Adapting and Evaluating Influence-Estimation Methods for Gradient-Boosted Decision Trees

Jonathan Brophy   JBROPHY@CS.UOREGON.EDU
Zayd Hammoudeh   ZAYD@CS.UOREGON.EDU
Daniel Lowd   LOWD@CS.UOREGON.EDU
Department of Computer and Information Science
University of Oregon
Eugene, OR 97403, USA

Editor: Eric Laber

Abstract

Influence estimation analyzes how changes to the training data can lead to different model predictions; this analysis can help us better understand these predictions, the models making those predictions, and the data sets they're trained on. However, most influence-estimation techniques are designed for deep learning models with continuous parameters. Gradient-boosted decision trees (GBDTs) are a powerful and widely-used class of models; however, these models are black boxes with opaque decision-making processes. In the pursuit of better understanding GBDT predictions and generally improving these models, we adapt recent and popular influence-estimation methods designed for deep learning models to GBDTs. Specifically, we adapt representer-point methods and TracIn, denoting our new methods TREX and BoostIn, respectively; source code is available at `https://github.com/jjbrophy47/tree_influence`. We compare these methods to LeafInfluence and other baselines using 5 different evaluation measures on 22 real-world data sets with 4 popular GBDT implementations. These experiments give us a comprehensive overview of how different approaches to influence estimation work in GBDT models. We find BoostIn is an efficient influence-estimation method for GBDTs that performs equally well or better than existing work while being four orders of magnitude faster. Our evaluation also suggests the gold-standard approach of leave-one-out (LOO) retraining consistently identifies the single-most influential training example but performs poorly at finding the most influential set of training examples for a given target prediction.

Keywords: training data influence, gradient-boosted decision trees, instance attribution, influence functions, TracIn

1. Introduction

Influence estimation analyzes how changes to the training data can lead to different model predictions and helps investigate questions such as, "how would this prediction change if I were to *remove* these training examples?" Analyzing the influence of training data on model predictions can provide a better understanding of model behavior (Adadi and Berrada, 2018; Ferreira and Monteiro, 2020; Tjoa and Guan, 2020), improve model quality via data set or model debugging (Yeh et al., 2018), help quantify the value of different training examples (Ghorbani and Zou, 2019), and much more (Koh and Liang, 2017).

For example, Braun et al. (2022) leverage training data influence analysis to gain actionable insights into real-world black box models – specifically, to mitigate the effect of labeling noise on automated medical image diagnosis. Mislabeled training instances are often memorized by expressive models, resulting in these instances displaying abnormal influence scores (Feldman and Zhang, 2020). Braun et al. apply this insight by automatically downweighting training instances with outlier influence scores (Pruthi et al., 2020), resulting in medical imaging models that are both more accurate and more robust. By mitigating the effect of mislabeled data, Braun et al.'s improved models also require less training data, reducing the need for expensive, domain-specific annotators. Hammoudeh and Lowd's (2022b) influence estimation survey details additional applications of influence analysis, including algorithmic fairness (Black and Fredrikson, 2021), defenses against poisoning attacks (Hammoudeh and Lowd, 2022b), active learning (Liu et al., 2021), and data augmentation (Oh et al., 2021).

Existing work in this burgeoning subfield has focused mainly on deep learning systems (Koh and Liang, 2017; Yeh et al., 2018; Pruthi et al., 2020). However, gradient-boosted decision trees (GBDTs) are a powerful class of models that generally outperform deep learning and other more traditional learning algorithms on *tabular* data (Prokhorenkova et al., 2018; Popov et al., 2019), one of the most common types of data used in industry (Sun et al., 2019); they are also regularly used to win challenges on data-competition websites such as Kaggle and DrivenData (Fogg, 2016; Bojer and Meldgaard, 2021). Despite their predictive prowess, GBDTs are black-box models with opaque decision-making processes (Lundberg and Lee, 2017; Lundberg et al., 2018). Influence estimation may help us better understand how GBDTs make predictions, remove unwanted biases, and ultimately improve our models.

Influence functions were one of the first methods proposed for influence estimation, initially in differentiable models (Koh and Liang, 2017) and later in trees (Sharchilev et al., 2018). Representer point methods (Yeh et al., 2018) offer greater efficiency and an interpretation of the model as a sum over contributions from all training points, based on representer theorems. More recently, dynamic influence estimation methods such as TracIn (Pruthi et al., 2020) and HyDRA (Chen et al., 2021) look at the influence of training examples throughout the training process, rather than only focusing on the properties of the final model.

However, tree ensembles have not yet benefited from the advantages of new representer point and dynamic influence estimation methods. Thus, we adapt these techniques designed for deep learning models to GBDTs. Specifically, we propose *TREX* (§3.4), our adaptation of representer-point methods (Yeh, 1998), and *BoostIn* (§3.3), our adaptation of TracIn (Pruthi et al., 2020).

Since influence estimation may behave differently in trees than in deep learning models, we evaluate a wide range of techniques with varying methodologies to better understand how best to do influence estimation in GBDTs. However, empirically evaluating the merit of different influence methods tends to vary in the literature, in part because there is no single criterion for defining the optimal set of training examples that influence a prediction (Koh and Liang, 2017; Yeh et al., 2018; Hanawa et al., 2021). Thus, to get a broad overview of performance, we focus on quantitative evaluations rather than the qualitative analysis that is sometimes used to evaluate influence-estimation methods. Specifically, we approximate the optimal set by ranking training examples based on their individual influences and measure changes in model predictions after performing some operation on a subset of the most

influential examples (e.g., removal) and retraining the model; this evaluation methodology provides a quantitative measure of the fidelity of each method, that is, how well a method predicts actual model behavior. We use different evaluation measures since the effectiveness of an influence method may be operation dependent. For example, the training instances identified as most influential may have the biggest impact on a target prediction when *removed*, but not when *their labels are flipped* instead; hence, we evaluate each influence method in various contexts (§4).

We conduct extensive experiments on 22 real-world data sets to compare eight different influence-estimation methods using 5 different evaluation measures and 4 popular modern GBDT implementations : LightGBM (Ke et al., 2017), XGBoost (Chen and Guestrin, 2016), CatBoost (Prokhorenkova et al., 2018), and gradient boosting from Scikit-Learn (Pedregosa et al., 2011). Our results suggest the adaptation of TracIn (Pruthi et al., 2020)—which we denote *BoostIn*—is a solid default choice for influence estimation in GBDTs. BoostIn is over *four orders of magnitude* faster than the current state-of-the-art: LeafInfluence (Sharchilev et al., 2018)—the adaptation of Influence Functions (Koh and Liang, 2017) to GBDTs— and provides better influence estimates than competing methods in the majority of tested contexts, on average (§5).

Furthermore, leave-one-out (LOO)—removing training examples one at a time, retraining the model after each removal, and measuring the prediction difference between the original and retrained models—consistently identifies the *single*-most influential training example for a given target prediction by definition; however, we find LOO does a poor job of selecting the most influential *set* of training examples that contribute to a given target prediction (§5.4). We find LOO often induces small but significant changes to the structure of the trees as a result of removing one or very few training examples; we also observe LOO has a very low correlation with any of the other influence-estimation methods (§5.3). Thus, when considering more training examples for an influence estimate, we find methods using a fixed tree-structure assumption are better able to find a *set* of training examples that most influence a given target prediction than methods that do not.

In the following sections, we detail the problem of influence estimation both generally and in the context of GBDTs (§2). We review previous work adapting influence functions to GBDTs, and then present our new methods adapting more recent approaches (§3). We evaluate all methods in multiple contexts, including removing and relabeling the most influential training examples and observing the effect on one and many target predictions after retraining the model (§4, §5). We then describe additional related work (§6), and finally conclude and discuss avenues for future work (§7).

2. Preliminaries and Problem Formulation

For *feature space* $\mathcal{X} \subseteq \mathbb{R}^p$ and *target space* $\mathcal{Y} \in \{\{-1, +1\}, \mathbb{Z}, \mathbb{R}\}$, let $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$. Let $D := \{(x_i, y_i)\}_{i=1}^n \subseteq \mathcal{Z}$ be the full *training set*, where $x_i \in \mathcal{X}$ is a p-dimensional vector $(x_{i,j})_{j=1}^p$ and $y_i \in \mathcal{Y}$. We use $\mathbf{x} := \{x_i\}_{i=1}^n$ and $\mathbf{y} := \{y_i\}_{i=1}^n$ to denote the full training *feature matrix* and *target vector*, respectively. $z_i := (x_i, y_i) \in D$ is the ith training instance and $z_e := (x_e, y_e) \in \mathcal{Z}$ an arbitrary target instance.

Let $A : 2^D \to \mathcal{F}$ be a *learning algorithm*, where 2^D is the *power set* of D and $\mathcal{F}$ is the *hypothesis class*. $f := A(D)$ is the *model* A yields when training on data set D. A *loss function* is denoted $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

We use notation "$\sim$" to denote a deep learning variable, to distinguish it from those corresponding to trees; for example, $\tilde{f}$ denotes a deep learning model; $\tilde{\ell}$ is a deep learning loss function, etc.

2.1 Gradient-Boosted Decision Trees (GBDTs)

Gradient boosting (Friedman et al., 2000; Friedman, 2001) is a powerful machine-learning algorithm that iteratively adds weak learners to construct model $f : \mathcal{X} \to \mathbb{R}$ that minimizes some empirical risk (i.e., $f = \arg\min_{\hat{f}} \sum_{i=1}^{n} \ell(y_i, \hat{f}(x_i))$). The model is defined recursively where:

$$f_0(x) = \gamma \tag{1}$$

$$\vdots$$

$$f_t(x) = f_{t-1}(x) + m_t(x), \tag{2}$$

in which γ is an initial estimate (typically the mean of the training targets: $\bar{\mathbf{y}} := \frac{1}{n} \sum_i y_i$), f_{t-1} is the model at the previous iteration, m_t is a weak learner added during iteration t to improve the model. Gradient-boosted decision trees (GBDTs) use regression trees as the weak learners, with each tree typically chosen to approximate the negative gradient (Malinin et al., 2021):

$$g_t(x, y) = \left. \frac{\partial \ell(y, \hat{y})}{\partial \hat{y}} \right|_{\hat{y}=f_{t-1}(x)}, \tag{3}$$

$$\therefore m_t = \arg\min_{\hat{m}} \frac{1}{n} \sum_{i=1}^{n} (-g_t(x_i, y_i) - \hat{m}(x_i))^2. \tag{4}$$

The weak learner at iteration t partitions the instance space into a set of M_t disjoint regions $\{r_{t,1}, \ldots, r_{t,M_t}\}$. Each region $r_{t,l}$ is called a *leaf* whose parameter value $\theta_{t,l}$ is typically determined (given a fixed structure) using a one-step Newton-estimation method (Chen and Guestrin, 2016; Ke et al., 2017):

$$h_t(x, y) = \left. \frac{\partial^2 \ell(y, \hat{y})}{\partial \hat{y}^2} \right|_{\hat{y}=f_{t-1}(x)}, \tag{5}$$

$$\therefore \theta_{t,l} = -\left(\frac{\sum_{i \in I_{t,l}} g_t(x_i, y_i)}{\sum_{i \in I_{t,l}} h_t(x_i, y_i) + \lambda} \right) \eta \tag{6}$$

in which $I_{t,l} = \{z_j \mid z_j \in r_{t,l}\}_{j=1}^{n}$ is the instance set of the lth leaf for the tth tree, $g_t(x_i, y_i)$ and $h_t(x_i, y_i)$ are the first and second derivatives of the ith training instance with respect to $\hat{y}_i$, respectively (Eqs. 3 and 5), λ is a regularization constant, and η is the learning rate. Thus, m_t can be written as

$$m_t(x) = \sum_{l=1}^{M_t} \theta_{t,l} \mathbb{1}[x \in r_{t,l}] \tag{7}$$

4

where $\mathbb{1}$ is the indicator function. The final GBDT model $f = f_T$ generates a prediction for a target example x_e by summing the initial estimate with the values for the leaves x_e is assigned to across all T trees: $f(x_e) = f_0 + \sum_{t=1}^{T} m_t(x_e)$.

2.2 Influence Estimation

Existing influence estimation methods attempt to compute the influence of each training example z_i on the prediction of a given target example z_e. Informally, a training example is *influential* if its inclusion in the training data impacts the learned model and its predictions. Leave-one-out (LOO) defines influence as the difference between training with the entire training data set and training with a specified example excluded. The Shapley value is similar to LOO, but computes an expectation of LOO over all (exponentially many) subsets of the original training data. In general, the impact of removing an example depends on which other examples are removed. Existing methods compute a single number for each individual training example in order to generate a *ranking* amongst the training data. Standard influence estimation metrics then evaluate this ordering by analyzing sequential subsets of the most influential examples (Hammoudeh and Lowd, 2022a).

Following previous work (Koh and Liang, 2017; Pruthi et al., 2020), we analyze the influence of training examples on a given target example by computing their impact on the *loss* of that target example.[1] To this end, we define an *influence-estimation method* $\mathcal{I} : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}$ as a function that quantifies the effect of training instance z_i on target example z_e given learning algorithm A, training set D, and loss function ℓ. Note that *influence value* $\mathcal{I}(z_i, z_e)$ can be positive or negative, denoting that z_i *reduces* or *increases* the model's loss on target z_e, respectively.[2]

LOO: Leave-One-Out. The simplest and most intuitive approach to estimating the influence of a training example z_i on a target example z_e is to ignore the existing model and simply rerun A on a pruned data set without z_i, and then measure the change in loss[3] on z_e:

$$\mathcal{I}_{LOO}(z_i, z_e) = \ell(y_e, A(D \setminus \{z_i\})(x_e)) - \ell(y_e, A(D)(x_e)). \tag{8}$$

Repeating this naive approach for all training examples is known as *leave-one-out* (LOO) (Cook and Weisberg, 1982). LOO is agnostic to virtually all machine learning models, easy to understand, easy to implement, and is regularly described as a gold-standard influence-estimation method (Ghorbani and Zou, 2019). Empirically, we find this approach performs relatively poorly for identifying a practical *set* of influential examples for a given target prediction in GBDTs (§5.4); furthermore, this approach becomes prohibitively expensive as the data set size or model complexity increases.

Data Shapley: Expected Marginal Influence. A different way of determining the contributions of individual examples belonging to a group is via Shapley values, a game-theoretic method for distributing contributions among involved players (Shapley, 1953). Data Shapley (Ghorbani and Zou, 2019) is a model-agnostic approach that applies the idea of Shapley

---

1. A *target example* is simply the example we compute influence values for, either in the train or test set.
2. Following convention (Pruthi et al., 2020), we refer to training examples that reduce the target example's loss as *proponents* and those that increase loss as *opponents*.
3. When A is non-deterministic, one would need to retrain *multiple* times on the same data set to compute the *expected* change in loss.

values to influence estimation in which the marginal contribution of z_i on the loss of z_e can be written as:

$$\mathcal{I}_{DShap}(z_i, z_e) = C \sum_{S \subseteq D \backslash \{z_i\}} \frac{\ell(y_e, A(S)(x_e)) - \ell(y_e, A(S \cup \{z_i\})(x_e))}{\binom{n-1}{|S|}}, \tag{9}$$

where C is a constant and S represents all possible subsets of the training data without z_i. Equation (9) computes the *expected* marginal impact of a single example given a subset of the training data, but is also far more intractable than LOO.

SubSample: Simplified Data Shapley. More recently, Feldman and Zhang (2020) propose a method that quantifies the amount of memorization acquired by a deep learning model during training. Their approach – *SubSample* – computes the memorization level of a training example as well as the influence of a training example on the accuracy of a learning algorithm A by training τ different models on random subsets of D. SubSample combines the predictions of these τ models to estimate each training instance's marginal influence. Thus, SubSample can be viewed as a simplified version of Data Shapley that considers a single pre-specified subset size instead of all possible subset sizes. The expected marginal-influence effect of z_i on the loss of z_e is then defined as:

$$\mathcal{I}_{Sub}(z_i, z_e) = \mathbb{E}_{S \sim U(D \backslash \{z_i\}, m-1)}[\ell(y_e, A(S \cup \{z_i\})(x_e))]$$
$$- \mathbb{E}_{S \sim U(D \backslash \{z_i\}, m)}[\ell(y_e, A(S)(x_e))]. \tag{10}$$

In Equation (10), $U(D, m)$ represents the uniform distribution over all subsets of D with size $m := |S| < n$. For this approach to be tractable and produce meaningful influence values, m must be small enough to provide sufficient cases in which $z_i \notin S$, but large enough to train reasonable approximations to the original model. SubSample is much more efficient than Data Shapley (Eq. 9) and in most cases LOO (Eq. 8), especially when $\tau \ll n$.

The three influence estimators above work for *any* model. A number of other influence-estimation methods have been developed in the context of deep learning and other differentiable, parametric models (Koh and Liang, 2017; Pruthi et al., 2020; Yeh et al., 2018). These parametric approaches cannot be applied directly to discrete, tree-structured, non-parametric models like GBDTs. In the next section, we introduce these deep learning-based influence estimators, and describe the adaptation of each method to estimate influence in GBDTs.

