Boosting with Label Noise

Derek Huang, Riya Mokashi, Iva Porfirova

BAC Advanced Team

May 5, 2020

Table of contents

Motivation

- Past experiences with data containing label noise and with AdaBoost giving performance short of what was desired
- Theoretical interest in recent advances made in gradient boosting
- Practical interest in robustness of boosting models on classification problems when training and validation data have corrupted labels
- How well does XGBoost perform on mislabeled data in comparison to `sklearn` AdaBoost and gradient tree boosting?

Gradient boosting

- For a given input space $\mathcal{X} \subseteq \mathbb{R}^n$ and output space $\mathcal{Y} \subseteq \mathbb{R}$, *boosting* iteratively fits a *generalized additive model* $F : \mathcal{X} \to \mathbb{R}$ where

$$F(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x}) \tag{1}$$

  Here $\forall m \in \{1, \dots M\}$, $\beta_m \in \mathbb{R}$, $\phi_m \in \mathcal{H}$, where $\mathcal{H}$ can be conveniently thought of as a subset of $L^2(\mathcal{X})$.

- That is, at iteration m, given an incomplete model F_{m-1}, where F_{m-1} has only the first $m-1$ terms in F, choose β_m, ϕ_m such that

$$\beta_m, \phi_m = \arg\min_{\beta,\phi} \frac{1}{|\mathcal{D}|} \sum_{k=1}^{|\mathcal{D}|} L(y_k, F_{m-1}(\mathbf{x}_k) + \beta\phi(\mathbf{x}_k)) \tag{2}$$

Gradient boosting

- Here $L : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}_+$ is a **convex**, **twice-differentiable** loss function. This iterative process is *forward stagewise additive modeling*.
- Impossible to solve (2) directly, so we do greedy approximation using *functional gradient descent*, choosing β_m, ϕ_m such that

$$\phi_m = \arg\min_{\phi} \frac{1}{|\mathcal{D}|} \sum_{k=1}^{|\mathcal{D}|} \widetilde{L}(-\partial_{F_{m-1}(\mathbf{x}_k)} L_k, \phi(\mathbf{x}_k)) \tag{3}$$

$$\beta_m = \arg\min_{\beta} \frac{1}{|\mathcal{D}|} \sum_{k=1}^{|\mathcal{D}|} L(y_k, F_{m-1}(\mathbf{x}_k) + \beta\phi_m(\mathbf{x}_k)) \tag{4}$$

Here $\widetilde{L} : \mathbb{R}^2 \to \mathbb{R}_+$ is another convex loss function, and

$$\partial_{F_{m-1}(\mathbf{x}_k)} L_k \triangleq \frac{\partial L}{\partial F_{m-1}(\mathbf{x}_k)}(y_k, F_{m-1}(\mathbf{x}_k))$$

The abstract picture

- Define the *expected loss* functional $J : \mathcal{H} \to \mathbb{R}_+$, where

$$J[f] \triangleq \mathbb{E}[L(Y, f(X))] = \int_{\mathcal{X}\times\mathcal{Y}} L(y, f(\mathbf{x}))\psi_{X,Y}(\mathbf{x}, y)\, dy\, d\mathbf{x}$$

  Here $\psi_{X,Y}$ is the joint density function of X, Y.
- We want $F = \arg\min_{f\in\mathcal{H}} J[f]$ when the shape of J is unknown. *Functional gradient descent* updates guesses for F, with the $(m-1)$th guess F_{m-1}, by using the *functional gradient* $\delta J[F_{m-1}]/\delta f$ to find a step $\zeta_m = \arg\min_{\zeta\in\mathcal{H}} J[F_{m-1} + \zeta]$ such that $J[F_m] \leq J[F_{m-1}]$.
- One can show that[1] for every iteration m,

$$\zeta_m(\mathbf{x}) = \gamma_m \psi_X(\mathbf{x}) \mathbb{E}\left[-\frac{\partial L}{\partial F_{m-1}(\mathbf{x})}(Y, F_{m-1}(\mathbf{x}))\right] \tag{5}$$

  Here ψ_X is the marginal density of X and γ_m is the step size.

[1] Derivation starts on slide 31 in the appendix.

A concrete picture

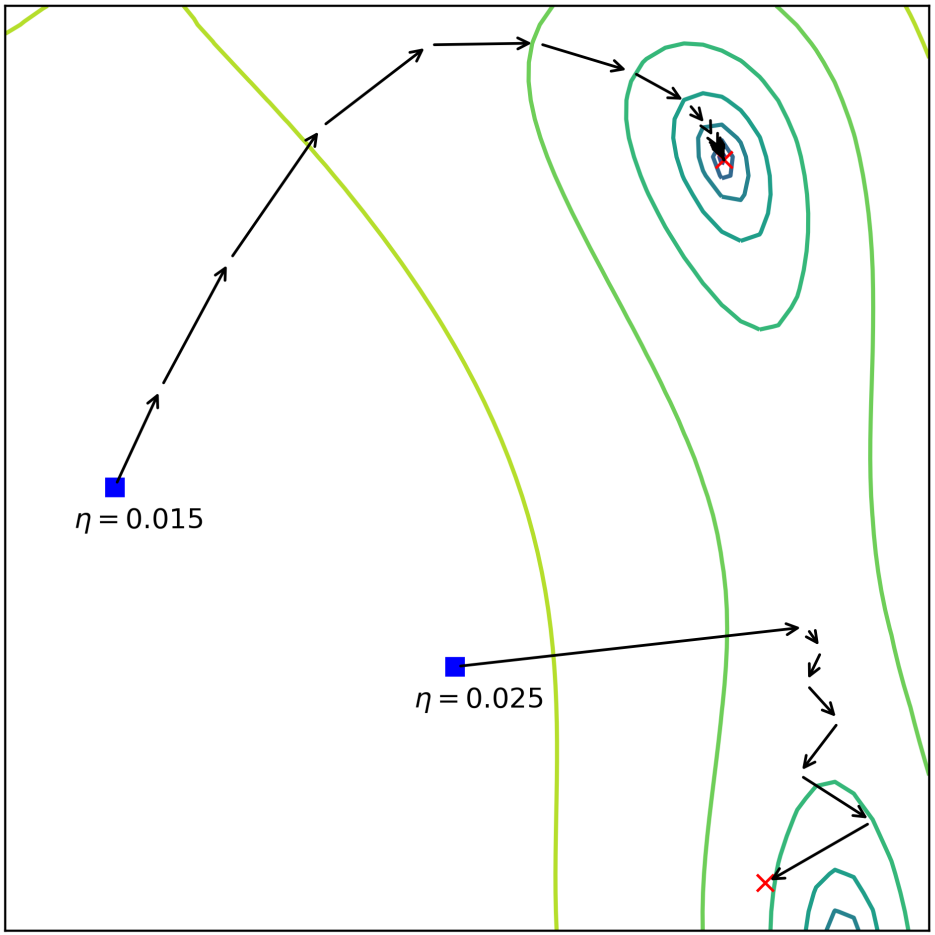


Figure 1: Parameter gradient descent for the Himmelblau function.

