Datamodels:

Predicting Predictions with Training Data

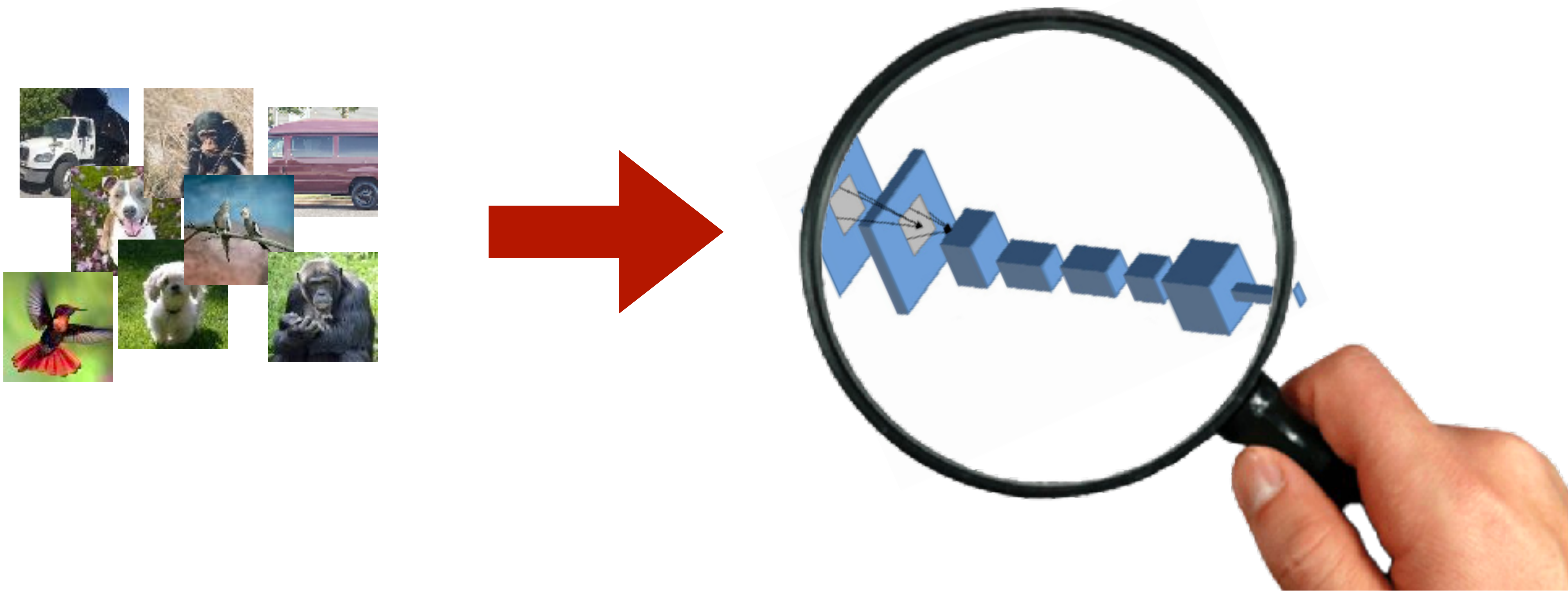
Aleksander Mądry

(with Logan Engstrom, Andrew Ilyas, Guillaume Leclerc, Sam Park)

@aleks_madry

madry-lab.ml

How Do We Look at the ML Pipeline?

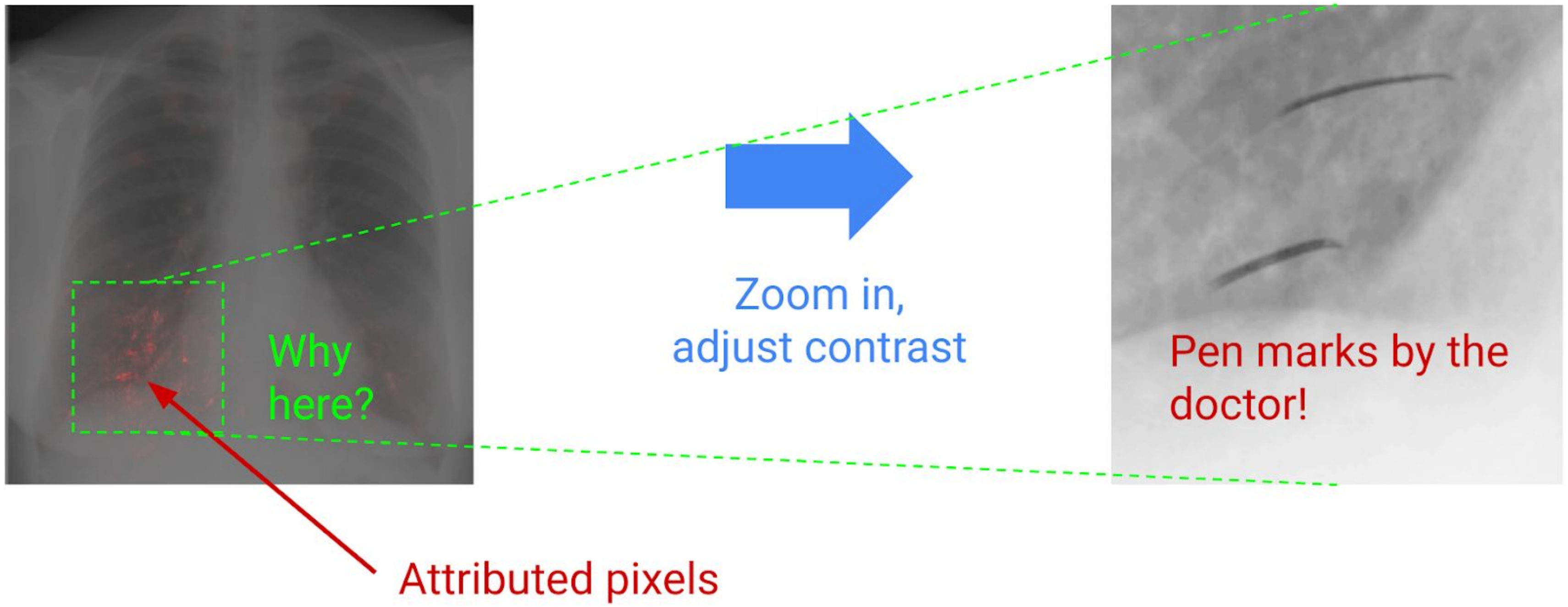


Indeed: Data Matters (a Lot)