3. Adapting Influence Methods to GBDTs

This section first reviews *LeafRefit* (§3.1) and *LeafInfluence* (§3.2), work by Sharchilev et al. (2018) adapting influence functions (Koh and Liang, 2017) to GBDTs. Next, we introduce *BoostIn* (§3.3), our adaptation of TracIn (Pruthi et al., 2020), and theoretically compare BoostIn to LeafInfluence. Lastly, we describe our adaptation of representer-point methods (Yeh et al., 2018) to GDBTs—a method we denote *TREX* (§3.4). For a more detailed discussion and comparison of existing parametric influence analysis methods, see the survey of Hammoudeh and Lowd (2022a).

Before describing the influence estimation methods specific to GBDTs, we define an important assumption made by all the tree-based methods in this section.

Simplifying Assumption 1 (Fixed Structure Approximation) $\forall S \subseteq D$, *GBDT models* $A(D)$ *and* $A(D \setminus S)$ *have the same structure.*

Assumption 1 allows the influence of z_i to be estimated while treating the structure of each tree as fixed, where feature splits are considered part of the structure of each tree, precluding the influence of z_i on any node split. Assumption 1 thus enables the methods detailed in this section to efficiently estimate influence values for larger datasets. Without this assumption, estimating the effect of removing z_i from a GBDT model would require considering all the possible ways in which the structure of each tree in the ensemble could change due to the removal of z_i, and how those changing structures would impact the loss on the target example z_e.

3.1 LeafRefit: LOO with Fixed Tree Structures

To estimate the influence of a training example on a target example *without* having to retrain from scratch, Sharchilev et al. (2018) develop *LeafRefit*, a method that computes the influence of z_i on the loss of z_e in GBDTs by *refitting* all leaf values without z_i. LeafRefit is a theoretically efficient alternative to LOO that assumes a fixed structure while refitting leaves and computing influence values (Assumption 1).

However, despite the theoretical advantage of LeafRefit avoiding recomputing node splits or retraining completely from scratch, LeafRefit still needs to recompute leaf values, which can be expensive.

3.2 LeafInfluence: Adapting Influence Functions

Influence functions (Jaeckel, 1972; Hampel, 1974) is a technique from robust statistics that analyzes how the continuous parameters $\tilde{\theta}$ of a differentiable model change when a training example z_i is upweighted by a small amount w_i, resulting in updated parameters $\tilde{\theta}_{w_i} := \arg\min_{\tilde{\theta} \in \tilde{\Theta}} \frac{1}{n} \sum_{j=1}^{n} \tilde{\ell}(z_j, \tilde{\theta}) + w_i \tilde{\ell}(z_i, \tilde{\theta})$ where $\tilde{\ell}(z_i, \tilde{\theta})$ is the loss of z_i when using parameters $\tilde{\theta}$. The influence of w_i on the model parameters can be approximated *without* having to retrain from scratch:

$$\frac{d\tilde{\theta}_{w_i}}{dw_i}\bigg|_{w_i=0} = -H_{\tilde{\theta}}^{-1} \nabla_{\tilde{\theta}} \tilde{\ell}(z_i, \tilde{\theta}), \tag{11}$$

where $H_{\tilde{\theta}} = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\tilde{\theta}}^2 \tilde{\ell}(z_i, \tilde{\theta})$ (Cook and Weisberg, 1982). Koh and Liang (2017) use this result to develop an influence estimator for deep learning models by analyzing the changing parameters due to upweighting z_i, and then analyzing the effect the changing model parameters have on the loss of a target example z_e:

$$
\begin{aligned}
\mathcal{I}_{IF}(z_i, z_e) &= \nabla_{\tilde{\theta}} \tilde{\ell}(z_e, \tilde{\theta})^\intercal \frac{d\tilde{\theta}_{w_i}}{dw_i}\bigg|_{w_i=0} \\
&= -\nabla_{\tilde{\theta}} \tilde{\ell}(z_e, \tilde{\theta})^\intercal H_{\tilde{\theta}}^{-1} \nabla_{\tilde{\theta}} \tilde{\ell}(z_i, \tilde{\theta}).
\end{aligned}
\tag{12}
$$

Koh and Liang's (2017) method (henceforth referred to simply as "influence functions") provides an efficient approximation to LOO for shallow deep learning models with strictly

convex and twice-differentiable loss functions; for larger models or non-convex losses, however, the approximation is typically poor (Basu et al., 2020; Bae et al., 2022; Hammoudeh and Lowd, 2022b).

For GBDTs, Sharchilev et al. (2018) develop *LeafInfluence*, an adaptation of influence functions and an approximation of LeafRefit that analyzes the *perturbation* of leaf values and the resulting effect on the loss of z_e, where:

$$
\begin{aligned}
\mathcal{I}_{LI}(z_i, z_e) &= \frac{\partial \ell(y_e, \hat{y}_e)}{\partial w_i}\Big|_{\hat{y}_e = f(x_e)} \\
&= \frac{\partial \ell(y_e, \hat{y}_e)}{\partial \hat{y}_e}\Big|_{\hat{y}_e = f(x_e)} \cdot \frac{\partial f(x_e)}{\partial w_i}.
\end{aligned} \tag{13}
$$

Equation (13) approximates the change in z_e's loss as a result of upweighting z_i.[4] The effect of upweighting z_i on the *final* GBDT model (second term in Eq. 13) is defined as:

$$
\frac{\partial f(x_e)}{\partial w_i} = \sum_{t=1}^{T} \frac{\partial \theta_{t,l_e}(f_{t-1}(\mathbf{x}))}{\partial w_i}, \tag{14}
$$

in which l_e is the index of the leaf assigned to x_e at iteration t, θ_{t,l_e} is the corresponding leaf value, and $f_{t-1}(\mathbf{x})$ are the intermediate predictions at iteration $t-1$. The partial derivative of the l_eth leaf at iteration t with respect to the ith training instance is defined as:

$$
\frac{\partial \theta_{t,l_e}(f_{t-1}(\mathbf{x}))}{\partial w_i} = -\frac{\mathbb{1}[i \in I_{t,l_e}](h_{t,i}\theta_{t,l_e} + g_{t,i}) + \sum_{j \in I_{t,l_e}}(k_{t,j}\theta_{t,l_e} + h_{t,j})J(f_{t-1}(\mathbf{x}))_{i,j}}{\sum_{j \in I_{t,l_e}} h_{t,j}}, \tag{15}
$$

in which $g_{t,i} = g_t(x_i, y_i)$ (Eq. 3), $h_{t,i} = h_t(x_i, y_i)$ (Eq. 5), $k_{t,i} = \partial^3 \ell(y_i, \hat{y}_i)/\partial \hat{y}_i^3|_{\hat{y}_i = f_{t-1}(x_i)}$, and $J(f_{t-1}(\mathbf{x}))_{i,j} = J(f_{t-2}(\mathbf{x}))_{i,j} + \partial \theta_{t-1, R_{t-1}(x_j)}(f_{t-2}(\mathbf{x}))/\partial w_i$ is a Jacobian matrix that accumulates the changing intermediate predictions of all training examples as a result of upweighting z_i. Thus, the estimated influence of training example z_i on the loss of z_e can be found by running x_e through a new ensemble with new leaf values defined by Eq. (15) and multiplying the result by the derivative of the loss with respect to the original prediction (Eq. 13).

Approximating the influence of a *single* training example via LeafInfluence (Eq. 13) may be more efficient than LeafRefit, but the computation is an expensive operation in general, mainly due to the *cascade effect* of changing predictions. For example, upweighting z_i changes the second-iteration predictions of all examples in the same leaf as z_i, which then changes the leaf values for all leaves containing those examples in the second iteration and the predictions of the examples in those leaves for the third iteration; this process repeats for subsequent iterations, potentially necessitating the tracking and updating of all intermediate training-example predictions throughout the ensemble. The runtime complexity of LeafInfluence for computing $\mathcal{I}_{LI}(z_i, z_e)$ is $O(Tn^2)$, and computing influence values for *all* training examples is $O(Tn^3)$. Empirically, we find LeafInfluence to be only marginally faster than LeafRefit, and surprisingly even slower than simply retraining from scratch (§5.2).

---

4. In our experiments, we negate Eq. (13) to be consistent with the concept of proponents and opponents, defined in §2.2.

LeafInfluence-SinglePoint: Mitigating the Cascade Effect. By restricting LeafInfluence to analyze *only* the intermediate predictions of z_i and the parameters (leaf values) containing z_i, Sharchilev et al. (2018) introduce LeafInfluence-SinglePoint (which we henceforth call LeafInfSP), a rough approximation to the main proposed approach (Eq. 13):

$$\mathcal{I}_{LI_{SP}}(z_i, z_e) = \frac{\partial \ell(y_e, \hat{y}_e)}{\partial \hat{y}_e}\Big|_{\hat{y}_e = f_T(x_e)} \cdot \left(\sum_{t=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_e)] \left(\frac{\partial \theta_{t, R_t(x_i)}(f_{t-1}(x_i))}{\partial w_i} \right) \right) \tag{16}$$

in which

$$\frac{\partial \theta_{t, l=R_t(x_i)}(f_{t-1}(x_i))}{\partial w_i} = -\frac{(g_{t,i} + h_{t,i}\theta_{t,l}) + (h_{t,i} + k_{t,i}\theta_{t,l})J(f_{t-1}(x_i))}{\sum_{j \in I_{t,l}} w_j h_{t,j}}, \tag{17}$$

$R_t : \mathcal{X} \rightarrow \{1, \ldots, M_t\}$ is a function that maps an instance x to the lth leaf of the tth tree, and $J(f_{t-1}(x_i)) = J(f_{t-2}(x_i)) + \partial\theta_{t-1, R_{t-1}(x_i)}(f_{t-2}(x_i))/\partial w_i$ analyzes the changing intermediate predictions of *only* z_i throughout the ensemble, mitigating the problem of approximating the change in parameter values for leaves which do not contain z_i. Both LeafInfSP and LeafInfluence approximate the total change in model parameters and estimate the effect of this change on the final model prediction. In the next section, however, we introduce BoostIn, a method that estimates the influence of z_i on z_e *throughout* the training process. We then compare BoostIn to LeafInfSP and LeafInfluence, highlighting the similarities and differences.

3.3 BoostIn: Adapting TracIn

TracIn (Pruthi et al., 2020) is an influence-estimation method designed for deep learning models that analyzes the impact of z_i on z_e *throughout* the training process. TracIn is based on the fundamental theorem of calculus that decomposes the difference between a function at two points using the gradients along the path between those two points.

The idealized version of TracIn assumes every model update uses one training example z^t at each iteration t and thus defines the influence of z_i as the total reduction in loss on z_e whenever z_i is used to update the model. More formally, the idealized version of TracIn can be defined as:

$$\mathcal{I}_{TracInIdeal}(z_i, z_e) = \sum_{t:z^t=z_i} \tilde{\ell}(z_e, \tilde{\theta}_t) - \tilde{\ell}(z_e, \tilde{\theta}_{t+1}). \tag{18}$$

This notion of influence has the appealing property that the sum of influences for all training examples on z_e is *exactly* the total reduction in loss during training:

$$\sum_{i=1}^{n} \mathcal{I}_{TracInIdeal}(z_i, z_e) = \tilde{\ell}(z_e, \tilde{\theta}_0) - \tilde{\ell}(z_e, \tilde{\theta}_T). \tag{19}$$

However, deep learning models are almost never trained in this fashion, and are typically trained using a batch or minibatches. Furthermore, the target example would need to be known ahead of training, or the entire training process would need to be repeated, which is generally intractable. Thus, a reasonable heuristic approximation is to save the model at

various *checkpoints* throughout training in which it is assumed each training example has been processed exactly once between checkpoints; the influence of z_i any target example z_e's loss can be estimated via a sum of first-order approximations:

$$\mathcal{I}_{TracInCP}(z_i, z_e) = \sum_{t \in \mathcal{T}} \eta_t \nabla \tilde{\ell}(z_i, \tilde{\theta}_t) \cdot \tilde{\ell}(z_e, \tilde{\theta}_t), \tag{20}$$

where $\mathcal{T} \subseteq \{1, \dots, T\}$ are the iteration numbers corresponding to the $|\mathcal{T}|$ checkpoints and η_t is the learning rate at iteration t.

BoostIn is our adaptation of TracIn to GBDTs. While TracIn computes influence by summing over gradient updates, the analog in a GBDT is to sum over trees, where each tree represents a functional gradient update (Eq. 4). BoostIn first considers all intermediate GBDT models constructed during training as checkpoints $(f_0, f_1, \dots, f_T)$. Recall the difference between any two intermediate models f_t and f_{t-1} is the weak learner m_t multiplied by η_t (the learning rate at iteration t), and that each training example is visited exactly once between checkpoints since m_t computes the gradients of *all* training examples using the predictions from the previous iteration (Eq. 4). In contrast to TracIn, in which a checkpoint typically consists of updates from minibatches that are partitions of the randomly ordered training data, each checkpoint during GBDT training uses all the data at once; thus, training order for GBDTs is irrelevant.

BoostIn processes each intermediate model using Assumption 1 to analyze the effect of training example z_i on the leaf it belongs to at each iteration. For the tth iteration, the approximate change in leaf value due to z_i corresponds to an estimated change in prediction of z_e only if z_i and z_e are in the same leaf at that iteration. The estimated change in prediction then approximates the change in loss on z_e. Finally, BoostIn aggregates these approximations across all iterations, simulating the effect of z_i on z_e throughout the training process. More formally, BoostIn uses the chain rule to analyze the marginal effect each changing leaf value has on the loss of z_e, and sums these marginal effects across checkpoints:

$$\mathcal{I}_{BoostIn}(z_i, z_e) = \sum_{t=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_e)] \left(-\frac{d\ell(y_e, \hat{y}_e)}{dw_i} \bigg|_{\hat{y}_e = f_t(x_e)} \right)$$

$$= \sum_{t=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_e)] \left(-\frac{\partial \ell(y_e, \hat{y}_e)}{\partial \hat{y}_e} \bigg|_{\hat{y}_e = f_t(x_e)} \cdot \frac{\partial f_t(x_e)}{\partial w_i} \right)$$

$$= \sum_{t=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_e)] \left(\frac{\partial \ell(y_e, \hat{y}_e)}{\partial \hat{y}_e} \bigg|_{\hat{y}_e = f_t(x_e)} \cdot \eta_t \frac{\partial \theta_{t,l}}{\partial w_i} \right) \tag{21}$$

in which $\frac{\partial f_t(x_e)}{\partial w_i} \approx -\eta_t \frac{\partial \theta_{t,l}}{\partial w_i}$, $l = R_t(x_e)$, and the partial derivative of $\theta_{t,l}$ with respect to w_i is defined as:

$$\frac{\partial \theta_{t,l}}{\partial w_i} = \frac{g_{t,i} + h_{t,i} \, \theta_{t,l}}{\sum_{j \in I_{t,l}} h_{t,j} + \lambda}. \tag{22}$$

Again, note Eq. (21) enforces a constraint that attributes nonzero influence only when z_i and z_e are in the same leaf at a given iteration. Also, when processing an intermediate model

at iteration t, BoostIn avoids the cascade effect by approximating the change in only the $R_t(x_i) = l$th leaf for weak learner m_t.

How BoostIn Relates to LeafInfluence-SinglePoint. The main idea of LeafInfSP (Eq. 16) is to *accumulate* the changes in the leaf values affected by upweighting z_i and multiply this result by the *final* prediction of the GBDT model on x_e. In contrast, BoostIn (Eq. 21) multiplies each leaf-value derivative by the corresponding *intermediate* prediction of x_e at each iteration (only relevant when z_i and z_e are in the same leaf at a given iteration t), then takes the sum over all intermediate results. This is an important distinction as BoostIn analyzes the cumulative change in loss *throughout* the training process, not just on the final model prediction.

In terms of computation, BoostIn and LeafInfSP have the same runtime complexity for computing the influence of a *single* training example $O(T)$, and computing influence values for *all* training examples $O(Tn)$. Empirically, however, the original implementation of LeafInfSP[5] is not optimized to realize this lower runtime complexity as it is implemented in conjunction with the full LeafInfluence approach; thus, as a minor contribution, we implement an optimized version of LeafInfSP and demonstrate in §5.2 that our implementation achieves similar runtime performance compared to BoostIn, as expected. Overall, BoostIn is a solid choice for influence estimation in GBDTs, providing on par or better influence estimates than LeafInfSP and LeafInfluence while being orders of magnitude more efficient than LeafInfluence.

3.4 TREX: Adapting Representer-Point Selection

Representer theorems (Schölkopf et al., 2001) state the optimal solutions of many learning problems can be represented in terms of the training examples. In particular, the non-parametric representer theorem (Schölkopf et al., 2001, Theorem 4) applies to empirical risk minimization within a reproducing kernel Hilbert space (RKHS); this theorem covers a wide range of linear and kernelized machine-learning methods.

