SMARTboost learning for tabular data

Paolo Giordani[1]

[1]Norwegian Business School, Oslo

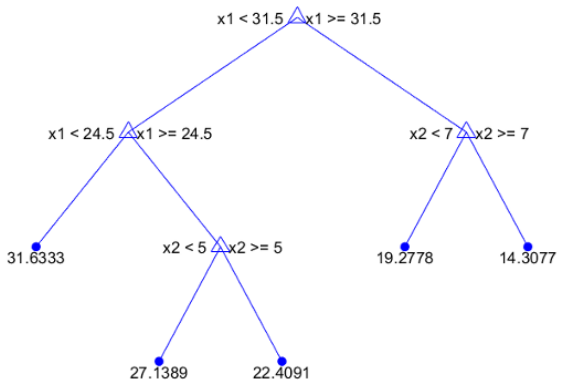
November 2022

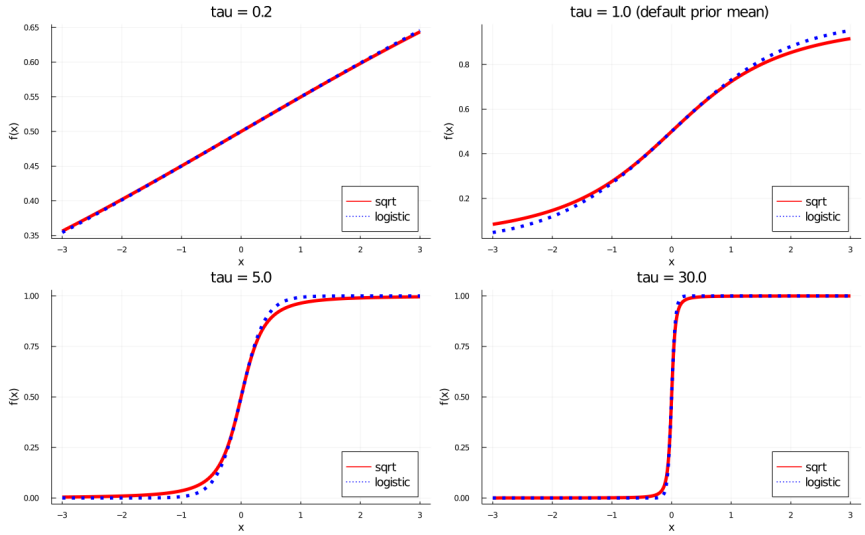
What is SMARTboost?

- Boosting not of standard regression trees, but of 'trees' with smooth threshold.
- Expected to be more accurate with continuous or mixed discrete-continuous features.
- Modeling choices and heuristics to reduce the high computing costs.
- Thoughtful default priors reduce the need for CV.
- Priors also improve performance with small n and/or small SNR.
- Open source Julia code, with R wrapper.

https://github.com/PaoloGiordani/SMARTboost.jl

Standard and Smooth Trees





Why SMARTboost?

Originally designed to be more effective than GBM for finance data, which are often:

1. Low SNR and panel data $\Rightarrow$ Low *effective* sample size.
2. Highly persistent and/or non-stationary features $\Rightarrow$ more frequent need to extrapolate.
3. Reasonably well-behaved continuous distributions.

Work in progress testing on a broader range of datasets.

Why low effective sample size

- Panel data with substantial cross-correlation, e.g. equal weighted portfolio of SP500 stocks $\approx$ 5 uncorrelated stocks.
- Low SNR. e.g. $R^2 = 0.02\,(0.01)$ requires roughly 50 (100) times larger n than $R^2 = 0.5$ to have the same $var(f(x))$.
- $n = 500 \times 20 \times 12 = 120k$ could be $\approx n = 12$ at $R^2 = 0.5$.

