UNDERSTANDING AND ADAPTING TREE ENSEMBLES: A TRAINING DATA PERSPECTIVE

by

JONATHAN BROPHY

A DISSERTATION

Presented to the Department of Computer Science
and the Division of Graduate Studies of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

December 2022

DISSERTATION APPROVAL PAGE

Student: Jonathan Brophy

Title: Understanding and Adapting Tree Ensembles: A Training Data Perspective

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer Science by:

Daniel Lowd	Chair
Thanh Nguyen	Core Member
Stephen Fickas	Core Member
Benjamin Hutchinson	Institutional Representative

and

Krista Chronister	Vice Provost for Graduate Studies

Original approval signatures are on file with the University of Oregon Division of Graduate Studies.

Degree awarded December 2022

© 2022 Jonathan Brophy
All rights reserved.

DISSERTATION ABSTRACT

Jonathan Brophy

Doctor of Philosophy

Department of Computer Science

December 2022

Title: Understanding and Adapting Tree Ensembles: A Training Data Perspective

Despite the impressive success of deep-learning models on unstructured data (e.g., images, audio, text), tree-based ensembles such as random forests and gradient-boosted trees are hugely popular and remain the preferred choice for tabular or structured data, and are regularly used to win challenges on data-competition websites such as Kaggle and DrivenData. Despite their impressive predictive performance, tree-based ensembles lack certain characteristics which may limit their further adoption, especially for safety-critical or privacy-sensitive domains such as weather forecasting or predictive medical modeling.

This dissertation investigates the shortcomings currently facing tree-based ensembles—lack of explainable predictions, limited uncertainty estimation, and inefficient adaptability to changes in the training data—and posits that numerous improvements to tree-based ensembles can be made by analyzing the relationships between the training data and the resulting learned model. By studying the effects of one or many training examples on tree-based ensembles, we develop solutions for these models which (1) increase their predictive explainability, (2) provide accurate uncertainty estimates for individual predictions, and (3) efficiently adapt learned models to accurately reflect updated training data.

This dissertation includes previously published coauthored material.

CURRICULUM VITAE

NAME OF AUTHOR: Jonathan Brophy

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR
Cal Poly SLO, San Luis Obispo, CA

DEGREES AWARDED:

Doctor of Philosophy, Computer Science, 2022, UO
Master of Science, Computer and Information Science, 2017, UO
Bachelor of Science, Electrical Engineering, 2014, Cal Poly SLO

AREAS OF SPECIAL INTEREST:

Natural Language Processing
Data Privacy
Interpretability and Explainability
Relational Learning

PROFESSIONAL EXPERIENCE:

Graduate Research Employee, Winter 2016-Winter 2022
Graduate Teaching Employee, Fall 2015, Spring 2022

GRANTS, AWARDS AND HONORS:

Highlighted Reviewer, ICLR 2022
Henry V. Howe Scholarship, University of Oregon, 2016

PUBLICATIONS:

Brophy, J. & Lowd, D. (2022). Instance-Based Uncertainty Estimation for Gradient-Boosted Regression Trees. *International Conference on Neural Information Processing Systems (NeurIPS).*

Brophy, J. & Lowd, D. (2022). Instance-Based Uncertainty Estimation for Gradient-Boosted Regression Trees. *ICML Workshop on Distribution-Free Uncertainty Quantification (DFUQ).*

Xie, Z., **Brophy, J.**, Noack, A., You, W., Asthana, K., Perkins, C., Reis, S., Hammoudeh, Z., Lowd, D. & Singh, S. (2021). What Models Know About Their Attackers: Deriving Attacker Information From Latent Representations. *EMNLP Workshop on Analyzing and Interpreting Neural Networks for NLP (BlackboxNLP).*

Brophy, J. & Lowd, D. (2021). Machine Unlearning for Random Forests. *International Conference on Machine Learning (ICML).*

Brophy, J. & Lowd, D. (2021). DART: Data Addition and Removal Trees. *AAAI Workshop on Privacy-Preserving AI (PPAI).*

Brophy, J. & Lowd, D. (2020). TREX: Tree-Ensemble Representer-Point Explanations. *ICML Workshop on Extending Explainabile AI Beyond Deep Models and Classifiers (XXAI).*

Brophy, J. & Lowd, D. (2020). EGGS: A Flexible Approach to Relational Modeling of Social Network Spam. *AAAI Workshop on Statistical Relational Learning in AI (STARAI).*

Brophy, J. & Lowd, D. (2017). Collective Classification of Social Network Spam. *AAAI Workshop on Artificial Intelligence and Cyber Security (AICS).*

ACKNOWLEDGEMENTS

I would like to express my deep appreciation and gratitude for my advisor, Daniel Lowd, whose unrelenting patience and care for his students makes him truly one of a kind, and an absolute joy to work with. I would also like to thank the other members of my committee—Thanh Nguyen, Stephen Fickas, and Benjamin Hutchinson—for their perspectives and discussions.

I am forever grateful for my parents, whose love and never-ending support fuels my drive to be a better person. I am also indebted to my sister, who has been my life's role model. I also want to thank my partner Kelsey Leib for her patience and support throughout this journey (especially on those long work days!).

Many thanks to Cheri Smith for helping me navigate graduate school from day one, I could not have done this without you. I also want to thank my graduate student peers and lab mates: Jacob Lambert, Irin Mannan, Devkishen Sisodia, Wencong You, Nicole Marsaglia, Adam Noack, Zayd Hammoudeh, and so many more, your friendships have shaped my graduate experience in countless positive ways. Finally, I want to thank Darren Sholes and all my closest friends, your support means the world to me.

This work is funded by the Army Research Office (ARO) grant W911NF-15-1-0265, and by the Defense Advanced Research Projects Agency (DARPA), agreement number HR00112090135. This work also benefited from access to the University of Oregon high performance computer, Talapas.

In memory of my grandparents: James, Muriel, Iwao, and Jennie

TABLE OF CONTENTS

Chapter Page

1. INTRODUCTION 1

1.1. Challenges 1

1.2. Thesis Statement and Dissertation Outline 4

2. BACKGROUND AND RELATED WORK 6

2.1. Tabular Data 6

2.2. Notation 7

2.3. Individual Tree-Based Models 7

2.4. Tree Ensembles 9

2.5. Influence Estimation 12

2.6. Uncertainty Estimation 17

2.7. Machine Unlearning 20

3. IDENTIFYING INFLUENTIAL TRAINING EXAMPLES 26

3.1. Adapting Influence Methods to GBTs 28

3.1.1. LeafRefit: LOO with Fixed Tree Structures 28

3.1.2. LeafInfluence: Adapting Influence Functions 28

3.1.3. BoostIn: Adapting TracIn 32

3.1.4. TREX: Adapting Representer-Point Selection 35

3.1.5. Similarity-Based Influence 37

3.2. Summary of Influence-Estimation Methods 38

3.3. Methodology 39

3.3.1. Datasets and Methods 40

3.3.2. Single Test Instance: Removing Influential Training Examples . 41

Chapter Page

3.3.3. Single Test Instance: Targeted Training-Label Edits 42

3.3.4. Multiple Test Instances: Removing Influential Training Examples 43

3.3.5. Multiple Test Instances: Adding Training-Label Noise 43

3.3.6. Multiple Test Instances: Fixing Mislabelled Training Examples 44

3.4. Results and Analyses 45

3.4.1. Summary of Results 46

3.4.2. Runtime Comparison 47

3.4.3. Correlation Between Influence Methods 49

3.4.4. The Structural Fragility of LOO 50

3.5. Summary 52

4. QUANTIFYING PREDICTION UNCERTAINTY 54

4.1. Instance-Based Uncertainty 55

4.1.1. Identification of High-Affinity Neighbors 55

4.1.2. Modeling the Output Distribution 57

4.1.3. Summary 59

4.1.4. Computational Efficiency 59

4.2. Experiments 62

4.2.1. Implementation and Reproducibility 62

4.2.2. Methodology 63

4.2.3. Probabilistic and Point Predictions 64

4.2.4. Different Base Models 65

4.2.5. Posterior Modeling 66

4.2.6. Variance Calibration 67

4.2.7. Sampling Trees 68

4.3. Summary 70

Chapter Page

5. EFFICIENT MODEL ADAPTATION 71
5.1. DaRE Forests 72
5.1.1. Retraining Minimal Subtrees 73
5.1.2. Sampling Valid Thresholds 73
5.1.3. Random Splits 75
5.1.4. Complexity Analysis 77
5.2. Experiments 78
5.2.1. Datasets 79
5.2.2. Hyperparameter Tuning 79
5.2.3. Methodology 79
5.2.4. Deletion Efficiency Results 80
5.2.5. Effect of d_{rmax} and k on Deletion Efficiency 83
5.2.6. Space Overhead 84
5.3. Summary 86
6. CONCLUSION AND FUTURE DIRECTIONS 87
6.1. Summary of Contributions 87
6.2. Future Directions 88

APPENDICES

A. IDENTIFYING INFLUENTIAL TRAINING EXAMPLES 91
A.1. Implementation and Dataset Details 91
A.2. Experiment Details 97
A.2.1. Summary of Results: All Datasets 97
A.2.2. Removing Examples (Single Test) 98
A.2.3. Targeted Label Edits (Single Test) 100
A.2.4. Removing Examples (Multiple Test) 101

Chapter Page

A.2.5. Adding Noise (Multiple Test) 102

A.2.6. Fixing Mislabelled Examples (Multiple Test) 103

A.2.7. Runtime Comparison 104

A.2.8. Correlation Between Influence Methods 105

A.2.9. The Structural Fragility of LOO 106

A.3. Additional Experiments 107

A.3.1. Predictive Performance of GBDTs 107

B. QUANTIFYING PREDICTION UNCERTAINTY 111

B.1. Implementation and Experiment Details 111

B.1.1. Metrics 111

B.1.2. Datasets 111

B.1.3. Hyperparameters 115

B.1.4. Additional Metrics 118

B.1.5. Runtime 123

B.2. Additional Experiments 125

B.2.1. Probabilistic Performance Without Variance Calibration . . . 125

B.2.2. Comparison to k-Nearest Neighbors 127

B.2.3. Comparison to Bayesian Additive Regression Trees 129

B.2.4. Different Tree-Sampling Strategies 131

B.2.5. Leaf Density 133

C. EFFICIENT MODEL ADAPTATION 135

C.1. Algorithmic Details 135

C.1.1. Exact Deletion: Proof of Theorem 5.1.1 135

C.1.2. Training Complexity: Proof of Theorem 5.1.2 138

C.1.3. Training Complexity: Proof of Theorem 5.1.3 138

Chapter Page

C.1.4. Space Complexity: Proof of Theorem 5.1.4 139

C.1.5. Complexity of Slightly-Less-Naive Retraining 140

C.1.6. Node Statistics 141

C.1.7. Batch Deletion 142

C.1.8. Pseudocode 143

C.2. Implementation and Experiment Details 144

C.2.1. Datasets 144

C.2.2. Predictive Performance of DaRE Forests 147

C.2.3. Effect of d_{rmax} on Deletion Efficiency 151

C.2.4. Effect of k on Deletion Efficiency 152

REFERENCES CITED 153

LIST OF FIGURES

Figure		Page
1.	Example decision tree (1a) and resulting dataset partitioning (1b) for a toy binary classification dataset in which the aim is to predict whether or not one should eat at a particular pizza place given the rating and price.	8
2.	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 3); 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.	45
3.	*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.	48
4.	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.	49
5.	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.2.9.	51

Figure		Page

6. *Left*: Distribution of training-example affinities to a randomly selected test example z_{te} 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. 52

7. IBUG workflow. Given a GBRT model and an input instance x, IBUG collects the training examples at each leaf x traverses to, keeps the k most frequent examples, and then uses those examples to model the output distribution. 56

8. *Left*: Distribution of the k-nearest training instances for 5 randomly-selected test instances from the MEPS (top) and Wine (bottom) datasets. *Right*: Test NLL (with standard error) when modeling the posterior using two different distributions (lower is better). IBUG can model parametric *and* non-parametric distributions that better fit the underlying data than assuming normality. 67

9. Runtime comparison. *Left*: Total train time (including tuning). *Right*: Average prediction time per test example. Results are shown for all datasets, averaged over 10 folds (exact values are in §B.1.5, Tables B.9 and B.10). On average, IBUG has comparable training times to PGBM and CBU, but is relatively slow for prediction. 68

10. Change in probabilistic (NLL) performance (top) and average prediction time (in seconds) per test example (bottom) as a function of τ for six datasets with trees sampled *first-to-last*; lower is better. NGBoost, PGBM, and CBU are added for additional context. The shaded regions represent the standard error. Overall, average prediction time decreases significantly as τ decreases while test NLL often remains relatively stable, enabling IBUG to generate probabilistic predictions with significant increased efficiency. 69

11. Deletion efficiency of DaRE RF. *Top & Middle*: Number of instances deleted in the time it takes the naive retraining approach to delete one instance using the random and worst-of-1000 adversaries, respectively (error bars represent standard deviation). *Bottom*: The increase in test error when using R-DaRE RF relative to the predictive performance of G-DaRE RF (error bars represent standard error). 81

Figure Page

12. Effect of increasing d_{rmax}on deletion efficiency (left), predictive performance (middle), and the cost of retraining (right) using the random (top) and worst-of-1000 (bottom) adversaries for the Bank Marketing dataset. The predictive performance is independent of the adversary, as performance is measured before any deletions occur. Error bars represent standard deviation and standard error for the left and middle plots, respectively. In short, we see that increasing d_{rmax} increases deletion efficiency but initially gradually degrades predictive performance. Similar analysis for other datasets are in the Appendix: §C.2.3. 82

13. Effect of increasing k on predictive performance (left) and deletion efficiency (right) for the Surgical dataset using the random adversary; d_{rmax} is held fixed at 0. Error bars represent standard error and standard deviation for the left and right plots, respectively. Analysis for other datasets is in the Appendix: §C.2.4. 84

A.14. High-level overview of results including *all* data sets; thus, LeafRefit and LeafInfluence are not included in this analysis. 97

A.15. 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. 98

A.16. Change in loss for a random test example (averaged over 100 runs) as training examples are removed. Higher is better. 99

A.17. 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. 100

A.18. 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. 101

Figure Page

A.19. 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. 102

A.20. 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. 103

A.21. Spearman correlation coefficient between influence methods for each GBDT type, averaged over 100 test examples and SDS data sets. 105

A.22. Spearman correlation coefficient between influence methods averaged over 100 test examples, all GBDT types, and either classification or regression data sets. 105

A.23. Change in loss (averaegd over 100 test instances) as training examples are removed *one at a time*. Higher is better. The gray box shows the spike in loss after removing a single training example identified by LOO. 106

B.1. Probabilistic (NLL) performance (lower is better) and average prediction time (in milliseconds) per test example (lower is better) as a function of τ for different sampling techniques. *Top*: sample trees uniformly at random, *middle*: sample trees first-to-last (in terms of boosting iteration), *bottom*: sample trees last-to-first. All methods result in similar prediction times; however, *first-to-last* sampling typically provides the best NLL with the fewest number of trees sampled. 132

B.2. Average % train visited / tree during affinity computation for each dataset. Results are averaged over test set w/ s.d; lower is better. In general, the number of training instances visited per tree is highly dependent on the dataset; and for some datasets, is also highly dependent on the test example (points with large standard deviations). 133

Figure Page

B.3. Average % train visited at each iteration during affinity computation for different datasets. Results show averages over test set w/ s.d.; lower is better. Overall, leaf densities are dataset dependent. However, for LightGBM and XGBoost, weak learners later in training tend to pool a larger proportion of training instances into fewer leaves; in contrast, CatBoost has less dense leaves and training instances are more equally distributed among the leaves in each tree. 134

C.1. Effect of d_{rmax} on deletion efficiency. 151

C.2. Effect of k on deletion efficiency. 152

LIST OF TABLES

Table Page

1. Summary of influence-estimation methods. 39

2. Probabilistic (CRPS) performance for each method on each dataset. Lower is better. Normal distributions are used for all probabilistic predictions. Results are averaged over 10 folds, and standard errors are shown in subscripted parentheses. The best method for each dataset is bolded, as well as those with standard errors that overlap the best method. *Bottom row*: Head-to-head comparison between IBUG/IBUG+CBU and each method showing the number of wins, ties, and losses (W-T-L) across all datasets. On average, IBUG+CBU provides the most accurate probabilistic predictions. . . . 64

3. Probabilistic (CRPS, NLL) performance on the test set for IBUG using different base models. Results are averaged over 10 folds, and standard errors are shown in subscripted parentheses; lower is better. On 6 and 5 datasets, respectively, either IBUG-LightGBM or IBUG-XGBoost significantly outperforms IBUG-CatBoost on the validation set and subsequently on the test set, demonstrating the potential for improved probabilistic performance by using IBUG with different base models. 66

4. Probabilistic performance comparison of each method with vs. without variance calibration. In all cases, calibration maintains or improves performance; it is especially helpful for CBU. 68

5. Dataset Summary. n = no. instances, p = no. attributes, % Positive = positive label percentage, Metric = predictive performance metric. . . 80

6. Summary of the deletion efficiency results. Specifically, the minimum, maximum, and geometric mean of the speedup vs. the naive retraining method across all datasets. 83

Table Page

7. Memory usage (in megabytes) for the training data, G-DARE RF, and an SKLearn RF (SKRF) trained using the same values of T and d_{max} as G-DARE RF. The total memory usage for the G-DARE RF model is broken down into: 1) the structure of the model needed for making predictions (Structure); 2) the additional statistics stored at all decision nodes (Decisions); and 3) the additional statistics and training-instance pointers stored at all leaf node (Leaves). The space overhead for G-DARE RF to enable efficient data deletion is measured as a ratio of the total memory usage of (data + G-DARE RF) to (data + SKRF). Results are averaged over five runs and the standard error is shown in parentheses. 85

A.8. Dataset 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). 92

A.9. 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. 104

A.10. 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 A.8 for reference. 107

A.11. 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 A.8 for reference. 109

B.1. Dataset summary after preprocessing. 115

B.2. Hyperparameters selected most often over 10 folds for each dataset when optimizing CRPS. 116

B.3. Hyperparameters selected most often over 10 folds for each dataset when optimizing NLL. 117

Table Page

B.4. Point (RMSE ↓) performance for each method on each dataset. *Bottom row*: Head-to-head wins-ties-losses. 118

B.5. Probabilistic (NLL ↓) performance for each method on each dataset. *Bottom row*: Head-to-head wins-ties-losses. 119

B.6. Probabilistic (check score a.k.a. “pinball loss” ↓) performance. *Bottom row*: Head-to-head wins-ties-losses. 120

B.7. Probabilistic (interval score ↓) performance. *Bottom row*: Head-to-head wins-ties-losses. 121

B.8. Probabilistic (MACE ↓ / sharpness ↓) performance. Standard errors are omitted for brevity. 122

B.9. Total train (including tuning) time (in seconds). 123

B.10. Average prediction time per text example (in milliseconds). 124

B.11. Probabilistic (CRPS ↓) performance *without* variance calibration. . . . 125

B.12. Probabilistic (NLL ↓) performance *without* variance calibration. 126

B.13. Probabilistic (CRPS) performance comparison of IBUG against two different nearest-neighbor models. kNN estimates the conditional mean and variance using two different k values; and kNN-CB estimates the variance in the same way as kNN, but uses the scalar output from the CatBoost model to estimate the conditional mean. Overall, these results suggest affinity is a better measure of similarity than Euclidean distance for uncertainty estimation in GBRTs. 128

B.14. Probabilistic (CRPS ↓) performance comparison between IBUG and BART. 129

B.15. Point (RMSE ↓) performance comparison between IBUG and BART. . 130

C.1. Dataset summary including the main predictive performance metric used for each dataset, either average precision (AP) for datasets whose positive label percentage $<$ 1%, AUC for datasets between [1%, 20%], or accuracy (Acc.) for all remaining datasets. 147

Table Page

C.2. Predictive performance comparison of G-DaRE RF to: an extremely randomized trees model (RT) [79], an Extra Trees [79] model (ET), and a popular and widely used random forest implementation from Scikit-Learn (SKLearn) with (*) and without bootstrapping. The numbers in each cell represent either average precision, AUC, or accuracy as specified by Table C.1; results are averaged over five runs and the standard error is shown in subscripted parentheses. . . . 148

C.3. Hyperparameters selected for the G-DaRE and R-DaRE (using error tolerances of 0.1%, 0.25%, 0.5%, and 1.0%) models. 150

C.4. Training times (in seconds) for the G-DaRE model using the hyperparameters selected in Table C.3. Mean and standard deviations (S.D.) are computed over five runs. 150

CHAPTER 1

INTRODUCTION

Despite the impressive success of deep-learning models on unstructured data (e.g., images, audio, text) [29, 164], tree-based ensemble models such as random forests [25] and gradient-boosted trees [73] remain the preferred choice for *tabular* or *structured* data [88, 159, 191], even with the recent interest and advancements of deep learning on tabular data [9, 102, 106, 112, 158]. Tree ensembles are regularly used to win challenges on data-competition websites such as Kaggle and DrivenData [21, 72], and Kaggle CEO Anthony Goldbloom recently described gradient-boosted trees as the most "glaring difference" between what is used on Kaggle and what is "fashionable in academia" [110]. Despite the long-lasting success of tree-based models, particularly on classification and regression tasks, significant limitations exist which may prevent their further adoption, especially for certain applications such as those in privacy-sensitive and safety-critical domains.

1.1 Challenges

Traditionally, machine-learning (ML) models are evaluated on a set of held-out never-before-seen test data, in which accuracy and predictive performance is prioritized when selecting which model to use. This is largely still true today, however, the focus on ML models has expanded from analyzing overall model performance to analyzing *individual* predictions. This sentiment is reinforced by the U.K.'s release of the General Data Protection Regulation (GDPR) [64], in which the "Right to an Explanation" clause allows users to request an explanation for any automated decisions that may significantly impact their lives. For example, an ML

model trained to predict whether or not to give a bank loan to an individual would fall under this category.

Tree-based ensembles tend to achieve high predictive performance, but unfortunately lack explainability of their decisions. However, explainable artificial intelligence (XAI) is a a new research subfield [68, 207] dedicated to making ML models (including tree-based ensembles) more explainable. A popular and widely-used method for explaining ML predictions is to quantify which *features* are most important for a *given* prediction [137, 169]. However, this approach is not always adequately sufficient, which leads us to a more recent and complimentary methodology.

Challenge #1: Can we identify the *training examples* most responsible for a given prediction? This direction is not only complimentary to the features-based approach (in which a combination of methods may best explain an individual prediction), but may also be used to identify outliers, incorrectly-labeled, poisoned [131, 196], or noisy training data which can degrade the overall performance of the model. Naive approaches to this problem are generally intractable [80]. Thus, we develop new techniques of influential-example identification for discrete tree-based ensembles by adapting efficient solutions from continuous deep learning models. Later in Chapter 3, we introduce an adapted influence estimation method capable of effectively identifying the most influential training examples for a given target example prediction from a tree-based ensemble.

As previously mentioned, tree-based models perform particularly well for tabular regression and classification tasks. However, in contrast to models built for classification, tree-based models built for regression only produce a scalar value for the output, and provide no *uncertainty* about the prediction. Uncertainty

estimates are particularly important for certain domain applications such as financial forecasting [2], weather prediction [83], and medical modeling [10]. For example, a clinic-mortality model that predicts a patient will live 5 ± 5.5 years post-operation is more informative than a model that outputs only a single number. This additional information signifies a large degree of uncertainty behind the prediction, adds explainability to the model, and ultimately helps recipients of the automated decision (e.g., doctors, patients) make more informed decisions.

Challenge #2: How can we accurately quantify the *uncertainty* of a given prediction? To tackle this problem, we take an *instance-based* approach, leveraging the structure of a given learned tree ensemble to identify the training examples "most similar" to a given target example. We then use this subset of training examples to model the conditional output probability distribution. We find this simple approach can accurately and flexibly model the posterior for a given target example (Chapter 4).

In addition to the "Right to an Explanation", the GDPR also contains a "Right to be Forgotten" [64] clause stating that users can request their data be deleted from companies using their data, upon request. It is increasingly likely that this clause will require companies to not only remove personal data from their databases, but also retrain any models previously trained with that data. To further illustrate these privacy implications, *membership-inference* [190] and *model-inversion* [215] attacks provide evidence that the learned representation of a model after training is in some sense a transformed version of the data they are trained on, enabling adversaries to reverse-engineer the model to accurately infer what examples were used to train the model.

Removing examples from an ML model can be done by simply deleting the unwanted examples from the training data and retraining the model from scratch. However, this naive approach is often very costly, especially as the size of the dataset, complexity of the model, or number of deletion requests increases. For example, it is not uncommon for a single modern ML models to take hours, days, weeks, etc. to train [29].

Challenge #3: Can we efficiently adapt a learned model to accurately reflect updated training data? We address this challenge in the context of random forests, in which we make deletions efficient by carefully storing a minimal set of statistics needed to rebuild the parts of the model that need retraining in response to a given deletion request. We introduce this approach later in Chapter 5 and demonstrate its ability to delete data *orders of magnitude* faster than retraining from scratch while sacrificing *little to no* predictive power.

1.2 Thesis Statement and Dissertation Outline

We combine the challenges tree-based ensembles face into one coherent sentiment. **Can we better understand, improve, and efficiently adapt tree-based models by studying the relationship between the model and the data they are trained on?** In addressing these challenges, we find that indeed tree ensembles can be more explainable, accurate, and adaptable by analyzing the relationships and effects between learned models and their training data.

The outline for the rest of this dissertation is as follows. We provide all necessary background information in Chapter 2, address the main shortcomings of tree-based ensembles (posed in §1.1) in Chapters 3–5, and finally summarize our work and provide valuable future research directions in Chapter 6 that may enable

the continued success of tree-based ensembles and further their adoption to an even wider range of applications and domains.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we first describe the type of data well-suited for tree-based methods, the notation we use throughout this dissertation, and formal descriptions of the different tree-based models in widespread use today. Then, we provide detailed background and related work regarding influence estimation, uncertainty estimation, and machine unlearning, three overlapping topics outlined in Chapter 1 whose advancements have the potential to significantly increase the understandability, predictive capacity, and adaptability of tree ensemble methods. The problems and related work in this chapter serve as the foundation of this dissertation.

2.1 Tabular Data

In this work, we focus on *tabular* data—data that can be represented as a table—which is one of the most common types of data used in real-world ML applications [16, 44, 199]. Tabular data is also referred to as "structured" since the features are often inherently meaningful. Contrast tabular data with "unstructured" data such as audio, text, or video, in which deep learning methods are often successfully applied since they can automatically extract meaningful features from the low-level input (e.g., pixels, characters, etc.).

Tabular data is also often *heterogeneous*—data which contains significantly different types of features—for example, a medical dataset may contain one attribute about a patient's blood pressure and another attribute about the patient's age. These two attributes are on completely different scales; fortunately, tree-based models naturally handle heterogeneous data without needing to transform all attributes to be on the same scale.

2.2 Notation

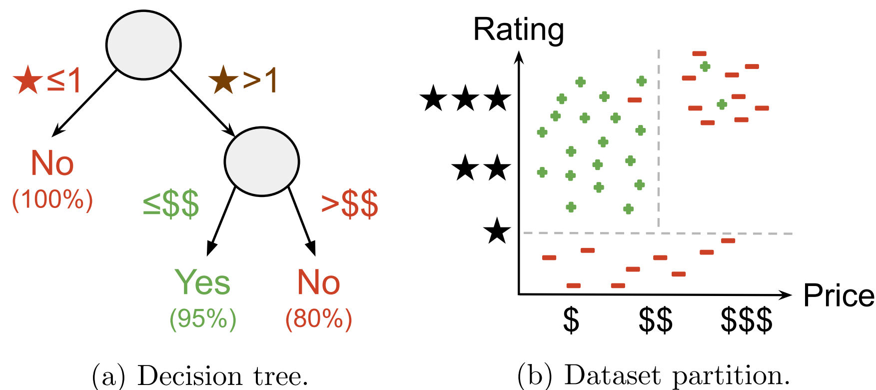
Formally, we assume an instance space $\mathcal{X} \subseteq \mathbb{R}^p$ and possible targets $\mathcal{Y} \subset \{\{-1,+1\},\mathbb{Z},\mathbb{R}\}$. Let $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^n$ be a training dataset in which each instance $x_i \in \mathcal{X}$ is a p-dimensional vector $(x_{i,j})_{j=1}^p$ and $y_i \in \mathcal{Y}$. We refer to $P = \{j\}_{j=1}^p$ as the set of possible attributes. We use $\mathbf{x} := \{x_i\}_{i=1}^n$, $\mathbf{y} := \{y_i\}_{i=1}^n$, $z_i := (x_i, y_i)$ to denote the ith training instance, and $z_{te} := (x_e, y_e)$ to denote a target instance.

We also define a (possibly randomized) *learning algorithm* $\mathcal{A} : \mathcal{D} \to \mathcal{F}$ as a function from a data set $\mathcal{D}$ to a model in hypothesis space $\mathcal{F}$, and a *loss function* $\ell : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$. 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. Next, we formally describe tree-based models, which are known for excelling at problems represented as tabular data.

2.3 Individual Tree-Based Models

Tree-based models generally come in two forms: as a single tree, or as an ensemble of trees. Since tree ensembles are made up of individual trees, we review single tree-based models first.

Decision Tree. A *decision tree* is a tree-structured model in which each leaf is associated with a prediction value equal to the mean of the data labels assigned to that leaf, and each internal node is a decision node associated with an attribute $a \in P$ and threshold value $v \in \mathbb{R}$. The outgoing branches of the decision node partition the data based on the chosen attribute and threshold. Given a target example $x_{te} \in \mathcal{X}$, the prediction of a decision tree can be found by traversing the tree, starting at the root and following the branches consistent with the attribute



(a) Decision tree. (b) Dataset partition.

Figure 1. Example decision tree (1a) and resulting dataset partitioning (1b) for a toy binary classification dataset in which the aim is to predict whether or not one should eat at a particular pizza place given the rating and price.

values in x. Traversal ends at one of the leaf nodes, where the prediction is equal to the value of the leaf node.

Decision trees are typically learned in a recursive manner, beginning by picking an attribute a and threshold v at the root that optimizes an empirical split criterion such as the Gini index [26]:

$$G_{D,\mathcal{Y}}(a,v) = \sum_{b \in \{\ell,r\}} \frac{|D_b|}{|D|} \left(1 - \sum_{y \in \mathcal{Y}} \left(\frac{|D_{b,y}|}{|D_b|}\right)^2\right) \tag{2.1}$$

or entropy [161]:

$$H_{D,\mathcal{Y}}(a,v) = \sum_{b \in \{\ell,r\}} \frac{|D_b|}{|D|} \left(\sum_{y \in \mathcal{Y}} -\frac{|D_{b,y}|}{|D_b|} \log_2 \frac{|D_{b,y}|}{|D_b|}\right), \tag{2.2}$$

in which $D \subseteq \mathcal{D}$ is the input dataset to a decision node, $D_\ell = \{(x_i, y_i) \in D \mid x_{i,a} \leq v\}$, $D_r = D \setminus D_\ell$, $D_{\ell,y} = \{(x_i, y_i) \in D_\ell \mid y_i = y\}$, and $D_{r,y} = \{(x_i, y_i) \in D_r \mid y_i = y\}$. Once a and v have been chosen for the root node, the data is partitioned into mutually exclusive subsets based on the value of v, and a child node is learned for each data subset. The process terminates when the entire subset has the same label or the tree reaches a specified maximum depth $d_{\max}$. Figure 1 shows an example of a learned decision tree and its partitioning of the dataset.

Regression Tree. A *regression tree* is similarly structured to a decision tree, however the training labels used to build a regression tree are real-valued, i.e., $\mathcal{Y} \in \mathbb{R}$. As a result, a decision node in a regression tree chooses the split which most reduces the total variance of the dataset D at that node. More formally, the variance reduction for attribute a and threshold v is:

$$V_D(a, v) = var(y) - \sum_{b \in \{\ell, r\}} \frac{|D_b|}{|D|} \, var(y_b), \tag{2.3}$$

in which y and y_b are the vectors of labels for D and D_b, respectively, and var is the variance function. The leaf value in a regression tree is simply the mean output label of the training instances assigned to that leaf. Finally, the prediction of a regression tree is performed in the same manner as a decision tree.

2.4 Tree Ensembles

Although simple and arguably very interpretable, shallow single decision/regression-tree models (i.e., an individual tree grown to a small maximum depth) tend to have low representational power, while deeper trees suffer from high variance in their predictions. To achieve a more balanced bias-variance trade-off, tree ensembles combine multiple individual trees (typically built with a shallow maximum depth)—called *weak learners*—to form a *strong learner*, which tends to perform extremely well in practice [21, 72]. Tree ensembles generally come in two flavors: random forests and gradient-boosted trees.

Random Forest. A *random forest* (RF) is an ensemble of decision/regression trees which predicts the mean output value from its component trees. Two sources of randomness are used to increase diversity among the trees. First, each tree in the ensemble is trained from a bootstrap sample of the original training data, with some instances excluded and some included multiple times. Second, each

decision node is restricted to a random subset of attributes, and the split criterion is optimized over this subset rather than over all attributes. Training is easily parallelizable for RFs since each tree is trained independent of all other trees in the ensemble.

Gradient-Boosted Trees. Gradient boosting [73, 75] is a powerful machine-learning algorithm that iteratively adds weak learners to construct a model $f : \mathcal{X} \to \mathbb{R}$ that minimizes some empirical risk $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. The model is defined by a recursive relationship:

$$f_0(x) = \gamma, \tag{2.4}$$

$$\vdots$$

$$f_t(x) = f_{t-1}(x) + \eta \; m_t(x), \tag{2.5}$$

in which f_0 is the base learner, γ is an initial estimate, f_t is the model at iteration t, m_t is the weak learner added during iteration t to improve the model, and η is the learning rate. Gradient-boosted decision trees (GBDTs) choose ℓ to be logistic loss and γ to be the logit of the positive class (for binary classification[1]); gradient-boosted regression trees (GBRTs) set ℓ to be the mean squared error (MSE) and γ as $\frac{1}{n}\sum_{i=1}^{n} y_i$ (mean output of the training instances). Both GBDTs and GBRTs use *regression trees* as weak learners, and each weak learner is typically chosen to approximate the negative gradient [143]:

$$m_t = \arg\min_{\hat{m}} \; \frac{1}{n} \sum_{i=1}^{n} (-g_t(x_i, y_i) - \hat{m}(x_i))^2, \tag{2.6}$$

in which $g_t(x_i, y_i) = \frac{\partial \ell(y_i, \hat{y}_i)}{\partial \hat{y}_i}\Big|_{\hat{y}_i = f_{t-1}(x)}$ is the functional gradient of the ith training instance at iteration t w.r.t. $\hat{y}_i = f_{t-1}(x_i)$.

[1] For multiclass classification, GBDTs use the log of the class priors distribution.

The weak learner at iteration t partitions the instance space into a set of M_t disjoint regions $R_t = \cup_{l=1}^{M_t} r_{t,l}$. Each region is a leaf whose parameter value $\theta_{t,l}$ is typically determined (given a fixed structure) using a one-step Newton-estimation method [39, 118]:

$$\theta_{t,l} = -\frac{\sum_{i \in I_{t,l}} g_t(x_i, y_i)}{\sum_{i \in I_{t,l}} h_t(x_i, y_i) + \lambda}, \tag{2.7}$$

in which $I_{t,l} = \{z_i \mid z_i \in r_{t,l}\}$ is the instance set of leaf l for tree t, $h_t(x_i, y_i) = \frac{\partial^2 \ell(y, \hat{y})}{\partial \hat{y}^2}\Big|_{\hat{y}=f_{t-1}(x)}$ is the second derivative of the ith training instance w.r.t. $\hat{y}_i$, and λ is a regularization constant. Thus, the output of m_t can be written as

$$m_t(x_{te}) = \sum_{l=1}^{M_t} \theta_{t,l} \mathbb{1}[x_{te} \in r_{t,l}], \tag{2.8}$$

in which $\mathbb{1}$ is the indicator function. The final model $f = f_T$ generates a prediction by summing the values for the leaves x_{te} is assigned to across all T trees, and applying an activation function υ to the output: $\hat{y}_{te} = \upsilon\left(\sum_{t=1}^{T} m_t(x_{te})\right)$. For GBDTs, υ is the sigmoid (softmax) function for binary (multiclass) classification; for GBRTs, υ is the identity function.

Gradient-boosted trees (GBTs) tend to have better predictive performance than RFs in practice. However, RFs are arguably more interpretable than GBTs, and GBTs must be trained sequentially since each tree in the ensemble is trained using the residual error from the model at the previous iteration. Despite their predictive prowess, GBTs are black-box models with opaque decision-making processes [137, 138]. *Influence estimation* may help us better understand how GBTs make predictions, remove unwanted biases, and ultimately improve our models.

2.5 Influence Estimation

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 [3, 68, 207] or improve model quality via data set or model debugging [221]. Influence estimation may also be a useful tool in accurately assessing whether a model is fulfilling its end of a *narrowly-specified contract*, and help determine if the trust/distrust in that contract is *warranted*; the resulting analysis may then be used as a stepping stone towards building trustworthy models and ultimately fostering Human-AI trust [109]. Additional applications include: identifying domain adaptation/domain shift [80], data valuation [111], data set poisoning, identifying memorized examples [66], and many more [125].

Existing influence estimation methods attempt to compute the influence of each training example z_i on the prediction of a given target example z_{te}. Informally, a training example is *influential* if its inclusion in the training data impacts the learned model and its predictions. Following previous work [125, 160], we analyze the influence of training examples on z_{te} by computing their impact on the *loss* of z_{te}.[2] To this end, we define an *influence-estimation method*, $\mathcal{A}(\mathcal{D}) \times \mathcal{D} \times \mathcal{L} \times (\mathcal{X} \times \mathcal{Y}) \rightarrow \mathbb{R}^n$, as a function from a model $\mathcal{A}(\mathcal{D})$, data set $\mathcal{D}$, loss function ℓ, and target example z_{te} to a vector of *influence values*, one for each training example. Note these influence values can be either positive or negative, indicating they *reduce* or *increase* the loss of the target example, respectively.[3]

[2]Target example z_{te} can be either in the train or test set.

[3]Following convention [160], we refer to training examples that reduce the loss of the target example as *proponents* and those that increase loss as *opponents*.

Leave-One-Out Retraining. The simplest and most intuitive approach to estimating the influence of a training example z_i on a target example z_{te} is to ignore the existing model and simply rerun $\mathcal{A}$ on an updated data set without z_i, and then measure the change in loss[4] on z_{te}:

$$\mathcal{I}_{LOO}(z_i, z_{te}) = \ell(y_{te}, \mathcal{A}(D \setminus z_i)(x_{te})) - \ell(y_{te}, \mathcal{A}(D)(x_{te})). \tag{2.9}$$

Repeating this naive approach for all training examples is known as *leave-one-out* retraining (LOO). 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 [80].

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 amongst involved players [183]. Data Shapley [80] is a model-agnostic approach that applies the idea of Shapley values to influence estimation in which the marginal contribution of z_i on the loss of z_{te} can be written as:

$$\mathcal{I}_{DShap}(z_i, z_{te}) = C \sum_{S \subseteq D \setminus z_i} \frac{\ell(y_{te}, \mathcal{A}(S)(x_{te})) - \ell(y_{te}, \mathcal{A}(S \cup \{z_i\})(x_{te}))}{\binom{n-1}{|S|}}, \tag{2.10}$$

where C is a constant and S represents all possible subsets of the training data without z_i. Equation (2.10) computes the *expected* marginal impact of a single example given a subset of the training data, but is also far more intractable than LOO.

[4]When $\mathcal{A}$ is non-deterministic, one would need to retrain *multiple* times on the same data set to compute the *expected* change in loss.

