Instance explanations

CSEP 590B: Explainable AI

Ian Covert & Su-In Lee
University of Washington

Motivation

- So far, we've focused on the role of *features*, *concepts* and *neurons*
- We haven't considered an important ingredient for ML models: **the data**
  - Which training examples influence an individual prediction?
  - How does each example contribute to the model's accuracy?

Instance explanations

- Consider training the model with subsets of the training dataset
- This can help understand the influence of individual data examples
  - Either on an individual prediction, or on global model behavior (e.g., accuracy)
- Score the training examples
  - Identify valuable or problematic examples
  - Determine possible changes to the dataset

Analogy to removal-based explanations

- Recall: removal-based explanations measure the impact of holding out **features (columns)**
- Here, we'll measure the impact of holding out **training data (rows)**

- Clear parallels:
  - How to efficiently remove training examples?
  - Which model behavior to explain?
  - How to summarize a sample's importance?

Today

- Section 1
  - Counterfactual explanations
- Section 2
  - Leave-one-out
  - Data Shapley
  - Monitoring training dynamics

Main idea

- Score each example by the *leave-one-out* approach
  - Modify training dataset by removing only that example, then retrain the model
  - Measure change in the desired quantity (e.g., accuracy)
- Can be computationally costly, but there are special cases and approximations

Brute force approach

- Simply retrain the model with each new dataset
- Practical when model training is fast and dataset size is not too large
  - Linear regression
  - Random forests
  - Can warm-start initialization (e.g., linear/logistic regression)
- Otherwise, can be very slow
  - E.g., neural networks, large datasets

Data deletion

- Train a model with the full dataset
  - Then update the model (exactly or approximately) to reflect data deletion
- Approximations for linear regression
  - Cook & Weisberg, "Residuals and influence in regression" (1982)
  - Izzo et al., "Approximate data deletion from machine learning models" (2021)
- Exact approach for random forests
  - Brophy & Lowd, "Machine unlearning for random forests" (2020)
- Approximation for deep learning models
  - Koh & Liang, "Understanding black-box predictions via influence functions" (2017)

Example result

Accurate for linear regression

Decently accurate for CNNs

Approximating loss change from removing single samples

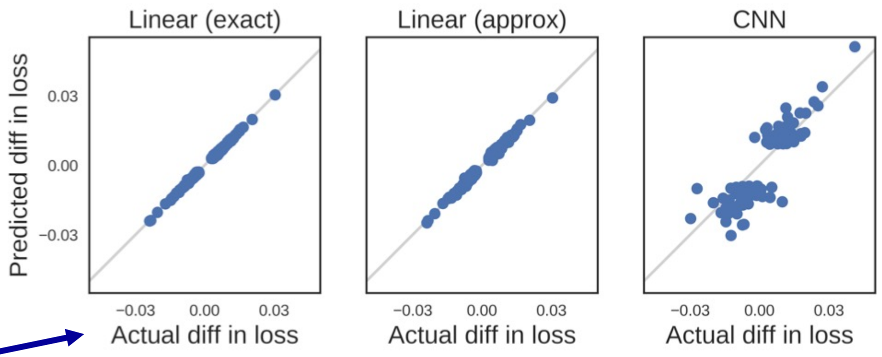


Figure 2. **Influence matches leave-one-out retraining.** We arbitrarily picked a wrongly-classified test point z_{test}, but this trend held more broadly. These results are from 10-class MNIST. **Left:** For each of the 500 training points z with largest $|\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})|$, we plotted $-\frac{1}{n} \cdot \mathcal{I}_{\text{up,loss}}(z, z_{\text{test}})$ against the actual change in test loss after removing that point and retraining. The inverse HVP was solved exactly with CG. **Mid:** Same, but with the stochastic approximation. **Right:** The same plot for a CNN, computed on the 100 most influential points with CG. For the actual difference in loss, we removed each point and retrained from $\tilde{\theta}$ for 30k steps.

Koh & Liang, "Understanding black-box predictions via influence functions" (2017)

Example result (cont.)