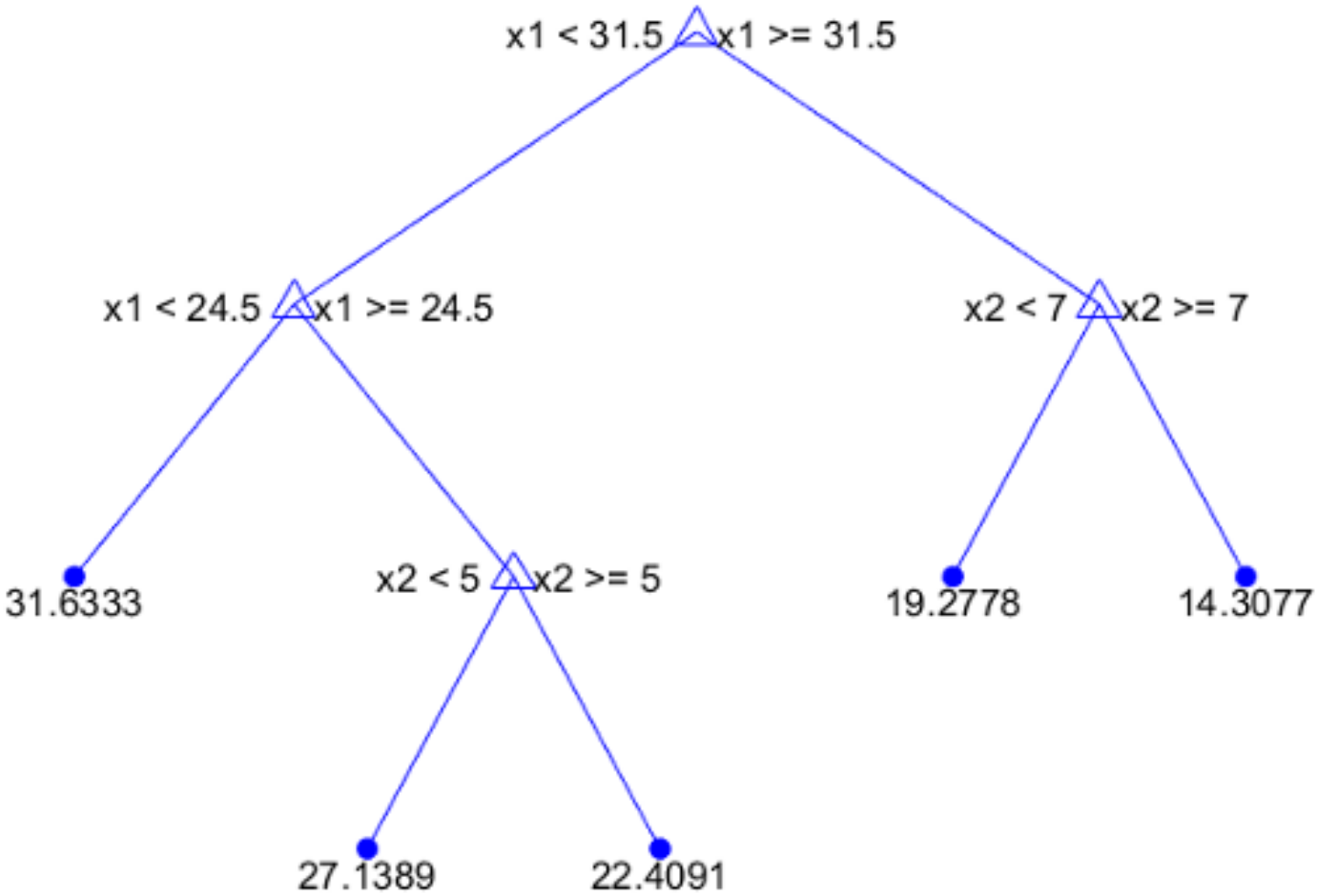
I want something that can do well at low SNR but also be able to recover very complex functions if n and/or SNR allow it.

Boosting Trees

"Boosting lives at the cutting edge of modern prediction methodology. They fit models of breathtaking complexity ... and are routinely used as prediction engines in a wide variety of industrial and scientific applications." Efron and Hastie (2016)

- Current default method for tabular data, particularly heterogeneous data. Unified methodology for regression and classification.
- Several open-source, multi-platform, industry-quality implementations: XGBoost (Chen and Guestrin 2016), LightGBM (Microsoft), CatBoost (Yandex).
- Fairly similar performance. I focus on the best-known, XGBoost. Used in many ML platforms: Google's AI Cloud, Amazon's Azure, Uber's Michelangelo.

A Simple Regression Tree (or Decision Tree)



Regression Trees

With a single tree:

$$E(y_i|x_i) = \sum_{m=1}^{M} \beta_m I(x_i \in R_m),$$

where the dummies $I(x_i \in R_m)$ are non-overlapping and the estimation of β_m is straightforward given $I(x_i \in R_m)$, which in turn depends on all parent split variables and split points. Full optimization impossible: heuristic greedy optimizers.

Boosting Trees for regression problems

- A single tree, however deep, fits and forecasts poorly. Ensembles of trees much more promising. Full optimization impossible. MCMC (BART) challenging.
- Boosting can be seen as a heuristic to fit an ensemble of B trees (or other base):

$$E(y|x) = \Sigma_{b=1}^{B} T_b(x; \theta_b)$$

- Recursively fits the residuals from the previous ensemble.