Yeh et al. (2018) apply representer theorems to deep learning models by using the layers (except the final layer) of the network $\tilde{\theta}_1$ as a feature map $\tilde{\mathbf{f}}_i = \tilde{\phi}(x_i; \tilde{\theta}_1)$, and training a kernelized model with L2 regularization on the transformed features. Specifically, for surrogate linear model f^*, parameters ψ^* represent the regularized, empirical risk minimizer:

$$\psi^* = \arg\min_{\psi} \lambda||\psi||^2 + \frac{1}{n}\sum_{i=1}^{n}\tilde{\ell}(y_i, f^*(\tilde{\mathbf{f}}_i; \psi)). \tag{23}$$

Since ψ^* is a stationary point, the gradient of the loss is zero:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{\partial\tilde{\ell}(y_i, f^*(\tilde{\mathbf{f}}_i; \psi^*))}{\partial\psi^*} + 2\lambda\psi^* = 0, \tag{24}$$

$$\therefore \psi^* = -\frac{1}{2\lambda n}\sum_{i=1}^{n}\frac{\partial\tilde{\ell}(y_i, f^*(\tilde{\mathbf{f}}_i; \psi^*))}{\partial\psi^*} = \sum_{i=1}^{n}\alpha_i\tilde{\mathbf{f}}_i, \tag{25}$$

---

5. `https://github.com/bsharchilev/influence_boosting`

in which $\alpha_i = -\frac{1}{2\lambda n}\frac{\partial \tilde{\ell}(y_i,\hat{y}_i)}{\partial \hat{y}_i}\big|_{\hat{y}_i=f^*(\tilde{\mathbf{f}}_i;\psi^*)}$ is the global importance of z_i to the overall surrogate model. Finally, the pre-activation prediction of an arbitrary target example z_e can be decomposed as a linear combination of the training examples:

$$f^*(\tilde{\mathbf{f}}_e;\psi^*) = \sum_{i=1}^{n}\alpha_i\langle\tilde{\mathbf{f}}_i,\tilde{\mathbf{f}}_e\rangle, \tag{26}$$

in which $\langle\tilde{\mathbf{f}}_i,\tilde{\mathbf{f}}_e\rangle$ represents the similarity between z_i and z_e in the transformed feature space; $\alpha_i\langle\tilde{\mathbf{f}}_i,\tilde{\mathbf{f}}_e\rangle$ is referred to as the *representer value* for z_i given z_e, and its magnitude is largest when the magnitudes of *both* the training example weight α_i and the similarity of z_i to z_e is large. The representer value of z_i can be positive or negative, representing the attribution of z_i towards or away from the predicted value of z_e, respectively.

To adapt representer-point methods to GBDTs, we need a way of extracting the learned representation of a given GBDT model, similar to feature extraction in deep learning models. For this purpose, we use *supervised tree kernels* (Davies and Ghahramani, 2014; He et al., 2014; Bloniarz et al., 2016), a general approach for extracting the learned representation from a tree ensemble by using the structure of the trees. Tree kernels can be used to measure the similarity between two instances by analyzing how each example is processed by each tree in the ensemble (Moosmann et al., 2006). Tree kernels can outperform traditional nearest-neighbor methods at identifying the training instances most relevant to a prediction – in particular for large data dimensions.

Intuitively, two examples are predicted identically if they appear in the same leaf in every tree in the ensemble. Thus, we can define the degree of similarity between two data points by comparing the specific paths taken through each tree; we can incorporate additional flexibility into the similarity measure accounting for path overlap, node weights (number of examples at a node), and leaf values. More formally, we define tree kernels as dot products in an alternate feature representation defined by the feature mapping ϕ, i.e., $\langle\mathbf{f}_i,\mathbf{f}_e\rangle = \langle\phi(x_i;f),\phi(x_e;f)\rangle$. Note these supervised kernels are parameterized by the GBDT model f, since the computation necessarily depends on the structure of the ensemble, similar to taking the output of the penultimate layer in deep learning models. Based on work by Plumb et al. (2018) on local linear modeling, we define $\phi(x,f) = \bigcup_{t=1}^{T}(\frac{1}{n_{t,l}}\cdot\mathbb{1}[x\in r_{t,l}])_{l=1}^{M_t}$ as a sparse vector over all leaves in f; i.e., for each tree, the element at $R_t(x) = l$th leaf is nonzero and inversely weighted by $n_{t,l}$, the number of training examples assigned to the lth leaf.

Given feature mapping ϕ, we use Eq. (23) with loss function ℓ to train a surrogate model to approximate our original GBDT model, enabling the decomposition of a target prediction using Eq. (26). However, the resulting representer values only quantify the contribution of a training example z_i to the *prediction* of z_e, but we are interested in how z_i affects the loss of z_e; thus, we measure the influence of z_i on the loss of z_e by subtracting the representer value for z_i from the prediction decomposition of z_e and computing the change in loss:

$$\mathcal{I}_{TREX}(z_i,z_e) = \ell\Big(y_e,\upsilon\big(\hat{y}_e^* - \alpha_i\langle\mathbf{f}_i,\mathbf{f}_e\rangle\big)\Big) - \ell\big(y_e,\upsilon(\hat{y}_e^*)\big), \tag{27}$$

in which v is an activation function[6] and $\hat{y}_e^* = \sum_{j=1}^{n} \alpha_j \langle \mathbf{f}_j, \mathbf{f}_e \rangle$ is the surrogate model pre-activation prediction for z_e. We denote this method *TREX* (**T**ree-ensemble **R**epresenter **Ex**amples) and evaluate its influence-estimation quality in §5.

Similarity-Based Influence. As defined above, TREX measures the influence of an example z_i on a target example z_e using two pieces of information: the weight of the training example α_i, and the similarity to the target instance $\langle \mathbf{f}_i, \mathbf{f}_e \rangle$. To better understand the marginal effect of each component, we define an additional influence estimation method that skips training a surrogate model and *only* uses the tree-kernel similarity to quantify the influence of z_i on z_e:

$$\mathcal{I}_{TreeSim}(z_i, z_e) = \mathbb{1}[y_i = y_e]\langle \mathbf{f}_i, \mathbf{f}_e \rangle - \mathbb{1}[y_i \neq y_e]\langle \mathbf{f}_i, \mathbf{f}_e \rangle. \tag{28}$$

This method, *TreeSim*, attributes positive influence to examples with the *same* label as z_e, and negative otherwise;[7] this value is then scaled by the similarity between z_i and z_e.

3.5 Summary of Influence-Estimation Methods

Table 1 summarizes the influence-estimation methods discussed in this paper. In short, LOO and SubSample are model-agnostic approaches that compute the influence of training examples by removing them and retraining one or multiple models on revised data sets. The rest of the methods are model specific and assume a fixed-structure (Assumption 1) while computing influence values. LeafRefit recomputes all leaf values without a particular training instance in order to estimate the influence of that instance. LeafInfluence and LeafInfSP approximate this process by measuring the total aggregated change in model parameters and analyzing how this change affects the final target prediction. BoostIn approximates the change in leaf values and their effects on the loss of a target prediction at each intermediate model, tracing the influence of a training instance on a target prediction throughout the training process. Both BoostIn and LeafInfSP are asymptotically much more tractable than LeafRefit and LeafInfluence. TREX trains an interpretable kernel surrogate model that decomposes any prediction into a sum of the training instances, and TreeSim identifies the most influential training examples by how similar they are to the target example.

In the following sections, we evaluate the estimation quality of each method across a wide range of data sets and evaluation contexts.

4. Methodology

In our experiments, we order the training data based on the influence values generated by each method for a *single* test example (§4.2, §4.3) or a *set* of test examples (§4.4, §4.5, §4.6). We then evaluate these orderings in different contexts by removing (§4.2, §4.4), performing targeted label editing (§4.3), and adding label noise (§4.5) to the most influential examples and observing the effect on the model/predictions after *retraining*; the more a method de-

---

6. Typically a sigmoid or softmax for binary and multiclass classification tasks, respectively.
7. For regression, TreeSim treats z_i and z_e as having the same label if y_i and y_e are on the same side of the target prediction $\hat{y}$; that is, $\text{sgn}(\hat{y}_e - y_i) = \text{sgn}(\hat{y}_e - y_e)$, in which *sgn* is the signum function.

Method	Source	Changes	Assumes Fixed Struct.
LOO	-	-	
SubSample	Feldman and Zhang (2020)	-	
LeafRefit	Sharchilev et al. (2018)	-	✓
LeafInfluence	Koh and Liang (2017)	Sharchilev et al. (2018)	✓
LeafInfSP	Koh and Liang (2017)	Sharchilev et al. (2018)	✓
BoostIn	Pruthi et al. (2020)	§3.3 (Ours)	✓
TREX	Yeh et al. (2018)	§3.4 (Ours)	✓
TreeSim	Plumb et al. (2018)	§3.4 (Ours)	✓

Table 1: Summary of influence-estimation methods.

grades the resulting model predictions,[8] the higher that method is ranked. We also evaluate the effectiveness of each method at detecting noisy or mislabelled training examples (§4.6) as is often done in previous work (Ghorbani and Zou, 2019; Pruthi et al., 2020). Overall, we want influence methods with high fidelity, that is, methods that accurately predict the *behavior* of the model after some operation (e.g., removing the most influential training examples for a test example should *actually* increase the loss of the given test example after retraining without those' examples); in general, this evaluation protocol provides a quantitative measure of the effectiveness of each influence-estimation method. In the following subsections, we provide data set and method details, and then describe each experiment in detail; we present our results in §5.

4.1 Data Sets and Methods

Data sets. We evaluate on 22 real-world tabular data sets (13 binary-classification tasks, 1 multiclass-classification task, and 8 regression tasks) well-suited for boosted tree-based models. For each data set, we generate one-hot encodings for any categorical attributes and leave all binary and numeric attributes as is. For any data set without a predefined train/test split, we sample 80% of the data uniformly at random for training and use the rest for testing. All additional data set details are in the Appendix, §A.1.

Models. We train GBDT models using the most popular and modern implementations: LightGBM (Ke et al., 2017, LGB), XGBoost (Chen and Guestrin, 2016, XGB), Scikit-Learn (Pedregosa et al., 2011, SGB), and CatBoost (Prokhorenkova et al., 2018, CB). Each model is tuned using 5-fold cross-validation; selected hyperparameters are in §A.2: Table 4, with predictive performance comparisons in §A.2: Table 3.

Influence Methods and Baselines. We include the following influence-estimation methods in our evaluation: LOO (Eq. 8), SubSample (Eq. 10; we set $\tau = 4,000$ and $m = \lfloor 0.7n \rfloor$ as recommended by Feldman and Zhang (2020)), LeafRefit, LeafInfluence (Eq. 13), BoostIn (Eq. 21), LeafInfSP (Eq. 16), TREX (Eq. 27), and TreeSim (Eq. 28). We also include *Random* as an additional baseline, which assigns influence values via a standard

---

8. We measure changes in model predictions via logistic loss for classification tasks and squared error for regression tasks.

normal distribution: $\mathcal{I}_{Random}(z_i, z_e) \sim \mathcal{N}(0, 1)$. Due to the limited scalability of LeafRefit and LeafInfluence (see §5.2), we evaluate these methods on a subset of the data sets consisting of 13 smaller data sets and present results on this group, which we denote *small data subset (SDS)* (exactly which data sets are part of SDS is given in §A.1: Table 2); however, we observe the same trends when including all data sets in our analysis, which are in §A.3.

Implementation. Code containing all influence-estimation implementations and experiments is available at `https://github.com/jjbrophy47/tree_influence`; our implementations also include an optimized version of LeafInfSP. Supplemental implementation details as well as hardware details used for the experiments are in §A.

4.2 Single Test Instance: Removing Influential Training Examples

Inspired by the remove and retrain (ROAR) framework for measuring the impact of different features (Hooker et al., 2019), we evaluate the influence of training examples on a given test example by measuring the change in loss on the test example using a model retrained after removing the most influential training examples. In theory, the training examples with the most positive influence values decrease the loss of the test example the most; thus, removing them and retraining the model should increase the loss of the test example.

For this experiment, we generate influence values for a given test example z_e, and order the training instances from most positive (i.e., instances that decrease the loss of z_e the most) to most negative. Using this ordering, we remove 0.1%, 0.5%, 1%, 1.5%, and 2% of the training data, retraining a separate GBDT model on each modified version of the data set. We then measure the change in loss on z_e using the original and retrained models. We repeat this experiment for 100 randomly chosen test examples and compute the average increase in test loss per example. We then rank the influence-estimation methods by how much they increase the test loss on average; then, we average these rankings over removal percentages (0.1%, 0.5%, etc.), GBDT types, and data sets.

4.3 Single Test Instance: Targeted Training-Label Edits

In this section, instead of removing examples, we ask the counterfactual (Byrne, 2019; Keane and Smyth, 2020), "how would this prediction change if I were to edit the *label(s)* of the most influential training examples?" However, since most of the influence-estimation methods simulate the removal of a training example, we slightly adapt the estimators to simulate a changing training label for this experiment. We modify LOO to compute the influence of z_i on z_e via training-label edit by changing z_i to z_i^* (z_i with a new label y_i^*) and retraining the model. We similarly modify LeafRefit (LOO with a fixed structure) to refit leaf values with z_i^* instead z_i. Note these operations are equivalent to *removing* z_i and *adding* z_i^*. However, LeafInfluence, LeafInfSP, and BoostIn only *approximate* the removal of z_i, but these influence methods are able to estimate the influence of an instance that does not actually exist in the training data (see influence definitions in §3). Thus, we can approximate the influence of a changing training label by simulating the removal of z_i and the addition of z_i^*; that is, $\mathcal{I}(z_i \rightarrow z_i^*, z_e) \approx \mathcal{I}(z_i, z_e) - \mathcal{I}(z_i^*, z_e)$.

In this experiment, we use the same setup as §4.2, ordering the training examples from most positive to most negative. Using this ordering, we change 0.1%, 0.5%, 1%, 1.5%, and

2% of the training labels to the *same* chosen target label y^*.[9] We retrain the model after each batch modification, and measure the change in loss on the test example; we then rank each method by how much the loss increases, and average these results over modification percentages, GBDT types, and data sets.

4.4 Multiple Test Instances: Removing Influential Training Examples

We now analyze the effect of influential examples on a *set* of test instances. We sample 10% of the test examples uniformly at random to use as a validation set and generate influence values for each example. We then aggregate the influence values via a sum over the validation examples to get a single influence value per training example, and then rank the training examples from most positive (decreases the loss of the validation examples the most on aggregate) to most negative. Using this ordering, we remove examples in batches of 5%, removing up to a maximum 50% of the training data. We retrain a separate GBDT model after each batch removal and measure the change in loss to the original model on a held-out test set (that is, the test examples not used for the validation set). We then rank each influence-estimation method by how well it degrades the resulting model at each level of removal (5%, 10%, etc.), and average these rankings over removal percentages, GBDT types, and data sets.

4.5 Multiple Test Instances: Adding Training-Label Noise

For this experiment, we use the same setup as §4.4, including the orderings produced by each influence-estimation method. Then, instead of removing the most positively-influential examples, we add noise to them by randomly changing each of their labels: we flip labels for binary-classification tasks, sample new labels uniformly at random for multiclass-classification tasks, and sample new target values between the minimum and maximum values of $\mathbf{y}$ uniformly at random for regression tasks. This procedure can be viewed as an availability data poisoning attack (Steinhardt et al., 2017; Levine and Feizi, 2020; Hammoudeh and Lowd, 2023). We then retrain, remeasure, and rank the influence methods in the same way as §4.4.

4.6 Multiple Test Instances: Fixing Mislabelled Training Examples

Another way of evaluating the influence of training examples is via detecting and fixing noisy or mislabelled training instances. Intuitively, very *negatively-influential* training examples may signify outliers, mislabelled/noisy examples, etc., and possibly warrant manual inspection. We conduct this experiment in a similar fashion to those in the literature (Koh and Liang, 2017; Yeh et al., 2018; Ghorbani and Zou, 2019; Pruthi et al., 2020; Pleiss et al., 2020), sampling 40% of the training data uniformly at random and flipping their labels in the same way as §4.5. We then generate influence values in the same way as §4.4 and §4.5, but then rank the training examples from most negative (that is, examples that increase the loss of the validation examples the most on aggregate) to most positive. Using this

---

9. For binary-classification tasks, we choose y^* to be opposite $\hat{y}_e$ (the predicted label of z_e). For multiclass classification, y^* is randomly sampled from $\mathcal{Y} \setminus \{\hat{y}_e\}$. For regression, $y^* = \bar{\mathbf{y}} - (\bar{\mathbf{y}}/2)$ if $\hat{y}_e > \bar{\mathbf{y}}$, otherwise $y^* = \bar{\mathbf{y}} + (\bar{\mathbf{y}}/2)$.

ordering, we manually inspect 5%, 10%, 15%, 20%, 25%, and 30% of the training data, fixing the training examples whose labels had been flipped. We rank each method by how many mislabelled examples it detects at each level of manual inspection (5%, 10%, etc.); we then average these rankings over all inspection levels, GBDT types, and data sets. We also add a simple but effective baseline called *Loss* that orders the training examples to be checked based on their loss (high-loss training examples are checked first).