Approximating ζ_m

- Recall from (5) that the optimal step $\zeta_m \in \mathcal{H}$ is such that

$$\zeta_m(\mathbf{x}) = \gamma_m \psi_X(\mathbf{x}) \mathbb{E}\left[-\frac{\partial L}{\partial F_{m-1}(\mathbf{x})}(Y, F_{m-1}(\mathbf{x}))\right] = \gamma_m \psi_X(\mathbf{x}) \mathcal{E}_m^-(\mathbf{x})$$

- We can directly approximate $\mathcal{E}_m^-$ using (3), i.e. fitting ϕ_m where

$$\phi_m = \arg\min_{\phi} \frac{1}{|\mathcal{D}|} \sum_{k=1}^{|\mathcal{D}|} \widetilde{L}(-\partial_{F_{m-1}(\mathbf{x}_k)} L_k, \phi(\mathbf{x}_k))$$

- In practice, we treat ψ_X as constant[2], say $\psi_X \triangleq 1/|\mathcal{D}|$, where we can define $\beta_m \triangleq \gamma_m/|\mathcal{D}|$. After determining ϕ_m, β_m satisfies (4), i.e.

$$\beta_m = \arg\min_{\beta} \frac{1}{|\mathcal{D}|} \sum_{k=1}^{|\mathcal{D}|} L(y_k, F_{m-1}(\mathbf{x}_k) + \beta \phi_m(\mathbf{x}_k))$$

[2] This is because empirical risk minimization minimizes pointwise.

Empirical observations

- Boosting is an effective way to fit a generalized additive model stagewise and is known to reduce both bias and variance.
- But like any model with many degrees of freedom, boosting runs the danger of overfitting to the training data $\mathcal{D}$.
  - Consider AdaBoost. One can show that[3] as $M \to \infty$, boosting achieves the minimum training error rate, i.e. very tight fit to $\mathcal{D}$. With mislabeled examples, generalization can be poor.
- A practical problem is slow fitting; stagewise fitting not parallelizable.
- For gradient tree boosting, limited regularization methods.
  - Fixed max tree depth, learning rate $\eta_m \in (0, 1)$ to shrink β_m coefficients

[3] Proof starts on slide 29 in appendix.

Why XGBoost?

- Superior C++ implementation. Cache-aware memory access, out-of-core computing, parallelized tree fitting. Fast, scalable.
- Split-finding is sparsity aware. For any feature/threshold split (u, v), learns default directions for inputs missing the uth feature value.
  - Determines best split for when inputs missing uth feature value are put in left branch, best split when inputs missing uth feature value are put in right branch. Learns default direction by picking the better split.
- Rich software package with many wrappers for popular languages. Supports many differentiable loss functions and ensemble models.
  - Original XGBoost algorithm, boosting with tree dropout, linear basis function boosting, and even a random forest implementation.
- Reduced overfitting of individual trees
  - L^2 weight regularization, tree complexity penalty. Leads to a novel way of constructing a tree through recursive binary splitting.

XGBoost objective

- Consider fixed tree structure ξ_m, with terminal regions $\mathcal{G}_m$, weights ω_m, where $\mathcal{G}_m = \left\{G_m^{(1)}, \dots G_m^{(T_m)}\right\}$ and $\omega_m = \left\{w_{G_m^{(1)}}, \dots w_{G_m^{(T_m)}}\right\}$.
- The simplified m-th iteration objective function is[4]

$$\widetilde{J}_m(\mathcal{G}_m, \omega_m, \mathcal{D}) = \sum_{k=1}^{|\mathcal{D}|} \partial_{F_{m-1}(\mathbf{x}_k)} L_k \phi_{\mathcal{G}_m}(\mathbf{x}_k) + \frac{1}{2}\lambda \sum_{j=1}^{T_m} w_{G_m^{(j)}}^2 + \frac{1}{2}\sum_{k=1}^{|\mathcal{D}|} \partial^2_{F_{m-1}(\mathbf{x}_k)} L_k \phi_{\mathcal{G}_m}(\mathbf{x}_k)^2 + \gamma T_m$$

- $\phi_{\mathcal{G}_m}$ is a regression tree with terminal regions $\mathcal{G}_m$, $\gamma \geq 0$ the tree complexity penalty, $\lambda \geq 0$ a constant scaling the L^2 weight penalty.

[4] Apply 2nd-order Taylor expansion to (3) with penalty terms and drop constant.

Computing $w_{G_m^{(j)}}$

- Note $\phi_{\mathcal{G}_m}(\mathbf{x}) = \sum_{j=1}^{T_m} w_{G_m^{(j)}} \mathbb{I}\{\mathbf{x} \in G_m^{(j)}\}$. Then, defining the index set $I_H \triangleq \{k \in \mathbb{N} : \mathbf{x}_k \in H, (\mathbf{x}_k, y_k) \in \mathcal{D}\}$, $H \subset \mathcal{X}$, $H \neq \emptyset$, we have

$$\widetilde{J}_m(\mathcal{G}_m, \omega_m, \mathcal{D}) = \sum_{j=1}^{T_m} w_{G_m^{(j)}} \sum_{k \in I_{G_m^{(j)}}} \partial_{F_{m-1}(\mathbf{x}_k)} L_k + \frac{1}{2} \sum_{j=1}^{T_m} w_{G_m^{(j)}}^2 \left(\lambda + \sum_{k \in I_{G_m^{(j)}}} \partial^2_{F_{m-1}(\mathbf{x}_k)} L_k \right) + \gamma T_m$$

- Here we grouped first- and second-order partials evaluated at $(\mathbf{x}_k, y_k)$, $\forall k \in \{1, \dots |\mathcal{D}|\}$, by the $G_m^{(j)}$ each $\mathbf{x}_k$ belongs to.
- We note that $\widetilde{J}$ is strongly convex in the weights, which is convenient.

Computing $w_{G_m^{(j)}}$

- Analytical solution for the optimal weight $\hat{w}_{G_m^{(j)}}$ for each $G_m^{(j)}$, where

$$\hat{w}_{G_m^{(j)}} = -\frac{\sum_{k \in I_{G_m^{(j)}}} \partial_{F_{m-1}(\mathbf{x}_k)} L_k}{\lambda + \sum_{k \in I_{G_m^{(j)}}} \partial^2_{F_{m-1}(\mathbf{x}_k)} L_k} \tag{6}$$

- Easily check by computing $\partial \widetilde{J}_m / \partial w_{G_m^{(j)}}$ for each $w_{G_m^{(j)}}$ and setting these partials to zero. Note λ introduces shrinkage into each $\hat{w}_{G_m^{(j)}}$.
- Contrast this to CART gradient boosting, which prescribes that

$$\hat{w}_{G_m^{(j)}} = \arg\min_w \sum_{k \in I_{G_m^{(j)}}} L(y_k, F_{m-1}(\mathbf{x}_k) + w)$$

- No regularization in CART to prevent overfitting weights.