Boosting trees

1. Initialize $f_0(x_i) = 0$, $r_i = y_i$, *for* $i = 1, ..., n$. Then repeat step 2) B times.
2. Fit a single tree T_b to $r_{1:n}$. Update $\widehat{f}_b(x_i) = \widehat{f}_{b-1}(x_i) + \lambda T_b(x_i)$ and $r_i = y_i - \widehat{f}_b(x_i)$, where $0 < \lambda \leq 1$.

- Now world-class fit and predictions in many problems.

Boosting Trees for regression problems

- A single tree, however deep, fits and forecasts poorly. Ensembles of trees much more promising. Full optimization impossible. MCMC (BART) challenging.
- Boosting can be seen as a heuristic to fit an ensemble of B trees (or other base):

$$E(y|x) = \Sigma_{b=1}^{B} T_b(x; \theta_b)$$

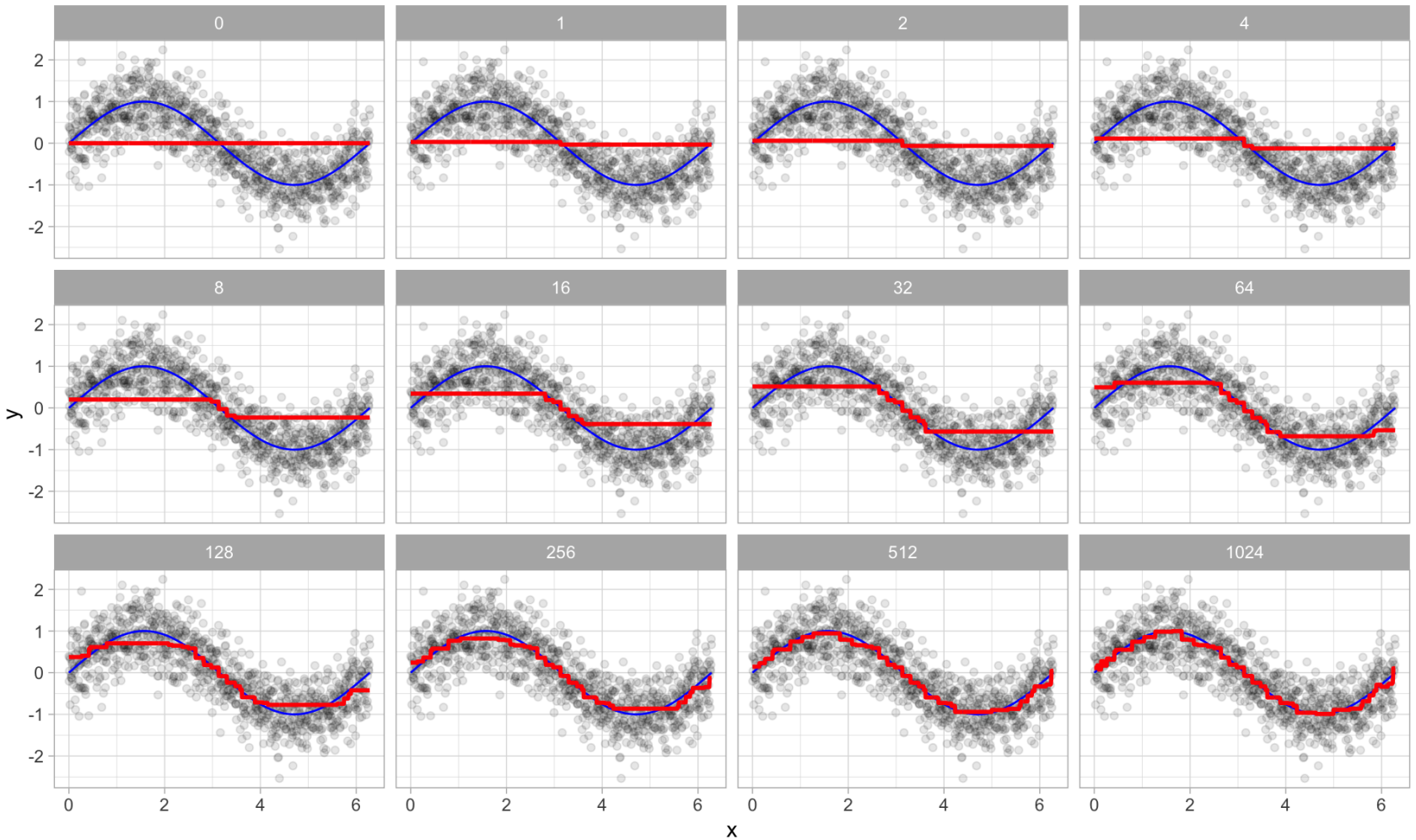
- Recursively fits the residuals from the previous ensemble.

