Datamodels: Predicting Predictions from Training Data

Andrew Ilyas [*1] **Sung Min Park** [*1] **Logan Engstrom** [*1] **Guillaume Leclerc** [1] **Aleksander Mądry** [1]

Abstract

We present a conceptual framework, *datamodeling*, for analyzing the behavior of a model class in terms of the training data. For any fixed "target" example x, training set S, and learning algorithm, a *datamodel* is a parameterized function $2^S \to \mathbb{R}$ that for any subset of $S' \subset S$—using only information about which examples of S are contained in S'—predicts the outcome of training a model on S' and evaluating on x. Despite the complexity of the underlying process that is being approximated (e.g. end-to-end training and evaluation of deep neural networks), we show that even simple *linear* datamodels successfully predict model outputs. We then demonstrate that datamodels give rise to a variety of applications, such as: accurately predicting the effect of dataset counterfactuals; identifying brittle predictions; finding semantically similar examples; quantifying train-test leakage; and embedding data into a well-behaved and feature-rich *representation space*.

1. Introduction and Setup

What kinds of biases does my (machine learning) system exhibit? On what subpopulations does it perform well (or poorly)? A recent body of work suggests that the answers to these questions lie within both the learning algorithm and the training data used. However, it is often difficult to understand *how* these two factors combine to yield model predictions. In this work, we present *datamodeling*—a framework for tackling this issue by forming an explicit model for predictions in terms of the training data.

Setting. Consider a typical machine learning setup, starting with a training set S comprising d input-label pairs. The focal point of this setup is a *learning algorithm* $\mathcal{A}$ that takes in such a training set of input-label pairs, and outputs a trained model. This learning algorithm need not be deterministic—for example, $\mathcal{A}$ might encode the process of training a neural network from random initialization.

Now, consider a *fixed* target example x and define

$$f_{\mathcal{A}}(x; S) := \text{the outcome of training a model on } S \text{ using } \mathcal{A}, \text{ and evaluating it on the input } x, \quad (1)$$

where we leave "outcome" intentionally broad to capture a variety of settings that one might care about. For example, $f_{\mathcal{A}}(x; S)$ may be the cross-entropy loss of a classifier on x, or the error of a regression model on x. The potential stochasticity of $\mathcal{A}$ means $f_{\mathcal{A}}(x; S)$ is a random variable.

Goal. Broadly, we aim to understand how the training examples in S combine through the learning algorithm $\mathcal{A}$ to yield $f_{\mathcal{A}}(x; S)$ (again, for the *specific* example x that we are examining). Towards this goal, we will leverage a classic technique for studying complex black-box functions: *surrogate modeling* (Sacks et al., 1989). In surrogate modeling, one replaces complex functions with inexact but significantly easier-to-analyze approximations, then uses the latter to shed light on the behavior of the original functions.

In our setting, the complex black-box function is $f_{\mathcal{A}}(x; \cdot)$. We thus aim to find a simple *surrogate* function $g(S')$ whose output roughly matches $f_{\mathcal{A}}(x; S')$ for a variety of training sets S' (but again, for a *fixed* example x). Achieving this goal would reduce the challenge of scrutinizing $f_{\mathcal{A}}(x; \cdot)$—and more generally, the map from training data to predictions as mediated by learning algorithm $\mathcal{A}$—to the (hopefully easier) task of analyzing g.

Datamodeling. By parameterizing the surrogate function g (as g_θ, for a parameter vector θ), we transform the challenge of constructing a surrogate into a *supervised learning* problem. In this problem, the "training examples" are subsets $S' \subset S$ of the original task's training set S, and the corresponding "labels" are given by $f_{\mathcal{A}}(x; S')$ (which we can compute by training a new model on S' with algorithm $\mathcal{A}$, and evaluating on x). Our goal is then to fit a parametric function g_θ mapping the former to the latter.

We now formalize this as *datamodeling*—the framework that forms the basis of our work. In this framework, we first fix a distribution over subsets that we will use to collect the

[*]Equal contribution [1]MIT. Correspondence to: Andrew Ilyas <ailyas@mit.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

"training data" for g_θ,

$$\mathcal{D}_S := \text{a fixed distribution over subsets of } S \tag{2}$$

and then use $\mathcal{D}_S$ to collect a *datamodel training set* of such training data, i.e., a collection of pairs

$$\{(S_1,\ f_\mathcal{A}(x; S_1)), \ldots, (S_m,\ f_\mathcal{A}(x; S_m))\},$$

where $S_i \sim \mathcal{D}_S$, and again $f_\mathcal{A}(x; S_i)$ is the result of training a model on S_i and evaluating on x (cf. (1)).

In theory, g_θ can be any map that takes as input subsets of the training set, and returns estimates of $f_\mathcal{A}(x; \cdot)$. In this work, however, we greatly simplify matters by ignoring the actual *contents* of the subsets S_i, and instead focusing solely on the *presence* of each training example of S within S_i. In particular, we consider the *characteristic vector* corresponding to each S_i,

$$\mathbf{1}_{S_i} \in \{0,1\}^d \text{ such that } (\mathbf{1}_{S_i})_j = \begin{cases} 1 & \text{if } z_j \in S_i \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

a vector that indicates which elements of the original training set S are present in a given subset S_i. We then define a <u>*datamodel*</u> for a given input x as a function

$$g_\theta : \{0,1\}^d \to \mathbb{R}, \qquad \text{where}$$
$$\theta = \arg\min_w \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(g_w(\mathbf{1}_{S_i}),\ f_\mathcal{A}(x; S_i)\right), \tag{4}$$

and $\mathcal{L}(\cdot, \cdot)$ is a fixed loss function (e.g., squared-error). As intended, (4) places datamodels squarely within the realm of supervised learning: e.g., we can *validate* a given datamodel by sampling new subset-output pairs $\{(S_i, f_\mathcal{A}(x; S_i))\}$ and computing loss. For completeness, we restate the entire framework as Def. 1.

Definition 1 (Datamodeling)**.** *Consider a fixed training set S, a learning algorithm $\mathcal{A}$, a target example x, and a distribution $\mathcal{D}_S$ over subsets of S. For any set $S' \subset S$, let $f_\mathcal{A}(x; S')$ be the (stochastic) output of training a model on S' using $\mathcal{A}$, and evaluating on x. A <u>datamodel</u> for x is a parametric function g_θ optimized to predict $f_\mathcal{A}(x; S_i)$ from training subsets $S_i \sim \mathcal{D}_S$, i.e.,*

$$g_\theta : \{0,1\}^{|S|} \to \mathbb{R}, \qquad \textit{where}$$
$$\theta = \arg\min_w \widehat{\mathbb{E}}^{(m)}_{S_i \sim \mathcal{D}_S} \left[\mathcal{L}\left(g_w(\mathbf{1}_{S_i}),\ f_\mathcal{A}(x; S_i)\right)\right],$$

$\mathbf{1}_{S_i} \in \{0,1\}^{|S|}$ is the characteristic vector of S_i in S (see (3)), $\mathcal{L}(\cdot, \cdot)$ is a loss function, and $\widehat{\mathbb{E}}^{(m)}$ is an m-sample empirical estimate of the expectation.

We pause here to highlight two critical (yet somewhat subtle) properties of the datamodeling framework:

Model classes, not models: Datamodeling focuses on the entire distribution of models induced by the algorithm $\mathcal{A}$, rather than a specific model. Recent work suggests this distinction is particularly significant for modern learning algorithms (e.g., neural networks), as models can exhibit drastically different behavior depending on only random seed used during training (Nakkiran & Bansal, 2020; Jiang et al., 2021; D'Amour et al., 2020; Zhong et al., 2021).

Datamodels are target example-specific: A datamodel g_θ is selected to predict model outputs on a specific but arbitrary target example x. This x might be an example from the test set, a synthetically generated example, or even an example from the training set S itself. As a result, we will often work with *collections* of datamodels that correspond to given sets of target examples: it turns out (§ 3.1) that, as long as the training set S is fixed, computing a collection of datamodels simultaneously is not much harder than computing a single one.

1.1. Roadmap and contributions

The key contribution of our work is the *datamodeling framework* described above, which allows us to analyze the behavior of a machine learning algorithm $\mathcal{A}$ in terms of the training data. In the remainder of this work, we show how to instantiate, implement, and apply this framework.

Specifically, in Section 2, we consider a concrete instantiation of datamodeling in which the map g_θ is a *linear* function. In Section 3, we develop the remaining machinery required to apply this instantiation to deep neural networks trained on standard image datasets. Then, in the rest of the paper, we find that:

Datamodels successfully predict model outputs (§ 3.2): despite their simplicity, datamodels yield predictions that match expected model outputs on new sets S' drawn from the same distribution $\mathcal{D}_S$. (E.g., the Pearson correlation between predicted and ground-truth outputs is $r > 0.99$.)

Datamodels successfully predict counterfactuals (§ 4): predictions correlate with model outputs even on out-of-distribution training subsets (Figure 4 and Appendix E), allowing us to estimate the *causal effect* of removing training images on a given test prediction. Leveraging this ability, we find that *for 50% of CIFAR-10 test images, predictions can be made incorrect by removing (i.e., not even mislabeling) less than 200 target-specific training points (i.e., 4% of total training set size).*

Datamodel weights encode train-test similarity (§ 5.1): the highest-magnitude datamodel weights for a given target example tend to correspond to similar training examples, which we can use to identify (significant) train-test leakage.

Datamodels yield a well-behaved feature embedding

(§ 5.2): we can view the parameter θ of a given datamodel g_θ as a *feature embedding* of the corresponding target example x into $\mathbb{R}^d$. We find that this embedding induces a well-behaved representation space that has significant advantages over standard embeddings from the penultimate layer of a fixed network.

More broadly, datamodels turn out to be a versatile tool for understanding how learners leverage their training data.

2. Constructing (Linear) Datamodels

As described in Section 1, to build a datamodel for a specific target example x, we: **(a)** pick a parameterized class of functions g_θ; **(b)** sample a collection of subsets $S_i \subset S$ from a fixed training set according to a distribution $\mathcal{D}_S$; **(c)** for each subset S_i, train a model using algorithm $\mathcal{A}$, evaluate the model on target input x using the relevant metric (e.g., loss); collect the resulting pair $(\mathbf{1}_{S_i}, f_\mathcal{A}(x; S_i))$; **(d)** split the collected dataset of subset-output pairs into a datamodel training set of size m, a validation set of size m_{val}, and a test set of size m_{test}; **(e)** estimate parameters θ by fitting g_θ on subset-output pairs, i.e., by minimizing

$$\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(g_\theta(\mathbf{1}_{S_i}),\ f_\mathcal{A}(x; S_i)\right)$$

over the collected datamodel training set, and use the validation set to perform model selection.

We now explicitly instantiate this framework, with the goal of understanding the predictions of (deep) *classification* models (we discuss the precise experimental setup in the following section). To this end, we revisit the steps **(a-e)** above, and consider each relevant aspect—the sampling distribution $\mathcal{D}_S$, the output function $f_\mathcal{A}(x; S)$, the parameterized family g_θ, and the loss function $\mathcal{L}(\cdot, \cdot)$—separately:

(a) What surrogate function g_θ should we use? The first design choice to make is that of the family of parameterized surrogate functions g_θ that we will optimize over. At first, one might be tempted to use a complex family of functions in the hope of avoiding misspecification error. After all, g_θ is meant to be a surrogate for the end-to-end training of a deep classifier. In this work, however, we will instantiate datamodeling by taking $g_\theta(\cdot)$ to be a simple *linear* mapping

$$g_\theta(\mathbf{1}_{S_i}) := \theta^\top \mathbf{1}_{S_i} + \theta_0, \tag{5}$$

where we recall that $\mathbf{1}_{S_i}$ is the size-d *characteristic vector* of S_i within S (see (3)).

Remark 1. *While we will allow $g_\theta(\cdot)$ to fit a bias term as above, for notation convenience we omit θ_0 and write $\theta^\top \mathbf{1}_{S_i}$ to represent a datamodel prediction for the set S_i.*

(b) What distribution $\mathcal{D}_S$ over training subsets do we use? In step (a) of the estimation process above, we collect a "datamodel training set" by sampling subsets $S_i \subset S$ from a distribution $\mathcal{D}_S$. A reasonable first choice for $\mathcal{D}_S$—and indeed, the one we consider for the remainder of this work—is the distribution of random α-fraction subsets of the training set. Formally, we set

$$\mathcal{D}_S = \text{Uniform}\left(\{S' \subset S : |S'| = \alpha d\}\right) \tag{6}$$

This design choice reduces the problem of picking $\mathcal{D}_S$ to one of picking the *subsampling fraction* $\alpha \in (0, 1)$, a decision whose impact we explore in Appendix C. In practice, we estimate datamodels for *several* choices of α, as it turns out that the value of α corresponding to the most useful datamodels can vary by setting.

(c) What outputs $f_\mathcal{A}(x; S')$ should we track? Recall that for any subset $S' \subset S$ of the training set S, $f_\mathcal{A}(x; S')$ is intended to be a specific (potentially stochastic) function representing the output of a model trained on S' and evaluated on a target x. There are, however, several candidates for $f_\mathcal{A}(x; S')$ based on which output we opt to track.

In the context of understanding *classifiers*, perhaps the simplest such candidate is the *correctness* function (i.e., a stochastic function that is 1 if the model trained on S' is correct on x, and 0 otherwise). However, while the correctness function may be a natural choice for $f_\mathcal{A}(x; S')$, it ignores potentially valuable information about the model's confidence in a given decision.

A natural way to improve over our initial candidate would thus be to use continuous output function, such as cross-entropy loss or correct-label confidence. But which exact function should we choose? In Appendix B, we use a heuristic to guide our choice to the *correct-class margin*:

$$\begin{aligned} f_\mathcal{A}(x; S') := &\ (\text{logit for correct class}) \\ &- (\text{highest incorrect logit}). \end{aligned} \tag{7}$$

(e) What loss function $\mathcal{L}$ should we minimize? In step (e) above, we are free to pick any estimation algorithm for θ. This freedom of choice allows us to incorporate *priors* into the datamodeling process. In particular, one might expect that predictions on a given target example will not depend on every training example. We can thus incorporate a corresponding *sparsity prior* by adding ℓ_1 regularization, i.e., setting

$$\theta = \min_{w \in \mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \left(w^\top \mathbf{1}_{S_i} - f_\mathcal{A}(x; S_i)\right)^2 + \lambda \|w\|_1, \tag{8}$$

where we recall that d is the size of the original training set S. We can use cross-validation to select the regularization parameter λ for each specific target example x.

3. Predicting Outputs with Datamodels

We now demonstrate how datamodels can be applied in the context of deep neural networks—specifically, we consider deep image classifiers trained on two standard datasets: CIFAR-10 (Krizhevsky, 2009) and Functional Map of the World (FMoW) (Koh et al., 2020) (see Appendix A.1 for more information on each dataset).

Goal. As discussed in Section 1, our goal is to construct a *collection* of datamodels for each dataset, with each datamodel predicting the model-training outcomes for a *specific* target example. Thus, for both CIFAR and FMoW, we fix a deep learning algorithm (architecture, hyperparams, etc.; see Appendix A.2), and aim to estimate a datamodel for each test set example *and* training set example. To this end, we will obtain $n = 10,000$ "test set datamodels" and $n = 50,000$ "training set datamodels" for CIFAR (each being a linear model g_θ parameterized by a vector $\theta \in \mathbb{R}^d$, for $d = 50,000$); as well as $n = 3,138$ test set datamodels and $n = 21,404$ training set datamodels for FMoW (again, parameterized by $\theta \in \mathbb{R}^d$ where $d = 21,404$).

3.1. Implementation details

Before applying datamodels to our two tasks of interest, we address a few remaining technical aspects of datamodel estimation:

Simultaneously estimating datamodels for a collection of target examples. Rather than repeat the entire datamodel estimation process for each target example x of interest separately, we can estimate datamodels for an entire *set* of target examples simultaneously through model reuse. Specifically, we train a large pool of models on subsets $S_i \subset S$ sampled from the distribution $\mathcal{D}_S$, and use the *same* models to compute outputs $f_{\mathcal{A}}(x; S_i)$ for each target example x.

Collecting a (sufficiently large) datamodel training set. Recent advances in fast neural network training (Page, 2018; Leclerc et al., 2022) allow us to train a wealth of models on α-subsets of each training set *very* efficiently (e.g., we can train 40,000 models/day on an $8 \times$ A100 GPU machine; see Appendix A.3 for details). Table 1 summarizes the models trained.

Estimating datamodels with LASSO. For both datasets considered (CIFAR-10 and FMoW), we estimate datamodels using LASSO (8), yielding a datamodel g_{θ_i} for each example x_i in the test and training sets.[1] In Appendix D.2, we discuss optimizing LASSO solvers for the scale of our problem instances.

[1] Datamodels for the training set require a small modification. See Appendix D.1.

	Subset size (α)			
Dataset	0.1	0.2	0.5	0.75
CIFAR-10	1,500,000	750,000	300,000	600,000
FMoW	–	375,000	150,000	300,000

Table 1: The number of models (ResNet-9 for CIFAR and ResNet-18 for FMoW) used to estimate datamodels for each dataset. All models are trained from scratch using optimized code (Leclerc et al., 2022). For example, each $\alpha = 0.5$ model on CIFAR-10 takes 17s to train (on a single A100 GPU) to 90% accuracy.

3.2. Linear datamodels predict deep network training

We now assess the quality of the estimated datamodels in terms of how well they predict model outputs on *unseen* subsets (i.e., fresh samples from $\mathcal{D}_S$). We refer to this process as *on-distribution* evaluation because we are interested in subsets S_i, sampled from the same distribution $\mathcal{D}_S$ as the datamodel training set, but *not* the exact ones used for estimation. (In fact, recall in Section 2 we explicitly held out m_{test} subset-output pairs for evaluation.)

We focus here on the collection of datamodels corresponding to the CIFAR-10 test set, i.e., a set of linear datamodel parameters $\{\theta_1, \ldots, \theta_n\}$ corresponding to examples $\{x_1, \ldots, x_n\}$ for $n = 10,000$ (analogous results for FMoW are in Appendix D.3). In Figure 1, aggregating over both datamodels $\{g_{\theta_j}\}_{j=1}^n$ and heldout subsets $\{S_i\}_{i=1}^m$, we compare datamodel predictions $\theta_j^\top \mathbf{1}_{S_i}$ to *expected* true model outputs $\mathbb{E}[f_{\mathcal{A}}(x_j; S_i)]$ (which we estimate by training 100 models on the same subset S_i and averaging their output on x_j). Our results show a near-perfect correspondence between datamodel predictions and ground truth. Thus, for a given target example x, we can accurately predict the outcome of "training a neural network on a random training subset and computing correct-class margin on x" (a process that involves hundreds of SGD steps on a non-convex objective) as a simple *linear* function of the characteristic vector of the subset.

Sample complexity. We next study the dependence of estimation on the size of the datamodel training set m. Specifically, we measure the *on-distribution* average mean-squared error (MSE) of a group of datamodels $\{\theta_i\}_{i=1}^n$ as

$$\frac{1}{2n} \sum_{j=1}^{n} \left(\mathbb{E}_{S_i \sim \mathcal{D}_S} \left[\left(\theta_j^\top \mathbf{1}_{S_i} - f_{\mathcal{A}}(x_j; S_i) \right)^2 \right] \right). \quad (9)$$

To evaluate (9), we replace the inner expectation with an empirical average, again using a heldout set of samples that was not used for estimation.

In Figure 2, we plot average MSE as a function of the number of trained models m. To put the results into context,

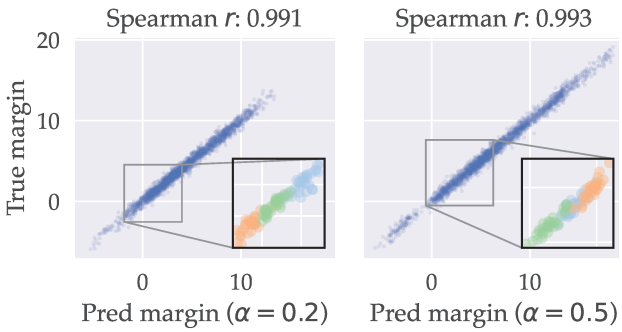


Figure 1: ***Linear*** **datamodels accurately predict margins.** Each point in the graphs above corresponds to a specific target example x_j *and* a specific held-out training set S_i from CIFAR-10. The y-coordinate represents the ground-truth margin $f_{\mathcal{A}}(x_j; S_i)$, averaged across T=100 models trained on S_i. The x-coordinate represents the *datamodel-predicted* value of the same quantity. We observe a strong linear correlation that persists even at the level of individual examples (the bottom-right panel shows the same data for three random target examples x_j color-coded by example). Corresponding plots for $\alpha = 0.1, 0.75$ and FMoW are in Figures D.3 and D.4.

we introduce the *optimal mean-squared error loss* (OPT), which is the MSE (9) with datamodel predictors $\theta_j^\top \mathbf{1}_{S_i}$ replaced by the optimal predictors $\mathbb{E}[f_{\mathcal{A}}(x_j; S_i)]$:

$$\text{OPT} = \frac{1}{2n} \sum_{j=1}^{n} \left(\mathbb{E}\left[\left(\mathbb{E}\left[f_{\mathcal{A}}(x_j; S_i) \right] - f_{\mathcal{A}}(x_j; S_i) \right)^2 \right] \right).$$

Note that OPT is independent of the estimator g_θ and measures only the inherent variance in the prediction problem, i.e., loss that will necessarily be incurred due only to inherent noise in deep network training.

In Appendix D.3, we further analyze the role of regularization and sparsity distribution for datamodels.

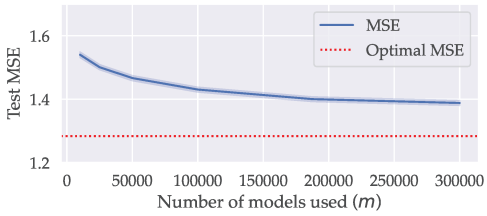


Figure 2: Average mean-squared error (9) for CIFAR-10 test set datamodels ($\alpha = 0.5$) as a function of the size of the datamodel training set m. The red line shows the optimal error based on inherent noise in model training.

4. Use Case: Counterfactual Prediction

So far, we have computed and evaluated datamodels entirely within a supervised learning framework. In particular, we constructed datamodels with the goal of predicting the outcome of training on *random* subsets of the training set (sampled from a distribution $\mathcal{D}_S$ (6)) and evaluating on a fixed target example x. Accordingly, for each target example x, we evaluated its datamodel g_θ by (a) sampling new random subsets S_i (from the same distribution); (b) training (a neural network) on each one of these subsets; (c) measuring correct-class margin on the target example x; and (d) comparing the results to the datamodel's predictions (namely, $g_\theta(S_i)$) in terms of *expected* mean-squared error (see (9)) over the distribution of subsets.

We will now go beyond this framework, and use datamodels to predict the outcome of training on *arbitrary* subsets of the training set. In particular, consider a fixed target example x with corresponding datamodel g_θ: for any subset S' of the training set S, we will use the *datamodel-predicted* outcome of training on S' and evaluating on x, i.e., $g_\theta(\mathbf{1}_{S'})$, in place of the *ground-truth* outcome $f_{\mathcal{A}}(x; S')$. Since S' is an *arbitrary* subset of the training set, it is "out-of-distribution" with respect to the distribution of fixed-size subsets $\mathcal{D}_S$ that we designed the datamodel to operate on. As such, using datamodel predictions in place of end-to-end-model training in this manner is not a priori guaranteed to work. Nevertheless, we will demonstrate (through two applications) that datamodels *can* in fact be effective proxies for end-to-end model training, even for such out-of-distribution subsets.

4.1. Measuring brittleness of individual predictions to training data removal

We first illustrate the utility of datamodels as a proxy for model training by using them to answer the question: *how brittle are model predictions to removing training data?* While all useful learning algorithms are data-dependent, cases where model behavior is sensitive to just a few data points are often of particular interest or concern (Broderick et al., 2021; Dwork et al., 2006). To quantify such sensitivity, we define the *data support* $\textsc{Support}(x)$ of a target example x as

$$\textsc{Support}(x) = \text{the smallest training subset } R \subset S \text{ such that classifiers trained on } S \setminus R \text{ misclassify } x \text{ on average.}^2 \quad (10)$$

Intuitively, examples with a small data support are the examples for which removing a small subset of the training data significantly changes model behavior, i.e., they are "brittle" examples by our criterion of interest. By computing $\textsc{Support}(x)$ for every image in the test set, we can

[2] That is, have negative expected margin (7).

thus get insight into how brittle model predictions are to removing training data.

Computing support. One way to compute $\textsc{Support}(x)$ for a given target example x would be to train several models on every possible subset of the training set S, then report the largest subset for which the example was misclassified on average—the complement of this set would be *exactly* $\textsc{Support}(x)$. However, exhaustively computing data support in this manner is simply intractable.

Using datamodels as a proxy for end-to-end model training provides an (efficient) alternative approach. Specifically, rather than training models on every possible subset of the training set, we can use datamodel-predicted outputs $g_\theta(S')$ to perform a *guided search*, and only train on subsets for which predicted margin on the target example is small. This strategy (described in detail in Appendix E.4) allows us to compute estimates of the data support while training only a handful of models per target example.

Results. We apply our algorithm to estimate $\textsc{Support}(x)$ for 300 random target examples in the CIFAR-10 test set. For over 90% of these 300 examples, we are able to *certify* that our estimated data support is *strictly larger than* the true data support $\textsc{Support}(x)$ (i.e., that we are not overestimating brittleness). We do this by training several models after excluding the estimated data support and checking that the target example is indeed misclassified on average.

We plot the distribution of estimated data support sizes in Figure 3. Around *half* of the CIFAR-10 test images have a datamodel-estimated data support comprising 250 images or less, meaning that removing a specific 0.4% of the CIFAR-10 training set induces misclassification. Similarly, 20% of the images had an estimated data support of less than *40 training images* (which corresponds to *0.08% of the training set*).

We also compare our estimates to some natural baselines, where we replace datamodel weights with another method to guide the search. All baselines yield much looser estimates of data support (Figure 3).

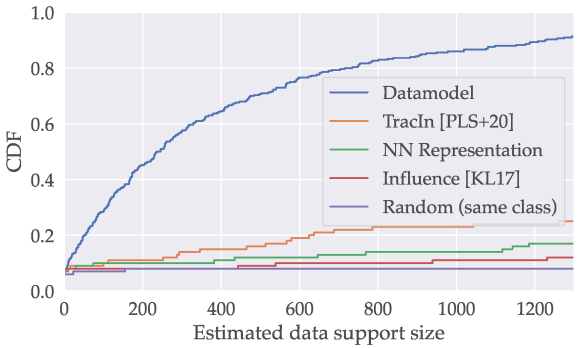


Figure 3: Characterizing brittleness. We use datamodels to estimate *data support* (i.e., the minimal set of training examples whose removal causes misclassification) for 300 random CIFAR-10 test examples, and plot the cumulative distribution of estimate sizes. Over 25% of examples can be misclassified by removing *less than 100* (example-specific) training images. Also, datamodels yield substantially better bounds on support size than baselines (see App. E.4.1 for details).