XGBoost split criteria

- Regularization parameters γ and λ directly affect tree construction.
- Rewrite objective $\widetilde{J}_m$ with optimal weights $\hat{\omega}_m$ using (6) as

$$\widetilde{J}_m(\mathcal{G}_m, \hat{\omega}_m, \mathcal{D}) = -\frac{1}{2}\sum_{j=1}^{T_m} \frac{\left(\sum_{k \in I_{G_m^{(j)}}} \partial_{F_{m-1}(\mathbf{x}_k)} L_k\right)^2}{\lambda + \sum_{k \in I_{G_m^{(j)}}} \partial^2_{F_{m-1}(\mathbf{x}_k)} L_k} + \gamma T_m$$

- Conditional on $\mathcal{G}_m$, this is the minimum of $\widetilde{J}_m$. Note the value $V(G_m^{(j)}, \mathcal{D})$ assigned to one region $G_m^{(j)}$ is such that

$$V(G_m^{(j)}, \mathcal{D}) = -\frac{1}{2}\frac{\left(\sum_{k \in I_{G_m^{(j)}}} \partial_{F_{m-1}(\mathbf{x}_k)} L_k\right)^2}{\lambda + \sum_{k \in I_{G_m^{(j)}}} \partial^2_{F_{m-1}(\mathbf{x}_k)} L_k} + \gamma$$

XGBoost split criteria

- To optimally split nonempty $H \subset \mathcal{X}$ into disjoint H_L, H_R, minimize $V(H_L, \mathcal{D}) + V(H_R, \mathcal{D}) - V(H, \mathcal{D}) \Leftrightarrow$ maximize its negation.
- Interpret latter as a *split quality score*, where optimal split of H solves

$$\max_{H_L, H_R} \max\{0, Q(H, \mathcal{D}) - Q(H_L, \mathcal{D}) - Q(H_R, \mathcal{D}) - \gamma\}$$

  Here we defined the nonnegative impurity $Q(A, \mathcal{D})$ for $A \subset \mathcal{X}$ as

$$Q(A, \mathcal{D}) = \frac{1}{2} \frac{\left(\sum_{k \in I_A} \partial_{F_{m-1}(\mathbf{x}_k)} L_k\right)^2}{\lambda + \sum_{k \in I_A} \partial^2_{F_{m-1}(\mathbf{x}_k)} L_k}$$

- A split is made only if its score is positive, so γ is a minimum threshold for split qualification, so higher $\gamma \Rightarrow$ sparser trees.

CART split criteria

- No such parallel in standard CART regression tree split qualification. CART mean squared error impurity of a nonempty $H \subset \mathcal{X}$ is

$$Q(H, \mathcal{D}) \triangleq \frac{1}{|H|} \sum_{k \in I_H} \left(-\partial_{F_{m-1}(\mathbf{x}_k)} L_k + \frac{1}{|H|} \sum_{k \in I_H} \partial_{F_{m-1}(\mathbf{x}_k)} L_k \right)^2$$

  Optimal split of H into nonempty disjoint H_L, H_R then solves

$$\min_{H_L, H_R} \left\{ \frac{|H_L|}{|H|} Q(H_L, \mathcal{D}) + \frac{|H_R|}{|H|} Q(H_R, \mathcal{D}) \right\}$$

- Split in CART based on pure mean squared error reduction, unlike with XGBoost, where γ and λ regularize tree construction.

Our hypothesis

- Using the definition from [5], a *robust learner* is a learning algorithm that is less influenced by noisy data and has
  1. Good performance on data unperturbed by noise.
  2. Low decrease in performance on low levels of noise, without serious performance deterioration when the noise level is high.
- Regularization reduces model overfitting. Since sensitivity to noise is reduced by reducing overfitting, it follows that regularization should reduce sensitivity to noise, i.e. increase learner robustness.
- XGBoost has more ways to regularize than standard gradient boosting methods. Is it more robust on data with corrupted labels?
- How will we measure robustness to noise?

Robustness metrics

- We use two measures of classifier robustness to noise from [5]. Given a fitted classifier f, validation set $\mathcal{D}$, and noise level $\nu \in [0, 1)$, define

$$\mathcal{E}(f, \nu, \mathcal{D}) \triangleq \frac{1 - \mathcal{A}_\nu(f, \mathcal{D})}{\mathcal{A}_0(f, \mathcal{D})} = \frac{1}{\mathcal{A}_0(f, \mathcal{D})} - \frac{\mathcal{A}_\nu(f, \mathcal{D})}{\mathcal{A}_0(f, \mathcal{D})} \tag{7}$$

$$\mathcal{R}(f, \nu, \mathcal{D}) \triangleq \frac{\mathcal{A}_0(f, \mathcal{D}) - \mathcal{A}_\nu(f, \mathcal{D})}{\mathcal{A}_0(f, \mathcal{D})} = 1 - \frac{\mathcal{A}_\nu(f, \mathcal{D})}{\mathcal{A}_0(f, \mathcal{D})} \tag{8}$$

