A review on Datamodels: a new approach to reveal connection between data and prediction

Yuichi Kajiura
Khoury Colledge of Computer Sciences
Northeastern University
Boston, MA 02115
kajiura.y@northeastern.edu

Abstract

We all know that a prediction is a combination of data and algorithm. However, we rarely understand how algorithm actually use the data to come up with a particular prediction. Datamodeling, a framework proposed by Ilyas et al. [2022] is a new approach to solve this issue by casting the relationship between data and prediction from complex models into a simple linear function. This paper summarizes the framework and discuss its potential and challenge, particularly towards applying the framework to large dataset that are used in industries.

1 Introduction

As recent success in machine learning attracts industries to utilize the technology, the demand on model explainability is rapidly growing. One way to satisfy this requirement is to develop a model that indicates a relevant feature utilized in the prediction [Ribeiro et al., 2016], and another is to show similar example in the training data [Koh and Liang, 2017]. However, achieving these goal in a robust manner is difficult because in modern learning algorithm (e.g. neural networks) model performance and individual prediction can significantly differ by just a random seed[D'Amour et al., 2020] or small noise[Zhong et al., 2021].

Ilyas et al. [2022] recently proposed a new approach, datamodeling, to overcome this challenge by smoothing out the random behavior of individual models through training hundreds of thousands of models and looking at data through a lens of these models - meaning that, using the trained models as a new dataset, they trained a simpler surrogate model for each target example that directly map relationships between original dataset and model predictions on that example. In the study, authors showed that a simple linear datamodel can predict outcome of end-to-end training of deep neural networks, and demonstrated a variety of applications of datamodels such as finding similar examples in dataset and embedding data into feature representation space and apply graph theoretic tools such as spectral clustering.

This paper aims to summarize the framework in relation to the other lines of work on model explainability and discuss its potential as well as practicality. Particularly, given the huge computational effort the framework requires, we touch on implementation details and estimate required resource to apply the framework to larger dataset, such as ImageNet.

2 Preliminaries

2.1 Datamodeling framework

2.1.1 Setting

Consider a machine learning setup, learning algorithm A using training dataset S. Now, consider a fixed input x and define

$$f_A(x; S) := \text{the outcome of training model on } S \text{ using } A \text{ and evaluating on } x,$$

where outcome can be variety of measures for example the cross-entropy loss of a classifier on x, or the squared-error of a regression model on x.

2.1.2 Goal

Ilyas et al. [2022] aim to understand how training example x in S combine through learning algorithm A to yield $f_A(x; S)$. We reduce this challenging task to easier problem using a classic technique for studying blackbox function, *surrogate modelling* [Sacks et al., 1989]. That is, approximating the black-box function $f_A(x; \cdot)$ by a simple *surrogate* function $g(S')$ whose output roughly matches $f_A(x; S')$ for a variety of training set S' (see Figure 1 for illustration).

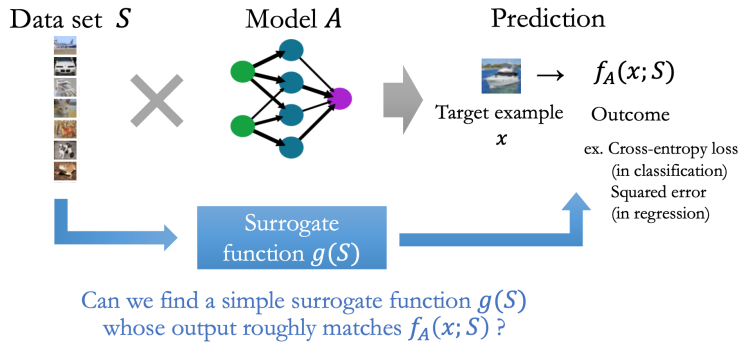

Figure 1: Illustration of surrogate modeling in the context of Ilyas et al. [2022].

