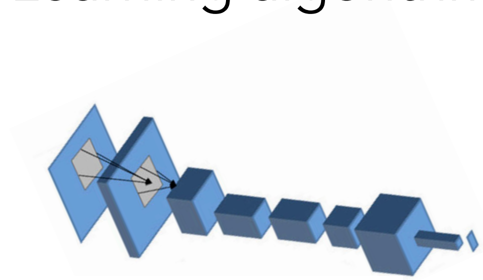
Anatomy of an ML Prediction

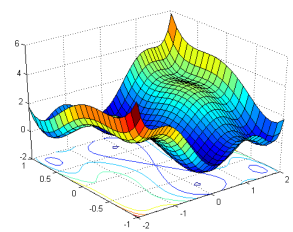
Training set S



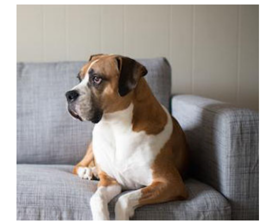
+

Learning algorithm





Test input x



→ **"dog" (85%)**

Model output

Question: How do training data and learning algorithms combine to yield model outputs?

Datamodels: Data-to-Output Modeling

What we are trying to compute (model output function):

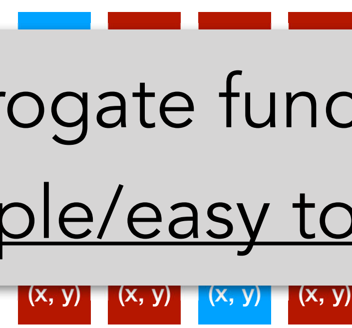
Output of interest on x
(think: margin of correct class)
after training on S'

$$f(x, S') \approx \hat{f}(x, S')$$

Datamodel for x

Datamodeling framework: Find a surrogate function $\hat{f}$ that <u>approximates</u> f, while also being <u>simple/easy to analyze</u>

Specific input x



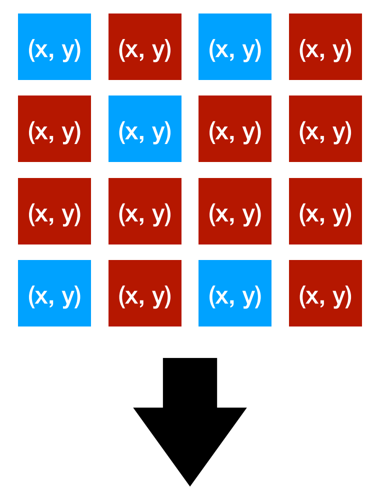
<u>Subset</u> S' of the training set S

Model Choice: Linear

$$\hat{f}(x, S') = \theta_x^{\top} \mathbf{1}_{S'}$$

Learned parameter: vector of weights (one weight per training example in S)

Indicator vector of S'

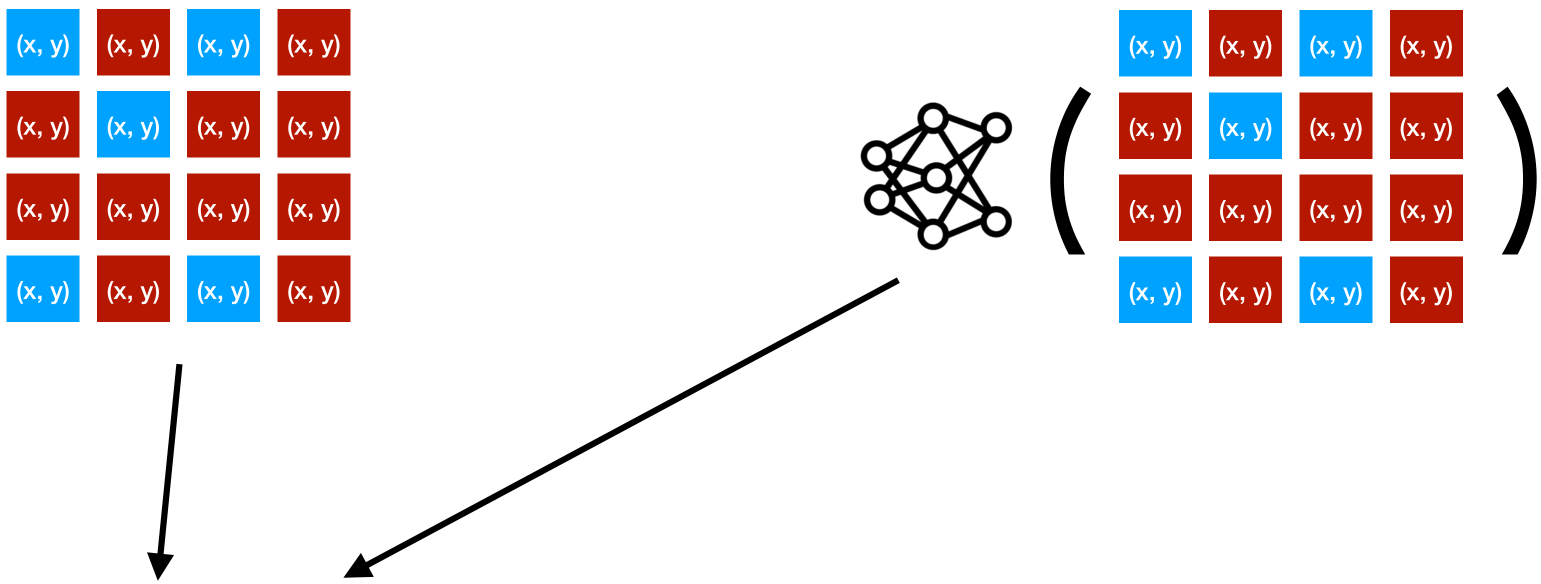


[1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0]

Remaining question: how do we fit the parameters θ_x?

How to fit a datamodel

Use supervised learning:

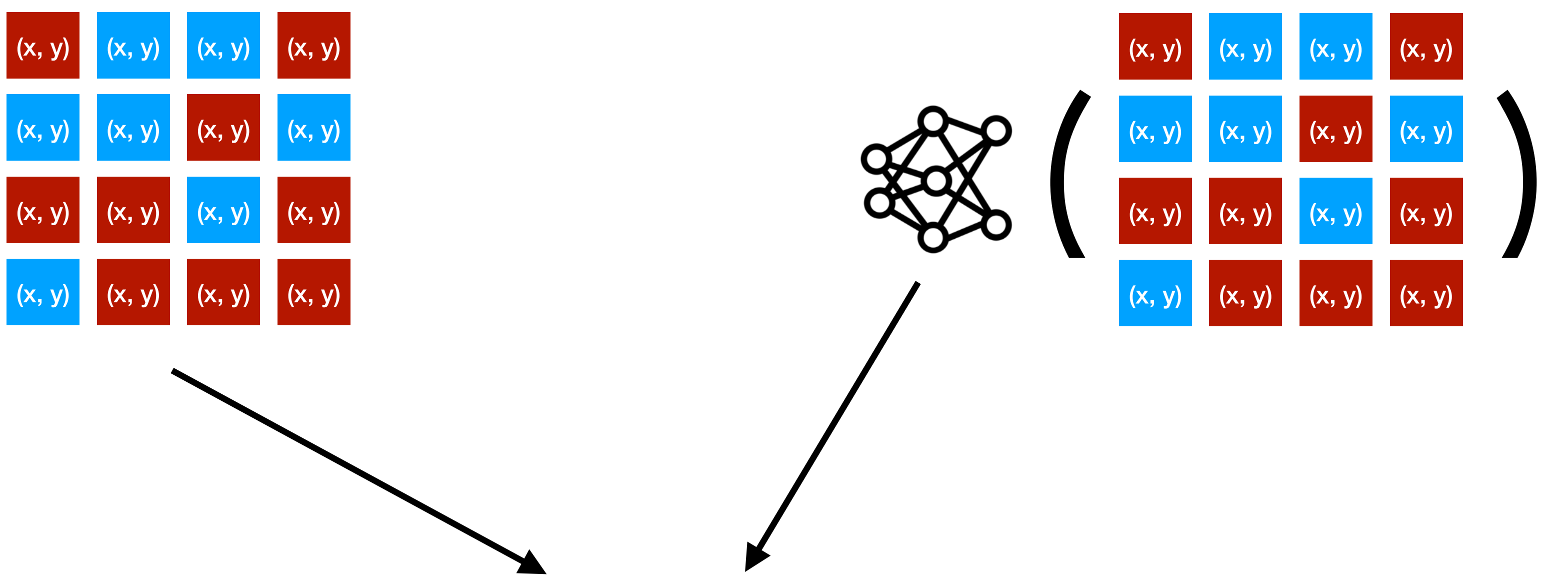


$$(S_1, f(x, S_1))$$

Fix a specific target example x

How to fit a datamodel

Use supervised learning:

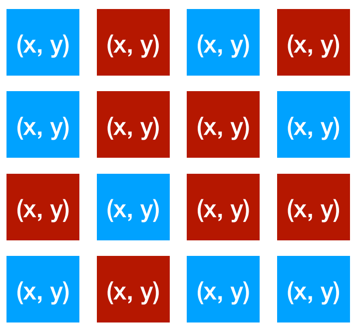


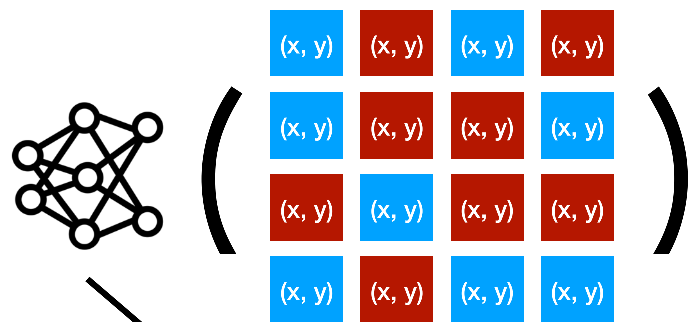
$$(S_1, f(x, S_1)), (S_2, f(x, S_2))$$

Fix a specific target example x

How to fit a datamodel

Use supervised learning**:**





$$\{(S_1, f(x, S_1)), (S_2, f(x, S_2)), \ldots, (S_m, f(x, S_m))\}$$

Then: Fit the linear model to this data

Fitting a datamodel

(for a **specific** target example x)

$$\{(S_1, f(x, S_1)), (S_2, f(x, S_2)), \ldots, (S_m, f(x, S_m))\}$$

Minimize over all possible weights

Datamodel prediction for margin on target example x after training on S_i, i.e., $g(S_i)$

ℓ_1 regularization (for sparsity + generalization)

$$\theta_x = \min_{w \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \left(w^\top \mathbf{1}_{S_i} - f(x, S_i) \right)^2 + \lambda \|w\|_1$$

Average over all sampled subsets S_i

True (observed) margin from training on S_i and evaluating on x

Putting it all together

Constructing datamodels for DNNs trained on CIFAR-10:

Requires training 1000s of models!
Made possible by FFCV (ffcv.io)

→ Repeat **500,000 times**:

→ Choose a random α-fraction of the CIFAR-10 trainset

→ Train a model (ResNet-9) on this subset

→ Measure **correct-class margin** on every test image

→ For each test image, record the pair:

(characteristic vector of the subset, vector of margins)

→ For each test image (10,000 total images):

→ Fit linear model from indicator vectors → margins

Result: **10,000 datamodels**, each parameterized by $\theta_x \in \mathbb{R}^{50{,}000}$

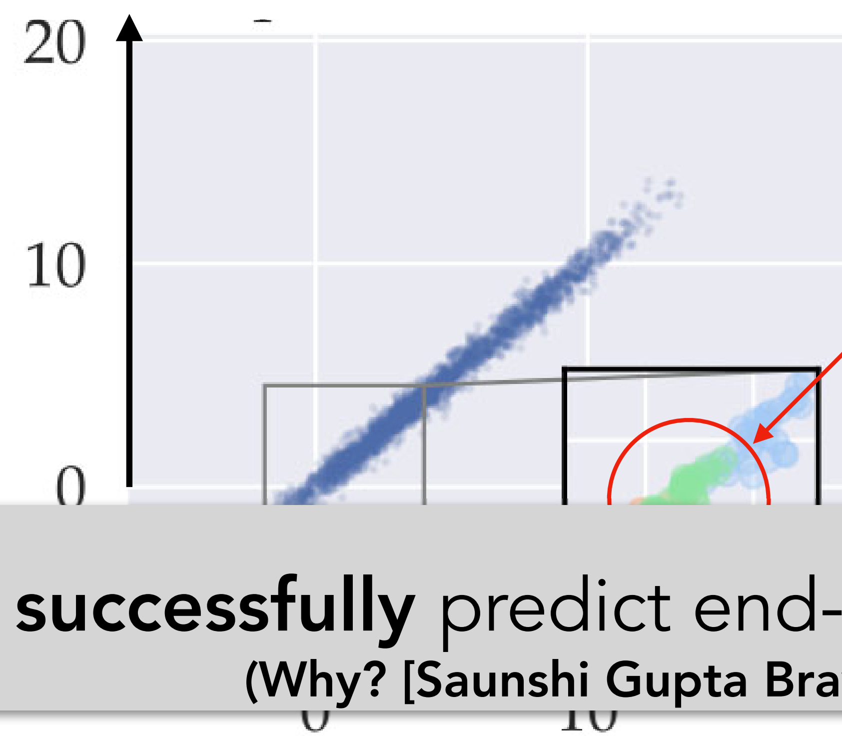
Evaluating datamodels

Idea: Sample *new subsets* S_i, compare predictions to reality

Specifically: Aggregate over **target examples** x (each with their own separate datamodel g_{θ_x}) and **random subsets** S_i of the training set:

Actual margin

$\mathbb{E}[f(x, S_i)]$



Individual target examples (i.e., only varying S_i)

Predicted margin $\theta_x \cdot \mathbf{1}_{S_i}$

(passing the characteristic vector of S_i through the datamodel for x)

Datamodels **successfully** predict end-to-end training

(Why? [Saunshi Gupta Braverman Arora 2022])

Applying datamodels

$$f(x, S') \approx \theta_x^\top \mathbf{1}_{S'}$$

Datamodels provide a versatile framework
for analyzing model predictions and data

We can use datamodels:

→ To analyze **model brittleness**

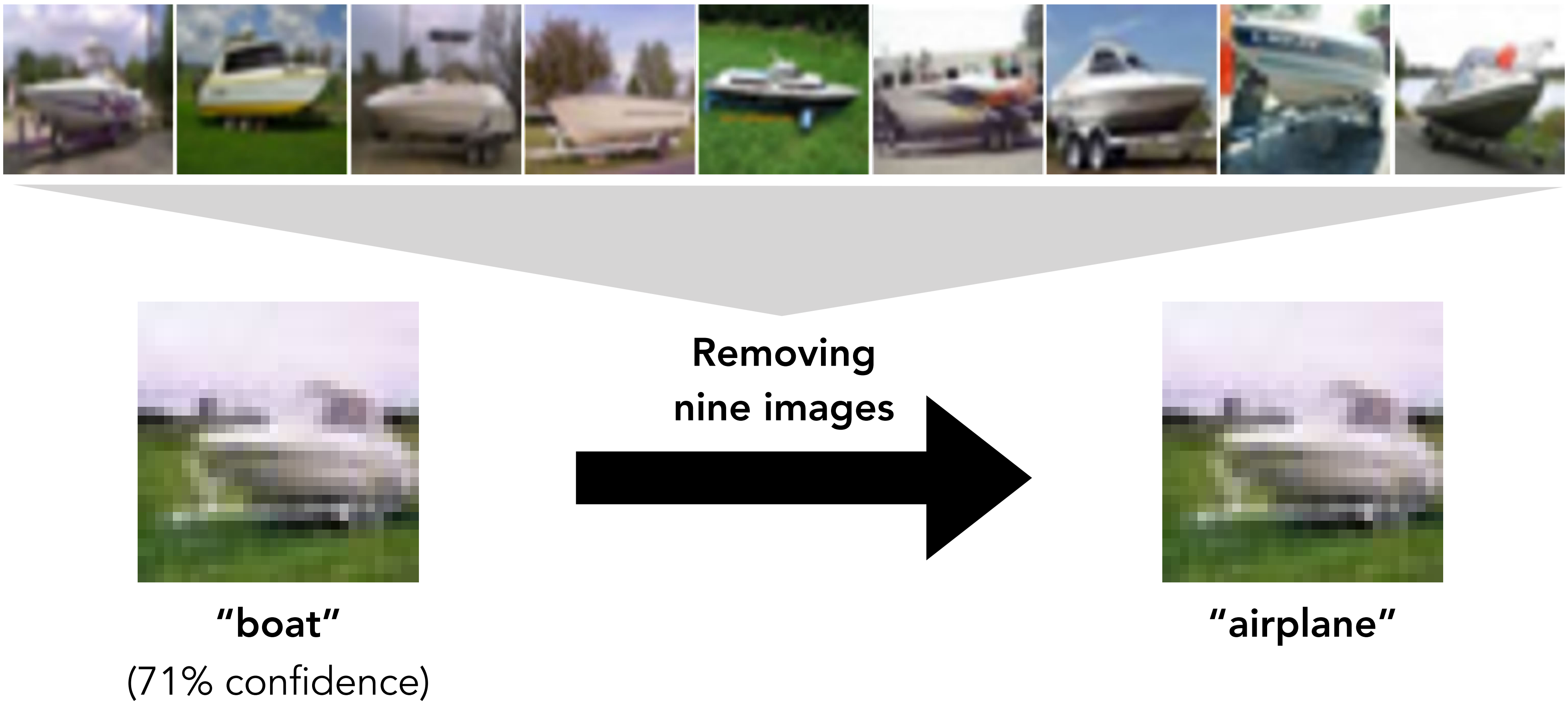
→ To predict **data counterfactuals**

→ To find **train-test leakage**

→ As a rich **embedding** that encodes latent structure

→ To **compare** learning algorithms [Shah Park I Madry 2022]

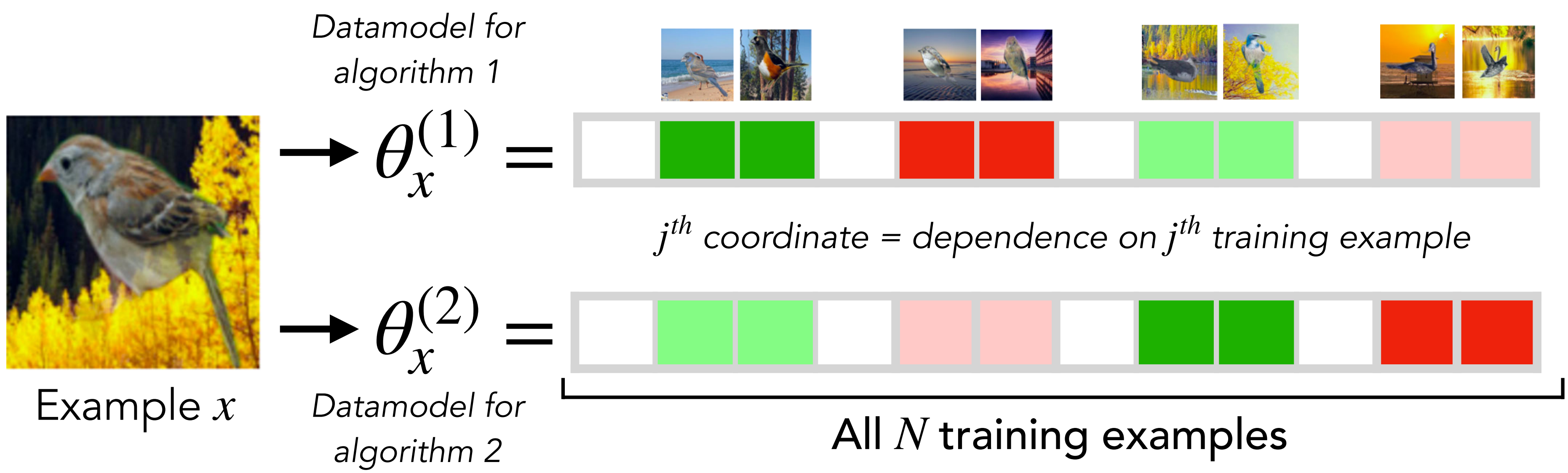
Datamodels: Analyzing model brittleness



~25% of examples misclassified by removing
< 0.2% of training examples

Datamodels: Comparing learning algs

Given Algorithms **1** and **2**, use datamodels to compare model classes M_1 and M_2 in terms of how they rely on training data



Datamodels $\theta_x^{(1)}$ and $\theta_x^{(2)}$ live in the **same** train set space → can make "apples-to-apples" comparison for example x

Takeaways so far

Datamodels:
A framework for understanding both data and predictions

→ Learn data-to-output mapping using supervised learning

→ Simple *linear* instantiation works really well

→ A versatile tool for model-data understanding

But: Very expensive to compute!