Tractable Approximation of Expected Marginal Influence. Recently, Feldman and Zhang [66] proposed a method that quantifies the amount of memorization acquired by a deep learning model during training. Their approach computes the memorization level of a training example as well as the influence of a training example on the accuracy of a given learning algorithm $\mathcal{A}$ by training τ different models on random subsets of the data, computing the expected marginal-influence effects for different training examples. We apply this approach as a tractable approximation of the computationally infeasible Data Shapley method, defining the expected marginal-influence effect of z_i on the loss of z_{te} as follows:

$$\mathcal{I}_{Sub}(z_i, z_{te}) = \mathbb{E}_{S \sim P(\mathcal{D},m)}[\ell(y_{te}, \mathcal{A}(S)(x_{te}))|z_i \in S] - \mathbb{E}_{S \sim P(\mathcal{D},m)}[\ell(y_{te}, \mathcal{A}(S)(x_{te}))|z_i \notin S]. \tag{2.11}$$

In Equation (2.11), $P(\mathcal{D}, m)$ represents the uniform distribution over all subsets of $\mathcal{D}$ with size $m < n$. 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. We refer to this approach as *SubSample* and note it is much more tractable than Data Shapley (Eq. 2.10) and in most cases LOO (Eq. 2.9), especially when $\tau \ll n$.

Model-Agnostic vs. Model-Specific. Leave-one-out (LOO) retraining defines influence as the difference between training with the entire training data set and training with the 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.

Both LOO and expected marginal influence work for any model, that is, they are model-agnostic approaches. However, model-specific methods can often be more effective in identifying influential training examples. The most well-known model-specific approach is *Influence functions* [125], one of the first methods proposed for influence estimation, first in differentiable models [48], then in deep neural networks [125], and later in trees [185].

Static vs. Dynamic Influence Estimation. Influence functions [125] approximate the influence of training examples using the *final* learned model, and thus is classified as a *static* influence estimation technique. Additionally, representer point methods [221] offer greater efficiency and an interpretation of the model as a sum over contributions from all training points, based on representer theorems. However, more recent approaches such as TracIn [160], HyDRA [41], and SGDCleanse [99] estimate the influence of training examples *throughout* training, providing a more accurate estimation of the total effect of a specific training example on the resulting learned model and subsequently on the prediction of a target example.

Adapting Influence Estimation. Previous work has extended influence functions or TracIn to other types of models, including variational autoencoders [128], NLP models with transformer architectures [228], and generative adversarial networks (GANs) [206]. However, tree ensembles have not yet benefited from the advantages of dynamic influence estimation methods, which show great promise in deep neural networks. Later in Chapter 3, we describe and

adapt recent influence estimation techniques designed for deep learning models to GBTs, demonstrate their ability to identify influential training examples in GBTs, and evaluate the tradeoffs between our adapted methods and model-agnostic approaches such as LOO.

Prototypes, Criticisms, and Dataset Maps. Tangentially-related to influence estimation, *prototypes* (and their complement: *criticisms*), attempt to summarize a data set by identifying training examples in high- and low-density regions of the input space [19, 121, 93]. Although typically model-agnostic, model-specific versions exist such as TreeProto [204]. 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 thus far, 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 [4, 202] 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.

Feature-Based Influence Estimation. Another highly related but significantly different body of research is *feature-based* influence estimation. There is a plethora of work in this area [81, 126, 138, 169, 179, 200] which estimates the loss of z_{te} 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 [12, 28, 96]. 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.

In addition to identifying the most influential training examples for a target prediction, accurately quantifying its predictive uncertainty provides another layer of useful information about that prediction and the data used to train the model.

2.6 Uncertainty Estimation

Classification tasks comprise a large fraction of important problems in supervised machine learning, and GBDTs inherently provide a level of confidence for each prediction by applying the sigmoid or softmax function to the aggregated output from all the trees in the ensemble. However, regression tasks represent an equally large and important subclass of problems which can range widely from financial [2] and retail-product forecasting [142] to weather [83, 84] and clinic-mortality prediction [10]. GBRTs are known to make accurate *point predictions* [141] but provide no estimate of the *prediction uncertainty*, which is desirable for both forecasting practitioners [22, 205] and the explainable AI (XAI) community [3, 68, 207] in general.

Probabilistic Regression. Our focus is on *probabilistic regression*—estimating the conditional probability distribution $P(y|x)$ for some target variable y given some input vector $x \in \mathcal{X}$. Unfortunately, traditional GBRT models only output scalar values. Under a squared-error loss function, these scalar values can be interpreted as the conditional mean in a Gaussian distribution with some (unknown) constant variance. However, homoscedasticity is a strong assumption and unknown constant variance has little value in a probabilistic prediction; thus, in order to allow heteroscedasticity, the predicted distribution

needs at least two parameters to convey both the magnitude and uncertainty of the prediction [61].

Natural Gradient Boosting. Natural Gradient Boosting (NGBoost) is a recent method by Duan et al. [61] that tackles the aforementioned problems by estimating the parameters of a desired distribution using a multi-parameter boosting approach that trains a separate ensemble for each parameter of the distribution. NGBoost employs the natural gradient to be invariant to parameterization, but requires the inversion of many small matrices (each the size of the number of desired parameters) to do so. Empirically, NGBoost generates state-of-the-art probabilistic predictions, but tends to underperform as a point predictor.

Probabilistic Gradient Boosting Machines. More recently, Sprangers et al. [195] introduced Probabilistic Gradient Boosting Machines (PGBM), a single model that optimizes for point performance, but can also generate accurate probabilistic predictions. PGBM treats leaf values as stochastic random variables, using sample statistics to model the mean and variance of each leaf value. PGBM estimates the output mean and variance of a target example using the estimated parameters of each leaf it is assigned to. The predicted mean and variance is then used as parameters in a specified distribution to generate a probabilistic prediction. PGBM has been shown to produce state-of-the-art probabilistic predictions; however, computing the necessary leaf statistics during training can be computationally expensive, especially as the number of leaves in the ensemble increases. Also, since only the mean and variance are predicted for a given test example, PGBM is limited to distributions using only location and scale to model the output.

CatBoost with Uncertainty. Finally, Malinin et al. [143] introduce CatBoost with uncertainty (CBU), a method that estimates uncertainty using ensembles of GBRT models. Similar to NGBoost, multiple ensembles are learned to output the mean and variance. However, CBU also constructs a *virtual ensemble*—a set of overlapping partitions of the learned GBRT trees—to estimate the uncertainty of a prediction by taking the mean of the variances output from the virtual ensemble. Their approach uses a recently proposed stochastic gradient Langevin boosting algorithm [212] to sample from the true posterior via the virtual ensemble (in the limit); however, their formulation of uncertainty is limited only to the first and second moments, similar to PGBM.

Instance-Based Uncertainty Estimation. Later in Chapter 4, we introduce a simple method that uses the closest training examples to a target example to model the prediction uncertainty of that example, where distance is measured using the structure of the tree ensemble. This method performs well on both point and probabilistic performance, can flexibly model the output, and can be applied to *any* GBRT model.

Additional Related Work. Traditional approaches to probabilistic regression include generalized additive models for location, scale, and shape (GAMLSS), which allow for a flexible choice of distribution for the target variable but are restricted to a pre-specified model form [170]. Prophet [205] also produces probabilistic estimates for generalized additive models, but has been shown to underperform as compared to more recent approaches [6, 180]. Bayesian methods [89, 150] naturally generate uncertainty estimates by integrating over the posterior; but, exact solutions are limited to simple models, and more complex

models such as Bayesian Additive Regression Trees (BART) [42, 139] require computationally expensive sampling techniques (e.g., MCMC [7]) to provide approximate solutions.

Other approaches to probabilistic regression tasks include conformal predictions [8, 181, 203] which produce confidence intervals via empirical errors obtained in the past, and quantile regression [100, 124, 144, 171]. Similar to PGBM, distributional forests (DFs) [176] estimate distributional parameters in each leaf, and average these estimates over all trees in the forest. However, DFs are variants of random forests (RFs), and GBRTs are known to regularly outperform RFs on regression problems, making this approach less suitable for high-performance tasks. Deep learning approaches for probabilistic regression [6, 166, 219] have also increased recently, with notable approaches such as DeepAR [173] and methods based on transformer architectures [132, 133].

The tools discussed thus far can often help identify noisy, mislabelled, or poisoned training examples with less human effort [96], it may then be desirable to remove these examples from the model to improve performance. However, this can be an expensive operation depending on the size of the dataset and the complexity of the model. In the next section, we describe efficient data-deletion methods [24, 28] that can efficiently *remove* undesirable training examples from a specified model, which may be especially useful in continual-learning settings [43, 122] where models may need to be updated regularly.

2.7 Machine Unlearning

Recent legislation [32, 33, 64] requiring companies to *remove* private user data upon request has prompted new discussions on data privacy and ownership [188], especially since membership inference attacks [35, 224] can

accurately test whether a model was trained with a given training example. Fulfilling this "right to be forgotten" [77, 129] may require updating any models trained on data requested to be deleted [216]. However, retraining a model from scratch on a revised dataset becomes prohibitively expensive as dataset sizes and model complexities increase [189]; the result is wasted time and computational resources, exacerbated as the frequency of data removal requests increases.

This motivation has given rise to the nascent field of *machine unlearning* [151], in which the goal is to "unlearn" specific training examples by updating a trained model to completely remove their *influence*. We base our definition of unlearning on prior work by Ginart et al. [82, Def. 3.1]. We define a (possibly randomized) *learning algorithm*, $\mathcal{A} : \mathcal{D} \to \mathcal{H}$, as a function from a dataset $\mathcal{D}$ to a model in hypothesis space $\mathcal{H}$. A *removal method*, $\mathcal{R} : \mathcal{A}(\mathcal{D}) \times \mathcal{D} \times (\mathcal{X} \times \mathcal{Y}) \to \mathcal{H}$, is a function from a model $\mathcal{A}(\mathcal{D})$, dataset $\mathcal{D}$, and an instance to remove from the training data (x, y) to a model in $\mathcal{H}$.

Exact Unlearning. Machine unlearning approaches can be classified into two broad categories: *exact unlearning* and *approximate unlearning*. For *exact unlearning* (a.k.a. *perfect unlearning*), the removal method must be equivalent to applying the training algorithm to the dataset with instance (x, y) removed. In the case of randomized training algorithms, we define equivalence as having identical probabilities for each model in $\mathcal{H}$:

$$P(\mathcal{A}(\mathcal{D} \setminus (x, y))) = P(\mathcal{R}(\mathcal{A}(\mathcal{D}), \mathcal{D}, (x, y))) \tag{2.12}$$

The simplest approach to exact unlearning is to ignore the existing model and simply rerun $\mathcal{A}$ on the updated dataset, $\mathcal{D} \setminus (x, y)$. We refer to this as the *naive retraining* approach. Naive retraining is agnostic to virtually all machine learning

models, easy to understand, and easy to implement. However, this approach becomes prohibitively expensive as the dataset size, model complexity, and number of deletion requests increase.

There are a number of works that support exact unlearning of SVMs [36, 40, 60, 116, 172, 211] in which the original goal was to accelerate leave-one-out cross-validation [182]. More recently, Cao and Yang [34] developed deletion mechanisms for several models that fall under the umbrella of non-adaptive SQ-learning [120] in which data deletion is efficient and exact (e.g. naive Bayes, item-item recommendation, etc.); Schelter [174] has also developed decremental update procedures for similar classes of models. Ginart et al. [82] introduced a quantized variant of the k-means algorithm [136] called Q-k-means that supports exact data deletion. Bourtoule et al. [23] and Aldaghri et al. [5] propose training an ensemble of deep learning models on disjoint "shards" of a dataset, saving a snapshot of each model for every data point; the biggest drawbacks are the large storage costs, applicability only to *iterative* learning algorithms, and the significant degradation of predictive performance.

Schelter et al. [175] enable efficient data removal for extremely randomized trees (ERTs) [79] without needing to save the training data by precomputing alternative subtrees for splits sensitive to deletions; aside from only being applicable to ERTs, they assume a very small percentage of instances can be deleted. Later in Chapter 5, we introduce a method that enables the exact removal of training data from random forest models.

Approximate Unlearning. In contrast to exact unlearning, approximate unlearning (a.k.a statistical unlearning) guarantees

$$\forall \mathcal{S} \subseteq \mathcal{H}, \mathcal{D}, (x, y) \in \mathcal{D} :\\ e^{-\epsilon} \leq \frac{P(\mathcal{R}(\mathcal{A}(\mathcal{D}), \mathcal{D}, (x, y)) \in S)}{P(\mathcal{A}(\mathcal{D} \setminus (x, y)) \in S)} \leq e^{\epsilon}. \tag{2.13}$$

Equation (2.13) is also known as ϵ-certified removal (Guo et al. [91], Equation 1).

Golatkar et al. [85, 86] propose a scrubbing mechanism for deep neural networks that does not require any retraining; however, the computational complexity of their approach is currently quite high. Guo et al. [91], Izzo et al. [108], and Wu et al. [220] propose different removal mechanisms for linear and logistic regression models that can be applied to the last fully connected layer of a deep neural network. Golatkar et al. [87] perform unlearning on a linear approximation of large-scale vision networks in a mixed-privacy setting. Fu et al. [76] propose an unlearning procedure for models in a Bayesian setting using variational inference.

Mitigation. Although not specifically designed as unlearning techniques, the following works propose different mechanisms for mitigating the impact of noisy, poisoned, or non-private training data. Baumhauer et al. [14] propose an output filtering technique that prevents private data from being leaked; however, their approach does not update the model itself, potentially leaking information if the model were still accessible. Wang et al. [218] and Du et al. [58] fine-tune their models on corrected versions of poisoned or corrupted training instances to mitigate backdoor attacks [90] on image classifiers and anomaly detectors, respectively. Although both approaches show promising empirical performance, they provide no guarantees about the extent to which these problematic training instances

are removed from the model [194]. Tople et al. [208] analyze privacy leakage in language model snapshots before and after they are updated.

Differential Privacy. Differential privacy (DP) [1, 37, 62] is a sufficient condition for approximate unlearning (in the case of a single deletion, sequential deletions may require using group DP [63]), but it is an unnecessary and overly strict one since machine unlearning does not require instances to be private [91]. Furthermore, differentially-private random forest models often suffer from poor predictive performance [69, 71]; this is because the privacy budget (typically denoted ϵ or β) must be split among all the trees in the forest, and among the different layers in each tree. This typically results in a meaningless privacy budget (e.g. $\epsilon > 10$) [71], a relaxed definition of DP [165], extremely randomized trees [70, 79], or very small forests (e.g. $T = 10$) [47].

Applications. A popular form of interpretability looks at how much each training instance contributes to a given prediction [41, 125, 160, 184, 221]. The naive approach to this task involves leave-one-out retraining for every training instance in order to analyze the effect each instance has on a target prediction, but this is typically intractable for most machine learning models and datasets. DaRE models can more efficiently compute the same training-instance attributions as the naive approach, making leave-one-out retraining a potentially viable option for generating instance-attribution explanations for random forest models.

Aside from removing user data for privacy reasons, one may also wish to efficiently remove outliers [55, 162] or training instances that are noisy, corrupted, or poisoned [148, 196]. As previously mentioned, these examples may be more

efficiently identified using one of the influence estimation techniques described in Chapter 3.

Our methods can also be used to *add* data to a random forest model, allowing for continuous updating as data is added and removed. This makes them well-suited to continual learning settings with streaming data [43, 122]. However, the hyperparameters may need to be periodicially retuned as the size or distribution of the data shifts from adding and/or deleting more and more instances.

Finally, this line of research promotes a more economically and environmentally sustainable approach to building learning systems; if a model can be continuously updated only as necessary and avoid frequent retraining, significant time and computational resources can be spared [92].

CHAPTER 3

IDENTIFYING INFLUENTIAL TRAINING EXAMPLES

J. Brophy; Z. Hammoudeh; D. Lowd. Adapting and Evaluating Influence-Estimation Methods for Gradient-Boosted Decision Trees. (In Submission to the Journal of Machine Learning Research)

In this chapter, we adapt recent and popular influence-estimation techniques designed for deep learning models to GBTs. Specifically, we introduce *TREX* (§3.1.4), the adaptation of representer-point methods [222], and *BoostIn* (§3.1.3), the adaptation of TracIn [160]. 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 GBTs. 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 [97, 125, 221].

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 (§3.3).

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 [118], XGBoost [39], CatBoost [159], and gradient boosting from Scikit-Learn [154]. Our results suggest the adaptation of TracIn [160]—which we denote *BoostIn*—is a solid default choice for influence estimation in GBTs. BoostIn is over *four orders of magnitude* faster than the current state-of-the-art: LeafInfluence [185]—the adaptation of Influence Functions [125] to GBTs—and provides better influence estimates than competing methods in the majority of tested contexts, on average (§3.4).

Furthermore, leave-one-out (LOO)—removing training examples one at a time, retraining the model for 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 (§3.4.4). 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 (§3.4.3). 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.

3.1 Adapting Influence Methods to GBTs

We first review *LeafRefit* (§3.1.1) and *LeafInfluence* (§3.1.2), work by Sharchilev et al. [185] adapting LOO and influence functions [125] to GBTs. Then we introduce *BoostIn* (§3.1.3), the adaptation of TracIn [160], theoretically compare BoostIn to LeafInfluence, and then adapt representer-point methods [221]—a method we denote *TREX* (§3.1.4).

3.1.1 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. [185] develop *LeafRefit*, a method that computes the influence of z_i on the loss of z_{te} in GBTs by *refitting* all leaf values without z_i while assuming a *fixed-structure.*

Assumption 1. *(Fixed Structure) The effect of removing a single training point z_i can be estimated while treating the structure of each tree as fixed. Feature splits are considered part of the structure of each tree, precluding the influence of z_i on any node split.*

Assumption 1 enables LeafRefit to avoid recomputing node splits for each tree; LeafRefit is thus LOO assuming a fixed structure. Although LeafRefit avoids retraining completely from scratch, recomputing leaf values is an expensive operation.

3.1.2 LeafInfluence: Adapting Influence Functions. Influence functions [48] 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, giving new parameters $\tilde{\theta}_{w_i,z_i} := \arg\min_{\tilde{\theta}\in\tilde{\Theta}} \frac{1}{n}\sum_{j=1}^{n} \tilde{\ell}(z_j,\tilde{\theta}) + w_i\tilde{\ell}(z_i,\tilde{\theta})$ in which $\tilde{\ell}(z_i,\tilde{\theta})$ is the loss of z_i when using parameters $\tilde{\theta}$. The influence of w_i on the model parameters can thus be

approximated *without* having to retrain from scratch:

$$\frac{d\tilde{\theta}_{w_i,z_i}}{dw_i}\Big|_{w_i=0} = -H_{\tilde{\theta}}^{-1}\nabla_{\tilde{\theta}}\tilde{\ell}(z_i,\tilde{\theta}), \tag{3.1}$$

where $H_{\tilde{\theta}} = \frac{1}{n}\sum_{i=1}^{n}\nabla^2_{\tilde{\theta}}\tilde{\ell}(z_i,\tilde{\theta})$. [125] use this result to develop an influence-estimation method 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_{te}:

$$\begin{aligned}\mathcal{I}_{IF}(z_i,z_{te}) &= \nabla_{\tilde{\theta}}\tilde{\ell}(z_{te},\tilde{\theta})^{\intercal}\frac{d\tilde{\theta}_{w_i,z_i}}{dw_i}\Big|_{w_i=0}\\ &= -\nabla_{\tilde{\theta}}\tilde{\ell}(z_{te},\tilde{\theta})^{\intercal}H_{\tilde{\theta}}^{-1}\nabla_{\tilde{\theta}}\tilde{\ell}(z_i,\tilde{\theta}).\end{aligned} \tag{3.2}$$

Their method (which we henceforth refer 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 or non-convex loss functions, however, it has recently been shown that this approximation typically does not hold [13].

For GBDTs, [185] 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_{te}, assuming fixed tree structures:

$$\begin{aligned}\mathcal{I}_{LI}(z_i,z_{te}) &= \frac{\partial\ell(y_{te},f(x_{te}))}{\partial w_i}\\ &= \frac{\partial\ell(y_{te},\hat{y_{te}})}{\partial\hat{y_{te}}}\Big|_{\hat{y_{te}}=f(x_{te})}\cdot\frac{\partial f(x_{te})}{\partial w_i}.\end{aligned} \tag{3.3}$$

Equation (3.3) approximates the change in loss on z_{te} as a result of upweighting z_i.[1] The effect of z_i on the *final* GBDT model (second term in Eq. 3.3)

[1]In our experiments, we take the negative of Eq. (3.3) to be consistent with the concept of proponents and opponents, defined in §2.5.

is defined as:

$$\frac{\partial f(x_{te})}{\partial w_i} = \sum_{t=1}^{T} \frac{\partial \theta_{t,l_e}(f_{t-1}(\mathbf{x}))}{\partial w_i}, \tag{3.4}$$

in which $l_e = R_t(x_{te})$ is the leaf assigned to x_{te} 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 leaf l_e at iteration t with respect to the ith training instance is defined as:

$$\frac{\partial \theta_{t,l_e}(f_{t-1}(\mathbf{x}))}{\partial w_i} = -\frac{\mathbb{1}[i \in I_{t,l_e}](h_{t,i}\theta_{t,l_e} + g_{t,i}) + \sum_{j \in I_{t,l_e}} (k_{t,j}\theta_{t,l_e} + h_{t,j}) J(f_{t-1}(\mathbf{x}))_{i,j}}{\sum_{j \in I_{t,l_e}} h_{t,j}}, \tag{3.5}$$

in which $g_{t,i} = g_t(x_i, y_i)$, $h_{t,i} = h_t(x_i, y_i)$, $k_{t,i} = \partial^3 \ell(y_i, \hat{y}_i)/\partial \hat{y}_i{}^3|_{\hat{y}_i = f_{t-1}(x_i)}$, 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 effect of a single training example z_i on the loss of z_{te} can be found by running x_{te} through a new ensemble with new leaf values defined by Equation (3.5) and multiplying the result by the derivative of the loss with respect to the original prediction (Equation 3.3).

Approximating the influence of a *single* training example via LeafInfluence (Eq. 3.3) 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 $\mathcal{I}_{LI}(z_i, z_{te})$ 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 (§3.4.2).

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, [185] introduce LeafInfluence-SinglePoint (which we henceforth call LeafInfSP), a rough approximation to the main proposed approach (Equation 3.3):

$$\mathcal{I}_{LI_{SP}}(z_i, z_{te}) = \frac{\partial \ell(y_{te}, \hat{y_{te}})}{\partial \hat{y_{te}}}\Big|_{\hat{y_{te}}=f_T(x_{te})} \cdot \left(\sum_{t=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_{te})] \left(\frac{\partial \theta_{t,R_t(x_i)}(f_{t-1}(x_i))}{\partial w_i} \right) \right), \tag{3.6}$$

in which

$$\frac{\partial \theta_{t,l=R_t(x_i)}(f_{t-1}(x_i))}{\partial w_i} = -\frac{(g_{t,i} + h_{t,i}\theta_{t,l}) + (h_{t,i} + k_{t,i}\theta_{t,l})\tilde{J}(f_{t-1}(x_i))}{\sum_{j \in I_{t,l}} w_j h_{t,j}} \tag{3.7}$$

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_{te} *throughout* the training process. We then compare BoostIn to LeafInfSP and LeafInfluence, higlighting the similarities and differences.

3.1.3 BoostIn: Adapting TracIn. *TracIn* [160] is an influence-estimation method designed for deep learning models that analyzes the impact of z_i on z_{te} *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_{te} whenever z_i is used to update the model; that is, $\mathcal{I}_{TracInIdeal}(z_i, z_{te}) = \sum_{t:z^t=z_i} \tilde{\ell}(z_{te}, \tilde{\theta}_t) - \tilde{\ell}(z_{te}, \tilde{\theta}_{t+1})$. This notion of influence has the appealing property that the sum of influences for all training examples on z_{te} is *exactly* the total reduction in loss during training: $\sum_{i=1}^{n} \mathcal{I}_{TracInIdeal}(z_i, z_{te}) = \tilde{\ell}(z_{te}, \tilde{\theta}_0) - \tilde{\ell}(z_{te}, \tilde{\theta}_T)$. However, deep learning models are almost never updated 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 on the loss of any target example z_{te} can then be computed via a sum of first-order approximations of the loss as follows:

$$\mathcal{I}_{TracInCP}(z_i, z_{te}) = \sum_{i=1}^{k} \eta_i \nabla \tilde{\ell}(z_i, \tilde{\theta}_t) \cdot \nabla \tilde{\ell}(z_{te}, \tilde{\theta}_t), \tag{3.8}$$

in which k is the number of checkpoints, and η_i is the learning rate at checkpoint i.

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 (Equation 2.6). Thus to adapt TracIn to GBDTs,

we first consider all intermediate GBDT models constructed during training as checkpoints $(f_0, f_1, \ldots, f_T)$. 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 (Equation 2.6).

We process 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_{te} only if z_i and z_{te} are in the same leaf at that iteration. The estimated change in prediction then approximates the change in loss on z_{te}. Finally, we aggregate these approximations across all iterations, simulating the effect of z_i on z_{te} throughout the training process. More formally, we use the chain rule to analyze the marginal effect each changing leaf value has on the loss of z_{te}, and sum these marginal effects across checkpoints:

$$\begin{aligned}
\mathcal{I}_{BoostIn}(z_i, z_{te}) &= \sum_{t=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_{te})] \left(-\frac{d\ell(y_{te}, f_t(x_{te}))}{dw_i} \right) \\
&= \sum_{t=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_{te})] \left(-\frac{\partial \ell(y_{te}, \hat{y_{te}})}{\partial \hat{y_{te}}}\Big|_{\hat{y_{te}}=m_t(x_{te})} \cdot \frac{\partial f_t(x_{te})}{\partial w_i} \right) \\
&= \sum_{t=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_{te})] \left(\frac{\partial \ell(y_{te}, \hat{y_{te}})}{\partial \hat{y_{te}}}\Big|_{\hat{y_{te}}=f_t(x_{te})} \cdot \eta_t \frac{\partial \theta_{t,l}}{\partial w_i} \right)
\end{aligned} \quad (3.9)$$

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

$$\frac{\partial \theta_{t,l}}{\partial w_i} = \frac{g_{t,i} + h_{t,i}\ \theta_{t,l}}{\sum_{j \in I_{t,l}} h_{t,j} + \lambda}. \quad (3.10)$$

We denote this adaptation of TracIn to GBDTs as *BoostIn*. Again, note Eq. (3.9) enforces a constraint that attributes nonzero influence only

when z_i and z_{te} 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 leaf $l = R_t(x_i)$ for weak learner m_t. The resulting runtime complexity of BoostIn for computing the influence of a *single* training example is $O(T)$, and computing influence values for *all* training examples is $O(Tn)$.

BoostIn vs. LeafInfluence-SinglePoint. The main idea of LeafInfSP (Eq. 3.6) is to *accumulate* the changes in the leaf values affected by upweighting z_i (only relevant when z_i and z_{te} are in the same leaf at a given iteration t) and multiply this result by the *final* prediction of the GBDT model on x_{te}. In contrast, BoostIn (Eq. 3.9) multiplies each leaf-value derivative by the corresponding *intermediate* prediction of x_{te} at each iteration (only relevant when z_i and z_{te} 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 $O(Tn)$. Empirically, however, the original implementation of LeafInfSP[2] is not optimized to realize this lower runtime complexity as it is implemented in conjunction with full LeafInfluence approach; thus, as a minor contribution, we implement an optimized version of LeafInfSP and demonstrate in §3.4.2 that our implementation achieves similar runtime performance compared to BoostIn, as expected. Overall, BoostIn is a solid choice for influence estimation in GBDTs,

[2] `https://github.com/bsharchilev/influence_boosting`

providing on par or better influence estimates than LeafInfSP and LeafInfluence while being orders of magnitude more efficient than LeafInfluence.

3.1.4 TREX: Adapting Representer-Point Selection.

Representer theorems [177] state the optimal solutions of many learning problems can be represented in terms of the training examples. In particular, the nonparametric representer theorem [177, Theorem 4] applies to empirical risk minimization within a reproducing kernel Hilbert space (RKHS); this covers a wide range of linear and kernelized machine-learning methods.

Yeh et al. [221] 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 $\mathbf{f}_i = \tilde{\phi}(x_i; \tilde{\theta}_1)$, and training a kernelized model with L2 regularization on the transformed features. Specifically, a surrogate model f^* parameterized by ψ^* is solved via the following optimization problem:

$$\psi^* = \arg\min_{\psi} \lambda ||\psi||^2 + \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f^*(\mathbf{f}_i; \psi)). \tag{3.11}$$

Once a stationary point is reached with high precision, the gradient of the loss can be set equal to zero:

$$\frac{1}{n} \sum_{i=1}^{n} \frac{\partial \ell(y_i, f^*(\mathbf{f}_i; \psi^*))}{\partial \psi^*} + 2\lambda\psi^* = 0, \tag{3.12}$$

$$\therefore \psi^* = -\frac{1}{2\lambda n} \sum_{i=1}^{n} \frac{\partial \ell(y_i, f^*(\mathbf{f}_i; \psi^*))}{\partial \psi^*} = \sum_{i=1}^{n} \alpha_i \mathbf{f}_i, \tag{3.13}$$

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

$$f^*(x_{te}; \psi^*) = \sum_{i=1}^{n} \alpha_i k(x_i, x_{te}), \tag{3.14}$$

in which $k(x_i, x_{te}) = \langle \mathbf{f}_i, \mathbf{f}_e \rangle$ represents the similarity between z_i and z_{te} in the transformed feature space; $\alpha_i k(x_i, x_{te})$ is referred to as the *representer value* for z_i given z_{te}, and its magnitude is largest when the magnitudes of *both* the training example weight α_i and the similarity of z_i to z_{te} 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_{te}, 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* [20, 52, 103], a general approach for extracting the learned representation from a tree ensemble by using the structure of the trees. Tree kernels can measure the similarity between two instances by analyzing how each example is processed by each tree in the ensemble [146], and they have been shown to be an effective adaptive nearest-neighbors method [134] that can more effectively identify the most relevant training instances for a given instance than traditional nearest-neighbor approaches that operate in the original feature space and often suffer from poor performance as the dimensionality of the data increases.

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 extra flexibility into the similarity measure by taking into account path overlap, node weights (number of examples at a node), and leaf values. Formally, we define tree kernels as dot products in an alternate feature representation defined by the feature mapping ϕ, i.e., $k(x_i, x_{te}; f) = \langle \phi(x_i; f), \phi(x_{te}; f) \rangle = \langle \mathbf{f}_i, \mathbf{f}_e \rangle$. 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. [157] on local linear modeling, we define $\phi(x, f) = \cup_{t=1}^{T}(\frac{1}{n_{t,l}} \cdot \mathbb{1}[x \in r_{t,l}])_{l=1}^{M_t}$ as a sparse vector over all leaves in f; for each tree t, the element at leaf $R_t(x)$ is nonzero and inversely weighted by $n_{t,l}$, the number of training examples assigned to leaf l.

Given the feature mapping ϕ, we use Eq. (3.11) to train a surrogate model to approximate our original GBDT model, enabling the decomposition of a target prediction using Equation (3.14). However, the resulting representer values only quantify the contribution of a training example z_i to the *prediction* of z_{te}, but we are interested in how z_i affects the loss of z_{te}; thus, we measure the influence of z_i on the loss of z_{te} by subtracting the representer value for z_i from the prediction decomposition of z_{te} and computing the change in loss:

$$\mathcal{I}_{TREX}(z_i, z_{te}) = \ell\Big(y_{te}, \upsilon\big(\hat{y}_e^* - \alpha_i k(x_i, x_{te}; f)\big)\Big) - \ell\big(y_{te}, \upsilon(\hat{y}_e^*)\big), \tag{3.15}$$

in which υ is an activation function[3] and $\hat{y}_e^* = \sum_{j=1}^{n} \alpha_j k(x_i, x_{te}; f)$ is the surrogate model pre-activation prediction for z_{te}. We denote this method *TREX* (**T**ree-ensemble **R**epresenter **Ex**amples) and evaluate its influence-estimation quality in §3.4.

3.1.5 Similarity-Based Influence. As defined above, TREX measures the influence of an example z_i on a target example z_{te} using two pieces of information: the weight of the training example α_i, and the similarity to the target instance $k(x_i, x_{te}; f)$. To better understand the marginal effect of each component, we define an additional influence estimation method that skips training a surrogate

[3] Typically a sigmoid or softmax for binary and multiclass classification tasks.

model and *only* uses the tree-kernel similarity to quantify the influence of z_i on z_{te}:

$$\mathcal{I}_{TreeSim}(z_i, z_{te}) = \mathbb{1}[y_i = y_{te}]k(x_i, x_{te}; f) - \mathbb{1}[y_i \neq y_{te}]k(x_i, x_{te}; f). \tag{3.16}$$

This method, *TreeSim*, attributes positive influence to examples with the *same* label as z_{te}, and negative otherwise;[4] this value is then scaled by the similarity between z_i and z_{te}.

3.2 Summary of Influence-Estimation Methods

Table 1 provides a summary of the influence-estimation methods discussed in this paper. In summary, 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 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 provide approximations to 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 theoretically 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.

[4]For regression, TreeSim treats z_i and z_{te} as having the same label if y_i and y_{te} are on the same side of the target prediction $\hat{y}_e$; that is, $sgn(\hat{y}_e - y_i) = sgn(\hat{y}_e - y_{te})$, in which sgn is the signum function.

Method	**Source**	**Changes**	**Assume Fixed Structure**
LOO	-	-	
SubSample	[66]	-	
LeafRefit	[185]	-	✓
LeafInfluence	[125]	[185]	✓
LeafInfSP	[125]	[185]	✓
BoostIn	[160]	§3.1.3	✓
TREX	[221]	§3.1.4	✓
TreeSim	[157]	§3.1.4	✓

Table 1. Summary of influence-estimation methods.

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

3.3 Methodology

In our experiments, we order the training data based on the influence values generated by each method for a *single* test example (§3.3.2, §3.3.3) or a *set* of test examples (§3.3.4, §3.3.5, §3.3.6). We then evaluate these orderings in different contexts by removing (§3.3.2, §3.3.4), performing targeted label editing (§3.3.3), and adding label noise (§3.3.5) to the most influential examples and observing the effect on the model/predictions after *retraining*; the more a method degrades the resulting model predictions,[5] the higher that method is ranked. We also evaluate the effectiveness of each method at detecting noisy or mislabelled training examples (§3.3.6) as is often done in previous work [80, 160]. 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

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

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 §3.4.

3.3.1 Datasets and Methods.

This section describes the datasets, models, and influence methods used in our experiments.

Datasets. 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 [118, LGB], XGBoost [39, XGB], Scikit-Learn [154, SGB], and CatBoost [159, CB]. Each model is tuned using 5-fold cross-validation; selected hyperparameters are in §A.3.1: Table A.11, with predictive performance comparisons in §A.3.1: Table A.10.

Influence Methods and Baselines. We include the following influence-estimation methods in our evaluation: LOO (Equation 2.9), SubSample (Equation 2.11; we set $\tau = 4,000$ and $m = \lfloor 0.7n \rfloor$ as recommended by Feldman and Zhang, 2020), LeafRefit, LeafInfluence (Equation 3.3), BoostIn (Equation 3.9), LeafInfSP (Equation 3.6), TREX (Equation 3.15), and

TreeSim (Equation 3.16). We also include *Random* as an additional baseline, which assigns influence values via a standard normal distribution: $\mathcal{I}_{Random}(z_i, z_{te}) \sim \mathcal{N}(0, 1)$.

Due to the limited scalabilty of LeafRefit and LeafInfluence (see §3.4.2), 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 A.8); however, we observe the same trends when including all data sets in our analysis, which are in §A.2.1.

Implementation and Reproducibility. 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.1.

3.3.2 Single Test Instance: Removing Influential Training Examples.

Inspired by the remove and retrain (ROAR) framework for measuring the impact of different features [104], 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_{te}, and order the training instances from most positive (i.e., instances that decrease the loss of z_{te} 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_{te} 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.

3.3.3 Single Test Instance: Targeted Training-Label Edits. In this section, instead of removing examples, we ask the counterfactual [31, 119], "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 them to simulate a changing training label for this experiment. We modify LOO to compute the influence of z_i on z_{te} 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 §3.1). 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, $\mathcal{I}(z_i \rightarrow z_i^*, z_{te}) \approx \mathcal{I}(z_i, z_{te}) - \mathcal{I}(z_i^*, z_{te})$.

In this experiment, we use the same setup as §3.3.2, 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^*.[6] 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.

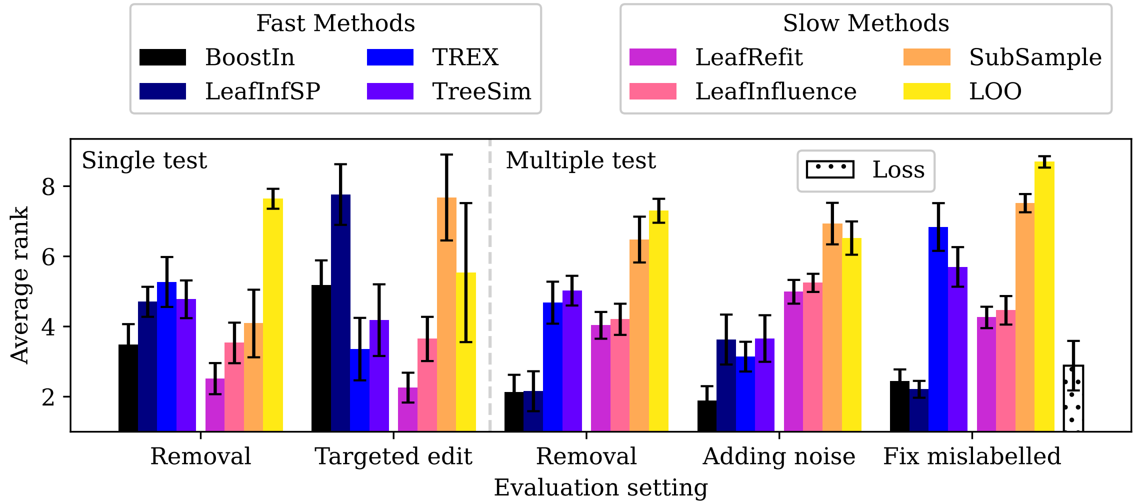
3.3.4 Multiple Test Instances: Removing Influential Training Examples. We now analyze the effect of influential examples on a *set* of test instances. We sample 10% of the test examples uniformly at random to use as a validation set and generate influence values for each example. We then aggregate the 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.

3.3.5 Multiple Test Instances: Adding Training-Label Noise. For this experiment, we use the same setup as §3.3.4, 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

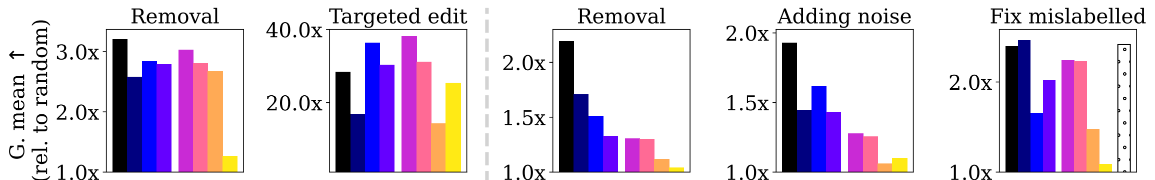
[6]For binary-classification tasks, we choose y^* to be opposite $\hat{y}_e$ (the predicted label of z_{te}). For multiclass classification, y^* is randomly sampled from $\mathcal{Y} \setminus \hat{y}_e$. For regression, $y^* = \bar{\mathbf{y}} - (\bar{\mathbf{y}}/2)$ if $\hat{y}_e > \bar{\mathbf{y}}$,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

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 [131, 196]. We then retrain, remeasure, and rank the influence methods in the same way as §3.3.4.

3.3.6 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 [80, 125, 156, 160, 221], sampling 40% of the training data uniformly at random and flipping their labels in the same way as §3.3.5. We then generate influence values in the same way as §3.3.4 and §3.3.5, 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 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).



(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 2. 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 3); 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.

3.4 Results and Analyses

Figure 2 shows a high-level overview of our results across evaluation settings; we partition the influence techniques into "fast" and "slow" methods (see §3.4.2

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 §3.4.4.

3.4.1 Summary of Results. Here we present an overview of the results for each experiment, with additional analyses in the Appendix, §A.2.2-§A.2.6.

Single Test: Removing Influential Examples. Figure 2a (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.2.1); increasing τ for larger data sets may improve performance, but may also significantly increase its running time. In terms of magnitude, all methods increase the loss more than random, though BoostIn has a slight advantage over all other methods (Figure 2b: left); surprisingly, LOO does only marginally better than random.

Single Test: Targeted Label Edits. Figure 2a (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 2b: middle-left).

Multiple Test: Removing Influential Examples. Figure 2a (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 2b-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.2.4.