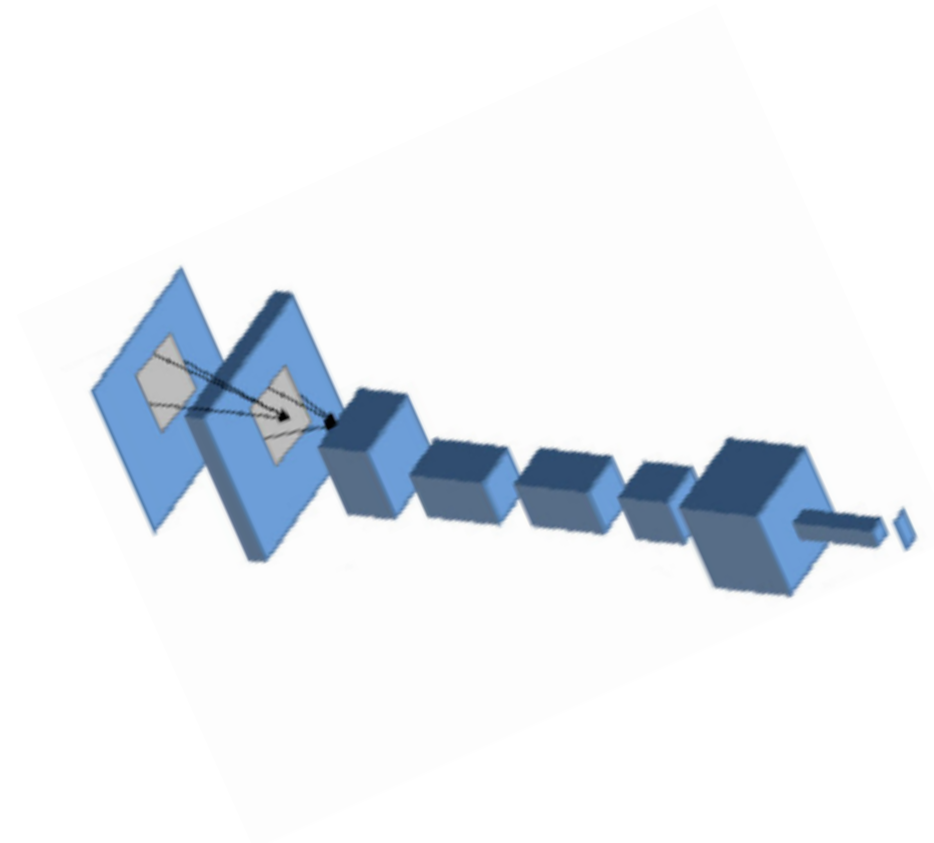
→ Data poisoning

"dog"

→ "Opportunistic" learning



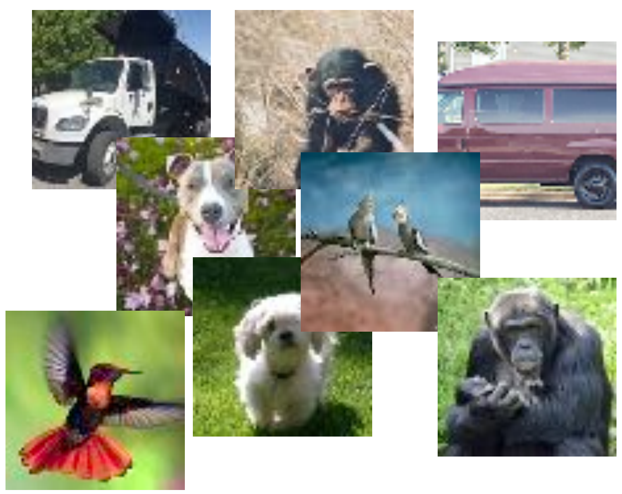
Model-Driven Understanding of Data



Can we analyze data as it's viewed/used by the models?

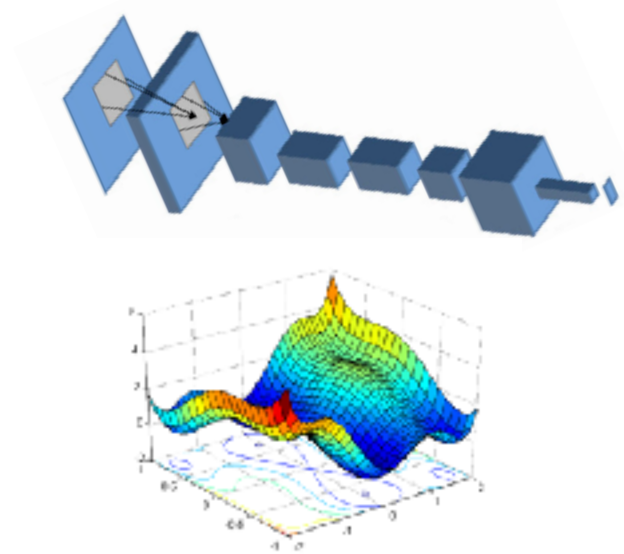
Basic Primitive: Scrutinizing Predictions

Training set S

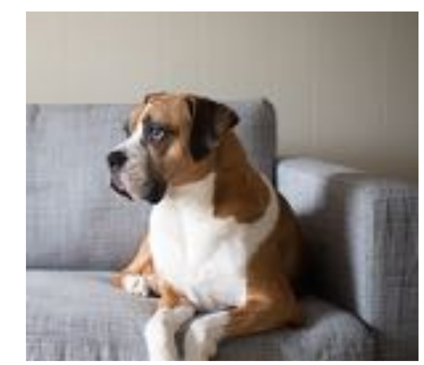


+

Learning algorithm



Test input x



"Dog" 85%

Which training inputs impact this prediction the most?