4.2. Predicting data counterfactuals

In the last section, a simple application of datamodels as a proxy for model training (on *arbitrary* subsets of the training set) enabled us to identify brittle predictions. In this section, we demonstrate another, more complex application of datamodels as a proxy for end-to-end training: predicting *data counterfactuals*.

For a fixed target example x, and a specific subset of the training set $R(x) \subset S$, a data counterfactual is the causal effect of removing the set of examples $R(x)$ on model outputs for x. In terms of our notation, this effect is precisely

$$\mathbb{E}\left[f_{\mathcal{A}}(x; S) - f_{\mathcal{A}}(x; S \setminus R(x))\right].$$

Such data counterfactuals can be helpful tools for finding brittle predictions (as in the previous section), estimating *group influence* (as done by (Koh et al., 2019) for linear models), and more broadly for understanding how training examples combine (through the lens of the model class) to produce test-time predictions.

Estimating data counterfactuals. Just as in the last section, we again use datamodels beyond the supervised learning regime in which they were developed. In particular, we predict the outcome of a data counterfactual as

$$g_\theta(\mathbf{1}_S) - g_\theta(\mathbf{1}_{S \setminus R(x)}),$$

where again g_θ is the datamodel for a given target example of interest. Since g_θ is a linear function in our case, the above *predicted data counterfactual* actually simplifies to

$$\theta^\top \mathbf{1}_S - \theta^\top \mathbf{1}_{S \setminus R(x)} = \theta^\top \mathbf{1}_{R(x)}.$$

Our goal now is to demonstrate that datamodels are useful predictors of data counterfactuals across a variety of removed sets $R(x)$. To accomplish this, we use a large set of target examples. For each such target example, we consider different subset sizes k; for each chosen k, we use a variety of heuristics (detailed in Appendix E.3) to select a set $R(x)$ comprising k "examples of interest." Specifically, we consider six values of k (the size of the removed subset) ranging from 10 to 1280 examples (i.e., $0.02\% - 2.6\%$ of

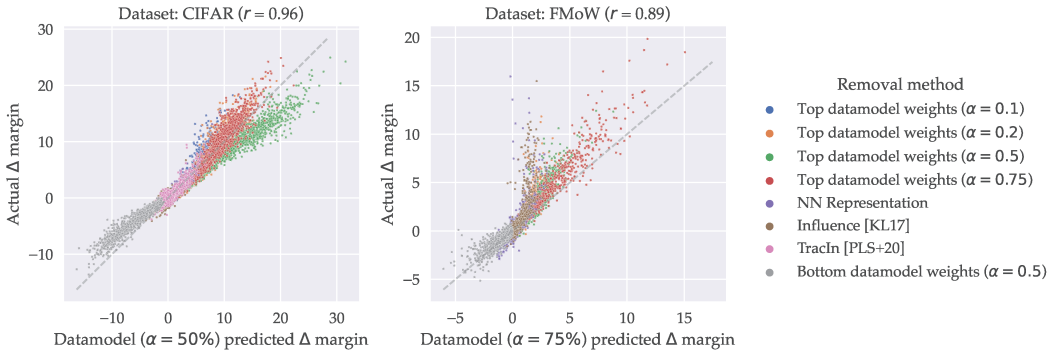


Figure 4: Datamodels predict data counterfactuals. Each point in the graphs above corresponds to a test example and a subset $R(x)$ of the original training set S, identified by a (color-coded) heuristic. The y-coordinate of each point represents the *true* difference, in terms of model output on x, between training on S, and training on $S \setminus R(x)$. The x-coordinate of each point represents the *datamodel-predicted* value of this quantity. We plot results for **(left)** CIFAR-10 and **(right)** FMoW. Datamodel predictions are predictive of the underlying counterfactuals, with Pearson coefficients r of 0.96/0.90 for CIFAR/FMoW respectively. See Appendix E for more experimental details and results.

the training set). Overall, the outcome of our procedure is, for each target example, both *true* and *datamodel-predicted* data counterfactuals for *30 different training subsets* $R(x)$ (six values of k and five different heuristics).

Results. In Figure 4, we plot datamodel-predicted data counterfactuals against true data counterfactuals, aggregating across all target examples x, values of k, and selection heuristics for $R(x)$. We find a strong correlation between these two quantities. In particular, across all factors of variation, predicted and true data counterfactuals have Spearman correlation $\rho = 0.98$ and $\rho = 0.94$ for CIFAR-10 and FMoW respectively. Even more strongly, the two quantities are correlated roughly *linearly*: we obtain (Pearson) correlations of $r = 0.96$ (CIFAR) and $r = 0.90$ (FMoW) between counterfactuals and their estimates on aggregate. In fact, correlations become even more pronounced when restricting to any single class of removed sets (i.e., any single hue in Figure 4).

Limits of datamodel predictions. Due to both estimation error (i.e., we might not have trained enough models to identify *optimal* linear datamodels) and misspecification error (i.e., the optimal datamodel might not be linear), we don't expect a perfect correspondence between datamodel-predicted outputs $g_\theta(\mathbf{1}_{S'})$ and true outputs $f(x; S)$ for *all* 2^d possible subsets of the training set. Indeed, this is part of the reason why we estimated datamodels for several values of α, only one of which is shown in Figure 4. The others, shown in Appendix E.10, still display strong correlation between true and predicted model outputs, but behave qualitatively differently than the ones shown above (i.e., each value of α is better or worse at predicting the outcomes of certain types of counterfactuals).

5. Other Use Cases

Beyond predicting data counterfactuals, datamodels unlock a multitude of mechanisms for understanding data, model predictions, and the connections between them. In the following two subsections, we briefly outline two such mechanisms—we defer a more in-depth discussion of them to Appendices F and G respectively.

5.1. Train-test similarity

We first explore datamodels as a mechanism for finding similar training examples to any given target example. In particular, for an example x with linear datamodel g_θ, we interpret the training examples corresponding to the highest-magnitude coordinates of θ as the "nearest neighbors" of x. Among other applications (see Appendix F), one way to leverage this functionality is as a way to study *train-test leakage*: cases where test examples are near-duplicates of training examples.

Train-test leakage in CIFAR. To find train-test leakage in CIFAR-10 we collect—for each target example x in the test set—the ten largest coordinates of the example's datamodel parameter vector. We use the corresponding ten training images as candidates for train-test leakage, and validate the candidates using Amazon Mechanical Turk to get a "annotation score" for every test example-candidate pair. Finally, we compute the "leakage score" for each test example as the highest annotation score (over all of its candidate train images). We use this leakage score as a proxy for whether or not the given image constitutes train-test leakage.

In Figure 5, we plot the distribution of leakage scores over the CIFAR-10 test set, along with random train-test pairs stratified by their annotation score. As the annotation score

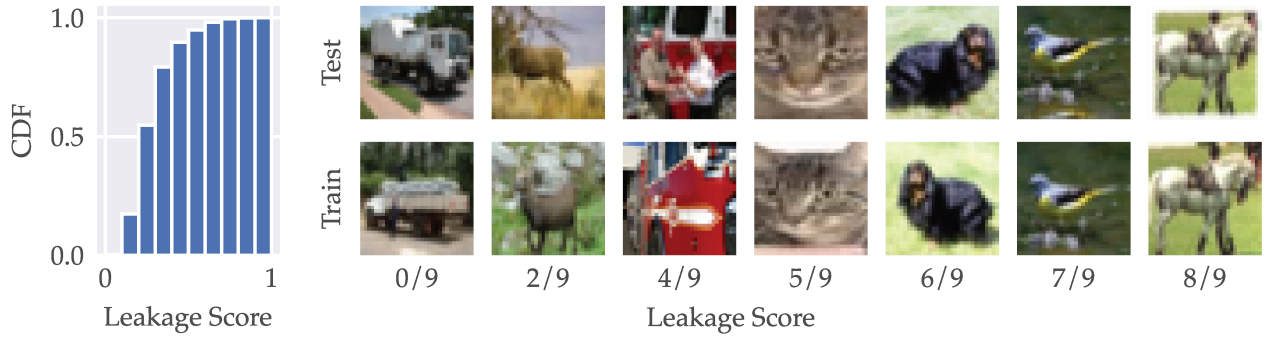


Figure 5: (Left) Histogram of the train-test leakage scores for each image of the CIFAR test set. **(Right)** Example train-test pairs stratified by their leakage score. A majority of annotators consider 10% of the test set as train-test leakage based on the identified candidates.

increases, pairs appear more likely to correspond to leakage (see Figure F.9 for more pairs). Furthermore, *roughly 10% of test set images were labeled as train-test leakage* by over half of the annotators that reviewed them.

In Appendix F.2.2, we perform a similar analysis on FMoW and show that datamodels outperform a natural baseline based on representation distance of a pretrained network.

5.2. Feature embedding

Finally, we explore another use case, enabled by an alternative perspective on datamodels. Consider a target example x with a corresponding *linear* datamodel g_θ, parameterized by a vector $\theta \in \mathbb{R}^d$, where d is the training set size. Instead of viewing the vector θ as just a parameter of the predictor g_θ, we cast it as a *feature representation* for the target example itself, i.e., a *datamodel embedding* of x into $\mathbb{R}^d$. Since the datamodel g_θ is a linear function of the presence of each point in the training set, each coordinate of this datamodel embedding corresponds to a weight for a specific training example. One can thus think of a datamodel embedding as a feature vector that represents a target example x in terms of how predictive each training example is of model behavior on x. By embedding an entire dataset of examples $\{x_i\}$ as a set of feature vectors $\{\theta_i \in \mathbb{R}^d\}$, we can hope to uncover structure in the set of examples by looking for structure in their datamodel embeddings, i.e., in the (Euclidean) space $\mathbb{R}^d$. We illustrate a simple example of finding latent structure by clustering the embeddings.

Spectral clustering. Datamodel embeddings induce a natural *similarity measure* between two examples x_1 and x_2, namely $d(x_1, x_2) = K(\theta_1, \theta_2)$, where $K(\cdot, \cdot)$ is any kernel function. Thus, given a set of examples $\{x_1, \ldots x_k\}$, we can compute an *adjacency matrix* $A \in \mathbb{R}^{k \times k}$, whose nodes are examples and whose edges are $A_{ij} = d(x_i, x_j)$. Such a graph unlocks a myriad of graph-theoretic tools for exploring datasets through the lens of datamodels, a complete exploration of which is beyond the scope of our work: instead, we focus on just one such tool, namely spectral clustering.

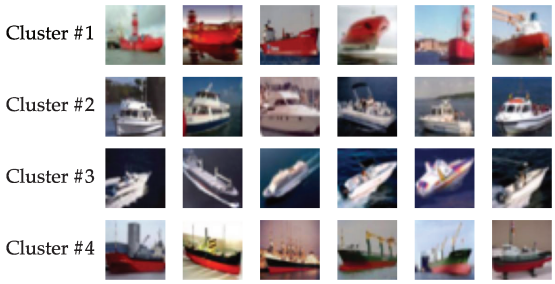


Figure 6: Spectral clustering on datamodel embeddings finds subpopulations. For each CIFAR-10 class, we first compute a similarity score between all datamodel embeddings, then run spectral clustering on the resulting matrix. Each cluster seems to correspond to a specific subpopulation with shared, distinctive visual features. See Figures G.1 and G.2 for more examples from other classes.

We run an off-the-shelf spectral clustering algorithm on the graph induced by the similarity matrix A above for the images in the CIFAR-10 test set. This procedure gives rise to (Figure 6) a simple unsupervised method for uncovering subpopulations in datasets.

In Appendix G.2, we analyze the embeddings further with principal components analysis (PCA) and show that the principal components capture meaningful features. We also highlight some advantages of datamodel embeddings over standard deep representations (e.g., based on the penultimate layer of a DNN).

6. Related Work

Datamodels build on (and indeed, are inspired by) a wide variety of prior work across machine learning and statistics, and these fields also suggest natural ways of improving or applying datamodels. We now briefly discuss the works *most* related to datamodeling (with a focus on disambiguation)—an extensive discussion of related work is found in Appendix H, and future work in Appendix I.

Most related to our work is the notion of *empirical influence* (Hampel et al., 2011), which measures the impact of removing a training point on a given parameter. As estimating influences naïvely is prohibitively expensive, Koh & Liang (2017) apply first-order influence function *approximation* to study both classical machine learning models and penultimate-layer embeddings from neural networks. However, their applicability to modern deep neural networks remain limited (Basu et al., 2021; Feldman & Zhang, 2020). A separate line of work proposes other approximation schemes for influences (Feldman & Zhang, 2020; Ghorbani & Zou, 2019; Jia et al., 2019). In particular, Feldman & Zhang (2020) define a subsampling-based influence estimator that trades off sample efficiency with accuracy (w.r.t. the true empirical influence). We can interpret their form of influence approximations as a specific instance of datamodeling (as discussed at length in Appendix H.1); however, our instantiation predicts model behavior significantly better (Appendix J).

Finally, image-level surrogate models such as LIME (Ribeiro et al., 2016; Lundberg & Lee, 2017; Sokol et al., 2019), fundamentally differ from datamodels. In particular, LIME and its counterparts study how the predictions of a *fixed* model changes as one varies the pixels of the target example. In contrast, datamodels hold the *target example* fixed, and instead study how the *training data* changes the predictions of the *learning algorithm*.

7. Conclusion

We present datamodeling, a framework for framing the output of model training as a simple function of the presence of each training data point. We show that a simple linear instantiation of datamodeling enables us to predict model outputs accurately, and facilitates a variety of applications.

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318. ACM, 2016.

Azizpour, H., Razavian, A. S., Sullivan, J., Maki, A., and Carlsson, S. Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence*, 2015.

Bagdasaryan, E., Poursaeed, O., and Shmatikov, V. Differential privacy has disparate impact on model accuracy. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Bansal, Y., Nakkiran, P., and Barak, B. Revisiting model stitching to compare neural representations. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Bartlett, P. L., Montanari, A., and Rakhlin, A. Deep learning: a statistical viewpoint. In *arXiv preprint arXiv:2103.09177*, 2021.

Barz, B. and Denzler, J. Do we train on test data? purging cifar of near-duplicates. In *Journal of Imaging*, 2020.

Basu, S., Pope, P., and Feizi, S. Influence functions in deep learning are fragile. In *International Conference on Learning Representations (ICLR)*, 2021.

Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. Analysis of representations for domain adaptation. In *Neural Information Processing Systems (NeurIPS)*, 2007.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

Bresler, G. and Nagaraj, D. A corrective view of neural networks: Representation, memorization and learning. In *Conference on Learning Theory (COLT)*, 2020.

Broderick, T., Giordano, R., and Meager, R. An automatic finite-sample robustness metric: Can dropping a little data change conclusions? In *Arxiv preprint arXiv:2011.14999*, 2021.

Brown, G., Bun, M., Feldman, V., Smith, A., and Talwar, K. When is memorization of irrelevant training data necessary for high-accuracy learning? In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

Carlini, N., Liu, C., Kos, J., Erlingsson, Ú., and Song, D. The secret sharer: Measuring unintended neural network memorization & extracting secrets. In *USENIX Security Symposium*, 2019.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

Carter, S., Armstrong, Z., Schubert, L., Johnson, I., and Olah, C. Activation atlas. *Distill*, 2019.

Charpiat, G., Girard, N., Felardos, L., and Tarabalka, Y. Input similarity from the neural network perspective. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Chatterjee, S. Learning and memorization. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Christie, G., Fendley, N., Wilson, J., and Mukherjee, R. Functional map of the world. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2018.

D'Amour, A., Heller, K. A., Moldovan, D., Adlam, B., Alipanahi, B., Beutel, A., Chen, C., Deaton, J., Eisenstein, J., Hoffman, M. D., Hormozdiari, F., Houlsby, N., Hou, S., Jerfel, G., Karthikesalingam, A., Lucic, M., Ma, Y., McLean, C. Y., Mincu, D., Mitani, A., Montanari, A., Nado, Z., Natarajan, V., Nielson, C., Osborne, T. F., Raman, R., Ramasamy, K., Sayres, R., Schrouff, J., Seneviratne, M., Sequeira, S., Suresh, H., Veitch, V., Vladymyrov, M., Wang, X., Webster, K., Yadlowsky, S., Yun, T., Zhai, X., and Sculley, D. Underspecification presents challenges for credibility in modern machine learning. In *Arxiv preprint arXiv:2011.03395*, 2020.

Daskalakis, C., Dikkala, N., and Panageas, I. Regression from dependent observations. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pp. 881–889, 2019.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. In *arXiv preprint arXiv:1708.04552*, 2017.

Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., and Naor, M. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2006.

Eberhardt, F. and Scheines, R. Interventions and causal inference. In *Philosophy of Science*, 2007.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Tran, B., and Madry, A. Adversarial robustness as a prior for learned representations. In *ArXiv preprint arXiv:1906.00945*, 2019.

Feldman, V. Does learning require memorization? a short tale about a long tail. In *Symposium on Theory of Computing (STOC)*, 2019.

Feldman, V. and Zhang, C. What neural networks memorize and why: Discovering the long tail via influence estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 2881–2891, 2020.

Friedman, J., Hastie, T., and Tibshirani, R. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 2010.

Gazagnadou, N., Gower, R. M., and Salmon, J. Optimal mini-batch and step sizes for saga. In *International Conference on Machine Learning (ICML)*, 2019.

Ghorbani, A. and Zou, J. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning (ICML)*, 2019.

Guérin, J., Gibaru, O., Thiery, S., and Nyiri, E. CNN features are also great at unsupervised classification. In *Arxiv preprint arXiv:1707.01700*, 2017.

Guyon, I. and Elisseeff, A. An introduction to variable and feature selection. In *Journal of Machine Learning Research (JMLR)*, 2003.

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. *Robust statistics: the approach based on influence functions*, volume 196. John Wiley & Sons, 2011.

Hanawa, K., Yokoi, S., Hara, S., and Inui, K. Evaluation of similarity-based explanations. In *International Conference on Learning Representations (ICLR)*, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Hooker, S., Courville, A., Dauphin, Y., and Frome, A. Selective brain damage: Measuring the disparate impact of model pruning. In *arXiv preprint arXiv:1911.05248*, 2019.

Huang, J., Zhang, T., and Metaxas, D. Learning with structured sparsity. In *Journal of Machine Learning Research (JMLR)*, 2011.

Jia, R., Dao, D., Wang, B., Hubis, F. A., Hynes, N., Gürel, N. M., Li, B., Zhang, C., Song, D., and Spanos, C. J. Towards efficient data valuation based on the shapley value. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, 2019.

Jiang, Y., Nagarajan, V., Baek, C., and Kolter, J. Z. Assessing generalization of sgd via disagreement. In *Arxiv preprint arXiv:2106.13799*, 2021.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, 2017.

Koh, P. W., Ang, K.-S., Teo, H. H., and Liang, P. On the accuracy of influence functions for measuring group effects. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Beery, S., et al. Wilds: A benchmark of in-the-wild distribution shifts. *arXiv preprint arXiv:2012.07421*, 2020.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

Krizhevsky, A. Learning multiple layers of features from tiny images. In *Technical report*, 2009.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Leclerc, G., Salman, H., Ilyas, A., Vemprala, S., Engstrom, L., Vineet, V., Xiao, K., Zhang, P., Santurkar, S., Yang, G., et al. 3db: A framework for debugging computer vision models. In *arXiv preprint arXiv:2106.03805*, 2021.

Leclerc, G., Ilyas, A., Engstrom, L., Park, S. M., Salman, H., and Madry, A. ffcv. https://github.com/libffcv/ffcv/, 2022.

Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., and Carlini, N. Deduplicating training data makes language models better. In *Arxiv preprint arXiv:2107.06499*, 2021.

Li, T., Levina, E., and Zhu, J. Prediction models for network-linked data. In *The Annals of Applied Statistics*, 2019.

Lundberg, S. and Lee, S.-I. A unified approach to interpreting model predictions. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Mania, H., Miller, J., Schmidt, L., Hardt, M., and Recht, B. Model similarity mitigates test set overuse. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9993–10002, 2019.

Martinsson, P. and Tropp, J. Randomized numerical linear algebra: foundations & algorithms. In *arXiv preprint arXiv:2002.01387*, 2020.

Massias, M., Gramfort, A., and Salmon, J. Celer: a fast solver for the lasso with dual extrapolation. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

Nakkiran, P. and Bansal, Y. Distributional generalization: A new kind of generalization. In *Arxiv preprint arXiv:2009.08092*, 2020.

Neal, R. *Bayesian Learning for Neural Networks*. Springer, 1996.

Olah, C., Mordvintsev, A., and Schubert, L. Feature visualization. In *Distill*, 2017.

Owen, G. Multilinear extensions of games. In *Management Science*, 1972.

Page, D. CIFAR-10 Fast. GitHub Repository, 10 2018. URL https://github.com/davidcpage/cifar10-fast.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. In *Journal of Machine Learning Research*, volume 12, pp. 2825–2830, 2011.

Pezeshkpour, P., Jain, S., Wallace, B. C., and Singh, S. An empirical comparison of instance attribution methods for nlp. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

Pruthi, G., Liu, F., Sundararajan, M., and Kale, S. Estimating training data influence by tracing gradient descent. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Ribeiro, M. T., Singh, S., and Guestrin, C. "why should i trust you?": Explaining the predictions of any classifier. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

Robertson, T., Wright, F., and Dykstra, R. L. *Order Restricted Statistical Inference*. Wiley Series in Probability and Statistics, 1988.

Rosenfeld, E., Winston, E., Ravikumar, P., and Kolter, Z. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning (ICML)*, 2020.

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. Design and analysis of computer experiments. In *Statistical Science*, volume 4, pp. 409–423. Institute of Mathematical Statistics, 1989. URL http://www.jstor.org/stable/2245858.

Sagawa, S., Raghunathan, A., Koh, P. W., and Liang, P. An investigation of why overparameterization exacerbates spurious correlations. In *International Conference on Machine Learning*, pp. 8346–8356. PMLR, 2020.

Sellam, T., Yadlowsky, S., Wei, J., Saphra, N., D'Amour, A., Linzen, T., Bastings, J., Turc, I., Eisenstein, J., Das, D., Tenney, I., and Pavlick, E. The multiberts: Bert reproductions for robustness analysis. In *Arxiv preprint arXiv:2106.16163*, 2021.

Shapley, L. Notes on the n-person game—ii: The value of an n-person game, the rand corporation, the rand corporation. In *Research Memorandum*, 1951.

Shawe-Taylor, J. and Cristianini, N. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Sokol, K., Hepburn, A., Santos-Rodriguez, R., and Flach, P. blimey: Surrogate prediction explanations beyond lime. In *Arxiv preprint arXiv:1910.13016*, 2019.

Spearman, C. The proof and measurement of association between two things. In *The American Journal of Psychology*, 1904.

Wang, T., Zeng, Y., Jin, M., and Jia, R. A unified framework for task-driven data quality management. In *ArXiv preprint arXiv:2106.05484*, 2021.

Wong, E., Santurkar, S., and Madry, A. Leveraging sparse linear layers for debuggable deep networks. In *International Conference on Machine Learning (ICML)*, 2021.

Xiao, H., Xiao, H., and Eckert, C. Adversarial label flips attack on support vector machines. In *European Conference on Artificial Intelligence (ECAI)*, 2012.

Yeh, C.-K., Kim, J. S., Yen, I. E. H., and Ravikumar, P. Representer point selection for explaining deep neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR)*, 2016.

Zhang, C., Bengio, S., Hardt, M., Mozer, M. C., and Singer, Y. Identity crisis: Memorization and generalization under extreme overparameterization. In *International Conference on Learning Representations (ICLR)*, 2020.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. The unreasonable effectiveness of deep features as a perceptual metric. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

Zhong, R., Ghosh, D., Klein, D., and Steinhardt, J. Are larger pretrained language models uniformly better? Comparing performance at the instance level. In *Findings of the Association for Computational Linguistics (Findings of ACL)*, 2021.

Appendices

A. Experimental Setup

A.1. Datasets

CIFAR-10. We use the standard CIFAR-10 dataset (Krizhevsky, 2009).

FMoW. FMoW (Christie et al., 2018) is a land use classification dataset based on satellite imagery. WILDS (Koh et al., 2020) uses a subset of FMoW and repurposes it as a benchmark for out-of-distribution (OOD) generalization; we use same the variant (presized to 224x224, single RGB image per example rather than a time sequence). We perform our analysis only on the in-distribution train/test splits (e.g. overlapping years) as our focus is not on OOD settings. Also, we limit our data to the year 2012. (These restrictions are only for convenience, and our framework can easily extend and scale to more general settings.)

Properties of both datasets are summarized in Table A.1.

Table A.1: Properties of datasets used.

Dataset	Classes	Size (Train/Test)	Input Dimensions
CIFAR-10	10	50,000/10,000	$3 \times 32 \times 32$
FMoW	62	21,404/3,138	$3 \times 224 \times 224$

A.2. Models and hyperparameters

CIFAR-10. We use a ResNet-9 variant from Kakao Brain[3] optimized for fast training. The hyperparameters (Table A.2) were chosen using a grid search. We use the standard batch SGD. For data augmentation, we use random 4px random crop with reflection padding, random horizontal flip, and 8×8 CutOut (DeVries & Taylor, 2017).

For counterfactual experiments with ResNet-18 (Figure E.6), we use the standard variant (He et al., 2016).

FMoW. We use the standard ResNet-18 architecture (He et al., 2016). The hyperparameters (Table A.2) were chosen using a grid search, including over different optimizers (SGD, Adam) and learning rate schedules (step decay, cyclic, reduce on plateau). As in Koh et al. (2020), we do not use any data augmentation. Unlike prior work, we do not initialize from a pre-trained ImageNet model; while this results in lower accuracy, this allows us to focus on the role of the FMoW dataset in isolation.

Table A.2: Hyperparameters for used model class.

Dataset	Initial LR	Batch Size	Epochs	Cyclic LR Peak Epoch	Momentum	Weight Decay
CIFAR-10	0.5	512	24	5	0.9	5e-4
FMoW	0.4	512	15	6	0.9	1e-3

Performance. In Table A.3, we show for each dataset the accuracies of the chosen model class (with its specific hyperparameters), across different values of α.

A.3. Training infrastructure

Computing resources. We train our models on a cluster of machines, each with 9 NVIDIA A100 GPUs and 96 CPU cores. We also use half-precision to increase training speed.

Data loading. We use FFCV (Leclerc et al., 2022), which removes the data loading bottleneck for smaller models and allows us achieve a throughput of over 5,000 CIFAR-10 models a day *per* GPU.

Data processing. Our datamodel estimation uses (the characteristic vectors) of training subsets and model outputs (margins) on train and test sets. Hence, we do not need to store any model checkpoints, as it suffices to store the training subset and the model outputs after evaluating at the end of training. In particular, training subsets and model outputs can be stored

[3] https://github.com/wbaek/torchskeleton/blob/master/bin/dawnbench/cifar10.py

Table A.3: Accuracies for our chosen model classes on CIFAR-10 and FMoW across varying α.