Multiple Test: Adding Label Noise. Our results show BoostIn clearly outperforms all other methods in terms of rank (Figure 2a: middle-right) and relative loss increase (Figure 2b: 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.2.5.

Multiple Test: Fixing Mislabelled Examples. Figure 2a (right) shows BoostIn and LeafInfSP performing best, slightly outranking Loss; however, all three methods perform comparably in terms of relative magnitude, on average (Figure 2b: 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.2.6.

3.4.2 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.[7] We repeat each experiment 5 times, and average the results over GBDT types.

[7]For a clearer comparison, no parallelization is used for any method.

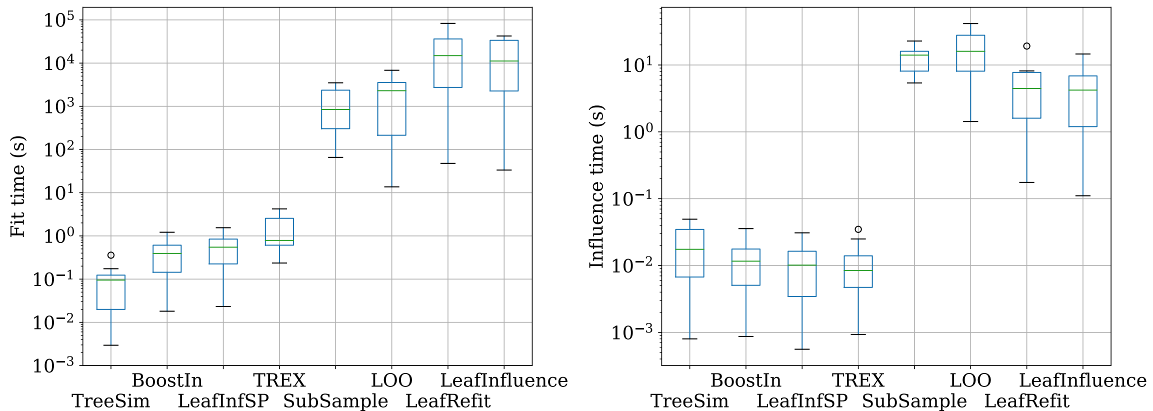


Figure 3. Left: Average setup time for each explainer. *Right*: Average time to compute influence values of all training examples for one test example. Results are averaged over 5 folds and GBDT types; each box plot represents average running times across all SDS datasets. TreeSim, BoostIn, LeafInfSP, and TREX represent "fast" methods with low setup and influence times, and are separated from the remaining "slow" methods by orders of magnitude efficiency.

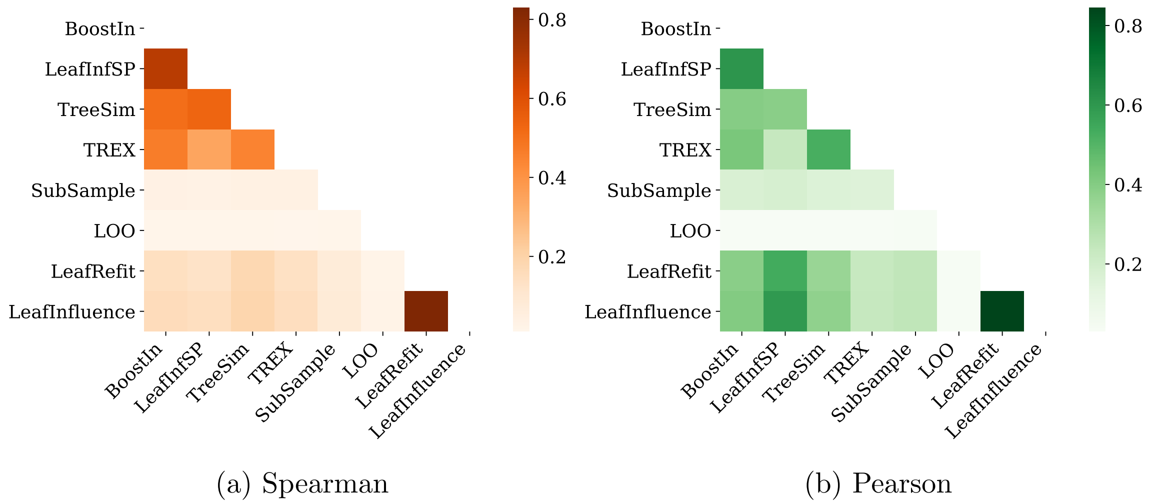
Figure 3 shows the runtime of each approach broken down into two components: "fit time" and "influence time". Fit time is the time to initialize and set up the explainer, and influence time is the time to compute the influence of all training examples for one test example; note the log scale. TreeSim 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.

3.4.3 Correlation Between Influence Methods. To better understand the relationships between different influence methods, we compute the Spearman rank [227] and Pearson correlation coefficients between every pair of influence methods using the values generated for each test example in §3.3.2. We then average these correlations over all test examples, GBDT types, and data sets.



(a) Spearman

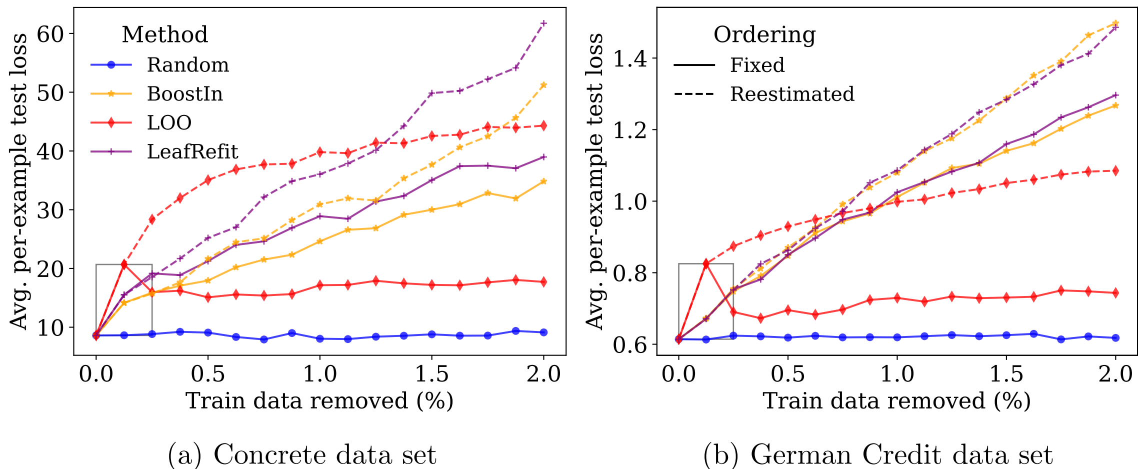
(b) Pearson

Figure 4. 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 4 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 §3.1.3. 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 2) prompts us to investigate LOO further in §3.4.4. 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.2.8.

3.4.4 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 §3.3.2, 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 5 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.2.9). This suggests that LOO indeed often chooses the single-most influential example to the test instance, but fails to provide the most influential set of examples based on its initial ordering.



(a) Concrete data set

(b) German Credit data set

Figure 5. 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.2.9.

To quantify how much this initial 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 5). 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. This suggests that greedily selecting training examples that have a large impact on the test loss (such as LOO) may fare worse than methods using a fixed-structure assumption when considering a larger number of examples.

Figure 6 validates structural changes are induced by a single removal ordered by LOO, measured via training-example affinities (number of times the training examples are assigned to the same leaf as the test example across all

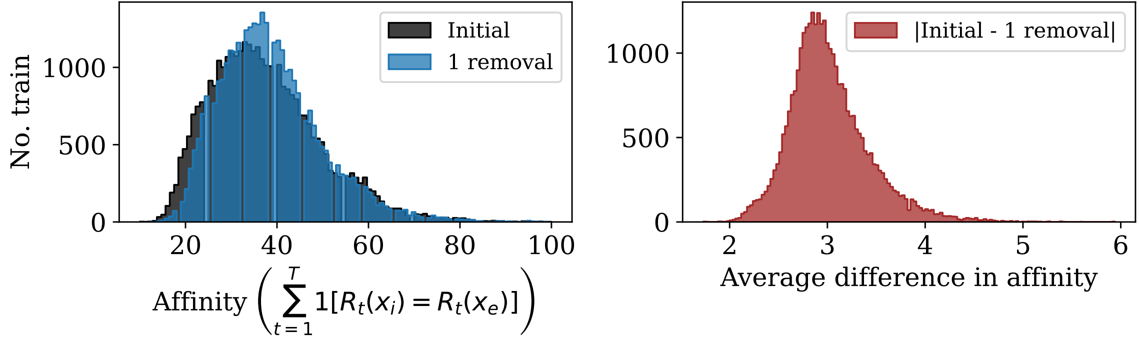


Figure 6. Left: Distribution of training-example affinities to a randomly selected test example z_{te} using an LGB model trained on the Adult data set before (initial) and after the removal of a single training example (1 removal) ordered using LOO. *Right*: Average change in affinity over 100 randomly selected test examples. The changing distribution of affinity values signals structural changes to the tree structures after only a single removal.

trees in the ensemble). These structural changes may help explain why LOO has significantly different performance (Figure 2) and low correlation (Figure 4) with the other methods.

3.5 Summary

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.

CHAPTER 4

QUANTIFYING PREDICTION UNCERTAINTY

J. Brophy and D. Lowd. Instance-Based Uncertainty Estimation for Gradient-Boosted Regression Trees. In Proceedings of the Thirty-Sixth International Conference on Neural Information Processing Systems. New Orleans, LA. 2022.

In this chapter, we introduce a simple yet effective method for enabling *any* GBRT point-prediction model to produce probabilistic predictions. Our proposed approach, Instance-Based Uncertainty estimation for Gradient-boosted regression trees (*IBUG*), has two key components: 1) We leverage the fact that GBRTs accurately model the conditional mean and use this point prediction as the mean in a probabilistic forecast; and 2) We identify the k training examples with the greatest *affinity* to the test instance and use these examples to estimate the uncertainty of the test prediction. We define the affinity between two instances as the number of times both instances appear in the same leaf throughout the ensemble. Thus, our method acts as a wrapper around any given GBRT model, such as LightGBM [118], XGBoost [39], CatBoost [159], or any other model with (potentially) improved point-prediction performance invented in the future.

In experiments on 21 regression benchmark datasets and one synthetic dataset, we demonstrate the effectiveness of IBUG to deliver on par or improved probabilistic performance as compared to existing state-of-the-art methods while maintaining state-of-the-art point-prediction performance. We also show that probabilistic predictions can be improved by applying IBUG to different GBRT models, something that NGBoost and PGBM cannot do. Additionally, IBUG can use the training instances closest to the target example to directly model

the output distribution using any parametric *or* non-parametric distribution; again, something NGBoost, PGBM, *and* CBU cannot do. Finally, we show that sampling trees dramatically improves runtime efficiency for computing training-example affinities without having a significant detrimental impact on the resulting probabilistic predictions, allowing IBUG to scale to larger datasets.

4.1 Instance-Based Uncertainty

Instance-based methods such as k-nearest neighbors have been around for decades and have been useful for many different machine learning tasks [155]. However, defining neighbors based on a fixed metric like euclidean distance may lead to suboptimal performance, especially as the dimensionality of the dataset increases. More recently, it has been shown that random forests can be used as an adaptive nearest neighbors method [52, 134] which identifies the most similar examples to a given instance using the learned model structure. This *supervised tree kernel* can more effectively measure the similarity between examples, and has been used for clustering [146] and local linear modeling [20] as well as instance-[27] and feature-based attribution explanations [157], for example.

In this work, we apply the idea of a supervised tree kernel to help model the *uncertainty* of a given GBRT prediction. Our approach, ***I****nstance-****B****ased* ***U****ncertainty estimation for* ***G****radient-boosted regression trees* (IBUG), identifies the neighborhood of similar training examples to a target example using the structure of the GBRT, and then uses those instances to generate a probabilistic prediction (Figure 7). IBUG works for *any* GBRT, and can more flexibly model the output than competing methods.

4.1.1 Identification of High-Affinity Neighbors.

At its core, IBUG uses the k training examples with the largest *affinity* to the target example

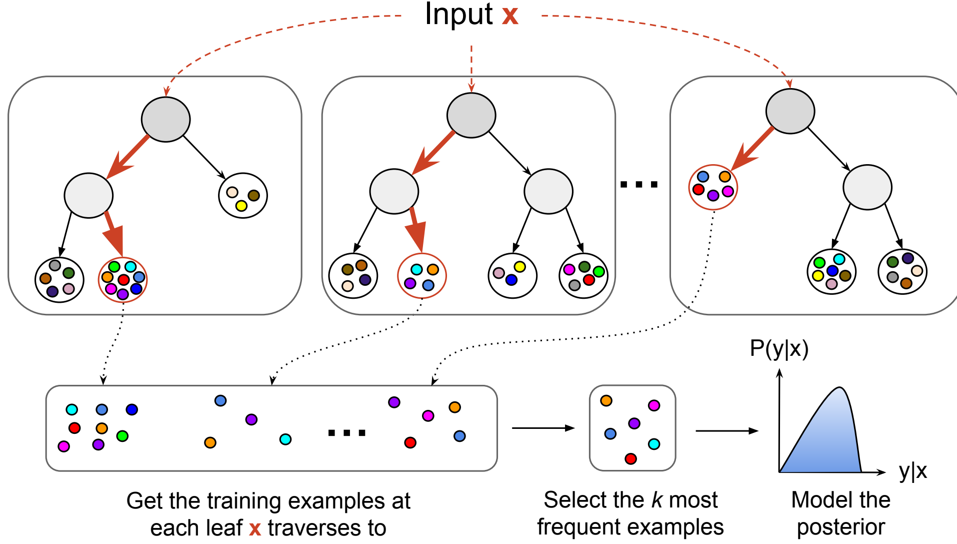


Figure 7. IBUG workflow. Given a GBRT model and an input instance x, IBUG collects the training examples at each leaf x traverses to, keeps the k most frequent examples, and then uses those examples to model the output distribution.

to model the conditional output distribution. Given a GBRT model f, we define the affinity between two examples simply as the number of times each instance appears in the same leaf across all trees in f. Thus, the affinity of the ith training example x_i to a target example x_{te} can be written as:

$$A(x_i, x_{te}) = \sum_{i=1}^{T} \mathbb{1}[R_t(x_i) = R_t(x_{te})], \tag{4.1}$$

in which $R_t(x_i)$ is the leaf x_i is assigned to for tree t. Algorithm 1 summarizes the procedure for computing affinity scores for all training examples. This metric clusters similar examples together based on the learned model representation (i.e., the tree structures). Intuitively, if two examples appear in the same leaf in every tree throughout the ensemble, then both examples are predicted in an identical manner. One may also view Eq. (4.1) as an indication of which training examples most often affect the leaf values x_{te} is assigned to and thus implicitly which examples are likely to have a big effect on the prediction $\hat{y}_{te}$. This similarity metric

is similar to the random forest kernel [52], however, unlike random forests, GBRTs are typically constructed to a shallower depth, resulting in more training examples assigned to the same leaf (see §B.2.5 for additional details about leaf density in GBRTs).

Algorithm 1 IBUG affinity computation.

```
1: ComputeAffinities(Input instance x ∈ 𝒳, GBRT model f)
2:    A ← 0⃗                                   ▷Initialize training-example affinities.
3:    for t = 1 . . . T do
4:       Get instance set I_t^l for leaf l = R_t(x)
5:       for i ∈ I_t^l do
6:          A_i ← A_i + 1                                ▷Increment affinity score.
7:    Return A
```

Algorithm 2 IBUG probabilistic prediction.

```
1: ProbPredict(Input x ∈ 𝒳, GBRT model f, k highest-affinity neighbors A^(k),
   minimum variance ρ, variance calibration parameters γ and δ, target
   distribution D)
2:    μ_ŷ ← f(x)                                         ▷GBRT scalar output.
3:    σ²_ŷ ← max(σ²(A^(k)), ρ)                 ▷Compute σ² and ensure σ² > 0.
4:    σ²_ŷ ← γσ²_ŷ + δ                    ▷Variance calibration, Equation (4.2).
5:    Return D(A^(k)|μ_ŷ, σ²_ŷ)           ▷Probabilistic prediction, Equation (4.3).
```

4.1.2 Modeling the Output Distribution.

IBUG has a multitude of choices when modeling the conditional output distribution. The simplest and most common approach is to model the output assuming a Gaussian distribution [61, 195]. We use the scalar output of f: $\mu_{\hat{y}_{te}} = f(x_{te})$ to model the conditional mean since GBRTs already produce accurate point predictions. Then, we use the k training instances with the largest affinity to x_{te}—we denote this set $A^{(k)}$—to compute the variance $\sigma^2_{\hat{y}_{te}}$.

Calibrating prediction variance. The k-nearest neighbors generally do a good job of determining the relative uncertainty of different predictions, but on

some datasets, the resulting variance is systematically too large or too small. To correct for this, we apply an additional affine transformation before making the prediction:

$$\sigma^2_{\hat{y}_{te}} \leftarrow \gamma \sigma^2_{\hat{y}_{te}} + \delta, \tag{4.2}$$

where γ and δ are tuned on validation data after k has been selected. Instead of exhaustively searching over all values of γ or δ, we use either the multiplicative factor (tuning γ with $\delta = 0$) or the additive factor (tuning δ with $\gamma = 1$), and choose between them using their performance on validation data.

We find this simple calibration step consistently improves probabilistic performance for not only IBUG, but competing methods as well, and at a relatively small cost compared to training the model. Thus, any future probabilistic estimator should at least consider including a multiplicative or additive correction when estimating the predicted variance.

Flexible posterior modeling. In general, we can generate a probabilistic prediction using $\mu_{\hat{y}_{te}}$ and $\sigma^2_{\hat{y}_{te}}$ for any distribution that uses location and scale (note PGBM and CBU can *only* model these types of distributions). However, IBUG can additionally use $A^{(k)}$ to directly fit any continuous distribution D, including those with high-order moments:

$$\hat{D}_{te} = D(A^{(k)} | \mu_{\hat{y}_{te}}, \sigma^2_{\hat{y}_{te}}). \tag{4.3}$$

Eq. (4.3) is defined such that D can be fit directly with $A^{(k)}$ using MLE (maximum likelihood estimation) [149], or may be fit using $\mu_{\hat{y}_{te}}$ or $\sigma^2_{\hat{y}_{te}}$ as fixed parameter values with $A^{(k)}$ fitting any other parameters of the distribution. Overall, directly fitting all or some additional parameters in D—for example, the shape parameter in

a Weibull distribution—is a benefit over PGBM and CBU, which can only optimize for a *global* shape value using a gridsearch-like approach with extra validation data.

Note that NGBoost can model any parameterized distribution, but must specify this choice before training; in contrast, IBUG can optimize this choice *after* training. Additionally, IBUG may choose D to be a *non-parametric density estimator* such as KDE (kernel density estimation) [186], which PGBM, CBU, and NGBoost cannot do.

4.1.3 Summary. In summary, Algorithm 2 provides pseudocode for generating a probabilistic prediction with IBUG. Note Algorithms. 1 and 2 work for *any* GBRT model, allowing practitioners to employ IBUG to adapt multiple different point predictors into probabilistic estimators and select the model with the best performance. Empirically, we show using different base models for IBUG can result in improved probabilistic performance than using just one (§4.2.4).

IBUG is a nearest neighbors approach and thus seems well-suited to estimating aleatoric uncertainty—remaining uncertainty due to irreducible error or the inherent stochasticity in the system [107]—since it can quantify the range of outcomes to be expected given the observed features. However, we use predictions on held-out data to tune the number of nearest neighbors and the variance calibration hyperparameters; thus, we effectively optimize prediction uncertainty encompassing both aleatoric uncertainty and epistemic uncertainty—error due to the imperfections of the model and the training data [57, 143]. The evaluation measures in our experiments thus also focus on predictive uncertainty.

4.1.4 Computational Efficiency. We now discuss the runtime of IBUG and methods for increasing its efficiency.

Training efficiency. Since IBUG works with standard GBRT models, it inherits the training efficiency of modern GBRT implementations such as XGBoost [39], LightGBM [118], and CatBoost [159]. It also benefits from any future developments in training efficiency, with no need to update the IBUG algorithm.

Prediction efficiency. If there are T trees in the ensemble and each leaf has at most n_l training instances assigned to it, then IBUG's prediction time is $O(Tn_l)$, since it considers each instance in each leaf. Note training instances that do not appear in a leaf with the test instance x_{te} do not increase prediction time; what matters most is thus the number of instances at each leaf. We find LightGBM often induces regression trees with large leaves—in some cases, over half the dataset is assigned to a single leaf. Thus, prediction time still grows with the size of the dataset, as is typical for instance-based methods. This higher prediction time is the price IBUG pays for greater flexibility.

Prediction efficiency can be increased at training time by using deeper GBRTs with fewer instances in each leaf, after training by subsampling the instances considered for predictions, or at prediction time by sampling the trees used to compute affinities. We explore this last option in the next subsection.

Sampling Trees. The most expensive operation when generating a probabilistic prediction with IBUG is computing the affinity vector (Eq. 4.1). In order to increase prediction efficiency, we can instead work with a subset of the trees $\tau < T$ in the ensemble. We can build this subset by sampling trees uniformly at random, taking the first trees learned (representing the largest gradient steps), or the last trees learned (representing the fine-tuning steps).

By sampling trees, the runtime complexity reduces to $O(\tau n_l)$, which provides significant speedups when $\tau \ll T$. In our empirical evaluation, we find that taking a subset of the first trees learned generally works best, significantly increasing prediction efficiency while maintaining accurate probabilistic predictions (§4.2.7).

Accelerated k Tuning. Choosing an appropriate value of k is critical for generating accurate probabilistic predictions in IBUG. Thus, we aim to tune k using a held-out validation dataset $\mathcal{D}_{val} \subset \mathcal{D}$ and an appropriate probabilistic scoring metric such as negative log likelihood (NLL). Unfortunately, typical tuning procedures would result in the same affinity vectors being computed—an expensive operation—for each candidate value of k. To mitigate this issue, we perform a custom tuning procedure that reuses computed affinity vectors for all values of k. More specifically, IBUG computes an affinity vector A for a given validation example x_{val}, and then sorts A in descending order (i.e., largest affinity first). Then, IBUG takes the top k training instances, and generates and scores the resulting probabilistic prediction. For each subsequent value of k, the same sorted affinity list can be used, avoiding duplicate computation. We summarize this procedure in Algorithm 3.

Once k is chosen, we may encounter a new unseen target instance in which the variance of $A^{(k)}$ for that target example is zero or extremely small. In this case, we set the predicted test variance to ρ, which is set during tuning to the minimum (nonzero) variance computed over all predictions in the validation set for the chosen k. In practice, we find instances of abnormally low variance to be rare with appropriately chosen values of k.

Algorithm 3 IBUG accelerated tuning of k.

Require: Validation dataset $\mathcal{D}_{val} \subset \mathcal{D}$, GBRT model f, list of candidates K, target distribution D.

1: **for** $(x_j, y_j) \in \mathcal{D}_{val}$ **do**
2: $A \leftarrow$ ComputeAffinities(x_j, f) ▷Algorithm 1.
3: $A \leftarrow$ Argsort A in descending order ▷Use the same ordering for each k.
4: **for** $k \in K$ **do**
5: $A^{(k)} \leftarrow$ Take first k training instances(A, k)
6: $\hat{D}^k_{y_j} \leftarrow$ ProbPredict$(x_j, f, A^{(k)}, \rho, 1, 0, D)$ ▷Algorithm 2.
7: $S^k_j \leftarrow V(y_j, \hat{D}^k_{y_j})$ ▷Save validation score.
8: $k \leftarrow$ Select best k from S
9: $\rho \leftarrow$ Select min. σ^2 from $\hat{D}^k$
10: **Return** k, ρ

4.2 Experiments

In this section, we demonstrate IBUG's ability to produce competitive probabilistic and point predictions as compared to current state-of-the-art methods on a large set of regression datasets (§4.2.2, §4.2.3). Then, we show that IBUG can use different base models to improve probabilistic performance (§4.2.4), flexibly model the posterior distribution (§4.2.5), and use approximations to speed up predictions while maintaining competitive performance (§4.2.7).

4.2.1 Implementation and Reproducibility. We implement IBUG in Python, using Cython—a Python package allowing the development of C extensions—to store a unified representation of the model structure. IBUG currently supports all modern gradient boosting frameworks including XGBoost [39], LightGBM [118], and CatBoost [159]. Experiments are run on publicly available datasets using an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.6GHz with 60GB of RAM @ 2.4GHz. Links to all data sources as well as the code for IBUG and all experiments is currently available online.[1]

[1]`https://github.com/jjbrophy47/ibug`.

4.2.2 Methodology.

We now compare IBUG's probabilistic and point prediction to NGBoost [61] and PGBM [195] on 21 benchmark regression datasets and one synthetic dataset. Additional dataset details are in §B.1.2

Metrics. We compute the average continuous ranked probability score (CRPS ↓) and negative log likelihood (NLL ↓) [84, 226] over the test set to evaluate probabilistic performance. To evaluate point performance, we use root mean squared error (RMSE ↓). For all metrics, lower is better. See §B.1 for detailed descriptions.

Protocol. We follow a similar protocol to Sprangers et al. [195] and Duan et al. [61]. We use 10-fold cross-validation to create 10 90/10 train/test folds for each dataset. For each fold, the 90% training set is randomly split into an 80/20 train/validation set to tune any hyperparameters. Once the hyperparameters are tuned, the model is retrained using the entire 90% training set. For probabilistic predictions, a normal distribution is used to model the output.

Significance testing. To determine which of two methods performs better on a given dataset under a given metric, we use a two-tailed paired t-test over the 10 random folds with a significance level of $p < 0.05$. We also report counts of the number of datasets in which a given method performed significantly better ("Win"), worse ("Loss"), or not different ("Tie") relative to a comparator.

Hyperparameters. We tune NGBoost the same way as in Duan et al. [61]. Since PGBM, CBU, and IBUG both optimize a point prediction metric, we tune their hyperparameters similarly. We also tune variance calibration parameters γ

and δ for each method (§4.1.2). Exact values evaluated and selected are in §B.1.3.

Unless specified otherwise, we use CatBoost [159] as the base model for IBUG.

Table 2. Probabilistic (CRPS) performance for each method on each dataset. Lower is better. Normal distributions are used for all probabilistic predictions. Results are averaged over 10 folds, and standard errors are shown in subscripted parentheses. The best method for each dataset is bolded, as well as those with standard errors that overlap the best method. *Bottom row*: Head-to-head comparison between IBUG/IBUG+CBU and each method showing the number of wins, ties, and losses (W-T-L) across all datasets. On average, IBUG+CBU provides the most accurate probabilistic predictions.

Dataset	NGBoost	PGBM	CBU	IBUG	IBUG+CBU
Ames	$38346_{(547)}$	$10872_{(355)}$	$11008_{(330)}$	$\mathbf{10434}_{(367)}$	$\mathbf{10194}_{(368)}$
Bike	$12.4_{(0.955)}$	$1.183_{(0.041)}$	$0.833_{(0.036)}$	$0.974_{(0.048)}$	$\mathbf{0.766}_{(0.032)}$
California	$1e11_{(1e11)}$	$0.222_{(0.001)}$	$0.217_{(0.001)}$	$0.213_{(0.001)}$	$\mathbf{0.207}_{(0.001)}$
Communities	$0.068_{(0.002)}$	$0.068_{(0.002)}$	$0.067_{(0.002)}$	$\mathbf{0.065}_{(0.002)}$	$\mathbf{0.065}_{(0.002)}$
Concrete	$3.410_{(0.182)}$	$1.927_{(0.086)}$	$\mathbf{1.788}_{(0.077)}$	$1.849_{(0.098)}$	$\mathbf{1.741}_{(0.082)}$
Energy	$0.519_{(0.043)}$	$\mathbf{0.147}_{(0.006)}$	$0.196_{(0.009)}$	$\mathbf{0.143}_{(0.009)}$	$0.157_{(0.008)}$
Facebook	$4.022_{(0.099)}$	$3.554_{(0.095)}$	$3.211_{(0.059)}$	$3.073_{(0.066)}$	$\mathbf{2.977}_{(0.070)}$
Kin8nm	$0.095_{(0.001)}$	$0.061_{(0.001)}$	$0.057_{(0.001)}$	$\mathbf{0.051}_{(0.001)}$	$\mathbf{0.051}_{(0.001)}$
Life	$2.897_{(1.465)}$	$0.815_{(0.027)}$	$0.772_{(0.024)}$	$0.794_{(0.023)}$	$\mathbf{0.731}_{(0.022)}$
MEPS	$\mathbf{5.527}_{(0.196)}$	$6.448_{(0.092)}$	$6.050_{(0.109)}$	$6.150_{(0.114)}$	$6.016_{(0.113)}$
MSD	$4.524_{(0.005)}$	$4.576_{(0.005)}$	$4.363_{(0.004)}$	$4.410_{(0.005)}$	$\mathbf{4.347}_{(0.004)}$
Naval	$0.003_{(0.000)}$	$0.000_{(0.000)}$	$0.000_{(0.000)}$	$0.000_{(0.000)}$	$\mathbf{0.000}_{(0.000)}$
News	$\mathbf{2191}_{(47.5)}$	$2361_{(52.6)}$	$2346_{(52.6)}$	$2545_{(41.0)}$	$2380_{(52.1)}$
Obesity	$3.208_{(0.028)}$	$1.860_{(0.022)}$	$\mathbf{1.740}_{(0.017)}$	$1.866_{(0.021)}$	$1.771_{(0.019)}$
Power	$2.105_{(0.023)}$	$1.531_{(0.019)}$	$\mathbf{1.473}_{(0.022)}$	$1.542_{(0.020)}$	$\mathbf{1.471}_{(0.021)}$
Protein	$5427_{(5409)}$	$1.823_{(0.011)}$	$1.788_{(0.009)}$	$1.784_{(0.008)}$	$\mathbf{1.742}_{(0.009)}$
STAR	$132_{(1.589)}$	$\mathbf{131}_{(1.380)}$	$\mathbf{130}_{(1.283)}$	$\mathbf{130}_{(1.214)}$	$\mathbf{129}_{(1.198)}$
Superconductor	$2.405_{(0.028)}$	$\mathbf{0.126}_{(0.004)}$	$0.150_{(0.004)}$	$0.153_{(0.006)}$	$\mathbf{0.128}_{(0.004)}$
Synthetic	$5.779_{(0.042)}$	$\mathbf{5.737}_{(0.039)}$	$\mathbf{5.739}_{(0.040)}$	$\mathbf{5.731}_{(0.040)}$	$\mathbf{5.730}_{(0.040)}$
Wave	$571020_{(883)}$	$3891_{(73.9)}$	$2349_{(10.3)}$	$2679_{(16.0)}$	$\mathbf{2026}_{(9.538)}$
Wine	$0.385_{(0.005)}$	$\mathbf{0.323}_{(0.005)}$	$0.337_{(0.006)}$	$\mathbf{0.322}_{(0.006)}$	$\mathbf{0.321}_{(0.006)}$
Yacht	$1.177_{(0.158)}$	$\mathbf{0.292}_{(0.042)}$	$\mathbf{0.281}_{(0.048)}$	$\mathbf{0.276}_{(0.048)}$	$\mathbf{0.255}_{(0.046)}$
IBUG W-T-L	17-3-2	11-9-2	9-5-8	-	1-6-15
IBUG+CBU W-T-L	17-3-2	15-6-1	18-2-2	15-6-1	-

4.2.3 Probabilistic and Point Predictions.

We first compare IBUG's probabilistic and point predictions to each baseline on each dataset. See Table 2 for detailed CRPS results; for brevity, results for additional probabilistic

metrics (e.g., NLL) as well as point performance results are in §B.1.4. Our main findings are as follows:

- On probabilistic performance, IBUG performs equally well or better than NGBoost and PGBM, winning on 17 and 11 (out of 22) datasets respectively, while losing on only 2 and 2 (respectively). Since CBU and IBUG performance is similar, we combine the two approaches, averaging their outputs; we denote this simple ensemble *IBUG+CBU*. Surprisingly, IBUG+CBU works very well, losing on only a maximum of 2 datasets when faced head-to-head against any other method; these results suggest IBUG and CBU are complimentary approaches.

- On point performance, PGBM, CBU, and IBUG performed significantly better than NGBoost; this is consistent with previous work and is perhaps unsurprising since NGBoost is optimized for probabilistic performance, not point performance. However, IBUG generally performed better than PGBM, winning on 13 datasets and losing on only 1 dataset; and performed slightly better than CBU, winning on 6 datasets with no losses.

We also compare IBUG with two additional baselines—kNN and BART [42]—shown in §B.2.2–B.2.3. We find IBUG generally outperforms these methods in both probabilistic and point performance. Overall, the results in this section suggest IBUG generates both competitive probabilistic and point predictions compared to existing methods.

4.2.4 Different Base Models. Here we experiment using different base models for IBUG besides CatBoost [159]; specifically, we use LightGBM [118] and XGBoost [39], two popular gradient boosting frameworks. Table 3 shows that

Table 3. Probabilistic (CRPS, NLL) performance on the test set for IBUG using different base models. Results are averaged over 10 folds, and standard errors are shown in subscripted parentheses; lower is better. On 6 and 5 datasets, respectively, either IBUG-LightGBM or IBUG-XGBoost significantly outperforms IBUG-CatBoost on the validation set and subsequently on the test set, demonstrating the potential for improved probabilistic performance by using IBUG with different base models.

Dataset	Test CRPS (↓) CatBoost	LightGBM	XGBoost
Bike	$0.974_{(0.048)}$	$\mathbf{0.819}_{(0.024)}$	$\mathbf{0.849}_{(0.012)}$
MSD	$4.410_{(0.005)}$	$\mathbf{4.372}_{(0.005)}$	$4.418_{(0.005)}$
News	$2545_{(41.0)}$	$\mathbf{2436}_{(50.8)}$	$2551_{(56.0)}$
Power	$1.542_{(0.020)}$	$1.536_{(0.022)}$	$\mathbf{1.518}_{(0.018)}$
Protein	$1.784_{(0.008)}$	$\mathbf{1.683}_{(0.009)}$	$1.788_{(0.008)}$
Supercon.	$0.153_{(0.006)}$	$\mathbf{0.090}_{(0.005)}$	$\mathbf{0.010}_{(0.003)}$

Dataset	Test NLL (↓) CatBoost	LightGBM	XGBoost
Bike	$1.886_{(0.056)}$	$\mathbf{1.292}_{(0.048)}$	$\mathbf{1.662}_{(0.024)}$
MSD	$3.415_{(0.002)}$	$\mathbf{3.409}_{(0.002)}$	$\mathbf{3.402}_{(0.002)}$
Naval	$-6.208_{(0.010)}$	$\mathbf{-6.281}_{(0.007)}$	$-5.853_{(0.014)}$
Obesity	$2.646_{(0.009)}$	$\mathbf{2.593}_{(0.016)}$	$\mathbf{2.624}_{(0.010)}$
Supercon.	$0.783_{(0.181)}$	$\mathbf{-0.496}_{(0.169)}$	$20.4_{(23.2)}$

using a different base model can result in improved probabilistic performance. This highlights IBUG's agnosticism to GBRT type, enabling practitioners to apply IBUG to future models with improved point prediction performance.

4.2.5 Posterior Modeling. One of the unique benefits of IBUG is the ability to directly model the output using empirical samples (Figure 8), giving practitioners a better sense of the output distribution for specific predictions. IBUG can optimize a distribution *after* training, and has more flexibility in the types of distributions it can model—from distributions using just location and scale to those with high-order moments as well as non-parametric density estimators. To test this flexibility, we model each probabilistic prediction using the following distributions: normal, skewnormal, lognormal, Laplace, student t, logistic, Gumbel, Weibull, and KDE; we then select the distribution with the best average NLL on the validation set, and evaluate its probabilistic performance on the test set.

Figure 8 demonstrates that the selected distributions for the MEPS and Wine datasets achieve better probabilistic performance than assuming normality. Qualitatively, the empirical densities of $A^{(k)}$ for a randomly sampled set of test

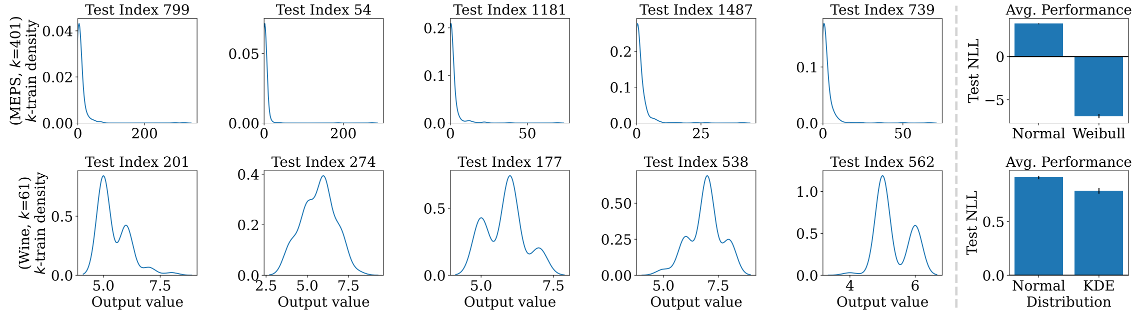


Figure 8. Left: Distribution of the k-nearest training instances for 5 randomly-selected test instances from the MEPS (top) and Wine (bottom) datasets. *Right*: Test NLL (with standard error) when modeling the posterior using two different distributions (lower is better). IBUG can model parametric *and* non-parametric distributions that better fit the underlying data than assuming normality.

instances reaffirms the selected distributions. As an additional comparison, we report CBU achieves a test NLL of $3.699_{\pm 0.038}$ and $1.025_{\pm 0.028}$ for the MEPS and wine datasets (respectively) using a normal distribution, while IBUG achieves $-6.887_{\pm 0.260}$ and $0.785_{\pm 0.025}$ using Weibull and KDE estimation (respectively). For the MEPS dataset, the selected Weibull distribution takes a shape parameter, which IBUG estimates directly on a *per prediction* basis using $A^{(k)}$and MLE. In contrast, PGBM or CBU would need to optimize a global shape value using a validation set, which is likely to be suboptimal for individual predictions.

4.2.6 Variance Calibration. Table 4 shows probabilistic performance comparisons of each method against itself with and without variance calibration. In all cases, variance calibration (§4.1.2) either maintains or improves performance for all methods, especially CBU. Overall, these results suggest that variance calibration should be a standard procedure for probabilistic prediction, unless using a method that has particularly well-calibrated predictions to begin with. We therefore use variance calibration in all of our results.

Table 4. Probabilistic performance comparison of each method with vs. without variance calibration. In all cases, calibration maintains or improves performance; it is especially helpful for CBU.

	CRPS			NLL		
Method	Wins	ties	Losses	Wins	Ties	Losses
NGBoost	9	13	0	1	21	0
PGBM	13	9	0	11	11	0
CBU	17	5	0	11	11	0
IBUG	13	9	0	5	17	0

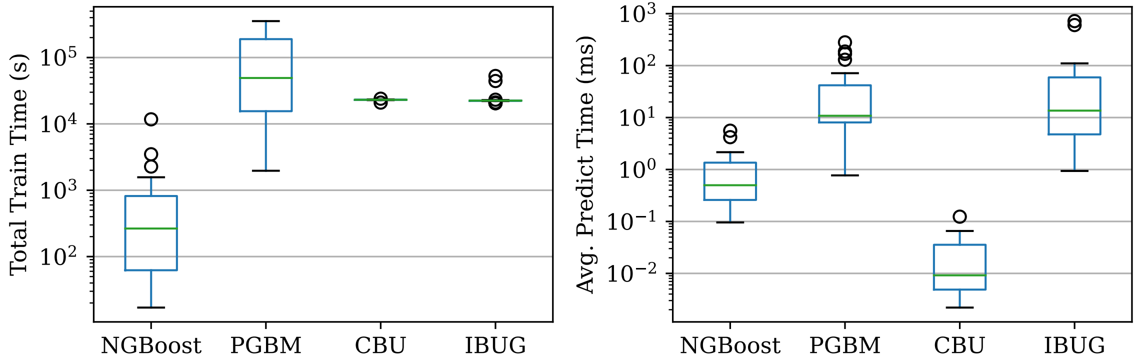


Figure 9. Runtime comparison. *Left*: Total train time (including tuning). *Right*: Average prediction time per test example. Results are shown for all datasets, averaged over 10 folds (exact values are in §B.1.5, Tables B.9 and B.10). On average, IBUG has comparable training times to PGBM and CBU, but is relatively slow for prediction.