"Classic" Approach: Influence Functions

[Cook Weisberg 1980]

Specifically: Approximate leave-one-out influences

$$\widehat{Infl}\,[x_i \to x_j] = \Pr[\text{model trained on } S \text{ is correct on } x_j] - \Pr[\text{model trained on } S \setminus \{x_i\} \text{ is correct on } x_j]$$

→ **[Koh Liang 2017]:** Approx. using Hessian (of penultimate layer) of a specific model, but:

→ Affected by model-training variability

→ Penultimate layer does not seem to capture all the info

→ **[Feldman Zhang 2020]:** More direct estimation

But: Can we get a more direct read?

Goal: Understand how the training data yields model outputs through the lens of training algorithm

In particular:

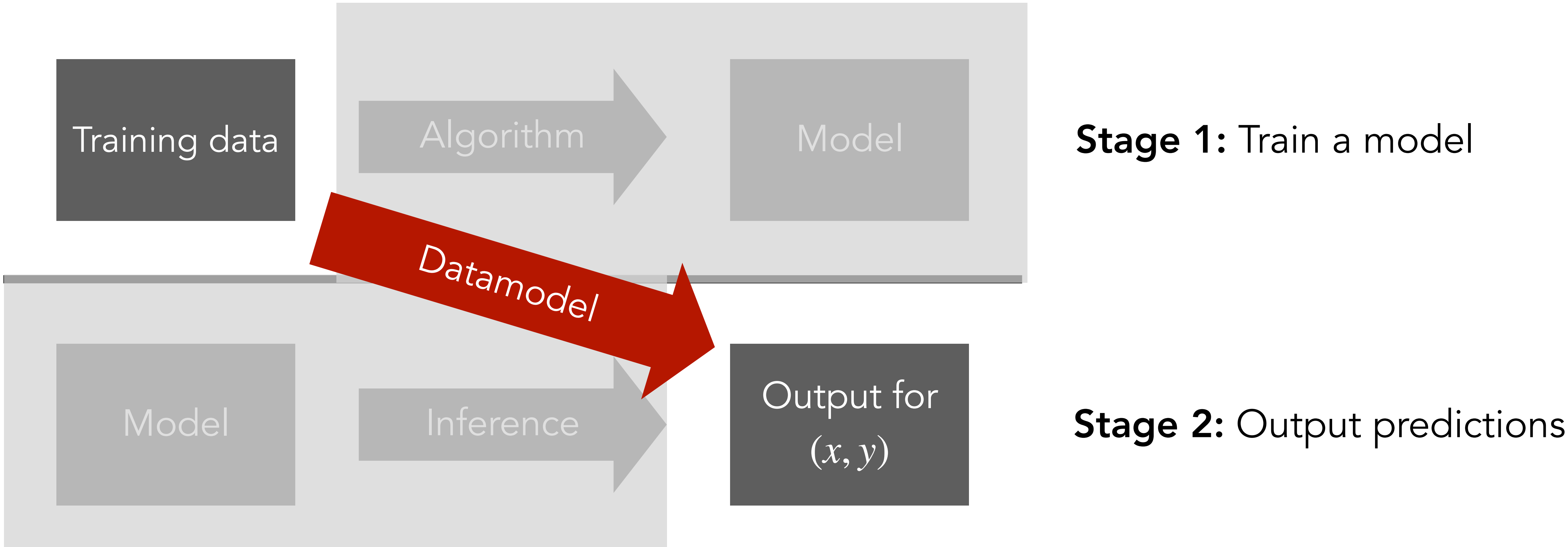
→ Go beyond the focus on a single-input impact

→ Be able to grasp more nuanced aspects of predictions than "just" them being correct/incorrect

→ Get a way to explicitly analyze how well we are doing

Our Proposed Approach:
Datamodels

Datamodels: Data-to-Output Modeling



Idea: Completely abstract away <u>everything</u> "in the middle"

("Smoothing out" the randomness/idiosyncrasies of model training)

Datamodels: Data-to-Output Modeling

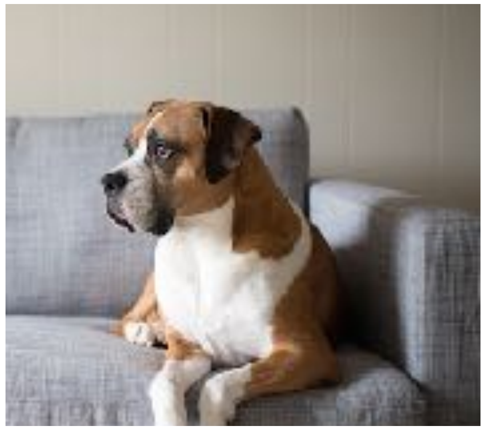
What we are trying to compute:

"Smoothened" output of interest
(think: margin of correct class)