	Accuracy (%)	
Subset size (α)	CIFAR-10	FMoW
1.0	93.00	33.76
0.75	91.77	31.16
0.5	89.61	25.97
0.2	81.62	14.70
0.1	71.60	N/A

as $m \times n$ or $m \times d$ matrices, with one row for each model instance and one column for each train or test example. All subsequent computations only require the above matrices.

B. Selecting Output Function to Model

In this section, we outline a heuristic method for selecting the output function $f_{\mathcal{A}}(x; S)$ to model. The heuristic is neither sufficient *nor* necessary for least-squares regression to work, but may provide some signal as to which output may yield better datamodels.

The first problem we would like to avoid is "output saturation," i.e., being unable to learn a good datamodel due to insufficient variation in the output. This effect is most pronounced when we measure model correctness: indeed, over 30% of the CIFAR-10 test set is either always correct or always incorrect over all models trained, making datamodel estimation impossible. However, this issue is not unique to correctness. We propose a very simple test inspired by the idealized ordinary least squares model to measure how normally distributed a given type of model output is.

Normally distributed residuals. In the idealized ordinary least squares model, the observed outputs $f_{\mathcal{A}}(x; S)$ would follow a normal distribution with fixed mean $(\theta^\star)^\top \mathbf{1}_S$ and unknown variance, where $\theta^\star$ is the true parameter vector. Although we cannot guarantee this condition, we can measure the "normality" of the outputs (again, for a single fixed subset), with the intuition that the more normal the observed outputs are, the better a least-squares regression will work. Hence, compare different output functions by estimating the noise distribution of datamodels given each choice of output function. We leverage our ability—in contrast to typical settings for regression— to sample multiple response variables $f_{\mathcal{A}}(x; S)$ for a fixed S (by retraining several models on the same data and recording the output on a fixed test example).

In Figure B.1, we show the results of normality test for residuals arising from different choices of $f_{\mathcal{A}}(\cdot; S)$: correctness function, confidence on the correct class, cross-entropy loss, and finally correct-class margin[4]. Correct-class margins is the only choice of $f_{\mathcal{A}}(\cdot; S)$ where the p-values are distributed nearly uniformly, which is consistent with the outputs being normally distributed. Hence, we choose to use the correct-class margins as the dependent variable for fitting our datamodels.

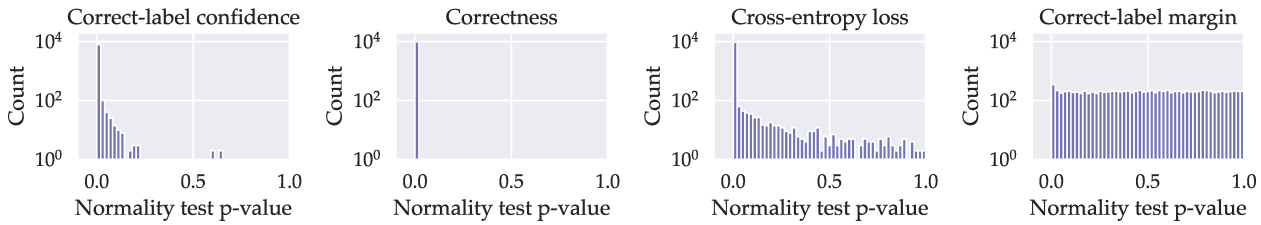


Figure B.1: Correct-class margins are close to normally distributed. For each choice of output function $f_{\mathcal{A}}$: we (a) fix a random subset $S' \sim \mathcal{D}_S$, where S is the CIFAR-10 train set, (b) train 200 models on S' and evaluate them on the entire CIFAR-10 test set, and (c) for each image x_i in the test set, calculate a p-value for rejecting the normality of $f_{\mathcal{A}}(x_i; S')$. We plot a histogram of these p-values above. For every output function other than correct-class margin, almost every test is rejected, whereas for margins the distribution of p-values is uniform across $[0, 1]$, which is consistent with the null hypothesis (normality).

[4]Correct-class margin is the difference between the correct-class logit and the highest incorrect-class logit; it is unbounded by definition, and its sign indicates the correctness of the classification.

C. The Role of the Subsampling fraction α

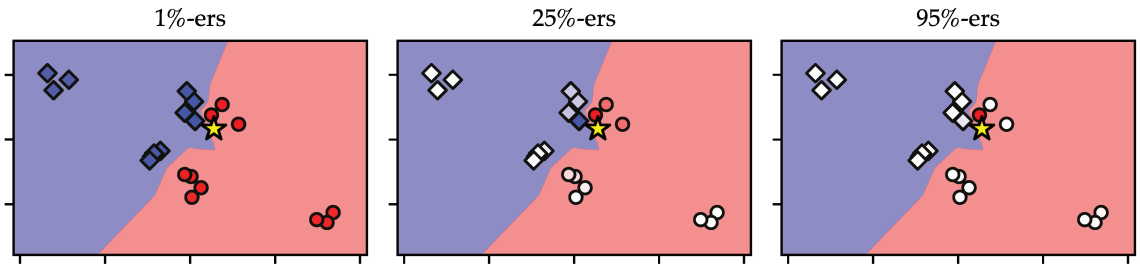


Figure C.1: Datamodels capture data relationships at varying levels of granularity. We illustrate the role of *subsampling fraction* α of datamodels by considering a nearest-neighbor classifier in two dimensions. In the datamodel for the target example ($\star$, yellow), the red (blue) examples have positive (negative) weights, with the shade indicating the magnitude. At large values of α (right), the model identifies only local relationships. Meanwhile, at small values of α (left), we can identify more global relationships, but at the cost of granularity. Intermediate values of α (middle) provide a smooth tradeoff between these two regimes.

We have used datamodels estimated using several choices of the subsampling fraction α, and saw that the value of α corresponding to the most useful datamodels can vary by setting. In particular, the visualizations in Figure F.2 suggest that datamodels estimated with lower α (i.e., based on smaller random training subsets) find train-test relationships driven by larger groups of examples (and vice-versa). Here, we explore this intuition further using thought experiment, toy example, and numerical simulation to intuit how different choices of α can lead to substantively different datamodels.

First, consider the task of estimating a datamodel for a prototypical image x—for example, a plane on a blue sky background. As $\alpha \to 1$, the sets S_i sampled from $\mathcal{D}_S$ are relatively large—if these sets have enough other images of planes on blue skies, we will observe little to no variation in $f_{\mathcal{A}}(x; S_i)$, since any predictor trained on S_i will perform very well on x. As a result, a datamodel for x estimated with $\alpha \to 1$ may assign very little weight to any particular image, even if in reality their *total* effect is actually significant.

Decreasing α, then, offers a solution to this problem: in particular, we allow the datamodel to observe cases where entire *groups* of training examples are not present, and *re-distribute* the corresponding effect back to the constituents of the group (i.e., assigning them all a share of the weight).

Now, consider a highly atypical yet correctly classified example, whose correctness relies on just the presence of just a few images from the training set. In this setting, datamodels estimated with a small value of α may be unable to isolate these training points, since they will constantly distribute variation in $f_{\mathcal{A}}(x; S_i)$ among a large group of non-present images. Meanwhile, using a large value of α allows the estimated datamodel to place weight on the correct training images (since x will be classified correctly until some of the important training images are not present in S_i).

In line with this intuition, decreasing α in Figure C.1 (i.e., moving from right to left) leads to datamodels that assign weight to increasingly large neighborhoods of points around the target input. This example and the above reasoning lead us to hypothesize that larger (respectively, smaller) α are better-suited to cases where model predictions are driven by smaller (respectively, larger) groups of training examples. In next subsection, we perform a more quantitative analysis of the role of α, this time by studying an underdetermined linear regression model on data that is organized into overlapping *subpopulations*.

C.1. Linear Regression Simulation

At a high level, our intuition for the subsampling fraction[5] α is that datamodels estimated with higher α tend to detect more *local* effects (i.e., those driven by smaller groups of examples, such as near-duplicates or small subpopulations), while those estimated with lower α detect more *global* effects (i.e., those driven by larger groups of images, such as large subpopulations or subclass biases). To solidify and corroborate this intuition about α, we analyze a basic simulated setting.

[5] See Section 2 for definition.

Setup. We consider an underdetermined linear regression model operating on n data points with d binary features, i.e., $x_i \in \{0,1\}^d, y_i \in \mathbb{R}$. Let $X \in \mathbb{R}^{n \times d}$ and $y \in \mathbb{R}^n$ denote their matrix and vector counterparts. S is training set consisting of these n samples, and we use an equally sized held-out set S_V for evaluation.

The feature coordinates are distributed as Bernoulli variables of varying frequency:

$$x_{ik} \sim \text{Bernoulli}(p_k) \text{ for } 1 \leq i \leq n, \tag{11}$$
$$\text{where } p_k \in \left\{ \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, \frac{4}{10}, \frac{5}{10} \right\} \text{ for } 1 \leq k \leq d.$$

Each feature $k \in [d]$ naturally defines a *subpopulation* S_k, the group of training examples with feature k active, i.e., $S_k := \{x_i \in S : x_{ik} = 1\}$. Features with *lower* (resp. *higher*) frequency p_k are intended to capture more *local* (resp. more *global*) effects.

The observed labels are generated according to a linear model $y := X\boldsymbol{w} + \mathcal{N}(0, \epsilon)$, where $\boldsymbol{w}$ is the true parameter vector and $\epsilon > 0$ is a constant. We generate samples with $d = 150, n = 125$ and use linear regression[6] to estimate $\boldsymbol{w}$.

Now, to use datamodels to analyze the above "training process" of fitting a linear regression model, we will model the output function $f_{\mathcal{A}}(;)$ given by the prediction of the linear model at point x_j when w is estimated with samples $S \subset S$, e.g.

$$f_{\mathcal{A}}(x_j; S) = (X_S^\top (X_S X_S^\top)^{-1} y_S) \cdot x_j \tag{12}$$

We generate $m = 1,000,000$ subsampled training subsets[7] along with their evaluations, and use ordinary least squares (OLS) to fit the datamodels. (Note that the use of OLS here is separate from the use of linear regression above as the original model class.)

Analysis. Our hypothesis is that datamodels estimated with lower (resp. higher) α are better at detecting the effect of features of higher (resp. lower) frequency. To test this, we estimate datamodels for the entire test set (stacking them into a matrix $\boldsymbol{\Theta} \in \mathbb{R}^{n \times n}$, where $\boldsymbol{\Theta}_{\cdot,j}$ is the datamodel for x_j) . We do this for varying values of $\alpha \in (0,1)$, and evaluate how well each set of datamodels predicts the effects of features across different frequencies. First, to evaluate a datamodel on some feature k, we can compare the following two quantities for different test examples $x_j \in S_V$:

(a) The *actual* effect of removing the subpopulation S_k on x_j, i.e., $f_{\mathcal{A}}(x_j; S) - f_{\mathcal{A}}(x_j; S \setminus S_k)$,

(b) The *datamodel-predicted* effect of removing S_k, i.e., $\sum_{x_i \in S} \boldsymbol{\Theta}_{ij} \cdot \mathbf{1}\{x_i \in S_k\}$.

To quantify the predictiveness of the datamodel at frequency p, we compute the *Pearson correlation* between the above two quantities over all features k with frequency p and all test examples. We repeat this evaluation varying p and the datamodel (varying α). According to our intuition, for features k with lower (resp. higher) frequency p_k, this correlation should be maximized at higher (resp. lower) values of α, where the datamodels capture more local (resp. global) effects. Figure C.2 accurately reflects this intuition: more local (i.e., less frequent) features are best detected at higher α.

[6] As the system is underdetermined, we use the pseudoinverse of X to find the solution with the smallest norm.

[7] Large sample size make sampling error negligible.

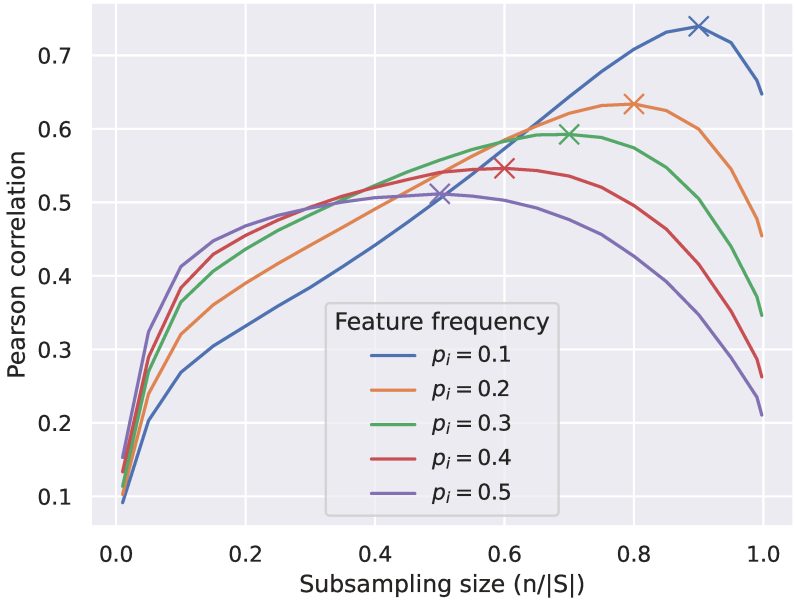


Figure C.2: The role of the subsampling fraction α in a simulated linear model. The data consists of d-dimensional binary vectors x_i, which comprise (overlapping) *subpopulations* S_k defined by a shared feature k, and their corresponding labels are generated according to a linear model, i.e. $y := X\boldsymbol{w} + \mathcal{N}(0, \epsilon)$. We estimate datamodels using various α, and measure their ability to detect features at different frequencies p. To quantify latter, we compute the *Pearson correlation* between i) the *actual effect* of removing the subpopulation S_k on a test example and ii) the *datamodel-predicted* effect, across all features with frequency p. Each line in the above plot coresponds to features of a particular frequency p, and shows the correlation (y-axis) while varying the datamodel (α, x-axis). Consistent with our intuition, we observe that higher (resp. lower) values of α are better at detecting less (resp. more) frequent features, i.e. more local (resp. global) effects.

D. Regression

D.1. Computing datamodels for training examples.

Recall that the target example x for which we estimate a datamodel can be arbitrary. In particular, x could itself be a training example—indeed, as we mention above, our goal is to estimate a datamodel for every image in the FMoW and CIFAR-10 test *and* training sets. When x is in the training set, however, we slightly alter the datamodel estimation objective (8) to exclude training sets S_i containing the target example:

$$\theta = \min_{w \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}\{x \notin S_i\} \cdot \left(w^\top \mathbf{1}_{S_i} - f_{\mathcal{A}}(x; S_i)\right)^2 + \lambda \|w\|_1. \tag{13}$$

D.2. Solver details

As mentioned in Section 2, we construct datamodels by running ℓ_1-regularized linear regression, predicting correct-class margins from characteristic vectors, or *masks*, $\mathbf{1}_{S_i}$. The resulting optimization problem is rather large: for example, estimating datamodels for $\alpha = 50\%$ requires running LASSO with a covariate matrix X of size $50,000 \times 300,000$, which corresponds to about 60GB of data; for $\alpha = 10\%$, datamodels this increases five folds as there are 1.5 million models. Moreover, we need to solve up to $60,000$ regression problems (one datamodel each train / test example). The large-scale nature of our estimation problem rules out off-the-shelf solutions such as scikit-learn (Pedregosa et al., 2011), GLMNet (Friedman et al., 2010), or Celer (Massias et al., 2018), all of which either runs out of memory or does not terminate within reasonable time.

Note that solving large linear systems efficiently is an area of active research ((Martinsson & Tropp, 2020)), and as a result we anticipate that datamodel estimation could be significantly improved by applying techniques from numerical optimization. In this paper, however, we take a rather simple approach based on the SAGA algorithm of (Gazagnadou et al., 2019). Our starting point is the GPU-enabled implementation of Wong et al. (2021)—while this implementation terminated (unlike the CPU-based off-the-shelf solutions), the regressions are still prohibitively slow (i.e., on the order of several GPU-hours per single datamodel estimation). To address this, we make the following changes:

Fast dataloading. The first performance bottleneck turns out to be in dataloading. More specifically, SAGA is a minibatch-based algorithm: at each iteration, we have to read B masks (50,000-dimensional binary vectors) and B outputs (scalars) and move them onto the GPU for processing. If the masks are read from disk, I/O speed becomes a major bottleneck—on the other hand, if we pre-load the entire set of masks into memory, then we are not able to run multiple regressions on the same machine, since each regression will use essentially the entire RAM disk. To resolve this issue, we use the FFCV library (Leclerc et al., 2022) for dataloading—FFCV is based on memory mapping, and thus allows for multiple processes to read from the same memory (combining the benefits of the two aforementioned approaches). FFCV also supports batch pre-loading and parallelization of the data processing pipeline out-of-the-box—adapting the SAGA solver to use FFCV cut the runtime significantly.

Simultaneous outputs. Next, we leverage the fact that the SAGA algorithm is trivially parallelizable across different instances (sharing the same input matrix), allowing us to estimate *multiple* datamodels at the same time. In particular, we estimate datamodels for the entire test set in one pass, effectively cutting the runtime of the algorithm by the test set size (e.g., 10,000 for CIFAR-10).

Optimizations. In order to parallelize across test examples, we need to significantly reduce the GPU memory footprint of the SAGA solver. We accomplish this through a combination of simple code optimization (e.g., using in-place operations rather than copies) as well as writing a few custom CUDA kernels to speed up and reduce the memory consumption of algorithms such as soft thresholding or gradient updating.

Experimental details. For each dataset considered, we chose a maximum λ: 0.01 for CIFAR-10 test, 0.1 for CIFAR-10 trainset, and 0.05 for FMoW datamodels. Next, we chose $k = 100$ logarithmically spaced intermediate values between $(\lambda/100, \lambda)$ as the regularization path. We ran one regression per intermediate λ, using $m - 50,000$ samples (where m is as in the table in Figure 1 (right)) to esitmate the parameters of the model and the remaining $50,000$ samples as a validation set. For each image in the test set, we select the λ corresponding to the best-performing predictor (on the heldout set) along the regularization path. We then *re-run* the regression once more using these optimal λ values and the full set of m samples.

D.3. Additional analyses

The role of regularization. We study the effect of the regularization parameter λ (cf. (8) and (13)) on datamodel performance. In Figure D.2 we plot the variation in average MSE, on both *in-sample* subsets (i.e., the exact subsets that we used to optimize (8)) and unseen subsets, as we vary the regularization parameter λ in (8). We find that—as predicted by classical learning theory—setting $\lambda = 0$ leads to *overfit* datamodels, i.e., estimators g_θ that perform well on the exact subsets that were used to estimate them, but are poor output predictors on *new* subsets S_i sampled from $\mathcal{D}_S$. (In fact, using $m = 300,000$ trained models with $\lambda = 0$ results in higher MSE than using only $m = 10,000$ with optimal λ, i.e., the left-most datapoint in Figure 1).

Sparsity distribution. In Figure D.1, we compare the sparsity distribution of datamodels across different values of α.

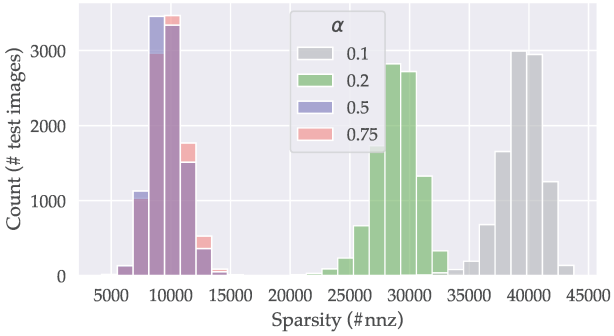


Figure D.1: Sparsity distribution of different datamodels. Above shows the distribution of datamodel sparsity over test examples on CIFAR-10, compared across different α; sparsity decreases with higher α.

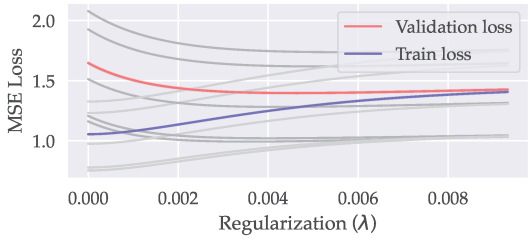


Figure D.2: The role of regularization. Average in-sample and out-of-sample MSE (i.e., (9)) for datamodels on CIFAR-10 estimated by optimizing the regularized least-squares objective (8) for varying λ.

Omitted results.

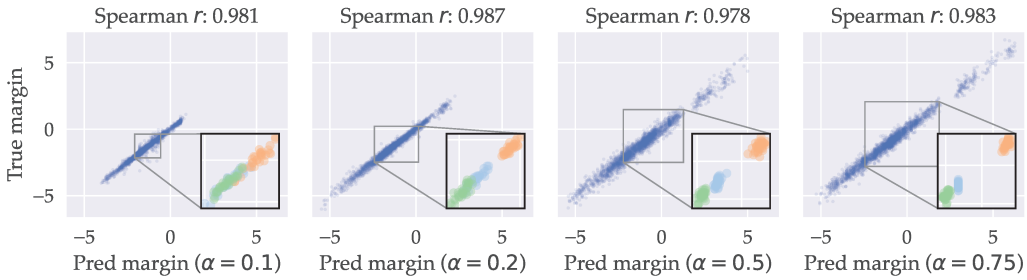


Figure D.3: Identical results to Figure 1 for FMoW.

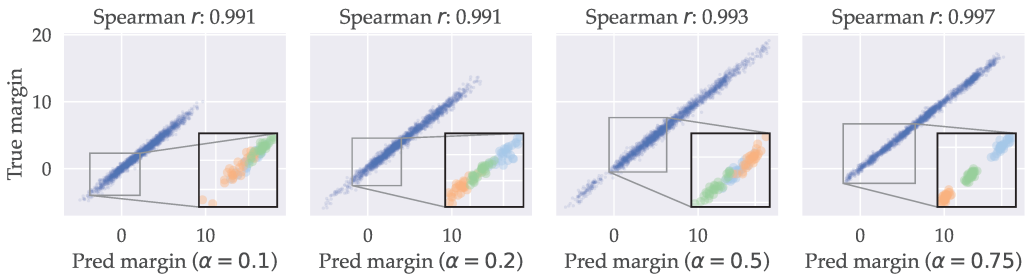


Figure D.4: Identical results to Figure 1 for CIFAR-10 for all values of α.

E. Datamodels for Counterfactual Prediction

E.1. General setup

Sample selection. For all of our counterfactual experiments, we use a random sample of the respective test datasets. We select at random 300 test images for CIFAR-10 (class-balanced; 30 per class) and 100 test images for FMoW. For the CIFAR-10 baselines, we consider counterfactuals for a 100 image subset of the 300.

Size of counterfactuals. For CIFAR-10, we remove top $k = \{10, 20, 40, 80, 160, 320, 640, 1280\}$ images and bottom $k = \{20, 40, 80, 160, 320\}$ where applicable. For FMoW, we remove top and bottom $k = \{10, 20, 40, 80, 160, 320, 640\}$.

Reducing noise by averaging. Each counterfactual (i.e., training models on a given training set S') is evaluated over T trials to reduce the variance that arises purely from non-determinism in model training. We use $T = 20$ for CIFAR-10 and FMoW, and $T = 10$ for CIFAR-10 baselines. In Appendix E.7, we show that using sufficiently high T is important for reducing noise.

Control values. To calculate the actual effects in all of our counterfactual evaluation, we need control values $\mathbb{E}[f_{\mathcal{A}}(x; S)]$ for the "null," i.e, margins when trained on 100% of the data. We estimate this by averaging over models on trained on the full training set (10,000 for CIFAR-10 and 500 for FMoW).

E.2. Baselines

We describe the baseline methods used to generate data support estimates and counterfactuals. Each of the methods gives a way to select training examples that are most similar or influential to a target example. As in prior work (Hanawa et al., 2021; Pezeshkpour et al., 2021), we consider a representative set of baselines spanning both methods based on representation similarity and gradient-based methods, such as influence functions.

Representation distance. We use ℓ_2 distances in the penultimate layer's representation to rank the training examples in order of similarity to the target test example. We also evaluated dot product, cosine, and mahalanobis distances, but they did not show much variation in their counterfactual effects.[8]

In order to more fairly compare with datamodels—so that we can disentangle the variance reduction from using many models and the additional signal captured by datamodels—we also averaged up to 1000 models' representation distances[9], but this had no discernible difference on the size of the counterfactual effects.

Influence functions. We apply the influence function approximation introduced in (Koh & Liang, 2017). In particular, we use the following first-order approximation for the influence of z on the loss L evaluated at z_{test}:

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta \ell(z_{\text{test}}, \widehat{\theta})^\top H_{\widehat{\theta}}^{-1} \nabla_\theta \ell(z, \widehat{\theta})$$

where $\widehat{\theta}$ is the empirical risk minimizer on the training set and H is the Hessian of the loss. The influence here is just the dot product of gradients, weighted by the Hessian. We approximate these influence values by using the methods in (Koh & Liang, 2017) and as implemented (independently) in `pytorch-influence-functions`.[10] As in (Koh & Liang, 2017), we take a pretrained representation (of a ResNet-9 model, same as that modeled by our datamodels), and compute approximate influence functions with respect to only the parameters in the last linear layer.

TracIn. Pruthi et al. (2020) define an alternative notion of influence: the influence of a training example z on a test example z' is the total change in loss on z' contributed by updates from mini-batches containing z—intuitively, this measures whether gradient updates from z are helpful to learning example z'. They approximate this in practice with `TracInCP`, which considers checkpoints $\theta_{t_1}, ..., \theta_{t_k}$ across training, and sums the dot product of the gradients at z and z' at each checkpoint:

$$\texttt{TracInCP}(z, z') = \sum_{i=1}^{k} \eta_i \nabla_\theta \ell(z, \theta_{t_i}) \cdot \nabla_\theta \ell(z', \theta_{t_i})$$

One can view `TracInCP` as a variant of the gradient dot product, but averaged over models at different epochs) and

[8] With the exception of dot product, which performs poorly due to lack of normalization; this is consistent with the findings in Hanawa et al. (2021).

[9] We simply average the ranks from each model, but there are potentially better ways to aggregate them.

[10] https://github.com/nimarb/pytorch_influence_functions

weighted by the learning rate η_i.

Random baseline. We also consider a random baseline of removing examples from the same class.

E.3. Removed sets for counterfactual evaluation

We generate the removed sets $R(x)$ as follows:

(a) setting $R(x)$ to be the nearest k training examples to the target example x using the baseline methods described in Appendix E.2: *influence function* (Koh & Liang, 2017), *TracIn* (Pruthi et al., 2020), or *distance in pre-trained representation space* (Bengio et al., 2013).