Introducing and then identifying mislabeled examples

Influence function approximation outperforms ranking by loss value

Checking and correcting examples in different orders

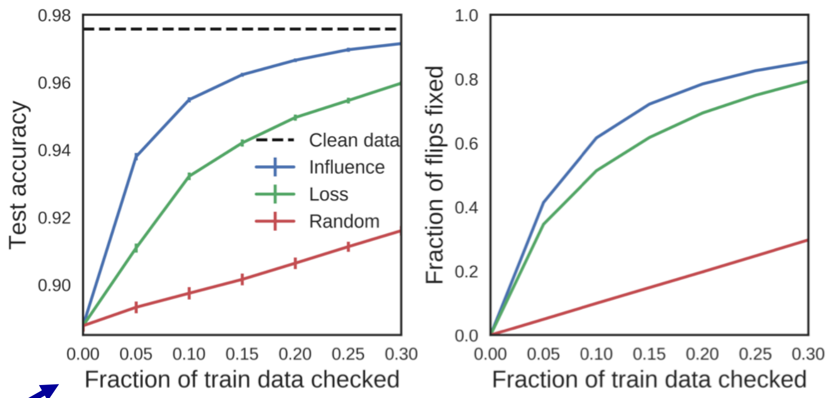


Figure 6. **Fixing mislabeled examples.** Plots of how test accuracy (left) and the fraction of flipped data detected (right) change with the fraction of train data checked, using different algorithms for picking points to check. Error bars show the std. dev. across 40 repeats of this experiment, with a different subset of labels flipped in each; error bars on the right are too small to be seen. These results are on the Enron1 spam dataset (Metsis et al., 2006), with 4,147 training and 1,035 test examples; we trained logistic regression on a bag-of-words representation of the emails.

Koh & Liang, "Understanding black-box predictions via influence functions" (2017)

Related: leverage scores

- In linear regression, a measure of training sample influence
  - Training data $X \in \mathbb{R}^{n \times d}$ and $Y \in \mathbb{R}^n$
  - Optimal parameters for $\hat{y} = \beta^\top x$ are given by:

$$\beta^* = (X^\top X)^{-1} X^\top Y$$

- Then, predictions are given by:

$$\hat{Y} = X\beta^* = X(X^\top X)^{-1} X^\top Y = HY$$

- Entries of “hat matrix” H have the following interpretation:

$$H_{ij} = \frac{\partial \hat{y}_i}{\partial y_j}$$

- Entries $H_{ii} \in [0, 1]$ are known as *leverage scores* (self-influence)
  - Can be used to identify outliers

Related: Cook's distance

- In linear regression, measures the impact of removing individual samples
- For the ith sample, Cook's distance is defined as:

$$D_i = \frac{\sum_j \left(\hat{y}_j - \hat{y}_{j(i)}\right)^2}{ds^2}$$

  - $\hat{y}_{j(i)}$ is prediction for x_j when excluding (x_i, y_i) from data
  - $s^2 = \frac{\|Y - \hat{Y}\|^2}{n-d}$ is normalized mean squared error (using all data)

- Perhaps surprisingly, Cook's distance can be calculated using leverage scores:

$$D_i = \frac{(y_i - \hat{y}_i)^2}{ds^2}\left[\frac{H_{ii}}{(1 - H_{ii})^2}\right]$$

Remarks

- **Pros:**
  - Leave-one-out is simple, easy to explain
  - Relatively efficient
  - Approximations available for cases that aren’t efficient
- **Cons:**
  - Removing individual samples may not properly quantify influence

Today

- Section 1
  - Counterfactual explanations
- Section 2
  - Leave-one-out
  - Data Shapley
  - Monitoring training dynamics

Main idea

- Removing individual samples is not informative
  - Instead, consider all possible subsets of training data and calculate **Shapley values**
  - Shapley value = impact from adding a sample, averaged across all training data permutations
- Here, we focus on model accuracy as a function of training data
  - Application to *data valuation*: compensating users for their data, or pricing for data markets

Ghorbani & Zou, "Data Shapley: Equitable valuation of data for machine learning" (2019)

Notation

- Consider a model f_θ, often a neural network
  - Given training data $D = \{(x_i, y_i)\}_{i=1}^{n}$, the model is trained by minimizing loss:

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

- Instead, we can train on subset of data $S \subseteq D$
- Let $V(S)$ represent model's performance score (e.g., accuracy) after training on S
- We want Shapley values $\phi_i(V)$ for all training samples $i = 1, \ldots, n$

Approximate retraining

- Data Shapley uses an approximate approach for retraining deep neural networks
  - Freeze all but last layer's parameters
  - Apply PCA to reduce dimensionality of final representation
  - Retraining reduces to low-dimensional logistic regression (much faster)
- Alternatively, take a single gradient step per training example
  - A worse approximation, and order matters (not ideal)

Shapley value computation

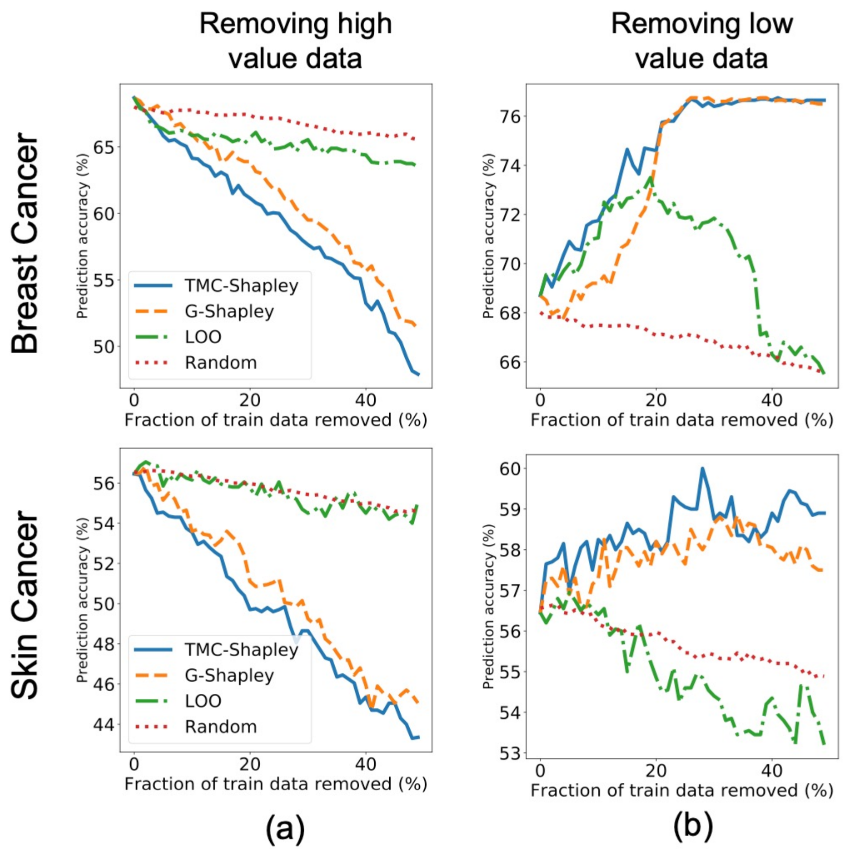
- **Problem:** Shapley values require checking all possible training sets (or all possible data orderings)
  - Intractable even for small datasets (100 samples)

- Data Shapley uses a permutation estimator
  - Recall Lecture 3, HW1
  - Small modifications for further acceleration (early truncation)
  - Correct given enough sampled permutations
    - In practice, keep sampling until roughly converged

Results

(Similar to insertion & deletion metrics, see Lecture 5)

- Removing high-value data hurts performance
- Removing low-value data helps performance
- Gradient approximation (G-Shapley) is only slightly worse



Ghorbani & Zou, "Data Shapley: Equitable valuation of data for machine learning" (2019)