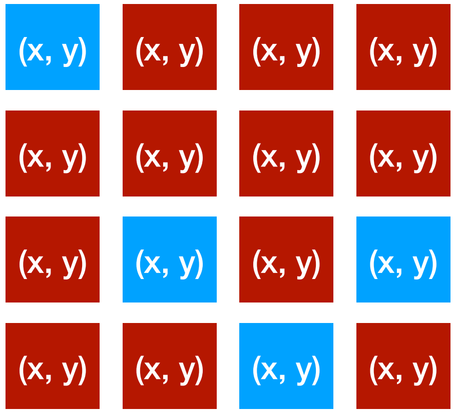
$$f(x, S) \quad \approx \quad g(x, S)$$

Datamodel

Want it to be simple/easy to analyze



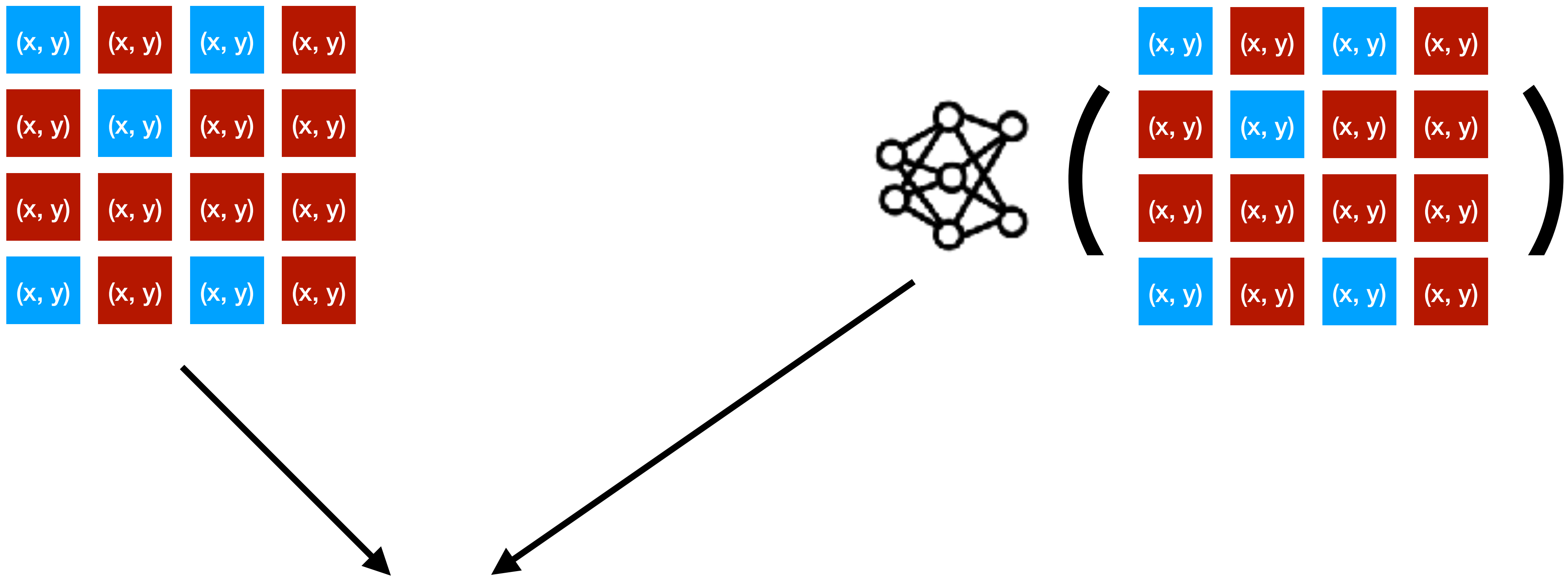
Specific input x



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

How To Find Such A Datamodel?

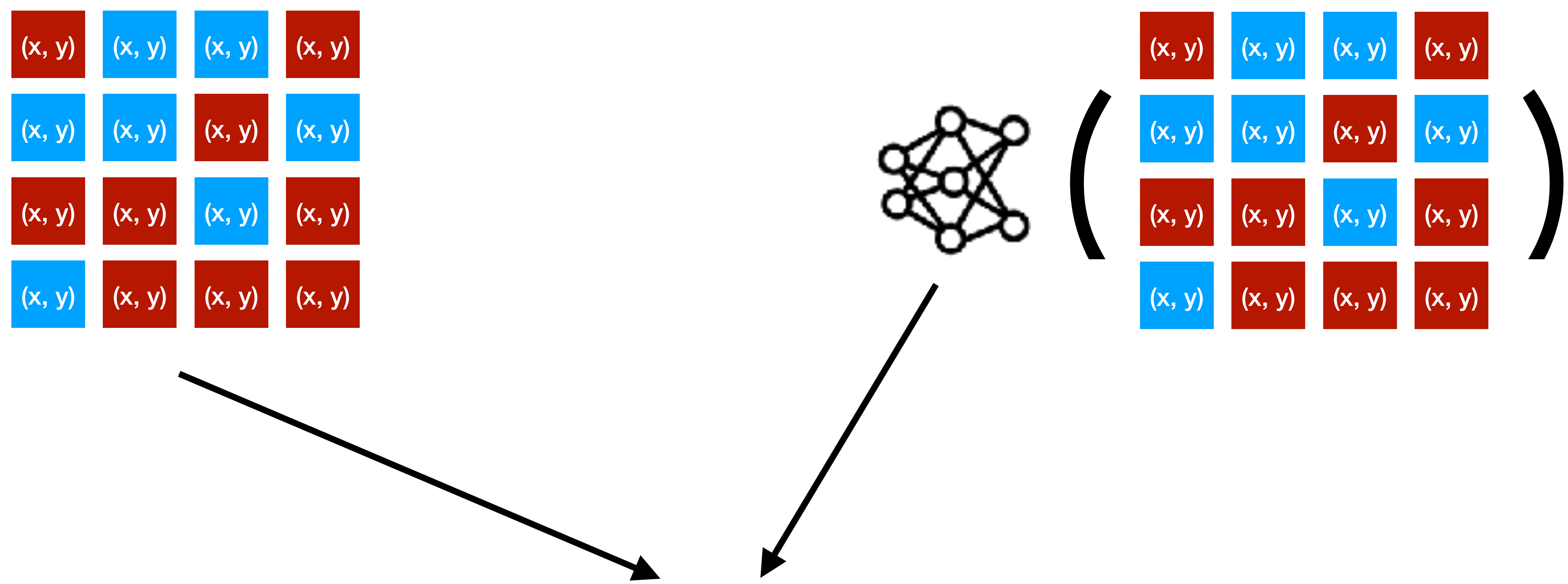
Simple: Just treat it as a regression problem