(b) setting $R(x)$ to be the *maximizer* of the datamodel-predicted counterfactual, i.e.,

$$R(x) = \arg\max_{|R|=k} g_\theta(S) - g_\theta(S \setminus R) = \arg\max_{|R|=k} \theta^\top \mathbf{1}_R.$$

(Note that since our datamodels are linear, this simplifies to excluding the training examples corresponding to the top k coordinates of the datamodel parameter θ.)

(c) setting $R(x)$ to be the training images corresponding to the *bottom* (i.e., most negative) k coordinates of the datamodel weight θ.

E.4. Data support estimation

Setup. We use datamodels together with counterfactual evaluations in a guided search to efficiently estimate upper bounds on the size of data supports. For a given target example x with corresponding datamodel g_θ, we want to find candidate training subsets of small size k whose removal most reduces the classification margin on x:

$$G_k \coloneqq \arg\min_{|G|=k} g_\theta(S \setminus G_k). \tag{14}$$

Because g_θ is a linear model in our case, the solution to the above minimization problem is simply the set corresponding to the largest k coordinates of the datamodel parameter θ:

$$G_k = \arg\max_{G \subset S; |G|=k} \theta^\top \mathbf{1}_G = \text{top-}k \text{ indices of } \theta. \tag{15}$$

Our goal is to the find the smallest of these subsets $\{G_k\}_k$ so that $f_\mathcal{A}(x; S \setminus G_k) < 0$, i.e., the example is misclassified on average as per our definition.[11] Thus, for each target x, we try several values of $k \in \{10, 20, 40, 80, 160, 320, 640, 1280\}$, training models on the set $S \setminus G_k$ and evaluating the resulting models on x.[12] We train $T = 20$ models on each counterfactual $S \setminus G_k$ to reduce variance.

(Given that we are using datamodels as surrogates afterall, one might wonder if the above counterfactual evaluations are actually necessary—one could instead consider estimating the optimal k directly from θ. We revisit a heuristic estimation procedure based on this idea at the end of this subsection.)

Estimation methodology. We assume that the expected margin $h(k) \coloneqq f_\mathcal{A}(x; S \setminus G_k)$ after removing k examples decreases *monotonically* in k; this is expected from the linearity of our datamodels and is further supported empirically (see Figure E.1). Then, our goal is to estimate the unique zero[13] $\widehat{k}$ of the above function $h(k)$ based on (noisy) samples of $h(k)$ at our chosen values of k. Note that by definition, $\widehat{k}$ is an upperbound on $\textsc{Support}(x)$. Now, because of our monotonicity assumption, we can cast estimating $\widehat{k}$ as instance of an isotonic regression problem (Robertson et al., 1988)); this effectively performs piecewise linear interpolation, while ensuring that monotonity constraint is not violated. We use `sklearn`'s `IsotonicRegression` to fit an estimate $h(k)$, and use this to estimate $\widehat{k}$.

Verifying support estimates. Due to stochasticity in evaluating counterfactuals, the estimate $\widehat{k}$ is noisy. Thus, it is possible that $\widehat{k}$ is not a valid upperbound on $\textsc{Support}(x)$, e.g. removing top $\widehat{k}$ examples do not misclassify x. In fact,

[11] Note that $\mathbb{E} f_\mathcal{A}(x; \cdot) < 0$ does not imply that the probability of misclassification is > 50%. Nonetheless, it is a natural threshold.

[12] While a binary search over k for each x would be more sample efficient, we collect the entire grid of samples for simplicity.

[13] More precisely, the upper ceiling as data support is defined as an integer quantity.

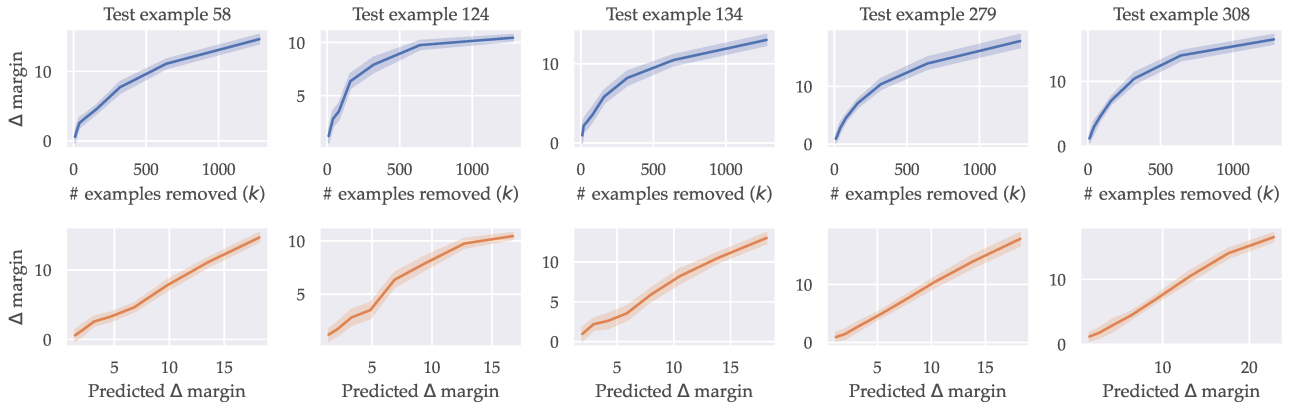


Figure E.1: Counterfactuals for individual examples. We plot the results of counterfactual evaluations (using $\alpha = 0.5$ datamodels) for five individual examples, shown in separate columns. **(Top)** The actual Δ margin changes monotonically with number of examples removed (k), corroborating the monotonicity assumption used in estimating data supports. **(Bottom)** On x, we instead plot the *predicted* Δ margin using datamodels. This shows that the linearity seen in Figure 4 manifests even at a local level.

removing $G_{\widehat{k}}$ and re-training shows that only 67% of the images are actually misclassified. To establish an upperbound on $\textsc{Support}(x)$ that has sufficient coverage, we evaluate the counterfactuals after removing an additional 20% of highest datamodel weights, e.g. removing top $\widehat{k} \times 1.2$ examples for each test example. When an additional 20% of training examples are removed, 92% of test examples are misclassified. Hence, we use $\widehat{k} \times 1.2$ for our final estimates of $\textsc{Support}(x)$.

E.4.1. Estimation using baselines

As baselines, we use the same guided search algorithm described above, but instead of using datamodel-predicted values to guide the search, we select the candidate subset using each of the baselines methods described in Appendix E.2. In particular, we choose the candidate subset R_k for a given k as follows:

1. *Representation distance*: top k closest training examples to x as measured by ℓ_2 distance in the representation space of a pre-trained ResNet-9.

2. *Influence estimates (influence functions and TracIn)*: top k training examples with highest (most positive) estimated influence on the target example x:

3. *Random*: first k examples from a random ordering[14] of training examples from the same class as x.

E.4.2. Heuristic estimates for data support

While we constructively estimate the data supports by training models on counterfactuals and using the above estimation procedure, we can also consider a simpler and cheaper heuristic to estimate $\textsc{Support}(x)$ assuming the fidelity of the linear datamodels: compute the smallest k s.t. the sum of the k highest datamodel weights for x exceeds the average margin of x. In Figure E.2, we compare the predicted data supports based on this heuristic to the estimated ones from earlier, and find that they are highly correlated. In practice, this can be a more efficient alternative to quantify brittleness without additional model training (beyond the initial ones to estimate the datamodels).

E.5. Brittleness to mislabeling

Note that the brittleness we consider in Section 4.1 (i.e., brittleness to *removing* training examples) is substantively different than brittleness to *mislabeling* examples (as in *label-flipping attacks* (Koh & Liang, 2017; Xiao et al., 2012; Rosenfeld et al., 2020)). In particular, brittleness to removal indicates that there exists a small set of training images whose presence

[14]The random ordering is fixed across different choices of k, but not across different targets.

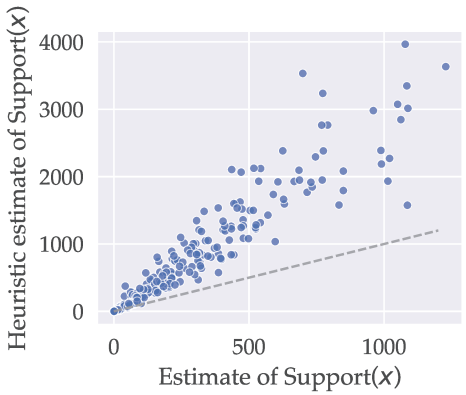


Figure E.2: Heuristic predictions for data supports. For each of the 300 test examples shown, the x-coordinate represents the previous estimates based on counterfactuals, and the y-coordinate represents the heuristic estimate.

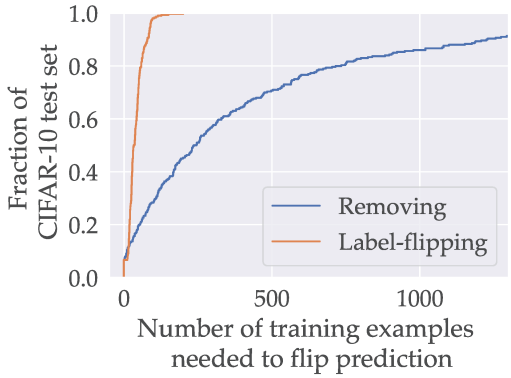


Figure E.3: Brittleness to mislabeling. We estimate an upperbound on the the smallest number of training images that can be mislabeled to flip a given target image. Much fewer images are required compared simply removing them (blue), as mislabeling provides additional signal to the model.

is *necessary* for correct classification of the target example (thus motivating the term "data support"). Meanwhile, label-flipping attacks can succeed even when the target example has a large data support, as (consistently) mislabeling a set of training examples provide a much stronger signal than simply removing them. Nevertheless, we can easily adapt the experiment from Section 4.1 to test brittleness to mislabeling.

We take the same 300 random CIFAR-10 test examples and analyze them as follows: First, we find for each example the *incorrect* class with the highest average logit (across ∼10,000 models trained on the full training set). Then, we construct counterfactual datasets similarly as in Appendix E.4 where we take the top $k = \{2, 4, ..., 256\}$ training examples with the highest datamodel weights, but this time mislabel them with the incorrect class identified earlier. After training $T = 20$ models on each counterfactual, for each target example we estimate the number of mislabeled examples at which the expected margin becomes zero, using the same estimation procedure described in Appendix E.4.

The resulting mislabeling brittleness estimates are shown in Figure E.3. As one might expect, test predictions are even *more* brittle to data mislabeling than removal—for 50% of the CIFAR-10 test set, mislabeling *35 target-specific training examples* suffices to flip the corresponding prediction.

E.6. Comparing raw effect sizes

Instead of comparing the data support estimates (which are derived quantities), here we directly compare the average counterfactual effect (i.e. delta margins) of groups selected using different methods. Figure E.4 shows again that datamodels identify much larger effects. Among baselines, we see that TracIn performs best, followed by representation distance. We also see that the representation baseline does not gain any additional signal from simple averaging over models.

E.7. Effect of training stochasticity

As described in Appendix E.1, we re-train up to $T = 20$ models for *each* counterfactual to reduce noise that arises soley from stochasticity of model training. These additional samples significantly reduces unexplained variance: Figure E.5 shows the reduction in variance ("thickness" in the y-direction) and the resulting increase in correlation as the number of re-training runs is increased from $T = 1$ to $T = 20$.

E.8. Transfer to different architecture

While the main premise of datamodeling is understanding how data is used by a given *fixed* learning algorithm, it is natural to ask how well datamodels can predict across different learning algorithms. We expect some degradation in predictivness,

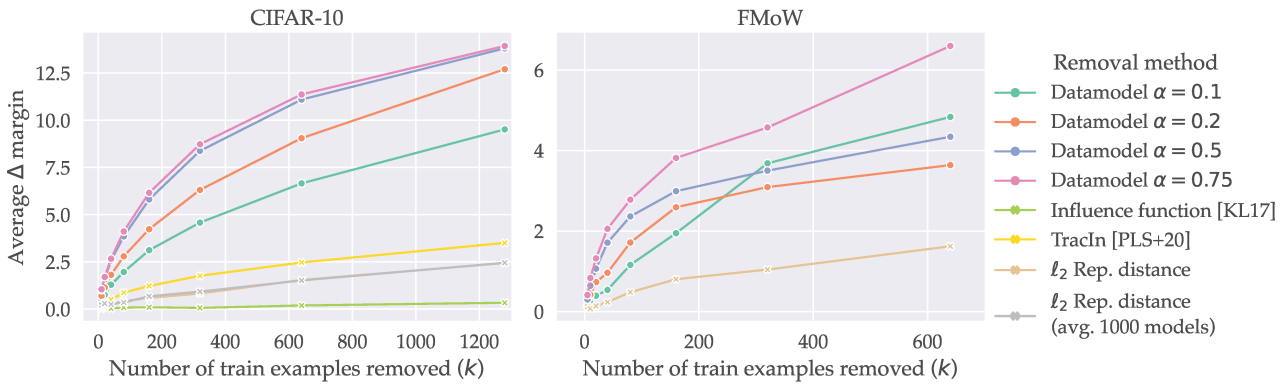


Figure E.4: Comparing effect sizes with baselines. This shows the raw evaluations of counterfactuals generated using different methods, which were also used for estimating data supports. The y-axis shows the effect on margin averaged across all target examples when top k examples are removed for each target using each of the methods. Datamodels identify much larger effects compared to baselines. Among baselines, TracIn(Pruthi et al., 2020) performs the best. For representation distance, there is no noticeable gain from reducing stochasticity by averaging over more models (1000 vs 1).

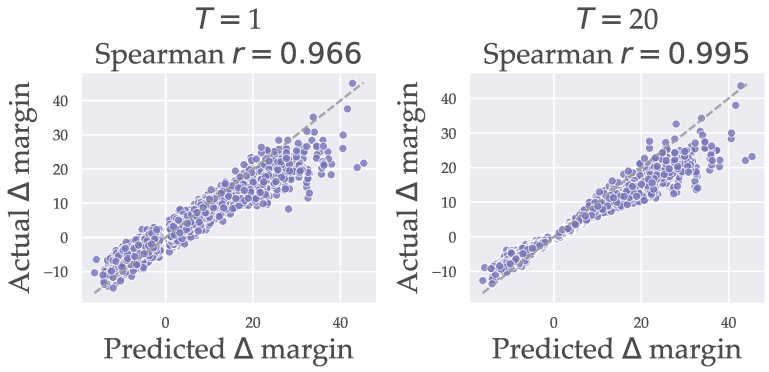


Figure E.5: Effect of model averaging. T is the number of models trained per counterfactual.

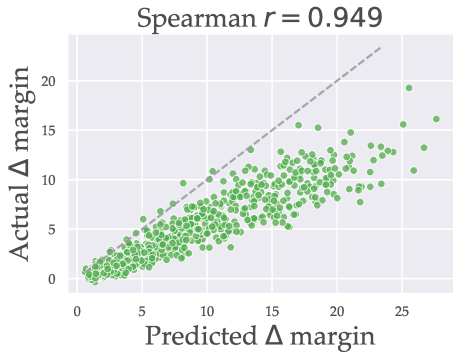


Figure E.6: Transfer across model classes.

as datamodels are fit to a particular learning algorithm; at the same time, we also expect some transfer of predictive power as modern deep neural networks are known to make similar predictions and errors (Mania et al., 2019).

Here, we study one of the factors in a learning algorithm, the choice of architecture. We take the same counterfatuals and evaluate them on ResNet-18 models, using the same training hyperparameters. As expected, the original datamodels continue to predict accurate counterfactuals for the new model class but with some degradation (Figure E.6).

E.9. Stress testing

Section 4 showed that datamodels excel at predicting counterfactuals across a variety of removal mechanisms. In an effort to find cases where datamodel predictions are not predictive of data counterfactuals, we evaluate the following additional counterfactuals:

- **Larger groups of examples (up to 20% of the dataset)**: we remove $k = 2560, 5120, 10240$ top weights using different datamodels $\alpha = 0.1, 0.2, 0.5, 0.75$. The changes in margin have more unexplained variance when larger number of images are removed; nonetheless, the overall correlation remains high ((Figure E.7)).

- **Groups of training examples whose predicted effects are *zero***: we remove $k = 20, 40, 80, 160, 320, 640, 1280, 2560$ examples with zero weight ($\alpha = 0.5$), chosen randomly among all such examples. All of tested counterfactuals had negligible impact on the actual margin, consistent with the prediction of datamodels (Figure E.8a).
- **Groups of examples whose predicted effect is *negative* according to baselines**: we test TracIn and influence functions. (We do not consider the representation distance baseline here is there is no obvious way of extracting this information from it.) Correlation degrades but remains high (Figure E.8b). Note that the relative scale of the effects is much smaller compared to counterfactuals generated using datamodels (Figure E.8a).

In general, although there is some reduction in datamodels' predictiveness, we nevertheless find that datamodels continue to be accurate predictors of data counterfactuals.

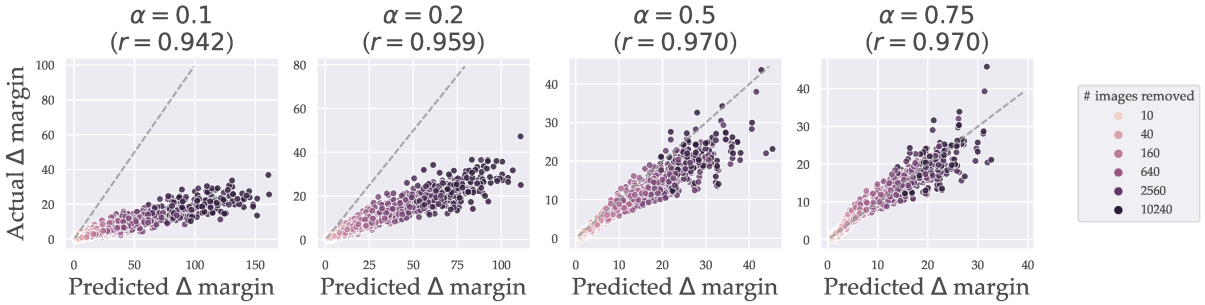
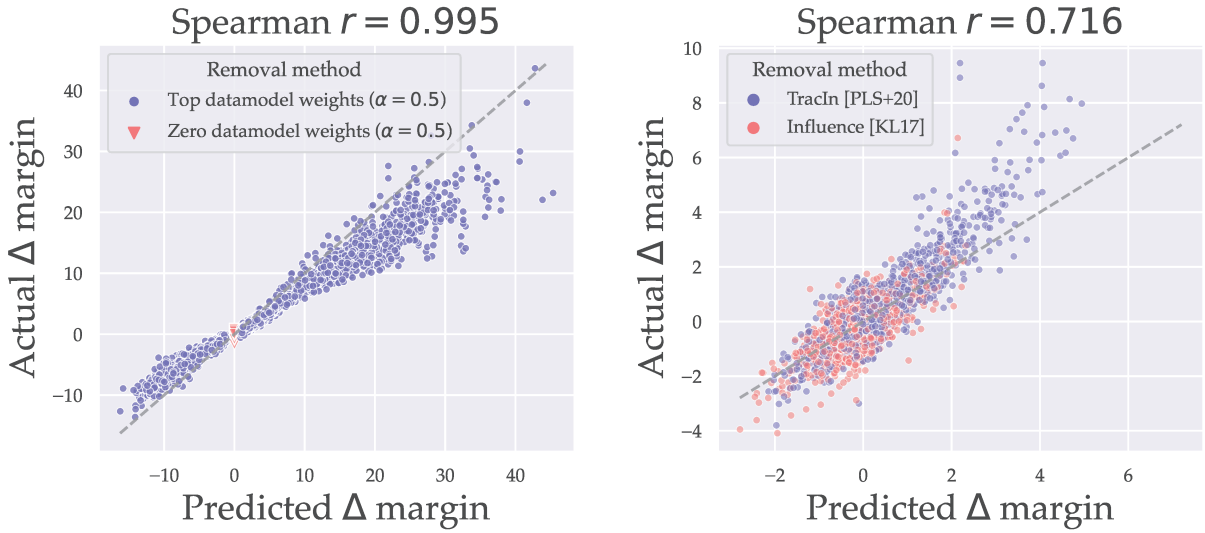


Figure E.7: Stress testing datamodel counterfactuals by removing a large number of images. Plot shows datamodel counterfactuals from before ($k = 10, ..., 1280$) along with additional samples $k = 2560, 5120, 110240$ (shown with darker hue).



(a) Comparing counterfactuals with highest vs *zero* predicted effect using datamodels ($\alpha = 0.5$)

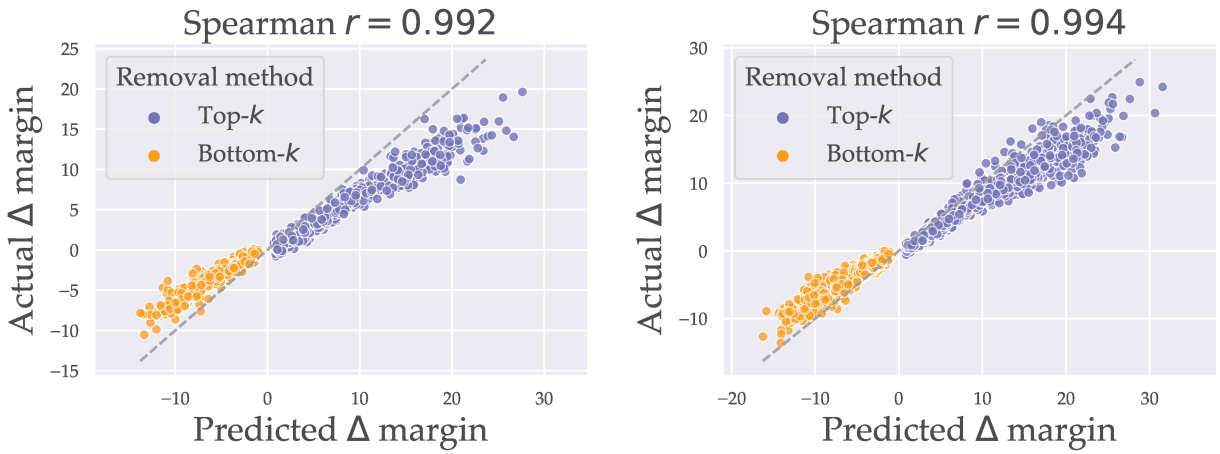
(b) Removing top k positive and negative influence training examples according to baseline methods.

Figure E.8: Stress testing counterfactual prediction.

Counterfactuals relative to a random control. All of the counterfactuals studied so far are relative a fixed control (the entire training set). Here, we consider counterfactuals relative to a *random* control $S_0 \sim \mathcal{D}_S$ at $\alpha = 0.5$ (i.e. $|S_0| = \alpha|S|$). The motivation for considering the shifted control is two folds: first, the counterfactuals generated relative to such S' are closer in distribution to the original distribution to which datamodels were fit to, so it is natural to study datamodels in this

regime; second, this tests whether the counterfactual predictability is robust to the exact choice of the trainset. Latter is desirable, as ultimately we would like to understand how models behave on training sets similar in distribution to S, not the exact train set.

To implement above, after removing a target group G from the full train set S, we subsample the remainder S/G with probability α. We adjust the control values accordingly to $\mathbb{E}_{S_0 \sim \mathcal{S}}[f_{\mathcal{A}}(x; S_0)]$, where $\mathcal{D}_S$ is the $\alpha = 0.5$ subsampling distribution. The results show that one can indeed also predict counterfactuals relative a random control (Figure E.9).



(a) Random control ($\alpha = 0.5$ regime)

(b) Fixed control ($\alpha = 1.0$)

Figure E.9: Datamodels can predict counterfactuals relative to random controls. As in Figure 4, each point in the graphs above corresponds to a test example and a counterfactual trainset S' (a subset of the full training set, S). The counterfactual is relative a *random* control $S_0 \sim \mathcal{D}_S$ with $\alpha = 0.5$, e.g. a set randomly subsampled at 50%. The y-coordinate of each point represents the expected *ground-truth* difference, in terms of model output on x, between training on a random S_0, and training on S'. The x-coordinate of each point represents the *datamodel-predicted* value of this quantity. **(a)** We use the $\alpha = 0.5$ datamodels to predict counterfactuals generated by removing, for each test example, the training inputs corresponding to the top-k and bottom-k (for several k) datamodel weights. **(b)** Same, but relative to a fixed control $S_0 = S$, e.g. the full train set.

E.10. Additional plots for different α values

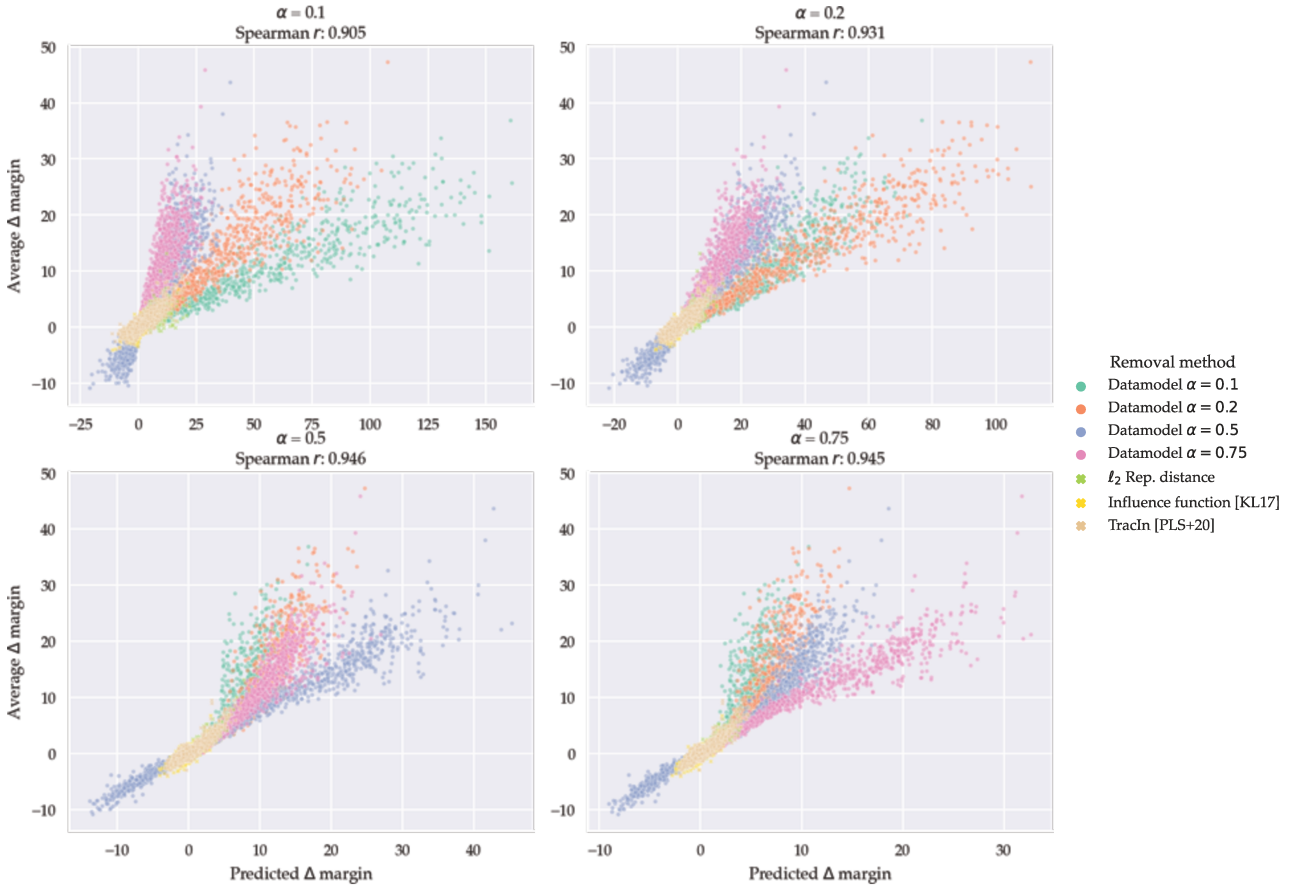


Figure E.10: Varying α for counterfactual prediction (CIFAR-10). Same plot as in Figure 4, except varying the datamodels used for prediction; each plot uses datamodels with the given α. As before, each point in the graphs above corresponds to a test example and a subset $R(x)$ of the original training set S, identified by a (color-coded) heuristic. The y-coordinate of each point represents the *ground-truth* difference, in terms of model output on x, between training on S, and training on $S \setminus R(x)$. The x-coordinate of each point represents the *datamodel-predicted* value of this quantity.