Additionally, §B.2.1 shows performance results for all methods *without* variance calibration. Overall, we observe similar trends as when applying calibration (Table 2).

4.2.7 Sampling Trees. Figure 9 shows the runtime for each method broken down into total training time (including tuning) and prediction time per test example. On average, IBUG has similar training times to PGBM and CBU, but on some datasets, IBUG is roughly an order of magnitude faster than

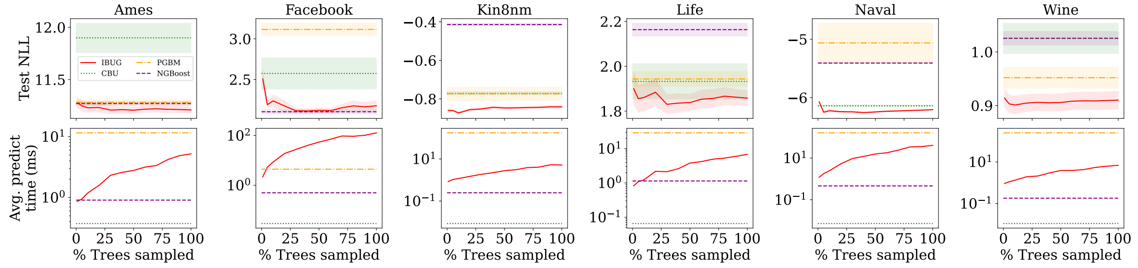


Figure 10. Change in probabilistic (NLL) performance (top) and average prediction time (in seconds) per test example (bottom) as a function of τ for six datasets with trees sampled *first-to-last*; lower is better. NGBoost, PGBM, and CBU are added for additional context. The shaded regions represent the standard error. Overall, average prediction time decreases significantly as τ decreases while test NLL often remains relatively stable, enabling IBUG to generate probabilistic predictions with significant increased efficiency.

PGBM. For predictions, IBUG is similar to PGBM but relatively slow compared to NGBoost and CBU.

However, by sampling $\tau < T$ trees when computing the affinity vector, IBUG can significantly reduce prediction time. Figure 10 shows results when sampling trees first-to-last, which typically works best over all tree-sampling strategies (alternate sampling strategies are evaluated in §B.2.4). As τ decreases, we observe average prediction time decreases roughly 1-2 orders of magnitude while probabilistic performance remains relatively stable until τ/T reaches roughly 1–5%, at which point probabilistic performance sometimes starts to decrease more rapidly. Note for the Ames and Life datasets, IBUG can reach the same average prediction time as NGBoost while maintaining the same or better probabilistic performance than NGBoost, PGBM, and CBU. These results demonstrate that if speed is a concern, IBUG can approximate the affinity computation to speed up prediction times while maintaining competitive probabilistic performance.

4.3 Summary

IBUG uses ideas from instance-based learning to enable probabilistic predictions for *any* GBRT point predictor. IBUG generates probabilistic predictions by using the k-nearest training instances to the test instance found using the structure of the trees in the ensemble. Our results on 22 regression datasets demonstrate this simple wrapper produces competitive probabilistic and point predictions to current state-of-the-art methods, most notably NGBoost [61], PGBM [195], and CBU [143]. We also show that IBUG can more flexibly model the posterior distribution of a prediction using any parametric *or* non-parametric density estimator. IBUG's one limitation is relatively slow prediction time. However, we show that approximations in the search for the k-nearest training instances can significantly speed up prediction time; predictions are also easily parallelizable in IBUG.

CHAPTER 5

EFFICIENT MODEL ADAPTATION

J. Brophy and D. Lowd. Machine Unlearning for Random Forests. In Proceedings of the Thirty-Eighth International Conference on Machine Learning. Virtual. 2021.

Current work on deleting data from machine learning models has focused mainly on recommender systems [34, 174], K-means [82], SVMs [36], logistic regression [91, 174], and deep neural networks [14, 85, 220]; however, there is very limited work addressing the problem of efficient data deletion for tree-based models [175]. Thus, in this chapter, we focus on decision trees and random forests [26, 75] because of their popularity and wide-spread use [138] on many classification and regression tasks [18, 78, 123, 135, 217], and because of their simpler training regime compared to GBDTs.

We introduce DaRE (**Da**ta **R**emoval-**E**nabled) Forests (a.k.a DaRE RF), a variant of random forests that supports the efficient removal of training instances. DaRE RF works with discrete tree structures, in contrast to many related works on efficient data deletion that assume continuous parameters. The key components of DaRE RF are to retrain subtrees only as needed, consider only a subset of valid thresholds per attribute at each decision node, and to strategically place completely random nodes near the top of each tree to avoid costly retraining. 2) We provide algorithms for training and subsequently removing data from a DaRE forest. 3) We evaluate DaRE RF's ability to efficiently perform sequences of deletions on 13 real-world binary classification datasets and one synthetic dataset, and find that DaRE RF can typically delete data 2-4 orders of magnitude faster than retraining from scratch while sacrificing less than 1% in terms of predictive performance.

Random Forests. We base our methods on a minor variation of a standard RF, one that does not use bootstrapping. Bootstrapping complicates the removal of training instances, since one instance may appear multiple times in the training data for one tree. There is also empirical evidence that bootstrapping does not improve predictive performance [54, 145, 225], which was consistent with our own experiments (Appendix: §C.2.2, Table C.2). Since predictive performance was already similar, we saw no need to add the extra bookkeeping to handle this complexity.

5.1 DaRE Forests

We now describe DaRE (**Da**ta **R**emoval-**E**nabled) forests (a.k.a. DaRE RF), an RF variant that enables the efficient removal of training instances.

Theorem 5.1.1. *Data deletion for DaRE forests is* exact *(see Eq. 2.12), meaning that removing instances from a DaRE model yields exactly the same model as retraining from scratch on updated data.*

This is also equivalent to certified removal [91] with $\epsilon = 0$. Proofs to all theorems are in §C.1 of the Appendix.

A DaRE forest is a tree ensemble in which each tree is trained independently on a copy of the training data, considering a random subset of $\tilde{p}$ attributes at each split to encourage diversity among the trees. In our experiments we use $\tilde{p} = \lfloor \sqrt{p} \rfloor$. Since each tree is trained independently, we describe our methods in terms of training and updating a single tree; the extension to the ensemble is trivial.

DaRE forests leverage several techniques to make deletions efficient: (1) only retrain portions of the model where the structure must change to match the updated database; (2) consider at most k randomly-selected thresholds per attribute; (3) introduce random nodes at the *top* of each tree that minimally

depend on the data and thus rarely need to be retrained. We present abridged versions for training and updating a DaRE tree in Algorithms 1 and 2, respectively, with full explanations below. Detailed pseudocode for both operations is in the Appendix, §C.1.8.

5.1.1 Retraining Minimal Subtrees. We avoid unnecessary retraining by storing statistics at each node in the tree. For decision nodes, we store and update counts for the number of instances $|D|$ and positives instances $|D_{\cdot,1}|$, as well as $|D_\ell|$ and $|D_{\ell,1}|$ for a set of k thresholds per attribute. This information is sufficient to recompute the split criterion of each threshold without iterating through the data. For leaf nodes, we store and update $|D|$ and $|D_{\cdot,1}|$, along with a list of training instances that end at that leaf. These statistics are initialized when training the tree for the first time (Alg. 4). We find this additional overhead has a negligible effect on training time.

When deleting a training instance $(x, y) \in D$, these statistics are updated and used to check if a particular subtree needs retraining. Specifically, decision nodes affected by the deletion of (x, y) update the statistics and recompute the split criterion for each attribute-threshold pair. If a different threshold obtains an improved split criterion over the currently chosen threshold, then we retrain the subtree rooted at this node. The training data for this subtree can be found by concatenating the instance lists from all leaf-node descendants. If no retraining occurs at any decision node and a leaf node is reached instead, its label counts and instance list are updated and the deletion operation is complete. See Alg. 5 for pseudocode.

5.1.2 Sampling Valid Thresholds. The optimal threshold for a continuous attribute will always lie between two training instances with adjacent

Algorithm 4 Building a DaRE tree / subtree.

```
 1: Input: data D, depth d
 2: if stopping criteria reached then
 3:    node ← LeafNode()
 4:    save instance counts(node, D)                    ▷ Pos./neg. counts: |D|, |D.,1|
 5:    save leaf-instance pointers(node, D)
 6:    compute leaf value(node)
 7: else
 8:    if d < d_rmax then
 9:       node ← RandomNode()
10:       save instance counts(node, D)                 ▷ Pos./neg. counts: |D|, |D.,1|
11:       a ← randomly sample attribute(D)
12:       v ← randomly sample threshold ∈ [a_min, a_max)
13:       save threshold statistics(node, D, a, v)      ▷ Left/right counts: |D_ℓ|, |D_r|
14:    else
15:       node ← GreedyNode()
16:       save instance counts(node, D)                 ▷ Pos./neg. counts: |D|, |D.,1|
17:       A ← randomly sample p̃ attributes(D)
18:       for a ∈ A do
19:          C ← get valid thresholds(D, a)
20:          V ← randomly sample k valid thresholds(C)
21:          for v ∈ V do
22:             save threshold statistics(node, D, a, v)
23:       scores ← compute split scores(node)
24:       select optimal split(node, scores)
25:    D.ℓ, D.r ← split on selected threshold(node, D)
26:    node.ℓ = Train(D_ℓ, d + 1)                       ▷ Algorithm 4
27:    node.r ← Train(D_r, d + 1)                       ▷ Algorithm 4
28: Return node
```

feature values containing opposite labels; if the two training instances have the same label, the split criterion improves by increasing or decreasing v. We refer to these as *valid* thresholds, and any other threshold as *invalid*. More precisely, a threshold v between two adjacent values v_1 and v_2 for a given attribute a is valid if and only if there exist instances (x_1, y_1) and (x_2, y_2) such that $x_{1,a} = v_1$, $x_{2,a} = v_2$, and $y_1 \neq y_2$.

Only considering valid thresholds substantially reduces the statistics we need to store and compute at each node. We gain further efficiency by randomly

sampling k valid thresholds and only considering these thresholds when deciding which attribute-threshold pair to split on. We treat k as a hyperparameter and tune its value when building a DaRE model. One might suspect that only considering a subset of thresholds for each attribute may lead to decreased predictive performance; however, our experiments show that relatively modest values of k (e.g. $5 \leq k \leq 25$) are sufficient to providing accurate predictions, and in some cases lead to improved performance (Appendix: §C.2.2, Table C.2).

When deleting an instance at a given node, we must determine if any threshold has become invalid. To accomplish this efficiently, at each node we also save and update the number of instances in which attribute a equals v_1, the number in which a equals v_2, and the number of positive instances matching each of those criteria. When an attribute threshold becomes invalid, we sort and iterate through the node data D, resampling the invalid threshold to obtain a new valid threshold.

5.1.3 Random Splits. The third technique for efficient model updating is to choose the attribute and threshold for some of the decision nodes at random, independent of the split criterion. Specifically, given the data at a particular decision node $D \subseteq \mathcal{D}$, we sample an attribute $a \in P$ uniformly at random, and then sample a threshold v in the range $[a_{\min}, a_{\max})$, the min. and max. values for a in D. We henceforth refer to these decision nodes as "random" nodes, in contrast to the "greedy" decision nodes that optimize the split criterion. Random nodes store and update $|D_\ell|$ and $|D_r|$, statistics based on the sampled threshold, and retrain only if $|D_\ell| = 0$ or $|D_r| = 0$ (i.e. v is no longer in the range $[a_{\min}, a_{\max})$); however, since random nodes minimally depend on the statistics of the data, they rarely need to be retrained. Random nodes are placed in the upper layers of the tree and greedy nodes are used for all other layers (excluding leaf

Algorithm 5 Deleting a training instance from a DaRE tree.

Require: Start at the root node.

```
 1: Input: node, depth d, instance to remove (x, y).
 2: update instance counts(node, (x, y))               ▷ Pos./neg. counts: |D| and |D_{·,1}|
 3: if node is a LeafNode then
 4:    remove (x, y) from leaf-instance pointers(node, (x, y))
 5:    recompute leaf value(node)
 6:    remove (x, y) from database and return
 7: else
 8:    update decision node statistics(node, (x, y))
 9:    if node is a RandomNode then
10:       if node.selected threshold is invalid then
11:          D ← get data from leaf instances(node) \ (x, y)
12:          if node.selected attribute (a) is not constant then
13:             v ← resample threshold ∈ [a_min, a_max)
14:             D.ℓ, D.r ← split on new threshold(node, D, a, v)
15:             node.ℓ, r ← Train(D.ℓ, d + 1), Train(D.r, d + 1)
16:          else
17:             node ← Train(D, d)                                   ▷ Algorithm 4
18:          remove (x, y) from database and return
19:    else
20:       if ∃ invalid attributes or thresholds then
21:          D ← get data from leaf instances(node) \ (x, y)
22:          resample invalid attributes and thesholds(node, D)
23:       scores ← recompute split scores(node)
24:       a, v ← select optimal split(node, scores)
25:       if optimal split has changed then
26:          D.ℓ, D.r ← split on new threshold(node, D, a, v)
27:          node.ℓ, r ← Train(D.ℓ, d + 1), Train(D.r, d + 1)
28:          remove (x, y) from database and return
29:    if x_{·,a} ≤ v then
30:       Delete(node.ℓ, d + 1, (x, y))                              ▷ Algorithm 5
31:    else
32:       Delete(node.r, d + 1, (x, y))                              ▷ Algorithm 5
```

nodes). We introduce d_{rmax} as another hyperparameter indicating how many layers from the top the tree should use for random nodes (e.g. the top two layers of the tree are all random nodes if d_{rmax}= 2).

Intuitively, nodes near the top of the tree contain more instances than nodes near the bottom, making them more expensive to retrain if necessary. Thus, we

can significantly increase deletion efficiency by replacing those nodes with random ones. We can also maintain comparable predictive performance to a model with no random nodes by using greedy nodes in all subsequent layers, resulting in a greedy model built on top of a random projection of the input space [101].

In our experiments, we compare DaRE RF with random splits to those without, to evaluate the benefits of adding these random nodes. We refer to DaRE models with random nodes as random DaRE (R-DaRE) and those without as greedy DaRE (G-DaRE). G-DaRE RF can also be viewed as a special case of R-DaRE RF in which $d_{\text{rmax}} = 0$.

5.1.4 Complexity Analysis.

The time for training a DaRE forest is *identical* to that of a standard RF:

Theorem 5.1.2. *Given $n = |\mathcal{D}|$, T, $d_{\max}$, and $\tilde{p}$, the time complexity to train a DaRE forest is $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$.*

The overhead of storing statistics and instance pointers is negligible compared to the cost of iterating through the entire dataset to score all attributes at each node. The key difference is in the deletion time, which can be much better depending on how much of each tree needs to be retrained:

Theorem 5.1.3. *Given $d_{\max}$, $\tilde{p}$, and k, the time complexity to delete a single instance $(x, y) \in \mathcal{D}$ from a DaRE tree is $\mathcal{O}(\tilde{p}\,k\,d_{\max})$, if the tree structure is unchanged and the attribute thresholds remain valid. If a node with $|D|$ instances has invalid attribute thresholds, then the additional time to choose new thresholds is $\mathcal{O}(|D| \log |D|)$. If a node with $|D|$ instances at level d needs to be retrained, then the additional retraining time is $\mathcal{O}(\tilde{p}\,|D|\,(d_{\max} - d)))$.*

When the structure is unchanged, this is much more efficient than naive retraining, especially if the number of thresholds considered (k) is much smaller

than n. In the worst case, if the split changes at the root of every tree, then deletion in a DaRE forest is no better than naive retraining. In practice, this is very unlikely, since different trees in the forest consider different sets of $\tilde{p}$ attributes at the root, and the difference between the best and second-best attribute-threshold pairs is usually bigger than a single instance.

Choosing new thresholds also requires iterating through the training instances at a node. Thresholds only become invalid when an instance adjacent to the threshold is removed, so this is an infrequent event when k is much smaller than n. To analyze this empirically, we evaluate our methods with both random and adversarially chosen deletions, approximating the average- and worst-case, respectively.

The main storage costs for a DaRE forest come from storing sets of attribute-threshold statistics at each greedy node, and the instance lists for the leaf nodes.

Theorem 5.1.4. *Given $n = |\mathcal{D}|$, $d_{\max}$, k, T, and $\tilde{p}$, the space complexity of a DaRE forest is $\mathcal{O}(k\,\tilde{p}\,2^{d_{\max}}\,T + n\,T)$.*

In our experiments, we analyze the space overhead of a DARE forest by measuring its memory consumption as compared to a standard RF, quantifying the time/space trade-off introduced by DARE RF to enable efficient data deletion.

5.2 Experiments

Here we empirically evaluate DaRE RF and attempt to answer the following research questions. Can we use G-DaRE RF to efficiently delete a significant number of instances as compared to naive retraining (**RQ1**)? Can we use R-DaRE RF to further increase deletion efficiency while maintaining comparable predictive performance (**RQ2**)?

5.2.1 Datasets. We conduct our experiments on 13 publicly-available datasets that represent problems well-suited for tree-based models, and one synthetic dataset we call Synthetic. For each dataset, we generate one-hot encodings for any categorical variable and leave all numeric and binary variables as is. For any dataset without a designated train and test split, we randomly sample 80% of the data for training and use the rest for testing. A summary of the datasets is in Table 5, and additional dataset details are in the Appendix: §C.2.1.

5.2.2 Hyperparameter Tuning. Due to the range of label imbalances in our datasets (Table 5 and Appendix: §C.2.1, Table C.1) we measure the predictive peformance of our models using average precision (AP) [229] for datasets with a positive label percentage $< 1\%$, AUC [98] for datasets between $[1\%, 20\%]$, and accuracy (acc.) for the remaining datasets. Using these metrics and Gini index as the split criterion, we tune the following hyperparameters: the maximum depth of each tree d_{max}, the number of trees in the forest T, and the number of thresholds considered per attribute for greedy nodes k. Our protocol for tuning d_{rmax} is as follows: first, we tune a greedy model (i.e. by keeping $d_{\text{rmax}} = 0$ fixed) using 5-fold cross-validation. Once the optimal values for d_{max}, T, and k are found, we tune d_{rmax} by incrementing its value from zero to d_{max}, stopping when the model's cross-validation score exceeds a specified error tolerance as compared to the greedy model; for these experiments, we tune d_{rmax} using absolute error tolerances of 0.1%, 0.25%, 0.5%, and 1.0%. Selected hyperparameter values are in the Appendix: §C.2.2, Table C.3.

5.2.3 Methodology. We measure relative efficiency or speedup as the number of instances a DaRE model deletes in the time it takes the naive retraining approach to delete one instance (i.e. retrain without that instance); the number

Table 5. Dataset Summary. n = no. instances, p = no. attributes, % Positive = positive label percentage, Metric = predictive performance metric.

Dataset	n	p	**% Positive**	**Metric**
Surgical	14,635	90	25.2%	Accuracy
Vaccine	26,707	185	46.4%	Accuracy
Adult	48,842	107	23.9%	Accuracy
Bank Mktg.	41,188	63	11.3%	AUC
Flight Delays	100,000	648	19.0%	AUC
Diabetes	101,766	253	46.1%	Accuracy
No Show	110,527	99	20.2%	AUC
Olympics	206,165	1,004	14.6%	AUC
Census	299,285	408	6.2%	AUC
Credit Card	284,807	29	0.2%	AP
CTR	1,000,000	13	2.9%	AUC
Twitter	1,000,000	15	17.0%	AUC
Synthetic	1,000,000	40	50.0%	Accuracy
Higgs	11,000,000	28	53.0%	Accuracy

of instances deleted gives us the speedup over the the naive approach.[1] We also measure the predictive performance of R-DaRE RF prior to deletion and compare its predictive performance to that of G-DaRE RF. Each experiment is repeated five times.

We determine the order of deletions using two different adversaries: *Random* and *Worst-of-1000*. The random adversary selects training instances to be deleted uniformly at random, while the worst-of-1000 adversary selects each instance by first selecting 1,000 candidate instances uniformly at random, and then choosing the instance that results in the most retraining, as measured by the total number of instances assigned to all retrained nodes across all trees.

5.2.4 Deletion Efficiency Results. Here we present the results of G-DaRE RF and R-DaRE RF against the random and worst-of-1000 adversaries.

[1] System hardware specifications are in the Appendix: §C.2.

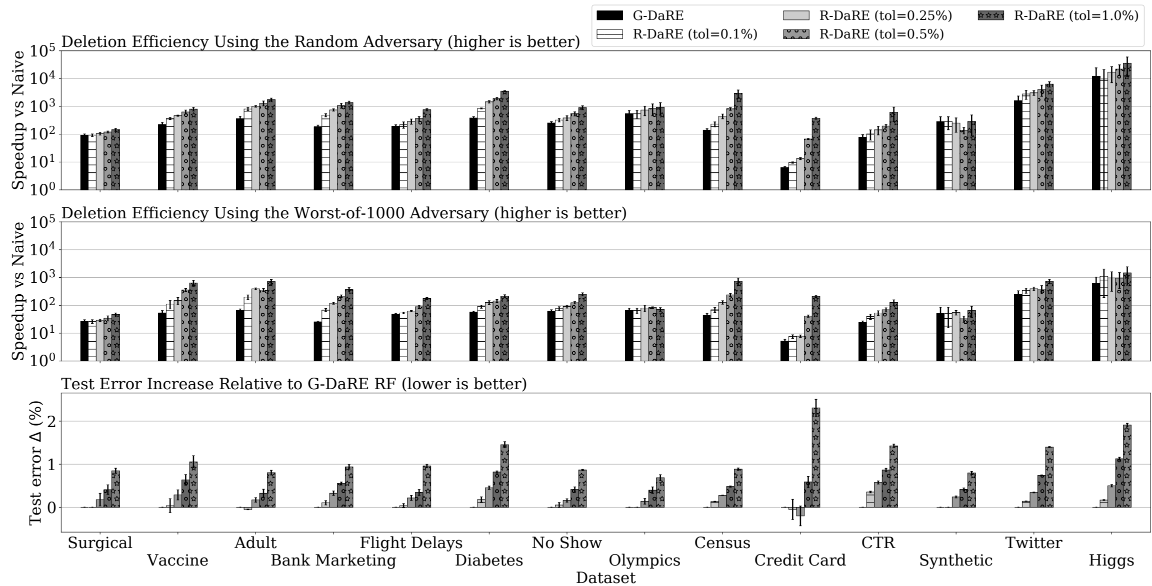


Figure 11. Deletion efficiency of DaRE RF. *Top & Middle*: Number of instances deleted in the time it takes the naive retraining approach to delete one instance using the random and worst-of-1000 adversaries, respectively (error bars represent standard deviation). *Bottom*: The increase in test error when using R-DaRE RF relative to the predictive performance of G-DaRE RF (error bars represent standard error).

Random Adversary. We present the results of the deletion experiments using the random adversary in Figure 11 (top). We find that G-DaRE RF is usually at least two orders of magnitude faster than naive retraining, while R-DaRE RF is faster than G-DaRE RF to a varying degree depending on the dataset and error tolerance. R-DaRE RF is also able to maintain comparable predictive performance to G-DaRE RF, typically staying within a test error difference of 1% depending on which tolerance is used to tune d_{rmax} (Figure 11: bottom).

As an example of DaRE RF's utility, naive retraining took 1.3 hours to delete a single instance for the Higgs dataset. R-DaRE RF ($tol = 0.25\%$ resulting in $d_{\text{rmax}}= 3$) deleted over 17,000 instances in that time, an average of 0.283s per deletion, while the average test set error increased by only 0.5%. In this case,

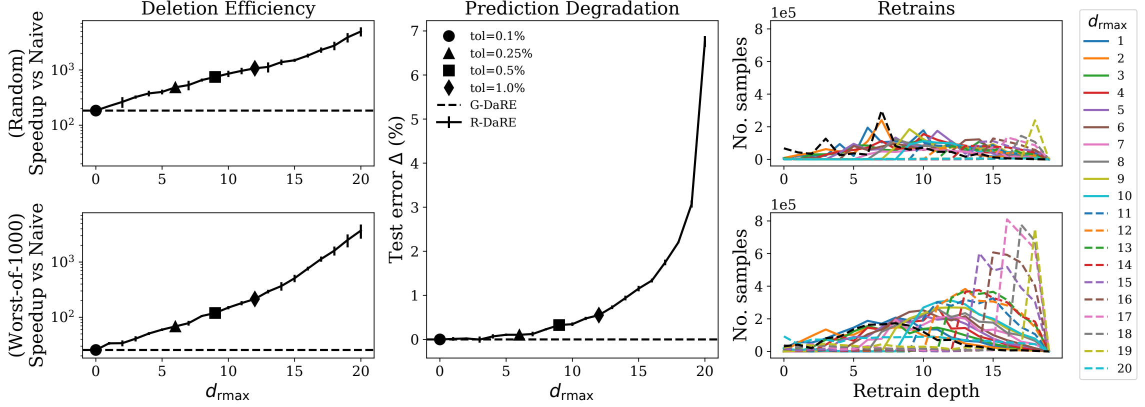


Figure 12. Effect of increasing d_{rmax}on deletion efficiency (left), predictive performance (middle), and the cost of retraining (right) using the random (top) and worst-of-1000 (bottom) adversaries for the Bank Marketing dataset. The predictive performance is independent of the adversary, as performance is measured before any deletions occur. Error bars represent standard deviation and standard error for the left and middle plots, respectively. In short, we see that increasing d_{rmax} increases deletion efficiency but initially gradually degrades predictive performance. Similar analysis for other datasets are in the Appendix: §C.2.3.

R-DaRE RF provides a speedup of over four orders of magnitude, providing a tractable solution for something previously intractable.

Worst-of-1000 Adversary. Against the more challenging worst-of-1000 adversary (Figure 11: middle), the speedup over naive deletion remains large, but is often an order of magnitude smaller. While R-DaRE models also decrease in efficiency, they maintain a significant advantage over G-DaRE RF, showing very similar trends of increased relative efficiency as when using the random adversary.

Summary. A summary of the deletion efficiency results is in Table 6. When instances to delete are chosen randomly, G-DaRE RF is more than 250x faster than naively retraining after every deletion (taking the geometric mean over the 14 datasets). By adding randomness, R-DaRE models achieve even larger speedups,

Table 6. Summary of the deletion efficiency results. Specifically, the minimum, maximum, and geometric mean of the speedup vs. the naive retraining method across all datasets.

Model	Minimum	Maximum	Geometric Mean
Random Adversary			
G-DaRE	6x	12,232x	257x
R-DaRE (tol=0.1%)	10x	9,735x	366x
R-DaRE (tol=0.25%)	13x	17,044x	494x
R-DaRE (tol=0.5%)	68x	22,011x	681x
R-DaRE (tol=1.0%)	145x	35,856x	1,272x
Worst-of-1000 Adversary			
G-DaRE	5x	626x	52x
R-DaRE (tol=0.1%)	8x	1,106x	79x
R-DaRE (tol=0.25%)	8x	961x	102x
R-DaRE (tol=0.5%)	33x	950x	139x
R-DaRE (tol=1.0%)	47x	1,476x	263x

from 360x to over 1,200x, depending on the predictive performance tolerance (0.1% to 1.0%). The more sophisticated worst-of-1000 adversary can force more costly retraining. In this case, G-DaRE RF is more than 50x faster than naive retraining, and R-DaRE RF ranges from 80x to 260x depending on the tolerance.

5.2.5 Effect of d_{rmax} and k on Deletion Efficiency. Figure 12 details the effect d_{rmax} has on deletion efficiency under each adversary for the Bank Marketing dataset[2]. As expected, we see that deletion efficiency increases as d_{rmax} increases. Predictive performance degrades as d_{rmax} increases, but initially degrades gracefully, maintaining a low increase in test error even as the top ten layers of each tree are replaced with random nodes (+0.346% test error).

Figure 12 also shows the number of instances retrained at each depth, across all trees in the model. We immediately notice the increase in retraining cost when switching from the random (top-right plot) to the worst-of-1000 (bottom-right

[2]Other datasets show similar trends; see the Appendix: §C.2.3.

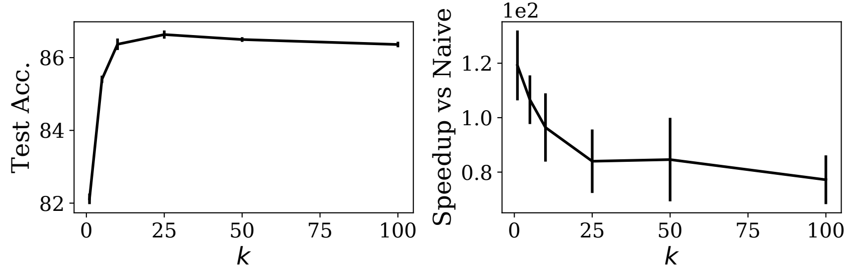


Figure 13. Effect of increasing k on predictive performance (left) and deletion efficiency (right) for the Surgical dataset using the random adversary; d_{rmax} is held fixed at 0. Error bars represent standard error and standard deviation for the left and right plots, respectively. Analysis for other datasets is in the Appendix: §C.2.4.

plot) adversary, especially at larger depths. This matches our intuition since nodes deeper in the tree have fewer instances; each instance thus has a larger influence on the resulting split criterion over all attributes at a given node and increases the likelihood that a chosen attribute may change, resulting in more subtree retraining.

Figure 13 shows the effect increasing k has on predictive performance and deletion efficiency for the Surgical dataset[3]. In general, we find k introduces a trade-off between predictive performance and deletion efficiency. However, our experiments show that modest values of k can achieve competitive predictive performance while maintaining a high degree of deletion efficiency and incurring low storage costs.

5.2.6 Space Overhead. This section shows the space overhead of DARE forests by breaking the memory usage of G-DARE RF into three constituent parts: 1) the structure of the model that is needed for making predictions, 2) the additional statistics stored at each decision node, and 3) the additional statistics and training-instance pointers stored at each leaf node. Parts 2) and 3), plus the

[3]Other datasets show similar trends; see the Appendix: §C.2.4.

Table 7. Memory usage (in megabytes) for the training data, G-DARE RF, and an SKLearn RF (SKRF) trained using the same values of T and d_{max} as G-DARE RF. The total memory usage for the G-DARE RF model is broken down into: 1) the structure of the model needed for making predictions (Structure); 2) the additional statistics stored at all decision nodes (Decisions); and 3) the additional statistics and training-instance pointers stored at all leaf node (Leaves). The space overhead for G-DARE RF to enable efficient data deletion is measured as a ratio of the total memory usage of (data + G-DARE RF) to (data + SKRF). Results are averaged over five runs and the standard error is shown in parentheses.

		G-DARE RF					
Dataset	**Data**	Structure	Decisions	Leaves	**Total**	**SKRF**	**Overhead**
Surgical	4	15	388	14	417	31	12.0x
Vaccine	16	18	426	14	458	37	8.9x
Adult	14	9	227	16	252	18	8.3x
Bank Mktg.	8	23	455	33	511	51	8.8x
Flight Delays	207	37	3,030	171	3,238	66	12.6x
Diabetes	83	125	4,968	199	5,292	257	15.8x
No Show	35	91	2,511	203	2,805	187	12.8x
Olympics	663	27	3,196	338	3,561	57	5.9x
Census	326	33	1,737	169	1,939	63	5.8x
Credit Card	27	5	105	457	567	7	17.5x
CTR	45	6	485	642	1,133	10	21.4x
Twitter	48	186	2,450	693	3,329	332	8.9x
Synthetic	131	128	5,661	357	6,146	114	25.6x
Higgs	1,021	935	39,416	3,787	44,138	1,325	19.3x

size of the data, constitute the space needed by G-DARE RF to enable efficient data removal.

Table 7 shows the space overhead of G-DARE RF after training. We also show the training set size for each dataset, and the total memory usage of an SKLearn RF model using the same values for T and d_{max} as G-DARE RF.

As expected, decision-node statistics often make up the bulk of the space overhead for G-DARE RF; two exceptions are the Credit Card and CTR datasets, in which the size of the training-instance pointers outweigh the relatively low number of decision nodes (an average of 238 and 726 per tree, respectively) for

those models. The total memory usage of the G-DARE RF *model* is 10-113x larger than that of the SKLearn RF model. However, since both approaches require the training data to enable deletions (G-DARE RF may need to retrain subtrees; SKLearn RF needs to retrain using the naive approach), the relative overhead of G-DARE RF is the ratio of (data + G-DARE RF) to (data + SKLearn RF); this results in an overhead of 6–26x, quantifying the time/space trade-off for efficient data deletion.

5.3 Summary

DaRE RF is a random forest variant that supports efficient model updates in response to repeated deletions of training instances. We find that, on average, DaRE models are 2-3 orders of magnitude faster than the naive retraining approach with no loss in accuracy, and additional efficiency can be achieved if slightly worse predictive performance is tolerated.

DaRF RF is a discrete tree-structured model, in contrast to previous unlearning works which assume continuous parameters. Data deletions in DaRE models are also exact, thus membership inference attacks [35, 224] are guaranteed to be unsuccessful for instances deleted from the model. DaRE models also reduce the need for deletion verification methods [187, 194]. However, one must be aware that DaRE models (as well as any unlearning method) can leak which instances are deleted if an adversary has access to the model before *and* after the deletion [38]. Although privacy is a strong motivator for this work, there are a number of other useful applications for DaRE forests and machine unlearning in general.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

Equipping tree ensembles with the tools we have developed in this dissertation as well as the potential future tools and insights gained from following the possible research directions listed here are likely to further increase the popularity and adoption of tree ensembles to many more important domains and applications.

6.1 Summary of Contributions

In this dissertation, we have examined and tackled significant shortcomings of the popular and widely-used class of tree-ensemble models. Namely, we have addressed three problem areas which modern tree-based ensembles face: (1) lack of explainability for individual predictions; (2) missing prediction uncertainty estimates for tree-based ensembles built for regression tasks; and (3) inefficient updates to models in response to changes in the training data (e.g., deletion requests).

We tackled (1) by adapting efficient influence-estimation methods designed for continuous deep learning models to discrete tree-based ensembles; our main approach, called *BoostIn*, efficiently provides competitive influential estimates to existing methods with orders of magnitude more efficiency (Chapter 3). For (2), we leverage the learned structure of the tree ensembles to identify a neighborhood of semantically similar training examples to a target example that can be used to accurately estimate the uncertainty of that target example prediction; we call this approach *IBUG*, and find it can accurately and more flexibly model the posterior than existing approaches (Chapter 4). Finally, we tackle (3) with a variant of random forests called *DaRE RF* which maintains a minimal set of statistics that

can be used to retrain parts of the model in need of retraining when responding to changes in the training data; we find DaRE RF is orders of magnitude more efficient than the naive approach of retraining from scratch while sacrificing little to no predictive power (Chapter 5).

6.2 Future Directions

In this section, we outline a number of potentially valuable research directions for tree ensembles that, if successful, would further improve their adaptability, interpretability, predictive capabilities, and make them more robust to adversarial attacks.

Adaptations from Deep Learning. We have shown in Chapter 3 that adapting recent influence estimation methods from deep learning leads to more efficient and effective influence estimation techniques in GBTs. Thus, it is reasonable to wonder what other concepts useful to deep learning models would also benefit tree-based ensembles if properly adapted. For example, the introduction of attention mechanisms in transformer-based architectures has led to a transformative success for deep learning models, especially for NLP tasks [214]; perhaps similar attention-based mechanisms adapted for tree-based models and applied to tabular data may lead to better predictive performance. Memorization is also a big concern in deep learning models and has recently generated much research interest [66]. GBTs have considerable representational power as is evident by their predictive prowess, thus similar analytical techniques used to identify memorized examples in deep learning may also be relevant in GBTs. Backdoor attacks [90, 219], data poisoning attacks [196], and adversarial attacks in general [193] are also potentially viable concepts to adapt from deep learning

research which, if successful in tree ensembles, could highlight new vulnerabilities and possibly lead to research that make tree ensembles even more robust.

Improving Instance-Based Uncertainty Estimation. The main limitation of IBUG is its relatively slow inference time, this is because IBUG is inherently an instance-based learner, whose inference time complexity scales with the size of the training set. We have already shown in Chapter 4 that sampling trees can signficantly reduce prediction time, often by 1-2 orders of magnitude. However, exploring additional approximations such as instance subsampling or reweighting may further significantly reduce the inference time of IBUG and lead to IBUG being more widely adopted. We have also shown that IBUG is a complementary approach to CBU (CatBoost with uncertainty), thus combining IBUG with other approaches to probabilistic regression such as conformal prediction [8] could be a valuable research direction and may lead to more accurate uncertainty estimators that ultimately provide more interpretable predictions to practitioners.

Certified Predictions. Dataset poisoning is the process of carefully adding or modifying a strategic set of training examples to produce a model with low predictive performance or a model that generates targeted predictions, and is quickly becoming an important security concern in the ML community [131]. One potential vulnerable target is regression models (including tree ensembles), in which arbitrary changes to the training data could induce arbitrary changes to predictions of the learned model. A promising model-agnostic certified approach was recently introduced by Hammoudeh and Lowd [95], however, developing methods specific to tree ensembels may result in tighter bounds, ultimately providing more significant

certifications to ML practitioners looking to protect their models in the face of potential dataset attacks.

DaRE RF Applications and Extensions. There are many exciting opportunities and applications of DaRE forests, from maintaining user privacy to building interpretable models to cleaning data, all without retraining from scratch. At its best, DaRE RF was more than four orders of magnitude faster than naive retraining, so it has the potential to enable new applications of model updating that were previously intractable. One main limitation of DaRE RF, however, is the significant increase in memory consumption over traditional RF models, thus exploring how to reduce the overall space complexity of DaRE models would be greatly beneficial and possibly further the adoption of DaRE models to smaller more constrained devices such as mobile phones. Additionally, we focused mainly on *removing* examples from random forests in Chapter 5, however, it is very likely that a similar approach can be used to *add* data as well, allowing practitioners to easily update their models by efficiently adding new data as they arrive and removing old outdated data as necessary. Finally, we chose random forests when researching the possibility of efficient data removal due to its simplicity compared to gradient-boosted trees, however, gradient-boosted trees tend to outperform random forests in terms of predictive performance. Thus, exploring the possibility of extending DaRE to GBTs would be valuable future work.

APPENDIX A

IDENTIFYING INFLUENTIAL TRAINING EXAMPLES

In this chapter, we provide implementation details, experiment details, and additional analyses regarding the influence estimation methods from Chapter 3.

A.1 Implementation and Dataset 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 [15], 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[1] [154], XGBoost [39], LightGBM [118], and CatBoost [159].

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 C.1 summarizes the data sets after preprocessing.

- **Adult** [59] 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).

[1] For our experiments, we use `HistGradientBoostingRegressor` and `HistGradientBoostingClassifier`.

Dataset	**Task**	**Metric**	**#instances**	**% Pos.**	**#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 A.8. Dataset 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).

– **Bank** [59, 147] 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** [59, 127] 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** [130, 152] 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** [59, 222] consists of 1,030 instances of concrete characterized by 8 attributes. The aim is to predict the compressive strength of the concrete (regression).

- **Credit** [59, 223] 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** [59, 198] consists of 101,766 instances of patient and hospital readmission outcomes (46,902 readmitted) characterized by 55 attributes (binary classification).

- **Energy** [59, 209] 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** [168] 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** [59] 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** [59, 140] 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** [163] 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** [49, 59] 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** [105] 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** [201] 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).