2.1.3 Datamodeling

By parameterizing the surrogate function as g_θ, Ilyas et al. [2022] convert the task of constructing a surrogate function as *supervised learning* problem trained from a collection of subsets $S' \subset S$ and corresponding $f_A(x; S')$. The framework is consisted from 5 steps:

Step 1: Define g_θ. In Ilyas et al. [2022], it is defined as a simple linear model

$$g_\theta(S') := \theta^\top \mathbf{1}_{S'} + \theta_0,$$

where $\theta \in \mathbb{R}^{|S|}$ and

$$\mathbf{1}_{S'} \in \{0,1\}^{|S|} \text{ such that } (\mathbf{1}_{S'})_j = \begin{cases} 1 \text{ if } z_j \in S' \\ 0 \text{ otherwise,} \end{cases}$$

Step 2: Collect subsets of training data $(S_1, ..., S_m)$ using a fixed uniform distribution

$$D_S = \text{Uniform}(\{S' \subset S : |S'| = \alpha|S|.\}),$$

where $\alpha \in (0, 1)$ is a *subsampling fraction*.

Step 3: Using the collected subsets and learning algorithm A (ex. ResNet), train a collection of models $(M_1, ..., M_m)$.

Step 4: Evaluate trained models on each datapoint x_j in training set and test set. That is, calculate

$$F = \begin{bmatrix} f_A(x_1; S_1) & f_A(x_2; S_1) & \dots & f_A(x_n; S_1) \\ f_A(x_1; S_2) & f_A(x_2; S_2) & \dots & f_A(x_n; S_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_A(x_1; S_m) & f_A(x_2; S_m) & \dots & f_A(x_n; S_m) \end{bmatrix},$$

where Ilyas et al. [2022] instantiate $f_A(x_j; S_i)$ as *correct class margin*:

$$f_A(x; S') := (\text{logit for correct class}) - (\text{highest incorrect logit})$$

Step 5: Train datamodels $g_{\theta_{x_j}}$ for each datapoint x_j by using the collected *datamodel training set*,

$$(S_1, f_A(x_j; S_1)), ..., (S_m, f_A(x_j; S_m)),$$

and by minimizing a fixed loss function L. That is,

$$\theta_{x_j} = \arg\min_w \frac{1}{m} \sum_{i=1}^{m} L(g_w(S_i), f_A(x_j; S_i)) \;\; \forall x_j \in \text{training set and test set.}^1$$

Ilyas et al. [2022] assume that predictions on a given target will depend on limited number of related training examples and incorporate such a *sparsity prior* by adding l_1 regularization, i.e., setting

$$L = \left(w^\top \mathbf{1}_{S_i} - f_A(x_j; S_i)\right)^2 + \lambda_j ||w||_1$$

Finally, we obtain a collection of datamodels (See Figure 2 for an illustration of these steps).