Results

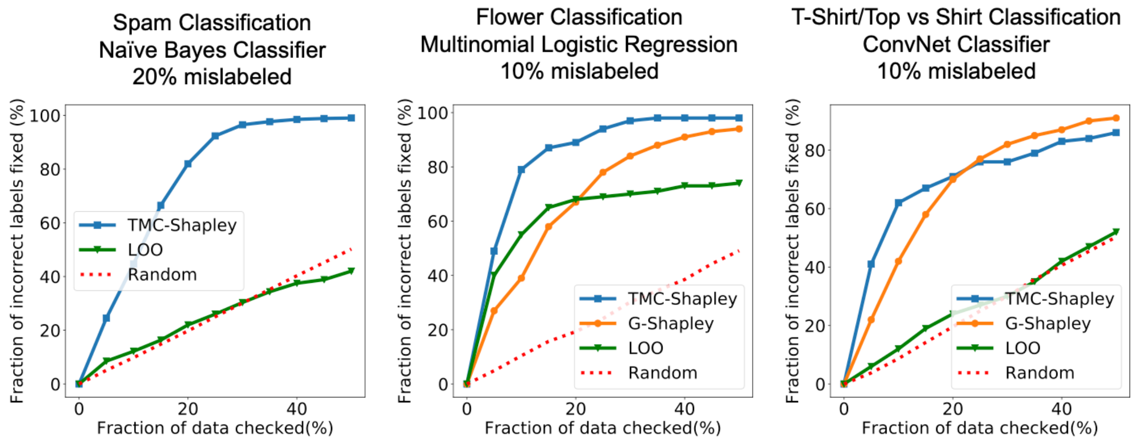


Figure 3. **Correcting Flipped Labels** We inspect train data points from the least valuable to the most valuable and fix the mislabeled examples. As it is shown, Shapley value methods result in the earliest detection of mislabeled examples. While leave-one-out works reasonably well on the Logistic Regression model, it's performance on the two other models is similar to random inspection.

Ghorbani & Zou, "Data Shapley: Equitable valuation of data for machine learning" (2019)

Remarks

- **Pros:**
  - Removing groups of samples gives better understanding of data importance
  - Data Shapley performs better than leave-one-out on dataset refinement metrics
- **Cons:**
  - Much more computationally costly
    - Challenging for large datasets (>10k examples)
    - Faster approximations exist for some special cases (not discussed here)

Further reading

- Ghorbani et al., "A distributional framework for data valuation" (2020)
- Jia et al., "Towards efficient data valuation based on the Shapley value" (2019)
- Kwon et al., "Efficient computation and analysis of distributional Shapley values" (2021)
- Kwon & Zou, "Beta Shapley: A unified and noise-reduced data valuation framework for machine learning" (2022)
- Tang et al., "Data valuation for medical imaging using Shapley value and application to large-scale chest X-ray dataset" (2021)

Today

- Section 1
  - Counterfactual explanations
- Section 2
  - Leave-one-out
  - Data Shapley
  - Monitoring training dynamics

Main idea

- Retraining a model with different training data is slow
  - Instead, monitor a model's performance over the course of single training run
    - Only for models trained via SGD (DNNs, which are challenging for other approaches)
    - Aim to understand training sample influence
    - Ideally adds minimal time to model training
- Can be approached in many different ways
  - We'll discuss just a couple examples

(Un)forgettable examples

- While training DNN via SGD, check a sample's loss every time it appears in a minibatch
  - An example is said to be *forgotten* if it was once correctly classified, and later becomes incorrect
  - *Forgettable examples*: those that are forgotten at least once
  - *Unforgettable examples*: examples that are never forgotten after being learned

Toneva et al., "An empirical study of example forgetting during deep neural network learning" (2019)