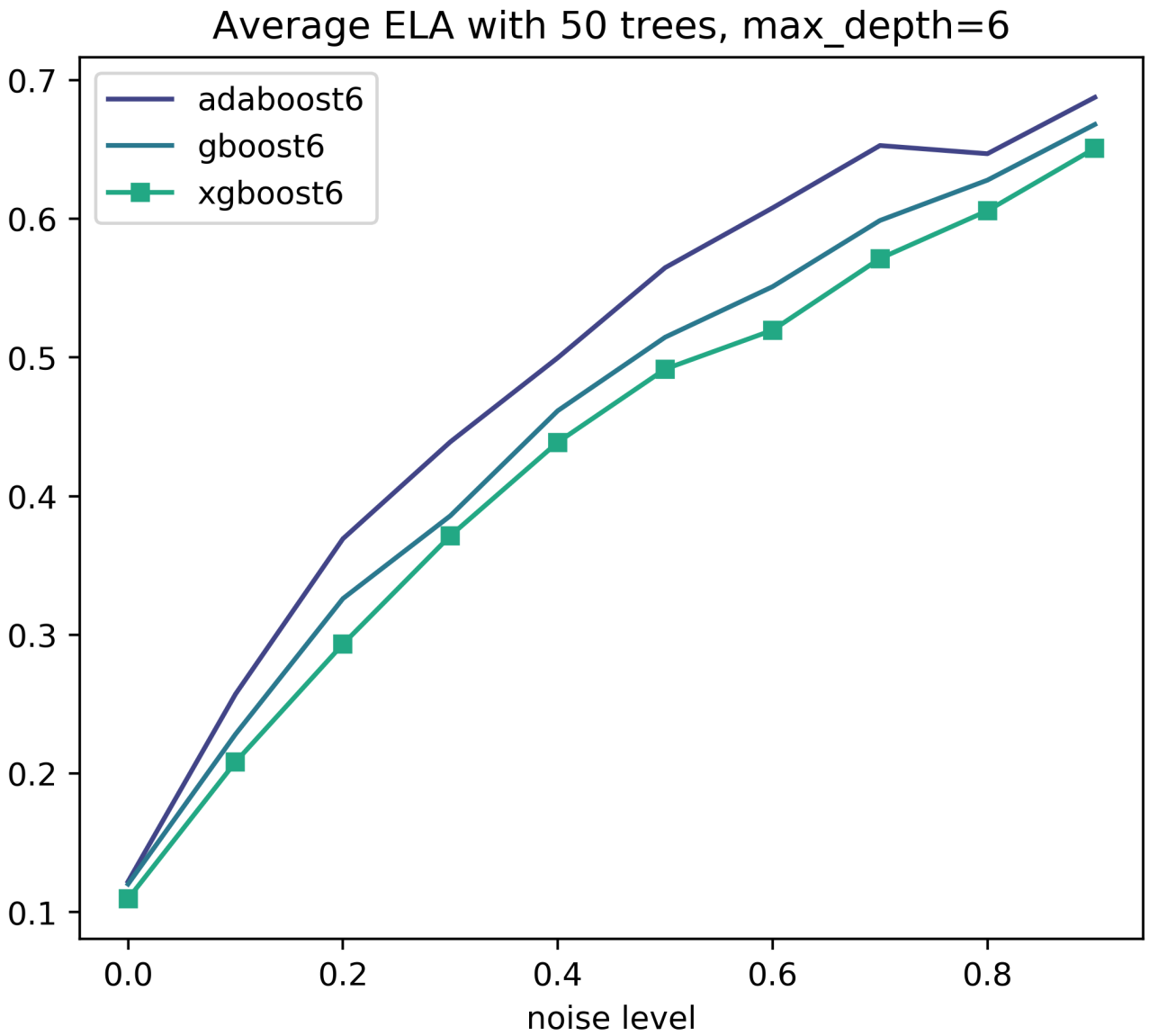
  Here $\mathcal{A}_\nu(f, \mathcal{D})$ is the accuracy of f on $\mathcal{D}$ with noise level $\nu \in [0, 1)$.
- $\mathcal{E}(f, \nu, \mathcal{D})$ is the *equalized loss of accuracy*, or ELA, introduced in [5]. $\mathcal{R}(f, \nu, \mathcal{D})$ is the *relative loss of accuracy*, or RLA.
  - Essentially the **negative** percent change from $\mathcal{A}_0(f, \mathcal{D})$ to $\mathcal{A}_\nu(f, \mathcal{D})$.
- We show later why as stated in [5], ELA is preferred to RLA.

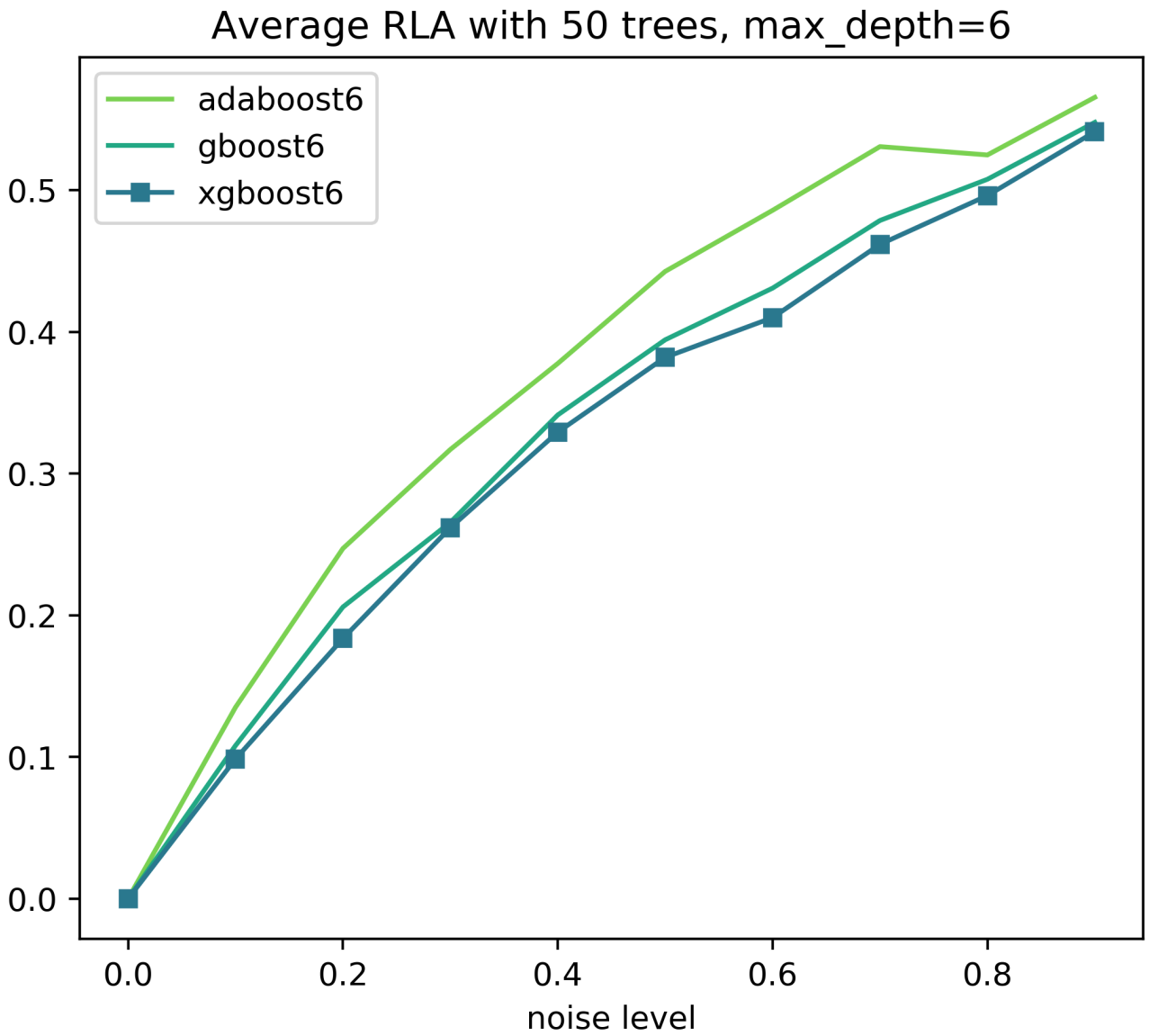
Model evaluation

- Compare **average** robustness of XGBoost gradient tree boosting against `sklearn` AdaBoost and gradient tree boosting.
  - Used 16 data sets with varying input dimension and sample counts, with minimal preprocessing, only removing missing values and binarizing categorical features, for "off-the-shelf" performance.
- Too many data sets and models to run by hand. Instead, built model evaluation system `kyulib`[5] with single entry point.
  - Convenience features include control of model and plotting config through JSON files, model outputs written in pickled dict format, and warm starting to allow reformatting of plots without reevaluating.
  - Simple model evaluation and reevaluation, ex.

```
$ ./noisyeval.py config/depth6trees50.json
```

[5] Source viewable on GitHub here.