$$(S_1, \vec{f}_1)$$

How To Find Such A Datamodel?

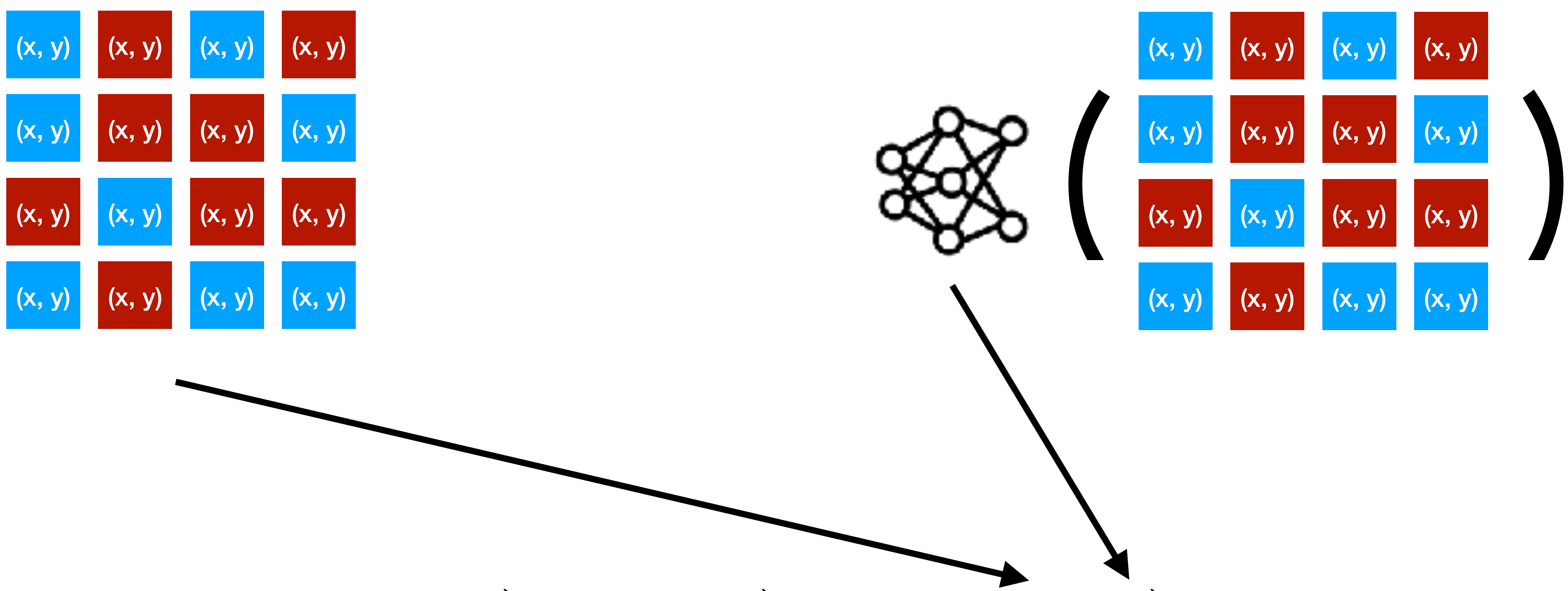
Simple: Just treat it as a regression problem



$$(S_1, \vec{f}_1), (S_2, \vec{f}_2)$$

How To Find Such A Datamodel?

Simple: Just treat it as a regression problem



$$\{(S_1, \vec{f}_1), (S_2, \vec{f}_2), \ldots, (S_m, \vec{f}_m)\}$$

Then: Fit a model to this data

Two Emerging Questions

How to generate the data?

→ Just sample random α-fraction subsets of S, for $\alpha \in (0,1)$

What class of models to fit?

→ **Turns out:** A simple choice works already very well

Model Choice: Linear(-ish)

$$g(x, S) = \theta_x^\top m(S)$$

Vector of weights assigned to training points

Binary (one-hot) representation of S

→ We fit vectors θ_x for all inputs x of interest

→ **To fit this datamodel:** Train **~500K (!)** different classifiers

(How to do that? **See: ffcv.io**)

So: How can this be useful?