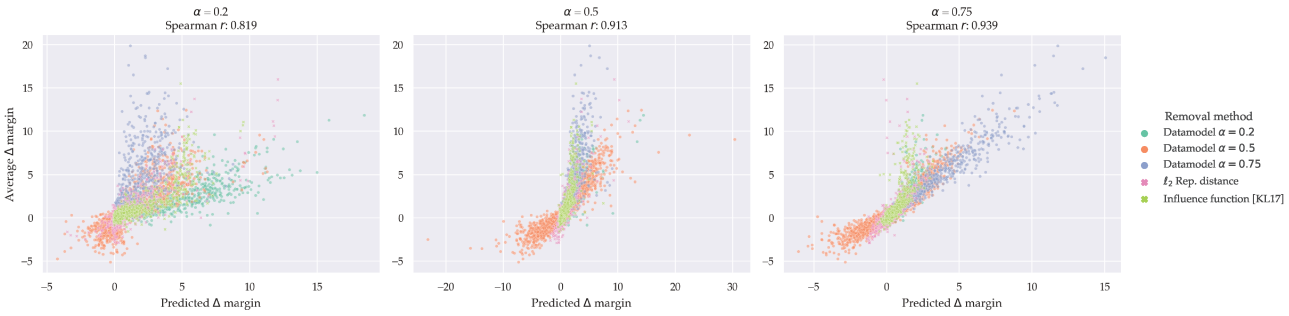


Figure E.11: Varying α for counterfactual prediction (FMoW). Same plot as in Figure 4, except varying the datamodels used for prediction; each plot uses datamodels with the given α. As before, each point in the graphs above corresponds to a test example and a subset $R(x)$ of the original training set S, identified by a (color-coded) heuristic. The y-coordinate of each point represents the *ground-truth* difference, in terms of model output on x, between training on S, and training on $S \setminus R(x)$. The x-coordinate of each point represents the *datamodel-predicted* value of this quantity.

F. Datamodels for Train-Test Similarity

We explore in detail the application introduced in Section 5.1: identifying similar training examples to a given test example. One can use such examples, e.g., to identify issues in large datasets such as duplicated training examples (Lee et al., 2021) or train-test leakage (Barz & Denzler, 2020) (test examples that have near-duplicates in the training set).

Recall that in our instantiation, datamodels predict model output (for a fixed target example) as a *linear* function of the presence of each training example, i.e., we predict the output of training on a subset S' of the training set S as

$$g_\theta(\mathbf{1}_{S'}) = \theta^\top \mathbf{1}_{S'}.$$

A benefit of parameterizing datamodels as simple linear functions is that we can use the magnitude of the coordinates of θ to ascertain *feature importance* (Guyon & Elisseeff, 2003). In particular, since in our case each feature coordinate (i.e., each coordinate of $\mathbf{1}_{S'}$) actually represents the presence of a particular training example, we can interpret the highest-magnitude coordinates of θ as the indices of the training examples whose presence (or absence) is most predictive of model behavior (again, on the fixed target example in context).

In this section, we study these high-magnitude training examples more closely, and find that (a) they visually resemble the target image, yielding a method for finding similar training examples to a given target; and (b) as a result, datamodels can automatically detect train-test leakage.

F.1. Finding similar training examples

Motivated by the feature importance view described above, we visualize (in Figures F.1 and F.3) a random set of target examples from the CIFAR-10 test set, together with the CIFAR-10 training images that correspond to the highest-magnitude datamodel coordinates for each test image.

Results. Indeed, for a given target example, the highest-magnitude datamodel coordinates—both positive and negative—consistently correspond to visually similar training examples.

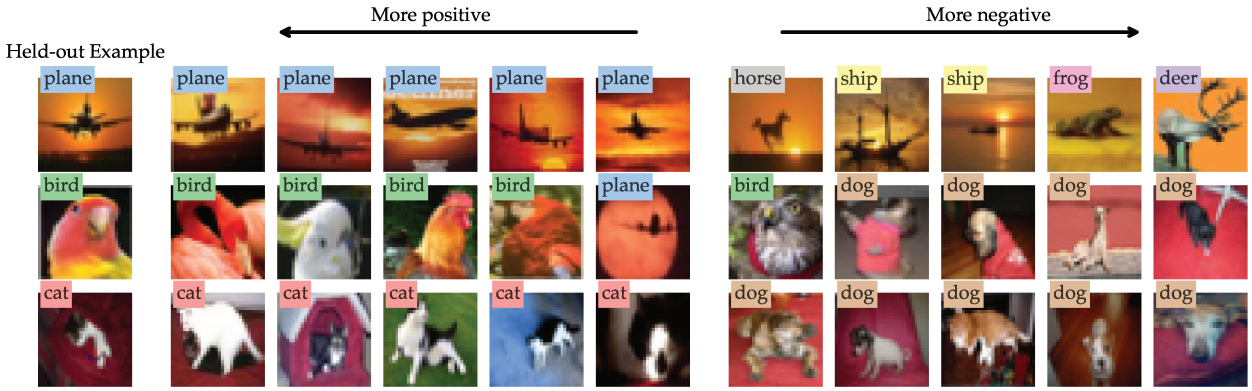


Figure F.1: Large datamodel weights correspond to similar images. Randomly choosing test examples and visualizing their most negative- and positive-weight examples for $\alpha = 50\%$, we find that large magnitude train examples share similarities with their test examples. Top negative weights generally correspond to visually similar images from other classes.

Furthermore, the exact training images that are surfaced by looking at high-magnitude weights differ based on the subsampling parameter α that we use while constructing the datamodels (Figures F.2 and F.4). (Recall from Section 2 that α controls the size of the random subsets used to collect the datamodel training set—a datamodel estimated with parameter α is constructed to predict outcomes of training on random training subsets of size $\alpha \cdot d$, where d is the training set size.) In Figure F.2, for example, on the left a datamodel estimated using $\alpha = 10\%$ identifies a set of training images that are highly similar to the target example, while the $\alpha = 50\%$ datamodel does not. Our hypothesis here—which we expand upon in Section C—is that datamodels estimated with lower α (i.e., based on smaller random training subsets) find train-test relationships driven by larger groups of examples (and vice-versa).

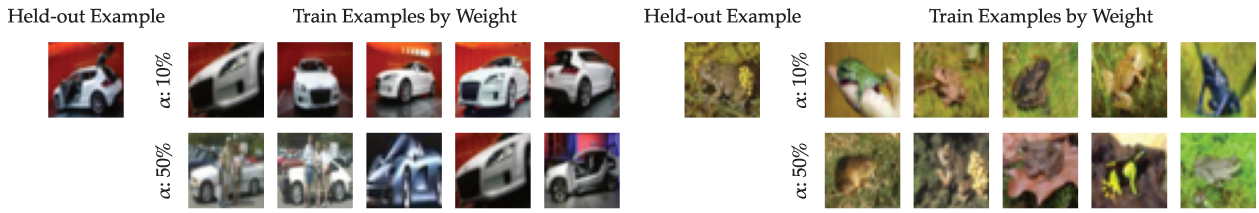


Figure F.2: Datamodels of different α surface qualitatively different images. For each target example (taken from the CIFAR-10 test set), we consider two different datamodels: one estimated with $\alpha = 10\%$ (i.e., constructed to predict model outputs on the target example after training on random 10% subsets of the CIFAR-10 training set), and the other estimated with $\alpha = 50\%$. For each datamodel, we visualize the training examples corresponding to the largest coordinates of the parameter vector θ. On the left we see an example where the datamodel estimated with $\alpha = 10\%$ detects a set of near-duplicates of the target example that the $\alpha = 50\%$ datamodel does not identify. See Figure F.4 for more examples.

Influence functions. Another method for finding similar training images is *influence functions*, which aim to estimate the effect of removing a single training image on the loss (or correctness) for a given test image. A standard technique from robust statistics (Hampel et al., 2011) (applied to deep networks by Koh & Liang (2017)) uses first-order approximation to estimate influence of each training example. We find (Figure F.5), that the high-influence and low-influence examples yielded by this approximation (and similar methods) often fail to find similar training examples for a given test example.

Another approach based on *empirical* influence approximation was used by Feldman & Zhang (2020), who (successfully) use their estimates to identify similar train-test pairs in image datasets as we do above. We discuss empirical influence approximation and its connection with datamodeling in Section H.1.

FMoW. We show similar results for FMoW: in Figure F.6 we show *randomly selected* target images along with their top-weight train images, using datamodels of different α. In Figure F.7 we show more examples of test images and their corresponding top train images as we vary α.

Figure F.3: Additional examples of held-out images and their corresponding highest and lowest datamodel weight training images.

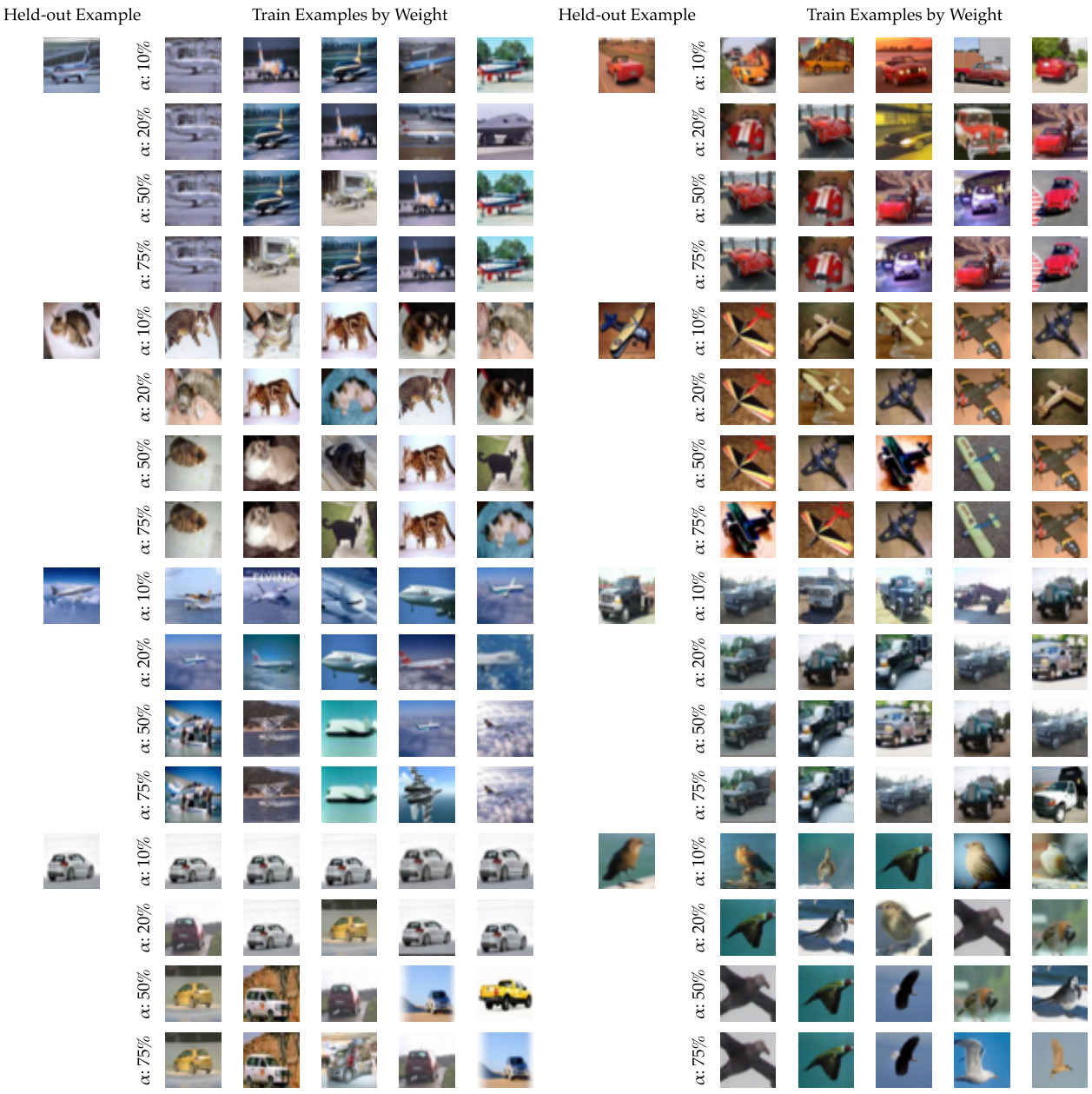


Figure F.4: Additional examples of held-out images and corresponding most relevant training examples, while varying α.

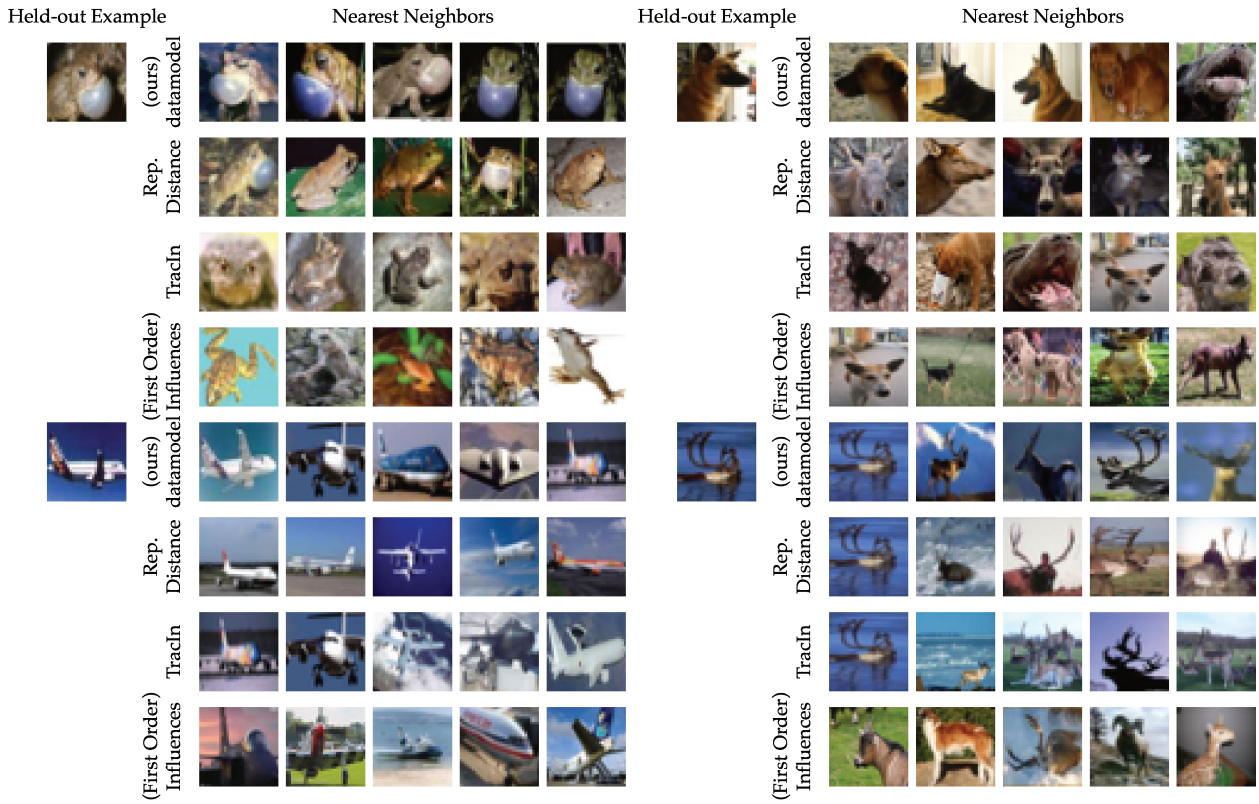


Figure F.5: Comparisons of nearest neighbors found using different methods.

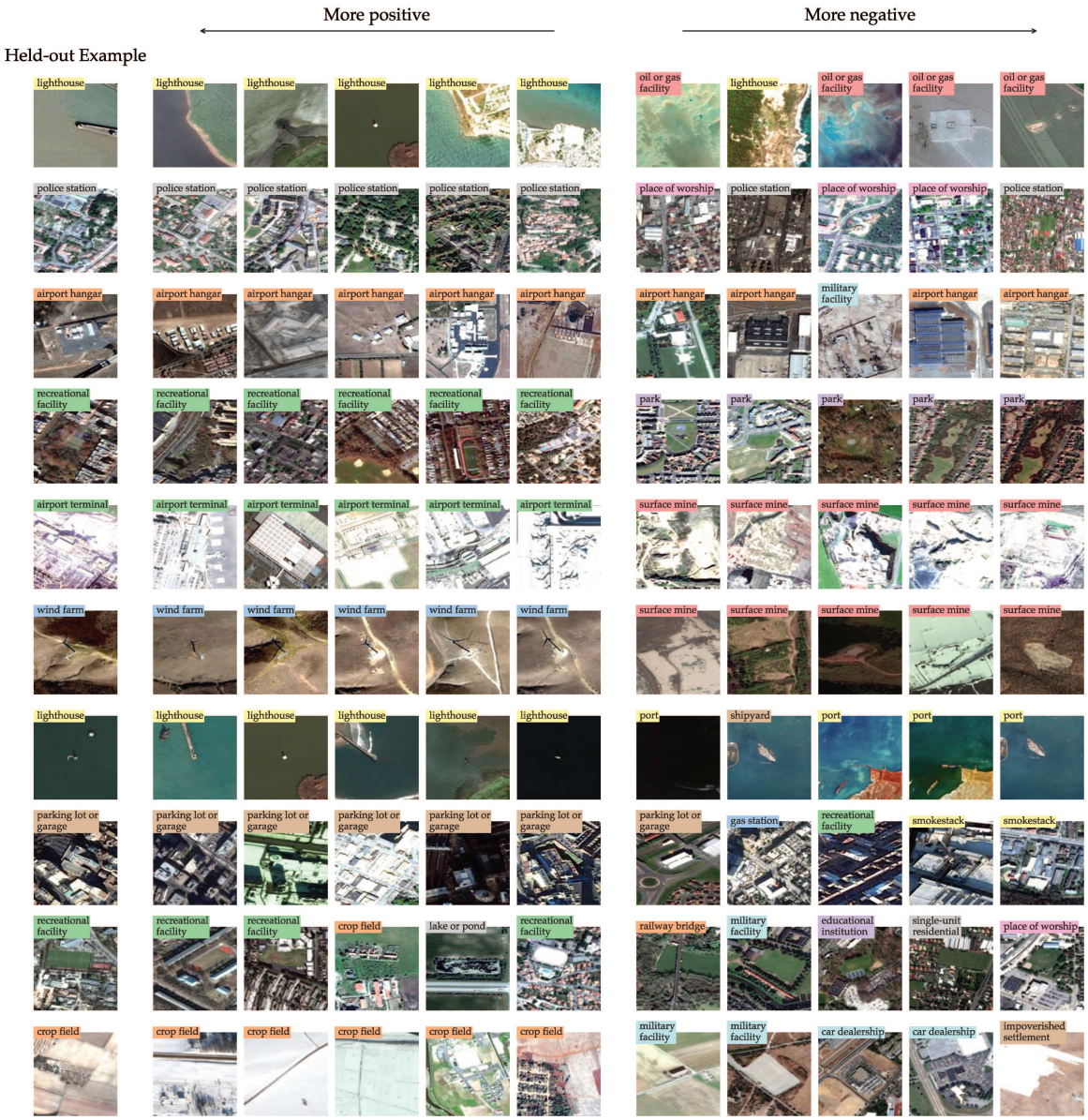


Figure F.6: FMoW examples of held-out images and their corresponding highest and lowest datamodel weight training images.

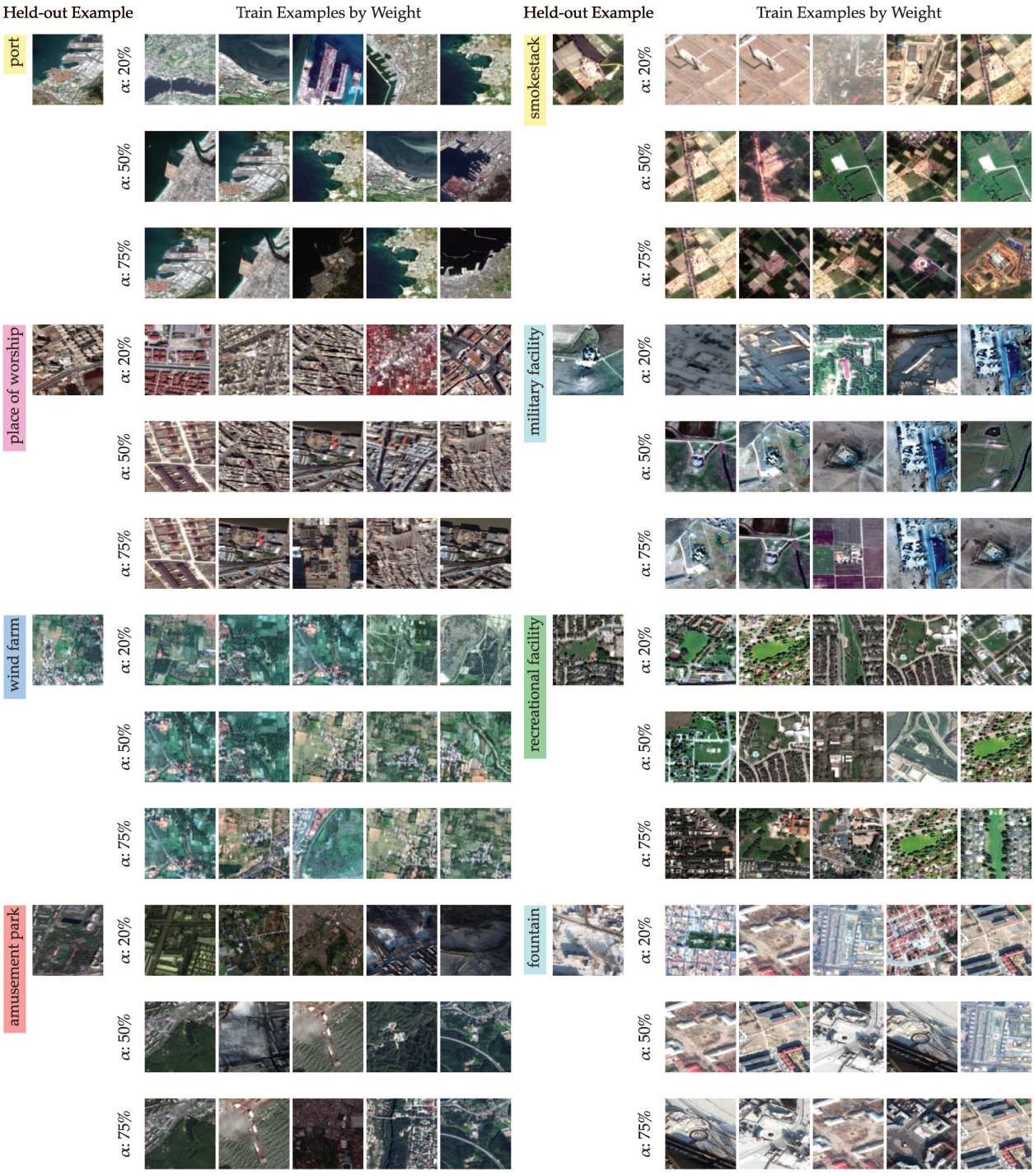


Figure F.7: FMoW examples of held-out images and corresponding most relevant training examples, while varying α.

F.2. Identifying train-test leakage

We now leverage datamodels' ability to surface similar training examples to a given target in order to identify *same-scene* train-test leakage: cases where test examples are near-duplicates of, or clearly come from the same scene as, training examples. Below, we use datamodels to uncover evidence of train-test leakage on both CIFAR and FMoW, and show that datamodels outperform a natural baseline for this task.

F.2.1. Train-test leakage in CIFAR-10

To find train-test leakage in CIFAR-10, we collect ten candidate training examples for each image in the CIFAR-10 test set—corresponding to the ten largest coordinates (top 5 and bottom 5) of the test example's datamodel parameter. We then show crowd annotators (using Amazon Mechanical Turk[15]; Figure F.8 shows the interface) tasks that consist of a random CIFAR-10 test example accompanied by its candidate training examples. We ask the annotators to label any of the candidate training images that constitute instances of same-scene leakage (as defined above). We show each task (i.e., each test example) to nine annotators, and compute the "annotation score" for each of the test example's candidate training examples as the fraction of annotators who marked it as an instance of leakage. Finally, we compute the "leakage score" for each test example as the highest annotation score (over all of its candidate train images)—we use the leakage score as a proxy for whether or not the given image constitutes train-test leakage.

In Figure 5, we plot the distribution of leakage scores over the CIFAR-10 test set, along with random train-test pairs stratified by their annotation score. As the annotation score increases, pairs (qualitatively) appear more likely to correspond to leakage; we show more examples of (train, test) pairs stratified by annotation score in Figure F.9. Overall roughly 10% of test set images were labeled as train-test leakage by over half of the annotators that reviewed them. The vast majority of potential leakage found corresponded to training examples by the positive datamodel weight (one of the top 5).

Comparison with CIFAIR. Barz & Denzler (2020) present CIFAIR, a version of CIFAR with fewer duplicates. The authors define duplicates slightly differently than we define pairs constituting same scene train-test leakage, see Section 3.2 of their work compared to the screenshot of the annotation interface in Appendix Figure F.8. The authors identify train-test leakage by using a deep neural network to measure representation space distances between images across training partitions and manually inspecting the lowest distances.