Average ELA with 50 trees, max_depth=6
adaboost6
gboost6
xgboost6
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8
noise level

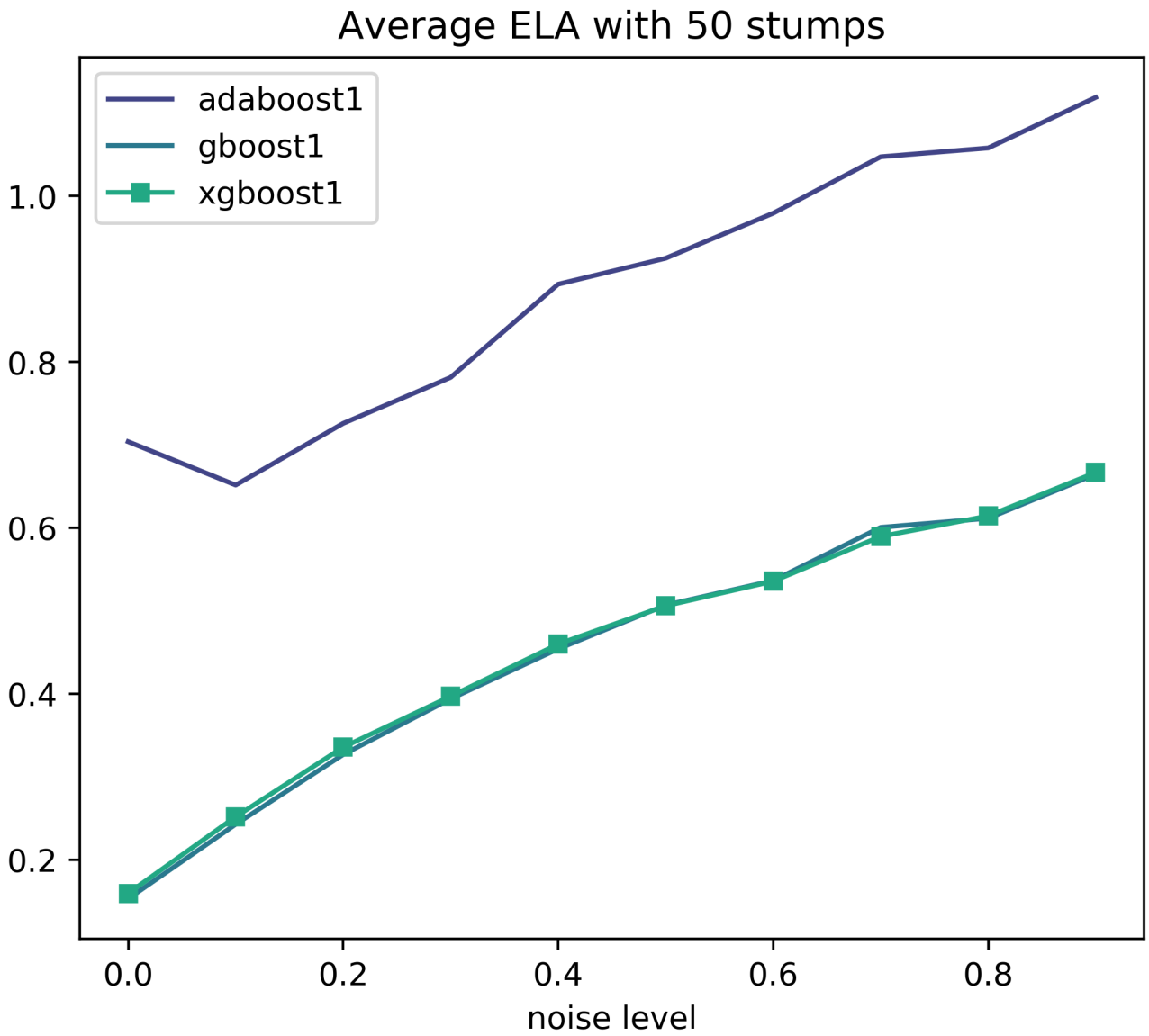

Average RLA with 50 trees, max_depth=6
adaboost6
gboost6
xgboost6
0.0
0.1
0.2
0.3
0.4
0.5
0.0
0.2
0.4
0.6
0.8
noise level

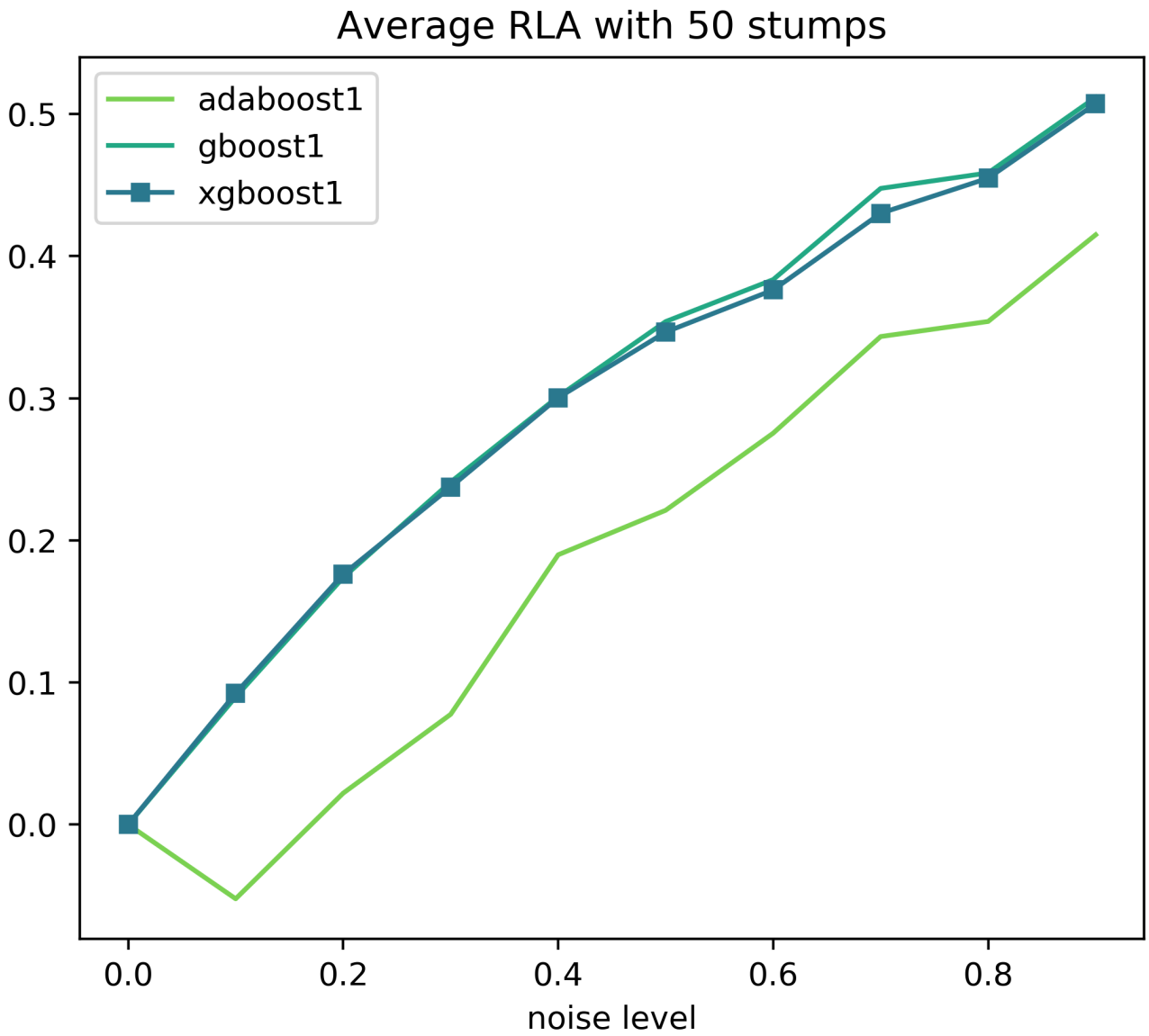
Why prefer ELA?