Understanding Data with Datamodels

Datamodels turn out to be a versatile framework for analyzing ML predictions

In particular, they provide:

→ A causal characterization of model decisions

→ A perceptually meaningful similarity measure (for images)

→ A (good) embedding of datapoints into Euclidean space

→ A graph representation of the training data structure

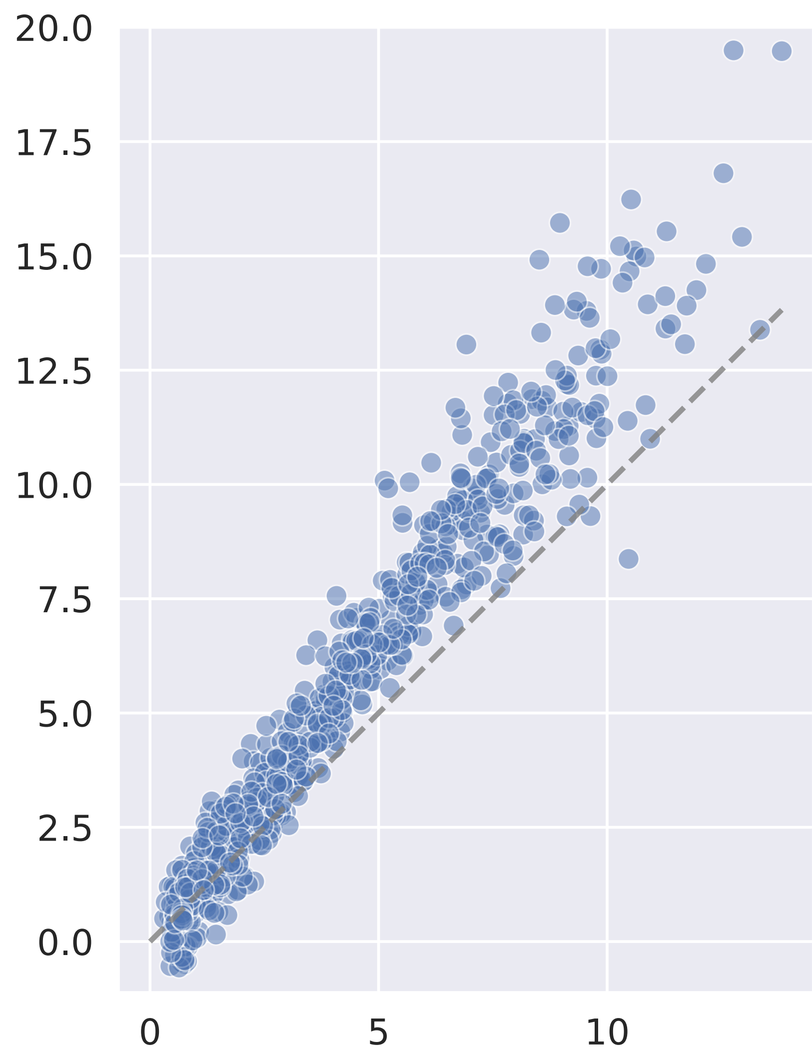
Datamodels: Causal Perspective

Goal: Estimate $f(x, S')$ without explicitly training on S'

Datamodels: Causal Perspective

Observed effect:

$$f(x, S) - f(x, S')$$



Predicted effect: $g(x, S) - g(x, S')$

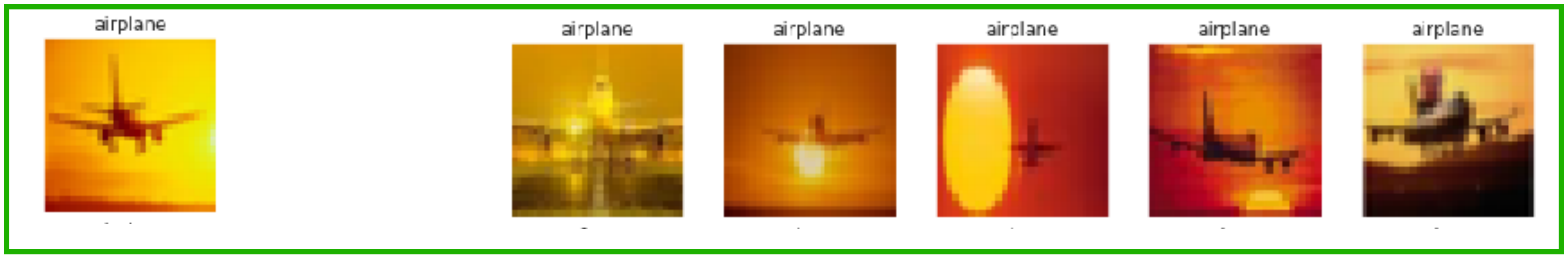
Results: Datamodels provide accurate counterfactuals (even for a different α regime)