Can we do things faster?

Stepping back: Data attribution

A **data attribution method** is a function $\tau : \mathscr{X} \to \mathbb{R}^{|S|}$

Intuitively: $\tau(x)_i$ = importance of the i-th training example

$\tau($  $) =$

[0.1 -0.02 0.03 -0.04 0.2 0.03 -0.1]

Ex: Influence functions, Shapley values, TracIn

[Ghorbani Zou '19, Jia et al. '19, Pruthi et al. '19, Feldman Zhang '20]

Question: How to compare different methods?

Main idea: Connect back to datamodels!

Formalizing attribution with datamodels

A **data attribution method** is a function $\tau : \mathscr{X} \to \mathbb{R}^{|S|}$

Indicator vector of $S' \subset S$

[1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0]

$\tau(x)_i$ = "effect" of training example x_i on model output at x

$$\hat{f}(x, S') = \mathbf{1}_{S'} \cdot \boxed{\tau(x)}$$

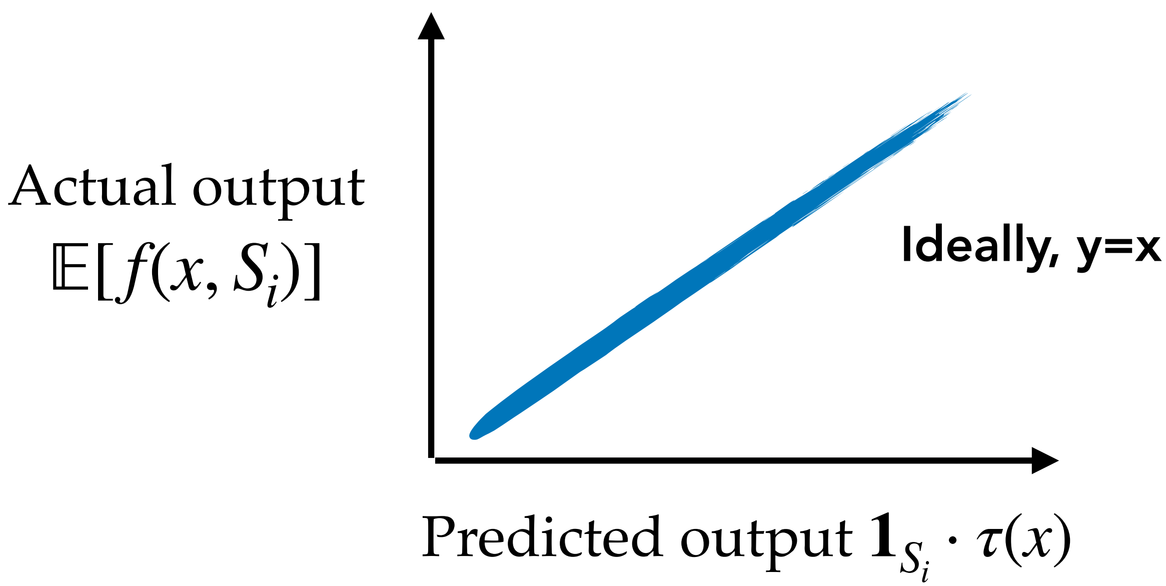
Data attribution method

Want $\tau(\cdot)$ to assign high score to *counterfactually meaningful* training examples

So: Construct "predicted" output from attribution scores

Formalizing attribution with datamodels

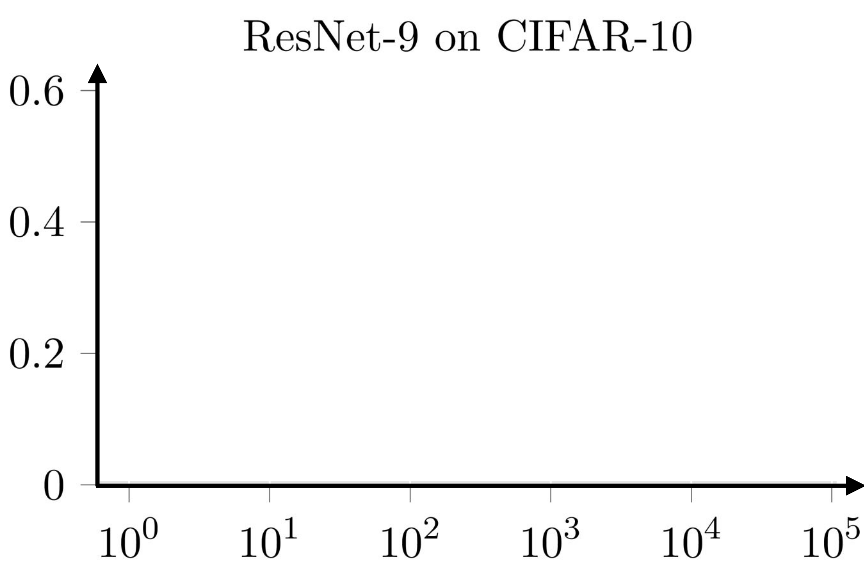
Evaluate predictiveness: Sample *new subsets* S_i,
compare actual model outputs and outputs <u>predicted</u> by τ

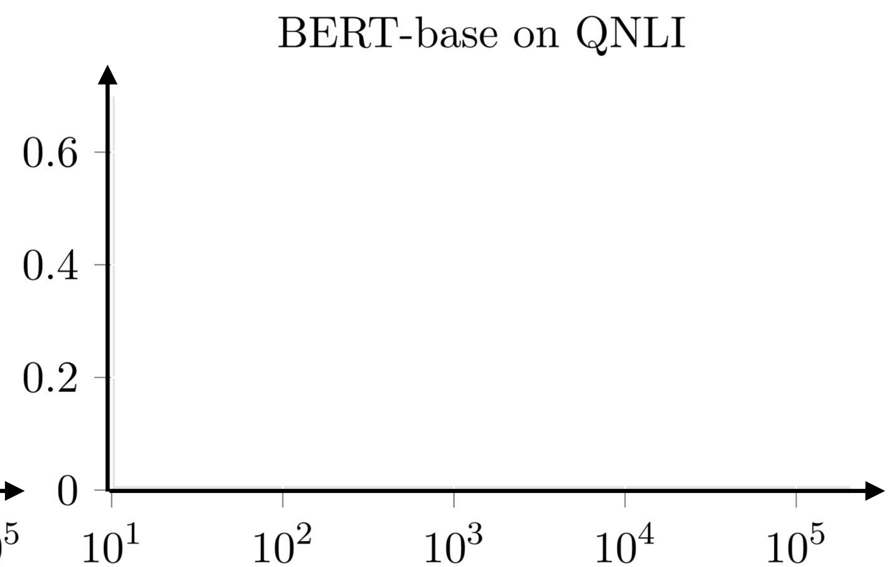


Metric (Linear Datamodeling Score):
Correlation between <u>actual</u> and <u>predicted</u> outputs