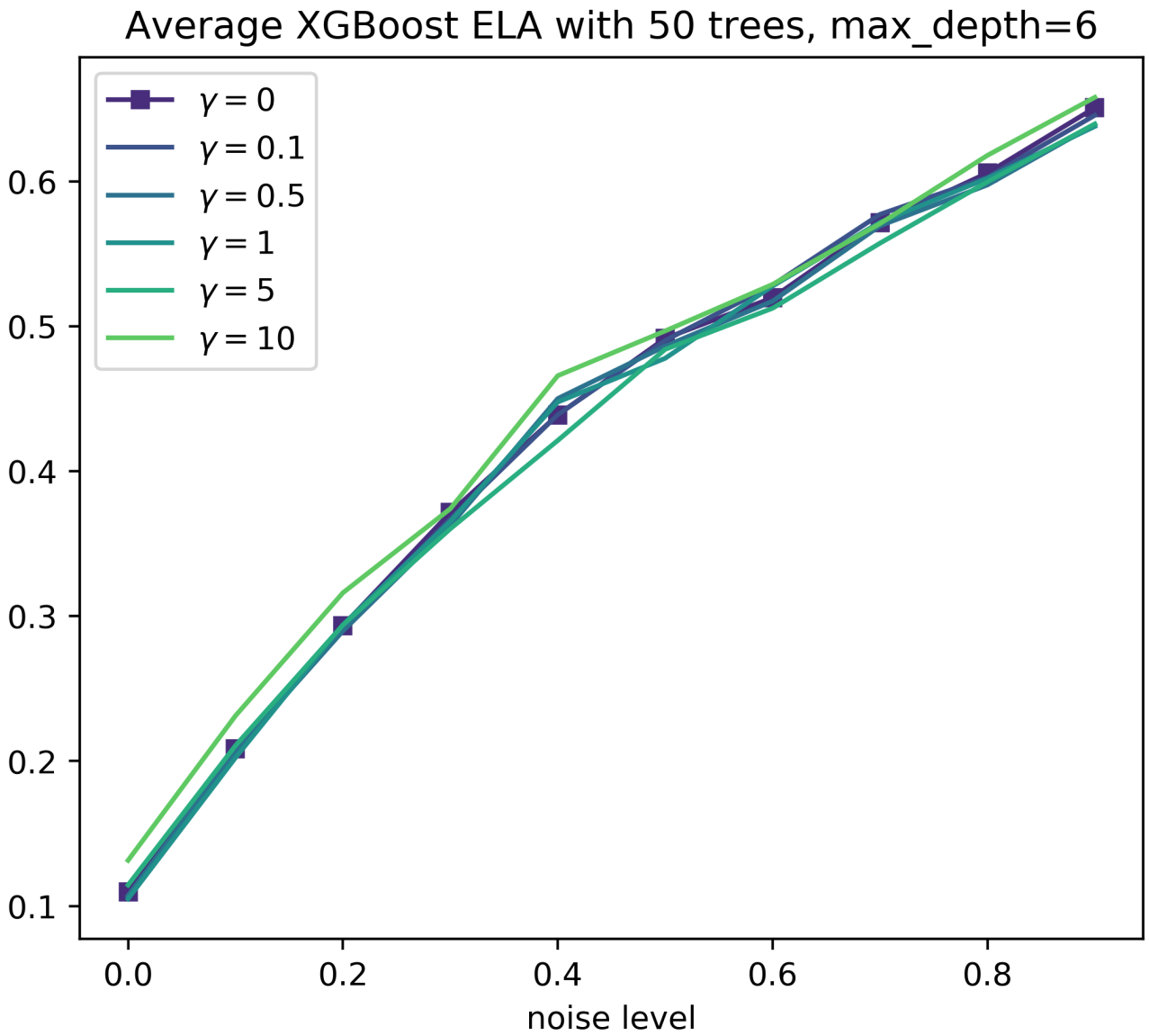
- RLA is natural but has practical shortcomings, discussed in [5]. Ex.
  - Does not account for differences in baseline learner accuracy. Consider two classifiers f, g. Suppose that $\mathcal{A}_\nu(f, \mathcal{D}) < \mathcal{A}_\nu(g, \mathcal{D})$, $\forall \nu \in [0, 1)$, but that $\mathcal{A}_\nu(f, \mathcal{D})/\mathcal{A}_0(f, \mathcal{D}) = \mathcal{A}_\nu(g, \mathcal{D})/\mathcal{A}_0(g, \mathcal{D})$. Then