Results

Unforgettable examples are simpler, more prototypical

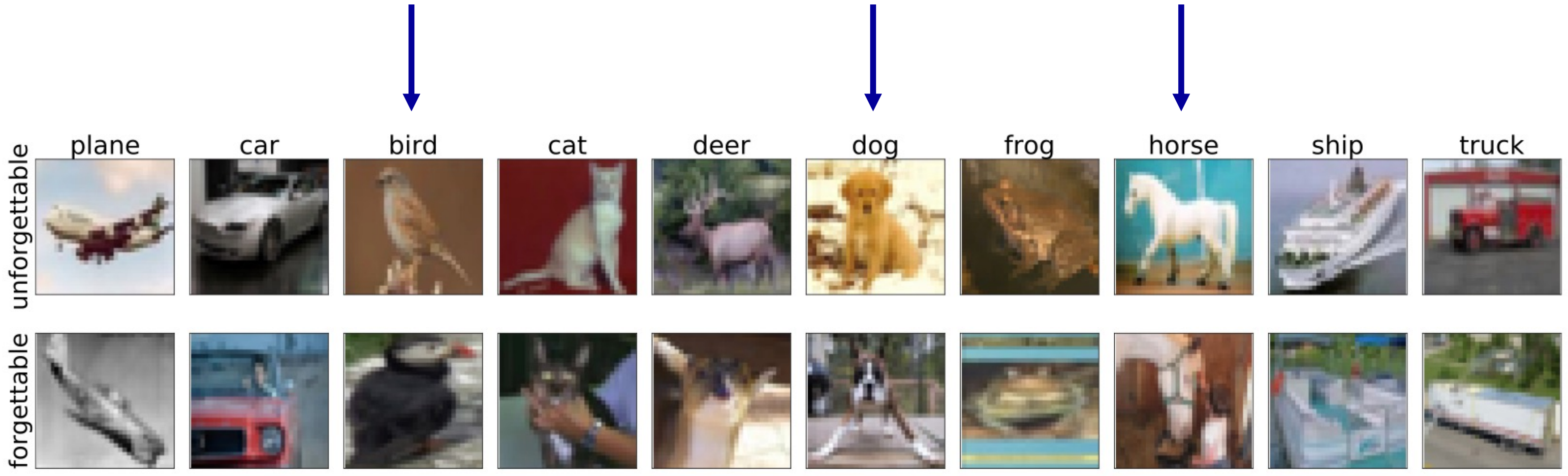
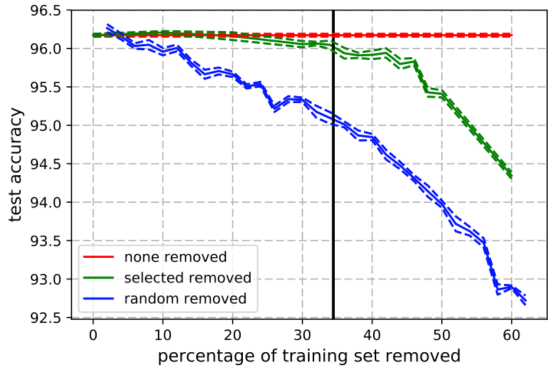


Figure 2: Pictures of unforgettable (*Top*) and forgettable examples (*Bottom*) of every *CIFAR-10* class. Forgettable examples seem to exhibit peculiar or uncommon features. Additional examples are available in Supplemental Figure 15.

Toneva et al., "An empirical study of example forgetting during deep neural network learning" (2019)

Results (cont.)

Can remove simpler, unforgettable examples without hurting performance



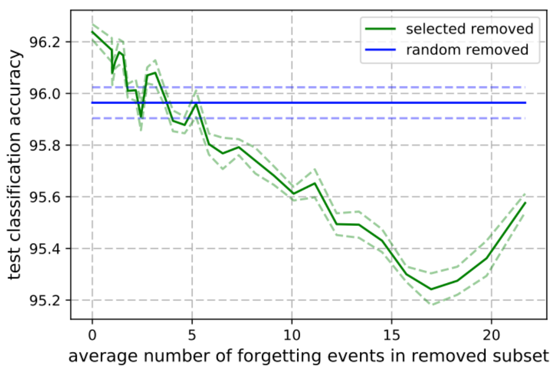


Figure 5: *Left* Generalization performance on *CIFAR-10* of ResNet18 where increasingly larger subsets of the training set are removed (mean +/- std error of 5 seeds). When the removed examples are selected at random, performance drops very fast. Selecting the examples according to our ordering can reduce the training set significantly without affecting generalization. The vertical line indicates the point at which all unforgettable examples are removed from the training set. *Right* Difference in generalization performance when contiguous chunks of 5000 increasingly forgotten examples are removed from the training set. Most important examples tend to be those that are forgotten the most.