Efficacy vs Efficiency

Data attribution should be both **effective** and **efficient**

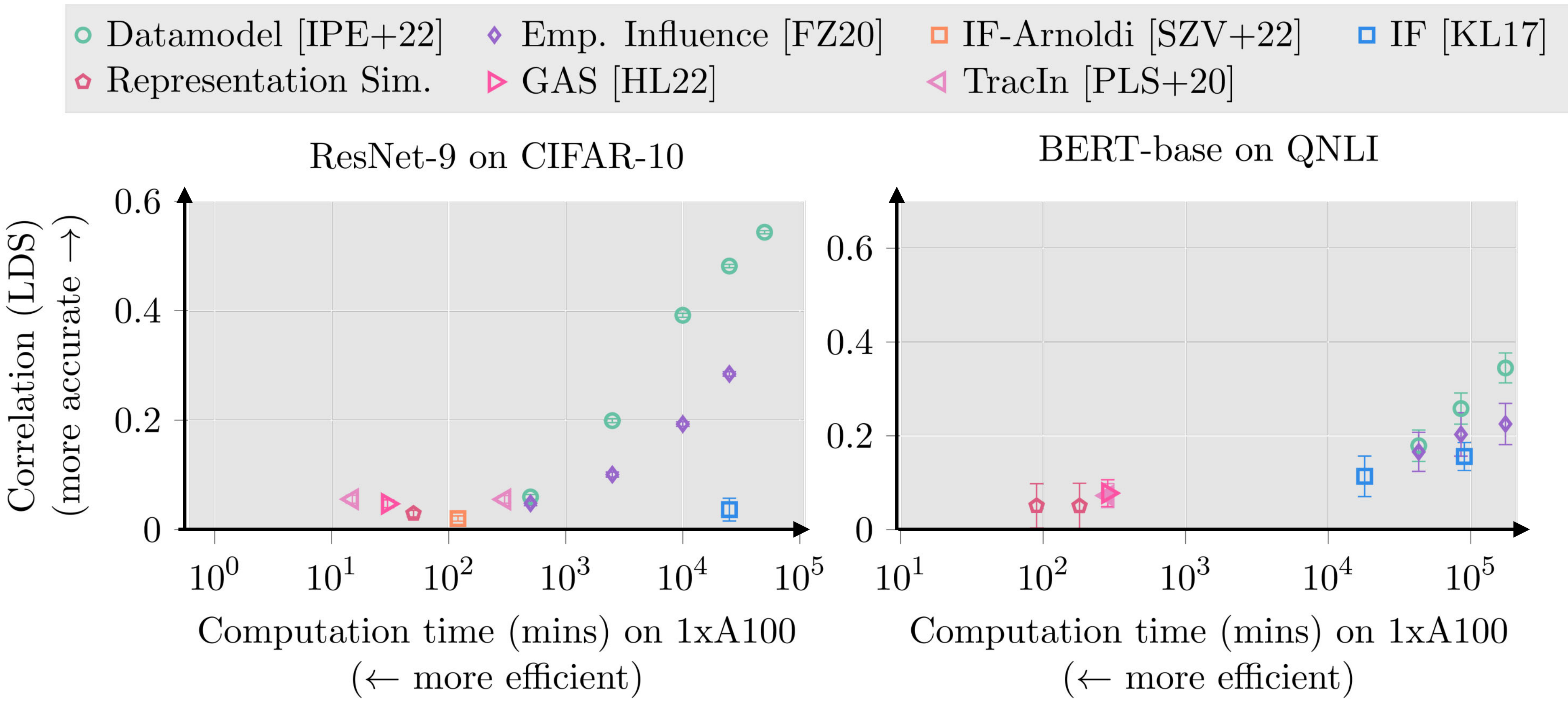




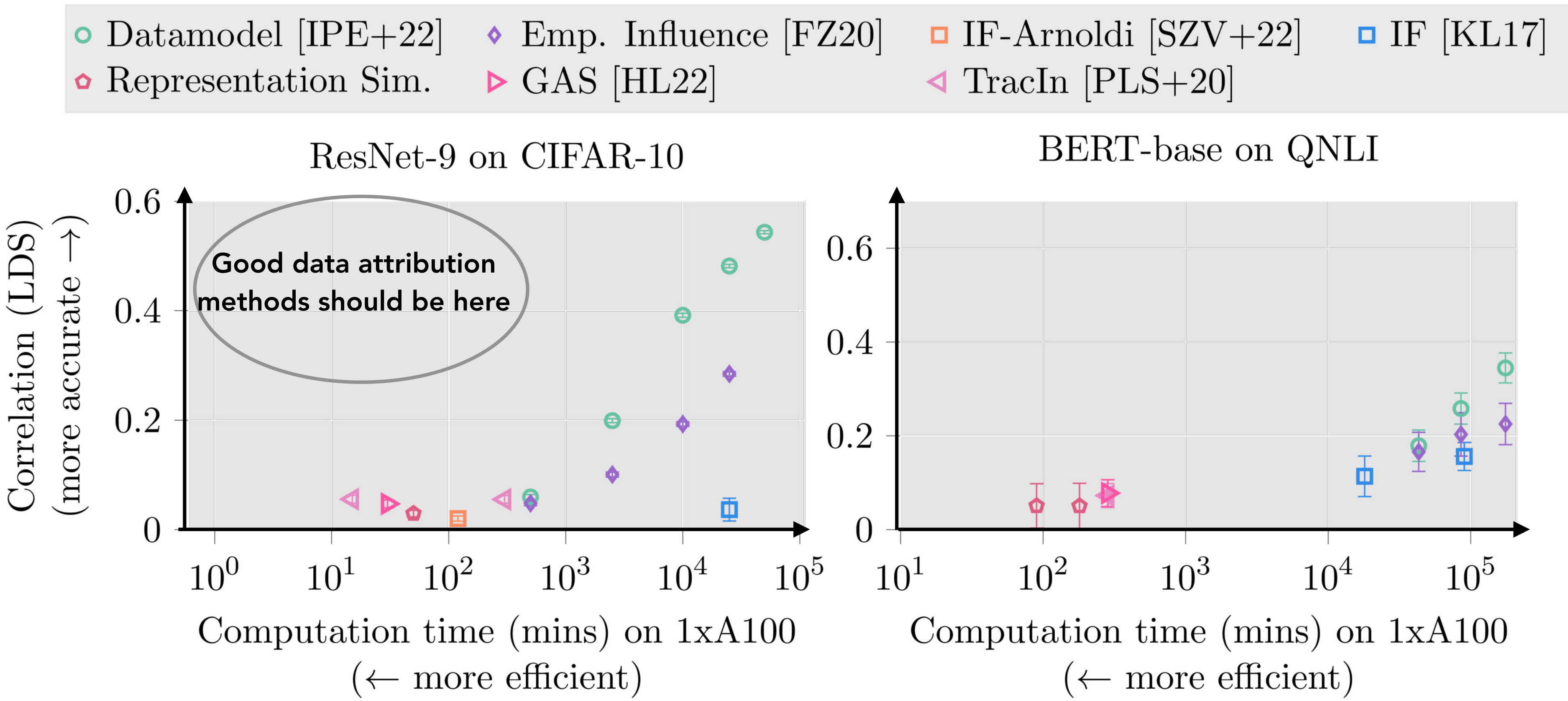
Linear Datamodeling Score (LDS)