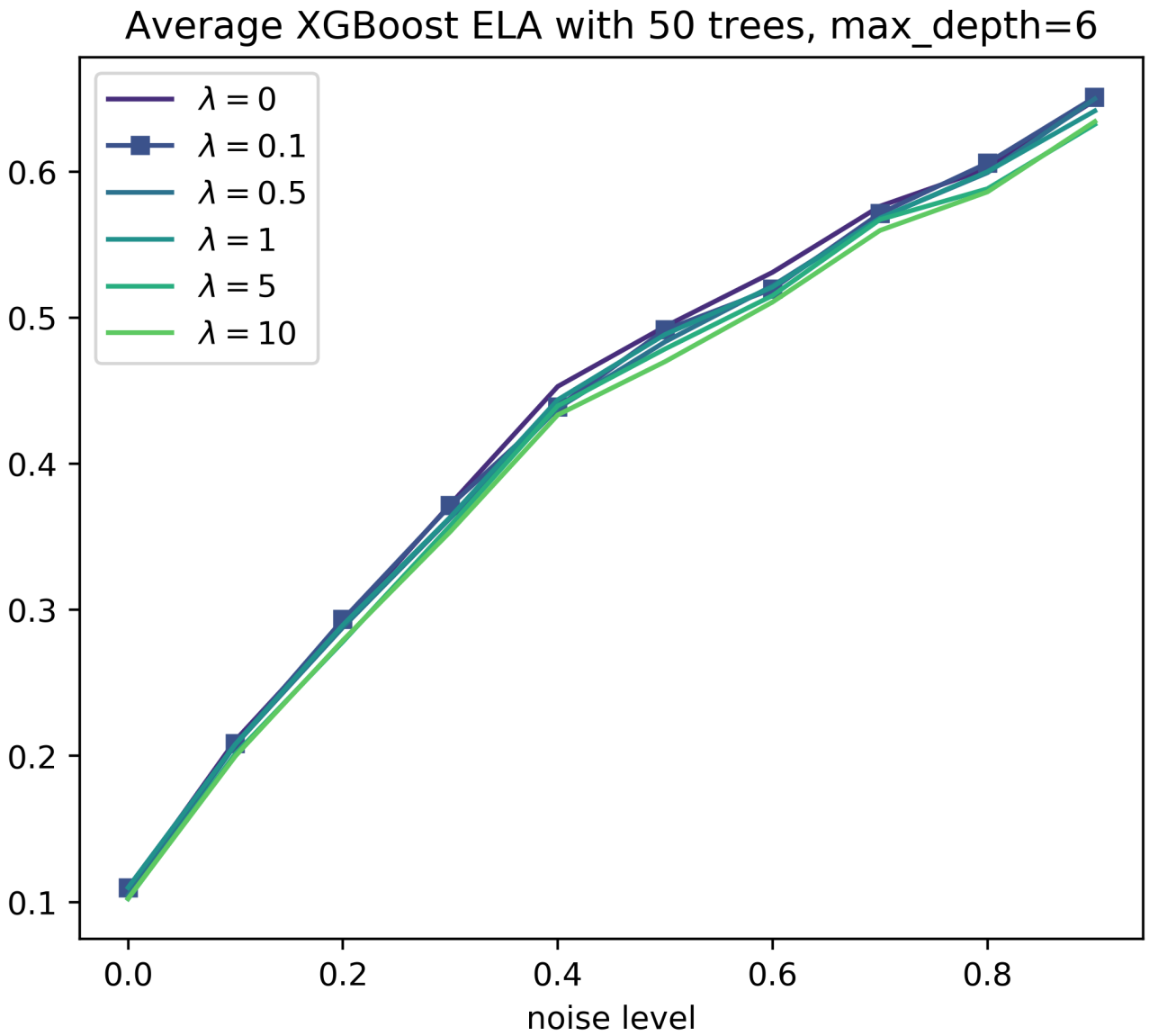
$$\mathcal{R}(f, \nu, \mathcal{D}) = 1 - \frac{\mathcal{A}_\nu(f, \mathcal{D})}{\mathcal{A}_0(f, \mathcal{D})} = 1 - \frac{\mathcal{A}_\nu(g, \mathcal{D})}{\mathcal{A}_0(g, \mathcal{D})} \triangleq \mathcal{R}(g, \nu, \mathcal{D})$$

    f, g equally robust under RLA, although f always has worse accuracy.
  - If for $\nu \in (0, 1)$, for some $f, \mathcal{D}$, $\mathcal{A}_\nu(f, \mathcal{D}) > \mathcal{A}_0(f, \mathcal{D})$, $\mathcal{R}(f, \nu, \mathcal{D}) < 0$. More frequently occurs for f with poor baseline accuracy.
    - Seemingly excellent but false robustness of f to noise.
- By changing $\mathcal{A}_0$ to 1 in the numerator in (7), ELA adds correction for differences in baseline learner accuracy, which means
  - Lower baseline accuracy equals higher starting ELA
  - ELA measure is guaranteed nonnegative for any $f, \nu, \mathcal{D}$


Average ELA with 50 stumps
adaboost1
gboost1
xgboost1
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
noise level


Average RLA with 50 stumps
adaboost1
gboost1
xgboost1
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
noise level


Average XGBoost ELA with 50 trees, max_depth=6
$\gamma = 0$
$\gamma = 0.1$
$\gamma = 0.5$
$\gamma = 1$
$\gamma = 5$
$\gamma = 10$
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8
noise level


Average XGBoost ELA with 50 trees, max_depth=6
$\lambda = 0$
$\lambda = 0.1$
$\lambda = 0.5$
$\lambda = 1$
$\lambda = 5$
$\lambda = 10$
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8
noise level

Conclusions

- Showed XGBoost has better "off-the-shelf" robustness to label noise on average compared to standard gradient tree boosting or AdaBoost
- Moderate effects on average robustness when γ and λ are adjusted
- XGBoost is a better alternative to `sklearn` for tree boosting

Future considerations

- Examine changes in robustness with respect to other hyperparameters
  - Row/column subsampling fraction, learning rate, tree dropout
- Explore regularized random forest implementation
  - Do γ and λ have greater effect on larger trees?
- Study underlying system implementation and weighted quantile sketch, used for approximate split-finding with large training sets, to better understand XGBoost's high scalability

AdaBoost analysis

Recall two-class AdaBoost fits the additive model $F : \mathcal{X} \to \mathbb{R}$, where

$$F(\mathbf{x}) = \sum_{m=1}^{M} \phi_m(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}_k)$$

Here $\alpha_m \in \mathbb{R}$, $f_m : \mathcal{X} \to \{-1, 1\}$, $f_m \in \mathcal{H}$ for appropriate $\mathcal{H}$, where

$$f_m = \arg\min_f \sum_{k \in I^c(f)} w_m^{(k)}$$

Here $\mathcal{D}$ is the training set of examples $(\mathbf{x}_k, y_k)$, $w_m^{(k)}$ the mth iteration weight for $(\mathbf{x}_k, y_k)$, where $\sum_{k=1}^{|\mathcal{D}|} w_m^{(k)} = 1$ and the index set $I^c(f) \triangleq \{k \in \mathbb{N} : y_k \neq f(\mathbf{x}_k), (\mathbf{x}_k, y_k) \in \mathcal{D}\}$. Define probability mass function $p_m : \{1, \ldots |\mathcal{D}|\} \times \mathcal{D} \to [0, 1]$, where $p_m(k, \mathcal{D}) \triangleq w_m^{(k)}$.