Datamodels: Similarity Measure

Inputs x of interest

Datamodels: Similarity Measure

Inputs x of interest Training points with the highest positive θ_x−weight



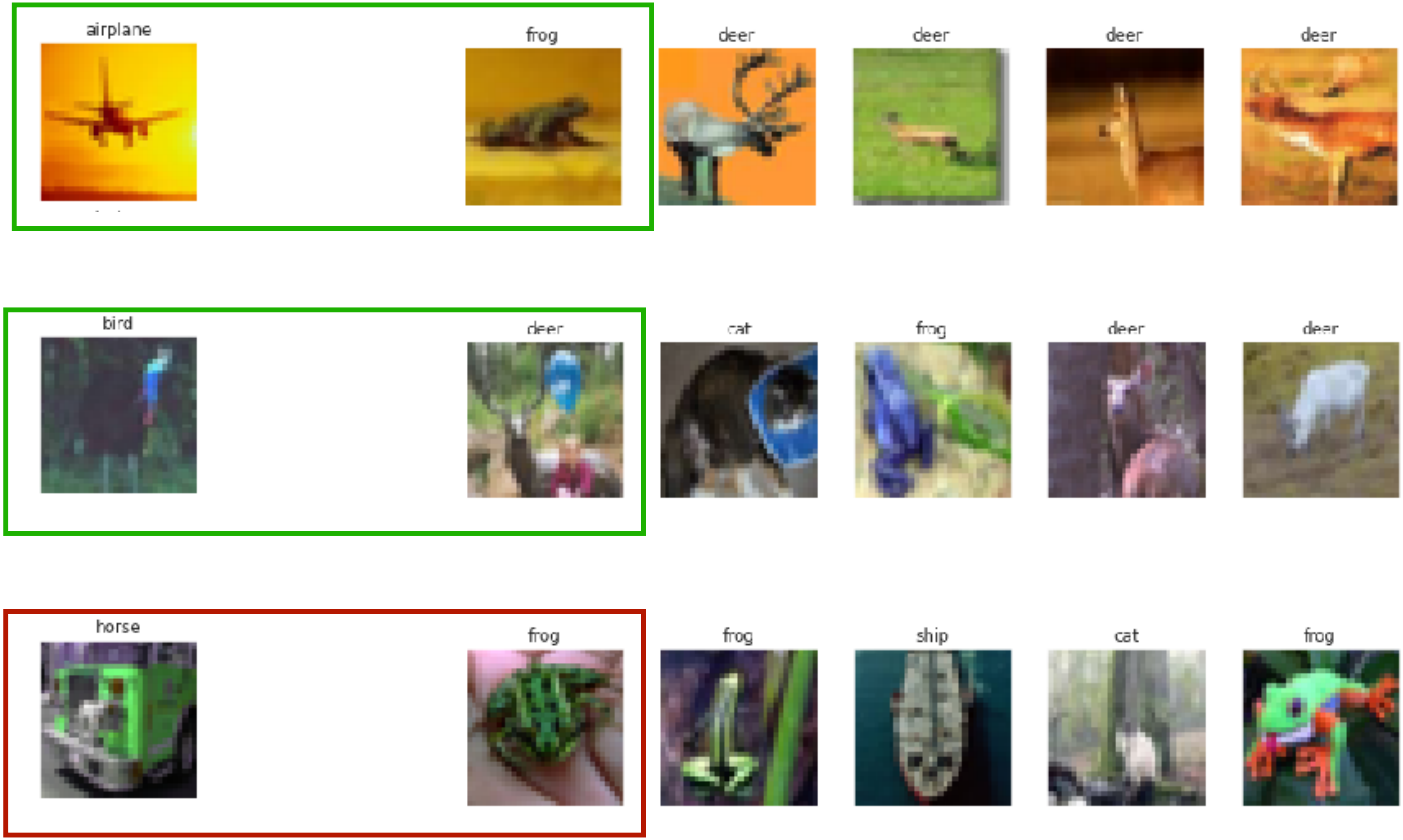
feature similarity

train-test duplication

train-test leakage

Datamodels: Similarity Measure

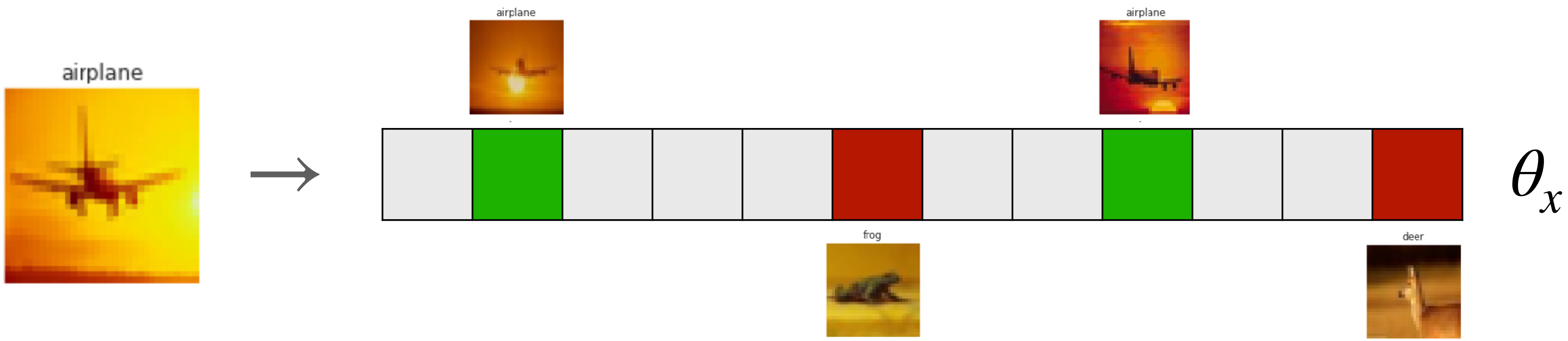
Inputs x of interest   Training points with the highest negative θ_x−weight