5. Results and Analyses

Figure 1 shows a high-level overview of our results across evaluation settings; we partition the influence techniques into "fast" and "slow" methods (see §5.2 for a runtime comparison) to give readers a sense of how much performance can be increased (if any) given more computational effort. Overall, BoostIn tends to perform best or equally best, except for the "targeted edit" experiment, in which LeafRefit and methods based on tree-kernel similarity such as TREX and TreeSim are more effective. Surprisingly, LOO performs consistently poorly; thus, we investigate this method further in §5.4.

5.1 Summary of Results

We present an overview of the results for each experiment in this section, with additional analyses in the Appendix, §A.4-§A.8.

Single Test Instance: Removing Influential Training Examples. Figure 1a (left) shows LeafRefit ranking highest, with BoostIn and LeafInfluence performing roughly the same. SubSample also performs well, although we observe a noticeable drop in its relative performance when adding larger data sets into our analysis (§A.3); increasing τ for larger data sets may improve performance, but may also significantly increase the running time. In terms of magnitude, all methods outperform random removal, with BoostIn having a slight advantage over all other methods (Figure 1b: left); surprisingly, LOO does only marginally better than random.

Single Test Instance: Targeted Training-Label Edits. Figure 1a (middle-left) shows LeafRefit ranking higher than all other methods; LeafInfluence, TREX, and TreeSim also rank highly. Although BoostIn is not ranked as highly as these methods, it is significantly more effective than LeafInfSP and SubSample, and its relative magnitude in terms of loss increase is similar to that of LeafInfluence and TreeSim (Figure 1b: middle-left).

Multiple Test Instances: Removing Influential Training Examples. Figure 1a (middle) shows BoostIn and LeafInfSP ranking significantly higher than all other methods; however, BoostIn tends to choose examples that increase the loss more than LeafInfSP, on average (Figure 1b-middle shows a 2.2x and 1.7x increase in loss relative to Random for BoostIn and LeafInfSP, respectively). Additional analyses for other predictive performance metrics such as accuracy are in §A.6.

Multiple Test Instances: Adding Training-Label Noise. Our results show BoostIn clearly outperforms all other methods in terms of rank (Figure 1a: middle-right) and relative loss increase (Figure 1b: middle-right), and suggest BoostIn may be an effective tool for providing untargeted data set poisoning attacks. Somewhat surprisingly, LeafInfSP does not perform as well on this task as compared to removing examples. Additional analyses for other predictive performance metrics are in §A.7.
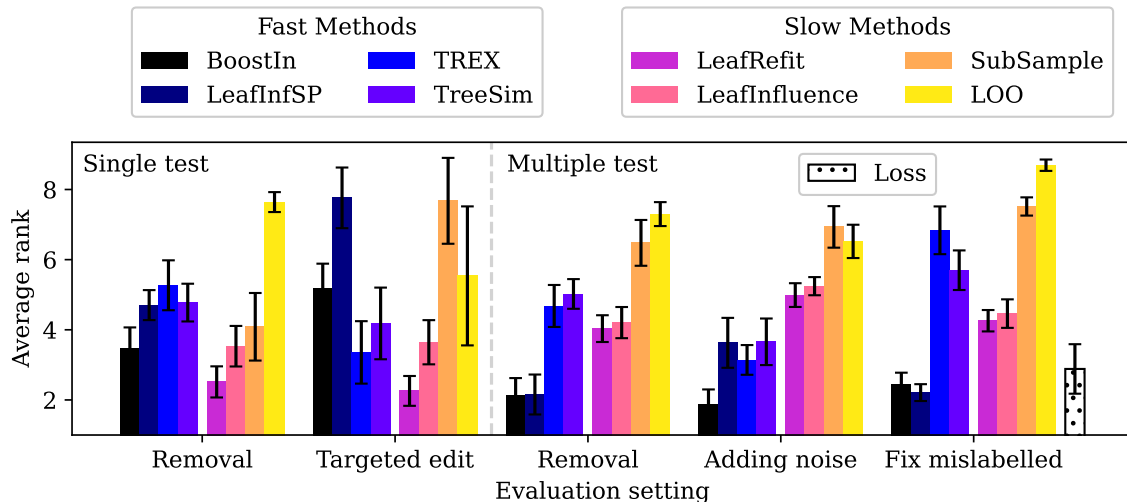
(a) Average rank of each method. For each evaluation setting, results are averaged over all checkpoints, tree types, and data sets; error bars represent 95% confidence intervals and are computed over data sets. Lower is better. We exclude Random since it performed consistently worse than all other methods in each setting.
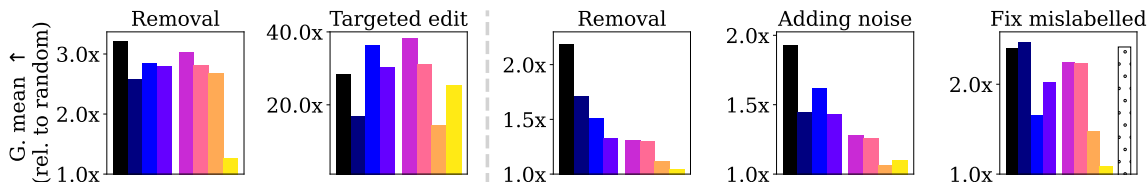
(b) Average *loss* increase (except for "fix mislabelled", which shows average increase in *mislabelled detection*) relative to random. For each evaluation setting, results are averaged over all checkpoints and tree types, then the geometric mean is computed over all data sets. Higher is better.

Figure 1: High-level overview of results showing (a) average rank and (b) relative impact of each method. Methods are grouped based on their relative efficiency (Figure 2); for both subfigures, evaluation settings left of the gray-dashed line represent experiments that compute influence values for a *single* test instance and measure the predictive impact on that instance (this is then repeated and averaged over 100 randomly-chosen test instances), while experiments right of the dashed line compute aggregate influence values for a *set* of test examples and measure the predictive impact on a held-out test set.

Multiple Test Instances: Fixing Mislabelled Training Examples. Figure 1a (right) shows BoostIn and LeafInfSP performing best, slightly outranking Loss; however, all three methods perform comparably in terms of relative magnitude, on average (Figure 1b: right). Predictive-performance improvements on the held-out test set after retraining the model on partially fixed versions of the data sets are in §A.8.

5.2 Runtime Comparison

To better understand the relative efficiency of each method, we measure the time it takes to compute all influences values for a single random test example for each dataset.[10] We repeat each experiment 5 times, and average the results over GBDT types.
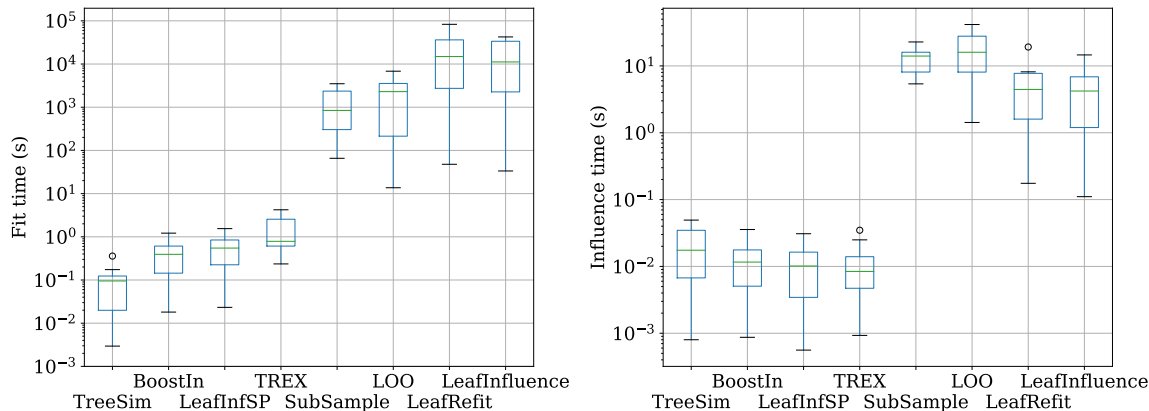

Figure 2: *Left*: Average setup time for each explainer. *Right*: Average time to compute influence values of all training examples for one test example. Results are averaged over 5 folds and GBDT types; each box plot represents average running times across all SDS datasets. TreeSim, BoostIn, LeafInfSP, and TREX represent "fast" methods with low setup and influence times, and are separated from the remaining "slow" methods by orders of magnitude efficiency.

Figure 2 shows the runtime of each approach broken down into two components: "fit time" and "influence time". Fit time is the time to initialize and set up the explainer, and influence time is the time to compute the influence of all training examples for one test example; note the log scale. TreeSim has the fastest setup time overall; however, TreeSim, BoostIn, LeafInfSP, and TREX all have low initialization and influence times compared to SubSample, LOO, LeafRefit, and LeafInfluence. We semantically group the former and latter methods into "fast" and "slow" groups, separated by orders of magnitude efficiency.

All methods in the "slow" group must train or approximate (and store) a separate model for each training example during setup, which becomes unwieldy as n increases (SubSample is the exception that trains a fixed $\tau = 4000$ models; however, this is still a significant number of models to train and store). When computing influence values, the "slow" methods predict using all models obtained during setup. In contrast, methods in the "fast" group are able to compute influence values using only one model instead of n or τ models.

Surprisingly, LeafInfluence and LeafRefit take significantly more time to set up than than LOO and have similar influence times, on average. Additionally, the relative efficiency of SubSample is very similar to LOO; this is mainly due to our choice of τ and the small data set sizes in the SDS. For larger data sets (assuming τ remains fixed) or smaller choices of τ, one can expect SubSample to be more efficient than LOO in general.

---

10. For a clearer comparison, no parallelization is used for any method.

5.3 Correlation Between Influence Methods

To better understand the relationships between different influence methods, we compute the Spearman rank (Zar, 2005) and Pearson correlation coefficients between every pair of influence methods using the values generated for each test example in §4.2. We then average these correlations over all test examples, GBDT types, and data sets.
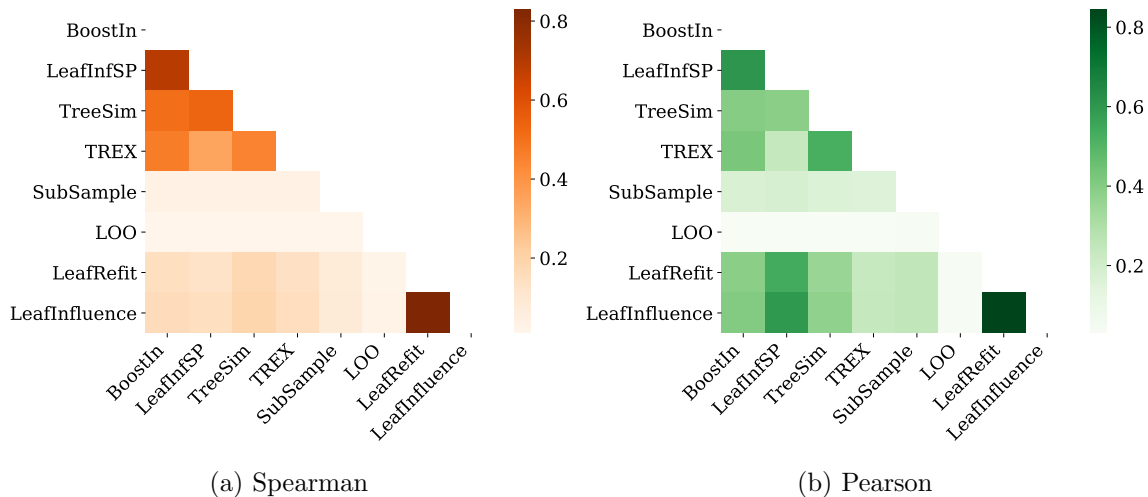

(a) Spearman

(b) Pearson

Figure 3: Average Spearman and Pearson correlation coefficients between every pair of influence methods; results are based on the rankings generated via the influence values for each test example, averaged over 100 test examples and then over tree types and data sets.

Figure 3 shows a high correlation between LeafRefit and LeafInfluence, which is expected since LeafInfluence is an approximation of LeafRefit. BoostIn is highly correlated with LeafInfSP, which we theoretically analyzed as relatively similar in §3.3. We also observe a cluster of similar methods: BoostIn, LeafInfSP, TreeSim, and TREX. Surprisingly, LOO is not highly correlated with any other method; the low correlation and relatively poor performance (Figure 1) prompts us to investigate LOO further in §5.4. This result also provides evidence that the previous state-of-the-art, LeafInfluence, is a poor approximation of LOO. Additional analysis regarding the correlation between influence methods is in §A.10.

5.4 The Structural Fragility of LOO

Throughout our experiments, LOO performs consistently worse than many of the other methods, often only doing marginally better than random. These results thus warrant further investigation.

To assess the performance of LOO in more depth, we use the same setup as §4.2, except we remove examples in increments of *one* instead of specified percentages (0.1%, 0.5%, etc.); we do the same for Random, BoostIn, and LeafRefit for additional context. Figure 4 shows the results, and we immediately notice a spike in test loss (outlined by a gray box) after the *first* removal for LOO higher than any of the other methods, followed by a drop

and general plateau of the test loss for subsequent removals. We consistently see this spike across data sets and GBDT types (additional examples are in §A.11). This shows that LOO's top-ranked example is indeed the most influential, but using LOO's ordering to pick a *group* of examples to remove is much less effective. In fact, on average, removing the top two examples (as ranked by LOO) has *less* impact on test loss than just removing one! In other words, *the fixed ordering produced by LOO means that highly-ranked examples can counteract the effects of other highly-ranked examples when removed.*

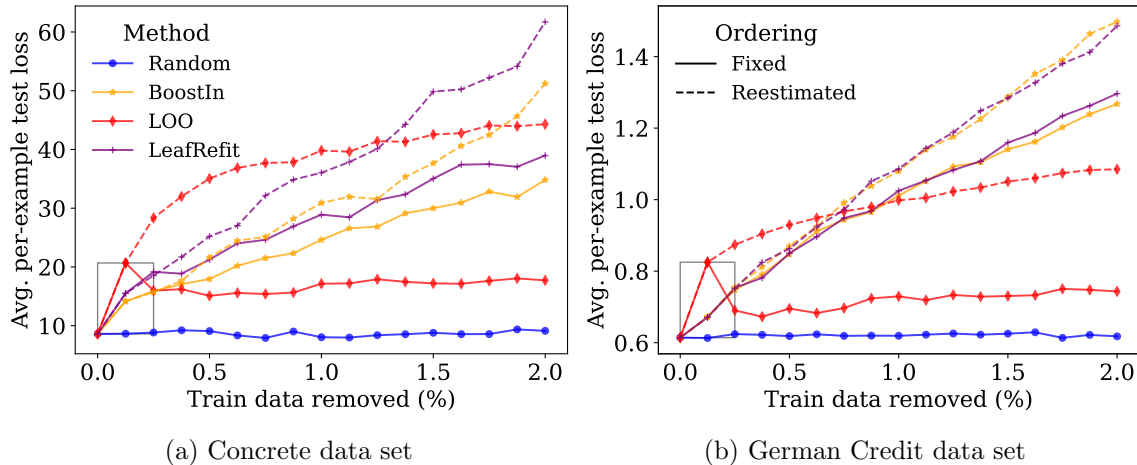(a) Concrete data set   (b) German Credit data set

Figure 4: Change in test-example loss (averaged over 100 test examples using LGB) after removing the most positively-influential training examples *one at a time* using a fixed ordering as well as a dynamic ordering that *reestimates* influence values for the remaining training data after each removal. The gray box highlights the large increase in test loss by LOO after removing only a single example. Additional examples are in the Appendix, §A.11.

To quantify how much this fixed ordering makes a difference, we *reestimate* influence values on the remaining training data after each deletion, dynamically reordering the training examples to be removed (Figure 4). We make two observations from this variation; first, we notice a significant improvement in the performance of not only LOO, but all methods (except for random). Second, even with improved performance, LOO tends to plateau after a certain point, being surpassed by both BoostIn and LeafRefit.