AdaBoost analysis

Suppose sequence $\{p_m\}$ converges uniformly to p_∞, i.e. $\exists M' \in \mathbb{N}$ s.t. $\forall m \geq M'$, $\forall (k, \mathcal{D}) \in \{1, \ldots |\mathcal{D}|\} \times 2^{\mathcal{X} \times \mathcal{Y}}$, $|p_m(k, \mathcal{D}) - p_\infty(l, \mathcal{D})| < \varepsilon$, $\forall \varepsilon > 0$. It then follows that $\forall m \geq M'$, $\phi_m = \alpha_{M'} f_{M'}$. Then, for $M > M'$,

$$F(\mathbf{x}) \triangleq \sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}) = \sum_{m=1}^{M'-1} \alpha_m f_m(\mathbf{x}) + \sum_{n=M'}^{M} \alpha_{M'} f_{M'}(\mathbf{x})$$

$$= \sum_{m=1}^{M'-1} \alpha_m f_m(\mathbf{x}) + (M - M' + 1)\alpha_{M'} f_{M'}(\mathbf{x})$$

Recall the final classifier $G : \mathcal{X} \to \{-1, 1\}$ is such that $G(\mathbf{x}) \triangleq \text{sgn} F(\mathbf{x})$. If we assume $\alpha_{M'} > 0$, then as $M \to \infty$, $G \to f_{M'}$ uniformly. In other words, as $M \to \infty$, AdaBoost will eventually find the learner $f_{M'}$ at some iteration M' that achieves the minimium weighted misclassification rate on $\mathcal{D}$.

Computing ζ_m

Suppose $\mathcal{H}$ is a reproducing kernel Hilbert space that is a subset of $L^2(\mathcal{X})$. Define the expected loss functional $J : \mathcal{H} \to \mathbb{R}_+$, where

$$J[f] = \mathbb{E}[L(Y, f(X))] = \int_{\mathcal{X}\times\mathcal{Y}} L(y, f(\mathbf{x}))\psi_{X,Y}(\mathbf{x}, y)\, dy\, d\mathbf{x}$$

Here $L : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}_+$ is a convex loss function and $\psi_{X,Y}$ is the joint density of random variables X, Y, and implicitly $\forall f \in \mathcal{H}$, $J[f] < \infty$. We wish to solve $\min_f J[f]$, but since this is in general impossible, we consider functional gradient descent. Suppose at iteration m, we have prior guess F_{m-1}, and want to find the step ζ_m such that $F_m = F_{m-1} + \zeta_m$, where naturally $\zeta_m \triangleq \arg\min_{\zeta\in\mathcal{H}} J[F_{m-1} + \zeta]$. As J is a linear functional, we consider its first variation $\delta J[F_{m-1}]$ and define a perturbation in F_{m-1} as $\delta F_{m-1} \triangleq \varepsilon\sigma$, for arbitrarily small $\varepsilon > 0$, $\sigma \in \mathcal{H}$.

Computing ζ_m

The first variation $\delta J[F_{m-1}]$ can then be written as

$$\begin{aligned}\delta J[F_{m-1}] &= J[F_{m-1} + \delta F_{m-1}] - J[F_{m-1}] \\ &= \mathbb{E}[L(Y, F_{m-1}(X)) + \delta F_{m-1}(X)] - \mathbb{E}[L(Y, F_{m-1}(X))]\end{aligned}$$

We can write $J[F_{m-1} + \delta F_{m-1}]$ using a first order Taylor expansion as

$$\mathbb{E}\left[L(Y, F_{m-1}(X)) + \frac{\partial L}{\partial F_{m-1}(X)}(Y, F_{m-1}(X))\delta F_{m-1}(X)\right]$$

Substituting, we then can write $\delta J[F_{m-1}]$ as

$$\delta J[F_{m-1}] = \mathbb{E}\left[\frac{\partial L}{\partial F_{m-1}(X)}(Y, F_{m-1}(X))\delta F_{m-1}(X)\right]$$

Computing ζ_m

Since we assumed $J[f] < \infty$, we can apply Fubini's theorem to write $\delta J[F_{m-1}]$ in terms of iterated integrals. Factoring $\psi_{X,Y}$ into the product of conditional density $\psi_{Y|\mathbf{x}}$ and marginal density ψ_X, we have

$$\begin{aligned}
\delta J[F_{m-1}] &= \int_{\mathcal{X}} \left[\int_{\mathcal{Y}} \frac{\partial L}{\partial F_{m-1}(\mathbf{x})}(y, F_{m-1}(\mathbf{x})) \psi_{Y|\mathbf{x}}(y)\, dy \right] \psi_X(\mathbf{x}) \delta F_{m-1}(\mathbf{x})\, d\mathbf{x} \\
&= \int_{\mathcal{X}} \psi_X(\mathbf{x}) \mathbb{E}\left[\frac{\partial L}{\partial F_{m-1}(\mathbf{x})}(Y, F_{m-1}(\mathbf{x})) \right] \delta F_{m-1}(\mathbf{x})\, d\mathbf{x} \\
&= \varepsilon \int_{\mathcal{X}} \psi_X(\mathbf{x}) \mathbb{E}\left[\frac{\partial L}{\partial F_{m-1}(\mathbf{x})}(Y, F_{m-1}(\mathbf{x})) \right] \sigma(\mathbf{x})\, d\mathbf{x}
\end{aligned}$$

Since $\mathcal{H} \subset L^2(\mathcal{X})$, we can define the functional gradient as in [1], as $\delta J[F_{m-1}]$ is the L^2 inner product $\varepsilon \langle \psi_X \mathcal{E}_Y, \sigma \rangle$, where

$$\mathcal{E}_Y(\mathbf{x}) \triangleq \mathbb{E}\left[\frac{\partial L}{\partial F_{m-1}(\mathbf{x})}(Y, F_{m-1}(\mathbf{x})) \right]$$

Computing ζ_m

Therefore, the functional gradient of J evaluated at F_{m-1} is

$$\nabla J[F_{m-1}] = \psi_X \mathcal{E}_Y$$

Recall that we want a step $\zeta_m = \arg\min_{\zeta \in \mathcal{H}} J[F_{m-1} + \zeta]$. Functional gradient descent with a scalar step size would thus prescribe that

$$\zeta_m = -\gamma_m \nabla J[F_{m-1}]$$

Here $\gamma_m > 0$ is the step size. Evaluated at $\mathbf{x} \in \mathcal{X}$ and expanded, we have

$$\zeta_m(\mathbf{x}) = \gamma_m \psi_X(\mathbf{x}) \mathbb{E}\left[-\frac{\partial L}{\partial F_{m-1}(\mathbf{x})}(Y, F_{m-1}(\mathbf{x}))\right]$$

This is exactly the result stated previously in (5).

References


Bagnell, D. (2010, Nov 14). *Functional gradient descent* [PDF]. Carnegie Mellon University. http://www.cs.cmu.edu/~16831-f12/notes/F12/16831_lecture21_danielsm.pdf.


Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. arXiv. https://arxiv.org/pdf/1603.02754.pdf.


Chen, T. (2014, October 22). *Introduction to Boosted Trees* [PDF]. University of Washington. https://homes.cs.washington.edu/~tqchen/data/pdf/BoostedTree.pdf.


Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning* (2nd. ed.). Springer.


Saez, J. A., Luengo, J., & Herrera, F. (2015). Evaluating the classifier behavior with noisy data considering performance and robustness: The Equalized Loss of Accuracy measure. *Neurocomputing*, *176*, 26-35. https://doi.org/10.1016/j.neucom.2014.11.086.