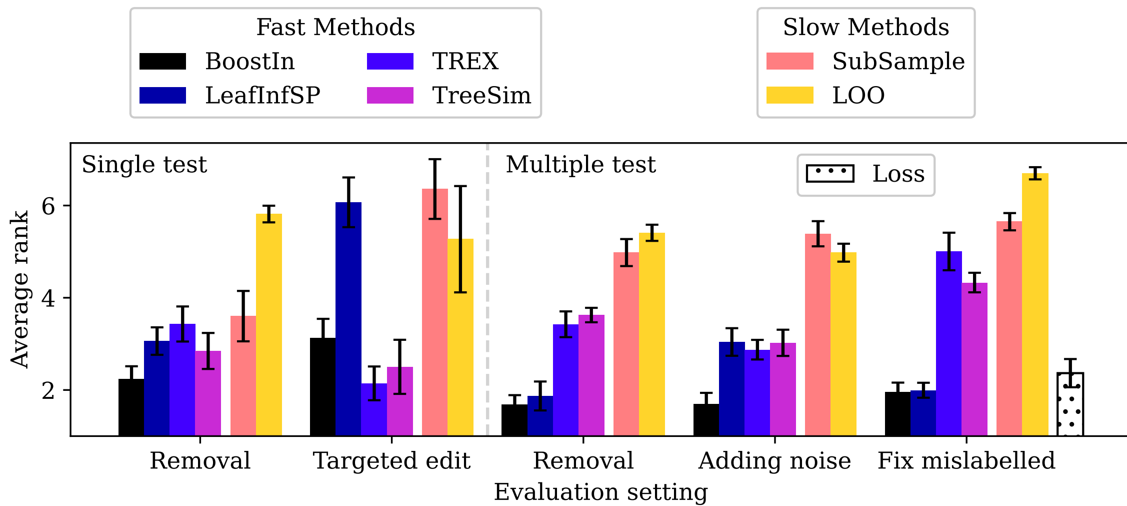
- **Power** [59, 117, 210] contains 9,568 readings of a Combined Cycle Power Plant (CCPP) at full work load. Each reading is characterized by 4 features. The aim is to predict the net hourly electrical energy output (regression).
- **Protein** [59] contains 45,730 tertiary-protein-structure instances characterized by 9 attributes. The aim is to predict the armstrong coefficient of the protein structure (regression).
- **Spambase** [59] 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** [115] 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 [178]. 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** [30, 56] 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** [50, 59] 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).

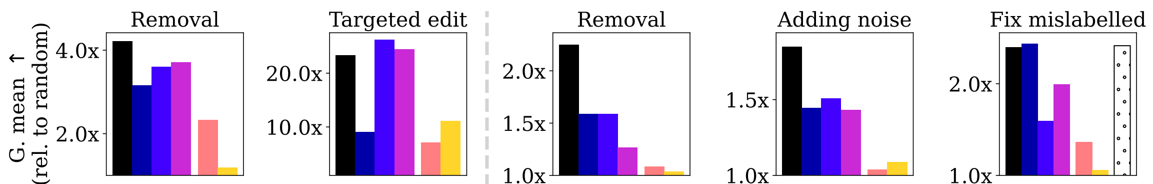
A.2 Experiment Details

Here we provide detailed analyses of the different influence estimation methods, including results using all datasets.

A.2.1 Summary of Results: All Datasets. Figure A.14 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.

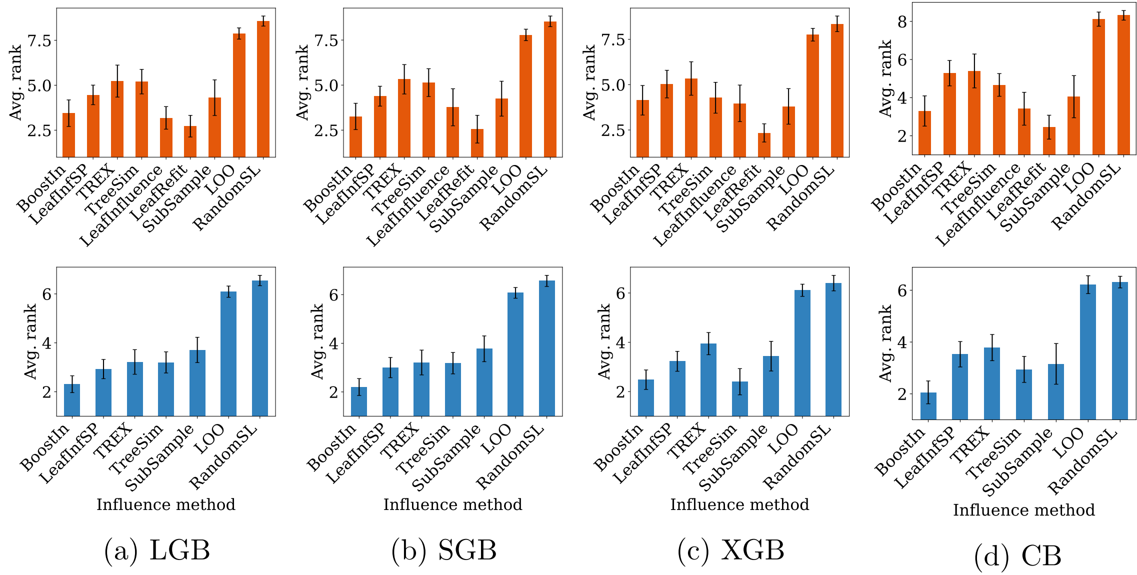
Figure A.14. High-level overview of results including *all* data sets; thus, LeafRefit and LeafInfluence are not included in this analysis.

A.2.2 Removing Examples (Single Test).

Figure A.15 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:[2]

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

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.

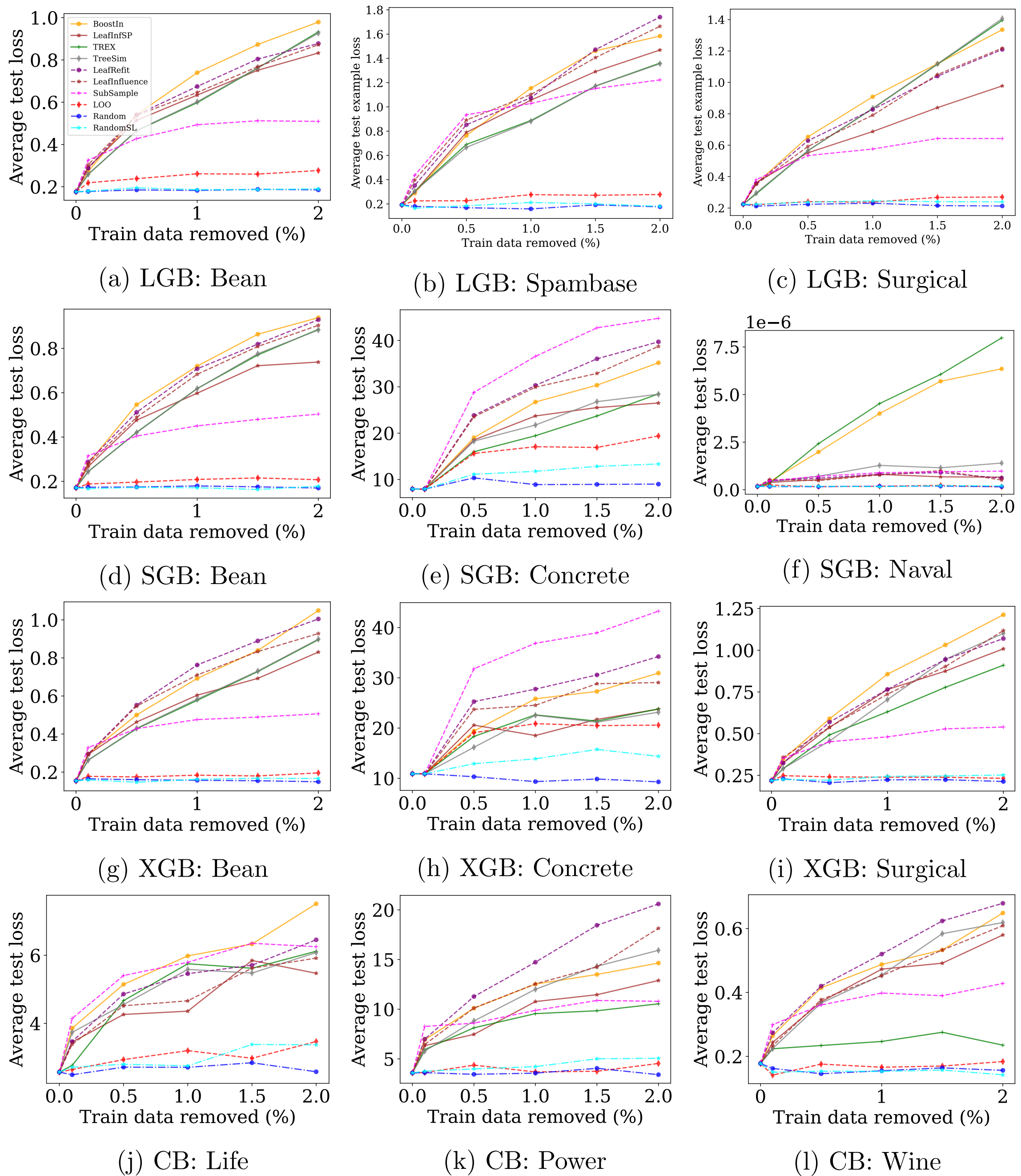


(a) LGB (b) SGB (c) XGB (d) CB

Figure A.15. 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 A.16 shows the change in loss for different GBDT types and data sets; methods using the fixed-structure assumption tend to perform best.

[2]For regression, $\mathcal{I}_{RandomSL}(z_i, z_e) = \mathcal{N}(\mu_i, \sigma_i)$ in which $\mu_i = 1/|y_i - y_e|$ and $\sigma_i = s.d.(|y_i - y_e|)$.



(a) LGB: Bean (b) LGB: Spambase (c) LGB: Surgical

(d) SGB: Bean (e) SGB: Concrete (f) SGB: Naval

(g) XGB: Bean (h) XGB: Concrete (i) XGB: Surgical

(j) CB: Life (k) CB: Power (l) CB: Wine

Figure A.16. Change in loss for a random test example (averaged over 100 runs) as training examples are removed. Higher is better.

A.2.3 Targeted Label Edits (Single Test). Figure A.17 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.

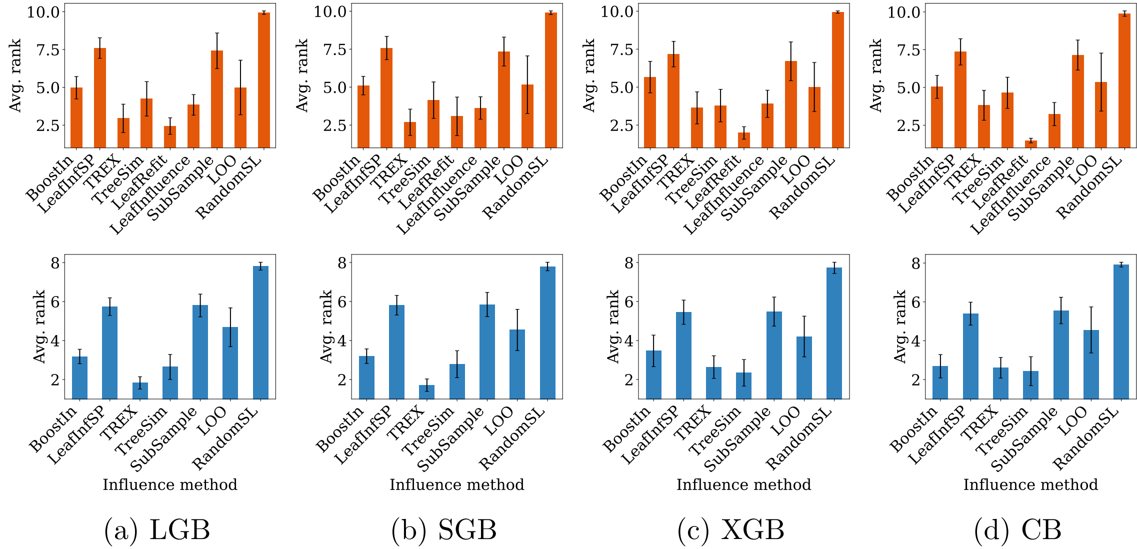


Figure A.17. Average ranks when editing training labels to a target label for a single test instance, shown for each GBDT type. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

A.2.4 Removing Examples (Multiple Test). Figure A.18 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.

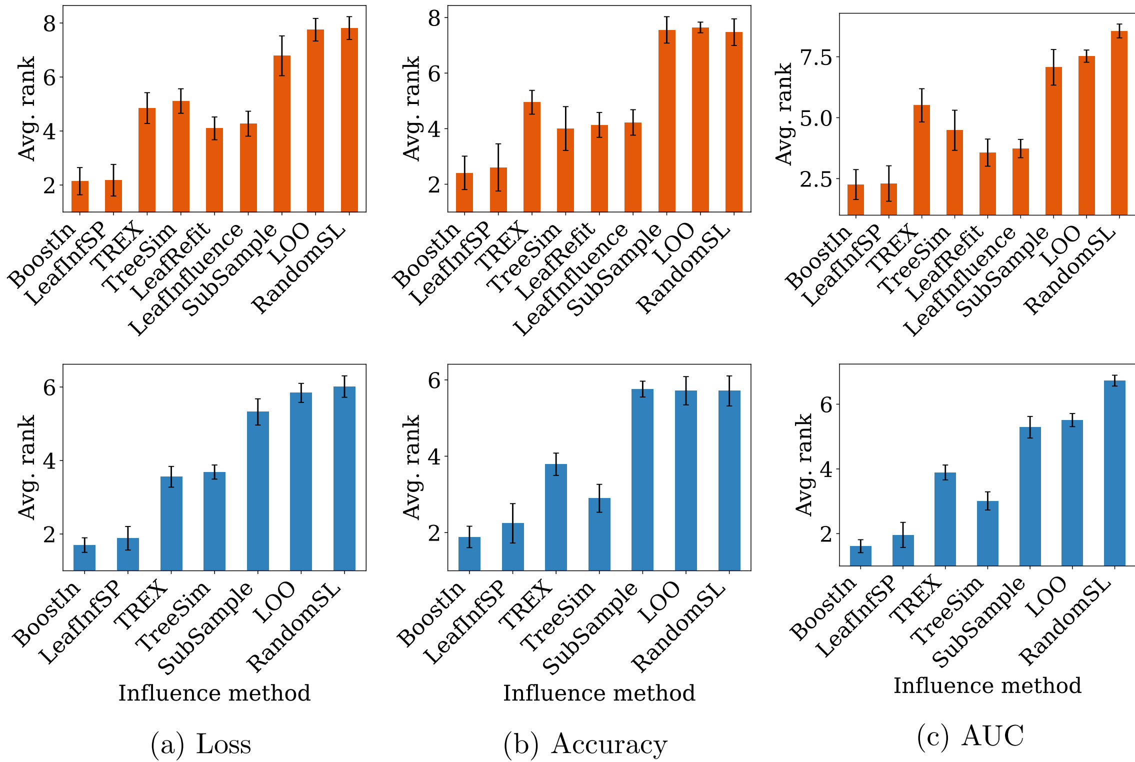


Figure A.18. Average ranks after removing training examples for multiple test instances and evaluating on a held-out test set using different predictive performance metrics. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints, tree types, and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

A.2.5 Adding Noise (Multiple Test). Figure A.19 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.

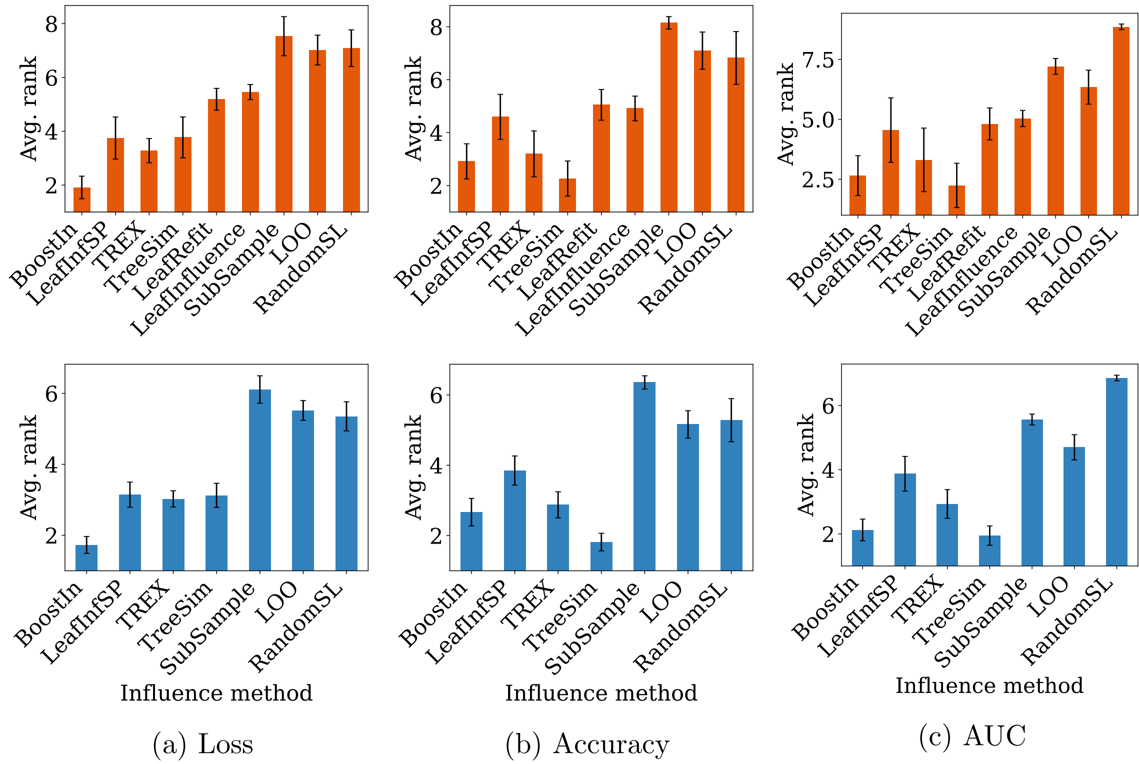
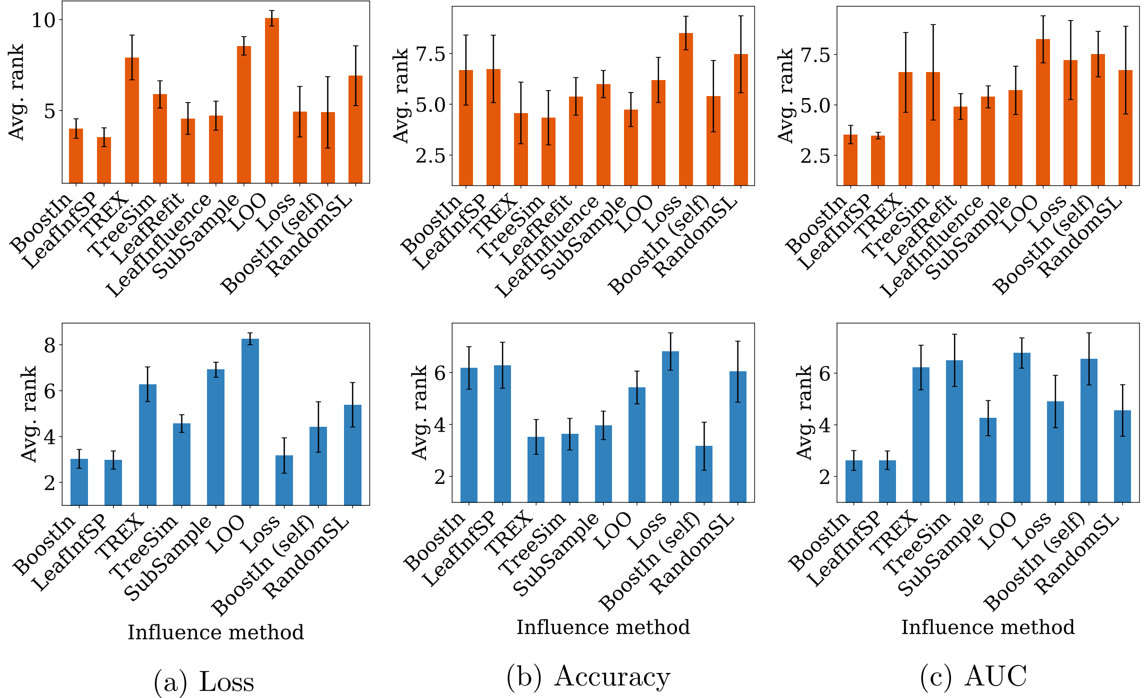


Figure A.19. Average ranks after add noise to training examples for multiple test instances and evaluating on a held-out test set using different predictive performance metrics. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints, tree types, and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

A.2.6 Fixing Mislabelled Examples (Multiple Test).

Figure A.20 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

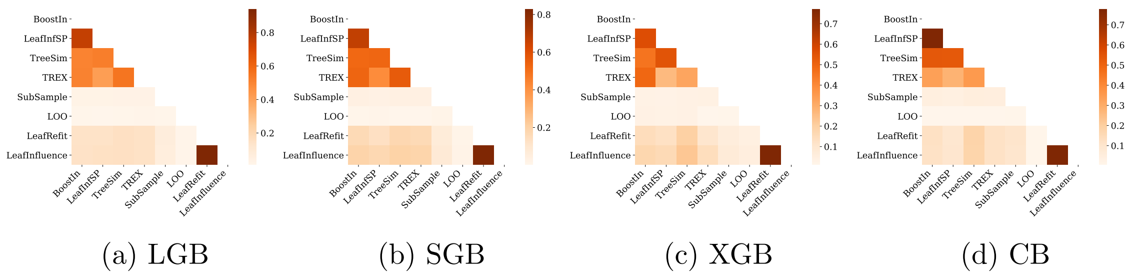
Figure A.20. Average ranks when measuring the predictive performance on the held-out test set after checking/fixing any noisy/mislabelled training examples. *Top row*: SDS data sets; *Bottom row*: all data sets. Results are averaged over checkpoints, tree types, and data sets. Error bars represent 95% confidence intervals computed over data sets. Lower is better.

A.2.7 Runtime Comparison. Table A.9 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

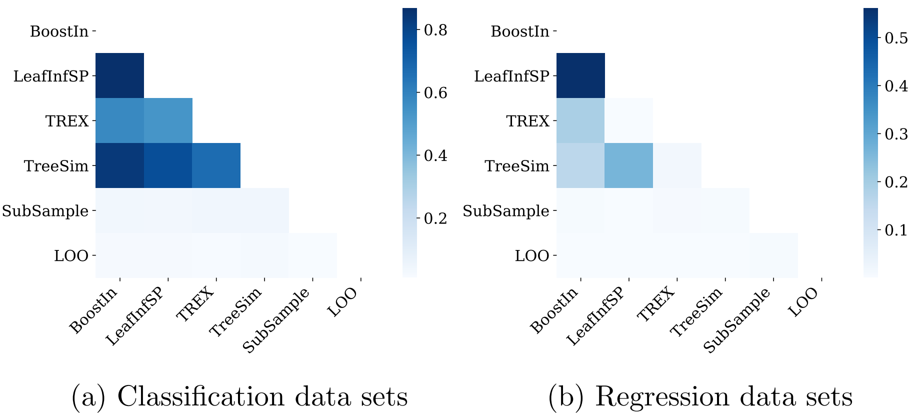
Table A.9. Time (in seconds) to compute all influences values for a single test instance for the SDS data sets. Each experiment is repeated 5 times, and results are averaged over GBDT types.

A.2.8 Correlation Between Influence Methods. Figure A.21 shows additional correlation heatmaps averaged over the SDS data sets for each GBDT type; overall, the trends remain the same across GBDT types. Figure A.22 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 A.21. Spearman correlation coefficient between influence methods for each GBDT type, averaged over 100 test examples and SDS data sets.

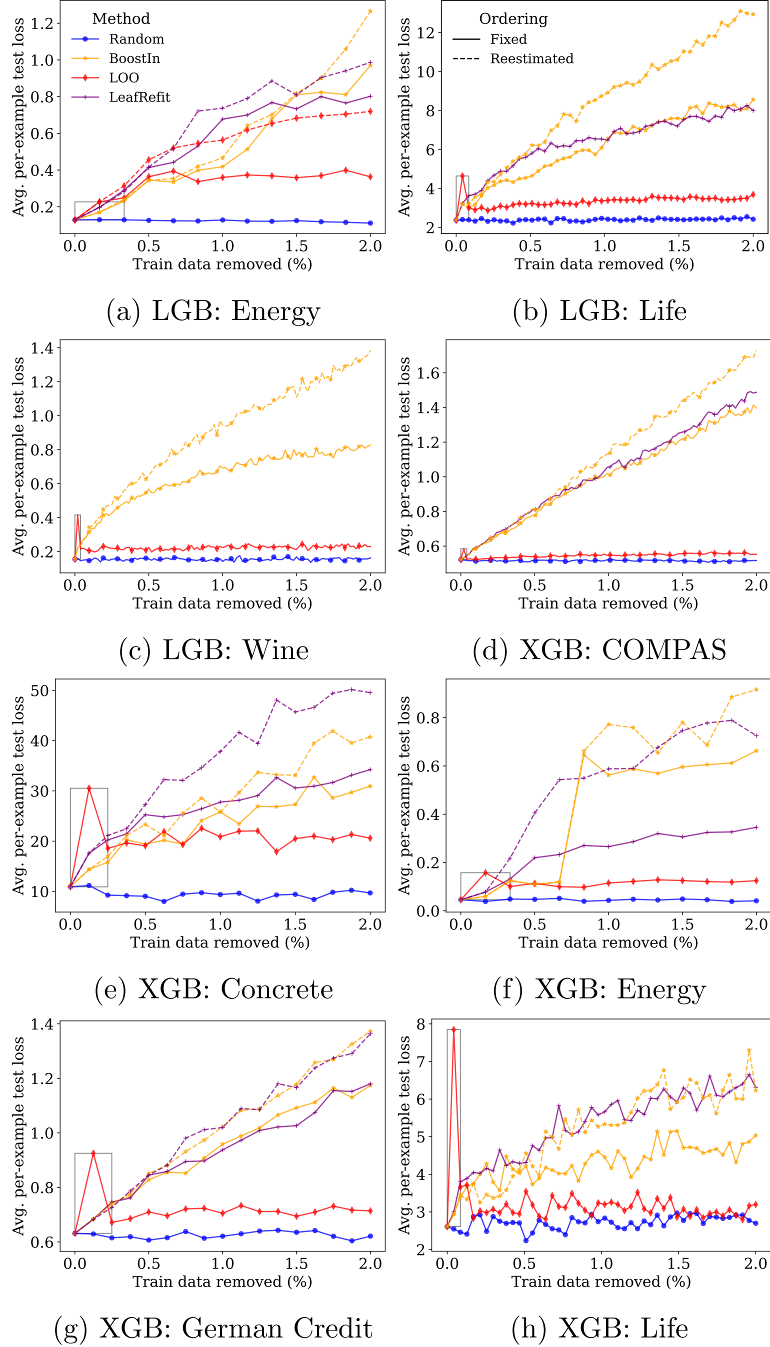


(a) Classification data sets (b) Regression data sets

Figure A.22. Spearman correlation coefficient between influence methods averaged over 100 test examples, all GBDT types, and either classification or regression data sets.

A.2.9 The Structural Fragility of LOO.

Figure A.23 shows 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 A.23. Change in loss (averaegd over 100 test instances) as training examples are removed *one at a time*. Higher is better. The gray box shows the spike in loss after removing a single training example identified by LOO.

A.3 Additional Experiments

Here we present additional experimental results, including predictive performance of GBDTs vs. non-GBDT models, and all values considered and selected during hyperparameter tuning.

A.3.1 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).

	GBDT				Non-GBDT					
Dataset	**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**	**0.909s**	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 A.10. 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 A.8 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 [154]; 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 [154]; 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 [154]; we tune k using values [3, 5, 7, 11, 15, 31, 61].
- Support vector machine (SVM): Support vector machine implementation from Scikit-Learn [154]; 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 [154]; 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 [154]; 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

	LGB		XGB		CB			SGB		
Dataset	T	$l_{\max}$	T	$d_{\max}$	T	$d_{\max}$	η	T	$l_{\max}$	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 A.11. 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 A.8 for reference.

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 A.8, and the selected hyperparameters for the LGB, XGB, CB, and SGB models are in Table A.11.

Table A.10 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 motivates the need for tailored influence-estimation methods for GBDT models to better understand their decision-making processes.

APPENDIX B
QUANTIFYING PREDICTION UNCERTAINTY

In this chapter, we provide implementation details, experiment details, and additional analyses for IBUG from Chapter 4.

B.1 Implementation and Experiment Details

We implement IBUG in Python, using Cython—a Python package allowing the development of C extensions—to store a unified representation of the model structure. IBUG currently supports all major modern gradient boosting frameworks including XGBoost [39], LightGBM [118], and CatBoost [159]. Experiments are run on an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.6GHz with 60GB of RAM @ 2.4GHz. We run our experiments on publicly available datasets. Links to all data sources as well as the code for IBUG and all experiments is currently available online.[1]

B.1.1 Metrics. We use the continuous ranked probability score (CRPS) and negative log likelihood (NLL) to measure probabilistic performance. CRPS is a quadratic measure of discrepancy between the cumulative distribution function (CDF) F of forecast $\hat{y}$ and the empirical CDF of the scalar observation y: $\int (F(\hat{y}) - \mathbb{1}[\hat{y} \geq y])^2 d\hat{y}$ in which $\mathbb{1}$ is the indicator function [84, 226]. To evaluate point performance, we use root mean squared error (RMSE): $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$.

B.1.2 Datasets. This section gives a detailed description for each dataset we use in our experiments.

- **Ames** [53] consists of 2,930 instances of housing prices in the Ames, Iowa area characterized by 80 attributes. The aim is to predict the sale price of a given house.

[1] `https://github.com/jjbrophy47/ibug`.

- **Bike** [59, 65] contains 17,379 measurements of the number of bikes rented per hour characterized by 16 attributes. The aim is to predict the number of bikes rented for a given hour.
- **California** [153] consists of 20,640 instances of median housing prices in various California districts characterized by 8 attributes. The aim is to predict the median housing price for the given district.
- **Communities** [59, 167] consists of 1,994 measurements of violent crime statistics based on crime, survey, and census data. The dataset is characterized by 100 attributes, and the aim is to predict the violent crime rate for a given population.
- **Concrete** [59, 222] consists of 1,030 instances of concrete characterized by 8 attributes. The aim is to predict the compressive strength of the concrete.
- **Energy** [59, 209] 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.
- **Facebook** [59, 192] consists of 40,949 Facebook posts characterized by 53 attributes. The aim is to predict the number of comments for a given post.
- **Kin8nm** [213] consists of 8,192 instances of the forward kinematics of an 8 link robotic arm. The aim is to predict the forward kinematics of the robotic arm.
- **Life** [163] consists of 2,928 instances of life expectancy estimates for various countries during one year. Each instance is characterized by 20 attributes,

and the aim is to predict the life expectancy of the country during a specific year.

- **MEPS** [46] consists of 16,656 instances of medical expenditure survey data. Each instance is characterized by 139 attributes, and the aim is to predict the insurance utilization for the given medical expenditure.

- **MSD** [17] consists of 515,345 songs characterized by 90 audio features constructed from each song. The aim is to predict what year the song was released based on the audio features.

- **Naval** [49, 59] 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.

- **News** [59, 67] consists of 39,644 Mashable articles characterized by 60 features. The aim is to predict the number of shares for a given article.

- **Obesity** [201] 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.

- **Power** [59, 117, 210] 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.

- **Protein** [59] contains 45,730 tertiary-protein-structure instances characterized by 9 attributes. The aim is to predict the armstrong coefficient of the protein structure.

- **STAR** [59, 197] contains 2,161 student-teacher achievement scores characterized by 39 attributes. The aim is to predict the student-teacher achievement based on the given intervention.

- **Superconductor** [59, 94] contains 21,263 potential superconductors characterized by 81 attributes. The aim is to predict the critical temperature of the given superconductor.

- **Synthetic** [25, 74] is a non-linear synthetic regression dataset in which the inputs are independent and uniformly distributed on the interval $[0, 1]$; the dataset contains 10,000 instances characterized by 100 attributes.

- **Wave** [59] consists of 287,999 positions and absorbed power outputs of wave energy converters (WECs) in four real wave scenarios off the southern coast of Australia (Sydney, Adelaide, Perth and Tasmania). The aim is to predict the total power output of a given WEC.

- **Wine** [50, 59] consists of 6,497 instances of Portuguese "Vinho Verde" red and white wine characterized by 11 features. The aim is to predict the quality of the wine from 0-10.

- **Yacht** [59] consists of 308 instances of yacht-sailing performance characterized by 6 attributes. The aim is to predict the residual resistance per unit weight of displacement.

For each dataset, we generate one-hot encodings for any categorical variable and leave all numeric and binary variables as is. Table B.1 shows a summary of the datasets after preprocessing.

Table B.1. Dataset summary after preprocessing.

Dataset	Source	n	p
Ames	[53]	2,930	358
Bike	[59, 65]	17,379	37
California	[153]	20,640	100
Communities	[59, 167]	1,994	100
Concrete	[59, 222]	1,030	8
Energy	[59, 209]	768	16
Facebook	[59, 192]	40,949	133
Kin8nm	[213]	8,192	8
Life	[163]	2,928	204
MEPS	[46]	15,656	139
MSD	[17]	515,345	90
Naval	[49, 59]	11,934	17
News	[59, 67]	39,644	58
Obesity	[201]	48,346	100
Power	[59, 117, 210]	9,568	4
Protein	[59]	45,730	9
STAR	[59, 197]	2,161	95
Superconductor	[59, 94]	21,263	82
Synthetic	[25, 74]	10,000	100
Wave	[59]	287,999	48
Wine	[50, 59]	6,497	11
Yacht	[59]	308	6

B.1.3 Hyperparameters. Tables B.2 and B.3 show hyperparameter values selected most often for each dataset when optimizing CRPS and NLL, respectively. We tune nearest-neighbor hyperparameter k using values [3, 5, 7, 9, 11, 15, 31, 61, 91, 121, 151, 201, 301, 401, 501, 601, 701], γ and δ using values [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0, 1e0, 1e1, 1e2, 1e3] with multipliers [1.0, 2.5, 5.0], number of trees T using values [10, 25, 50, 100, 250, 500, 1000, 2000] (since NGBoost has no hyperparameters to tune besides T, we tune T on the validation set using early stopping [61]), learning rate η using values [0.01, 0.1], maximum number of leaves h using values [15, 31, 61, 91], minimum number of leaves n_{ℓ_0} using values [1, 20], maximum depth d using values [2, 3, 5, 7, -1

(unlimited)], and ρ which selects the minimum variance computed from the validation set predictions. For the MSD and Wave datasets, we use a bagging fraction of 0.1 [61, 195].

Table B.2. Hyperparameters selected most often over 10 folds for each dataset when optimizing CRPS.

	NGBoost		PGBM					CBU				
Dataset	T	γ/δ	T	η	h	n_{ℓ_0}	γ/δ	T	η	d	n_{ℓ_0}	γ/δ
Ames	2000	γ:1e+00	2000	0.1	15	1	γ:2e+00	2000	0.1	7	1	δ:5e+03
Bike	2000	γ:5e-01	2000	0.01	61	1	δ:1e-08	2000	0.1	2	1	δ:5e-02
California	2000	δ:3e-02	1000	0.1	31	20	δ:3e-02	2000	0.1	-1	1	δ:1e-01
Communities	223	δ:1e-02	500	0.01	15	20	γ:1e+01	2000	0.01	7	1	δ:5e-02
Concrete	2000	δ:1e+00	2000	0.1	15	20	δ:1e-08	2000	0.1	5	1	δ:2e+00
Energy	2000	γ:5e-01	2000	0.1	15	1	γ:5e-01	2000	0.1	3	1	δ:1e-01
Facebook	2000	γ:1e+00	2000	0.01	15	1	γ:2e+00	2000	0.1	5	1	γ:1e+00
Kin8nm	581	δ:3e-02	2000	0.1	61	20	δ:5e-02	2000	0.1	7	1	δ:5e-02
Life	2000	γ:1e+00	2000	0.1	15	1	δ:2e-01	2000	0.1	5	1	δ:1e+00
MEPS	583	δ:1e-01	50	0.1	15	1	δ:1e-08	100	0.01	-1	1	γ:1e+00
MSD	2000	δ:1e-01	2000	0.01	91	20	γ:1e+01	2000	0.1	7	1	δ:5e-01
Naval	2000	δ:0e+00	2000	0.1	61	20	γ:3e-02	2000	0.1	7	1	δ:3e-04
News	2000	γ:5e-01	100	0.01	15	20	δ:2e+03	100	0.01	2	1	γ:5e-01
Obesity	2000	δ:1e-01	500	0.1	91	20	δ:1e-08	2000	0.1	7	1	δ:5e-01
Power	2000	δ:2e-01	500	0.1	91	1	δ:5e-01	2000	0.1	7	1	δ:1e+00
Protein	2000	δ:1e-01	2000	0.1	91	20	γ:2e+00	2000	0.1	7	1	δ:1e+00
STAR	187	δ:2e+01	1000	0.01	15	1	γ:1e+01	2000	0.01	-1	1	δ:5e+01
Superconductor	162	γ:5e-01	1000	0.01	15	20	δ:5e-02	2000	0.1	-1	1	δ:3e-02
Synthetic	208	δ:5e-01	500	0.01	15	20	δ:1e+01	2000	0.01	3	1	δ:1e+00
Wave	2000	δ:5e+03	2000	0.1	15	1	δ:1e-08	2000	0.1	-1	1	δ:2e+02
Wine	309	δ:5e-02	2000	0.01	91	20	δ:5e-01	2000	0.1	7	1	δ:2e-01
Yacht	2000	γ:5e-01	2000	0.1	15	1	δ:0e+00	2000	0.1	3	1	γ:2e+00

	CatBoost				IBUG		
Dataset	T	η	d	n_{ℓ_0}	k	ρ	γ/δ
Ames	2000	0.1	-1	1	5	2206	δ:3e-04
Bike	2000	0.1	2	1	3	0.471	γ:2e-01
California	2000	0.1	-1	1	7	2e-15	δ:1e-08
Communities	2000	0.01	-1	1	15	0.017	δ:0e+00
Concrete	2000	0.1	5	1	3	0.049	γ:5e-01
Energy	2000	0.1	5	1	3	0.035	γ:1e-01
Facebook	2000	0.1	-1	1	15	0.213	δ:1e-01
Kin8nm	2000	0.1	7	1	3	0.003	γ:5e-01
Life	2000	0.1	5	1	3	0.047	γ:5e-01
MEPS	250	0.01	5	1	201	1.08	δ:1e-07
MSD	2000	0.1	7	1	31	1.25	δ:1e-07
Naval	2000	0.1	7	1	3	1e-15	γ:5e-01
News	1000	0.01	2	1	15	163	γ:5e-01
Obesity	2000	0.1	7	1	5	0.306	γ:5e-01
Power	2000	0.1	7	1	5	0.220	δ:1e-01
Protein	2000	0.1	7	1	31	0.028	δ:1e-01
STAR	250	0.01	5	1	121	192	δ:1e-08
Superconductor	2000	0.1	5	1	3	5e-15	γ:1e-01
Synthetic	1000	0.01	7	1	401	9.34	δ:1e-08
Wave	2000	0.1	-1	1	3	2e-10	γ:2e-01
Wine	2000	0.1	7	1	15	0.268	δ:2e-08
Yacht	2000	0.1	2	1	3	0.196	γ:1e-01

Table B.3. Hyperparameters selected most often over 10 folds for each dataset when optimizing NLL.