$$G = [g_{\theta_1} \; ... \; g_{\theta_n}]^\top.$$

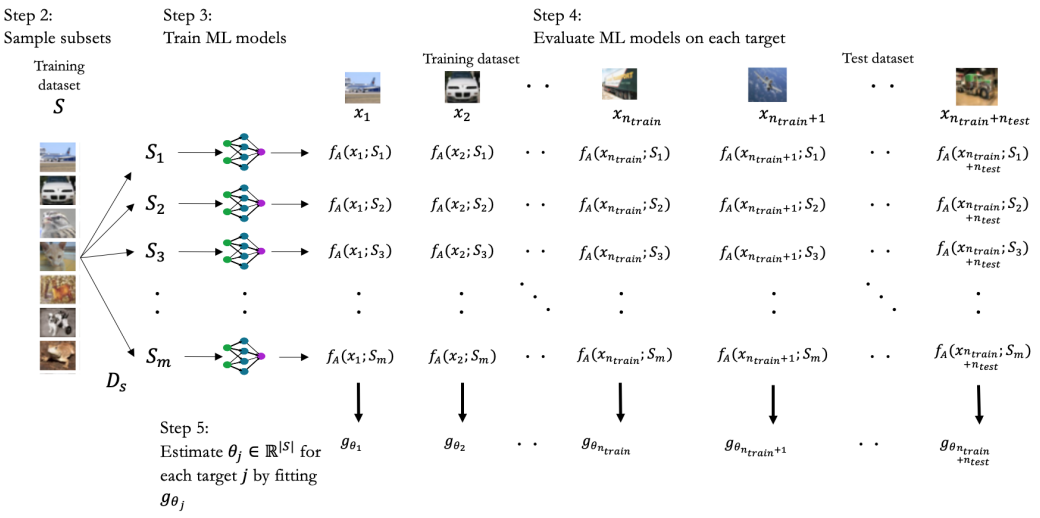

Figure 2: Illustration of Datamodeling framework (from Step 2 to Step 5).

2.2 Related works

2.2.1 Influence function

Empirical Influence Influence function is a method for finding similar training images by estimating *empirical influence* – the effect of removing a single training image x_i on the loss for a given test image x_j, i.e.,

[1] When x_j is in the training set, Leclerc et al. [2022] slightly altered the objective to exclude training set S_i containing the target example x_j

$\text{Infl}[x_i \to x_j] := \mathbb{P}(\text{model trained on } S \text{ is correct on } x_j) - \mathbb{P}(\text{model trained on } S \backslash x_i \text{ is correct on } x_j)$.

A standard approach to compute this value is to use first order approximation. Koh and Liang [2017] develop an efficient implementation for modern machine learning settings that requires only oracle access to gradients and Hessian-vector products. Ilyas et al. [2022] show that using this approach, particularly taking pretrained model(ResNet-9) and computing approximate influence function with respect to only the parameters in the last linear layer often fails to find similar training examples.

Feldman and Zhang [2020] argue that understanding the dynamics of the penultimate layer is insufficient for understanding deep models' decision mechanism and propose another approach to approximate empirical influence by:

$$\begin{aligned}\widehat{\text{Infl}}[x_i \to x_j] := \mathbb{P}_{S \sim D_S}(\text{ model trained on } S \text{ is correct on } x_j | x_i \in S) \\ - \mathbb{P}_{S \sim D_S}(\text{ model trained on } S \text{ is correct on } x_j | x_i \notin S).\end{aligned}$$

This estimator improves sample efficiency by reusing the same set of models to compute influences between different input pairs, and thus is typically computed with relatively few samples (i.e. $m < d(= |S|)$ in our setting) in contrast to datamodels. Ilyas et al. [2022] show that we can cast this estimator as a rescaled datamodel (trained by using zero-one loss of ML models).

TracIn Pruthi et al. [2020] propose a gradient based approximation of influence – the influence on a training example z on a test example z' is a total change in loss on z' contributed by updates from mini-batches containing z. They approximate this in practice by considering checkpoints $\theta_{t_1}, ...\theta_{t_k}$ across training where updates between each checkpoints includes all the training examples once, and sum the dot product of the gradients at z and z' at each checkpoint:

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

where $\{\eta_i\}$ is learning rates.

Ilyas et al. [2022] use empirical influence and TracIn as benchmarks (though primary purpose of these methods are not for prediction).

2.2.2 Surrogate models for interpretability

Surrogate model from *pixel-space* to predictions are popular tools in machine learning interpretability. For example, LIME [Ribeiro et al., 2016] constructs a local linear model mapping test images to model predictions, aiming to understand, for a $fixed$ model, how the features of a given test example change the prediction. In contrast, detamodels hold the test example fixed and study how the training set change the prediction.

Surrogate models are also used to evaluate data points for active learning and coreset selection. Coleman et al. [2020] find that shallow neural networks trained with fewer epochs can be a good proxy for a larger model.

3 Implementation, Results, and Applications

Ilyas et al. [2022] demonstrate how datamodels can be applied in the context of deep neural network using two datasets: CIFAR-10 [Krizhevsky, 2009] and Functional Map of the World (FMoW, a land use classification dataset based on satellite imagery) [Koh et al., 2020]. Properties of both datasets are summarized in Table 1.