F.2.2. Train-test leakage in FMoW

To identify train-test leakage on FMoW, we begin with the same candidate-finding process that we used for CIFAR-10. However, FMoW differs from CIFAR in that the examples (satellite images labeled by category, e.g., "port" or "arena") are annotated with *geographic coordinates*. These coordinates allow us to avoid crowdsourcing—instead, we compute the geodesic distance between the test image and each of the candidates, and use a simple threshold d (in miles) to decide whether a given test example constitutes train-test leakage.

Furthermore, we can calculate a "ground-truth" number of train-test leakage instances by counting the test examples whose *geodesic* nearest-neighbor in the training set is within the specified threshold d.[16] Comparing this ground truth to the number of instances of leakage found within the candidate examples yields a qualitative measure of the efficacy of our method (i.e., the quality of candidates we generate).

In Figure F.10, we plot this measure of efficacy (# instances found / # ground truth) as a function of the threshold d, and also visualize examples images from the FMoW test set together with their corresponding datamodel-identified training set candidates. To put our quantitative results into context, we compare the efficacy of candidates derived from top datamodel coordinates (i.e., the ones we use here and for CIFAR-10) to that of candidates derived from *nearest neighbors* in the representation space of a pretrained neural network (Bengio et al., 2013; Zhang et al., 2018) (examining such nearest neighbors is a standard way of finding train-test leakage, e.g., used by (Barz & Denzler, 2020) to study CIFAR-10 and CIFAR-100). Datamodels consistently outperform the baseline.

[15] We paid 12 cents per task completed, and used qualifications: locale in US/CA/GB and percentage of hits approved $> 95\%$.

[16] It turns out that despite having already been de-duplicated, about 20% and 80% of FMoW test images are within 0.25 and 2.6 miles of a training image, respectively—see Figure F.11.

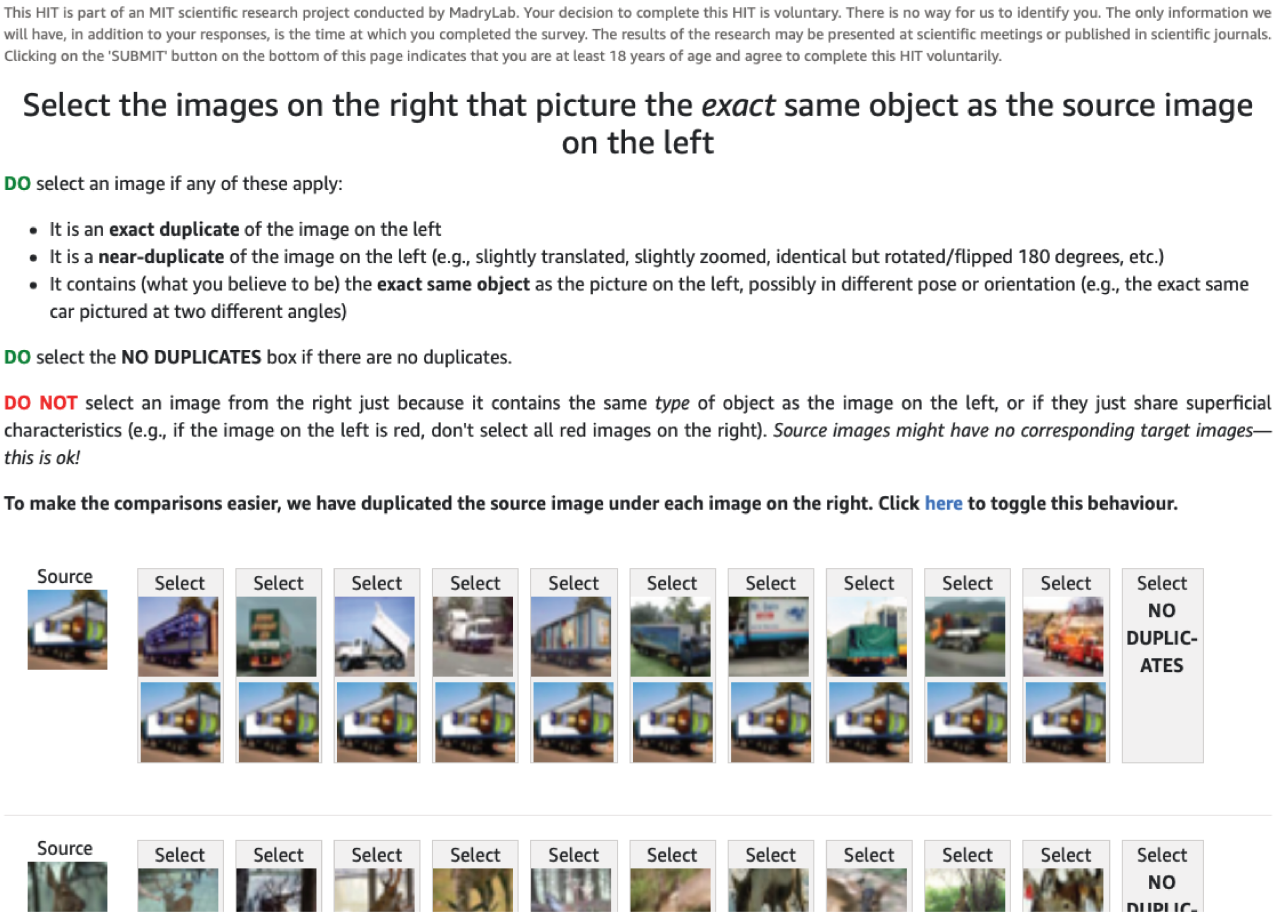


Figure F.8: The MTurk Interface, complete with instructions, shown to crowdsourced annotators. Note that there are 5 rows of images in the actual interface, some of which may require scrolling to get to.

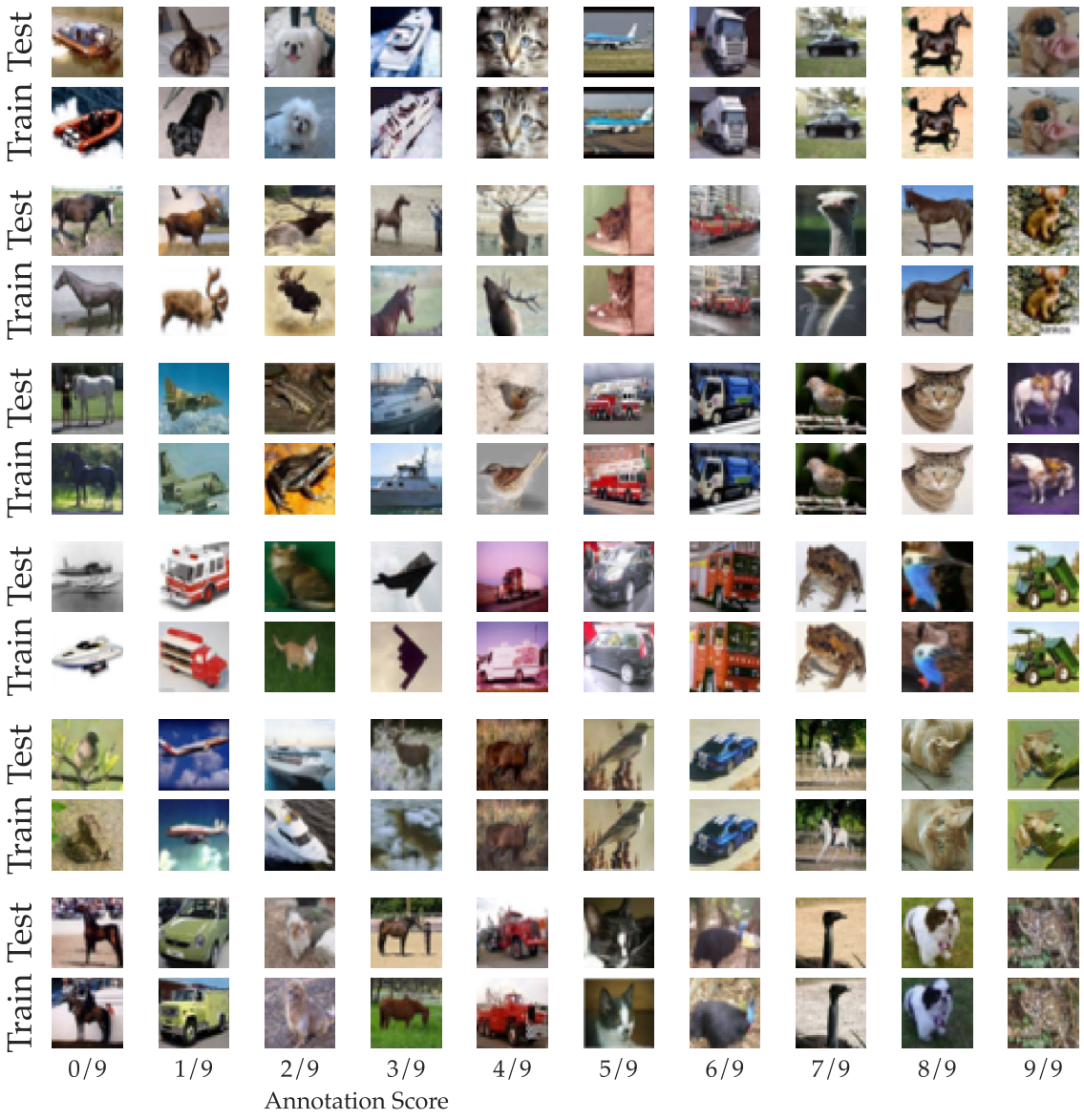


Figure F.9: More annotation scores paired with (train, test) leakage pairs.

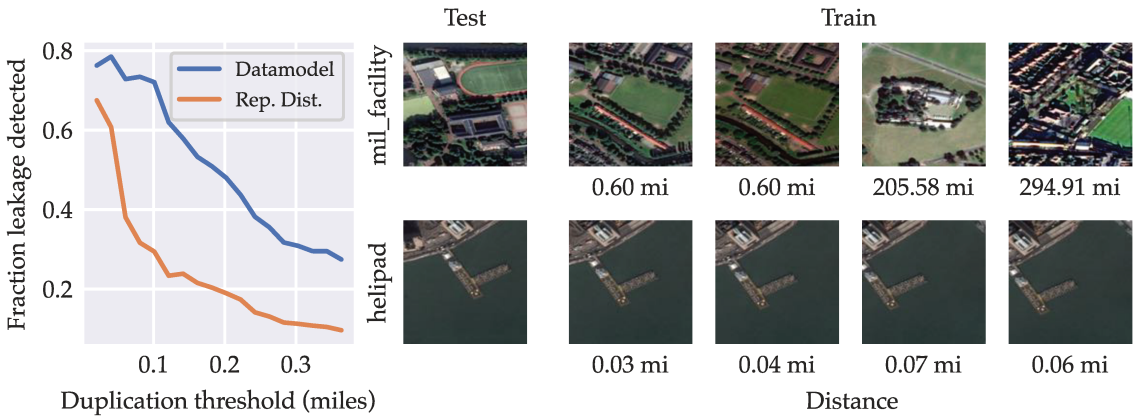


Figure F.10: Datamodels detect same-scene train-test leakage on FMoW. FMoW images are annotated with geographic coordinates. For any distance d, we call a test image x "leaked" if it is within d miles of *any* training image x_s. A leaked test image x is considered "detected" if the corresponding training image x_s has one of the 10 largest datamodel weights for x. **(Left)** With d on the x-axis, we plot the fraction of leaked test images that are also detected. As a baseline, we replace datamodel weights with (negative) distances in neural network representation space. **(Right)** for two test examples (top: random; bottom: selected), we show the most similar train examples (by datamodel weight), labeled by their distance to the test example.

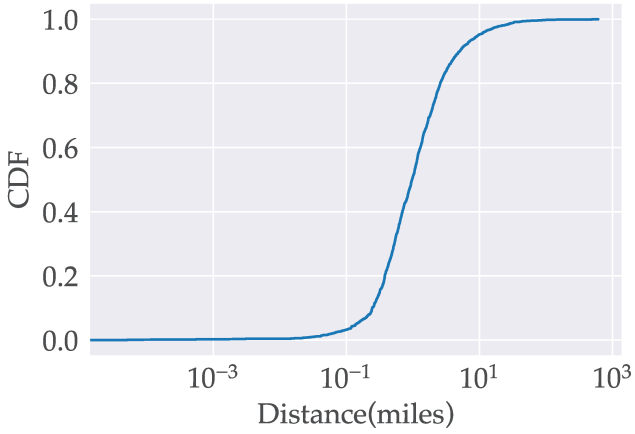


Figure F.11: CDF of distance in miles between each FMoW test set image and the nearest train set image.

G. Datamodels as a Feature Embedding

Using datamodels for counterfactual prediction (Section 4 and Appendix E) and train-test similarity (Section 5.1 and Appendix F) illustrate the utility of datamodels on an *per-example* level, i.e., for predicting the outcome of training on arbitrary training subsets and evaluating on a specific target example, or for finding similar training images (again, to a specific target). By adopting a new perspective on datamodels, Section 5.2 demonstrates that datamodels can also help uncover *global structure* in datasets of interest. We explore this further here.

Critically, the coordinates of a datamodel embedding have a *consistent* interpretation across datamodel embeddings, even for different target examples. That is, we expect similar target examples to be acted upon similarly by the training set, and thus have similar datamodel embeddings. In the same way, if model performance on two unrelated target examples is driven by two disjoint sets of training examples, their datamodel embeddings will be orthogonal. This intuition suggests that by embedding an entire dataset of examples $\{x_i\}$ as a set of feature vectors $\{\theta_i \in \mathbb{R}^d\}$, we may be able to uncover structure in the set of examples by looking for structure in their datamodel embeddings, i.e., in the (Euclidean) space $\mathbb{R}^d$.

In this section we demonstrate, through two applications, the potential for datamodel embeddings to discover dataset structure in this way. In §,G.1, we use datamodel embeddings to partition datasets into disjoint clusters, and in § G.2 we use principal component analysis to get more fine-grained insights into dataset structure. To emphasize our shift in perspective (i.e., from θ being just a parameter of a datamodel g_θ, to θ being an embedding for the target example x), we introduce an *embedding function* $\varphi(x) \mapsto \theta$ which maps a particular target example to the weights of its corresponding datamodel.

G.1. Spectral clustering with datamodel embeddings

We begin with a simple application of datamodel embeddings, and show that they enable high-quality clustering. Specifically, given two examples x_1 and x_2, datamodel embeddings induce a natural *similarity measure* between them:

$$d(x_1, x_2) \coloneqq K(\varphi(x_1), \varphi(x_2)), \tag{16}$$

where we recall that $\varphi(\cdot)$ is the *datamodel embedding function* mapping target examples to the weights of their corresponding datamodels, and $K(\cdot, \cdot)$ is any kernel function.[17] Taking this even further, for a set of k target examples $\{x_1, \ldots, x_k\}$, we can compute a full *similarity matrix* $A \in \mathbb{R}^{k \times k}$, whose entries are

$$A_{ij} = d(x_i, x_j). \tag{17}$$

Finally, we can view this similarity matrix as an *adjacency matrix* for a (dense) graph connecting all the examples $\{x_1, \ldots x_k\}$: the edge between two examples will be $d(x_i, x_j)$, which is in turn the kernelized inner product between their two datamodel weights. We expect similar examples to have high-weight edges between them, and unrelated examples to have (nearly) zero-weight edges between them.

Such a graph unlocks a myriad of graph-theoretic tools for exploring datasets through the lens of datamodels (e.g., cliques in this graph should be examples for which model behavior is driven by the same subset of training examples). A complete exploration of these tools is beyond the scope of our work: instead, we focus on one just such tool, namely spectral clustering.

At a high level, spectral clustering is an algorithm that takes as input any similarity graph G as well as the number of clusters C, and outputs a partitioning of the vertices of G into C disjoint subsets, in a way that (roughly) minimizes the total weight of inter-cluster edges. We run an off-the-shelf spectral clustering algorithm on the graph induced by the similarity matrix A above for the images in the CIFAR-10 test set. The results demonstrate a simple unsupervised method for uncovering subpopulations in datasets: Figure 6 shows examples of clusters from two classes, Figure G.1 compares top clusters for the horse class across different α. Figure G.2 and **??** shows additional clusters for eight other classes, apart from the ones shown in Figure 6.

Implementation. We use `sklearn`'s `cluster.SpectralClustering`. Internally, this computes similarity scores using the radial basis function (RBF) kernel on the datamodel embeddings. Then, it runs spectral clustering on the graph defined by the similarity matrix A: it computes a Laplacian L, represents each node using the first k eigenvectors of L, and runs k-means clustering on the resulting feature representations. We use $k = 100$.

[17] A kernel function $K(\cdot, \cdot)$ is a similarity measure that computes the inner product between its two arguments in a transformed inner product space (see (Shawe-Taylor & Cristianini, 2004) for an introduction). The RBF kernel is $K(v_1, v_2) = \exp\{-\|v_1 - v_2\|^2 / 2\sigma^2\}$

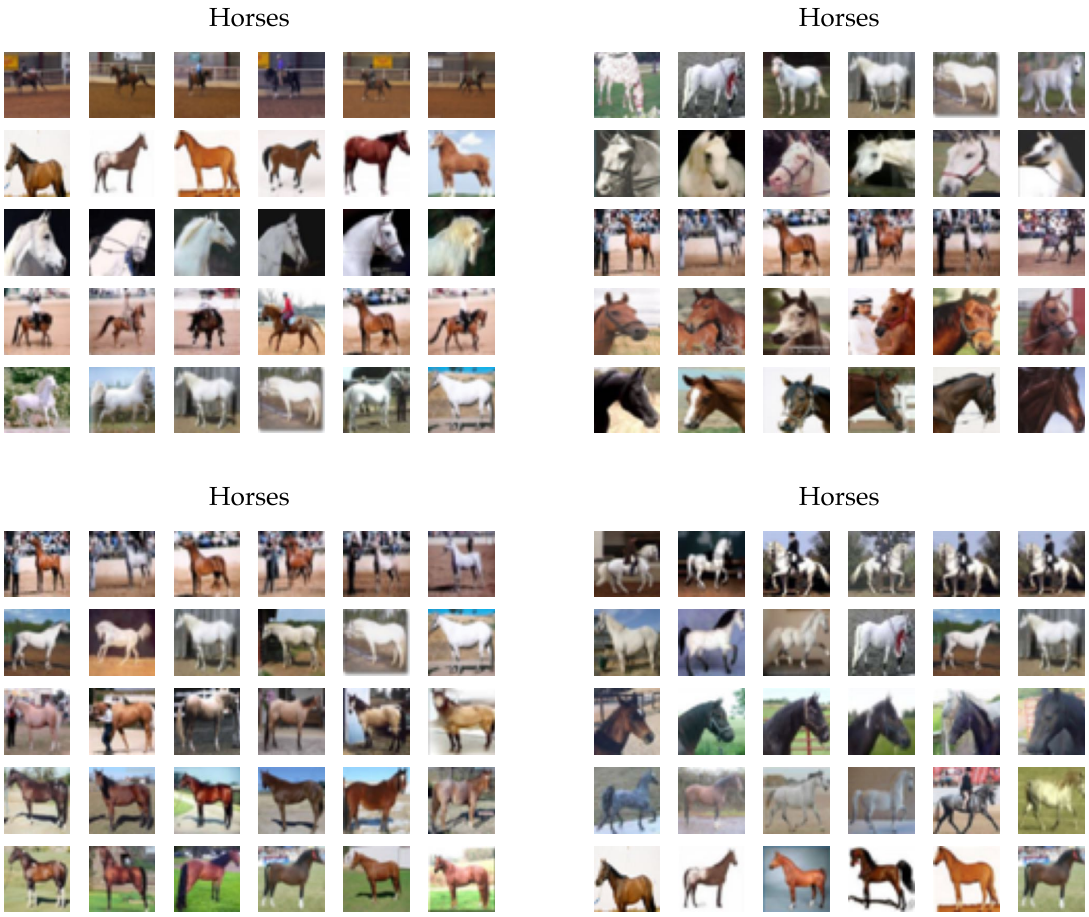


Figure G.1: Omitted spectral clustering results for datamodels computed with $\alpha = 10\%$ (top left), 20% (top right), 50% (bottom left), and 75% (bottom left).

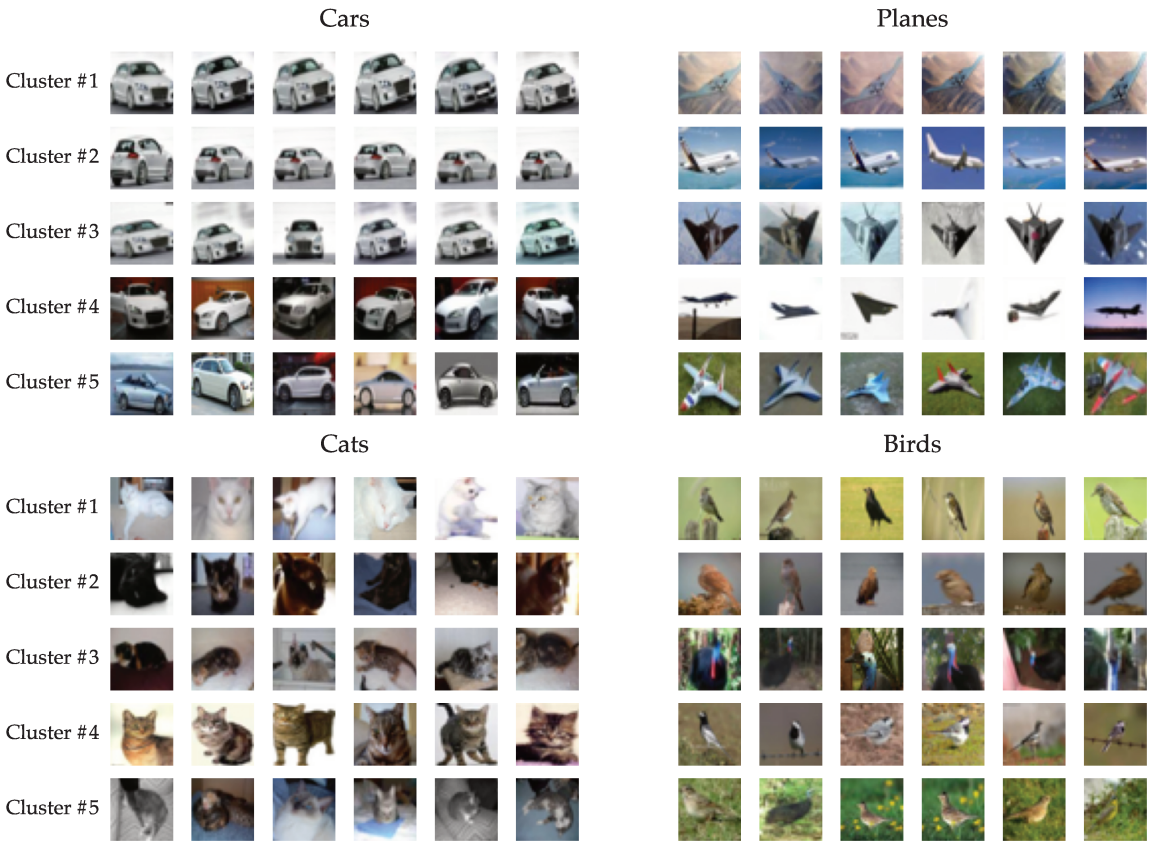


Figure G.2: Omitted spectral clustering results for classes other than those in the main paper (Figure 6).

Figure G.3: Additional omitted spectral clustering results for classes other than those in the main paper (Figure 6).

G.2. Analyzing datamodel embeddings with PCA

From the last section, we observe that datamodel embeddings encode enough information about their corresponding examples to cluster them into (at least qualitatively) coherent groups. We now try to gain even further insight into the structure of these datamodel embeddings, in the hopes of shedding light on the structure of the underlying dataset itself.

Datamodel embeddings are both high-dimensional and sparse, making analyzing them directly (e.g., by looking at the variation of each coordinate) a daunting task. Instead, we leverage a canonical tool for finding structure in high-dimensional data: principal component analysis (PCA). PCA is a dimensionality reduction technique which—given a set of embeddings $\{\varphi(x_i) \in \mathbb{R}^d\}$ and any $k \ll d$—returns a *transformation function* that maps any embedding $\varphi(x) \in \mathbb{R}^d$ to a new embedding $\widetilde{\phi}(x) \in \mathbb{R}^k$, such that:

(a) each of the k coordinates of the transformed embeddings is a (fixed) linear combination of the coordinates of the initial datamodel embeddings, i.e., $\widetilde{\varphi}(x) = \boldsymbol{M} \cdot \varphi(x)$ for a fixed $k \times d$ matrix $\boldsymbol{M}$;

(b) transformed embeddings preserve as much information as possible about the original ones (more formally, we find the matrix $\boldsymbol{M}$ that allows us to *reconstruct* the given set of embeddings $\{\varphi(x_i) \in \mathbb{R}^d\}$ from their transformed counterparts with minimal error).

Note that in (a), the i-th coordinate of a transformed embedding is always the *same* linear combination of the corresponding original embedding (and thus, each coordinate of the transformed embedding has a concrete interpretation as a weighted combination of datamodel coefficients). The exact coefficients of this combination (i.e., the rows of the matrix $\boldsymbol{M}$ above) are called the first k *principal components* of the dataset.

We apply PCA to the collection of datamodel embeddings $\{\varphi(x_i) \in \mathbb{R}^d\}_{i=1}^d$ for the CIFAR-10 training set, and use the result to to compute new k-dimensional embeddings for each target example in both the training set and the test set (i.e., by computing each target example's datamodel embedding then transforming it to an embedding in $\mathbb{R}^k$). We can then look at each coordinate in the new, much more manageable (k-dimensional) embeddings.[18]

Coordinates identify subpopulations. Our point of start in analyzing the transformed embeddings is to examine each transformed coordinate separately. In particular, in Figures G.4 and G.9 we visualize, for a few sample coordinate indices $i \in [k]$, the target examples whose transformed embeddings have particularly high or low values of the given coordinate (equivalently, these are the target examples whose datamodel embeddings have the highest or lowest projections onto the i-th principal component). We find that:

(a) The examples whose transformed embeddings have a large i-th coordinate all (visually) share a common feature: e.g., the first-row images in Figure G.4 share similar pose and color composition;

(b) This (visual) feature is consistent across both train and test set examples[19]; and

(c) For a given coordinate, the most positive images and most negative images (i.e., the left and right side of each row of Figure G.4, respectively) either (a) have a differing label but share the same common feature or (b) have the same label but differ along the relevant feature.

Principal components are *model-faithful*. We verify that not only are the groups of images found by PCA visually coherent, they are in fact rooted in how the model class makes predictions. To that end, we analyze how "removing" different principal components affect model predictions. More precisely, we remove training examples corresponding to:

- Top k most positive coordinates of the principal component vector
- Top k most negative coordinates of the principal component vector

Then, for each principal component direction considered, we measure their impact on three groups of held-out samples:

[18] One detail here is that we first *normalize* each datamodel embedding before transforming them (i.e., we transform $\varphi(x)/\|\varphi(x)\|$).