	NGBoost		PGBM					CBU				
Dataset	T	γ/δ	T	η	h	n_{ℓ_0}	γ/δ	T	η	d	n_{ℓ_0}	γ/δ
Ames	373	δ:2e+03	2000	0.1	15	1	γ:2e+00	2000	0.1	7	1	γ:1e+01
Bike	926	δ:0e+00	2000	0.01	61	1	δ:1e-08	2000	0.1	2	1	δ:0e+00
California	2000	δ:5e-02	1000	0.1	31	20	δ:1e-01	2000	0.1	-1	1	δ:2e-01
Communities	156	δ:1e-02	500	0.01	15	20	γ:1e+01	2000	0.01	7	1	δ:1e-01
Concrete	383	δ:1e+00	2000	0.1	15	20	δ:1e+00	2000	0.1	5	1	δ:2e+00
Energy	422	δ:1e-02	2000	0.1	15	1	δ:1e-08	2000	0.1	3	1	δ:1e-01
Facebook	549	δ:0e+00	2000	0.01	15	1	γ:5e+00	2000	0.1	5	1	γ:2e+00
Kin8nm	975	δ:1e-02	2000	0.1	61	20	δ:5e-02	2000	0.1	7	1	δ:1e-01
Life	366	δ:2e-01	2000	0.1	15	1	δ:1e+00	2000	0.1	5	1	δ:1e+00
MEPS	188	δ:1e+00	50	0.1	15	1	δ:1e-08	100	0.01	-1	1	δ:1e+00
MSD	2000	δ:3e-02	2000	0.01	91	20	γ:1e+01	2000	0.1	7	1	δ:1e+00
Naval	2000	δ:5e-05	2000	0.1	61	20	γ:5e-02	2000	0.1	7	1	δ:3e-04
News	38	δ:1e+03	100	0.01	15	20	γ:1e+01	100	0.01	2	1	δ:2e+03
Obesity	2000	δ:0e+00	500	0.1	91	20	δ:1e-08	2000	0.1	7	1	δ:5e-01
Power	275	δ:2e-01	500	0.1	91	1	δ:1e+00	2000	0.1	7	1	δ:2e+00
Protein	2000	δ:2e-01	2000	0.1	91	20	δ:2e+00	2000	0.1	7	1	δ:1e+00
STAR	176	δ:1e+01	1000	0.01	15	1	δ:2e+02	2000	0.01	-1	1	δ:5e+01
Superconductor	378	γ:1e+00	1000	0.01	15	20	γ:2e+00	2000	0.1	-1	1	δ:1e-01
Synthetic	284	δ:5e-01	500	0.01	15	20	δ:1e+01	2000	0.01	3	1	δ:1e+00
Wave	2000	γ:1e+00	2000	0.1	15	1	δ:5e+02	2000	0.1	-1	1	δ:2e+02
Wine	390	δ:5e-02	2000	0.01	91	20	γ:2e+01	2000	0.1	7	1	δ:5e-01
Yacht	356	δ:0e+00	2000	0.1	15	1	δ:5e-02	2000	0.1	3	1	δ:5e-01

	CatBoost				IBUG		
Dataset	T	η	d	n_{ℓ_0}	k	ρ	γ/δ
Ames	2000	0.1	-1	1	11	4673	δ:1e-08
Bike	2000	0.1	2	1	5	0.4	γ:2e-01
California	2000	0.1	-1	1	31	0.063	δ:0e+00
Communities	2000	0.01	-1	1	61	0.026	δ:0e+00
Concrete	2000	0.1	5	1	5	0.56	δ:1e-08
Energy	2000	0.1	5	1	3	0.087	γ:2e-01
Facebook	2000	0.1	-1	1	301	0.175	δ:1e-01
Kin8nm	2000	0.1	7	1	7	0.031	δ:0e+00
Life	2000	0.1	5	1	7	0.22	δ:2e-08
MEPS	250	0.01	5	1	301	1.76	δ:1e+00
MSD	2000	0.1	7	1	61	1.75	δ:1e-07
Naval	2000	0.1	7	1	5	4e-04	γ:5e-01
News	1000	0.01	2	1	301	994	δ:2e+03
Obesity	2000	0.1	7	1	9	0.529	δ:1e-07
Power	2000	0.1	7	1	15	0.861	δ:1e-07
Protein	2000	0.1	7	1	121	0.218	δ:5e-08
STAR	250	0.01	5	1	121	189	δ:1e-05
Superconductor	2000	0.1	5	1	7	0.019	γ:2e-01
Synthetic	1000	0.01	7	1	401	9.39	δ:1e-08
Wave	2000	0.1	-1	1	31	349	γ:2e-01
Wine	2000	0.1	7	1	61	0.297	δ:2e-08
Yacht	2000	0.1	2	1	3	0.196	γ:2e-01

B.1.4 Additional Metrics. In this section, we show results for point performance and probabilistic performance with additional metrics. Each table shows average results over the 10 random folds for each dataset, with standard errors in subscripted parentheses. We use the *Uncertainty Toolbox*[2] [45] to compute each metric. Lower is better for all metrics.

Point Performance and Negative-Log Likelihood. Tables B.4 and B.5 show point (RMSE) and probabilistic (NLL) performance of each method.

Table B.4. Point (RMSE ↓) performance for each method on each dataset. *Bottom row*: Head-to-head wins-ties-losses.

Dataset	NGBoost	PGBM	CBU	IBUG	IBUG+CBU
Ames	24580$_{(804)}$	**23541**$_{(1225)}$	**22576**$_{(924)}$	**22942**$_{(1388)}$	**22391**$_{(1119)}$
Bike	4.173$_{(0.076)}$	3.812$_{(0.225)}$	**2.850**$_{(0.192)}$	**2.826**$_{(0.200)}$	**2.708**$_{(0.202)}$
California	0.503$_{(0.003)}$	0.445$_{(0.001)}$	0.449$_{(0.002)}$	**0.432**$_{(0.001)}$	0.434$_{(0.002)}$
Communities	0.137$_{(0.004)}$	**0.135**$_{(0.004)}$	**0.133**$_{(0.004)}$	**0.133**$_{(0.004)}$	**0.132**$_{(0.004)}$
Concrete	5.485$_{(0.182)}$	3.840$_{(0.209)}$	**3.682**$_{(0.202)}$	**3.629**$_{(0.183)}$	**3.617**$_{(0.188)}$
Energy	0.461$_{(0.030)}$	0.291$_{(0.022)}$	0.381$_{(0.023)}$	**0.264**$_{(0.023)}$	0.303$_{(0.023)}$
Facebook	**20.8**$_{(1.102)}$	**20.5**$_{(0.867)}$	**20.1**$_{(0.913)}$	**20.0**$_{(0.903)}$	**19.9**$_{(0.929)}$
Kin8nm	0.176$_{(0.001)}$	0.108$_{(0.001)}$	0.103$_{(0.001)}$	**0.086**$_{(0.001)}$	0.091$_{(0.001)}$
Life	2.280$_{(0.032)}$	1.678$_{(0.059)}$	**1.637**$_{(0.058)}$	**1.652**$_{(0.055)}$	**1.610**$_{(0.056)}$
MEPS	**23.7**$_{(0.955)}$	**24.1**$_{(0.760)}$	**23.5**$_{(0.950)}$	**23.7**$_{(0.932)}$	**23.6**$_{(0.945)}$
MSD	9.121$_{(0.010)}$	8.804$_{(0.008)}$	8.743$_{(0.008)}$	8.747$_{(0.008)}$	**8.722**$_{(0.008)}$
Naval	0.002$_{(0.000)}$	0.001$_{(0.000)}$	0.001$_{(0.000)}$	**0.000**$_{(0.000)}$	**0.000**$_{(0.000)}$
News	**11162**$_{(1153)}$	**11047**$_{(1106)}$	**11036**$_{(1118)}$	**11036**$_{(1116)}$	**11032**$_{(1118)}$
Obesity	5.315$_{(0.022)}$	3.658$_{(0.033)}$	**3.572**$_{(0.038)}$	**3.576**$_{(0.037)}$	**3.567**$_{(0.037)}$
Power	3.836$_{(0.045)}$	3.017$_{(0.056)}$	**2.924**$_{(0.065)}$	**2.941**$_{(0.059)}$	**2.912**$_{(0.063)}$
Protein	4.525$_{(0.040)}$	**3.455**$_{(0.021)}$	3.520$_{(0.019)}$	3.512$_{(0.017)}$	3.493$_{(0.018)}$
STAR	233$_{(2.388)}$	**229**$_{(2.076)}$	**229**$_{(1.850)}$	**228**$_{(1.985)}$	**228**$_{(1.857)}$
Superconductor	**0.170**$_{(0.101)}$	0.425$_{(0.091)}$	0.463$_{(0.087)}$	0.427$_{(0.088)}$	0.419$_{(0.089)}$
Synthetic	**10.2**$_{(0.068)}$	**10.1**$_{(0.072)}$	**10.2**$_{(0.072)}$	**10.1**$_{(0.073)}$	**10.1**$_{(0.073)}$
Wave	13537$_{(32.7)}$	7895$_{(86.0)}$	4803$_{(37.5)}$	4899$_{(55.0)}$	**4020**$_{(33.5)}$
Wine	0.693$_{(0.010)}$	**0.603**$_{(0.010)}$	0.626$_{(0.010)}$	**0.596**$_{(0.012)}$	**0.598**$_{(0.011)}$
Yacht	0.761$_{(0.106)}$	0.809$_{(0.103)}$	**0.677**$_{(0.124)}$	**0.668**$_{(0.125)}$	**0.645**$_{(0.124)}$
IBUG	16-5-1	13-8-1	6-16-0	-	2-13-7
IBUG+CBU	18-3-1	12-9-1	16-6-0	7-13-2	-

[2] `https://uncertainty-toolbox.github.io/`

Table B.5. Probabilistic (NLL ↓) performance for each method on each dataset. *Bottom row*: Head-to-head wins-ties-losses.

Dataset	NGBoost	PGBM	CBU	IBUG	IBUG+CBU
Ames	$11.3_{(0.018)}$	$11.3_{(0.029)}$	$11.9_{(0.140)}$	$\mathbf{11.2}_{(0.030)}$	$11.5_{(0.092)}$
Bike	$1.942_{(0.024)}$	$1.929_{(0.078)}$	$\mathbf{1.184}_{(0.034)}$	$1.886_{(0.056)}$	$1.382_{(0.042)}$
California	$0.545_{(0.007)}$	$0.580_{(0.005)}$	$0.524_{(0.004)}$	$0.477_{(0.010)}$	$\mathbf{0.437}_{(0.016)}$
Communities	$\mathbf{-0.697}_{(0.045)}$	$\mathbf{-0.666}_{(0.034)}$	$\mathbf{-0.614}_{(0.109)}$	$\mathbf{-0.639}_{(0.135)}$	$\mathbf{-0.665}_{(0.116)}$
Concrete	$3.043_{(0.030)}$	$2.802_{(0.083)}$	$\mathbf{2.766}_{(0.086)}$	$2.980_{(0.146)}$	$\mathbf{2.695}_{(0.060)}$
Energy	$0.604_{(0.192)}$	$\mathbf{0.322}_{(0.182)}$	$\mathbf{0.406}_{(0.116)}$	$1.644_{(0.514)}$	$0.658_{(0.165)}$
Facebook	$\mathbf{2.102}_{(0.026)}$	$3.116_{(0.077)}$	$2.574_{(0.191)}$	$2.175_{(0.067)}$	$2.276_{(0.140)}$
Kin8nm	$-0.414_{(0.007)}$	$-0.774_{(0.034)}$	$-0.772_{(0.008)}$	$\mathbf{-0.841}_{(0.008)}$	$\mathbf{-0.847}_{(0.010)}$
Life	$2.163_{(0.029)}$	$1.943_{(0.033)}$	$1.932_{(0.079)}$	$1.858_{(0.033)}$	$\mathbf{1.783}_{(0.041)}$
MEPS	$\mathbf{3.722}_{(0.050)}$	$3.902_{(0.049)}$	$\mathbf{3.699}_{(0.038)}$	$3.793_{(0.052)}$	$\mathbf{3.675}_{(0.041)}$
MSD	$3.454_{(0.002)}$	$3.571_{(0.002)}$	$3.415_{(0.001)}$	$3.415_{(0.002)}$	$\mathbf{3.393}_{(0.001)}$
Naval	$-5.408_{(0.007)}$	$-5.064_{(0.338)}$	$-6.141_{(0.013)}$	$-6.208_{(0.010)}$	$\mathbf{-6.284}_{(0.007)}$
News	$10.9_{(0.268)}$	$\mathbf{10.7}_{(0.339)}$	$\mathbf{10.6}_{(0.205)}$	$\mathbf{10.6}_{(0.208)}$	$\mathbf{10.6}_{(0.192)}$
Obesity	$2.940_{(0.003)}$	$2.604_{(0.015)}$	$\mathbf{2.439}_{(0.009)}$	$2.646_{(0.009)}$	$2.515_{(0.010)}$
Power	$2.752_{(0.032)}$	$\mathbf{2.518}_{(0.021)}$	$2.538_{(0.019)}$	$2.575_{(0.036)}$	$\mathbf{2.514}_{(0.017)}$
Protein	$2.840_{(0.014)}$	$2.661_{(0.005)}$	$2.553_{(0.009)}$	$2.653_{(0.054)}$	$\mathbf{2.516}_{(0.010)}$
STAR	$6.869_{(0.013)}$	$6.866_{(0.012)}$	$\mathbf{6.866}_{(0.014)}$	$\mathbf{6.853}_{(0.008)}$	$\mathbf{6.852}_{(0.009)}$
Superconductor	$\mathbf{12.2}_{(13.1)}$	$\mathbf{0.035}_{(0.095)}$	$\mathbf{-0.014}_{(0.078)}$	$0.783_{(0.181)}$	$0.108_{(0.036)}$
Synthetic	$3.745_{(0.007)}$	$\mathbf{3.742}_{(0.006)}$	$\mathbf{3.741}_{(0.008)}$	$\mathbf{3.738}_{(0.007)}$	$\mathbf{3.738}_{(0.007)}$
Wave	$10.7_{(0.002)}$	$10.3_{(0.021)}$	$\mathbf{9.675}_{(0.003)}$	$10.5_{(0.030)}$	$9.760_{(0.046)}$
Wine	$1.025_{(0.013)}$	$0.952_{(0.020)}$	$1.025_{(0.028)}$	$\mathbf{0.910}_{(0.016)}$	$0.933_{(0.012)}$
Yacht	$0.905_{(0.232)}$	$\mathbf{0.357}_{(0.162)}$	$0.951_{(0.252)}$	$1.799_{(1.307)}$	$0.840_{(0.310)}$
IBUG	12-10-0	7-11-4	5-11-6	-	2-8-12
IBUG+CBU	15-6-1	10-10-2	13-6-3	12-8-2	-

Check and Interval scores. Tables B.6 and B.7 show results when measuring performance with two additional proper scoring rules [84], *check score* (a.k.a. "pinball loss") and *interval score* (evaluation using a pair of quantiles with expected coverage). Under these additional metrics, IBUG+CBU still outperform all other approaches.

Table B.6. Probabilistic (check score a.k.a. "pinball loss" ↓) performance. *Bottom row*: Head-to-head wins-ties-losses.

Dataset	NGBoost	PGBM	CBU	IBUG	IBUG+CBU
Ames	$19358_{(276)}$	$5487_{(179)}$	$5551_{(167)}$	$\mathbf{5266}_{(185)}$	$\mathbf{5145}_{(186)}$
Bike	$6.264_{(0.482)}$	$0.597_{(0.020)}$	$0.420_{(0.018)}$	$0.490_{(0.024)}$	$\mathbf{0.386}_{(0.016)}$
California	$8e{+}10_{(8e+10)}$	$0.112_{(4e-04)}$	$0.110_{(4e-04)}$	$0.107_{(5e-04)}$	$\mathbf{0.104}_{(4e-04)}$
Communities	$0.034_{(0.001)}$	$0.034_{(1e-03)}$	$0.034_{(9e-04)}$	$\mathbf{0.033}_{(9e-04)}$	$\mathbf{0.033}_{(9e-04)}$
Concrete	$1.722_{(0.092)}$	$0.972_{(0.043)}$	$\mathbf{0.902}_{(0.039)}$	$0.932_{(0.049)}$	$\mathbf{0.878}_{(0.041)}$
Energy	$0.262_{(0.022)}$	$\mathbf{0.074}_{(0.003)}$	$0.099_{(0.005)}$	$\mathbf{0.072}_{(0.005)}$	$0.079_{(0.004)}$
Facebook	$2.024_{(0.049)}$	$1.788_{(0.047)}$	$1.617_{(0.030)}$	$1.551_{(0.033)}$	$\mathbf{1.502}_{(0.035)}$
Kin8nm	$0.048_{(3e-04)}$	$0.031_{(5e-04)}$	$0.029_{(3e-04)}$	$\mathbf{0.026}_{(3e-04)}$	$\mathbf{0.026}_{(3e-04)}$
Life	$1.462_{(0.739)}$	$0.411_{(0.014)}$	$0.389_{(0.012)}$	$0.400_{(0.011)}$	$\mathbf{0.368}_{(0.011)}$
MEPS	$\mathbf{2.779}_{(0.098)}$	$3.246_{(0.046)}$	$3.050_{(0.055)}$	$3.100_{(0.057)}$	$3.033_{(0.056)}$
MSD	$2.283_{(0.003)}$	$2.310_{(0.002)}$	$2.203_{(0.002)}$	$2.226_{(0.002)}$	$\mathbf{2.195}_{(0.002)}$
Naval	$0.002_{(3e-05)}$	$2e{-}04_{(2e-05)}$	$2e{-}04_{(2e-06)}$	$1e{-}04_{(1e-06)}$	$\mathbf{1e{-}04}_{(8e-07)}$
News	$\mathbf{1102}_{(23.7)}$	$1188_{(26.3)}$	$1181_{(26.2)}$	$1280_{(20.5)}$	$1198_{(26.0)}$
Obesity	$1.620_{(0.014)}$	$0.939_{(0.011)}$	$\mathbf{0.879}_{(0.009)}$	$0.941_{(0.010)}$	$0.894_{(0.009)}$
Power	$1.063_{(0.012)}$	$0.773_{(0.010)}$	$\mathbf{0.744}_{(0.011)}$	$0.778_{(0.010)}$	$\mathbf{0.743}_{(0.011)}$
Protein	$2739_{(2730)}$	$0.920_{(0.006)}$	$0.902_{(0.005)}$	$0.900_{(0.004)}$	$\mathbf{0.880}_{(0.004)}$
STAR	$66.6_{(0.803)}$	$\mathbf{65.9}_{(0.697)}$	$\mathbf{65.7}_{(0.647)}$	$\mathbf{65.4}_{(0.613)}$	$\mathbf{65.4}_{(0.605)}$
Superconductor	$1.215_{(0.014)}$	$\mathbf{0.064}_{(0.002)}$	$0.076_{(0.002)}$	$0.077_{(0.003)}$	$\mathbf{0.064}_{(0.002)}$
Synthetic	$2.918_{(0.021)}$	$\mathbf{2.897}_{(0.020)}$	$\mathbf{2.898}_{(0.020)}$	$\mathbf{2.894}_{(0.020)}$	$\mathbf{2.894}_{(0.020)}$
Wave	$2.9e{+}05_{(446)}$	$1964_{(37.3)}$	$1186_{(5.194)}$	$1350_{(8.028)}$	$\mathbf{1023}_{(4.813)}$
Wine	$0.194_{(0.002)}$	$\mathbf{0.163}_{(0.003)}$	$0.170_{(0.003)}$	$\mathbf{0.162}_{(0.003)}$	$\mathbf{0.162}_{(0.003)}$
Yacht	$0.594_{(0.080)}$	$\mathbf{0.147}_{(0.021)}$	$\mathbf{0.142}_{(0.024)}$	$\mathbf{0.139}_{(0.024)}$	$\mathbf{0.128}_{(0.023)}$
IBUG	17-3-2	11-9-2	9-5-8	-	1-6-15
IBUG+CBU	17-3-2	15-6-1	18-2-2	15-6-1	-

Table B.7. Probabilistic (interval score ↓) performance. *Bottom row*: Head-to-head wins-ties-losses.

Dataset	NGBoost	PGBM	CBU	IBUG	IBUG+CBU
Ames	2.0e+05$_{(3492)}$	59165$_{(1952)}$	66337$_{(2499)}$	**57219**$_{(1941)}$	**55551**$_{(1994)}$
Bike	66.4$_{(6.425)}$	7.048$_{(0.411)}$	**4.270**$_{(0.136)}$	6.775$_{(0.324)}$	**4.263**$_{(0.191)}$
California	1e+12$_{(1e+12)}$	1.257$_{(0.008)}$	1.168$_{(0.008)}$	1.230$_{(0.020)}$	**1.119**$_{(0.006)}$
Communities	0.361$_{(0.013)}$	0.366$_{(0.012)}$	0.352$_{(0.011)}$	**0.343**$_{(0.013)}$	**0.339**$_{(0.011)}$
Concrete	17.2$_{(0.917)}$	11.1$_{(0.698)}$	**10.3**$_{(0.491)}$	12.1$_{(0.800)}$	**10.1**$_{(0.523)}$
Energy	2.711$_{(0.198)}$	**0.814**$_{(0.050)}$	0.998$_{(0.067)}$	0.912$_{(0.083)}$	**0.819**$_{(0.064)}$
Facebook	28.4$_{(0.909)}$	26.8$_{(1.125)}$	21.6$_{(0.692)}$	**17.4**$_{(0.476)}$	**17.1**$_{(0.509)}$
Kin8nm	0.458$_{(0.003)}$	0.311$_{(0.007)}$	0.292$_{(0.005)}$	0.302$_{(0.009)}$	**0.262**$_{(0.005)}$
Life	17.5$_{(9.900)}$	5.051$_{(0.239)}$	4.617$_{(0.198)}$	5.093$_{(0.264)}$	**4.332**$_{(0.207)}$
MEPS	42.2$_{(1.973)}$	44.3$_{(1.294)}$	**37.7**$_{(1.223)}$	**38.3**$_{(1.375)}$	**37.2**$_{(1.254)}$
MSD	24.5$_{(0.039)}$	24.8$_{(0.035)}$	22.3$_{(0.020)}$	22.4$_{(0.029)}$	**22.0**$_{(0.025)}$
Naval	0.014$_{(3e-04)}$	0.003$_{(3e-04)}$	0.002$_{(2e-05)}$	0.001$_{(3e-05)}$	**0.001**$_{(1e-05)}$
News	**16557**$_{(519)}$	**16242**$_{(556)}$	**16166**$_{(580)}$	18694$_{(373)}$	**16426**$_{(551)}$
Obesity	15.5$_{(0.153)}$	9.731$_{(0.125)}$	**8.747**$_{(0.083)}$	10.7$_{(0.139)}$	9.162$_{(0.086)}$
Power	10.6$_{(0.136)}$	8.146$_{(0.122)}$	**7.837**$_{(0.165)}$	8.512$_{(0.152)}$	**7.803**$_{(0.156)}$
Protein	36689$_{(36570)}$	10.1$_{(0.149)}$	9.277$_{(0.062)}$	9.322$_{(0.045)}$	**8.853**$_{(0.052)}$
STAR	642$_{(7.014)}$	637$_{(6.564)}$	**636**$_{(6.131)}$	**630**$_{(4.545)}$	**630**$_{(4.968)}$
Superconductor	12.0$_{(0.133)}$	0.776$_{(0.023)}$	0.755$_{(0.030)}$	1.150$_{(0.060)}$	**0.692**$_{(0.033)}$
Synthetic	28.4$_{(0.228)}$	**28.1**$_{(0.188)}$	**28.1**$_{(0.211)}$	**28.0**$_{(0.197)}$	**28.0**$_{(0.199)}$
Wave	3e+06$_{(3727)}$	20256$_{(323)}$	11748$_{(55.8)}$	16669$_{(117)}$	**10569**$_{(47.4)}$
Wine	1.930$_{(0.023)}$	**1.723**$_{(0.032)}$	1.793$_{(0.035)}$	**1.716**$_{(0.030)}$	**1.692**$_{(0.031)}$
Yacht	5.798$_{(0.808)}$	**1.621**$_{(0.248)}$	**1.796**$_{(0.419)}$	**1.955**$_{(0.433)}$	**1.619**$_{(0.406)}$
IBUG	18-3-1	8-9-5	4-8-10	-	0-6-16
IBUG+CBU	18-4-0	16-6-0	16-4-2	16-6-0	-

Calibration Error. Table B.8 shows the average MACE (mean absolute calibration error) and sharpness scores. Sharpness quantifies the average of the standard deviations and thus does not depend on the actual ground-truth label; therefore, MACE and sharpness are shown together, with better methods having both low calibration error and low sharpness scores.

We observe that NGBoost is particularly well-calibrated, but lacks sharpness, meaning the prediction intervals of NGBoost are generally too wide. PGBM tends to have very sharp prediction intervals, but high calibration error. In contrast, CBU tends to achieve both low calibration error and high sharpness in relation to the other methods. However, these results are with variance calibration (§4.1.2), which we note has a significant impact on the CBU approach. For example, the median improvement in MACE score (over datasets) for CBU when using variance calibration vs. without is greater than 3x.

Table B.8. Probabilistic (MACE ↓ / sharpness ↓) performance. Standard errors are omitted for brevity.

Dataset	NGBoost	PGBM	CBU	IBUG	IBUG+CBU
Ames	0.082/74148	**0.040**/**18432**	0.073/18867	0.068/23186	0.063/19791
Bike	0.070/190	0.140/2.077	**0.045**/2.136	0.096/**1.272**	0.051/1.595
California	**0.014**/3e+13	0.053/**0.344**	0.021/0.367	0.089/0.382	0.037/0.364
Communities	0.039/0.129	0.067/**0.120**	0.051/0.136	**0.035**/0.133	0.048/0.133
Concrete	0.056/6.889	0.068/3.002	0.096/3.177	0.115/**2.503**	**0.054**/2.708
Energy	0.127/1.497	0.093/0.252	0.054/0.373	0.103/**0.249**	**0.053**/0.296
Facebook	0.094/9.171	0.206/**4.309**	0.072/7.332	**0.061**/18.9	0.091/12.5
Kin8nm	0.020/0.182	0.037/0.108	**0.020**/0.096	0.126/**0.071**	0.045/0.081
Life	**0.039**/111	0.069/**1.103**	0.079/1.189	0.115/1.401	0.069/1.216
MEPS	**0.030**/**6.680**	0.074/8.200	0.119/14.1	0.086/17.2	0.106/15.3
MSD	**0.007**/7.749	0.036/**7.436**	0.012/8.137	0.039/9.088	0.031/8.519
Naval	**0.032**/0.006	0.279/1e-03	0.048/6e-04	0.059/**5e-04**	0.086/5e-04
News	0.104/**2170**	**0.085**/3289	0.101/2975	0.202/4803	0.109/3498
Obesity	0.012/5.996	0.065/3.451	**0.006**/3.102	0.095/2.957	0.043/**2.956**
Power	0.020/3.761	0.026/2.558	**0.018**/**2.299**	0.030/3.328	0.019/2.729
Protein	0.029/2e+06	0.076/**2.823**	0.037/3.144	**0.016**/3.977	0.046/3.498
STAR	0.025/248	0.031/250	0.030/**242**	**0.023**/245	0.025/243
Superconductor	0.074/7.993	0.102/0.240	**0.028**/0.322	0.205/**0.208**	0.041/0.240
Synthetic	**0.012**/10.4	0.023/10.9	0.019/**10.4**	0.012/10.4	0.014/10.4
Wave	0.129/1e+06	0.018/6403	**0.007**/**4310**	0.089/6496	0.042/5127
Wine	**0.017**/0.694	0.070/**0.540**	0.027/0.575	0.091/0.643	0.061/0.600
Yacht	0.115/4.057	0.174/0.690	0.098/0.508	0.124/**0.371**	**0.078**/0.412

B.1.5 Runtime. Tables B.9 and B.10 provide detailed runtime results for each method. Results are averaged over 10 folds, and standard deviations are shown in subscripted parentheses; lower is better. The last row in each table shows the Geometric mean over all datasets.

Table B.9. Total train (including tuning) time (in seconds).

Dataset	NGBoost	PGBM	CBU	IBUG
Ames	**417**$_{(587)}$	1.4e+05$_{(25170)}$	23181$_{(258)}$	22264$_{(796)}$
Bike	**195**$_{(143)}$	58246$_{(8207)}$	23207$_{(239)}$	22417$_{(794)}$
California	**315**$_{(90.8)}$	16958$_{(1173)}$	23141$_{(253)}$	22530$_{(649)}$
Communities	**38.0**$_{(21.2)}$	24491$_{(4246)}$	23023$_{(260)}$	22429$_{(483)}$
Concrete	**57.1**$_{(22.9)}$	4130$_{(3621)}$	22953$_{(265)}$	22402$_{(577)}$
Energy	**35.3**$_{(33.0)}$	2706$_{(601)}$	22783$_{(278)}$	22423$_{(602)}$
Facebook	**731**$_{(659)}$	3.5e+05$_{(58586)}$	23061$_{(310)}$	23145$_{(517)}$
Kin8nm	**77.8**$_{(39.6)}$	10489$_{(2181)}$	23142$_{(296)}$	22694$_{(529)}$
Life	**105**$_{(87.4)}$	83814$_{(25313)}$	23082$_{(273)}$	20531$_{(7050)}$
MEPS	**351**$_{(477)}$	2.3e+05$_{(41039)}$	23139$_{(327)}$	20491$_{(7004)}$
MSD	**11720**$_{(1022)}$	2.2e+05$_{(34478)}$	23972$_{(258)}$	52760$_{(16670)}$
Naval	**847**$_{(1804)}$	38882$_{(11481)}$	23133$_{(210)}$	20607$_{(7059)}$
News	**2275**$_{(138)}$	2.4e+05$_{(60642)}$	22960$_{(258)}$	22492$_{(448)}$
Obesity	**1569**$_{(2208)}$	3.2e+05$_{(64847)}$	23169$_{(259)}$	21040$_{(7086)}$
Power	**107**$_{(53.0)}$	12556$_{(1459)}$	23042$_{(338)}$	22445$_{(298)}$
Protein	**1430**$_{(2298)}$	40132$_{(5779)}$	23043$_{(277)}$	22865$_{(344)}$
STAR	**17.0**$_{(5.788)}$	20797$_{(3481)}$	22852$_{(263)}$	22074$_{(432)}$
Superconductor	**215**$_{(29.3)}$	2.1e+05$_{(42859)}$	23291$_{(706)}$	22503$_{(426)}$
Synthetic	**439**$_{(351)}$	1.0e+05$_{(15729)}$	23068$_{(333)}$	22394$_{(532)}$
Wave	**3487**$_{(342)}$	1.0e+05$_{(16204)}$	23394$_{(173)}$	44282$_{(16994)}$
Wine	**33.1**$_{(19.2)}$	15067$_{(2804)}$	20942$_{(7191)}$	22269$_{(451)}$
Yacht	**51.3**$_{(41.8)}$	1965$_{(87.7)}$	22915$_{(239)}$	22184$_{(433)}$
Geo. mean	**265**	43604	23017	23726

Table B.10. Average prediction time per text example (in milliseconds).

Dataset	NGBoost	PGBM	CBU	IBUG
Ames	**5.583**$_{(5.778)}$	9.505$_{(2.426)}$	**0.066**$_{(0.010)}$	4.851$_{(2.766)}$
Bike	**0.514**$_{(0.815)}$	**7.705**$_{(8.198)}$	**0.010**$_{(0.002)}$	61.6$_{(21.1)}$
California	0.243$_{(0.082)}$	**5.659**$_{(9.562)}$	**0.004**$_{(0.001)}$	23.4$_{(5.265)}$
Communities	0.393$_{(0.170)}$	11.5$_{(0.941)}$	**0.027**$_{(0.010)}$	1.803$_{(1.118)}$
Concrete	2.154$_{(0.884)}$	**44.8**$_{(57.5)}$	**0.043**$_{(0.019)}$	1.876$_{(0.726)}$
Energy	1.830$_{(1.369)}$	32.9$_{(12.1)}$	**0.053**$_{(0.027)}$	1.135$_{(0.300)}$
Facebook	0.533$_{(0.469)}$	**5.148**$_{(7.166)}$	**0.024**$_{(0.004)}$	105$_{(62.4)}$
Kin8nm	0.194$_{(0.107)}$	168$_{(89.7)}$	**0.008**$_{(0.002)}$	4.713$_{(0.454)}$
Life	1.466$_{(1.171)}$	31.5$_{(28.1)}$	**0.064**$_{(0.037)}$	6.376$_{(1.275)}$
MEPS	0.465$_{(0.121)}$	**9.510**$_{(23.6)}$	**0.005**$_{(0.002)}$	8.845$_{(7.866)}$
MSD	1.712$_{(0.347)}$	25.3$_{(1.933)}$	**0.003**$_{(7e-04)}$	603$_{(97.4)}$
Naval	0.280$_{(0.129)}$	**187**$_{(253)}$	**0.010**$_{(0.007)}$	41.9$_{(22.3)}$
News	0.577$_{(0.100)}$	0.771$_{(0.403)}$	**0.002**$_{(1e-03)}$	40.0$_{(38.0)}$
Obesity	0.988$_{(0.475)}$	10.1$_{(6.503)}$	**0.020**$_{(0.003)}$	110$_{(9.535)}$
Power	0.252$_{(0.115)}$	6.904$_{(4.256)}$	**0.007**$_{(0.002)}$	18.3$_{(17.5)}$
Protein	0.154$_{(0.080)}$	130$_{(73.7)}$	**0.004**$_{(7e-04)}$	90.8$_{(25.7)}$
STAR	0.353$_{(0.121)}$	10.0$_{(1.191)}$	**0.038**$_{(0.010)}$	0.937$_{(0.369)}$
Superconductor	0.096$_{(0.055)}$	**27.1**$_{(79.5)}$	**0.005**$_{(0.002)}$	52.6$_{(25.8)}$
Synthetic	**0.482**$_{(0.541)}$	2.461$_{(0.524)}$	**0.009**$_{(0.005)}$	**7.595**$_{(10.1)}$
Wave	**1.018**$_{(1.339)}$	**9.116**$_{(17.0)}$	**0.003**$_{(3e-04)}$	719$_{(106)}$
Wine	0.135$_{(0.061)}$	**280**$_{(281)}$	**0.009**$_{(0.002)}$	6.001$_{(1.696)}$
Yacht	4.192$_{(2.669)}$	71.5$_{(12.1)}$	**0.124**$_{(0.081)}$	1.237$_{(0.334)}$
Geo. mean	0.585	18.5	**0.013**	15.9

B.2 Additional Experiments

In this section, we present additional experimental results.

B.2.1 Probabilistic Performance Without Variance Calibration.

Tables B.11 and B.12 show the probabilistic performance of each method *without* variance calibration. Even without variance calibration, IBUG+CBU generally outperforms competing methods. Standard errors are shown in subscripted parentheses.

Table B.11. Probabilistic (CRPS ↓) performance *without* variance calibration.

Dataset	NGBoost	PGBM	CBU	IBUG	IBUG+CBU
Ames	$38279_{(564)}$	$12173_{(484)}$	$11948_{(386)}$	$\mathbf{10442}_{(373)}$	$\mathbf{10208}_{(392)}$
Bike	$13.9_{(1.856)}$	$1.274_{(0.054)}$	$\mathbf{0.835}_{(0.035)}$	$1.899_{(0.224)}$	$1.219_{(0.105)}$
California	$2e{+}11_{(2e+11)}$	$0.227_{(0.004)}$	$0.221_{(0.001)}$	$0.213_{(1e-03)}$	$\mathbf{0.207}_{(9e-04)}$
Communities	$0.068_{(0.002)}$	$0.077_{(0.004)}$	$0.070_{(0.002)}$	$\mathbf{0.065}_{(0.002)}$	$\mathbf{0.065}_{(0.002)}$
Concrete	$3.395_{(0.181)}$	$1.932_{(0.088)}$	$1.994_{(0.095)}$	$1.938_{(0.079)}$	$\mathbf{1.780}_{(0.085)}$
Energy	$0.539_{(0.042)}$	$\mathbf{0.151}_{(0.007)}$	$0.207_{(0.010)}$	$0.481_{(0.041)}$	$0.293_{(0.022)}$
Facebook	$4.022_{(0.099)}$	$3.860_{(0.149)}$	$3.214_{(0.058)}$	$3.072_{(0.066)}$	$\mathbf{2.971}_{(0.072)}$
Kin8nm	$0.095_{(6e-04)}$	$0.069_{(0.003)}$	$0.063_{(8e-04)}$	$0.052_{(4e-04)}$	$\mathbf{0.052}_{(6e-04)}$
Life	$2.897_{(1.465)}$	$0.836_{(0.035)}$	$0.852_{(0.030)}$	$0.794_{(0.022)}$	$\mathbf{0.739}_{(0.024)}$
MEPS	$\mathbf{5.529}_{(0.196)}$	$6.725_{(0.126)}$	$6.050_{(0.109)}$	$6.146_{(0.113)}$	$6.022_{(0.114)}$
MSD	$4.525_{(0.005)}$	$5.767_{(0.006)}$	$4.364_{(0.004)}$	$4.410_{(0.005)}$	$\mathbf{4.342}_{(0.004)}$
Naval	$0.003_{(6e-05)}$	$0.005_{(0.002)}$	$3e{-}04_{(3e-06)}$	$3e{-}04_{(2e-06)}$	$\mathbf{3e{-}04}_{(2e-06)}$
News	$\mathbf{2476}_{(38.9)}$	$2628_{(94.9)}$	$2712_{(59.7)}$	$2669_{(43.9)}$	$2593_{(49.7)}$
Obesity	$3.208_{(0.028)}$	$1.860_{(0.022)}$	$\mathbf{1.754}_{(0.017)}$	$1.882_{(0.019)}$	$1.772_{(0.018)}$
Power	$2.104_{(0.024)}$	$1.585_{(0.057)}$	$1.572_{(0.024)}$	$1.542_{(0.020)}$	$\mathbf{1.488}_{(0.022)}$
Protein	$5427_{(5409)}$	$1.932_{(0.014)}$	$1.822_{(0.010)}$	$1.785_{(0.008)}$	$\mathbf{1.740}_{(0.009)}$
STAR	$132_{(1.697)}$	$157_{(6.908)}$	$132_{(1.540)}$	$\mathbf{129}_{(1.225)}$	$\mathbf{130}_{(1.327)}$
Superconductor	$3.200_{(0.031)}$	$\mathbf{0.134}_{(0.005)}$	$0.151_{(0.004)}$	$0.303_{(0.025)}$	$0.201_{(0.013)}$
Synthetic	$5.778_{(0.043)}$	$6.946_{(0.242)}$	$\mathbf{5.769}_{(0.049)}$	$\mathbf{5.731}_{(0.040)}$	$\mathbf{5.735}_{(0.042)}$
Wave	$5.7e{+}05_{(886)}$	$4152_{(247)}$	$\mathbf{2350}_{(10.3)}$	$4905_{(12.3)}$	$3112_{(8.952)}$
Wine	$0.385_{(0.005)}$	$0.383_{(0.015)}$	$0.355_{(0.007)}$	$\mathbf{0.322}_{(0.006)}$	$\mathbf{0.321}_{(0.006)}$
Yacht	$1.187_{(0.142)}$	$\mathbf{0.310}_{(0.056)}$	$\mathbf{0.291}_{(0.050)}$	$0.644_{(0.068)}$	$0.394_{(0.042)}$
IBUG W-T-L	16-4-2	12-4-6	10-4-8	-	0-5-17
IBUG+CBU W-T-L	17-3-2	15-4-3	14-2-6	17-5-0	-

Table B.12. Probabilistic (NLL ↓) performance *without* variance calibration.