Datamodels: Embedding

Note: Weights θ_x can provide a (sparse) embedding of each x

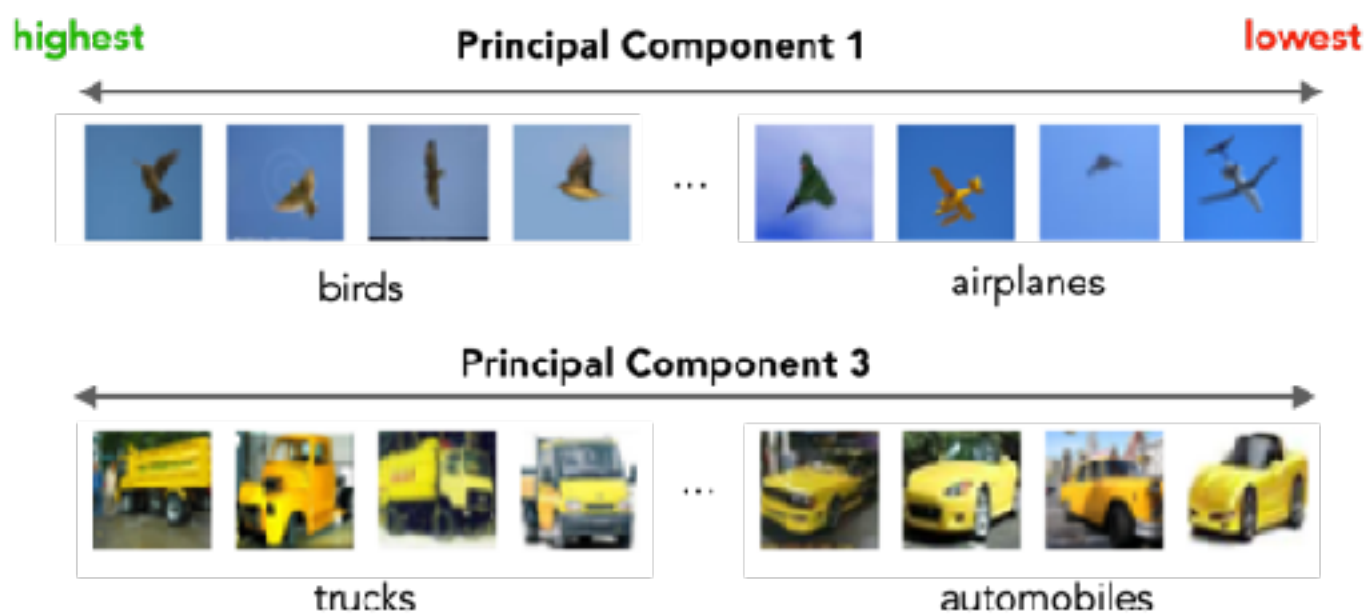
Result: A "smoothened" representation space



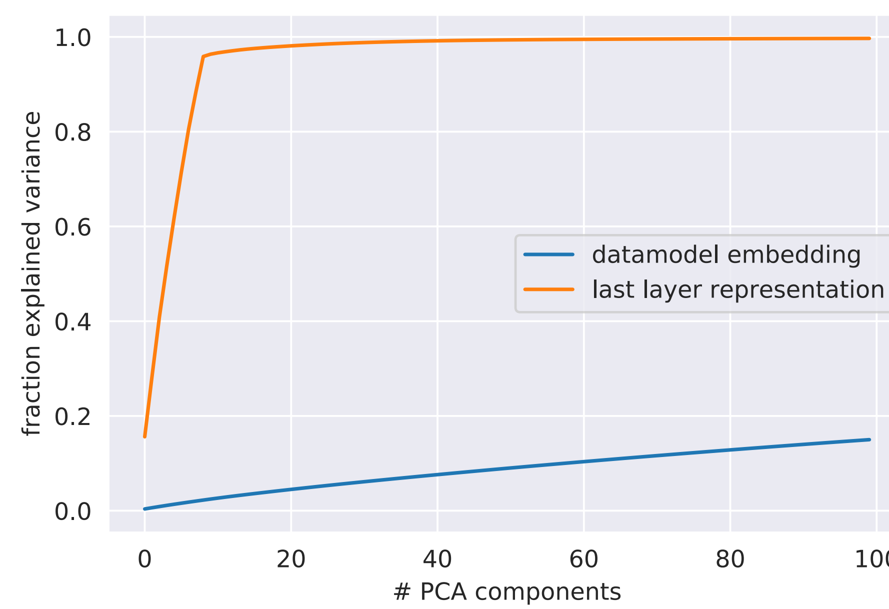
Now: What if we perform PCA on this embedding?

Datamodels: Embedding

Result: PCA recovers "features"

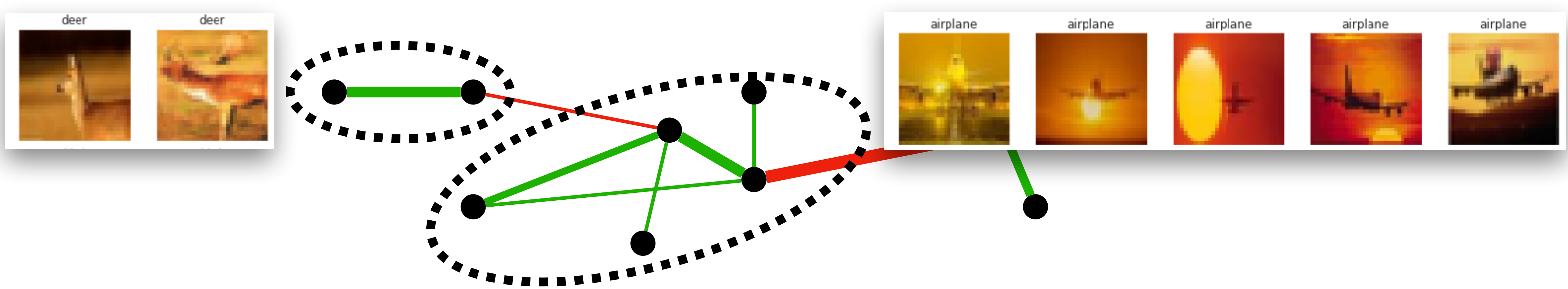


Interestingly: This PCA has far more non-trivial directions than a classifier representation space



Datamodels: Graph Perspective

Idea: Stack datamodel weights θ_x for **each** training point x to get an adjacency matrix



→ Enables us to use graph-theoretic algorithms to understand datasets

Takeaways

Datamodels:
A new framework for model-centric data understanding

→ Learn data-to-output mapping using regression

→ Simple *linear* instantiation works really well

→ Gives rise to a variety of primitives:

  → Predicting counterfactuals/analyzing model brittleness

  → Provides rich embedding/graph structure

  → What else?

See paper/blogpost for (much) more!

@aleks_madry

gradientscience.org