Toneva et al., "An empirical study of example forgetting during deep neural network learning" (2019)

Ranking classification margin

- **Idea:** mislabeled samples are more likely to be incorrectly classified
  - Rather than checking at the end of training, check throughout training
  - At each epoch t, calculate margin for sample (x, y) as follows:

$$M^{(t)}(x, y) = z_y^{(t)}(x) - \max_{i \neq y} z_i^{(t)}(x)$$

True logit minus max of other logits

- Then, average across T epochs for $\mathrm{AUM}(x, y)$

Pleiss et al., "Identifying mislabeled data using the area under the margin ranking" (2020)

Results

Use scores for mislabeled samples to determine which real ones to delete

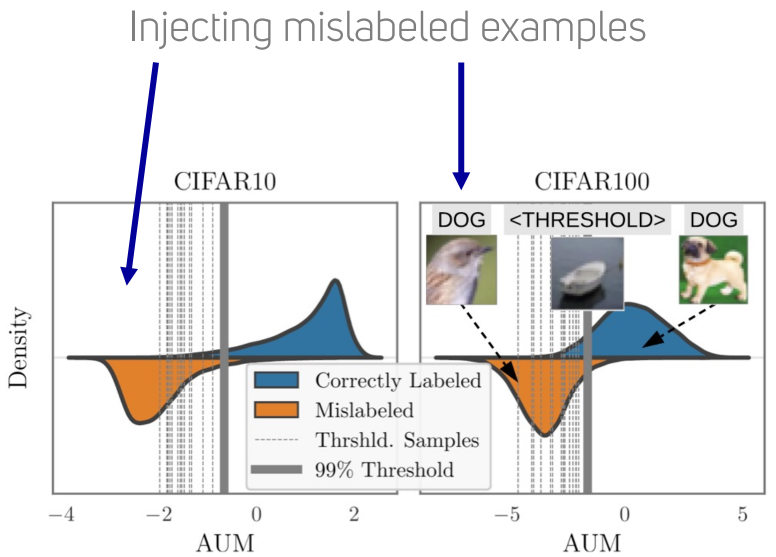


Figure 3: Illustrating the role of *threshold samples* on CIFAR10/100 with 40% mislabeled samples. Histograms of AUMs for correctly-labeled (blue) and mislabeled samples (orange). Dashed lines represent the AUM values of threshold samples. The 99th percentile of threshold AUMs (solid gray line) separates correctly- and mislabeled data.

Pleiss et al., "Identifying mislabeled data using the area under the margin ranking" (2020)

Results (cont.)

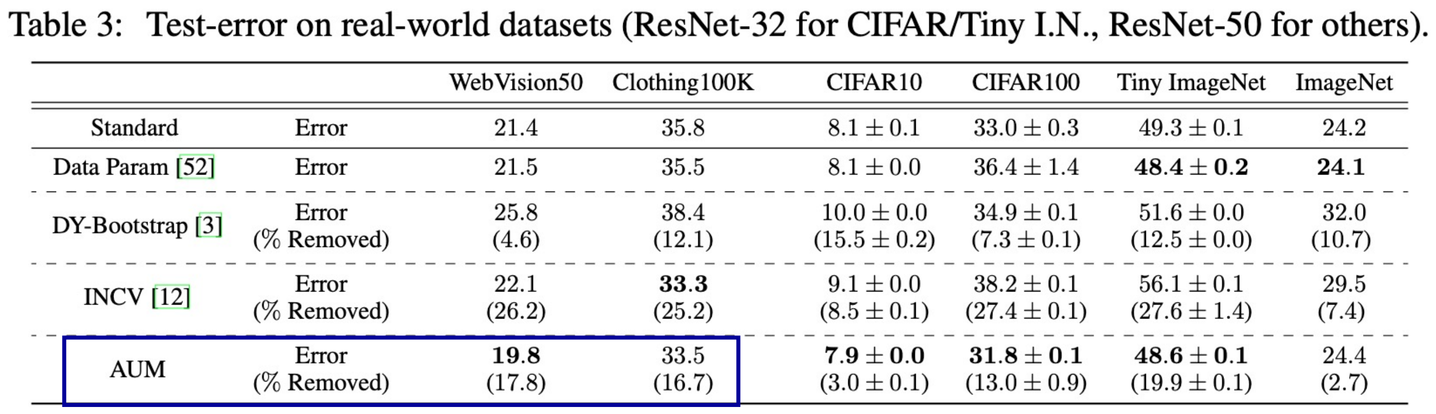


Table 3: Test-error on real-world datasets (ResNet-32 for CIFAR/Tiny I.N., ResNet-50 for others).