After trained thousands of hundreds of models(ResNet-9 for CIFAR-10 and ResNet18 for FMoW), 50,000 training set datamodels and 10,000 test set datamodels are obtained for CIFAR-10 (each being a linear model g_θ parameterized by a vector $\theta \in \mathbb{R}^{50,000}$); as well as 21,404 training set datamodels and 3,138 test set datamodels are obtained for FMoW (again, parameterized by $\theta \in \mathbb{R}^{21,404}$)

Table 1: Properties of dataset used

Dataset	Classes	Size (Train / Test)	Input Dimensions
CIFAR-10	10	50,000 / 10,000	3×32×32
FMoW	62	21,414 / 3,138	3×224×224

3.1 Implementation

3.1.1 Training ResNet for datamodel training set collection (Step 3 in Section 2.1.3)

In order to learn $\theta \in \mathbb{R}^{|S|}$, the size of datamodel training set to be collected should be sufficiently larger than the size of original training dataset S. As a result, Ilyas et al. [2022] trained 300,000 CIFAR-10 models (ResNet-9) and 150,000 FMoW models (ResNet-18) on $\alpha = 50\%$ subsets of each dataset (See Table 2 for the summary of the model trained).

Table 2: The number of models trained

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

CIFAR-10. Ilyas et al. [2022] used a ResNet-9 variant from Kakao Brain [2] optimized for fast training, chose hyperparameters using grid search and used the standard batch SDG.

FMoW. Ilyas et al. [2022] used the standard ResNet-18 architecture and chose hyperparameters using grid search, including over different optimizer(SGD, Adam).

Computing resource. Ilyas et al. [2022] trained models on a cluster of machines, each with 9 NVIDIA A100 GPUs and 96 CPU cores, using half-precision to increase speed.

Data loading. Ilyas et al. [2022] used FFCV [Leclerc et al., 2022], which removed the data loading bottleneck for smaller models and allowed them achieve a throughput of over 5,000 CIFAR-10 models a day *per* GPU.

3.1.2 Training Datamodels (Step 5 in Section 2.1.3)

Off-the-shelf LASSO solvers requires too much memory or are prohibitively slow for our values of n (the number of datamodels to estimate), m (the number of model trained = the size of datamodel training set) and d (the size of original tasks training set = the input dimensionality of the regression problem). For example, estimating CIFAR-10 datamodels for $\alpha = 50\%$ requires running LASSO with a covariate matrix X of size $50,000(= d) \times 300,000(= m)$, which corresponds to about 60GB of data. Moreover, we need to solve up to $60,000(= n)$ regression problems (one datamodel per each train / test example). Therefore, Ilyas et al. [2022] built a custom solver leveraging the SAGA algorithm of Gazagnadou et al. [2019] and its GPU-enabled implementation of Wong et al. [2021] and adding following changes:

Fast dataloading. At each iteration of SAGA, a minibatch-based algorithm, we have to read B masks (B × 50,000 dimensional binary vector of $\mathbf{1}_{S'}$) and move them onto the GPU for processing. Reading these from disk makes I/O speed a major bottleneck, while pre-loading these into memory makes running multiple regressions on the same machine not possible as those regressions will use the entire RAM disk. To resolve this issue, Ilyas et al. [2022] used the dataloading library FFCV

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

[Leclerc et al., 2022], which is based on memory mapping, and thus arrows for multiple process to read from the same memory.

Simultaneous output. Ilyas et al. [2022] parallelized SAGA algorithm across different instance (sharing the same input matrix) and estimated datamodels for the entire test set (e.g. 10,000 for CIFAR-10) in one pass.

Optimizations. In order to enable the parallelization, Ilyas et al. [2022] significantly reduced the GPU memory footprint of the SAGA solver through simple code optimizations (e.g. using in-place operations rather than copies) and writing a few custom CUDA kernels to speed up and reduce the memory consumption of algorithms (such as soft thresholding or gradient updating).