Correlation between **true** model output $f(x, S')$ and **predicted** model output $\mathbf{1}_{S_i} \cdot \tau(x)$

Evaluating attribution methods



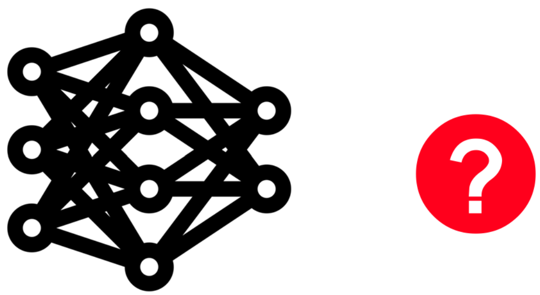
Evaluating attribution methods



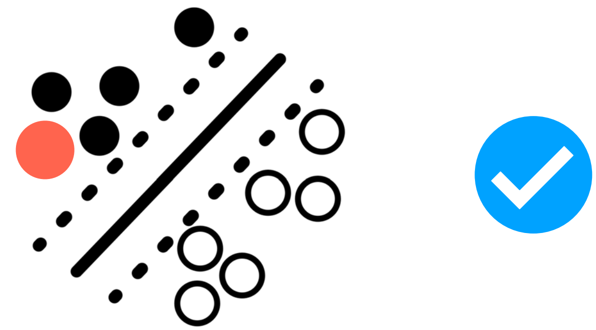
Can we design a method that is both **scalable** and **predictive** in large-scale settings?

Our approach: **TRAK**

Goal: Scalable and effective attribution for large-scale NNs



Arbitrary (differentiable) model



Generalized linear models

Q: Is there a simpler class of models that we can attribute well?

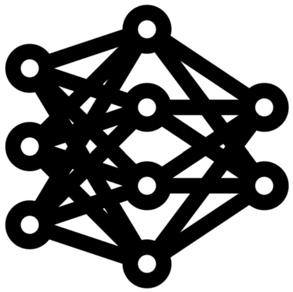
Yes! **Generalized linear models (GLM)**

[Pregibon '81] [Wojnowicz et al. '16] [Koh Ang Teo Liang '19]

Key idea: Reduce complex models → GLM, then apply known methods

Approximation approach: TRAK

Tracing with the **R**andomly-projected **A**fter **K**ernel



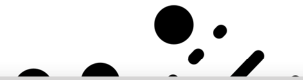
Inpu

Outp

Original neural network

Car

complicated



linear model

Inputs:

$\nabla_\theta f(x; \theta^\star)$

Output:

$\nabla_\theta f(x; \theta^\star)^\top \theta$

For the experts: TRAK linearizes the model using the *empirical neural tangent kernel (eNTK)*, also known as the **after kernel**

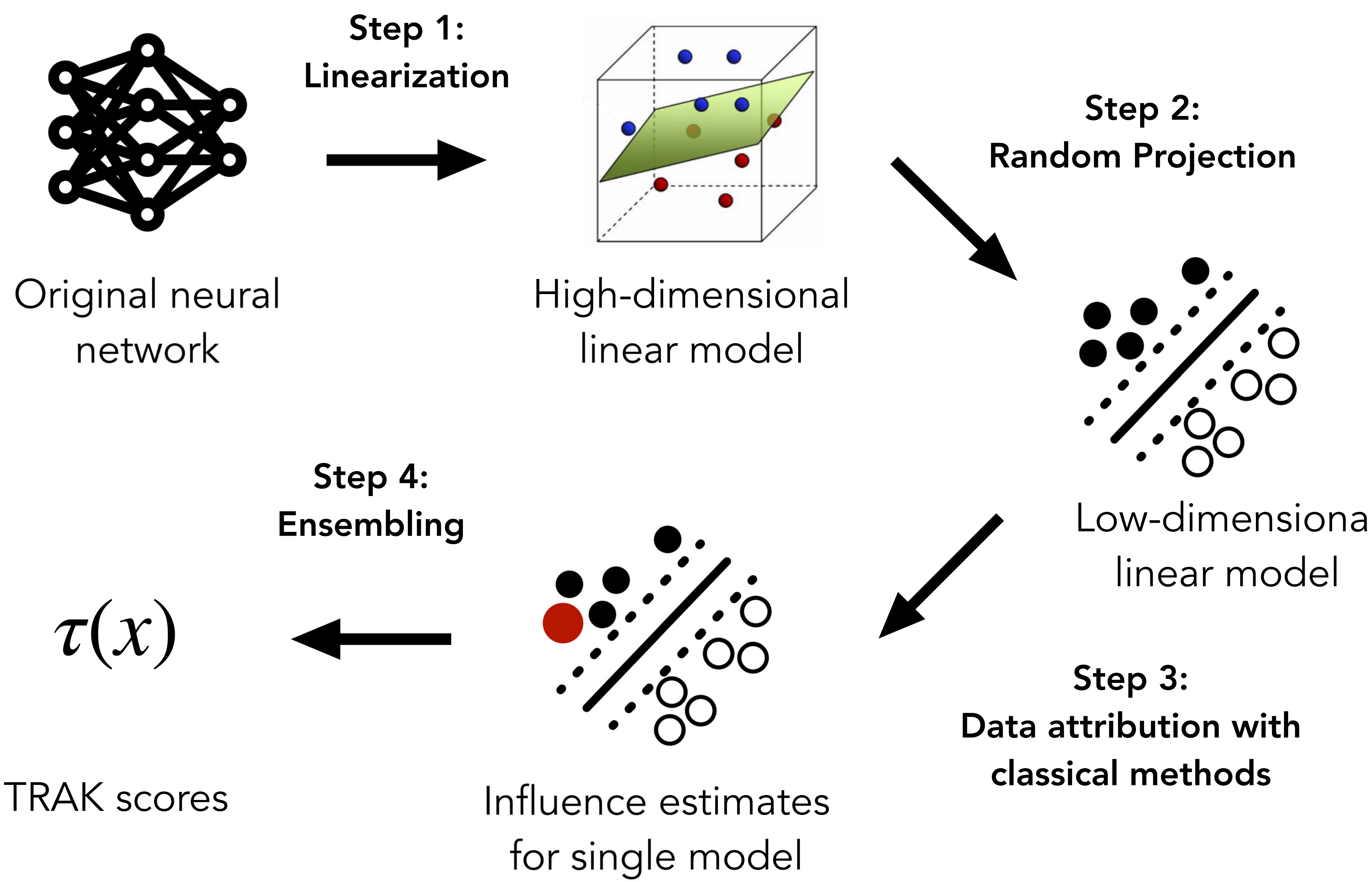
Our approach: Taylor approximation

$$f(x, \theta) \approx f(x; \theta^\star) + \nabla_\theta f(x; \theta^\star) \cdot (\theta - \theta^\star)$$