The contrast between LOO and LeafRefit is particularly illuminating, because as detailed in §3.1, the *only* difference between LOO and LeafRefit is that LeafRefit makes a fixed-structure assumption (i.e., z_i is "deleted" from the leaves where it appears, but the tree structure is left intact). Since LOO measures each single deletion's effect exactly, LOO outperforms all other methods in the *single deletion case* (shown by the "spike" in test loss). However, as the size of the deletion group increases beyond one, LOO *greedily* selects subsequent training examples to add to the deletion group. LOO's poor results provides evidence that considering structural changes while making greedy selections is counterproductive, often leading to suboptimal deletion groups, especially for larger group sizes. In contrast, LeafRefit ignores structural changes, instead selecting instances based on their
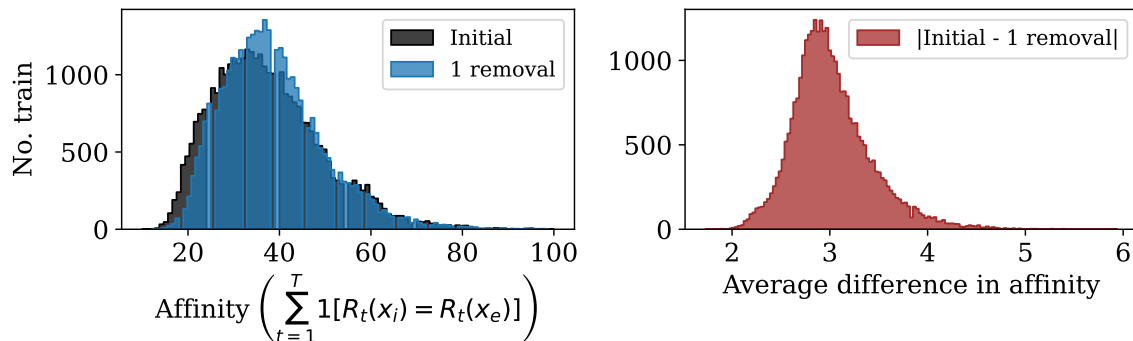
Figure 5: *Left*: Distribution of training-example affinities to a randomly selected test example z_e using an LGB model trained on the Adult data set before (initial) and after the removal of a single training example (1 removal) ordered using LOO. *Right*: Average change in affinity over 100 randomly selected test examples. The changing distribution of affinity values signals structural changes to the tree structures after only a single removal.

affinity to the target instance. Hence, as more instances are added to the deletion group, LeafRefit quickly outperforms LOO. This phenomenon holds regardless of whether each training instance's influence is measured once or when the remaining instances' influences are reestimated and reordered after each deletion.

The key takeaway of these results is that *the fixed structure assumption yields better group influence estimates*. In short, the assumption allows an estimator to ignore the noisy effects of structural changes and instead focus on the training instances' similarity to the target.

Figure 5 validates structural changes are induced by a single removal ordered by LOO; specifically, the figure reports the training-example affinities (i.e., the number of times the training examples are assigned to the same leaf as the test example across all trees in the ensemble).

6. Additional Related Work

Kong and Chaudhuri (2021) adapt TracIn to variational autoencoders while Zhang et al. (2021) adapt influence functions and TracIn to NLP models with transformer architectures, improving the understanding of influence estimation in the context of unsupervised and large deep learning models, respectively. In a similar vein, Terashita et al. (2021) adapt existing influence-estimation methods to generative adversarial networks (GANs).

HyDRA (Chen et al., 2021) and SGDCleanse (Hara et al., 2019) are two additional influence-estimation methods designed for deep learning models. They both compute the influence of z_i on z_e by estimating the effect of z_i on the *entire* model (including the optimization process), and then estimating the effect the changing model has on the loss of z_e. These approaches are similar but significantly different from influence functions, which measures the change in the model with respect to the *final* model parameters; this

is also different from TracIn, which treats each checkpoint as independent and aggregates the marginal *local* changes on the loss of z_e at each one. Translating these approaches to GBDTs would require a way of estimating the effect of z_i on the model *structure*, of which there are an exponential number of potential structures; thus, it is not immediately obvious how one would approximate the effect of z_i on all relevant models. In our experiments, LOO and SubSample substitute as reasonable approximations for estimating structural changes; however, it may be valuable future work to explore this direction in greater depth.

A tangentially-related direction of research is *prototypes* (and their complement: *criticisms*); prototypes attempt to summarize a data set by identifying training examples in high- and low-density regions of the input space (Bien and Tibshirani, 2011; Kim et al., 2016; Gurumoorthy et al., 2019). Although typically model-agnostic, model-specific versions exist such as TreeProto (Tan et al., 2020). These approaches tend to work well at providing a *global* perspective of a given data set; however, these approaches differ significantly from the methods described in this paper, which attempt to return the most influential training examples for a *given* test example or *set* of test examples. Similar to prototypes, data set cartography maps (Swayamdipta et al., 2020) provide a global perspective of the training data set, characterizing training examples as easy, hard, or ambiguous to learn by measuring the confidence and variability of their predictions throughout the training process.

A highly related but significantly different body of research is *feature-based* influence estimation. There is a plethora of work in this area (Ribeiro et al., 2016; Selvaraju et al., 2017; Lundberg et al., 2018; Ghorbani et al., 2019; Koh et al., 2020; Sundararajan and Najmi, 2020) which estimates the loss of z_e with respect to each *feature*. Although feature-based approaches are currently more prevalent than instance-attribution methods, instance-based influence techniques are becoming increasingly popular as machine-learning practitioners and researchers are shifting their focus from solely analyzing the quality of their models to analyzing the quality of their data and the effect their data has on their models (Barshan et al., 2020; Brophy and Lowd, 2021; Hammoudeh and Lowd, 2022b). Both influence approaches are not mutually exclusive, and using a combination of feature- and instance-based influence estimation may provide the most informative context for a given prediction.a

7. Conclusions and Future Work

In this work, we adapt recent popular influence-estimation methods designed for deep learning models to GBDTs, identify theoretical similarities between BoostIn and LeafInfluence, and provide a comprehensive evaluation of each influence method across many data sets using multiple GBDT implementations.

Overall, we find LeafRefit typically works best at finding influential examples for a *single* test instance; however, this method is extremely slow and intractable in most cases. Thus, we believe BoostIn is a viable alternative that performs comparably for the single test case, and significantly outperforms LeafRefit, LeafInfluence, and most other methods when identifying influential instances for a *set* of test instances while being orders of magnitude more efficient, providing an effective and scalable solution for influence estimation in GBDTs.

Our findings also suggest that LOO consistently identifies the *single-most* influential example to a given test prediction, but performs poorly at finding the most influential *set* of examples due to small but significant structural changes in response to removing one or very

few examples. We find methods assuming a fixed structure when computing influence values generally perform better than model-agnostic approaches that do not. These structural changes also help explain why LOO is not correlated with any other methods, and why the previous state-of-the-art, LeafInfluence, is actually a poor approximation of LOO.

For future work, further investigation regarding the structural sensitivity of GBDTs as it relates to influence estimation may lead to influence methods that better approximate the changing GBDT model resulting from one or more data removals. Another potential avenue is trying to determine the *exact* number of training examples responsible for a given prediction. Finally, testing the effectiveness of these methods for different applications mentioned in the introduction would also be valuable future work.

Acknowledgments

Appendix A. Implementation and Experiment Details

Experiments are run on an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.6GHz with 100GB of DDR4 RAM @ 2.4GHZ. We use Cython (Behnel et al., 2011), a Python package allowing the development of C extensions, to store a unified representation of the model structure to which we can then apply the specified influence-estimation method. Experiments are run using Python 3.9.6, and source code for all influence-estimation implementations and all experiments is available at `https://github.com/jjbrophy47/tree_influence`. The library currently supports all major modern gradient boosting frameworks including Scikit-learn[11] (Pedregosa et al., 2011), XGBoost (Chen and Guestrin, 2016), LightGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018).

A.1 Data Sets

We perform experiments on 22 real-world data sets that represent problems well-suited for tree-based models. For each data set, we generate one-hot encodings for any categorical variable and leave all numeric and binary variables as is. For any data set without a designated train and test split, we randomly sample 80% of the data for training and use the rest for testing. Table 2 summarizes the data sets after preprocessing.

- **Adult** (Dua and Graff, 2019) contains 48,842 instances (11,687 positive) of 14 demographic attributes to determine if a person's personal income level is more than $50K per year (binary classification).

---

11. For our experiments, we use `HistGradientBoostingRegressor` and `HistGradientBoostingClassifier`.

- **Bank** (Moro et al., 2014; Dua and Graff, 2019) consists of 41,188 marketing phone calls (4,640 positive) from a Portuguese banking institution. There are 20 attributes, and the aim is to figure out if a client will subscribe (binary classification).

- **Bean** (Koklu and Ozkan, 2020; Dua and Graff, 2019) consists of 13,611 images of grains. The aim is to classify each image into one of 7 different types of registered dry beans based on 16 features extracted from the image (multiclass classification).

- **COMPAS** (Larsen et al., 2016; Ofer, 2017) is a recidivism data set consisting of 6,172 defendants (2,751 deemed "high-risk") characterized by 11 attributes. The aim is to decide whether or not the defendant is at 'high-risk" to be a repeat offender (binary classification).

- **Concrete** (Yeh, 1998; Dua and Graff, 2019) consists of 1,030 instances of concrete characterized by 8 attributes. The aim is to predict the compressive strength of the concrete (regression).

- **Credit** (Yeh and Lien, 2009; Dua and Graff, 2019) consists of the payment credibility of 30,000 people in Taiwan (6,636 people with bad credibility). Each person is characterized by 23 attributes relating to default payments. The aim is to predict the credibility of the client (binary classification).

- **Diabetes** (Strack et al., 2014; Dua and Graff, 2019) consists of 101,766 instances of patient and hospital readmission outcomes (46,902 readmitted) characterized by 55 attributes (binary classification).

- **Energy** (Tsanas and Xifara, 2012; Dua and Graff, 2019) consists of 768 buildings in which each building is one of 12 different shapes and is characterized by 8 features. The aim is to predict the cooling load associated with the building (regression).

- **Flight** (Research and Administration, 2019) consists of 100,000 actual arrival and departure times of flights by certified U.S. air carriers; the data was collected by the Bureau of Transportation Statistics' (BTS) Office of Airline Information. The data contains 8 attributes and 19,044 delays. The task is to predict if a flight will be delayed (binary classification).

- **German** (Dua and Graff, 2019) consists of 1,000 credit applicants characterized by 20 attributes. The aim is to predict whether the person is a good or bad credit risk (binary classification).

- **HTRU2** (Lyon et al., 2016; Dua and Graff, 2019) consists of 17,898 pulsar candidates characterized by 8 attributes. The aim is to predict whether the pulsar is legitimate or a spurious example (binary classification).

- **Life** (Rajarshi, 2017) consists of 2,928 instances of life expectancy estimates for various countries during a specific year. Each instance is characterized by 20 attributes, and the aim is to predict the life expectancy of the country during a specific year (regression).

- **Naval** (Coraddu et al., 2016; Dua and Graff, 2019) consists of 11,934 instances extracted from a high-performing gas turbine simulation. Each instance is characterized by 16 features. The aim is to predict the gas turbine decay coefficient (regression).

- **No Show** (Hoppen, 2016) contains 110,527 instances of patient attendances for doctors' appointments (22,319 no shows) characterized by 14 attributes. The aim is to predict whether or not a patient shows up to their doctors' appointment (binary classification).

- **Obesity** (Suzanne, 2018) contains 48,346 instances of obesity rates for different states and regions with differing socioeconomic backgrounds. Each instance is characterized by 32 attributes. The aim is to predict the obesity rate of the region (regression).

- **Power** (Kaya et al., 2012; Tüfekci, 2014; Dua and Graff, 2019) contains 9,568 readings of a Combined Cycle Power Plant (CCPP) at full work load. Each reading is characterized by 4 features. The aim is to predict the net hourly electrical energy output (regression).

- **Protein** (Dua and Graff, 2019) contains 45,730 tertiary-protein-structure instances characterized by 9 attributes. The aim is to predict the armstrong coefficient of the protein structure (regression).

- **Spambase** (Dua and Graff, 2019) consists of 4,601 emails (1,813 spam) characterized by 57 attributes. The aim is to predict whether or not the email is spam (binary classification).

- **Surgical** (Kaggle, 2018) consists of 14,635 medical patient surgeries (3,690 surgeries with complications), characterized by 25 attributes; the goal is to predict whether or not a patient had a complication from their surgery (binary classification).

- **Twitter** uses the first 250,000 tweets (33,843 spam) of the HSpam14 data set (Sedhai and Sun, 2015). Each instance contains the tweet ID and label. After retrieving the text and user ID for each tweet, we derive the following attributes: no. chars, no. hashtags, no. mentions, no. links, no. retweets, no. unicode chars., and no. messages per user. The aim is to predict whether a tweet is spam or not (binary classification).

- **Vaccine** (Bull et al., 2016; DrivenData, 2019) consists of 26,707 survey responses collected between October 2009 and June 2010 asking people a range of 36 behavioral and personal questions, and ultimately asking whether or not they got an H1N1 and/or seasonal flu vaccine. Our aim is to predict whether or not a person received a seasonal flu vaccine (binary classification).

- **Wine** (Cortez et al., 2009; Dua and Graff, 2019) consists of 6,497 instances of Portuguese "Vinho Verde" red and white wine. Each instance is characterized by 11 features. The aim is to predict the quality of the wine from 0-10 (regression).

Data set	Task	Metric	No. instances	Pos. %	No. attr.	SDS?
Bank	binary	AUC	41,188	11.3	63	
Flight	binary	AUC	100,000	19.0	650	
HTRU2	binary	AUC	17,898	9.2	8	✓
No Show	binary	AUC	110,527	20.2	89	
Twitter	binary	AUC	250,000	13.5	14	
Adult	binary	Acc.	48,842	23.9	108	
COMPAS	binary	Acc.	6,172	44.6	10	✓
Credit Card	binary	Acc.	30,000	22.1	23	✓
Diabetes	binary	Acc.	101,766	46.1	255	
German	binary	Acc.	1,000	30.0	27	✓
Spambase	binary	Acc.	4,601	39.4	57	✓
Surgical	binary	Acc.	14,635	25.2	90	✓
Vaccine	binary	Acc.	26,707	46.6	155	
Bean	multiclass	Acc.	13,611	-	16	✓
Concrete	regression	MSE	1,030	-	8	✓
Energy	regression	MSE	768	-	16	✓
Life	regression	MSE	2,928	-	204	✓
Naval	regression	MSE	11,934	-	17	✓
Obesity	regression	MSE	48,346	-	100	
Power	regression	MSE	9,568	-	4	✓
Protein	regression	MSE	45,730	-	9	
Wine	regression	MSE	6,497	-	11	✓

Table 2: Data set summary after preprocessing. AUC = area under the ROC curve, Acc. = Accuracy, MSE = mean squared error, No. attr. = number of attributes, SDS = small data subset (data sets for which LeafRefit and LeafInfluence are tractable).

A.2 Predictive Performance of GBDTs

This section evaluates the predictive performance of the four most-popular modern gradient-boosting frameworks: LightGBM (LGB), XGBoost (XGB), CatBoost (CB), and Scikit-learn boosting (SGB).

Data set	GBDT				Non-GBDT					
	LGB	**XGB**	**CB**	**SGB**	**LR**	**DT**	**KNN**	**SVM**	**RF**	**MLP**
AUC (binary classification) (↑)										
Bank	**0.951**	0.947	0.948	0.949	0.932	0.930	0.930	0.930	0.924	0.934
Flight	0.748	**0.749**	0.745	0.747	0.707	0.696	0.687	0.662	0.688	0.720
HTRU2	**0.982**	0.981	0.981	0.981	0.978	0.966	0.964	0.955	0.977	0.972
No Show	0.621	**0.622**	0.621	0.621	0.601	0.603	0.592	0.538	0.612	0.609
Twitter	**0.927**	0.924	0.917	**0.927**	0.808	0.893	0.859	0.836	0.884	0.897
Accuracy (binary classification) (↑)										
Adult	**0.874**	**0.874**	**0.874**	0.873	0.853	0.861	0.803	0.852	0.852	0.818
COMPAS	0.752	0.770	**0.777**	0.770	0.768	0.747	0.746	0.760	0.768	0.769
Credit	**0.822**	**0.822**	0.820	0.821	0.810	**0.822**	0.780	0.819	0.821	0.760
Diabetes	0.648	0.648	**0.650**	0.648	0.637	0.631	0.602	0.643	0.628	0.626
German	0.735	0.710	0.705	**0.745**	0.730	0.730	0.720	0.720	0.720	0.645
Spambase	**0.957**	0.952	0.955	**0.957**	0.933	0.941	0.810	0.940	0.932	0.941
Surgical	**0.909**	**0.909**	**0.909**	0.908	0.800	0.894	0.886	0.803	0.821	0.788
Vaccine	0.811	0.811	0.807	**0.813**	0.807	0.780	0.771	0.805	0.785	0.750
Accuracy (multiclass classification) (↑)										
Bean	0.930	**0.931**	**0.931**	0.930	0.927	0.908	0.737	**0.931**	0.918	0.505
MSE (regression) (↓)										
Concrete	20.1	21.6	23.9	**18.8**	124.9	69.4	98.9	102.4	54.7	50.6
Energy	0.28	**0.10**	0.13	0.26	0.97	0.36	2.10	10.03	0.36	20.18
Life	**3.21**	3.30	3.22	3.40	3.72	6.98	69.36	8.44	6.99	6e4
Naval	**4e-7**	8e-7	6e-6	**4e-7**	3e-6	1e-6	6e-6	6e-5	2e-5	1e1
Obesity	**0.027**	0.043	0.033	0.038	0.115	0.043	5.191	0.786	0.676	1.0e4
Power	8.5	**8.4**	8.8	8.6	20.3	16.1	15.1	17.1	16.5	24.7
Protein	13.4	13.6	14.9	**13.3**	26.6	20.9	33.2	23.9	23.9	329.1
Wine	**0.384**	0.422	0.427	0.389	0.528	0.524	0.626	0.446	0.524	0.512

Table 3: Predictive performance of GBDTs against alternative methods: logistic regression (LR), decision tree (DT), k-nearest neighbor (KNN), support vector machine with an RBF kernel (SVM), random forest (RF), and a multilayer peceptron (MLP), all evaluated on the test set of each data set. We use MSE to evaluate regression models, accuracy (acc.) for models trained on multiclass data sets or binary data sets with a positive label percentage $> 20\%$, and AUC for the rest; see Table 2 for reference.