3.2 Results

To assess the quality of obtained datamodels, Ilyas et al. [2022] compared datamodel predictions $\theta_j^\top \mathbf{1}_{S'}$ to *expected* true model outputs $\mathbb{E}[f_A(x_j; S')]$ for $x_j \in$ test set (10,000 images for CIFAR-10) using $unseen$ subsets (i.e. fresh sample from D_S), where $\mathbb{E}[f_A(x_j; S')]$ is estimated by training 100 models on the same subset S' and averaging their output on x_j. As shown in Figure 3 and Figure 4, the results shows strong linear correlation between datamodel predictions and ground truth. Thus, given a target example x, a simple *linear* datamodel can accurately predict the outcome (correct class margin) of a neural network trained on a random training subset.

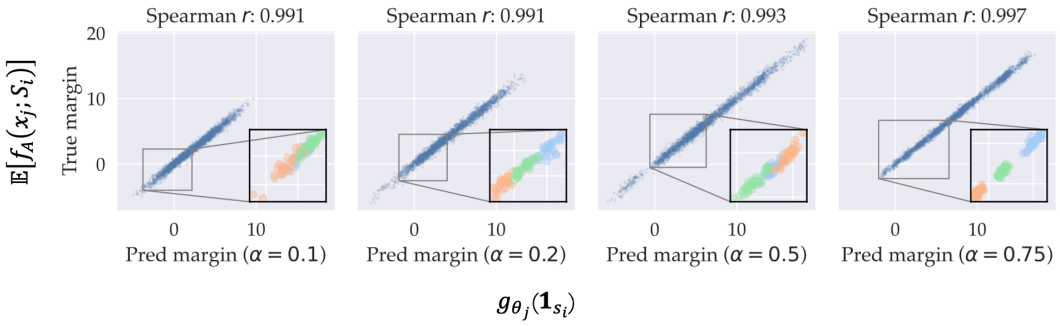

Figure 3: CIFAR-10 datamodels accurately predict margins

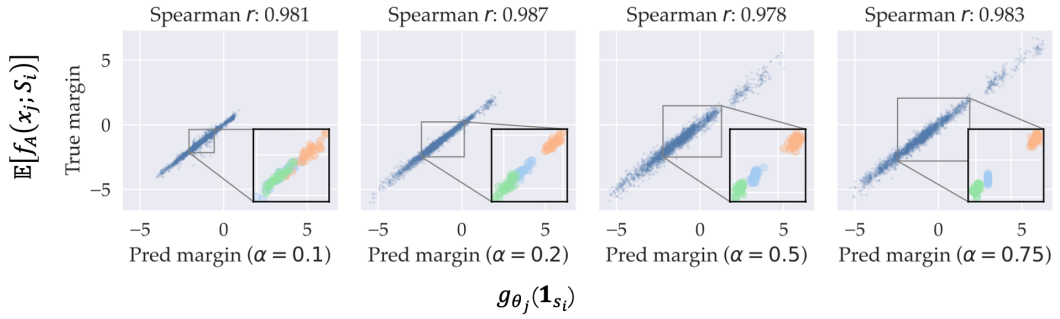

Figure 4: Identical results to Figure 3 for FMoW

3.3 Applications

In the context of the linear datamodels constructed in previous section, each parameter in θ_j of a trained datamodel g_{θ_j} represents a linear relationship between each training example and target example x_j. By leveraging this property, Ilyas et al. [2022] found various applications of datamodels.