[19] Recall that we computed the PCA transformation to preserve the information in only the *training set* datamodel embeddings. Thus, this result suggests that the transformed embeddings computed by PCA are not "overfit" to the specific examples that we used to compute it.

Figure G.4: PCA on datamodel embeddings. We visualize the top three principal components (PCs) and a randomly selected PC from the top 100. In the i-th row, the left-most (right-most) images are those whose datamodel embeddings have the highest (lowest) normalized projections onto the i-th principal component v_i. Highest magnitude images along each direction share qualitative features; moreover, images at opposite ends suggest a *feature tradeoff*—a combination of images in the training set that helps accuracy on one subgroup but hurts accuracy on the other. See Figure G.9 for more datamodel PCA components.

- The top 100 examples by most positive projection on the principal component
- The bottom 100 examples by most negative projection on the principal component
- The full test set

For each of these groups, we measure the mean change in margin after removing different principal component directions. Our results (Figure G.5) show that:

- Removing the most positive coordinates of the PC decreases margin on the test set examples with the most positive projections on the PC and increases margin on the examples with the most negative projections on the PC.
- Removing the most negative coordinates of the PC has the opposite effect, increasing margin on the positive projection examples and decreasing margin on the negative projection examples.
- Increasing the size of each removed set increases the effect magnitude.
- Removing PC's have negligible impact on the aggregate test set, indicating that the impact of different PC's are roughly "orthogonal," as one would expect based on the orthogonality of the PCs.

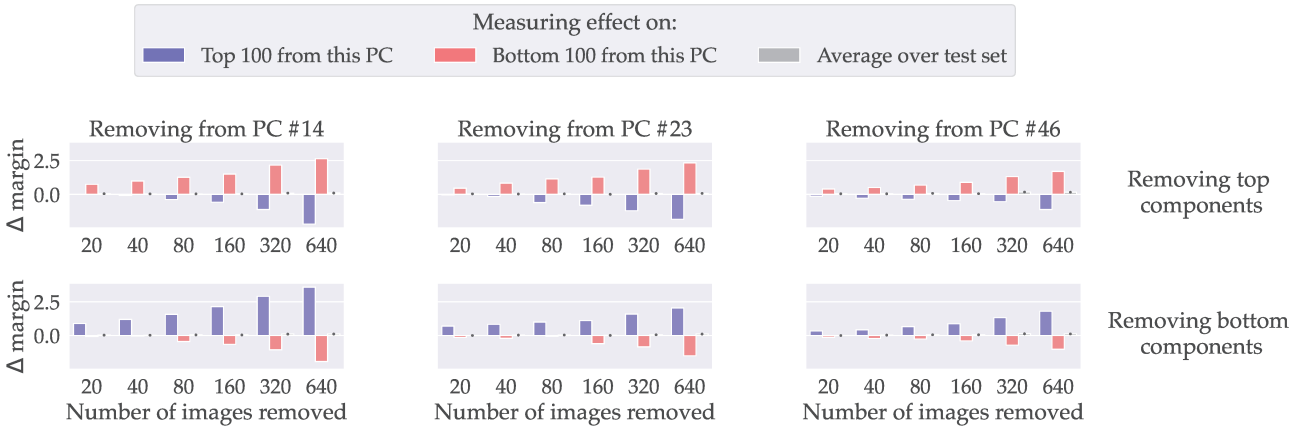


Figure G.5: PCA directions generalize and capture "orthogonal" directions. For each of the three principal components (PCs) above (randomly chosen from top 50), we consider the counterfactual of removing the training examples corresponding to the top or bottom k coordinates in the PC, and measure its average effect on different groups: (red) test examples with the highest projections on the PC; (blue) test examples with the lowest projections on the PC; and (grey) the entire test set. The direction of the effect is consistent with the datamodel embeddings; removing top (resp. bottom) coordinates decrease (resp. increase) the average margin on test examples whose embeddings are most aligned with the PC. Moreover, the negligible impact over the test set in aggregate shows that the different PCs, which are orthogonal in the embedding space (by definition), are also approximately "orthogonal" in terms of their effect on model predictions.

- Lastly, Figure G.6 shows that datamodels can accurately predict the counterfactual effect of the above removed groups, similarly as in Figure 4.

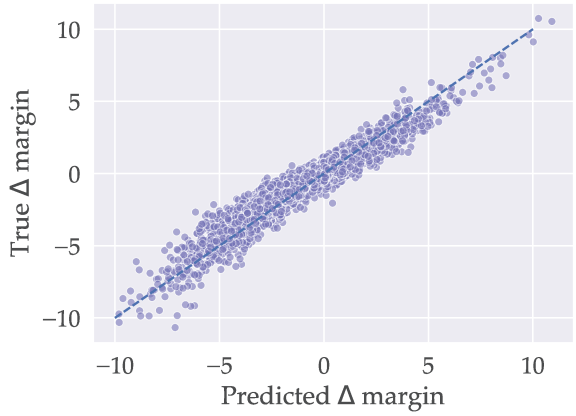


Figure G.6: Datamodels predict the effect of "removing" principal components. Each point corresponds to a *PCA counterfactual*: removing training examples with the largest weights in the principal component (i.e., top-k most positive or negative coordinates), and evaluating on test examples whose embeddings most align with the PC (e.g. smallest cosine distance). The y-coordinate of each point represents the *ground-truth* counterfactual effect (evaluated by retraining $T = 20$ times). The x-coordinate of each point represents the *datamodel-predicted* value of this quantity.

G.2.1. Advantages over penultimate-layer embeddings

In the context of deep neural networks, the word "embedding" typically refers to features extracted from the penultimate layer of a fixed pre-trained model (see (Bengio et al., 2013) for an overview). These "deep representations" can serve as an effective proxy for visual similarity (Barz & Denzler, 2020; Zhang et al., 2018), and also enable a suite of applications such as clustering (Guérin et al., 2017) and feature visualization (Olah et al., 2017; Engstrom et al., 2019; Azizpour et al.,

2015; Ben-David et al., 2007).

Here, we briefly discuss a few advantages of datamodel-based embeddings over their standard penultimate layer-based counterparts.

- **Axis-alignment**: First, datamodel embeddings are *axis-aligned*—each embedding component directly corresponds to index into the training set, as opposed to, e.g., a more abstract or qualitative concept. As a corollary, aggregating or comparing different datamodel embeddings for a given dataset is straightforward, and does not require any alignment tools or additional heuristics (this is not the case for network-based representations, for which the right way to combine representations—even for two models of the same architecture—is still disagreed upon (Kornblith et al., 2019; Bansal et al., 2021)). In particular, we can straightforwardly compare datamodel embeddings across different target examples, model architectures, training paradigms, or even datamodel estimation techniques: as long as the set of training examples being stays the same, any resulting datamodel has a uniform interpretation.
- **Richer representation**: The space of datamodel embeddings seems significantly richer than that of standard representation space. In particular, Figure G.7 shows that for standard representation space, *10 linear directions* suffice to capture *90% of the variation* in training set representations. The "effective dimension" of datamodel representations is much higher, with the top *500 principal components* explaining only *50% of the variation* in training set datamodel embeddings. This difference manifests qualitatively when we redo our PCA study on standard representations (Figure G.8): principal components beyond the 10th lack both the perceptual uniformity and train-test generalization exhibited by those of datamodel embeddings (e.g., for datamodels even the 76th principal component, shown in Figure G.4, exhibits these qualities).
- **Ingrained causality**: Finally, datamodel embeddings inherently encode information about how the model class generalizes: earlier we verified via counterfactuals that insights extracted from the principal components of Θ actually reflect underlying model class behavior.

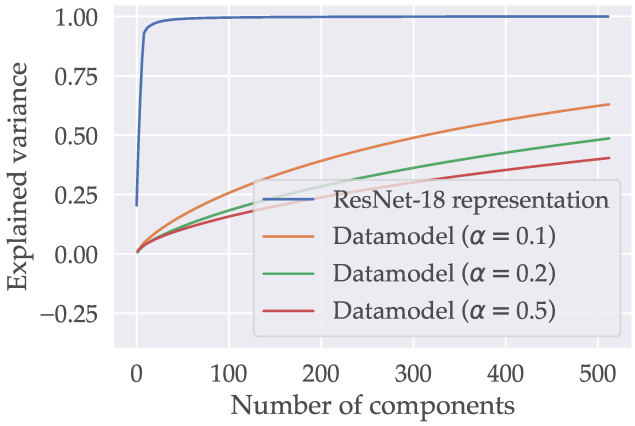


Figure G.7: Datamodel embeddings have a higher effective dimension than deep representations. For different embeddings, we plot the cumulative fraction of variance explained by the top k components while varying k. For a network layer based embedding, 95% of the variation in embedding space is captured by the first 10 principal components; meanwhile, datamodel embeddings need up to 500 components to capture even half of the variance. Here, we use a ResNet-18 model instead of ResNet-9 as it has more features in the representation layer (512 vs. 128); the plot looks similar for ResNet-9.

Figure G.8: Representation-based baseline for PCA. Visualization of highest magnitude images along top principal components of *representation* embeddings for CIFAR-10. In each row i, on the left we show the images with the highest normalized projections onto v_i, and on the right the images with the lowest projections. The PCs seem less coherent than those obtained from running PCA on datamodel embeddings.

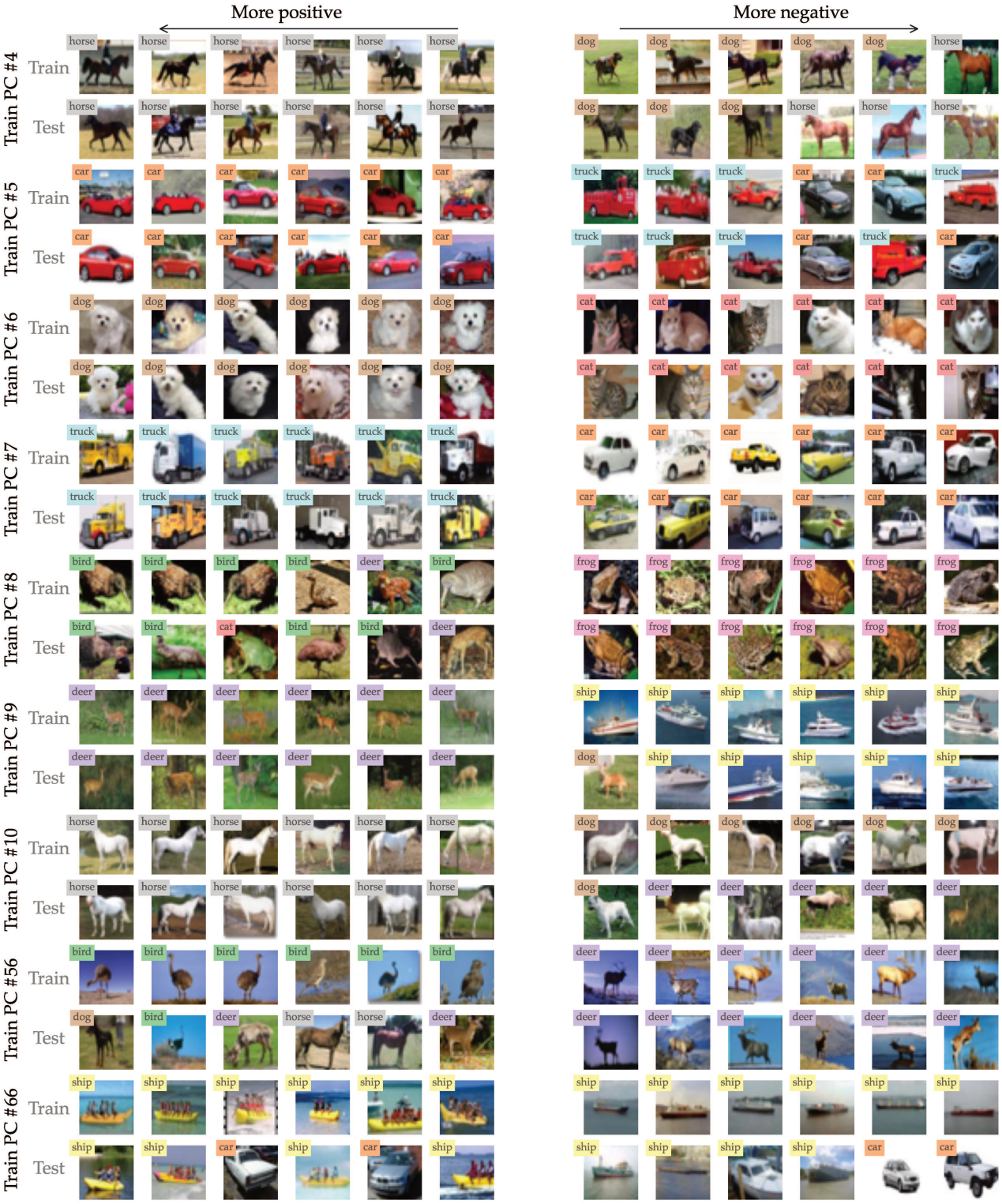


Figure G.9: The remainder of the top 10 PCA directions and two selected directions.

G.3. FMoW

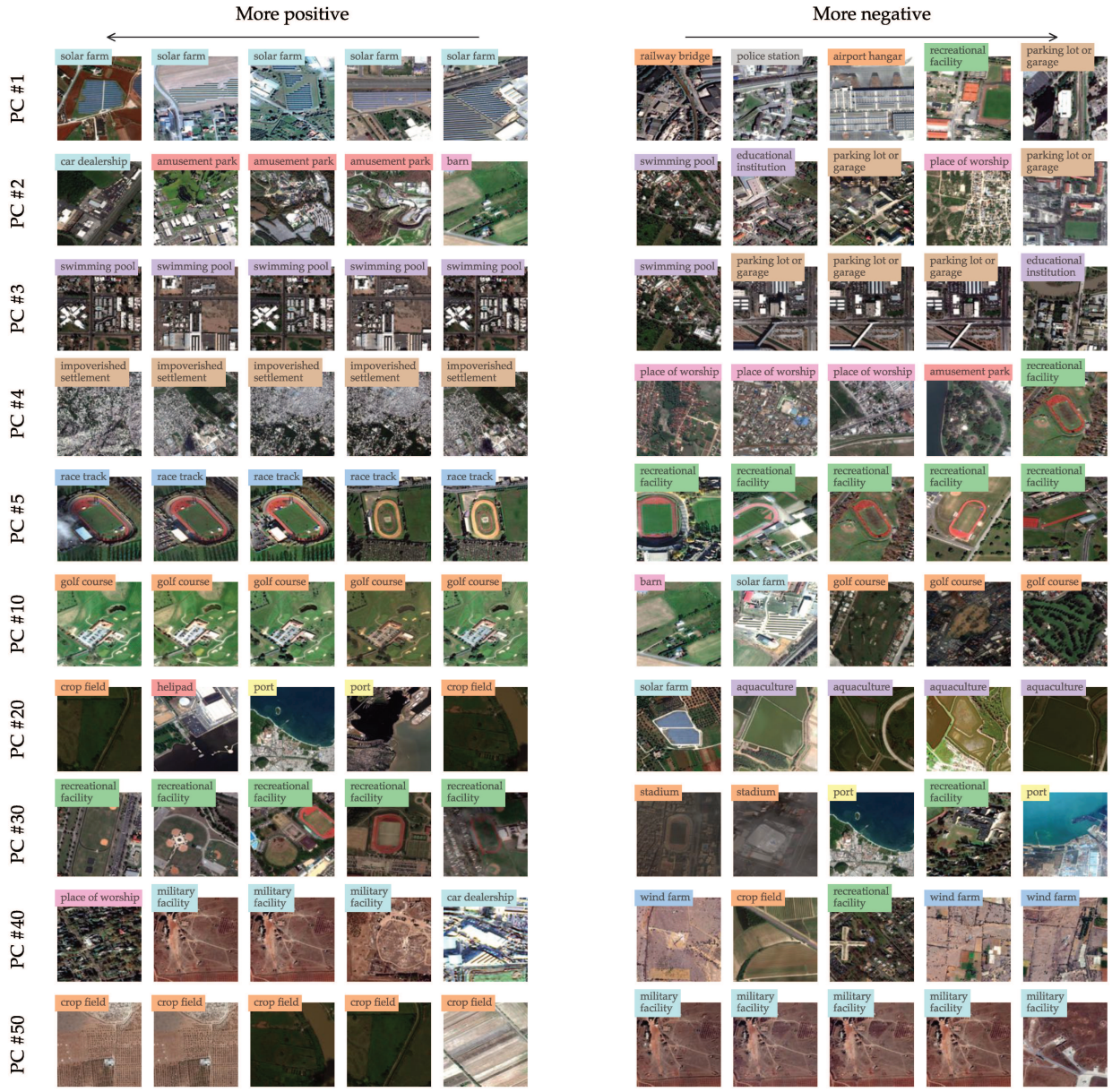


Figure G.10: FMoW top PCA components, using $\alpha = 20\%$. Top 5 and 5 selected from the top 50.

H. Related work

Datamodels build on a rich and growing body of literature in machine learning, statistics, and intepretability. In this section, we illustrate some of the connections to these fields, highlight a few of the most closely related works to ours.

H.1. Connecting datamodeling to empirical influence estimation

We start by discussing the particularly important connection between datamodels and another well-studied concept that has recently been applied to the machine learning setting: influence estimators. In particular, a recent line of work aims to compute the *empirical influence* (Hampel et al., 2011) of training points x_i on predictions $f(x_j)$, i.e.,

$$\text{Infl}[x_i \to x_j] \coloneqq \mathbb{P}\left(\text{model trained on } S \text{ is correct on } x_j\right) - \mathbb{P}\left(\text{model trained on } S \setminus \{x_i\} \text{ is correct on } x_j\right),$$

where randomness is taken over the training algorithm. Evaluating these influence functions naïvely requires training $C \cdot d$ models where d is again the size of the train set and C is the number of samples necessary for an accurate empirical estimate of the probabilities above. To circumvent this prohibitive sample complexity, a recent line of work has proposed approximation schemes for $\text{Infl}[x_i \to x_j]$. We discuss these approximations (and their connection to our work) more generally in Appendix H.2, but here we focus on a specific approximation used by Feldman & Zhang (2020) (and in a similar form, by (Ghorbani & Zou, 2019) and (Jia et al., 2019))[20]:

$$\begin{aligned}\widehat{\text{Infl}}[x_i \to x_j] = \mathbb{P}_{S\sim\mathcal{D}_S}\left(\text{model trained on } S \text{ is correct on } x_j | x_i \in S\right) \\ - \mathbb{P}_{S\sim\mathcal{D}_S}\left(\text{model trained on } S \text{ is correct on } x_j | x_i \notin S\right).\end{aligned} \tag{18}$$

This estimator improves sample efficiency by reusing the same set of models to compute influences between different input pairs. More precisely, Feldman & Zhang (2020) show that the size of the random subsets trades off sample efficiency (model reuse is maximized when $\alpha = 0.5$) and accuracy with respect to the true empirical influence (which is maximized at $\alpha \to 1$). Despite its different goal, formulation, and estimation procedure, it turns out that we can cast the difference-of-probabilities estimator (18) above as a rescaled datamodel (in the infinite-sample limit). In particular, in Appendix J.1 we show:

Lemma 1. *Fix a training set S of size n, and a test example x. For $i \in [m]$, let S_i be a random variable denoting a random 50%-subset of the the training set S. Let $\boldsymbol{w}_{infl} \in \mathbb{R}^n$ be the estimated empirical influences (18) onto x estimated using the sets S_i. Let $\boldsymbol{w}_{OLS}$ be the least-squares estimator of whether a particular model will get image x correct, i.e.,*

$$\boldsymbol{w}_{OLS} \coloneqq \arg\min_{w} \frac{1}{m}\sum_{i=1}^{m}\left(w^\top \boldsymbol{z_i} - \mathbf{1}\{\textit{model trained on } S_i \textit{ correct on } x\}\right)^2, \qquad \textit{where } \boldsymbol{z}_i = 2\cdot\mathbf{1}_{S_i} - \mathbf{1}_n.$$

Then, as $m \to \infty$, $\|(1+\frac{2}{n})\boldsymbol{w}_{OLS} - \frac{1}{2}\boldsymbol{w}_{infl}\|_2 \to 0$.

We illustrate this result quantitatively in Appendix J and perform an in-depth study of influence estimators as datamodels. As one might expect given their different goal, influence estimates significantly underperform explicit datamodels in terms of predicting model outputs with respect to every metric we studied (Table J.1, Figure J.1). We attempt to explain this performance gap and reconcile it with Lemma 1 in terms of the estimation algorithm (OLS vs. LASSO), scale (number of models trained), and output function (0/1 loss vs. margins).

In addition to forging a connection between datamodels and influence estimates, this result provides an alternate perspective on the parameter α. In light of our discussion in Appendix C, it suggests that α may control the *kinds* of correlations that are surfaced by empirical influence estimates.

H.2. Other connections

Influence functions and instance-based explanations.

First-order influence functions are a canonical tool in robust statistics that allows one to approximate the impact of removing a data point on a given parameter without re-estimating the parameter itself (Hampel et al., 2011). Koh & Liang (2017)

[20] In fact, (18) is ubiquitous—e.g., in causal inference, it is called the *average treatment effect* of training on x_i on the correctness of x_j.

apply influence functions to both a variety of classical machine learning models and to penultimate-layer embeddings from neural network architectures, to trace model's predictions back to individual training examples. In classical settings (namely, for a logistic regression model), Koh et al. (2019) find that influence functions are also useful for estimating the impact of *groups* of examples. On the other hand, Basu et al. (2021) finds that approximate influence functions scale poorly to deep neural network architectures; and Feldman & Zhang (2020) argue that understanding the dynamics of the penultimate layer is insufficient for understanding deep models' decision mechanisms. Other methods for influence approximation (or more generally, instance-level attribution) include gradient-based methods (Pruthi et al., 2020) and metrics based on representation similarity (Charpiat et al., 2019; Yeh et al., 2018)—see (Hanawa et al., 2021) for a more detailed overview. Finally, another related line of work (Ghorbani & Zou, 2019; Jia et al., 2019; Wang et al., 2021) uses *Shapley values* (Shapley, 1951) to assign a value to datapoints based on their contribution to some *aggregate* metric (e.g., test accuracy).

As discussed in Section H.1, datamodels serve a different purpose to influence functions—the former constructs an explicit statistical model, whereas the latter measures the counterfactual value of each training point. Nevertheless, we find that wherever efficient influence approximations and datamodels are quantitatively comparable (e.g., see Section 4 or Appendix J) datamodels predict model behavior better.

Pixel-space surrogate models for interpretability. Datamodels are essentially surrogate models for the function mapping training data to predictions. Surrogate models from *pixel-space* to predictions are popular tools in machine learning interpretability (Ribeiro et al., 2016; Lundberg & Lee, 2017; Sokol et al., 2019). For example, LIME (Ribeiro et al., 2016) constructs a local linear model mapping test images to model predictions. Such surrogate models try to understand, for a fixed model, how the features of a given test example change the prediction. In contrast, datamodels hold the test example fixed and instead study how the images present in the training set change the prediction.

In addition to the advantages of our data-based view stated in Section 1, datamodels have two additional advantages over pixel-level surrogate models: (a) a clear notion of *missingness* (i.e., it is easy to remove a training example but usually hard to remove a feature); (b) *globality* of predictions—pixel-level surrogate models are typically accurate within a small neighborhood of a given input in pixel space, whereas datamodels model entire distribution over subsets of the training set, and remain useful both on- and off-distribution.

Model understanding beyond fixed weights. Recall (from Section 1) that datamodels are, in part, inspired by the fact that re-training deep neural networks using the same data and model class leads to models with similar accuracies but vastly different individual predictions. This phenomenon has been observed more broadly. For example, Sellam et al. (2021) make this point explicitly in the context of BERT (Devlin et al., 2019) pre-trained language models. Similarly, Nakkiran & Bansal (2020) make note of this non-determinism for networks trained on the same training *distribution* (but not the same data), while Jiang et al. (2021) find that the same is true for networks trained on the same exact data. D'Amour et al. (2020) find that on out-of-distribution data even overall accuracy is highly random. More closely in spirit to our work, Zhong et al. (2021) find that non-determinism of individual predictions poses a challenge for comparing different model architectures. (They also propose a set of statistical techniques for overcoming this challenge.) More traditionally, the non-determinism is leveraged by Bayesian (Neal, 1996) and ensemble methods (Lakshminarayanan et al., 2017), which use a distribution over model weights to improve aspects of inference such as calibration of uncertainty.

Learning and memorization. Recent work (see (Feldman, 2019; Chatterjee, 2018; Zhang et al., 2016; Bresler & Nagaraj, 2020) and references therein) brings to light the interplay between learning and memorization, particularly in the context of deep neural networks. While memorization and generalization may seem to be at odds, the picture is more sutble. Indeed, Chatterjee (2018) builds a network of small lookup tables on small vision datasets to show that purely memorization-based systems can still generalize-well. Feldman (2019) suggests that memorization of atypical examples may be *necessary* to generalize well due to a long tail of subpopulations that arises in standard datasets. Feldman & Zhang (2020) find some empirical support for this hypothesis by identifying memorized images on CIFAR-100 and ImageNet and showing that removing them hurts overall generalization. Relatedly, Brown et al. (2021) proves that for certain natural distributions, memorization of a large fraction of data, even data irrelevant to the task at hand, is necessary for close to optimal generalization. For state of the art models, recent works (e.g., (Carlini et al., 2019; 2021)) show that one can indeed extract sensitive training data, indicating models' tendency to memorize.

Conversely, it has been observed that differentially private (DP) machine learning models—whose aim is precisely to avoid memorizing the training data—tend to exhibit poorer generalization than their memorizing counterparts (Abadi et al., 2016). Moreover, the impact on generalization from DP is disparate across subgroups (Bagdasaryan et al., 2019). A

similar effect has been noted in the context of neural network pruning (Hooker et al., 2019). Datamodeling may be a useful tool for studying these phenomena and, more broadly, the mechanisms mapping data to predictions for modern learning algorithms.

Brittleness of conclusions. A long line of work in statistics focuses on testing the *robustness* of statistical conclusions to the omission of datapoints. Broderick et al. (2021) study the robustness of econometric analyses to removing a (small) fraction of data. Their method uses a Taylor-approximation based metric to estimate the most influential subset of examples on some target quantity, similar in spirit to our use of datamodels to estimate data support for a target example (as in Figure 3). Datamodels may be a helpful tool for extending such robustness analyses to the context of state-of-the-art machine learning models.

I. Future work

Our instantiation of the datamodeling framework yields both good predictors of model behavior and a variety of direct applications. However, this instantiation is fairly basic and thus leaves significant room for improvement along several axes. More broadly, datamodeling provides a lens under which we can study a variety of questions not addressed in this work. In this section, we identify (a subset of) these questions and provide connections to existing lines of work on them across machine learning and statistics.