		WebVision50	Clothing100K	CIFAR10	CIFAR100	Tiny ImageNet	ImageNet
Standard	Error	21.4	35.8	8.1 ± 0.1	33.0 ± 0.3	49.3 ± 0.1	24.2
Data Param [52]	Error	21.5	35.5	8.1 ± 0.0	36.4 ± 1.4	**48.4 ± 0.2**	**24.1**
DY-Bootstrap [3]	Error (% Removed)	25.8 (4.6)	38.4 (12.1)	10.0 ± 0.0 (15.5 ± 0.2)	34.9 ± 0.1 (7.3 ± 0.1)	51.6 ± 0.0 (12.5 ± 0.0)	32.0 (10.7)
INCV [12]	Error (% Removed)	22.1 (26.2)	**33.3** (25.2)	9.1 ± 0.0 (8.5 ± 0.1)	38.2 ± 0.1 (27.4 ± 0.1)	56.1 ± 0.1 (27.6 ± 1.4)	29.5 (7.4)
AUM	Error (% Removed)	**19.8** (17.8)	33.5 (16.7)	**7.9 ± 0.0** (3.0 ± 0.1)	**31.8 ± 0.1** (13.0 ± 0.9)	**48.6 ± 0.1** (19.9 ± 0.1)	24.4 (2.7)

Deleting data from weakly labeled datasets (crowdsourced labels) improves performance

Pleiss et al., “Identifying mislabeled data using the area under the margin ranking” (2020)

TracIn

- Considers sample-to-sample influence when training model f_θ
  - Taking gradient step on sample $z = (x, y)$ influences loss on another sample z', denoted $\ell(\theta, z')$
  - Consider training steps t where $z_t = z$
  - Define idealized version as:

$$\mathrm{TracInIdeal}(z, z') = \sum_{t: z_t = z} \ell\left(\theta^{(t+1)}, z'\right) - \ell\left(\theta^{(t)}, z'\right)$$

Garima et al., "Estimating training data influence by tracing gradient descent" (2020)

TracIn (cont.)

- Idealized version is impractical
  - Can't check loss after every gradient step
  - We typically train using minibatches, not single examples

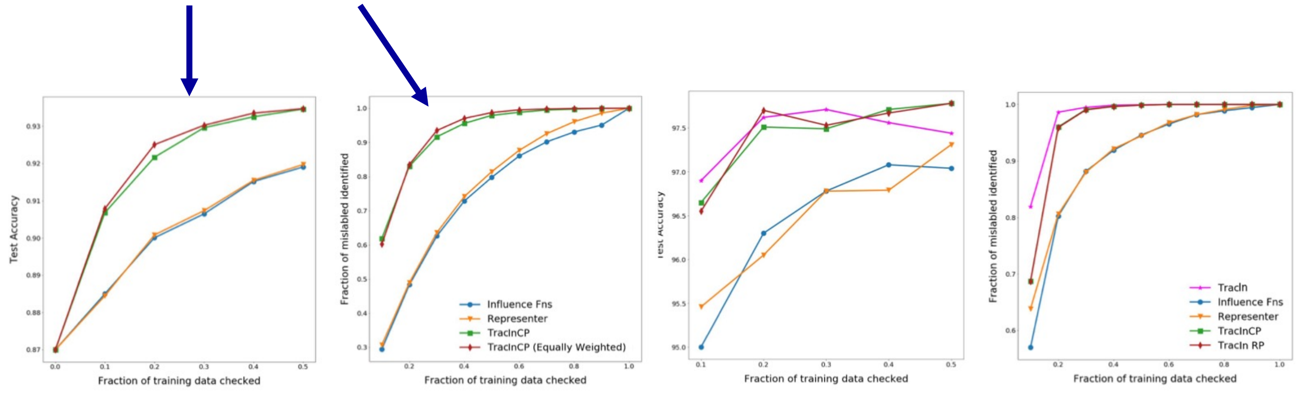
- The authors approximate $\mathrm{TracInIdeal}(z, z')$ using model checkpoints
  - Model versions are saved after multiple gradient steps
  - Derivation/equation is complicated (see Garima et al.)
  - Practical version called $\mathrm{TracInCP}(z, z')$