3.3.1 Identify brittle prediction

Although datamodels G are constructed for predicting the outcome of training on *random* subsets of the training set, datamodels can in fact be effective proxies for the outcome on an *arbitrary* subset:

$$f_A(x_j; S') \approx g_{\theta_j}(S') \text{ where } S' \nsim D_S.$$

Using this proxy, we want to answer the question: *"How brittle model predictions are on removing training data?"*. To quantify the sensitivity of model behavior towards data points, define *data support* of a target example x_j as

$$\text{SUPPORT}(x_j) = \text{the smallest training subset } R \in S$$
$$\text{s.t. classifiers trained on } S \setminus R \text{ misclassify } x_j \text{ on average}$$

In our context, misclassifying x_j means that $\mathbb{E}[f_A(x_j; S \setminus R)] \leq 0$. To compute SUPPORT($x_j$), we can use $g_{\theta_j}(S \setminus R)$ for guided search: removing subsets of data points from which the value of corresponding parameter in θ_j is the largest, until $g_{\theta_j}(S \setminus R)$ becomes negative. Using this method, Ilyas et al. [2022] estimated SUPPORT(x) for 300 random target examples in the CIFAR-10 test set. For 90% of these examples, the estimated subsets R indeed induced misclassification on the target (meaning that we are not underestimating the effect of removing these examples). As shown in Figure 5, around half of these examples are misclassified by removing a specific 0.4% (or 250 images) of CIFAR-10 training set, while other benchmarks results that removing same number of images flips the prediction of only 10-15% of 300 target examples (meaning that datamodels find much smaller set R that actually flips the results).

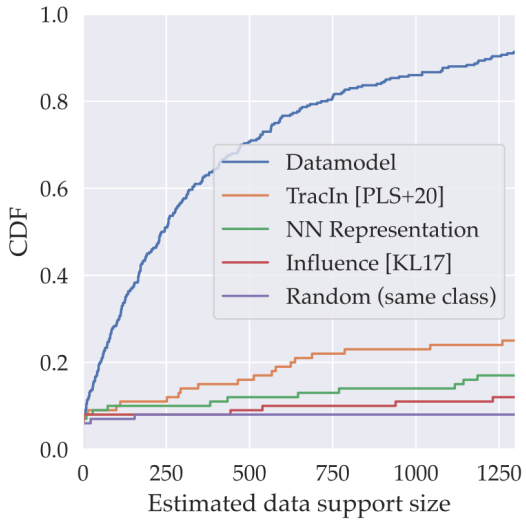

Figure 5: Characterizing brittleness

In addition to the brittleness to *removing* data, Ilyas et al. [2022] also conduct an experiment on the brittleness to *mislabeling* the data using datamodels, and find that, for 50% of the CIFAR-10 test set, mislabeling 35 *target-specific training examples* suffices to flip the corresponding prediction

3.3.2 Finding similar datapoints

Using the same intuition that the highest-magnitude coordinates of θ in datamodel $g_{\theta_j}(S')$ $(= \theta^\top \mathbf{1}_{S'})$ correspond to the training examples whose presence is most predictive of model behavior on target example x_j, Ilyas et al. [2022] show in Figure 6. that these high-magnitude training examples visually resemble target image for randomly chosen test examples. We can leverage this property for finding train-test leakage.