I.1. Improving datamodel estimation

In Section 2, we outlined our basic procedure for fitting datamodels: we first sample subsets uniformly at random, then fit a sparse linear model from (the characteristic vectors of) training subsets to model outputs (margins) via ℓ_1 regularization. We first discuss various ways in which this paradigm might be improved to yield even better predictions.

- **Correlation-aware estimation.** One key feature of our estimation methodology is that the same set of models is used to estimate datamodel parameters for an entire test set of images at once. This significantly reduces the sample complexity of estimating datamodels but also introduces a correlation between the errors in the estimated parameters. This correlation is driven by the fact that model outputs are not i.i.d. across inputs—for example, if on a picture of a dog x a given model has very large output (compared to the "average" model, i.e., if $f_{\mathcal{A}}(x; S_i) - \mathbb{E}[f_{\mathcal{A}}(x; S_i)]$ is large), the model is also more likely to have large output on another picture of a dog (as opposed to, e.g., a picture of a cat).

  Parameter estimation in the presence of such correlated outputs is an active area of research in statistics (see (Daskalakis et al., 2019; Li et al., 2019) and references therein). Applying the corresponding techniques (or modifications thereof) to datamodels may help calibrate predictions and improve sample-efficiency.

- **Confidence intervals for datamodels.** In this work we have focused on attaining point estimates for datamodel parameters via simple linear regression. A natural extension to these results would be to obtain *confidence intervals* around the datamodel weights. These could, for example, (a) provide interval estimates for model outputs rather than simple point estimates; and (b) decide if a training input is indeed a "significant" predictor for a given test input.

- **Post-selection inference.** Relatedly, the high input-dimensionality of our estimation problem and the sparse nature of the solutions suggests that a *two-stage* procedure might improve sample efficiency. In such procedures, one first selects (often automatically, e.g., via LASSO) a subset of the coefficients deemed to be "significant" for a given test example, then re-fits a linear model for *only* these coefficients. This two-stage approach is particularly attractive in settings where the number of subset-output pairs $(S_i, f_{\mathcal{A}}(x; S_i))$ is less than the size of the training set $|S|$ being subsampled.

  Unfortunately, using the data itself to perform model selection in this manner—a paradigm known as *post-selection inference*—violates the assumptions of classical statistical inference (in particular, that the model class is chosen independently of the data) and can result in significantly miscalibrated confidence intervals. Applying *valid* two-stage estimation to datamodeling would be an area for further improvement upon the protocol presented in our work.

- **Improving subset sampling.** Recall (cf. Section 2) that our framework uses a distribution over subsets $\mathcal{D}_S$ to generate the "datamodel training set." In this paper, we fixed $\mathcal{D}_S$ to be random α-subsets of the training set, and used a nearest-neighbors example (see Figure C.1) to provide intuition around the role of α. While this design choice did yield useful datamodels, it is unclear whether this class of distributions is optimal. In particular, a long line of literature in causal inference focuses on *intervention design* (Eberhardt & Scheines, 2007); drawing upon this line of work may lead to a better choice of subsampling distribution. Furthermore, one might even go beyond a fixed distribution $\mathcal{D}_S$ and instead choose subsets S_i *adaptively* (i.e., based on the datamodels estimated with the previously sampled subsets) in order to reduce sample complexity.

- **Devising better priors.** Finally, in this paper we employed simple least-squares regression with ℓ_1 regularization (tuned through a held-out validation set). While the advantage of this rather simple prior—namely, that datamodels are *sparse*—is that the resulting estimation methodology is largely data-driven, one may consider incorporating domain-specific knowledge to design better priors. For instance, one can use structured-sparsity (Huang et al., 2011) to take advantage of any additional structure.

I.2. Studying generalization

Datamodels also present an opportunity to study generalization more broadly:

- **Understanding linearity.** The key simplifying assumption behind our instantiation of the datamodeling framework is that we can approximate the final output of training a model on a subset of the training set as a *linear* function of the presence of each training point. While this assumption certainly leads to a simple estimation procedure, we have very little justification for why such a linear model should be able to capture the complexities of end-to-end model training on data subsets. However, we find that datamodels *can* accurately predict ground-truth model outputs (cf. Sections 2). In fact, we find a tight *linear* correlation between datamodel predictions and model outputs even on out-of-distribution (i.e., not in the support of $\mathcal{D}_S$) counterfactual datasets. Understanding *why* a simple linearity assumption leads to effective datamodels for deep neural networks is an interesting open question. Tackling this question may necessitate a better understanding of the training dynamics and implicit biases behind overparameterized training Bartlett et al. (2021); Sagawa et al. (2020).

- **Using sparsity to study generalization.** A recent line of work in machine learning studies the interplay between learning, overparameterization, and memorization (Feldman, 2019; Chatterjee, 2018; Zhang et al., 2016; Bresler & Nagaraj, 2020; Zhang et al., 2020). Datamodeling may be a helpful tool in this pursuit, as it connects predictions of machine learning models directly to the data used to train them. For example, the *data support* introduced in Section 4.1 provides a quantitative measure of "how memorized" a given test input is.

- **Theoretical characterization of the role of α.** In line with our intuitions in Appendix C, we have observed both qualitatively (e.g., Figure F.2) and quantitatively (e.g., Appendix C.1) that estimating datamodels using different values of α identifies correlations at varying granularities. However, despite empirical results around the clear role of α—Appendix C.1 even isolates its effect on datamodels for simple underdetermined linear regression—we lack a crisp *theoretical* understanding of how α affects our estimated datamodels. A better theoretical understanding of the role of α, even for simple models trained on structured distributions, can provide us with more rigorous intuition for the phenomena observed here, and can in turn guide the development of better choices of sampling distribution for datamodeling.

I.3. Applying datamodels

Finally, each of the presented perspectives in Sections 4 and 5 can be taken further to enable even better data and model understanding. For example:

- **Interpreting predictions.** For a given test example, the training images corresponding to the largest-magnitude datamodel weights both (a) share features in common with the test example; and (b) seem to be causally linked to the test example (in the sense that removing the training images flips the test prediction). This immediately suggests the potential utility of datamodels as a tool for *interpreting* test-time predictions in a counterfactual-centric manner. Establishing them as such requires further evaluation through, for example, human-in-the-loop studies.

- **Building data exploration tools.** In a similar vein, another opportunity for future work is in building user-friendly *data exploration* tools that leverage datamodel embeddings. In this paper we present the simplest such example in the form of PCA, but leave the vast field of data bias and feature discovery methods (cf. (Carter et al., 2019) and Leclerc et al. (2021) for a survey) unexplored.

J. Connection between Influence Estimation and Datamodels

J.1. Proof of Lemma 1

Lemma 1. *Fix a training set S of size n, and a test example x. For $i \in [m]$, let S_i be a random variable denoting a random 50%-subset of the the training set S. Let $\boldsymbol{w}_{infl} \in \mathbb{R}^n$ be the estimated empirical influences (18) onto x estimated using the sets S_i. Let $\boldsymbol{w}_{OLS}$ be the least-squares estimator of whether a particular model will get image x correct, i.e.,*

$$\boldsymbol{w}_{OLS} \coloneqq \arg\min_{w} \frac{1}{m} \sum_{i=1}^{m} \left(w^\top \boldsymbol{z_i} - \mathbf{1}\{\textit{model trained on } S_i \textit{ correct on } x\}\right)^2, \qquad \textit{where } \boldsymbol{z}_i = 2 \cdot \mathbf{1}_{S_i} - \mathbf{1}_n.$$

Then, as $m \to \infty$, $\|(1 + \frac{2}{n})\boldsymbol{w}_{OLS} - \frac{1}{2}\boldsymbol{w}_{infl}\|_2 \to 0$.

Proof. For convenience, we introduce the $m \times n$ binary *mask matrix* $\boldsymbol{A}$ such that $\boldsymbol{A}_{ij}$ is an indicator for whether the j-th training image was included in S_i. Note that $\boldsymbol{A}$ is a random matrix with fixed row sum of $n/2$. Next, we define the *output vector* $\boldsymbol{y} \in \{0,1\}^m$ that indicates whether a model trained on S_i was correct on x. Finally, we introduce the *count matrix* $\boldsymbol{C} = \text{diag}(\mathbf{1}^\top A)$, i.e., a diagonal matrix whose entries are the columns sums of $\boldsymbol{A}$, e.g. the number of times each example appears across m different masks.

We begin with $\boldsymbol{w}_{OLS}$. Consider the $n \times n$ matrix $\boldsymbol{\Sigma} = \frac{1}{m}\boldsymbol{Z}^\top\boldsymbol{Z} = \frac{1}{m}(2 \cdot \boldsymbol{A} - \mathbf{1}_{m\times n})^\top(2 \cdot \boldsymbol{A} - \mathbf{1}_{m\times n})$. The diagonal entries of this matrix are $\boldsymbol{\Sigma}_{ii} = 1$ (due to $\boldsymbol{A}$ having constant row sum), while the off-diagonal is

$$\boldsymbol{\Sigma}_{ab} = \frac{1}{m}\sum_{i=1}^{m} \begin{cases} +1 & \text{if training image } x_a, x_b \in S_i \text{ or } x_a, x_b \notin S_i \\ -1 & \text{otherwise.} \end{cases}$$

Since $\boldsymbol{\Sigma}$ has bounded entries ($|\Sigma_{ab}| \leq 1$), we have that for fixed n, $\lim_{m\to\infty} \boldsymbol{\Sigma} = \mathbb{E}[\boldsymbol{\Sigma}]$, and in particular

$$\boldsymbol{\Sigma}_{ab} \to \mathbb{P}(x_a, x_b \in S_i \text{ or } x_a, x_b \notin S_i) - (1 - \mathbb{P}(x_a, x_b \in S_i \text{ or } x_a, x_b \notin S_i))$$

$$\mathbb{P}(x_a, x_b \in S_i \text{ or } x_a, x_b \notin S_i) = 2 \cdot \left(\frac{\frac{n}{2}}{n} \cdot \frac{\frac{n}{2} - 1}{n}\right) = \frac{1}{2} - \frac{1}{n}$$

$$\text{Thus, } \boldsymbol{\Sigma}_{ab} \to -\frac{1}{2n}.$$

Now, using the Sherman-Morrison formula,

$$\boldsymbol{\Sigma}^{-1} = \frac{n}{n+2}\left(\boldsymbol{I} + \frac{2}{n}\mathbf{1}_{n\times n}\right)$$

By construction, the row sums of $\boldsymbol{Z} = 2 \cdot \boldsymbol{A} - \mathbf{1}_{m\times n}$ are 0, and so $\mathbf{1}_{n\times n} \cdot \boldsymbol{Z}^\top = 0$. Thus,

$$\boldsymbol{w}_{OLS} = (\boldsymbol{Z}^\top\boldsymbol{Z})^{-1}\boldsymbol{Z}^\top\boldsymbol{y} = \frac{1}{m}\left(\frac{1}{m}\boldsymbol{Z}^\top\boldsymbol{Z}\right)^{-1}\boldsymbol{Z}^\top\boldsymbol{y} = \frac{1}{m} \cdot \frac{n}{n+2}\boldsymbol{Z}^\top\boldsymbol{y}.$$

We now shift our attention to the empirical influence estimator $\boldsymbol{w}_{infl}$. Using our notation, we can rewrite the (vectorized) empirical influence estimator (18) as:

$$\begin{aligned}
\boldsymbol{w}_{infl} &= \boldsymbol{C}^{-1}\boldsymbol{A}^\top\boldsymbol{y} - (m \cdot \boldsymbol{I}_n - \boldsymbol{C})^{-1}\left(\mathbf{1}_{m\times n} - \boldsymbol{A}\right)^\top \boldsymbol{y} \\
&= \left(\boldsymbol{C}^{-1} - (m \cdot \boldsymbol{I}_n - \boldsymbol{C})^{-1}\right)\boldsymbol{A}^\top\boldsymbol{y} - (m \cdot \boldsymbol{I}_n - \boldsymbol{C})^{-1}\mathbf{1}_{m\times n}^\top\boldsymbol{y} \\
&= m \cdot \boldsymbol{C}^{-1}\left(m \cdot \boldsymbol{I}_n - \boldsymbol{C}\right)^{-1}\boldsymbol{A}^\top\boldsymbol{y} - (m \cdot \boldsymbol{I}_n - \boldsymbol{C})^{-1}\mathbf{1}_{m\times n}^\top\boldsymbol{y} \\
&= (m \cdot \boldsymbol{I}_n - \boldsymbol{C})^{-1}\left(m \cdot \boldsymbol{C}^{-1}\boldsymbol{A}^\top - \mathbf{1}_{m\times n}^\top\right)\boldsymbol{y}.
\end{aligned}$$

Now, as $m \to \infty$ for fixed n, the random variable $m\boldsymbol{C}^{-1}$ converges to $2 \cdot \boldsymbol{I}$ with probability 1. Thus,

$$m \cdot \boldsymbol{A}\boldsymbol{C}^{-1} - \mathbf{1}_{m\times n} \to 2 \cdot \boldsymbol{A} - \mathbf{1}_{m\times n},$$

and the empirical influence estimator $\boldsymbol{w}_{infl} \to \frac{2}{m}\boldsymbol{Z}^\top\boldsymbol{y}$, which completes the proof. □

Algorithm	# models (m)	Output type	Spearman r	MSE	AUC	Difference
Diff. of means	25,000	Correctness	0.028	N/A	0.529	
Diff. of means	100,000	Correctness	0.053	N/A	0.555	Under $\rightarrow$ Over-determined
Diff. of means	100,000	Margin	0.213	2.052	0.653	Output type
LASSO	100,000	Margin	0.320	1.382	0.724	Explicit datamodel

Table J.1: **Disentangling the effect of different factors in datamodel performance.** Each row shows a different estimator for datamodels. We begin with the empirical influence (or difference of means) on correctness computed with 25,000 models, which is in the overparameterized regime (as there are $d = 50,000$ variables). Then, we increase the number of models to an underparamterized regime. Next, we change the output type from correctness to margins. Lastly, we change the estimation algorithm from difference of means (which is approximately equivalent to OLS, as shown in Appendix J.1) to LASSO. Each of these changes brings about significant gains in the signal captured by datamodels, as measured by Spearman rank correlation, MSE, or AUC.

J.2. Evaluating influence estimates as datamodels

Lemma 1 suggests that we can re-cast empirical influence estimates as (rescaled) datamodels fit with least-squares loss. Under this view, (i.e., ignoring the difference in conceptual goal), we can differentiate between explicit datamodels and those arising from empirical influences along three axes:

- **Estimation algorithm**: Most importantly, datamodels *explicitly* minimize the squared error between true and predicted model outputs. Furthermore, datamodels as instantiated here use (a) a sparsity prior and (b) a bias term which may help generalization.
- **Scale**: Driven by their intended applications (where one typically only needs to estimate the highest-influence training points for a given test point), empirical influence estimates are typically computed with relatively few samples (i.e., $m < d$, in our setting) (Feldman & Zhang, 2020). In contrast, we find that for datamodel loss to plateau, one needs to estimate parameters using a much larger set of models.
- **Output type**: Finally, datamodels do not restrict to prediction of a binary correctness variable—in this paper, for example, for deep classification models we find that *correct-class margin* was best both heuristically and in practice.

In this section, we thus ask: how well do the rescaled datamodels that arise from empirical influence estimates predict model outputs? We address this question in the context of the three axes of variation described above. In order to make results comparable across different outputs types (e.g., correctness vs. correct-class margin), we measure correlation (in the sense of Spearman (1904)) between the predicted and true model outputs, in addition to MSE where appropriate. To ensure a conservative comparison, we also measure performance as a predictor of *correctness*. In particular, we treat $\boldsymbol{w}^\top \mathbf{1}_{S_i}$ as a continuous predictor of the binary variable $\mathbf{1}\{\text{model trained on } S_i \text{ is correct on } x\}$, and compute the AUC of this predictor (intuitively, this should favor empirical influence estimates since they are computed using correctnesses directly).

In Table J.1 we show the difference between empirical influence estimates (first row) and our final datamodel estimates (last row), while disentangling the effect of the three axes above using the rows in between. As expected, there is a vast difference in terms of correlation between the original empirical influence estimates and explicit datamodels. We further illustrate this point in Figure J.1, where we show how the correlation, MSE, and AUC vary with m for both empirical influence estimates and datamodels, as well as an intermediate estimator that uses the estimation procedure of empirical influence estimates but replaces correctness with margin.

J.3. Testing Lemma 1 empirically

In this section, we visualize the performance of empirical influence estimates ((Feldman & Zhang, 2020)) as datamodels. In Figures J.2a and J.2b we plot the distributions of $\boldsymbol{w}_{infl}^\top \mathbf{1}_{S_i} | y_i$ for different CIFAR-10 test examples; Figure J.2a shows these "conditional prediction distributions" for subsets S_i that were used to estimate the empirical influence, while Figure J.2b shows the corresponding distributions on held-out (unseen) subsets S_i. The figures suggest that (i) indeed, empirical influences are somewhat predictive of the correctness y_i, (ii) their predictiveness increases as number of samples $m \rightarrow \infty$

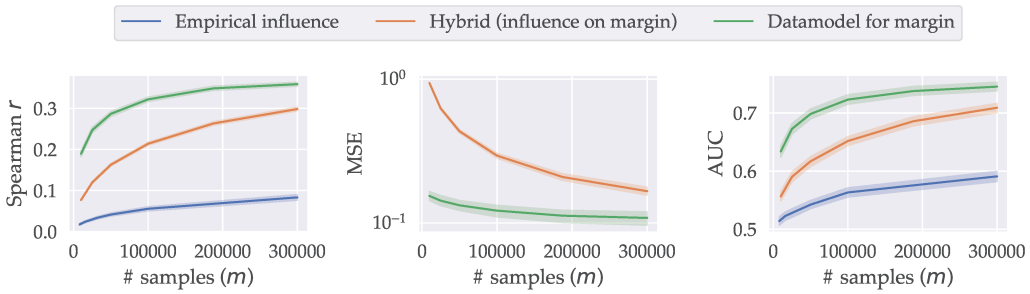
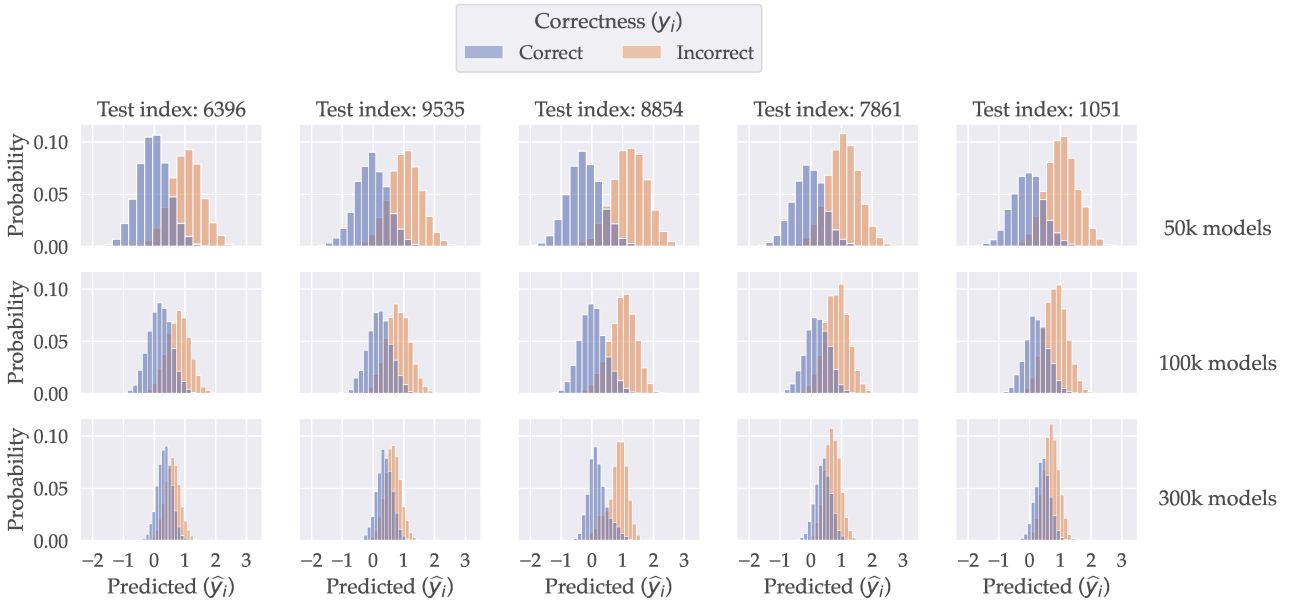
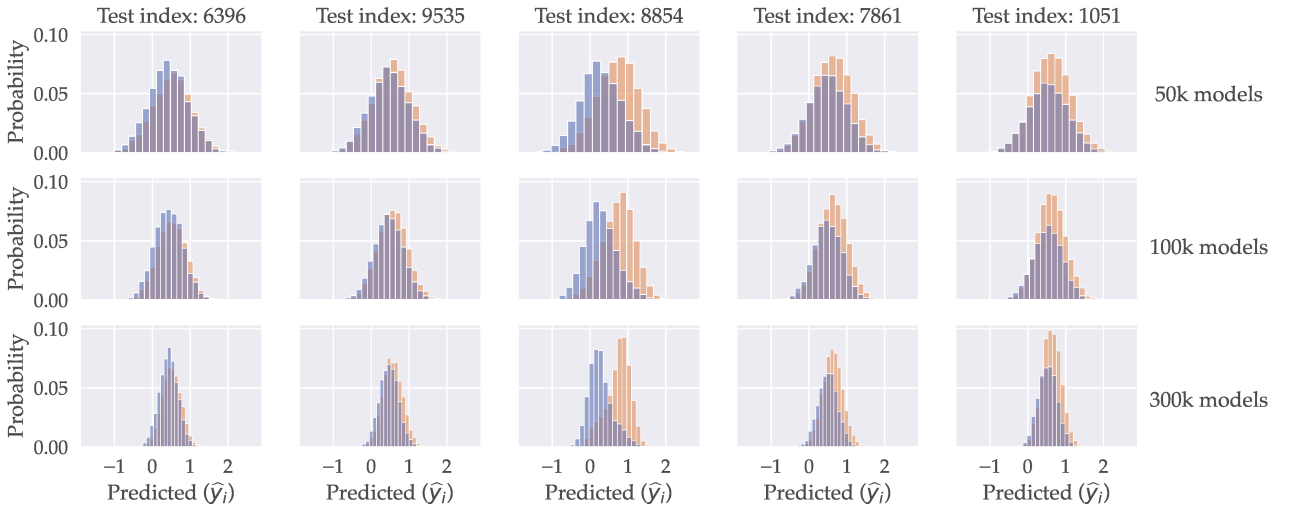


Figure J.1: Datamodels have significantly better sample complexity than empirical influences. We compare three estimators—empirical influence, empirical influence on margins, and ℓ_1-regularized linear regression on margins (datamodels)—across a wide range of sample sizes on three different metrics. All metrics are averaged over the entire test set (i.e. over 10,000 datamodels). For MSE, we only show the estimators on margins as different output types are incomparable. Across all metrics, datamodels capture significantly more signal than empirical influences using the same number of samples. Conversely, datamodels need far fewer samples to reach the same level of performance.

but is still rather low, and (c) a significant amount of the prediction error is generalization error, as the train predictions in Figure J.2a are significantly better-separated than the heldout predictions in Figure J.2b.



(a) Train masks



(b) Heldout masks

Figure J.2: Empirical influence estimates are (weak) datamodels. Each histogram illustrates the performance of empirical influences when the output of the corresponding datamodel is used as a statistic to distinguish between correct and incorrect predictions on the target example. Empirical influences can predict correctness on the "train set" of subsets (i.e. the masks used to estimate them), but suffer from significant generalization error when evaluated on a held-out set of subsets.

J.4. View of empirical influences as a Taylor approximation

Lemma 1 shows that we can interpret empirical influences as (rescaled) estimates of the weights of a *linear* datamodel. Here, we give an alternative intuition for why this is the case, even though the definition of empirical influence does not explicitly assume linearity anywhere: we show that the influences define a first-order Taylor approximation of the multilinear extension f of our target function F of interest, where the influences (approximately) correspond to first-order derivatives of f.

Recall that we want to learn some output of interest $F : 2^T \to \mathbb{R}$, say the probability of correctness on a test example z, as a function of the examples $S \subset T$ included in the training set. We first extend this function continuously so that we can take its derivatives. The multilinear extension (Owen, 1972) of set function F to the domain $[0,1]^n$ ($|T| = n$) is given by:

$$f(x) = \sum_{S \subseteq T} F(S) \prod_{i \in S} x_i \prod_{i \notin S} (1 - x_i) \tag{19}$$

$f(x)$ also has an intuitive interpretation: it is the expected value of $F(S)$ when S is chosen by including each x_i in the input with probability x_i.

Next, we take the derivative of f w.r.t. to the input x_i:

$$\frac{\partial f}{\partial x_i} = \underbrace{\sum_{S \subseteq T, i \in S} F(S) \prod_{j \in S, j \neq i} x_j \prod_{j \notin S} (1 - x_j)}_{\mathop{\mathbb{E}}_{s_j \sim Bern(x_j), s_i = 1} F(S)} - \underbrace{\sum_{S \subseteq T, i \notin S} F(S) \prod_{j \in S} x_j \prod_{j \notin S, j \neq i} (1 - x_j)}_{\mathop{\mathbb{E}}_{s_j \sim Bern(x_j), s_i = 0} F(S)}$$

Note that because f is multilinear, the derivative w.r.t. to x_i is constant in x_i, but not w.r.t. to other x_j. Now, observe that the above expression evaluated at $x_j = \alpha$, $\forall x_j$ corresponds approximately[21] to α-subsampled influence θ_i, of i on F: the first term corrresponds (using our earlier interpretation) to the expectation of $F(S)$ conditional on S including i, and the second to that conditional on S excluding i.

Finally, the first-order Taylor approximation of f around an x is given as:

$$f(x) \approx F(\emptyset) + \sum_i \frac{\partial f}{\partial x_i} \cdot x_i \approx F(\emptyset) + \sum_i \theta_i \cdot x_i$$

where θ_i are the empirical influences.

The role of α. The above perspective provides an alternative way to think the role of the sampling fraction α. The weights θ_i depend on the regime we are interested in; if we use α-subsampled influences, then we are effectively taking a local linear approximation of f in the regime around $\vec{x} = \alpha \cdot \vec{1}$.

Remark. Though we include the exposition above for completeness, this is a classical derivation that has appeared in similar form in prior works (Owen, 1972). Another connection is that *Shapley value* is equivalent to the integral of f along the "main diagonal" of the hypercube; it is effectively empirical influences averaged uniformly over the choice of α.

[21] There are two sources of approximation here. First, the α-subsampling used in our datamodel definition is defined globally (e.g. α fraction of entire train set), which is different from the i.i.d. $Bern(\alpha)$ sampling that is considered here. Second, we only observe noisy versions of $F(S)$.