Garima et al., "Estimating training data influence by tracing gradient descent" (2020)

Results

Introduced mislabeled examples, then tried to identify via self-influence (mislabeled examples should be strong proponents for themselves)

Mislabeled examples are highly ranked



(a) CIFAR-10 (b) MNIST

Figure 1: CIFAR-10 and MNIST Mislabelled Data Identification with `TracIn` Representer points, and Influence Functions. We use "Fraction of mislabelled identified" on the y axis to compare the effectiveness of each method. (RP = Random Projections, CP = CheckPoints)

Garima et al., "Estimating training data influence by tracing gradient descent" (2020)

Results (cont.)

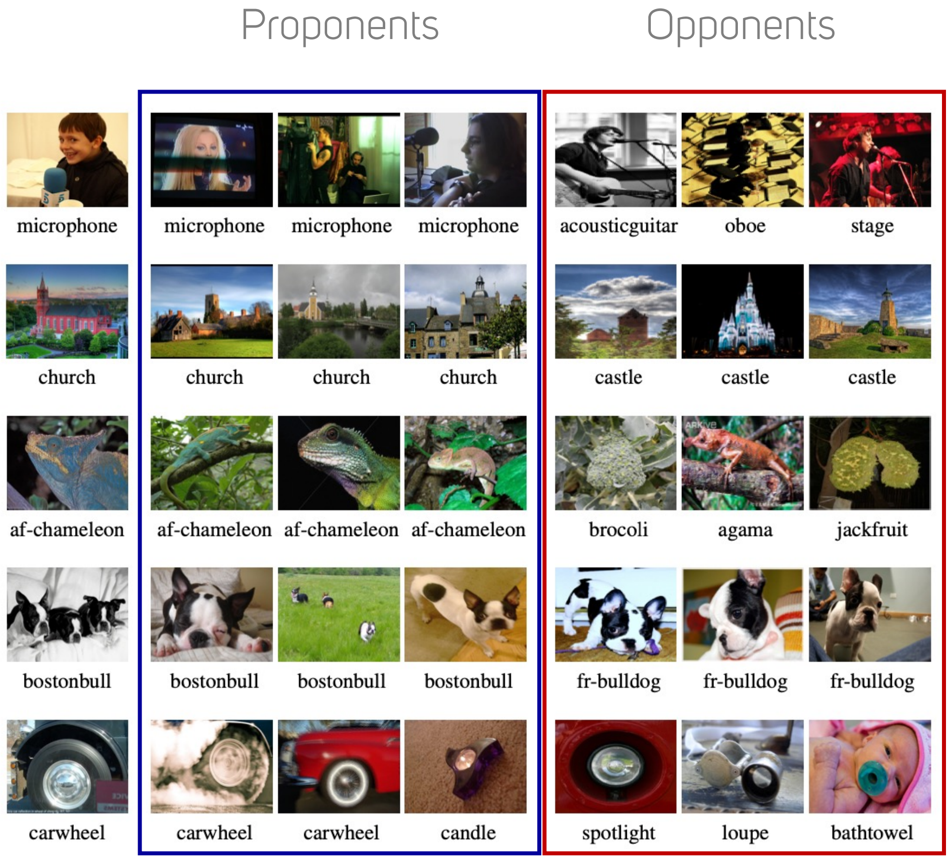


Figure 4: `TracIn` applied on Imagenet. Each row starts with the test example followed by three proponents and three opponents. The test image in the first row is classfied as band-aid and is the only misclassified example. (af-chameleon: african-chameleon, fr-bulldog: french-bulldog)

Garima et al., "Estimating training data influence by tracing gradient descent" (2020)

Remarks

- **Pros:**
  - Monitoring training dynamics has low computational cost compared to previous methods
- **Cons:**
  - Many different approaches for analyzing training dynamics, less clear which offer best performance
  - Results are stochastic, depend on training run and hyperparameters

Summary

- Diverse methods for understanding training data influence
- As with feature importance, instance importance can be used in a local or global manner
  - Focus on individual prediction, or dataset accuracy
- Computation is often challenging, but recent methods suggest interesting approximations