We compare LGB, XGB, CB, and SGB against alternative methods that are arguably more interpretable:

- Logistic regression (LR): Logistic regression implementation from Scikit-Learn (Pedregosa et al., 2011); we tune the regularization hyperparameter using values [l1, l2], and the penalty hyperparameter C using values [0.01, 0.1, 1.0].

- Decision tree (DT): Single decision tree implementation from Scikit-Learn (Pedregosa et al., 2011); we tune the split criterion hyperparameter using values [gini, entropy], the decision node splitter using values [best, random], and the maximum depth of the tree using values [3, 5, 10, no limit].

- k-nearest neighbor (KNN): k-nearest neighbor implementation from Scikit-Learn (Pedregosa et al., 2011); we tune k using values [3, 5, 7, 11, 15, 31, 61].

- Support vector machine (SVM): Support vector machine implementation from Scikit-Learn (Pedregosa et al., 2011); we use a radial basis function (RBF) kernel and tune the penalty hyperparameter C using values [0.01, 0.1, 1.0].

- Random forest (RF): Random forest implementation from Scikit-Learn (Pedregosa et al., 2011); we tune the number of trees using values [10, 25, 50, 100, 200], and maximum depth using values [2, 3, 4, 5, 6, 7].

- Multilayer perceptron (MLP): Multilayer perceptron implementation from Scikit-Learn (Pedregosa et al., 2011); we tune the number of layers and number of nodes per layer using values [(100,), (100, 100)].

For the LGB, XGB, CB, and SGB models, we tune the number of trees/boosting iterations (T) using values [10, 25, 50, 100, 200]. Since the LGB and SGB models grow trees in a leaf-wise (depth-first) manner, we tune the maximum number of leaves (l_{max}) for LGB and SGB using values [15, 31, 61, 91]. In contrast, we tune the the maximum depth (d_{max}) for XGB and CB using values [2, 3, 4, 5, 6, 7]. We also tune the learning rate (η) for CB using values [0.1, 0.3, 0.6, 0.9], and the maximum number of bins (b_{max}) for SGB using values [50, 100, 250].

We tune all hyperparameters using 5-fold cross-validation. We use mean squared error (MSE) to tune hyperparameters for regression tasks, accuracy for multiclass tasks, and accuracy for binary tasks with a positive label percentage $> 20\%$, otherwise we use area under the ROC curve (AUC). The associated task and metric used to tune hyperparameters for each data set is in Table 2, and the selected hyperparameters for the LGB, XGB, CB, and SGB models are in Table 4.

Table 3 shows the GBDT models consistently outperform the alternative models in terms of predictive performance. These results reaffirm the notion that GBDT models generally outperform more traditional machine-learning algorithms on tabular data and motivate the need for tailored influence-estimation methods for GBDT models to better understand their decision-making processes.

| Data set	LGB T	LGB $l_{\max}$	XGB T	XGB $d_{\max}$	CB T	CB $d_{\max}$	CB η	SGB T	SGB $l_{\max}$	SGB b_{max}
Bank	50	31	100	4	200	7	0.1	50	31	100
Flight	200	91	200	7	200	7	0.3	200	61	250
HTRU2	100	15	100	2	200	3	0.3	50	15	50
No Show	50	61	100	5	200	7	0.3	50	61	100
Twitter	200	91	200	7	200	7	0.6	200	91	250
Adult	100	31	200	3	200	4	0.6	200	15	250
COMPAS	25	91	50	3	50	4	0.3	50	15	50
Credit	50	15	10	3	50	5	0.1	25	15	100
Diabetes	200	31	200	3	200	5	0.3	200	31	100
German	25	15	10	4	100	5	0.1	25	15	100
Spambase	200	31	200	4	200	5	0.3	200	91	250
Surgical	200	15	50	5	200	4	0.1	100	31	250
Vaccine	100	15	100	3	200	4	0.1	100	15	50
Bean	25	15	25	6	200	3	0.3	25	15	50
Concrete	200	15	200	4	200	4	0.3	200	15	50
Energy	200	15	200	5	200	4	0.9	200	15	50
Life	200	61	200	5	200	6	0.3	200	31	250
Naval	200	91	100	7	200	7	0.6	200	91	250
Obesity	200	91	200	7	200	6	0.6	200	91	250
Power	200	61	200	7	200	6	0.6	200	61	250
Protein	200	91	200	7	200	7	0.3	200	91	250
Wine	200	91	100	7	200	7	0.3	200	91	100

Table 4: Hyperparameters selected for the GBDT models. The number of trees/boosting iterations (T), maximum number of leaves ($l_{\max}$), maximum depth ($d_{\max}$), learning rate (η), and maximum number of bins ($b_{\max}$) is found using 5-fold cross-validation. Data sets are grouped based on their task and metric used for evaluation; see Table 2 for reference.

A.3 Summary of Results: All Data Sets

Figure 6 shows a high-level summary of the results including *all* data sets. Overall, we observe very similar trends as when analyzing only the SDS data sets. However, we do notice a decrease in relative performance for SubSample; better performance may be achieved by increasing τ, but this may also significantly increase its running time.
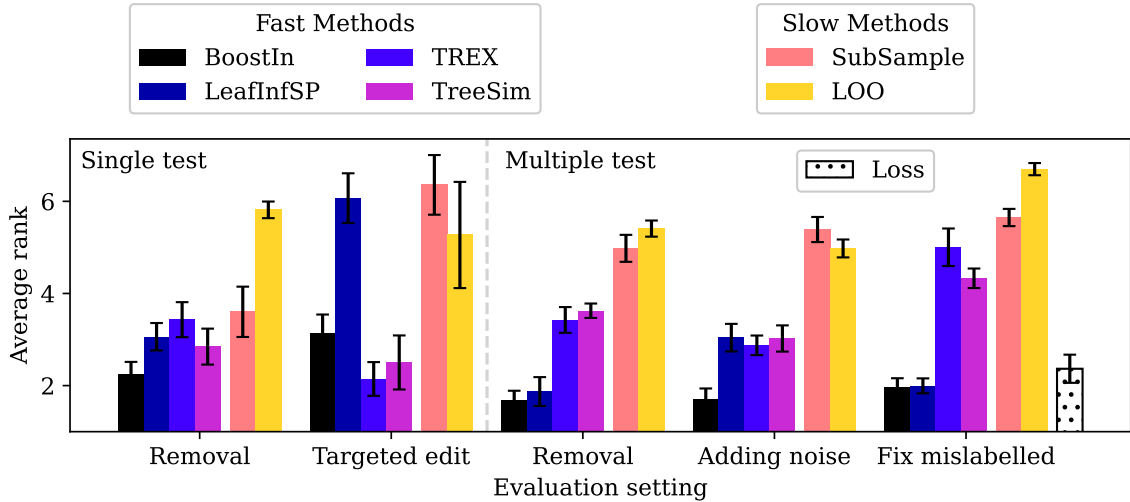
(a) Average rank of each method. For each evaluation setting, results are averaged over all checkpoints, tree types, and data sets; error bars represent 95% confidence intervals and are computed over data sets. Lower is better.
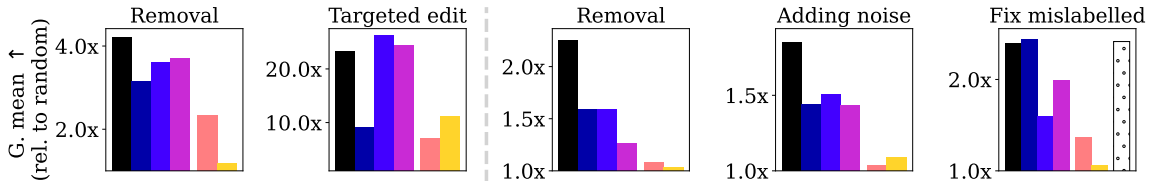
(b) Average *loss* increase (except for "fix mislabelled", which shows average increase in *mislabelled detection*) relative to random. For each evaluation setting, results are averaged over all checkpoints and tree types, then the geometric mean is computed over all data sets. Higher is better.

Figure 6: High-level overview of results including *all* data sets; thus, LeafRefit and LeafInfluence are not included in this analysis.

A.4 Removing Examples (Single Test): Additional Analysis

Figure 7 shows a more fine-grained analysis for the removal experiment involving a single test instance; it also includes an additional baseline: *RandomSL*, which assigns a randomly-sampled positive value for training examples with the same label as the test example, and negative otherwise:[12]

$$\mathcal{I}_{RandomSL}(z_i, z_e) = \mathbb{1}[y_i = y_e]U - \mathbb{1}[y_i \neq y_e]U, \ U \sim \mathcal{U}(0,1)$$

Overall, the trends are relatively consistent across GBDT types; however, we observe TreeSim tends to perform better on XGB and CB than LGB and SGB.
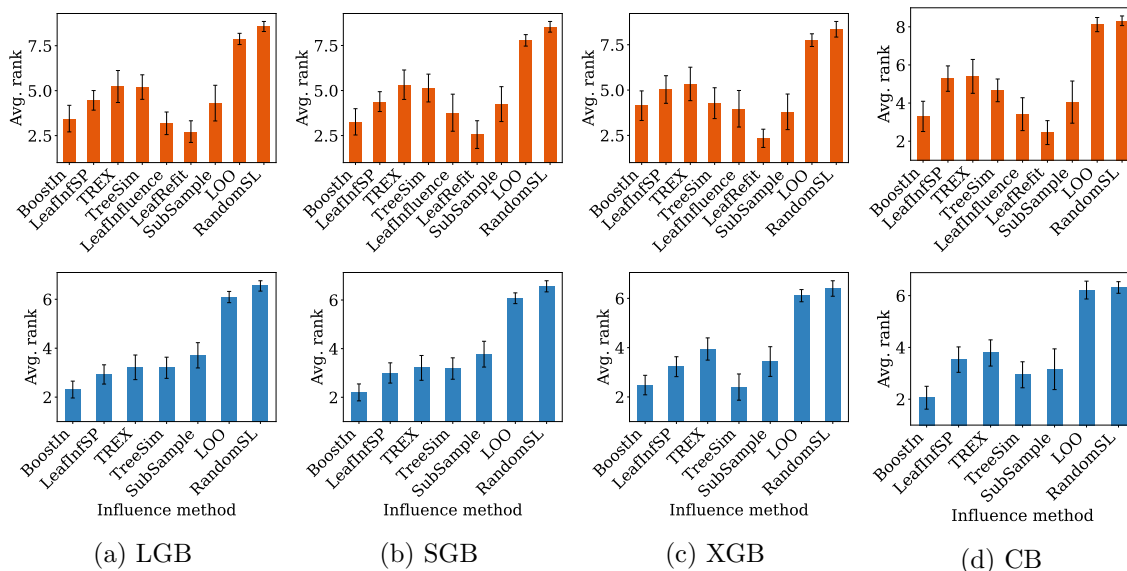
Figure 7: Average ranks when removing examples for a single test instance, shown for each GBDT type. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

Figure 8 shows the change in loss for different GBDT types and data sets. Methods using the fixed-structure assumption tend to perform best, with BoostIn, LeafRefit, and LeafInfluence performing slightly better than the rest on average.

---

12. For regression tasks, $\mathcal{I}_{RandomSL}(z_i, z_e) = \mathcal{N}(\mu_i, \sigma_i)$ in which $\mu_i = 1/|y_i - y_e|$ and $\sigma_i = s.d.(|y_i - y_e|)$.
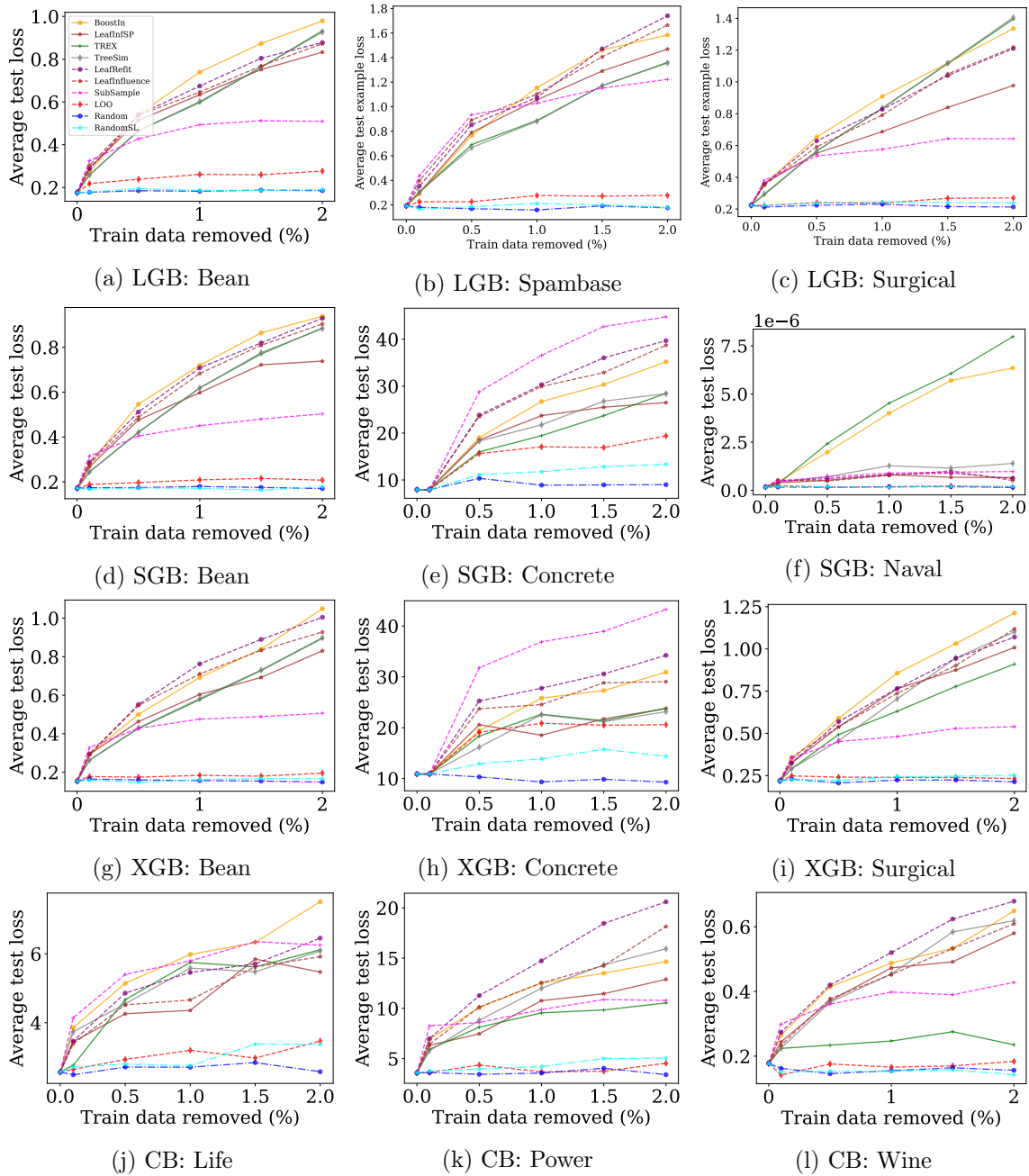
Figure 8: Change in loss for a random test example (averaged over 100 runs) as training examples are removed. Higher is better.