Dataset	NGBoost	PGBM	CBU	IBUG	IBUG+CBU
Ames	11.3$_{(0.023)}$	23.7$_{(6.813)}$	1676$_{(1093)}$	**11.2**$_{(0.031)}$	**11.3**$_{(0.070)}$
Bike	1.942$_{(0.024)}$	11.1$_{(4.073)}$	**1.264**$_{(0.080)}$	2.958$_{(0.106)}$	2.386$_{(0.091)}$
California	0.551$_{(0.010)}$	7.821$_{(7.026)}$	2.261$_{(0.341)}$	0.484$_{(0.009)}$	**0.375**$_{(0.009)}$
Communities	**-7e-01**$_{(0.056)}$	20.7$_{(7.235)}$	2.438$_{(1.168)}$	**-6e-01**$_{(0.136)}$	**-4e-01**$_{(0.269)}$
Concrete	3.062$_{(0.031)}$	3.102$_{(0.277)}$	684$_{(358)}$	**2.848**$_{(0.055)}$	**2.822**$_{(0.093)}$
Energy	**0.670**$_{(0.250)}$	**0.481**$_{(0.341)}$	6.129$_{(3.597)}$	1.461$_{(0.113)}$	1.048$_{(0.162)}$
Facebook	2.099$_{(0.026)}$	14.7$_{(5.523)}$	5.147$_{(1.834)}$	2.195$_{(0.070)}$	**2.044**$_{(0.045)}$
Kin8nm	-4e-01$_{(0.007)}$	35.0$_{(23.1)}$	59.5$_{(24.2)}$	-8e-01$_{(0.009)}$	**-9e-01**$_{(0.024)}$
Life	2.188$_{(0.044)}$	23.5$_{(20.6)}$	71.9$_{(38.8)}$	**1.889**$_{(0.038)}$	**1.885**$_{(0.100)}$
MEPS	**3.732**$_{(0.056)}$	11.3$_{(3.246)}$	**3.722**$_{(0.044)}$	3.820$_{(0.064)}$	**3.678**$_{(0.054)}$
MSD	3.454$_{(0.002)}$	65.6$_{(0.162)}$	3.450$_{(0.004)}$	3.415$_{(0.002)}$	**3.383**$_{(0.002)}$
Naval	-5e+00$_{(0.007)}$	-4e+00$_{(0.357)}$	-5e+00$_{(0.057)}$	-6e+00$_{(0.007)}$	**-6e+00**$_{(0.006)}$
News	**10.9**$_{(0.335)}$	130$_{(49.5)}$	**10.8**$_{(0.368)}$	**11.0**$_{(0.415)}$	**10.7**$_{(0.307)}$
Obesity	2.940$_{(0.003)}$	2.603$_{(0.015)}$	**2.488**$_{(0.009)}$	2.646$_{(0.009)}$	**2.493**$_{(0.009)}$
Power	2.769$_{(0.042)}$	11.0$_{(8.270)}$	5.304$_{(0.672)}$	**2.575**$_{(0.036)}$	**2.569**$_{(0.057)}$
Protein	2.841$_{(0.015)}$	5.299$_{(0.268)}$	3.291$_{(0.042)}$	2.747$_{(0.123)}$	**2.531**$_{(0.028)}$
STAR	6.872$_{(0.015)}$	23.2$_{(5.203)}$	6.989$_{(0.040)}$	**6.853**$_{(0.008)}$	**6.857**$_{(0.012)}$
Superconductor	**12.1**$_{(13.4)}$	10.5$_{(4.631)}$	**-6e-03**$_{(0.093)}$	1.151$_{(0.111)}$	0.602$_{(0.094)}$
Synthetic	3.746$_{(0.008)}$	27.3$_{(6.288)}$	3.782$_{(0.032)}$	**3.738**$_{(0.007)}$	**3.744**$_{(0.010)}$
Wave	10.7$_{(0.002)}$	22.2$_{(7.985)}$	**9.679**$_{(0.004)}$	10.9$_{(0.004)}$	10.4$_{(0.004)}$
Wine	1.029$_{(0.014)}$	109$_{(24.9)}$	578$_{(428)}$	**0.910**$_{(0.016)}$	0.968$_{(0.030)}$
Yacht	**0.904**$_{(0.232)}$	7.227$_{(4.096)}$	4.770$_{(2.330)}$	1.502$_{(0.308)}$	**1.204**$_{(0.519)}$
IBUG W-T-L	11-7-4	10-10-2	8-9-5	-	1-10-11
IBUG+CBU W-T-L	13-7-2	11-10-1	8-11-3	11-10-1	-

B.2.2 Comparison to k-Nearest Neighbors.

In this section, we compare IBUG to k-nearest neighbors, in which similarity is defined by Euclidean distance. For the nearest-neighbors approach, we tune two different k values, one for estimating the conditional mean, and one for estimating the variance. We also apply standard scaling to the data before training, and denote this method kNN in our results. Table B.13 shows that IBUG is consistently better than kNN in terms of probabilistic performance. However, we note that point predictions from GBRTs is typically better than kNNs, thus we also compare IBUG to a variant of kNN that uses CatBoost as a base model to estimate the conditional mean.

Euclidean Distance vs. Affinity. To test which similarity measure (Euclidean distance or affinity) is more effective, we use the output from CatBoost to model the conditional mean, we then use kNN or IBUG to find their respective k-nearest training examples to estimate the variance; we denote these methods kNN-CB[3] and IBUG-CB. For kNN-CB, we also reduce the dimensionality of the data by only using the most important features identified by the CatBoost model;[4] this helps kNN-CB combat the curse of dimensionality when computing similarity. Results of this comparison are in Table B.13, in which we observe IBUG-CB is always statistically the same or better than kNN-CB. These results suggest affinity is a more effective similarity measure than Euclidean distance for uncertainty estimation in GBRTs.

[3] Again, we apply standard scaling to the data before training kNN-CB.

[4] We tune the number of important features to use for kNN-CB using values [5, 10, 20].

Table B.13. Probabilistic (CRPS) performance comparison of IBUG against two different nearest-neighbor models. kNN estimates the conditional mean and variance using two different k values; and kNN-CB estimates the variance in the same way as kNN, but uses the scalar output from the CatBoost model to estimate the conditional mean. Overall, these results suggest affinity is a better measure of similarity than Euclidean distance for uncertainty estimation in GBRTs.

Dataset	kNN	kNN-CB	IBUG-CB
Bike	**0.932**$_{(0.029)}$	**0.978**$_{(0.049)}$	**0.974**$_{(0.048)}$
California	0.579$_{(0.001)}$	0.219$_{(1e-03)}$	**0.213**$_{(9e-04)}$
Communities	0.072$_{(0.002)}$	**0.065**$_{(0.002)}$	**0.065**$_{(0.002)}$
Concrete	4.645$_{(0.140)}$	**1.872**$_{(0.085)}$	**1.849**$_{(0.098)}$
Energy	0.875$_{(0.016)}$	**0.153**$_{(0.010)}$	**0.143**$_{(0.009)}$
Facebook	5.613$_{(0.065)}$	3.275$_{(0.068)}$	**3.073**$_{(0.066)}$
Kin8nm	0.067$_{(7e-04)}$	**0.051**$_{(5e-04)}$	**0.051**$_{(6e-04)}$
Life	4.738$_{(0.078)}$	**0.785**$_{(0.024)}$	**0.794**$_{(0.023)}$
MEPS	7.283$_{(0.220)}$	**6.181**$_{(0.107)}$	**6.150**$_{(0.114)}$
MSD	5.312$_{(0.006)}$	4.446$_{(0.004)}$	**4.410**$_{(0.005)}$
Naval	8e-04$_{(2e-05)}$	3e-04$_{(2e-06)}$	**2e-04**$_{(2e-06)}$
News	2654$_{(52.0)}$	**2597**$_{(52.3)}$	**2545**$_{(41.0)}$
Obesity	5.526$_{(0.013)}$	**1.900**$_{(0.043)}$	**1.866**$_{(0.021)}$
Power	2.074$_{(0.022)}$	**1.553**$_{(0.020)}$	**1.542**$_{(0.020)}$
Protein	3.241$_{(0.010)}$	**1.787**$_{(0.008)}$	**1.784**$_{(0.008)}$
STAR	140$_{(1.553)}$	**129**$_{(1.204)}$	**130**$_{(1.214)}$
Superconductor	3.445$_{(0.041)}$	**0.156**$_{(0.006)}$	**0.153**$_{(0.006)}$
Synthetic	6.136$_{(0.047)}$	**5.735**$_{(0.039)}$	**5.731**$_{(0.040)}$
Wave	11987$_{(36.6)}$	2700$_{(17.0)}$	**2679**$_{(16.0)}$
Wine	0.445$_{(0.004)}$	**0.322**$_{(0.006)}$	**0.322**$_{(0.006)}$
Yacht	3.354$_{(0.408)}$	**0.275**$_{(0.048)}$	**0.276**$_{(0.048)}$
IBUG-CB W-T-L	21-1-0	10-12-0	-

B.2.3 Comparison to Bayesian Additive Regression Trees.

BART (Bayesian Additive Regression Trees) takes a Bayesian approach to uncertainty estimation in trees [42]. Although well-grounded theoretically, BART requires expensive sampling techniques such as MCMC (Markov Chain Monte Carlo) to provide approximate solutions.

In this section, we compare IBUG to BART using a popular open-source implementation.[5] However, due to BART's computational complexity, we tune the number of trees for both IBUG and BART using values [10, 50, 100, 200], set the number of chains for BART to 5, and run our comparison using a subset of the datasets in our empirical evaluation consisting of 11 relatively small datasets.

Tables B.14 and B.15 show IBUG consistently outperforms BART in terms of both probabilistic and point performance.

Table B.14. Probabilistic (CRPS ↓) performance comparison between IBUG and BART.

Dataset	BART	IBUG
Bike	$4.521_{(0.119)}$	$\mathbf{0.974}_{(0.048)}$
California	$0.285_{(0.001)}$	$\mathbf{0.213}_{(9e-04)}$
Communities	$0.072_{(0.002)}$	$\mathbf{0.065}_{(0.002)}$
Concrete	$3.067_{(0.073)}$	$\mathbf{1.849}_{(0.098)}$
Energy	$0.402_{(0.023)}$	$\mathbf{0.143}_{(0.009)}$
Kin8nm	$0.107_{(8e-04)}$	$\mathbf{0.051}_{(6e-04)}$
Naval	$1e\text{-}03_{(1e-05)}$	$\mathbf{2e\text{-}04}_{(2e-06)}$
Power	$2.225_{(0.018)}$	$\mathbf{1.542}_{(0.020)}$
STAR	$134_{(1.493)}$	$\mathbf{130}_{(1.214)}$
Wine	$0.394_{(0.005)}$	$\mathbf{0.322}_{(0.006)}$
Yacht	$0.849_{(0.039)}$	$\mathbf{0.276}_{(0.048)}$
IBUG W-T-L	11-0-0	-

[5] `https://github.com/JakeColtman/bartpy`

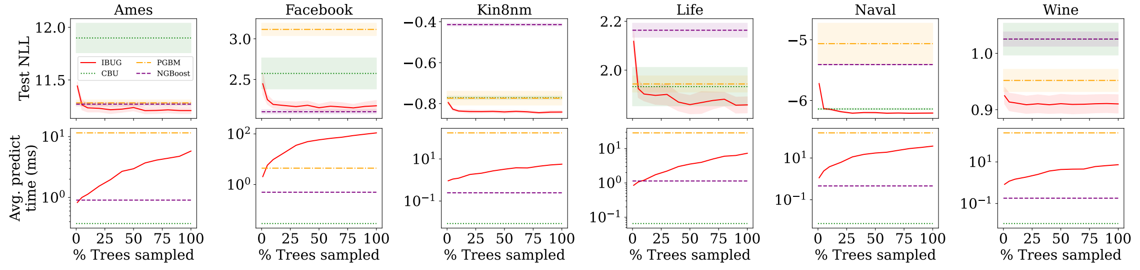
Table B.15. Point (RMSE ↓) performance comparison between IBUG and BART.

Dataset	BART	IBUG
Bike	$8.396_{(0.273)}$	$\mathbf{2.826}_{(0.200)}$
California	$0.547_{(0.003)}$	$\mathbf{0.432}_{(0.001)}$
Communities	$0.137_{(0.004)}$	$\mathbf{0.133}_{(0.004)}$
Concrete	$5.507_{(0.161)}$	$\mathbf{3.629}_{(0.183)}$
Energy	$0.685_{(0.039)}$	$\mathbf{0.264}_{(0.023)}$
Kin8nm	$0.186_{(0.001)}$	$\mathbf{0.086}_{(8e-04)}$
Naval	$0.002_{(2e-05)}$	$\mathbf{5e\text{-}04}_{(5e-06)}$
Power	$4.057_{(0.049)}$	$\mathbf{2.941}_{(0.059)}$
STAR	$234_{(2.479)}$	$\mathbf{228}_{(1.985)}$
Wine	$0.708_{(0.008)}$	$\mathbf{0.596}_{(0.012)}$
Yacht	$1.624_{(0.121)}$	$\mathbf{0.668}_{(0.125)}$
IBUG W-T-L	11-0-0	-

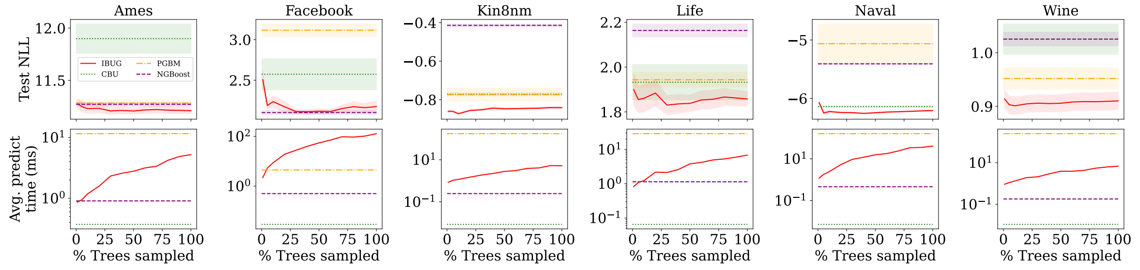
B.2.4 Different Tree-Sampling Strategies.

Figure B.1 shows the probabilistic (NLL) performance of IBUG as the number of trees sampled (τ) increases using three different sampling strategies: *uniformly at random*, *first-to-last*, and *last-to-first*.

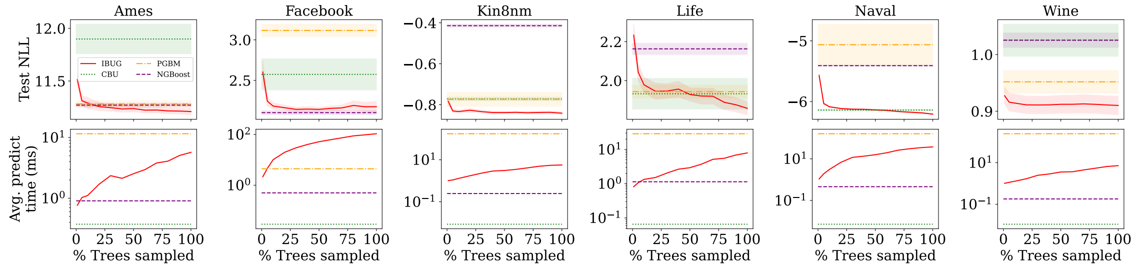
We observe that sampling trees *last-to-first* often requires sampling all trees in order to achieve the lowest NLL on the test set. When sampling *uniformly at random*, NLL tends to plateau starting around 10%. In contrast, sampling trees *first-to-last* on the Kin8nm, Naval, and Wine datasets requires 5% of the trees or less to result in the same or better NLL than when sampling all trees; these results provide some evidence that trees early in training contribute most, and suggest that sampling trees first-to-last may be most effective at obtaining the best probabilistic performance while sampling the fewest number of trees.



(a) Sampling trees *uniformly at random*.



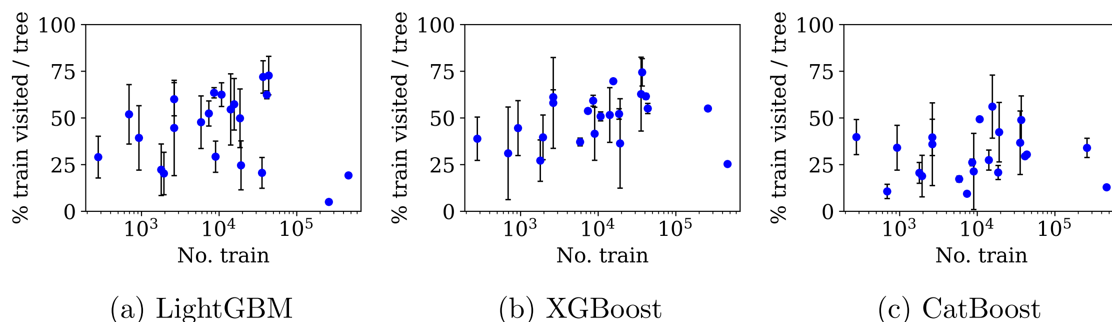
(b) Sampling trees *first-to-last*.



(c) Sampling trees *last-to-first*.

Figure B.1. Probabilistic (NLL) performance (lower is better) and average prediction time (in milliseconds) per test example (lower is better) as a function of τ for different sampling techniques. *Top*: sample trees uniformly at random, *middle*: sample trees first-to-last (in terms of boosting iteration), *bottom*: sample trees last-to-first. All methods result in similar prediction times; however, *first-to-last* sampling typically provides the best NLL with the fewest number of trees sampled.

B.2.5 Leaf Density. Figure B.2 shows the average percentage of train instances visited per tree as a function of the total number of training instances for each dataset. We note that for some datasets, CatBoost, LightGBM, and XGBoost induce regression trees with very dense leaves where over half the training instances belong to those leaves. Figure B.3 shows average leaf density for each tree in the GBRT.



(a) LightGBM (b) XGBoost (c) CatBoost

Figure B.2. Average % train visited / tree during affinity computation for each dataset. Results are averaged over test set w/ s.d; lower is better. In general, the number of training instances visited per tree is highly dependent on the dataset; and for some datasets, is also highly dependent on the test example (points with large standard deviations).

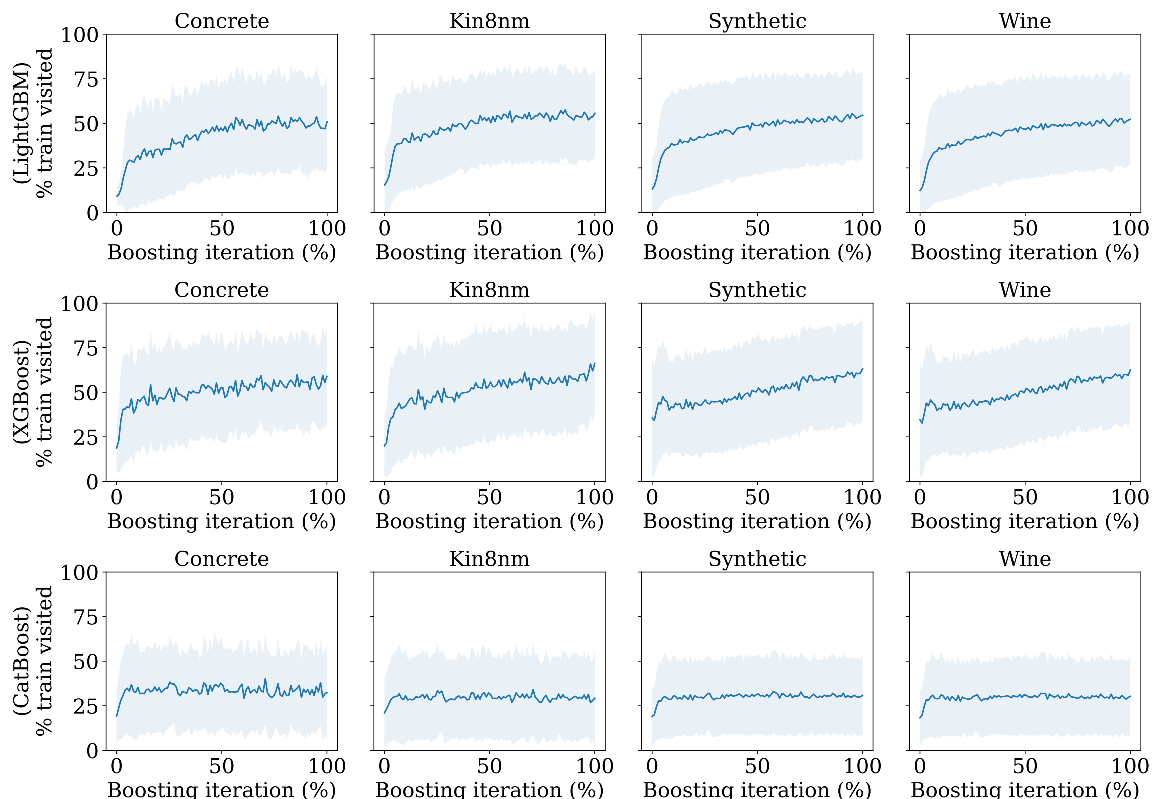


Figure B.3. Average % train visited at each iteration during affinity computation for different datasets. Results show averages over test set w/ s.d.; lower is better. Overall, leaf densities are dataset dependent. However, for LightGBM and XGBoost, weak learners later in training tend to pool a larger proportion of training instances into fewer leaves; in contrast, CatBoost has less dense leaves and training instances are more equally distributed among the leaves in each tree.

APPENDIX C

EFFICIENT MODEL ADAPTATION

In this chapter, we provide detailed proofs, implementation and experiment details, and additional analyses for DaRE RF from Chapter 5.

C.1 Algorithmic Details

Here we provide proofs to all Theorems and Lemmas related to unlearning in DaRE RFs.

C.1.1 Exact Deletion: Proof of Theorem 5.1.1.

We use the following Lemma to help prove the theorem of exact deletion for DaRE forests.

Lemma C.1.1. *The probability of selecting a valid set of thresholds S from a dataset D and then subsequently resampling any invalidated thresholds after the deletion of $(x, y) \in D$ is equivalent to the probability of selecting S from an updated dataset $D \setminus (x, y)$.*

Proof. The probability of choosing a valid set of thresholds S from $D \setminus (x, y)$ is $P^A(S) = 1/\binom{n-m}{k}$ in which n is the number of valid thresholds before the deletion, k is the number of thresholds to sample from the set of valid thresholds, and m is the number of thresholds that become invalid due to the deletion of (x, y). The probability of ending up with thresholds S by first choosing some set S^* and then resampling any thresholds invalidated by removing (x, y) is:

$$P^{B+R}(S) = \frac{1}{\binom{n}{k}} \sum_{i=0}^{m} \frac{\binom{m}{i}\binom{k}{i}}{\binom{n-k-(m-i)}{i}},$$

in which $\binom{n}{k}$ is the number of valid threshold sets for D; S^* may have up to m invalid thresholds, thus $\binom{m}{i}\binom{k}{i}$ is the number of ways i invalid thresholds out of k chosen thresholds could be resampled from the set of m invalid thresholds; and

$\binom{n-k-(m-i)}{i}$ is the number of valid threshold sets that can be resampled to, starting at a set with i invalid thresholds.

In the simplest case, $m = 0$ and no thresholds are invalidated, so the probability of choosing S in the updated dataset and original dataset are identical: $1/\binom{n}{k} = 1/\binom{n-0}{k}$. In the next simplest case, $m = 1$ and only a single threshold is invalidated. Thus, we could arrive at S by first sampling it with the original dataset (probability $1/\binom{n}{k}$) or by first sampling one of the k sets that includes the invalidated threshold and is otherwise identical to S, followed by resampling that threshold from the remaining $(n-1)-(k-1)$ valid and unselected thresholds to obtain S.

Thus, the total probability (for $m = 1$) is:

$$\begin{aligned}
P^{B+R}(S) &= \frac{1}{\binom{n}{k}}\left(1 + \frac{k}{n-k}\right) \\
&= \frac{1}{\binom{n}{k}}\left(\frac{n}{n-k}\right) \\
&= \frac{k!\,(n-k)!\,n}{n!\,(n-k)} \\
&= \frac{k!(n-1-k)!}{(n-1)!} \\
&= \frac{1}{\binom{n-1}{k}} \\
&= P^{A}(S)
\end{aligned}$$

For $m > 1$, we can reduce it to the $m = 1$ case by viewing it as a sequence of invalidating one threshold at a time. After invalidating one of the thresholds, the probability remains uniform, so by induction it continues to remain uniform after a second deletion, or a third, or any number. □

Theorem. *Data deletion for DaRE forests is* exact *(see Eq. 2.12), meaning that removing instances from a DaRE model yields exactly the same model as retraining from scratch on updated data.*

Proof. Exact unlearning is defined as having the same probability distribution over models by deletion as by retraining (Eq. 2.12). For discrete attributes, the node statistics used in model updating are precisely those used for learning the initial structure, so as the statistics are updated, the structure is updated to match what would be learned from scratch (in distribution).

For continuous attributes, we first discretize by uniformly sampling k thresholds from the set of all valid thresholds for that attribute. As instances are removed, if one of the sampled thresholds becomes invalid, then those thresholds are resampled to obtain a set of valid thresholds. Lemma C.1.1 shows the resulting probability of each set of valid thresholds remains uniform, identical to what it would be if the model were retrained from scratch.

The same logic and lemma also applies for attributes. If a deletion causes one or more attributes to become invalid (i.e. no more valid thresholds to sample), then those attributes are resampled to obtain a set of valid attributes, with all sets of valid attributes being equally likely.

Since each decision node in the tree operates on its own partition of the data D, then updating all relevant decision nodes and leaf nodes results in the entire tree being updated to match the updated dataset. The extension to the forest follows since all trees are independent; thus, the probability of a DaRE forest after removing instances is the same as retraining the model from scratch on updated data.

□

C.1.2 Training Complexity: Proof of Theorem 5.1.2.

Theorem. *Given $n = |\mathcal{D}|$, T, $d_{\max}$, and $\tilde{p}$, the time complexity to train a DaRE forest is $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$.*

Proof. When training a DaRE tree, we begin by choosing a split for the root by iterating through all n training instances and scoring $\tilde{p}$ randomly selected attributes. Generalizing this to nodes at other depths, there are (at most) 2^d nodes at depth d, and each of the n training instances is assigned to one of these nodes. Choosing all splits at depth d thus requires a total time of $\mathcal{O}(\tilde{p}\,n)$ across all depth-d nodes, since we again process every training instance when finding the best split for each node. Summing over all depths, the total time is $\mathcal{O}(\tilde{p}\,n\,d_{\max})$ to train a single DaRE tree or $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$ to train a forest of T trees. □

C.1.3 Training Complexity: Proof of Theorem 5.1.3.

Theorem. *Given $d_{\max}$, $\tilde{p}$, and k, the time complexity to delete a single instance $(x, y) \in \mathcal{D}$ from a DaRE tree is $\mathcal{O}(\tilde{p}\,k\,d_{\max})$, if the tree structure is unchanged and the attribute thresholds remain valid. If a node with $|D|$ instances has an invalid attribute threshold, then the additional time to choose new thresholds is $\mathcal{O}(|D| \log |D|)$. If a node with $|D|$ instances at level d needs to be retrained, then the additional retraining time is $\mathcal{O}(\tilde{p}\,|D|\,(d_{\max} - d)))$.*

Proof. Deleting an instance from a DaRE tree (Alg. 5) requires traversing the tree from the root to a leaf, updating node statistics, retraining a subtree (if necessary), and removing the instance from the tree's set of instances. Since there are $\tilde{p}$ candidate attributes at each node, subtracting the influence of (x, y) from the node statistics and checking for invalid attribute thresholds requires $\mathcal{O}(\tilde{p})$ time. Recomputing the score for each attribute-threshold pair requires $\mathcal{O}(\tilde{p}\,k)$, since we

have the necessary statistics and computing the Gini index can be done in constant time for each pair. Across all depths up to $d_{\max}$, this is a total time of $\mathcal{O}(\tilde{p}\, k\, d_{\max})$.

Choosing new thresholds requires making a list of all attribute values at a node, along with the associated labels. This can be done by traversing the subtree rooted at the node, visiting each leaf and collecting the attribute values from the instances at that leaf. Let $|D|$ be the total number of these instances. Since the number of leaves is bounded by the number of instances, traversing the subtree can be done in $\mathcal{O}(|D|)$ time, plus $\mathcal{O}(|D| \log |D|)$ time to sort the values. The remaining work of making a list of valid thresholds, randomly choosing k thresholds, and computing statistics for these k thresholds can all be done in $\mathcal{O}(|D|)$, since each requires (at most) a single pass through all $|D|$ instances. Thus, the total time is $\mathcal{O}(|D| \log |D|)$.

If the best attribute-threshold pair at a node has changed, then the subtree must be retrained. Let $|D|$ be the number of instances at the node and d be its depth. The time for retraining a subtree is identical to the time for retraining a DaRE tree, except that the number of instances is $|D|$ and the maximum depth (relative to this node) is $(d_{\max} - d)$. Thus, the total time is $\mathcal{O}(\tilde{p}\,(d_{\max} - d)\,|D|)$. □

C.1.4 Space Complexity: Proof of Theorem 5.1.4.

Theorem. *Given $\mathcal{D}$, $d_{\max}$, k, T, and $\tilde{p}$, the space complexity of a DaRE forest is $\mathcal{O}(k\,\tilde{p}\,2^{d_{\max}}\,T + n\,T)$.*

Proof. The space complexity of a DaRE tree with a single decision node is $\mathcal{O}(k\,\tilde{p} + n)$ since we need to store a constant $\mathcal{O}(1)$ amount of metadata for k thresholds times $\tilde{p}$ attributes as well as n pointers (one for each training instance) partitioned across the leaves in the tree. For a single DaRE tree with multiple decision nodes, we

need to multiply the first term in the previous result by $2^{d_{\max}}$ since there may be $2^{d_{\max}}$ decision nodes in a single DaRE tree; the second term remains the same as the training instances are still partitioned across all leaves in the tree. Thus, the space complexity of a single DaRE tree is $\mathcal{O}(k\,\tilde{p}\,2^{d_{\max}} + n)$. For a DaRE forest, we need to multiply this result by T; thus, the space complexity of a DaRE forest is $\mathcal{O}(k\,\tilde{p}\,2^{d_{\max}}\,T + n\,T)$. □

Assuming that we have at least one training instance assigned to each leaf, the number of leaves in each tree is at most n, and thus the total number of nodes per tree is at most $2n - 1$. This gives us an alternate bound of $\mathcal{O}(k\ \tilde{p}\ n\ T)$, which is proportional to the size of the training data times the number of thresholds and trees in the forest (in the worst case).

C.1.5 Complexity of Slightly-Less-Naive Retraining. The complexity of naive retraining is the same as training a DaRE forest from scratch, $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$, where $n = |\mathcal{D}|$.

A slightly smarter approach is to retrain only the portion of each tree that depends on the deleted node. For example, if the best split at the root of the tree after deleting an instance is the same as it was before, then the data will be partitioned between its two children the same way as before. One part of this partition never contained the deleted instance, and that fraction of the tree is unchanged. The other part of the partition has been changed by this deletion, so we must recurse, but only in that half of the tree.

This is potentially more efficient, but the efficiency gains are still bounded relative to the naive retraining approach. Choosing the split at the root still requires iterating through all training instances to compute statistics for each

attribute (and each split of each continuous attribute), for a total time of $\mathcal{O}(T\,\tilde{p}\,n)$ across all T trees.

Since the total time for retraining is at most $\mathcal{O}(T\,\tilde{p}\,n\,d_{\max})$, the gain from this optimized approach is at *most* a factor of $d_{\max}$ (10-20 in our experiments). This is ignoring the cost of scoring splits at the lower levels in the tree, or retraining the lower levels of the tree (as is often required after data deletion). Thus, the gain will be smaller in practice.

Therefore, a slightly-less-naive approach to retraining random forests could improve over the naive approach, but would still be substantially slower than our methods, which achieve speedups of several orders of magnitude (see Figure 11 and Table 6).

C.1.6 Node Statistics. A DaRE tree may consist of three types of nodes: greedy decision nodes, random decision nodes, and leaf nodes. Each stores a constant amount of metadata to enable efficient updates. In addition to the following type-specific statistics, each node stores $|D|$ and $|D_{\cdot,1}|$ the number of instances and the number of positive instances at that node.

- Greedy decision nodes: For each threshold for an attribute, we store $|D_l|$, $|D_{l,1}|$. This is a sufficient set of statistics needed to recompute the Gini index (Eq. 2.1) or entropy (Eq. 2.2) split criterion scores. Since a threshold in a greedy decision node is the midpoint between two adjacent attribute values, we also keep track of how many positive instances and the total number of instances are in each attribute value set; by updating this information, a DaRE tree can sample a new threshold when one is no longer valid.

- Random decision nodes: After selecting a random attribute, and then a random threshold within that attribute's min. and max. value range, we

only store $|\mathcal{D}_l|$ and $|\mathcal{D}_r|$. Updating these statistics informs the DaRE tree when the threshold value is no longer within the min. and max. value range of that attribute. At that point, the random decision node is retrained.

– Leaf nodes: For each leaf, we store pointers to the training instances that traversed to that leaf. This enables the DaRE tree to collect these training instances when needing to retrain any ancestor decision nodes higher in the tree.

C.1.7 Batch Deletion. Batch deletion is almost the same as deleting one instance, except we may need to recurse down multiple branches of each tree to find all relevant instances to delete, and we only retrain a given node (at most) once, rather than (up to) once for each instance deleted. This will naturally be more efficient, but waiting for a large batch may not be possible.

C.1.8 Pseudocode. Algorithm 6 provides detailed pseudocode. [1]

Algorithm 6 Pseudocode for building/deleting an instance from a DaRE tree.

```
1: Train(data D, depth d):
2:     if stopping criteria reached then
3:         node ← LeafNode(D)
4:     else
5:         if d < d_rmax then
6:             node ← RandomNode(D)
7:         else
8:             node ← GreedyNode(D)
9:         D.ℓ, D.r ← split on threshold(node, D)
10:        node.ℓ ← Train(D_ℓ, d + 1)
11:        node.r ← Train(D_r, d + 1)
12:    return node

13: Delete(node, depth d, remove (x, y)):
14:     node ⇚ update +/total(node, (x, y))
15:     if node is a LeafNode then
16:         node ⇚ remove (x, y) from leaf
17:         node ⇚ recompute leaf value(node)
18:         remove (x, y) from database
19:     else
20:         node ⇚ update dec. stats(node, (x, y))
21:         if node is a RandomNode then
22:             node ← RandDel(node, d, (x, y))
23:         else
24:             node ← GreedyDel(node, d, (x, y))
25:         a, v ← node.selected attr., threshold
26:         if no retraining occurred then
27:             if x_{·,a} ≤ v then
28:                 Delete(node.ℓ, d + 1, (x, y))
29:             else
30:                 Delete(node.r, d + 1, (x, y))
31:     return node

32: GreedyDel(node, d, remove (x, y)):
33:     A ← node.sampled attributes
34:     Ā ← get invalid attributes(A)
35:     if |Ā| > 0 then
36:         D ← leaf instances(node) \ (x, y)
37:         A* ← resample invalid(Ā, D)
38:         A ← A \ Ā ∪ A*
39:     for a ∈ A do
40:         V ← a.sampled valid thresholds
41:         V̄ ← get invalid thresholds(V)
42:         if |V̄| > 0 then
43:             D ← leaf instances(node) \ (x, y)
44:             V* ← resample invalid(V̄, D, a)
45:             V ← V \ V̄ ∪ V*
46:     scores ← recompute split scores(node)
47:     node ⇚ select optimal split(scores)
48:     if optimal split has changed then
49:         a, v ← node.selected attr., threshold
50:         D_ℓ, D_r ← split data(D, a, v)
51:         node.ℓ ← Train(D_ℓ, d + 1)
52:         node.r ← Train(D_r, d + 1)
53:     return node
```

```
1: LeafNode(data D):
2:     node ← Node()
3:     node ← SaveNode(node, D)
4:     node ⇚²compute leaf value(node)
5:     node ⇚ save leaf instances(node, D)
6:     return node

7: RandomNode(data D):
8:     node ← Node()
9:     node ← SaveNode(node, D)
10:    a ← sample attribute(D)
11:    v ← sample threshold ∈ [a_min, a_max)
12:    node ← SaveThresh(node, D, a, v)
13:    return node

14: GreedyNode(data D):
15:     node ← Node()
16:     node ← SaveNode(node, D)
17:     node ⇚ sample p̃ attributes(node, D)
18:     for a ∈ node.sampled attributes do
19:         C ← get valid thresholds(D, a)
20:         V ← sample k valid thresholds(C)
21:         for v ∈ V do
22:             node ← SaveThresh(node, D a, v)
23:     scores ← compute split scores(node)
24:     node ⇚ select optimal split(scores)
25:     return node

26: SaveNode(node, data D):
27:     node ⇚ instance count(D)
28:     node ⇚ positive instance count(D)
29:     return node

30: SaveThresh(node, D, a, v):
31:     node ⇚ L count(D, a, v)
32:     if node is a GreedyNode then
33:         node ⇚ L+ count(D, a, v)
34:         node ⇚ L/R adj. feature val.(D, a, v)
35:         node ⇚ L/R adj. val− count(D, a, v)
36:         node ⇚ L/R adj. val+ count(D, a, v)
37:     return node

38: RandDel(node, d, remove (x, y)):
39:     if selected threshold is invalid then
40:         D ← leaf instances(node) \ (x, y)
41:         if sel. attr. (a) is still valid then
42:             v ← resample ∈ [a_min, a_max)
43:             D_ℓ, D_r ← split(D, a, v)
44:             node.ℓ ← Train(D_ℓ, d + 1)
45:             node.r ← Train(D_r, d + 1)
46:         else
47:             node ← Train(D, d)
48:     return node
```

[1] We also use "*node* ⇚ ..." to denote updates to a node or its data. Node statistics details are in §C.1.6. See `https://github.com/jjbrophy47/dare_rf` for additional details.

C.2 Implementation and Experiment Details

Experiments are run on an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.6GHz with 70GB of RAM. No parallelization is used when building the independent decision trees. DaRE RF is implemented in the C programming language via Cython, a Python package allowing the development of C extensions. Experiments are run using Python 3.7. Source code for DaRE RF and all experiments is available at `https://github.com/jjbrophy47/dare_rf`.

C.2.1 Datasets.

- Surgical [115] consists of 14,635 medical patient surgeries (3,690 positive cases), characterized by 25 attributes; the goal is to predict whether or not a patient had a complication from their surgery.

- Vaccine [30, 56] 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.

- Adult [59] 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.

- Bank Marketing [59, 147] 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.

- Flight Delays [168] 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.

– Diabetes [59, 198] consists of 101,766 instances of patient and hospital readmission outcomes (46,902 readmitted) characterized by 55 attributes.

– No Show [105] 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.

– Olympics [114] contains 206,165 Olympic events over 120 years of Olympic history. Each event contains information about the athlete, their country, which Olympics the event took place, the sport, and what type of medal the athlete received. The aim is to predict whether or not an athlete received a medal for each event they participated in.

– Census [59] contains 40 demographic and employment attributes on 299,285 people in the United States; the survey was conducted by the U.S. Census Bureau. The goal is to predict if a person's income level is more than $50K.

– Credit Card [113] contains 284,807 credit card transactions in September 2013 by European cardholders. The transactions took place over two days and contains 492 fraudulent charges (0.172% of all charges). There are 28 principal components resulting from PCA on the original dataset, and two additional fetures: 'time' and 'amount'. The aim is to predict whether a charge is fraudulent or not.

– Click-Through Rate (CTR) [51] contains the first 1,000,000 instances of the Criteo 1TB Click Logs dataset, in which each row represents an ad that was displayed and whether or not it had been clicked on (29,040 ads clicked). The dataset contains 13 numeric attributes and 26 categorical attributes. However, due to the extremely large number of values for the categorical attributes, we restrict our use of the dataset to the 13 numeric attributes. The aim is to predict whether or not an ad is clicked on.

– Twitter uses the first 1,000,000 tweets (169,471 spam) of the HSpam14 dataset [178]. 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.

– Synthetic [154] contains 1,000,000 instances normally distributed about the vertices of a 5-dimensional hypercube into 2 clusters per class. There are 5 informative attributes, 5 redundant attributes, and 30 useless attributes. There is interdependence between these attributes, and a randomly selected 5% of the labels are flipped to increase the difficulty of the classification task.

– Higgs [11, 59] contains 11,000,000 signal processes (5,829,123 Higgs bosons) characterized by 22 kinematic properties measured by detectors in a particle accelerator and 7 attributes derived from those properties. The goal is to distinguish between a background signal process and a Higgs bosons process.

For each dataset, we generate one-hot encodings for any categorical variable and leave all numeric and binary variables as is. For any dataset without a

Table C.1. Dataset summary including the main predictive performance metric used for each dataset, either average precision (AP) for datasets whose positive label percentage $< 1\%$, AUC for datasets between [1%, 20%], or accuracy (Acc.) for all remaining datasets.

Dataset	**#train**	**% Pos.**	**#test**	**% Pos.**	p	**Metric**
Surgical	11,708	25.30	2,927	25.00	90	Acc.
Vaccine	21,365	46.60	5,342	45.60	185	Acc.
Adult	32,561	24.00	16,281	23.60	107	Acc.
Bank Marketing	32,951	11.40	8,237	10.90	63	AUC
Flight Delays	80,000	18.90	20,000	19.50	648	AUC
Diabetes	81,412	46.00	20,353	46.50	253	Acc.
No Show	88,422	20.14	22,105	20.41	99	AUC
Olympics	164,932	14.60	41,233	14.60	1,004	AUC
Census	199,523	6.20	99,762	6.20	408	AUC
Credit Card	227,846	0.18	56,961	0.17	29	AP
CTR	800,000	2.89	200,000	2.98	13	AUC
Twitter	800,000	16.96	200,000	16.83	15	AUC
Synthetic	800,000	50.00	200,000	50.00	40	Acc.
Higgs	8,800,000	53.00	2,200,000	53.00	28	Acc.

designated train and test split, we randomly sample 80% of the data for training and use the rest for testing. Table C.1 summarizes the datasets after preprocessing.

C.2.2 Predictive Performance of DaRE Forests. If extremely randomized trees exhibit the same predictive performance as their greedy counterparts, then adding and removing data can be done by simply updating class counts at the leaves and only retraining if a chosen threshold is no longer within the range of a chosen split attribute for a given decision node. Thus, this section compares the predictive performance of a G-DaRE forest against:

- Random Trees: Extremely randomized trees [79] in which each decision node selects an attribute to split on uniformly at random, and then selects the threshold by sampling a value in that attribute's [min, max] range uniformly at random.