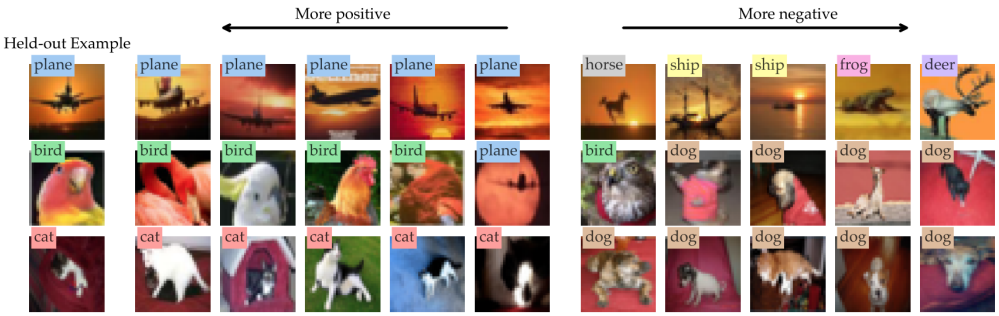

Figure 6: Large detamodel weights correspond to similar images

3.3.3 Feature embeddings

While applications above are *per-example* level, datamodels can also uncover *global structure* of data. The idea is to consider datamodel parameters θ_j as *a feature embedding* of target example x_j into the Euclidean space $\mathbb{R}^{|S|}$. Since g_θ is a linear function of each point in training sets, the coordinate of the datamodel embedding have a *consistent* interpretation across datamodel embeddings even for different target examples. That is, we expect similar target examples to be act 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 set of training examples, their datamodel embeddings will be orthogonal. Embedding an entire dataset S of examples $\{x_j\}$ as a set if feature vectors $\{\theta_j \in R^{|S|}\}$, we may uncover structure in the set of examples by looking for structure in their datamodel embeddings.

Ilyas et al. [2022] demonstrate the potential of datamodel embeddings through two applications.

Clustering Given two example x_1 and x_2, datamodel embeddings induce a natural $similaritymeasure$ between them:

$$d(x_1, x_2) := K(\theta_1, \theta_2),$$

where $K(\cdot, \cdot)$ is any kernel function (Ilyas et al. [2022] use RBF kernel for experiment). For a set of k target examples $\{x_1, ..., x_k\}$, we can compute a full *similarity matrix* $A \in R^{k \times k}$ whose entries are

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

We can view this similarity matrix as an *adjacency matrix* for a dense graph connecting all the examples $\{x_1, ..., x_k\}$, where similar examples have high-weight edges, and unrelated examples 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. As an example, Ilyas et al. [2022] perform spectral clustering [3]. The results for CIFAR-10 test set show it finds subpopulations in datasets (Figure 7).

PCA Recalling that datamodel embeddings are high-dimensional and sparse, we can leverage a canonical tool for finding structure in high-dimensional data: principal component analysis (PCA) [4]. Ilyas et al. [2022] apply PCA to the collection of datamodel embeddings for the CIFAR-10 training set, and compute new k-dimensional embeddings for each target example in both the training set and test set. As Figure 8 visualizes for a few sample coordinate indices $i \in k$, the target examples whose transformed embeddings have a large i-th coordinate share a common feature. Furthermore, for a

[3] 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 minimizes the total weight of inter-cluster edges

[4] 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 map any embedding $\varphi(x) \in \mathbb{R}^d$ to a new embedding $\tilde{\phi}(x) \in \mathbb{R}^k$, such that $\tilde{\phi}(x) = M \cdot \varphi(x)$ for a fixed $k \times d$ matrix M that preserves as much information as possible. The rows of M is called the first *k principal components* of the dataset

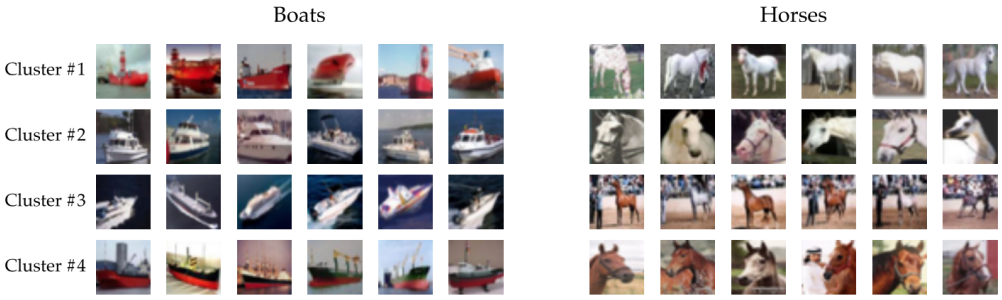

Figure 7: Spectral clustering on datamodel embeddings finds subpopulations

given coordinate, the most positive images and the most negative images either (a) have a different label but share the same common feature or (b) have the same label but differ along the relevant feature.