A.5 Targeted Label Edits (Single Test): Additional Analysis

Figure 9 shows more fine-grained ranking analysis for the targeted-label-edit experiment involving a single test instance. The trends are relatively consistent across GBDT types with LeafRefit consistently performing well for the SDS data sets, especially on XGB and CB; for larger data sets, TREX or TreeSim are solid alternative choices to LeafRefit.
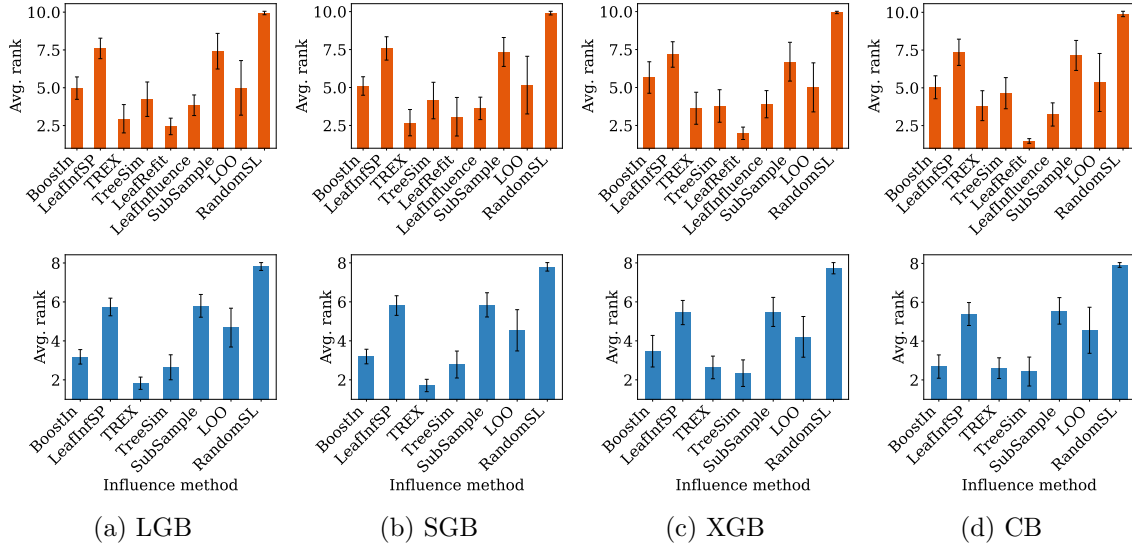

(a) LGB   (b) SGB   (c) XGB   (d) CB

Figure 9: Average ranks when editing training labels to a target label for a single test instance, shown for each GBDT type. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

A.6 Removing Examples (Multiple Test): Additional Analysis

Figure 10 shows the rankings when using different predictive performance measures (e.g., accuracy or AUC) when computing ranks. In contrast to loss, methods rank higher the more they decrease accuracy or AUC as training examples are removed. Overall, BoostIn and LeafInfSP are clear favorites across all three performance metrics.
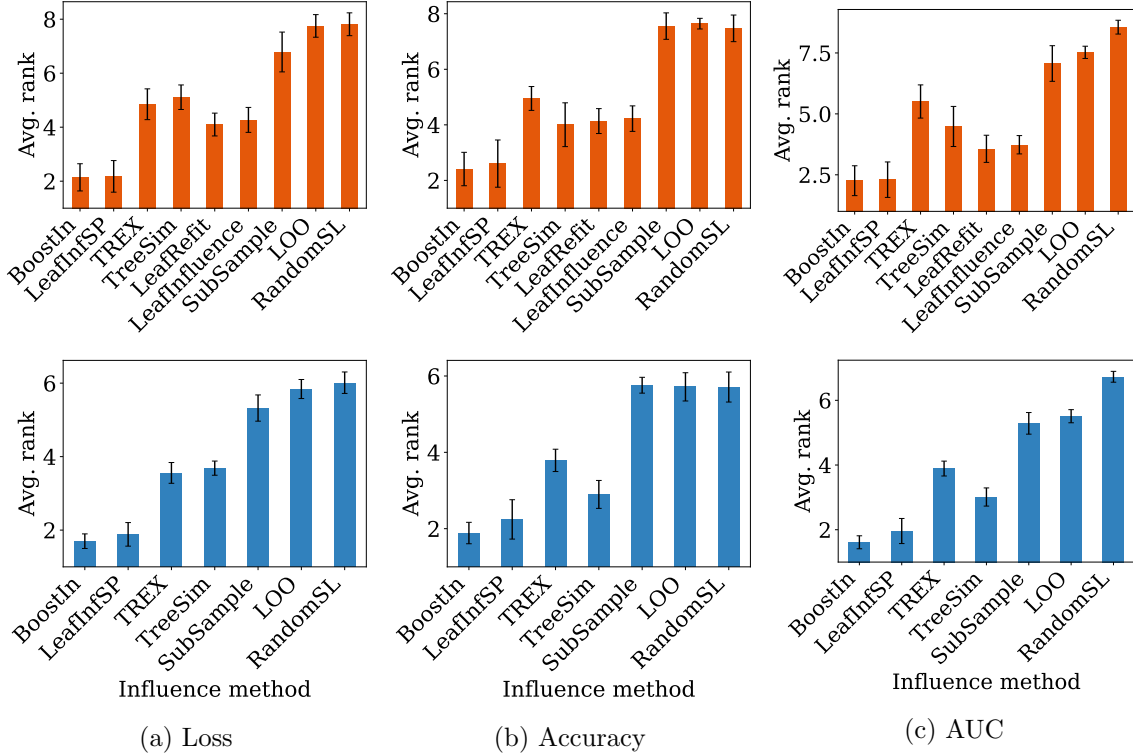

(a) Loss   (b) Accuracy   (c) AUC

Figure 10: Average ranks after removing training examples for multiple test instances and evaluating on a held-out test set using different predictive performance metrics. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints, tree types, and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

A.7 Adding Noise (Multiple Test): Additional Analysis

Figure 11 shows the rankings when using different predictive performance measures (e.g., accuracy or AUC) when computing ranks for the noise addition experiment. In contrast to loss, methods rank higher the more they decrease accuracy or AUC as training examples are removed. Overall, BoostIn is a clear favorite in terms of loss, but both BoostIn and TreeSim perform best for accuracy and AUC.

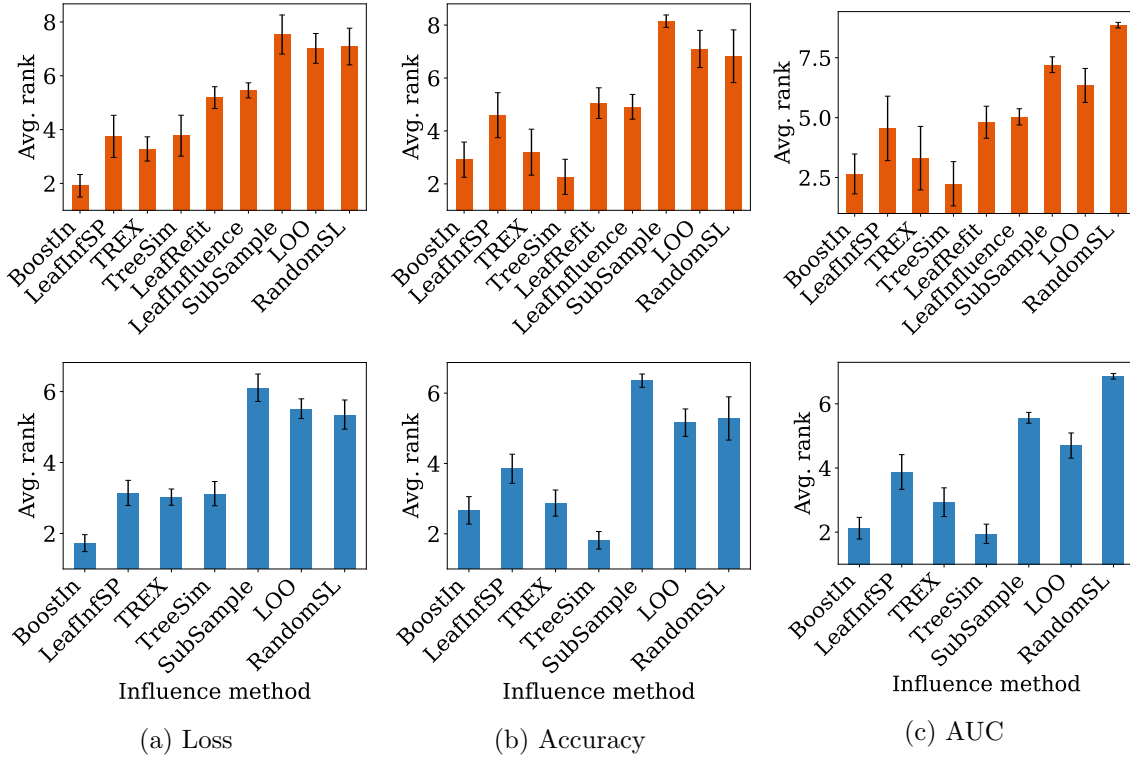(a) Loss   (b) Accuracy   (c) AUC

Figure 11: Average ranks after add noise to training examples for multiple test instances and evaluating on a held-out test set using different predictive performance metrics. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints, tree types, and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

A.8 Fixing Noisy/Mislabelled Examples (Multiple Test): Additional Analysis

Figure 12 shows the average rankings of each method when measuring predictive performance on a held-out test set as noisy/mislabelled training examples are checked and fixed. Methods rank higher the more they decrease loss and increase accuracy or AUC. We also add an additional baseline: BoostIn (self), which measures the influence of each training example on itself, i.e., $\mathcal{I}_{BoostIn}(z_i, z_i)$; those values are then used to order the training examples to be checked/fixed. Overall, BoostIn and LeafInfSP rank highest in terms of loss and AUC; however, TREX and TreeSim tend to work better in increasing accuracy.

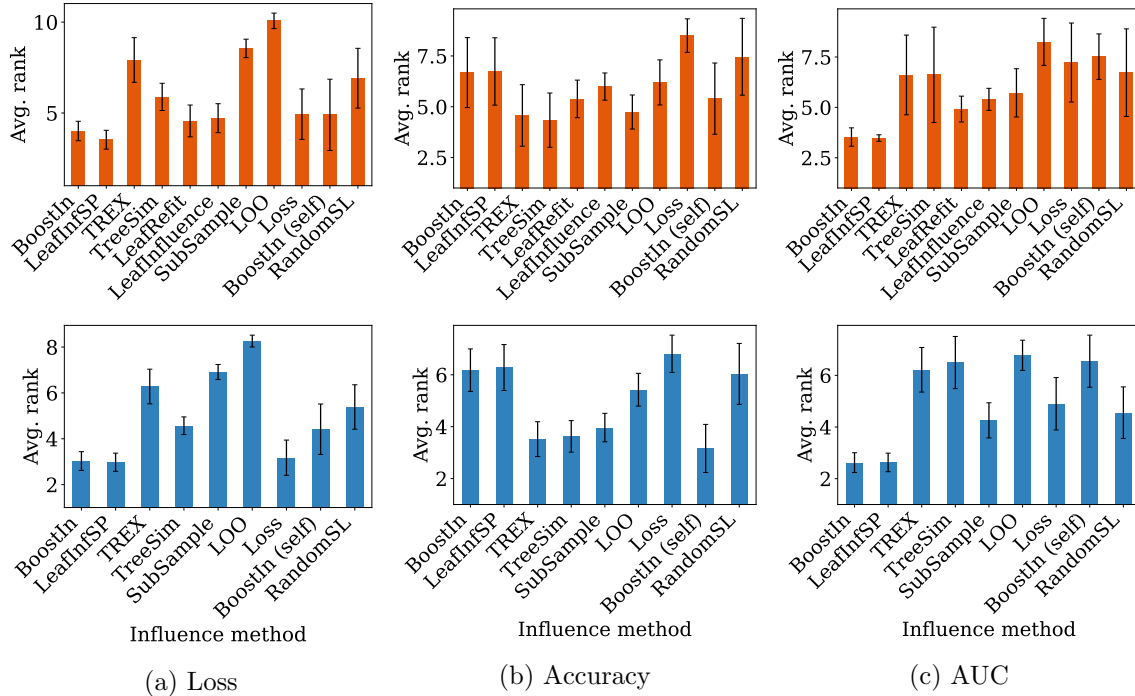

(a) Loss   (b) Accuracy   (c) AUC

Figure 12: Average ranks when measuring the predictive performance on the held-out test set after checking/fixing any noisy/mislabelled training examples. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints, tree types, and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

A.9 Runtime Comparison

Table 5 shows the total time of each method to compute all influence values for a single test instance (fit time + influence time). Each experiment is repeated 5 times, and results are averaged over GBDT types. Results are only shown for the SDS data sets since LeafRefit and LeafInfluence are too intractable to run on the non-SDS data sets.

Data set	TreeSim	BoostIn	LeafInfSP	TREX	SubS.	LOO	LeafRefit	LeafInf.
Bean	0.260	1.021	1.362	3.999	1414	4194	132503	70618
Compas	0.027	0.110	0.139	0.598	175	248	2763	2302
Concrete	0.028	0.175	0.231	0.269	469	101	463	346
Credit	0.068	0.212	0.282	0.837	449	3250	35359	33430
Energy	0.024	0.164	0.248	0.260	411	68	317	214
German	0.005	0.031	0.043	0.420	101	21	70	48
HTRU2	0.123	0.357	0.382	0.861	506	1969	43107	40499
Life	0.107	0.445	0.596	0.936	2604	1440	3414	2706
Naval	0.291	1.140	1.358	4.390	2288	5417	38599	33484
Power	0.230	1.072	1.331	6.636	1658	3390	30938	26201
Spambase	0.121	0.522	0.659	1.201	2724	3064	7935	6313
Surgical	0.171	0.528	0.617	1.694	1170	4135	36798	33455
Wine	0.171	1.020	1.281	2.792	1862	2763	14549	11419

Table 5: Time (in seconds) to compute all influences values for a single test instance for the SDS data sets. Each experiment is repeated 5 times, and results are averaged over GBDT types.

A.10 Correlation Between Influence Methods: Additional Analysis

Figure 13 shows additional correlation heatmaps averaged over the SDS data sets for each GBDT type; overall, the trends remain the same across GBDT types. Figure 14 shows the correlation between influence methods averaged over either all classification or regression data sets; overall, the methods are much less correlated for the regression data sets than the classification data sets (note the difference in legend values in both subplots).
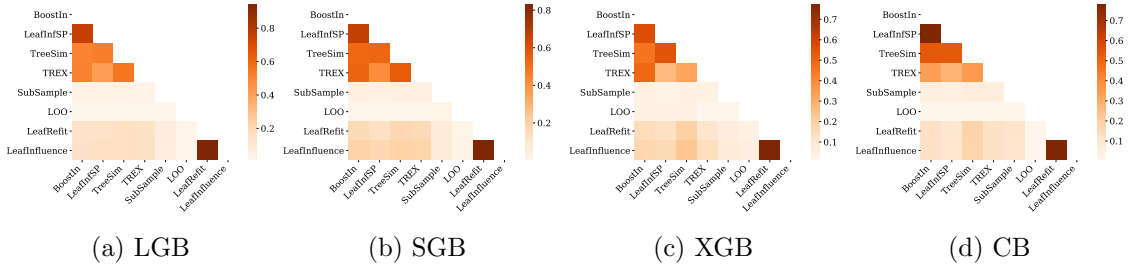

(a) LGB   (b) SGB   (c) XGB   (d) CB

Figure 13: Spearman correlation coefficient between influence methods for each GBDT type, averaged over 100 test examples and SDS data sets.

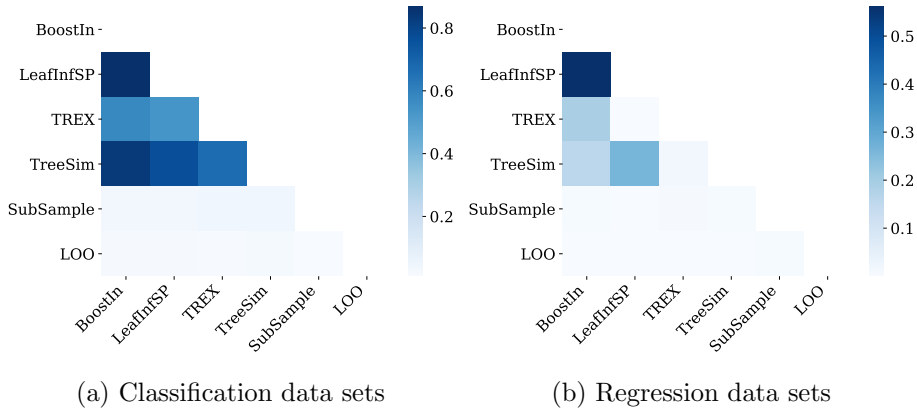(a) Classification data sets   (b) Regression data sets

Figure 14: Spearman correlation coefficient between influence methods averaged over 100 test examples, all GBDT types, and either classification or regression data sets.

A.11 The Structural Fragility of LOO: Additional Examples

Figure 15 shows additional examples of LOO choosing the *single-most* influential example.