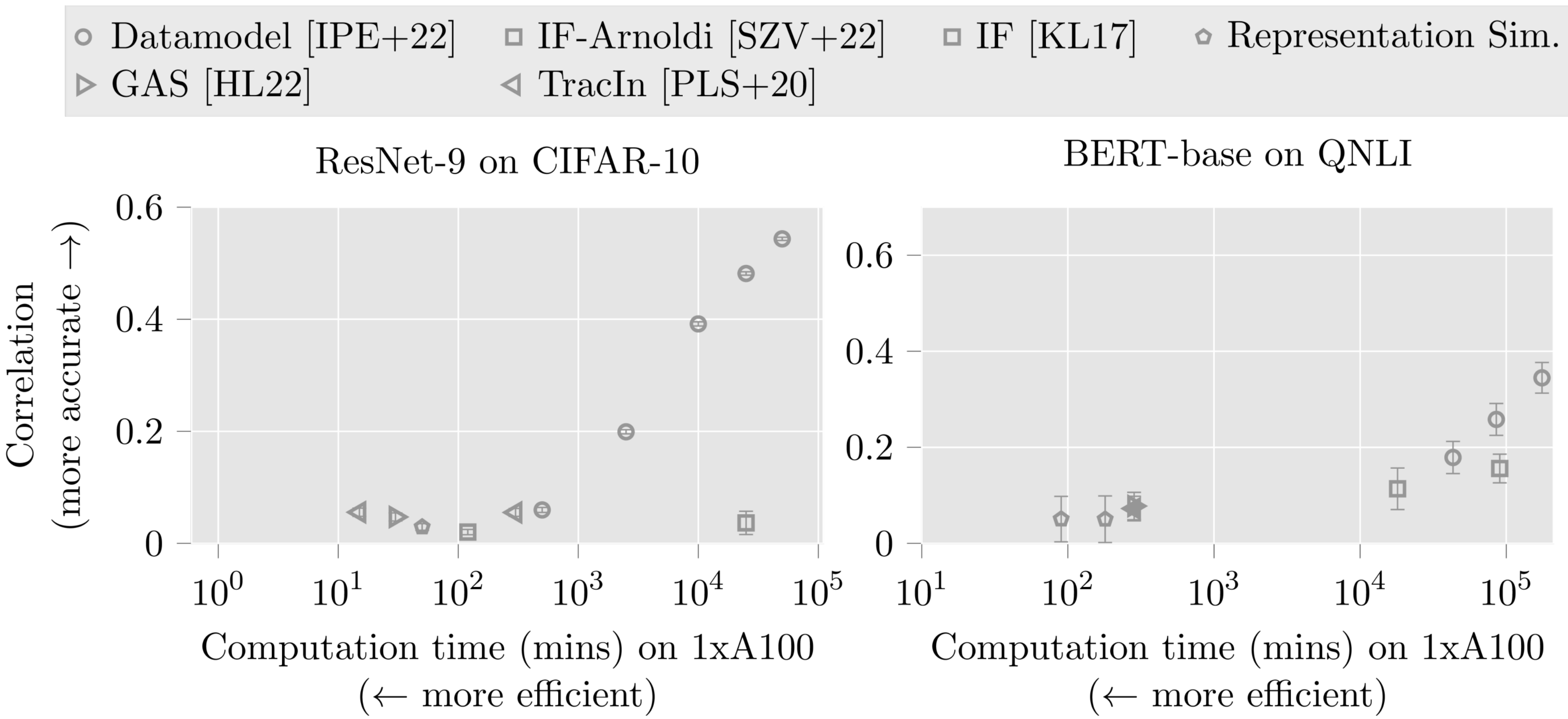
Final parameters (constant wrt θ)