Table C.2. Predictive performance comparison of G-DaRE RF to: an extremely randomized trees model (RT) [79], an Extra Trees [79] model (ET), and a popular and widely used random forest implementation from Scikit-Learn (SKLearn) with (*) and without bootstrapping. The numbers in each cell represent either average precision, AUC, or accuracy as specified by Table C.1; results are averaged over five runs and the standard error is shown in subscripted parentheses.

Dataset	RT	ET	SKRF	SKRF*	G-DaRE RF
Surgical	$0.783_{(0.001)}$	$0.805_{(0.001)}$	$0.848_{(0.001)}$	$0.846_{(0.001)}$	$\mathbf{0.867}_{(0.001)}$
Vaccine	$0.769_{(0.001)}$	$0.795_{(0.001)}$	$\mathbf{0.796}_{(0.001)}$	$0.793_{(0.002)}$	$0.794_{(0.001)}$
Adult	$0.802_{(0.003)}$	$0.847_{(0.001)}$	$\mathbf{0.863}_{(0.000)}$	$\mathbf{0.863}_{(0.000)}$	$0.862_{(0.001)}$
Bank Mktg.	$0.879_{(0.001)}$	$0.924_{(0.000)}$	$\mathbf{0.940}_{(0.001)}$	$\mathbf{0.940}_{(0.001)}$	$\mathbf{0.940}_{(0.001)}$
Flight Delays	$0.650_{(0.009)}$	$0.725_{(0.001)}$	$0.729_{(0.001)}$	$0.729_{(0.000)}$	$\mathbf{0.739}_{(0.000)}$
Diabetes	$0.551_{(0.003)}$	$0.631_{(0.001)}$	$0.643_{(0.000)}$	$0.642_{(0.001)}$	$\mathbf{0.645}_{(0.000)}$
No Show	$0.694_{(0.001)}$	$0.710_{(0.000)}$	$0.732_{(0.000)}$	$0.731_{(0.000)}$	$\mathbf{0.736}_{(0.000)}$
Olympics	$0.835_{(0.001)}$	$0.820_{(0.001)}$	$0.819_{(0.001)}$	$0.820_{(0.000)}$	$\mathbf{0.871}_{(0.000)}$
Credit Card	$0.799_{(0.002)}$	$0.840_{(0.004)}$	$0.837_{(0.002)}$	$0.831_{(0.005)}$	$\mathbf{0.846}_{(0.001)}$
Census	$0.915_{(0.001)}$	$0.936_{(0.000)}$	$0.945_{(0.000)}$	$0.945_{(0.000)}$	$\mathbf{0.946}_{(0.000)}$
CTR	$0.668_{(0.001)}$	$0.683_{(0.000)}$	$\mathbf{0.702}_{(0.000)}$	$0.700_{(0.000)}$	$0.701_{(0.000)}$
Twitter	$0.883_{(0.001)}$	$0.923_{(0.001)}$	$\mathbf{0.943}_{(0.000)}$	$0.942_{(0.000)}$	$\mathbf{0.943}_{(0.000)}$
Synthetic	$0.793_{(0.002)}$	$0.909_{(0.001)}$	$\mathbf{0.946}_{(0.001)}$	$0.945_{(0.000)}$	$0.945_{(0.000)}$
Higgs	$0.608_{(0.001)}$	$0.700_{(0.000)}$	$\mathbf{0.746}_{(0.000)}$	$0.744_{(0.000)}$	$0.744_{(0.000)}$

– Extra Trees: Similar to the extremely randomized trees model [79], except each decision node selects $\lfloor\sqrt{p}\rfloor$ attributes at random; a threshold is then selected for each attribute by sampling a value in that attribute's [min, max] range uniformly at random. Then, a split criterion such as Gini index or mutual information is computed for each attribute-threshold pair, and the best threshold is chosen as the split for that node.

– SKLearn RF: Standard RF implementation from Scikit-Learn [154].

– SKLearn RF (w/ bootstrap): Standard RF implementation from Scikit-Learn [154] with bootstrapping.

Table C.2 reports the predictive performance of each model on the test set after tuning using 5-fold cross-validation. We tune the number of trees in the forest

using values [10, 25, 50, 100, 250], and the maximum depth using values [1, 3, 5, 10, 20]. The maximum number of randomly selected attributes to consider at each split is set to $\lfloor\sqrt{p}\rfloor$. For the G-DaRE model, we also tune the number of thresholds to consider for each attribute, k, using values [5, 10, 25, 50]. We use 50%, 25%, 2.5%, and 2.5% of the training data to tune the Twitter, Synthetic, Click-Through Rate, and Higgs datasets, respectively, and 100% for all other datasets. Selected values for all hyperparameters are in Table C.3.

We find the predictive performance of the Random Trees and Extra Trees models to be consistently worse than the SKLearn and G-DaRE models. We also find that bootstrapping has a negligible effect on the SKLearn models. Finally, we observe that the predictive performance of the G-DaRE model is nearly identical to that of SKLearn RF, in which their scores are within 0.2% on 9/14 datasets, 0.4% on 1/14 datasets, and G-DaRE RF is significantly better than SKLearn RF on the Surgical, Flight Delays, Olympics, and Credit Card datasets.

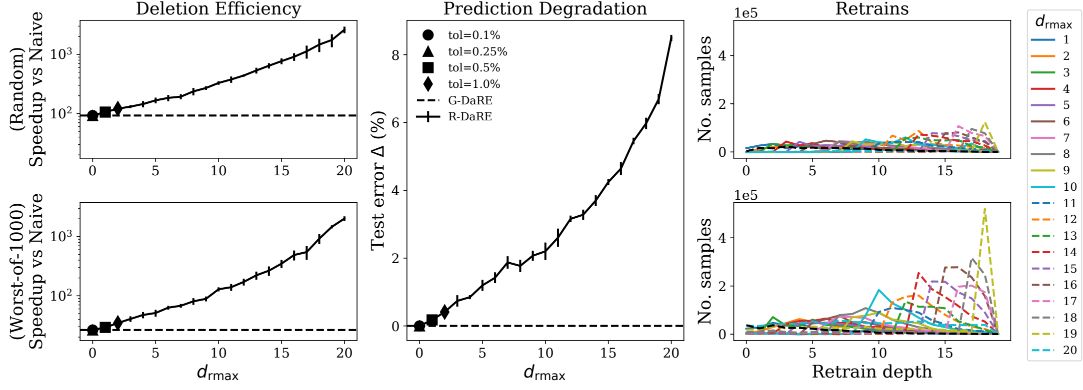
Table C.3. Hyperparameters selected for the G-DaRE and R-DaRE (using error tolerances of 0.1%, 0.25%, 0.5%, and 1.0%) models.

	G-DaRE & R-DaRE			**R-DaRE Only**			
Dataset	T	d_{max}	k	d_{rmax} **(0.1%)**	d_{rmax} **(0.25%)**	d_{rmax} **(0.5%)**	d_{rmax} **(1.0%)**
Surgical	100	20	25	0	1	2	4
Vaccine	50	20	5	5	7	11	14
Adult	50	20	5	10	13	14	16
Bank Marketing	100	20	25	6	9	12	14
Flight Delays	250	20	25	1	3	5	10
Diabetes	250	20	5	7	10	12	15
No Show	250	20	10	1	3	6	10
Olympics	250	20	5	0	1	2	3
Census	100	20	25	6	9	12	16
Credit Card	250	20	5	5	8	14	17
CTR	100	10	50	2	3	4	6
Twitter	100	20	5	2	4	7	11
Synthetic	50	20	10	0	2	3	5
Higgs	50	20	10	1	3	6	9

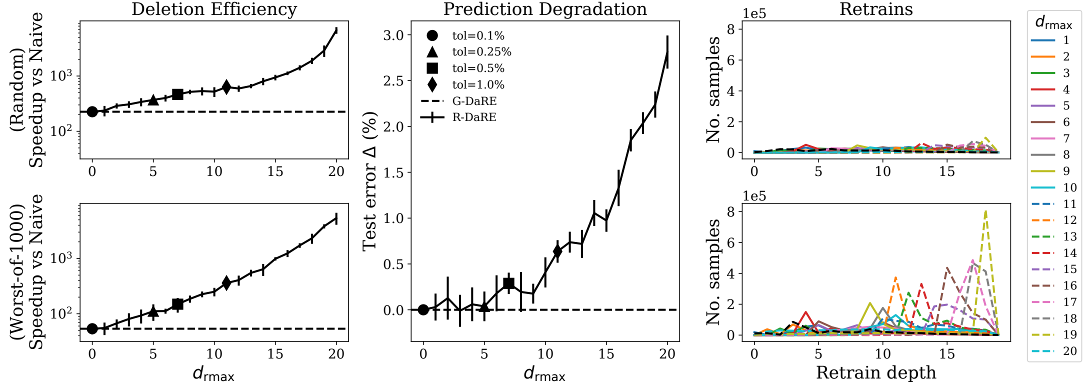
Table C.4. Training times (in seconds) for the G-DaRE model using the hyperparameters selected in Table C.3. Mean and standard deviations (S.D.) are computed over five runs.

Dataset	**Mean**	**S.D.**
Surgical	5.68	2.97
Vaccine	17.08	11.86
Adult	6.76	1.17
Bank Marketing	8.79	3.37
Flight Delays	262.00	50.39
Diabetes	141.91	39.12
No Show	77.65	20.33
Olympics	596.27	157.70
Census	127.40	9.57
Credit Card	616.65	166.00
Twitter	152.34	12.32
Synthetic	732.05	231.70
CTR	121.64	37.13
Higgs	5,016.44	146.34

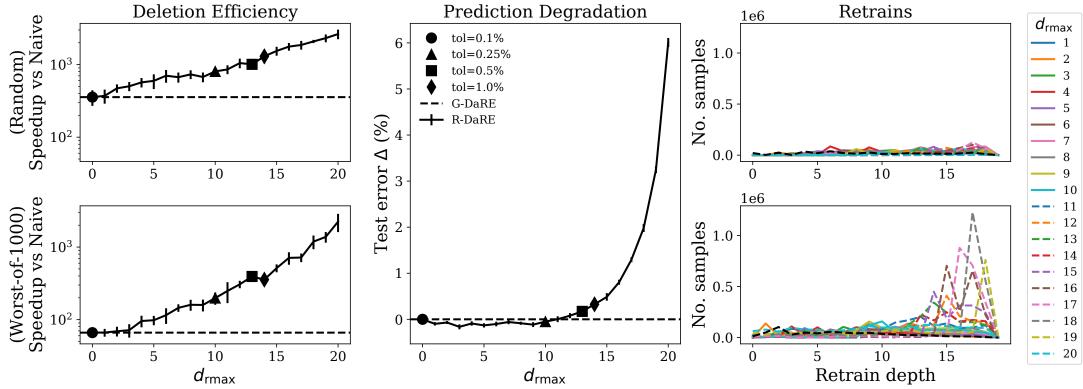
C.2.3 Effect of d_{rmax} on Deletion Efficiency. Figure C.1 shows the effect d_{rmax} has on deletion efficiency for the Surgical, Vaccine, and Adult datasets.



(a) Surgical



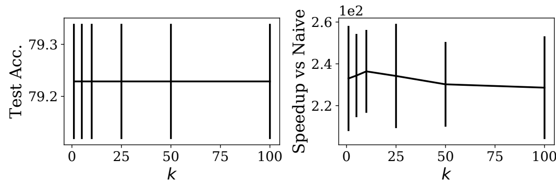
(b) Vaccine



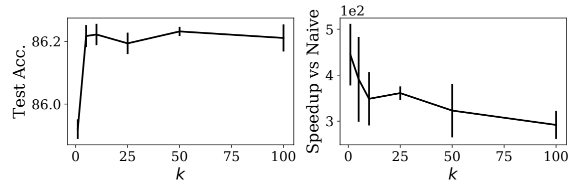
(c) Adult

Figure C.1. Effect of d_{rmax} on deletion efficiency.

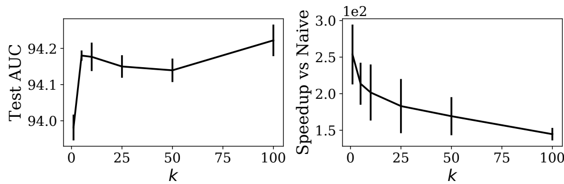
C.2.4 Effect of k on Deletion Efficiency. Figure C.2 shows the effect of k on deletion efficiency for different datasets. For k, we tested values [1, 5, 10, 25, 50, 100].



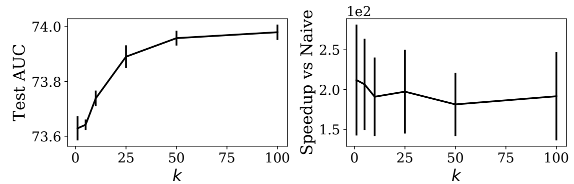
(a) Vaccine: All attributes are binary, thus k has no effect.



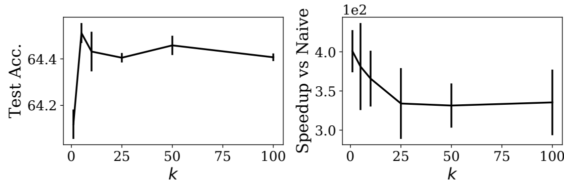
(b) Adult



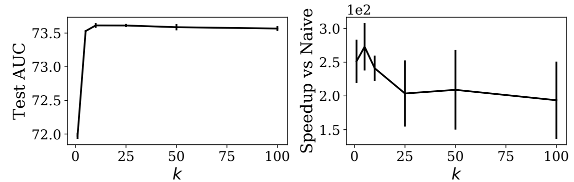
(c) Bank Marketing



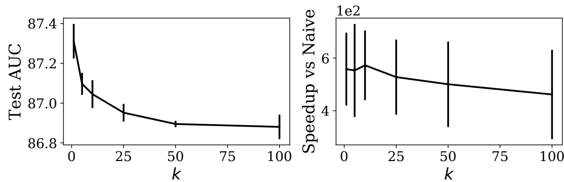
(d) Flight Delays



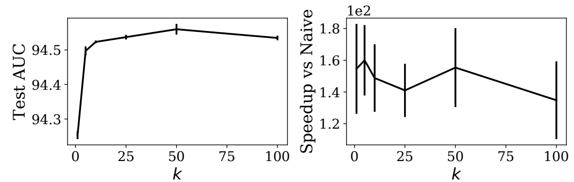
(e) Diabetes



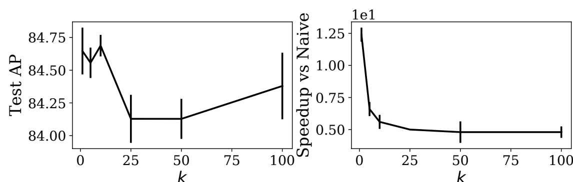
(f) No Show



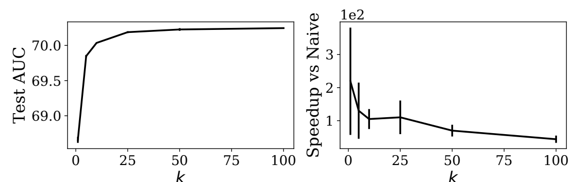
(g) Olympics: In this case, the randomness induced by a low k value actually helps predictive performance.



(h) Census



(i) Credit Card



(j) CTR

Figure C.2. Effect of k on deletion efficiency.

REFERENCES CITED

[1] M. Abadi, A. Chu, I. Goodfellow, et al. Deep learning with differential privacy. In *Proceedings of the Twenty-Third International ACM Conference on Computer and Communications Security*, 2016.

[2] Y. S. Abu-Mostafa and A. F. Atiya. Introduction to financial forecasting. *Applied Intelligence*, 6(3):205–213, 1996.

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

[4] C. Agarwal, D. D'souza, and S. Hooker. Estimating example difficulty using variance of gradients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10368–10378, 2022.

[5] N. Aldaghri, H. Mahdavifar, and A. Beirami. Coded machine unlearning. *IEEE Access*, 2021.

[6] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. S. Rangapuram, D. Salinas, J. Schulz, et al. GluonTS: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, 21(116):1–6, 2020.

[7] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1):5–43, 2003.

[8] A. N. Angelopoulos and S. Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

[9] S. Ö. Arik and T. Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, volume 35, pages 6679–6687, 2021.

[10] A. Avati, K. Jung, S. Harman, L. Downing, A. Ng, and N. H. Shah. Improving palliative care with deep learning. *BMC Medical Informatics and Decision Making*, 18(4):55–64, 2018.

[11] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 2014.

[12] E. Barshan, M.-E. Brunet, and G. K. 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.

[13] S. Basu, P. Pope, and S. Feizi. Influence functions in deep learning are fragile. In *Proceedings of the Eighth International Conference on Learning Representations*, 2020.

[14] T. Baumhauer, P. Schöttle, and M. Zeppelzauer. Machine unlearning: Linear filtration for logit-based classifiers. *arXiv preprint arXiv:2002.02730*, 2020.

[15] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13 (2):31–39, 2011.

[16] O. Benjelloun, S. Chen, and N. Noy. Google dataset search by the numbers. In *International Semantic Web Conference*, pages 667–682. Springer, 2020.

[17] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[18] G. Biau, E. Scornet, and J. Welbl. Neural random forests. *Sankhya A*, 2019.

[19] J. Bien and R. Tibshirani. Prototype selection for interpretable classification. *The Annals of Applied Statistics*, pages 2403–2424, 2011.

[20] A. Bloniarz, A. Talwalkar, B. Yu, and C. Wu. Supervised neighborhoods for distributed nonparametric regression. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 1450–1459, 2016.

[21] C. S. Bojer and J. P. Meldgaard. Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, 37(2): 587–603, 2021.

[22] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. Probabilistic demand forecasting at scale. In *Proceedings of the VLDB Endowment*, volume 10, pages 1694–1705. VLDB Endowment, 2017.

[23] L. Bourtoule, V. Chandrasekaran, C. Choquette-Choo, et al. Machine unlearning. In *Proceedings of the Forty-Second IEEE Symposium on Security and Privacy*, 2021.

[24] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE, 2021.

[25] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[26] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. CRC Press, 1984.

[27] J. Brophy and D. Lowd. Trex: Tree-ensemble representer-point explanations. In *ICML Workshop on Extending Explainable AI*, 2020.

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

[29] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

[30] P. Bull, I. Slavitt, and G. 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.

[31] R. M. 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.

[32] California. California consumer privacy act. `https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375`, 2018. [Online; accessed 16-April-2020].

[33] Canada. Pipeda. `https://www.priv.gc.ca/en/opc-news/news-and-announcements/2018/an_181010/`, 2018. [Online; accessed 19-August-2020].

[34] Y. Cao and J. Yang. Towards making systems forget with machine unlearning. In *Proceedings of the Thirty-Sixth IEEE Symposium on Security and Privacy*, 2015.

[35] N. Carlini, C. Liu, et al. The secret sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv preprint arXiv:1802.08232*, 2018.

[36] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Proceedings of the Fourteenth International Conference on Neural Information Processing Systems*, 2001.

[37] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. *JMLR*, 2011.

[38] M. Chen, Z. Zhang, et al. When machine unlearning jeopardizes privacy. In *Proceedings of the Twenty-Eighth International ACM Conference on Computer and Communications Security*, 2021.

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

[40] Y. Chen, J. Xiong, W. Xu, and J. Zuo. A novel online incremental and decremental learning algorithm based on variable support vector machine. *Cluster Computing*, 2019.

[41] Y. Chen, B. Li, H. Yu, P. Wu, and C. 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.

[42] H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010.

[43] A. Chrysakis and M.-F. Moens. Online continual learning from imbalanced data. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, 2020.

[44] M. Chui, J. Manyika, M. Miremadi, N. Henke, R. Chung, P. Nel, and S. Malhotra. Notes from the AI frontier: Insights from hundreds of use cases. *McKinsey Global Institute*, page 28, 2018.

[45] Y. Chung, I. Char, H. Guo, J. Schneider, and W. Neiswanger. Uncertainty toolbox: An open-source library for assessing, visualizing, and improving uncertainty quantification. *arXiv preprint arXiv:2109.10254*, 2021.

[46] J. W. Cohen, S. B. Cohen, and J. S. Banthin. The medical expenditure panel survey: A national information resource to support healthcare cost research and inform policy and practice. *Medical Care*, pages S44–S50, 2009.

[47] S. Consul and S. Williamson. Differentially private median forests for regression and classification. *arXiv preprint arXiv:2006.08795*, 2020.

[48] R. D. Cook and S. Weisberg. *Residuals and Influence in Regression.* Chapman and Hall, New York, 1982.

[49] A. Coraddu, L. Oneto, A. Ghio, S. Savio, D. Anguita, and M. 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.

[50] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[51] Criteo. Criteo click-through rate prediction. `https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/`, 2015. [Online; accessed 25-Januaray-2021].

[52] A. Davies and Z. Ghahramani. The random forest kernel and other kernels for big data from random partitions. *arXiv preprint arXiv:1402.4293*, 2014.

[53] D. De Cock. Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project. *Journal of Statistics Education*, 19(3), 2011.

[54] M. Denil, D. Matheson, and N. De Freitas. Narrowing the gap: Random forests in theory and in practice. In *Proceedings of the Thirty-First International Conference on Machine Learning*, 2014.

[55] Y. Dong, S. Hopkins, and J. Li. Quantum entropy scoring for fast robust mean estimation and improved outlier detection. In *Proceedings of the Thirty-Third International Conference on Neural Information Processing Systems*, 2019.

[56] 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].

[57] D. D'souza, Z. Nussbaum, C. Agarwal, and S. Hooker. A tale of two long tails. In *ICML Workshop on Uncertainty and Robustness in Deep Learning*, 2021.

[58] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song. Lifelong anomaly detection through unlearning. In *Proceedings of the Twenty-Sixth International ACM Conference on Computer and Communications Security*, 2019.

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

[60] H. Duan, H. Li, G. He, and Q. Zeng. Decremental learning algorithms for nonlinear langrangian and least squares support vector machines. In *Proceedings of the First International Symposium on Optimization and Systems Biology*, 2007.

[61] T. Duan, A. Anand, D. Y. Ding, K. K. Thai, S. Basu, A. Ng, and A. Schuler. Ngboost: Natural gradient boosting for probabilistic prediction. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2690–2700. PMLR, 2020.

[62] C. Dwork. Differential privacy. In *Proceedings of the Thirty-Third International Colloquium on Automata, Languages, and Programming*, 2006.

[63] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 2014.

[64] EU. Regulation (eu) 2016/679. `https://eur-lex.europa.eu/eli/reg/2016/679/oj`, 2016. [Online; accessed 16-April-2020].

[65] H. Fanaee-T and J. Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, pages 1–15, 2013.

[66] V. Feldman and C. 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.

[67] K. Fernandes, P. Vinagre, and P. Cortez. A proactive intelligent decision support system for predicting the popularity of online news. In *Proceedings of the 17th Portuguese Conference on Artificial Intelligence*, pages 535–546. Springer, 2015.

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

[69] S. Fletcher and M. Z. Islam. A differentially private decision forest. In *Proceedings of the Thirteenth Australasian Data Mining Conference*, 2015.

[70] S. Fletcher and M. Z. Islam. Differentially private random decision forests using smooth sensitivity. *Expert Systems with Applications*, 2017.

[71] S. Fletcher and M. Z. Islam. Decision tree classification with differential privacy: A survey. *ACM Computing Surveys*, 2019.

[72] A. 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].

[73] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

[74] J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, pages 1–67, 1991.

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

[76] S. Fu, F. He, Y. Xu, and D. Tao. Bayesian inference forgetting. *arXiv preprint arXiv:2101.06417*, 2021.

[77] S. Garg, S. Goldwasser, and P. N. Vasudevan. Formalizing data deletion in the context of the right to be forgotten. In *Proceedings of the Thirty-Ninth International Conference on the Theory and Applications of Cryptographic Techniques*, 2020.

[78] R. Genuer, J.-M. Poggi, C. Tuleau-Malot, and N. Villa-Vialaneix. Random forests for big data. *Big Data Research*, 9:28–46, 2017.

[79] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 2006.

[80] A. Ghorbani and J. 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.

[81] A. Ghorbani, J. Wexler, J. Y. Zou, and B. 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.

[82] A. Ginart, M. Guan, G. Valiant, and J. Y. Zou. Making AI forget you: Data deletion in machine learning. In *Proceedings of the Thirty-Third International Conference on Neural Information Processing Systems*, 2019.

[83] T. Gneiting and M. Katzfuss. Probabilistic forecasting. *Annual Review of Statistics and Its Application*, 1:125–151, 2014.

[84] T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477): 359–378, 2007.

[85] A. Golatkar, A. Achille, and S. Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *Proceedings of the Sixteenth European Conference on Computer Vision*, 2020.

[86] A. Golatkar, A. Achille, and S. Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision and Pattern Recognition*, 2020.

[87] A. Golatkar, A. Achille, A. Ravichandran, M. Polito, and S. Soatto. Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision and Pattern Recognition*, 2021.

[88] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko. Revisiting deep learning models for tabular data. In *Proceedings of the Thirty-Fifth International Conference on Neural Information Processing Systems*, volume 34, pages 18932–18943, 2021.

[89] A. Graves. Practical variational inference for neural networks. In *Proceedings of the Twenty-Fifth International Conference on Neural Information Processing Systems*, volume 24, 2011.

[90] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *Machine Learning and Computer Security Workshop*, 2017.

[91] C. Guo, T. Goldstein, A. Hannun, and L. van der Maaten. Certified data removal from machine learning models. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, 2020.

[92] A. Gupta, C. Lanteigne, and S. Kingsley. SECure: A social and environmental certificate for AI systems. In *ICML Workshop on Deploying and Monitoring Machine Learning Systems*, 2020.

[93] K. S. Gurumoorthy, A. Dhurandhar, G. Cecchi, and C. 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.

[94] K. Hamidieh. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, 154: 346–354, 2018.

[95] Z. Hammoudeh and D. Lowd. Reducing certified regression to certified classification, 2022.

[96] Z. Hammoudeh and D. Lowd. Identifying a training-set attack's target using renormalized influence estimation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS'22, 2022.

[97] K. Hanawa, S. Yokoi, S. Hara, and K. Inui. Evaluation of similarity-based explanations. In *Proceedings of the Ninth International Conference on Learning Representations*, 2021.

[98] J. Hanley and B. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 1982.

[99] S. Hara, A. Nitanda, and T. 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.

[100] H. Hasson, B. Wang, T. Januschowski, and J. Gasthaus. Probabilistic forecasting: A level-set approach. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, volume 34, 2021.

[101] J. Haupt and R. Nowak. Signal reconstruction from noisy random projections. *IEEE Transactions on Information Theory*, 2006.

[102] H. Hazimeh, N. Ponomareva, P. Mol, Z. Tan, and R. Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, pages 4138–4148. PMLR, 2020.

[103] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. 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.

[104] S. Hooker, D. Erhan, P.-J. Kindermans, and B. 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.

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

[106] X. Huang, A. Khetan, M. Cvitkovic, and Z. Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

[107] E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021.

[108] Z. Izzo, M. A. Smart, K. Chaudhuri, and J. Zou. Approximate data deletion from machine learning models: Algorithms and evaluations. *arXiv preprint arXiv:2002.10077*, 2020.

[109] A. Jacovi, A. Marasović, T. Miller, and Y. Goldberg. Formalizing trust in artificial intelligence: Prerequisites, causes and goals of human trust in AI. In *Proceedings of the Fourth International ACM Conference on Fairness, Accountability, and Transparency*, pages 624–635, 2021.

[110] T. Januschowski, Y. Wang, K. Torkkola, T. Erkkilä, H. Hasson, and J. Gasthaus. Forecasting with trees. *International Journal of Forecasting*, 2021.

[111] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. Hynes, N. M. Gürel, B. Li, C. Zhang, D. Song, and C. J. Spanos. Towards efficient data valuation based on the Shapley value. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, pages 1167–1176. PMLR, 2019.

[112] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka. Regularization is all you need: Simple neural nets can excel on tabular data. In *Proceedings of the Thirty-Fifth International Conference on Neural Information Processing Systems*, 2021.

[113] Kaggle. Credit card fraud detection. `https://www.kaggle.com/mlg-ulb/creditcardfraud/`, 2018. [Online; accessed 27-July-2020].

[114] Kaggle. 120 years of olympic history: Athletes and events. `https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results`, 2018. [Online; accessed 28-July-2020].

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

[116] M. Karasuyama and I. Takeuchi. Multiple incremental decremental learning of support vector machines. In *Proceedings of the Twenty-Second International Conference on Neural Information Processing Systems*, 2009.

[117] H. Kaya, P. Tüfekci, and F. 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.

[118] G. Ke, Q. 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.

[119] M. T. Keane and B. 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.

[120] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM (JACM)*, 1998.

[121] B. Kim, R. Khanna, and O. 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.

[122] J. Knoblauch, H. Husain, and T. Diethe. Optimal continual learning has perfect memory and is NP-hard. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, 2020.

[123] D. Kocev, C. Vens, et al. Tree ensembles for predicting structured outputs. *Pattern Recognition*, 2013.

[124] R. Koenker and K. F. Hallock. Quantile regression. *Journal of Economic Perspectives*, 15(4):143–156, 2001.

[125] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

[126] P. W. Koh, T. Nguyen, Y. S. Tang, S. Mussmann, E. Pierson, B. Kim, and P. Liang. Concept bottleneck models. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, pages 5338–5348. PMLR, 2020.

[127] M. Koklu and I. A. Ozkan. Multiclass classification of dry beans using computer vision and machine learning techniques. *Computers and Electronics in Agriculture*, 174:105507, 2020.

[128] Z. Kong and K. Chaudhuri. Understanding instance-based interpretability of variational auto-encoders. In *Proceedings of the Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[129] C. Kwak, J. Lee, et al. Let machines unlearn–machine unlearning and the right to be forgotten. *Proceedings of the Information Systems Security and Privacy*, 2017.

[130] J. Larsen et al. 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].

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

[132] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, volume 32, pages 5243–5253, 2019.

[133] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 2021.

[134] Y. Lin and Y. Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

[135] A. R. Linero and Y. Yang. Bayesian regression tree ensembles that adapt to smoothness and sparsity. *Journal of the Royal Statistical Society*, 2018.

[136] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 1982.

[137] S. M. Lundberg and S.-I. 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.

[138] S. M. Lundberg, G. G. Erion, and S.-I. Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

[139] Z. T. Luo, H. Sang, and B. Mallick. BAST: Bayesian additive regression spanning trees for complex constrained domain. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, volume 34, 2021.

[140] R. J. Lyon, B. Stappers, S. Cooper, J. M. Brooke, and J. 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.

[141] S. Makridakis, E. Spiliotis, V. Assimakopoulos, Z. Chen, A. Gaba, I. Tsetlin, and R. L. Winkler. The M5 uncertainty competition: Results, findings and conclusions. *International Journal of Forecasting*, 2021.

[142] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 2022.

[143] A. Malinin, L. Prokhorenkova, and A. Ustimenko. Uncertainty in gradient boosting via ensembles. In *Proceedings of the Ninth International Conference on Learning Representations*, 2021.

[144] N. Meinshausen and G. Ridgeway. Quantile regression forests. *Journal of Machine Learning Research*, 7(6), 2006.

[145] L. Mentch and G. Hooker. Quantifying uncertainty in random forests via confidence intervals and hypothesis tests. *Journal of Machine Learning Research*, 2016.

[146] F. Moosmann, B. Triggs, and F. 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.

[147] S. Moro, P. Cortez, et al. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 2014.

[148] M. Mozaffari-Kermani, S. Sur-Kolay, et al. Systematic poisoning attacks on and defenses for machine learning in healthcare. *Journal of Biomedical and Health Informatics*, 2014.

[149] I. J. Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1):90–100, 2003.

[150] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, 1996.

[151] T. T. Nguyen, T. T. Huynh, P. L. Nguyen, A. W.-C. Liew, H. Yin, and Q. V. H. Nguyen. A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*, 2022.

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

[153] R. K. Pace and R. Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

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

[155] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[156] G. Pleiss, T. Zhang, E. Elenberg, and K. 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.

[157] G. Plumb, D. Molitor, and A. S. Talwalkar. Model agnostic supervised local explanations. In *Proceedings of the Thirty-Second International Conference on Neural Information Processing Systems*, pages 2515–2524, 2018.

[158] S. Popov, S. Morozov, and A. Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *Proceedings of the Seventh International Conference on Learning Representations*, 2019.

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

[160] G. Pruthi, F. Liu, S. Kale, and M. 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.

[161] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Elsevier, 2014.

[162] M. Rahmani and P. Li. Outlier detection and robust PCA using a convex measure of innovation. In *Proceedings of the Thirty-Third International Conference on Neural Information Processing Systems*, 2019.

[163] K. 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].

[164] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[165] S. Rana, S. K. Gupta, and S. Venkatesh. Differentially private random forest with high utility. In *Proceedings of the Fifteenth International Conference on Data Mining*, 2015.

[166] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski. Deep state space models for time series forecasting. In *Proceedings of the Thirty-Second International Conference on Neural Information Processing Systems*, volume 31, pages 7785–7794, 2018.

[167] M. Redmond and A. Baveja. A data-driven software tool for enabling cooperative information sharing among police departments. *European Journal of Operational Research*, 141(3):660–678, 2002.

[168] Research and I. T. 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].

[169] M. T. Ribeiro, S. Singh, and C. 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.

[170] R. A. Rigby and D. M. Stasinopoulos. Generalized additive models for location, scale and shape. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 54(3):507–554, 2005.

[171] Y. Romano, E. Patterson, and E. Candes. Conformalized quantile regression. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, volume 32, pages 3543–3553, 2019.

[172] E. Romero, I. Barrio, and L. Belanche. Incremental and decremental learning for linear support vector machines. In *Proceedings of the Seventeenth International Conference on Artificial Neural Networks*, 2007.

[173] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[174] S. Schelter. "amnesia" - machine learning models that can forget user data very fast. In *Proceedings of the International Conference on Innovative Data Systems Research*, 2020.

[175] S. Schelter, S. Grafberger, and T. Dunning. Hedgecut: Maintaining randomised trees for low-latency machine unlearning. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, 2021.

[176] L. Schlosser, T. Hothorn, R. Stauffer, and A. Zeileis. Distributional regression forests for probabilistic precipitation forecasting in complex terrain. *The Annals of Applied Statistics*, 13(3):1564–1589, 2019.

[177] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *Proceedings of the Fourteenth International Conference on Computational Learning Theory*, pages 416–426. Springer, 2001.

[178] S. Sedhai and A. 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.

[179] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. 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.

[180] R. Sen, H.-F. Yu, and I. Dhillon. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 4837–4846, 2019.

[181] G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.

[182] J. Shao. Linear model selection by cross-validation. *Journal of the American Statistical Association*, 1993.

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

[184] B. Sharchilev, Y. Ustinovskiy, P. Serdyukov, and M. de Rijke. Finding influential training samples for gradient boosted decision trees. In *Proceedings of the Thirty-Fifth International Conference on Machine Learning*, 2018.

[185] B. Sharchilev, Y. Ustinovskiy, P. Serdyukov, and M. 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.

[186] S. J. Sheather and M. C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(3):683–690, 1991.

[187] S. Shintre and J. Dhaliwal. Verifying that the influence of a user data point has been removed from a machine learning classifier. `https://patents.google.com/patent/US10225277B1/en`, 2019.

[188] S. Shintre, K. A. Roundy, and J. Dhaliwal. Making machine learning forget. In *Proceedings of the Seventh International Annual Privacy Forum*, 2019.

[189] M. Shoeybi, M. Patwary, et al. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[190] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *Proceedings of the Thirty-Eighth IEEE Symposium on Security and Privacy*, 2017.

[191] R. Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[192] K. Singh, R. K. Sandhu, and D. Kumar. Comment volume prediction using neural networks and decision trees. In *IEEE UKSim-AMSS 17th International Conference on Computational Modeling and Simulation, UKSim2015*, March 2015.

[193] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju. Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods. In *Proceedings of the Third AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020.

[194] D. M. Sommer, L. Song, S. Wagh, and P. Mittal. Towards probabilistic verification of machine unlearning. *arXiv preprint arXiv:2003.04247*, 2020.

[195] O. Sprangers, S. Schelter, and M. de Rijke. Probabilistic gradient boosting machines for large-scale probabilistic regression. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021.

[196] J. Steinhardt, P. W. Koh, and P. Liang. Certified defenses for data poisoning attacks. In *Proceedings of the Thirty-First International Conference on Neural Information Processing Systems*, pages 3520–3532, 2017.

[197] J. H. Stock, M. W. Watson, et al. *Introduction to Econometrics*, volume 3. Pearson New York, 2012.

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

[199] B. Sun, L. Yang, W. Zhang, M. Lin, P. Dong, C. Young, and J. 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.

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

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

[202] S. Swayamdipta, R. Schwartz, N. Lourie, Y. Wang, H. Hajishirzi, N. A. Smith, and Y. 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.

[203] S. B. Taieb, R. Huser, R. Hyndman, M. Genton, et al. Probabilistic time series forecasting with boosted additive models: An application to smart meter data. *Department of Economics and Business Statistics, Monash University*, 2015.

[204] S. Tan, M. Soloviev, G. Hooker, and M. 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.

[205] S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[206] N. Terashita, H. Ohashi, Y. Nonaka, and T. Kanemaru. Influence estimation for generative adversarial networks. In *Proceedings of the Ninth International Conference on Learning Representations*, 2021.

[207] E. Tjoa and C. Guan. A survey on explainable artificial intelligence (XAI): Toward medical XAI. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[208] S. Tople, M. Brockschmidt, et al. Analyzing privacy loss in updates of natural language models. *arXiv preprint arXiv:1912.07942*, 2019.

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

[210] P. 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.

[211] A. Tveit, M. L. Hetland, and H. Engum. Incremental and decremental proximal support vector classification using decay coefficients. In *Proceedings of the Fifth International Conference on Data Warehousing and Knowledge Discovery*, 2003.

[212] A. Ustimenko and L. Prokhorenkova. SGLB: Stochastic gradient langevin boosting. In *International Conference on Machine Learning*, pages 10487–10496. PMLR, 2021.

[213] J. van Rijn. Kin8nm. `https://www.openml.org/d/189`, 2014. [Online; accessed 20-January-2022].

[214] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the Thirty-First International Conference on Neural Information Processing Systems*, volume 30, 2017.

[215] M. Veale, R. Binns, and L. Edwards. Algorithms that remember: Model inversion attacks and data protection law. *Philosophical Transactions of the Royal Society A*, 2018.

[216] E. F. Villaronga, P. Kieseberg, and T. Li. Humans forget, machines remember: Artificial intelligence and the right to be forgotten. *Computer Law & Security Review*, 2018.

[217] S. Wager and S. Athey. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 2018.

[218] B. Wang, Y. Yao, et al. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proceedings of the Fortieth IEEE Symposium on Security and Privacy*, 2019.

[219] Y. Wang, A. Smola, D. Maddix, J. Gasthaus, D. Foster, and T. Januschowski. Deep factors for forecasting. In *Proceedings of the Thirty-Sixth International Conference on Machine Learning*, pages 6607–6617. PMLR, 2019.

[220] Y. Wu, E. Dobriban, and S. B. Davidson. DeltaGrad: Rapid retraining of machine learning models. In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, 2020.

[221] C.-K. Yeh, J. S. Kim, I. E. Yen, and P. 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.

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

[223] I.-C. Yeh and C.-h. 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.

[224] S. Yeom, I. Giacomelli, et al. Privacy risk in machine learning: Analyzing the connection to overfitting. In *Proceedings of the Thirty-First IEEE Computer Security Foundations Symposium*, 2018.

[225] F. Zaman and H. Hirose. Effect of subsampling rate on subbagging and related ensembles of stable classifiers. In *Proceedings of the Third International Conference on Pattern Recognition and Machine Intelligence*, 2009.

[226] M. Zamo and P. Naveau. Estimation of the continuous ranked probability score with limited information and applications to ensemble weather forecasts. *Mathematical Geosciences*, 50(2):209–234, 2018.

[227] J. H. Zar. Spearman rank correlation. *Encyclopedia of Biostatistics*, 7, 2005.

[228] W. Zhang, Z. Huang, Y. Zhu, G. Ye, X. Cui, and F. 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.

[229] M. Zhu. Recall, precision and precision, average. *University of Waterloo, Waterloo*, 2004.