(a) LGB: Energy   (b) LGB: Life

(c) LGB: Wine   (d) XGB: COMPAS

(e) XGB: Concrete   (f) XGB: Energy
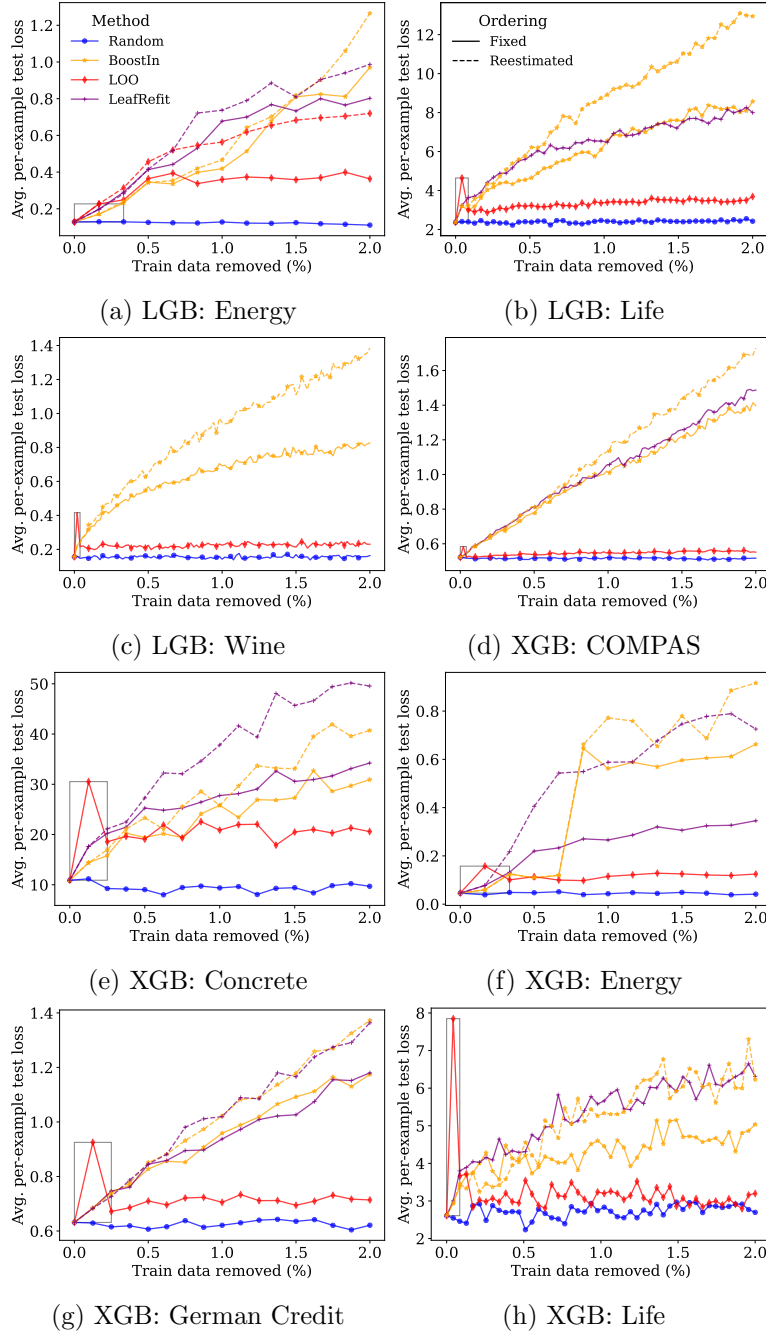
(g) XGB: German Credit   (h) XGB: Life

Figure 15: Change in loss (mean over 100 test instances) as training examples are removed *one at a time*. Higher is better. The gray box highlights the ability of LOO to identify the single-most influential training example on the given test instance.

References

Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.

Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger Grosse. If influence functions are the answer, then what is the question? In *Proceedings of the 36th Conference on Neural Information Processing Systems*, NeurIPS'22. Curran Associates, Inc., 2022.

Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. RelatIF: Identifying explanatory training samples via relative influence. In *Proceedings of the Twenty-Third International Conference on Artificial Intelligence and Statistics*, pages 1899–1909. PMLR, 2020.

Samyadeep Basu, Phil Pope, and Soheil Feizi. Influence functions in deep learning are fragile. In *Proceedings of the Eighth International Conference on Learning Representations*, 2020.

Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13 (2):31–39, 2011.

Jacob Bien and Robert Tibshirani. Prototype selection for interpretable classification. *The Annals of Applied Statistics*, pages 2403–2424, 2011.

Emily Black and Matt Fredrikson. Leave-one-out unfairness. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT'21, 2021.

Adam Bloniarz, Ameet Talwalkar, Bin Yu, and Christopher Wu. Supervised neighborhoods for distributed nonparametric regression. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 1450–1459, 2016.

Casper Solheim Bojer and Jens Peder Meldgaard. Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, 37(2):587–603, 2021.

Joschka Braun, Micha Kornreich, JinHyeong Park, Jayashri Pawar, James Browning, Richard Herzog, Benjamin Odry, and Li Zhang. Influence based re-weighing for labeling noise in medical imaging. In *2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI)*, pages 1–5. IEEE, 2022.

Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In *Proceedings of the Thirty-Eighth International Conference on Machine Learning*, pages 1092–1104. PMLR, 2021.

Peter Bull, Isaac Slavitt, and Greg Lipstein. Harnessing the power of the crowd to increase capacity for data science in the social sector. In *Proceedings of the International Workshop #Data4Good*, 2016.

Ruth MJ Byrne. Counterfactuals in explainable artificial intelligence (XAI): Evidence from human reasoning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 6276–6282, 2019.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the Twenty-Second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Yuanyuan Chen, Boyang Li, Han Yu, Pengcheng Wu, and Chunyan Miao. HyDRA: Hypergradient data relevance analysis for interpreting deep neural networks. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, volume 35, pages 7081–7089, 2021.

R Dennis Cook and Sanford Weisberg. *Residuals and Influence in Regression*. Chapman and Hall, New York, 1982. ISBN 041224280X.

Andrea Coraddu, Luca Oneto, Aessandro Ghio, Stefano Savio, Davide Anguita, and Massimo Figari. Machine learning approaches for improving condition-based maintenance of naval propulsion plants. *Journal of Engineering for the Maritime Environment, Part M*, 230(1):136–153, 2016.

Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

Alex Davies and Zoubin Ghahramani. The random forest kernel and other kernels for big data from random partitions. *arXiv preprint arXiv:1402.4293*, 2014.

DrivenData. Flu shot learning: Predict H1N1 and seasonal flu vaccines. `https://www.drivendata.org/competitions/66/flu-shot-learning/data/`, 2019. [Online; accessed 12-August-2020].

Dheeru Dua and Casey Graff. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2019. [Online; accessed 12-September-2021].

Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. In *Proceedings of the Thirty-Fourth International Conference on Neural Information Processing Systems*, volume 33, pages 2881–2891, 2020.

Juliana J Ferreira and Mateus S Monteiro. What are people doing about XAI user experience? A survey on AI explainability research and practice. In *Proceedings of the Tweenty-Second International Conference on Human-Computer Interaction*, pages 56–73. Springer, 2020.

Andrew Fogg. Anthony Goldbloom gives you the secret to winning Kaggle competitions. `https://www.kdnuggets.com/2016/01/anthony-goldbloom-secret-winning-kaggle-competitions.html`, 2016. [Online; accessed 12-September-2021].

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

Jerome H Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, pages 1189–1232, 2001.

Amirata Ghorbani and James Zou. Data Shapley: Equitable valuation of data for machine learning. In *Proceedings of the Thirty-Sixth International Conference on Machine Learning*, pages 2242–2251. PMLR, 2019.

Amirata Ghorbani, James Wexler, James Y Zou, and Been Kim. Towards automatic concept-based explanations. In *Proceedings of the Thirty-Third International Conference on Neural Information Processing Systems*, volume 32, pages 9277–9286, 2019.

Karthik S Gurumoorthy, Amit Dhurandhar, Guillermo Cecchi, and Charu Aggarwal. Efficient data representation by selecting prototypes with importance weights. In *Proceedings of the Nineteenth IEEE International Conference on Data Mining (ICDM)*, pages 260–269. IEEE, 2019.

Zayd Hammoudeh and Daniel Lowd. Training data influence analysis and estimation: A survey. *arXiv preprint arXiv:2212.04612*, 2022a. URL `https://arxiv.org/abs/2212.04612`.

Zayd Hammoudeh and Daniel Lowd. Identifying a training-set attack's target using renormalized influence estimation. In *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security*, CCS'22, Los Angeles, CA, 2022b. Association for Computing Machinery. URL `https://arxiv.org/abs/2201.10055`.

Zayd Hammoudeh and Daniel Lowd. Reducing certified regression to certified classification for general poisoning attacks. In *Proceedings of the 1st IEEE Conference on Secure and Trustworthy Machine Learning*, SaTML'23, 2023. URL `https://arxiv.org/abs/2208.13904`.

Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974.

Kazuaki Hanawa, Sho Yokoi, Satoshi Hara, and Kentaro Inui. Evaluation of similarity-based explanations. In *Proceedings of the Ninth International Conference on Learning Representations*, 2021.

Satoshi Hara, Atsushi Nitanda, and Takanori Maehara. Data cleansing for models trained with SGD. In *Proceedings of the Thirty-Third International Conference on Neural Information Processing Systems*, pages 4213–4222, 2019.

Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9. ACM, 2014.

Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. In *Proceedings of the Thirty-Third International Conference on Neural Information Processing Systems*, volume 32, pages 9737–9748, 2019.

Joni Hoppen. Medical appointment no shows. `https://www.kaggle.com/joniarroba/noshowappointments`, 2016. [Online; accessed 25-January-2021].

Louis A. Jaeckel. The infinitesimal jackknife. Technical report, Bell Laboratories, 1972.

Kaggle. Dataset surgical binary classification. `https://www.kaggle.com/omnamahshivai/surgical-dataset-binary-classification/version/1#`, 2018. [Online; accessed 29-July-2020].

Heysem Kaya, Pmar Tüfekci, and Fikret S Gürgen. Local and global learning methods for predicting power of a combined gas & steam turbine. In *Proceedings of the Second International Conference on Emerging Trends in Computer and Electronics Engineering (ICETCEE)*, pages 13–18, 2012.

Guolin Ke, Qi Meng, et al. LightGBM: A highly efficient gradient boosting decision tree. In *Proceedings of the Thirty-First International Conference on Neural Information Processing Systems*, volume 30, 2017.

Mark T Keane and Barry Smyth. Good counterfactuals and where to find them: A case-based technique for generating counterfactuals for explainable AI (XAI). In *Proceedings of the Twenty-Eighth International Conference on Case-Based Reasoning*, pages 163–178. Springer, 2020.

Been Kim, Rajiv Khanna, and Oluwasanmi O. Koyejo. Examples are not enough, learn to criticize! Criticism for interpretability. In *Proceedings of the Thirtieth International Conference on Neural Information Processing Systems*, pages 2280–2288, 2016.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, pages 5338–5348. PMLR, 2020.

Murat Koklu and Ilker Ali Ozkan. Multiclass classification of dry beans using computer vision and machine learning techniques. *Computers and Electronics in Agriculture*, 174: 105507, 2020.

Zhifeng Kong and Kamalika Chaudhuri. Understanding instance-based interpretability of variational auto-encoders. In *Proceedings of the Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

Jeff Larsen, Surya Mattu, Lauren Kirchner, and Julia Angwin. How we analyzed the COMPAS recidivism algorithm. `https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm`, 2016. [Online; accessed 12-September-2021].

Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defense against general poisoning attacks. In *Proceedings of the Thirty-Fourth International Conference on Neural Information Processing Systems*, 2020.

Zhuoming Liu, Hao Ding, Huaping Zhong, Weijia Li, Jifeng Dai, and Conghui He. Influence selection for active learning. In *Proceedings of the 18th International Conference on Computer Vision*, ICCV'21, 2021.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the Thirty-First International Conference on Neural Information Processing Systems*, pages 4768–4777, 2017.

Scott M Lundberg, Gabriel G Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

Robert J Lyon, BW Stappers, Sally Cooper, John Martin Brooke, and Joshua D Knowles. Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1): 1104–1123, 2016.

Andrey Malinin, Liudmila Prokhorenkova, and Aleksei Ustimenko. Uncertainty in gradient boosting via ensembles. In *Proceedings of the Ninth International Conference on Learning Representations*, 2021.

Frank Moosmann, Bill Triggs, and Frederic Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Proceedings of the Twentieth International Conference on Neural Information Processing Systems*, pages 985–992, 2006.

Sérgio Moro, Paulo Cortez, et al. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 2014.

Dan Ofer. COMPAS recidivism racial bias. `https://www.kaggle.com/danofer/compass`, 2017. [Online; accessed 12-September-2021].

Sejoon Oh, Sungchul Kim, Ryan A. Rossi, and Srijan Kumar. Influence-guided data augmentation for neural tensor completion. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, CIKM'21. ACM, 2021.

F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *JMLR*, 2011.

Geoff Pleiss, Tianyi Zhang, Ethan Elenberg, and Kilian Weinberger. Identifying mislabeled data using the area under the margin ranking. In *Proceedings of the Thirty-Fourth International Conference on Neural Information Processing Systems*, 2020.

Gregory Plumb, Denali Molitor, and Ameet S Talwalkar. Model agnostic supervised local explanations. In *Proceedings of the Thirty-Second International Conference on Neural Information Processing Systems*, pages 2515–2524, 2018.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *Proceedings of the Seventh International Conference on Learning Representations*, 2019.

Liudmila Prokhorenkova, Gleb Gusev, et al. CatBoost: Unbiased boosting with categorical features. In *Proceedings of the Thirty-Second International Conference on Neural Information Processing Systems*, 2018.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Proceedings of the Thirty-Fourth International Conference on Neural Information Processing Systems*, volume 33, 2020.

Kumar Rajarshi. Life expectancy (WHO). `https://www.kaggle.com/kumarajarshi/life-expectancy-who?ref=hackernoon.com&select=Life+Expectancy+Data.csv`, 2017. [Online; accessed 12-September-2021].

Research and Innovative Technology Administration. Airline on-time performance and causes of flight delays. `https://catalog.data.gov/dataset/airline-on-time-performance-and-causes-of-flight-delays-on-time-data`, 2019. [Online; accessed 16-April-2020].

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the Twenty-Second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *Proceedings of the Fourteenth International Conference on Computational Learning Theory*, pages 416–426. Springer, 2001.

Surendra Sedhai and Aixin Sun. HSpam14: A collection of 14 million tweets for hashtag-oriented spam research. In *Proceedings of the Thirty-Eighth International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

LS Shapley. The value of n-person games. *Annals of Mathematics Studies*, 28:307–317, 1953.

Boris Sharchilev, Yury Ustinovskiy, Pavel Serdyukov, and Maarten Rijke. Finding influential training samples for gradient boosted decision trees. In *Proceedings of the Thirty-Fifth International Conference on Machine Learning*, pages 4577–4585. PMLR, 2018.

Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *Proceedings of the Thirty-First International Conference on Neural Information Processing Systems*, pages 3520–3532, 2017.

Beata Strack, Jonathan P DeShazo, et al. Impact of HbA1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records. *BioMed Research International*, 2014.

Baohua Sun, Lin Yang, Wenhan Zhang, Michael Lin, Patrick Dong, Charles Young, and Jason Dong. SuperTML: Two-dimensional word embedding for the precognition on structured tabular data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, pages 9269–9278. PMLR, 2020.

Suzanne. CDC data: Nutrition, physical activity, & obesity. `https://www.kaggle.com/spittman1248/cdc-data-nutrition-physical-activity-obesity`, 2018. [Online; accessed 12-September-2021].

Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A Smith, and Yejin Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9275–9293, 2020.

Sarah Tan, Matvey Soloviev, Giles Hooker, and Martin T Wells. Tree space prototypes: Another look at making tree ensembles interpretable. In *Proceedings of the ACM-IMS International Conference on Foundations of Data Science*. ACM, 2020.

Naoyuki Terashita, Hiroki Ohashi, Yuichi Nonaka, and Takashi Kanemaru. Influence estimation for generative adversarial networks. In *Proceedings of the Ninth International Conference on Learning Representations*, 2021.

Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (XAI): Toward medical XAI. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Athanasios Tsanas and Angeliki Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012.

Pınar Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014.

Chih-Kuan Yeh, Joon Sik Kim, Ian EH Yen, and Pradeep Ravikumar. Representer point selection for explaining deep neural networks. In *Proceedings of the Thirty-Second International Conference on Neural Information Processing Systems*, pages 9311–9321, 2018.

I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, 1998.

I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.

Jerrold H Zar. Spearman rank correlation. *Encyclopedia of Biostatistics*, 7, 2005.

Wei Zhang, Ziming Huang, Yada Zhu, Guangnan Ye, Xiaodong Cui, and Fan Zhang. On sample based explanation methods for NLP: Faithfulness, efficiency and semantic evaluation. In *Proceedings of the Fifty-Ninth Annual Meeting of the Association for Computational Linguistics and the Eleventh International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5399–5411, 2021.