Figure 8: PCA on datamodel embeddings

4 Discussion

4.1 Applying datamodels to larger datasets

Given a huge computational resource required to construct datamodels, we will discuss here challenges in applying datamodels for larger dataset. Specifically, we consider a problem of applying datamodels to car category of ImageNet [Gebru et al., 2017] that consists of about 700,000 images (10 times larger than CIFAR-10 in terms of number, but 20 times smaller of the whole ImageNet dataset).

Training ML models Suppose we use 600,000 images out of 700,000 as a training data set and use ResNet-18 as a learning algorithm to approximate (Though the state-of-the-art model using for

ImageNet is ResNet-50 [5], we suppose smaller model can suffice the purpose of learning meaningful datamodels that discover the structure of dataset).

According to the benchmark in Leclerc et al. [2022], the training time of ImageNet using ResNet-18 (16 Epochs, 86.8% top5 accuracy) with one A100 GPU is 35 minutes. Therefore, we can estimate the time for training a ResNet-18 model on ImageNetCar dataset is 0.875 min per A100 GPU (when $\alpha = 0.5$). Assuming that we need datamodel training set 5 times larger than the dimensionality of datamodels (as Ilyas et al. [2022] did for CIFAR-10 and FMoW for $\alpha = 0.5$), we need to train 3 million models, which costs 43,750 GPU hours and $38,062.5 (when using A100 on Google Cloud[6]), making it impractical to apply datamodels to large dataset currently used in industries. Possible mitigation for this problem might be performing data selections [Coleman et al., 2020] and construct a linear datamodels of selected points.

Training datamodels At each iteration of regression it requires to read B $\times$ 600,000 dimentional binary vectors, which corresponds to about 40MB of data (when B = 512 as in Ilyas et al. [2022]). Solving for a datamodel requires about 6,000 such iterations (assuming only one pass of 3M pairs of datamodel training set), and we need to solve it for up to 700k datamodels. Though the details of FFCV[Leclerc et al., 2022] is not yet published, it may be infeasible to solve such a large problem by the SAGA based solver. In this setting, we will need to apply techniques from numerical optimization to solve large linear systems efficiently [Martinsson and Tropp, 2020], as suggested by Ilyas et al. [2022].

5 Conclusion

Datamodeling is an innovative approach to understand the relationship between data and prediction of complex model class, enabling a myriad of applications such as detecting similar data points and demystifying global data structure by interpreting datamodels as feature representations of each datapoint and applying graph-theoretic tools onto the adjacency matrix consisted from such feature representations. However, a huge computational resource required for constructing datamodels limit its application to small datasets. Therefore, further development should focus on how to reduce computational burden without sacrificing its accuracy, for example by combining point selection method or applying numerical optimization to solve large linear systems.

References

C. Coleman, C. Yeh, S. Mussmann, B. Mirzasoleiman, P. Bailis, P. Liang, J. Leskovec, and M. Zaharia. Selection via proxy: Efficient data selection for deep learning. In *International Conference on Learning Representations (ICLR)*, 2020.

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

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

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

T. Gebru, J. Krause, J. Deng, and L. Fei-Fei. Scalable annotation of fine-grained categories without experts. In *2017 CHI Conference on Human Factors in Computing System(CHI 2017)*, 2017.

[5] https://dawn.cs.stanford.edu/benchmark/

[6] https://cloud.google.com/blog/products/compute/a2-vms-with-nvidia-a100-gpus-are-ga

A. Ilyas, S. M. Park, L. Engstrom, G. Leclerc, and A. Madry. Datamodels: Predicting predictions from training data. In *Arxiv preprint arXiv:2202.00622*, 2022.

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

P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, S. Beery, J. Leskovec, A. Kundaje, E. Pierson, S. Levine, C. Finn, and P. Liang. Wilds: A benchmark of in-the-wild distribution shifts. In *arXiv preprint arXiv:2012.07421*, 2020.

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

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

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

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

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

J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science Vol. 4. 4.*, 4.4(409–423), 1989.

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

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