This is a linear function in the parameter θ

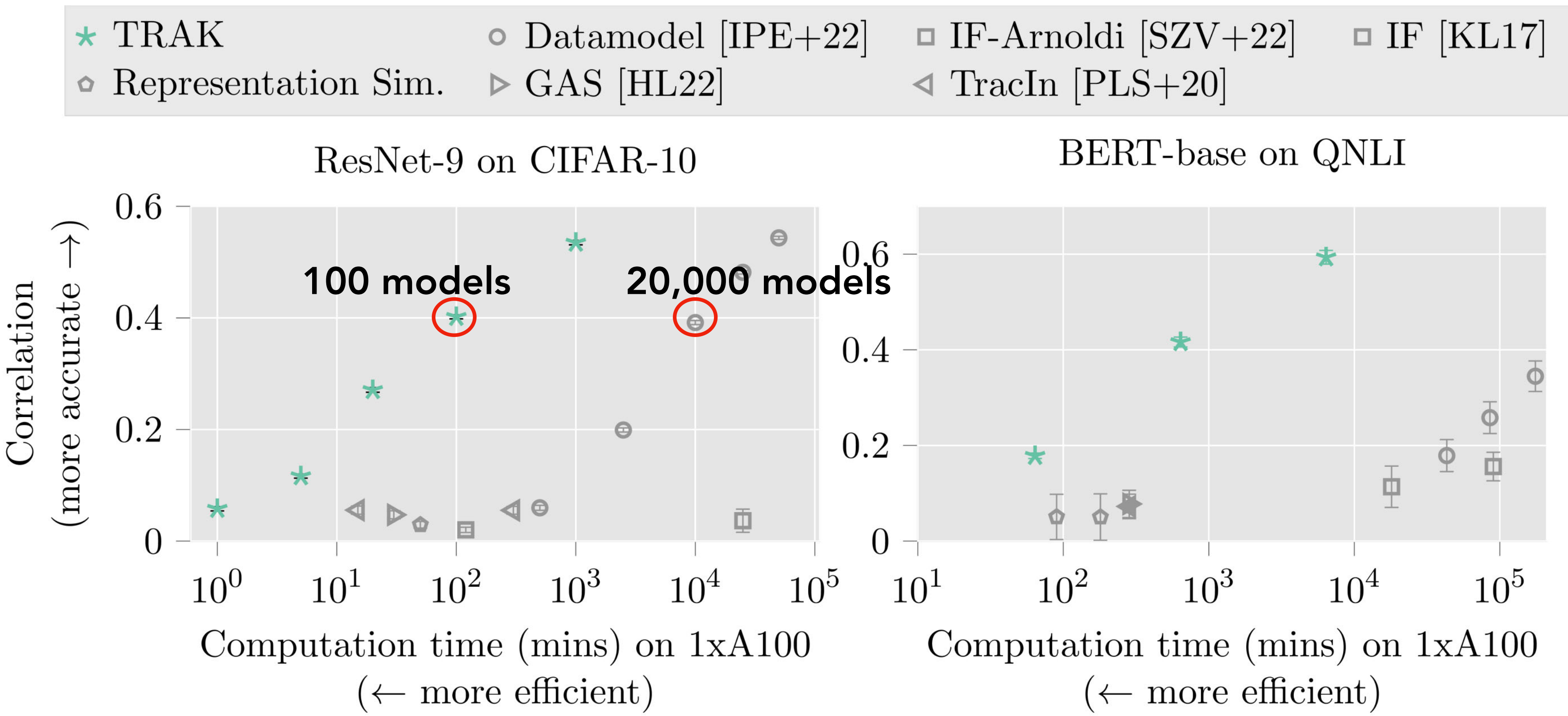
TRAK: Summary



Evaluating TRAK



Evaluating TRAK



In particular: TRAK speeds up datamodels by 100x-1000x

Applications

In our paper, we apply **TRAK** to:

- CLIP
- Language models
- ImageNet classifiers

OpenAI CLIP

BERT, mT5

IMAGENET

Applications

In our paper, we apply **TRAK** to:

- CLIP
- Language models
- ImageNet classifiers

BERT, mT5

Applying TRAK to LLMs

"Lionel Messi won the
Ballon d'Or seven times."

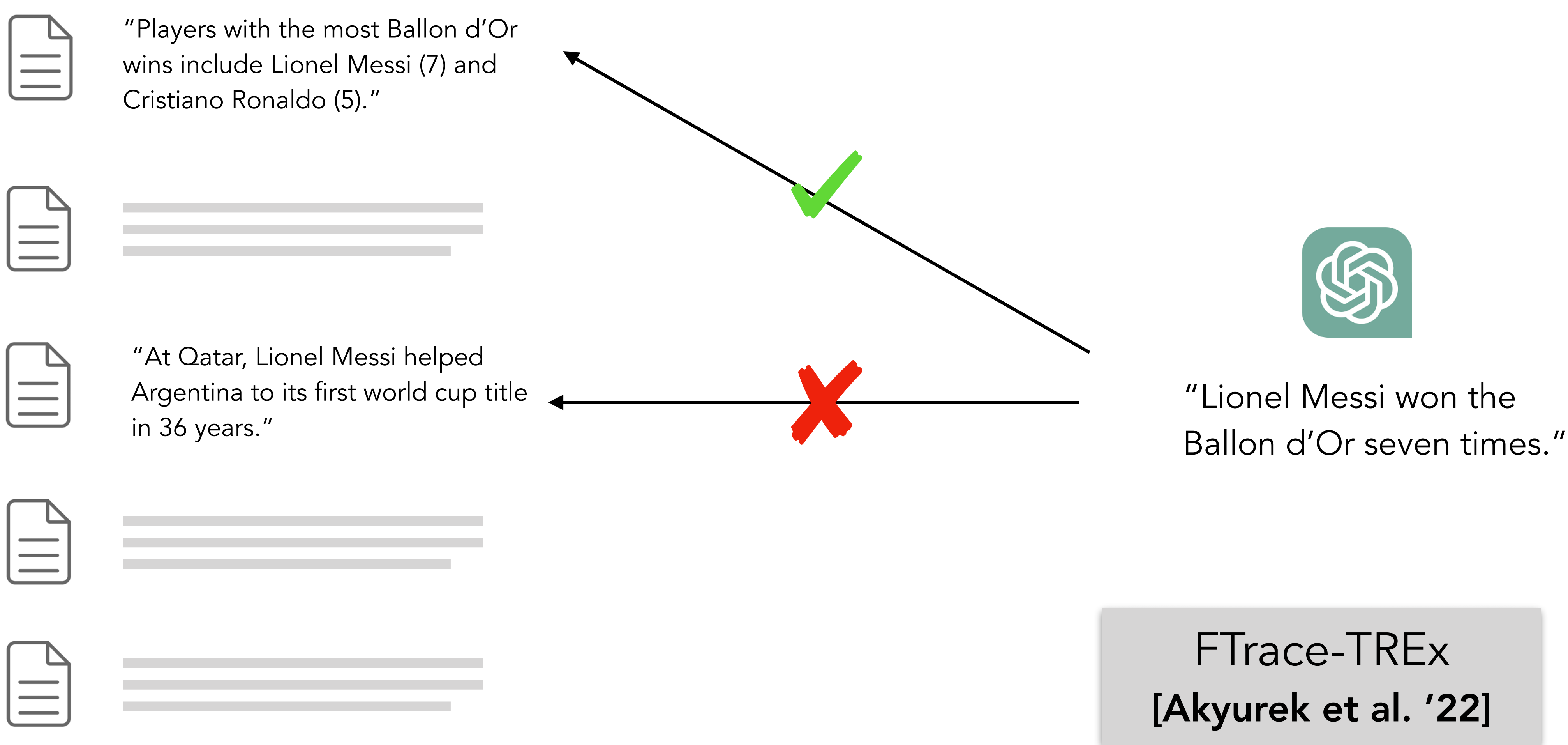
Possible questions to ask about this output:

→ Why did the language model output this answer?

→ Can we identify the training data that led to this output?

One lens for studying this question: Fact tracing

Applying TRAK to fact tracing



PyTorch API

```python
from torchvision import models

from trak import TRAKer

model = models.resnet18()
checkpoint = model.state_dict()
train_loader, val_loader = ...

traker = TRAKer(model=model, task='image_classification', train_set_size=...)

traker.load_checkpoint(checkpoint)
for batch in train_loader:
    traker.featurize(batch=batch, num_samples=batch_size)
traker.finalize_features()

traker.start_scoring_checkpoint(checkpoint, num_targets=...)
for batch in val_loader:
    traker.score(batch=batch, num_samples=batch_size)
scores = traker.finalize_scores()
```

Try it! github.com/MadryLab/trak

Explainability Recap

- Feature Attribution Methods
  - LIME (Local Interpretable Model-agnostic Explanations) algorithm
  - SHAP methods based on cooperative game theory
  - Saliency Maps (different versions)
  - Formal guarantees for feature attribution methods
  - Counterfactuals
  - Representation-based explanations

- Data attribution methods
  - Influence Functions
  - Datamodels

- Next lecture: Neurosymbolic Learning (guest lecture by PhD student Ziyang Li)