Boosting trees

1. Initialize $f_0(x_i) = 0$, $r_i = y_i$, *for* $i = 1, ..., n$. Then repeat step 2) B times.
2. Fit a single tree T_b to $r_{1:n}$. Update $\widehat{f}_b(x_i) = \widehat{f}_{b-1}(x_i) + \lambda T_b(x_i)$ and $r_i = y_i - \widehat{f}_b(x_i)$, where $0 < \lambda \leq 1$.

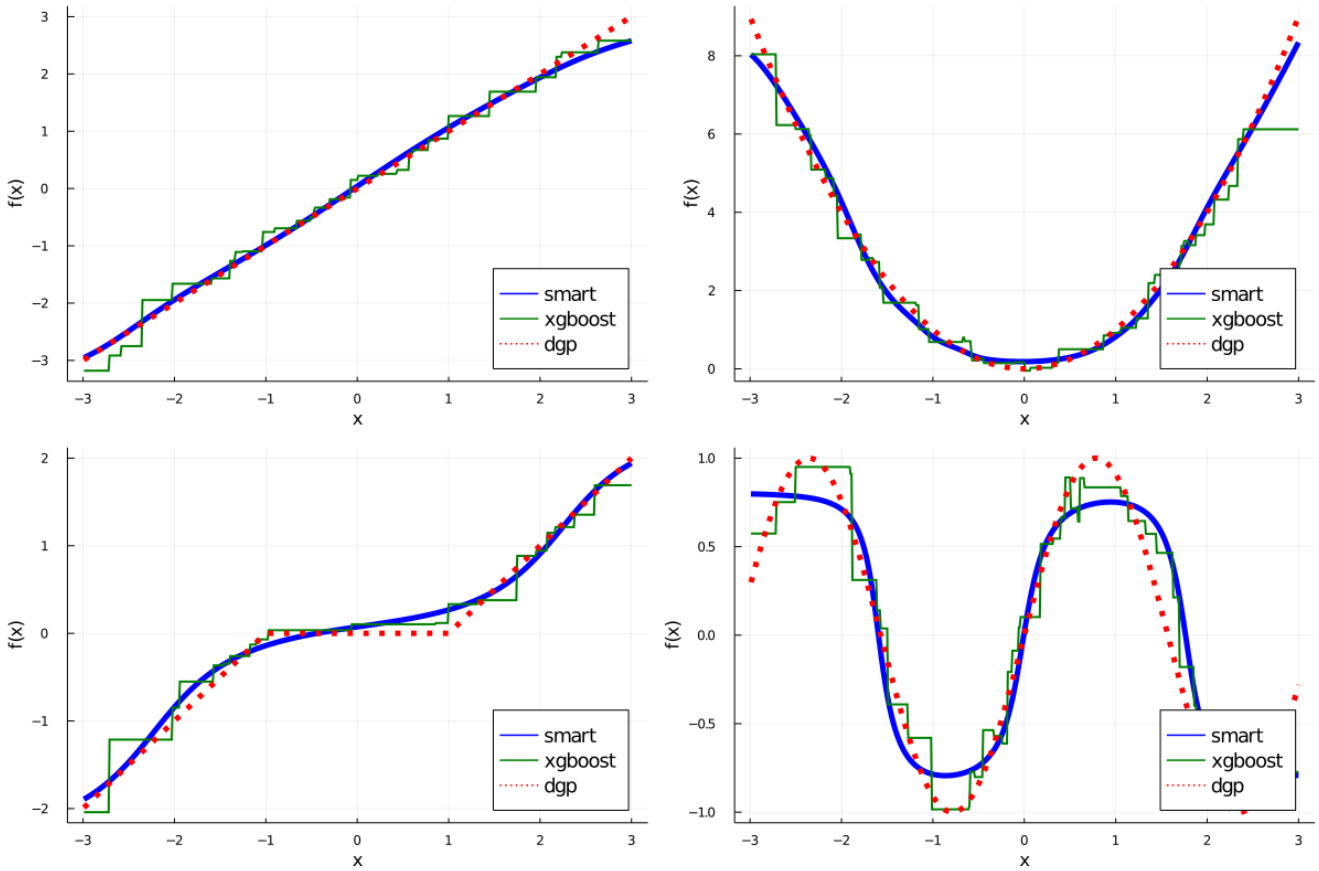
- Now world-class fit and predictions in many problems.

Boosting in action



"Why Does XGBoost Win Every Forecasting Competition?"

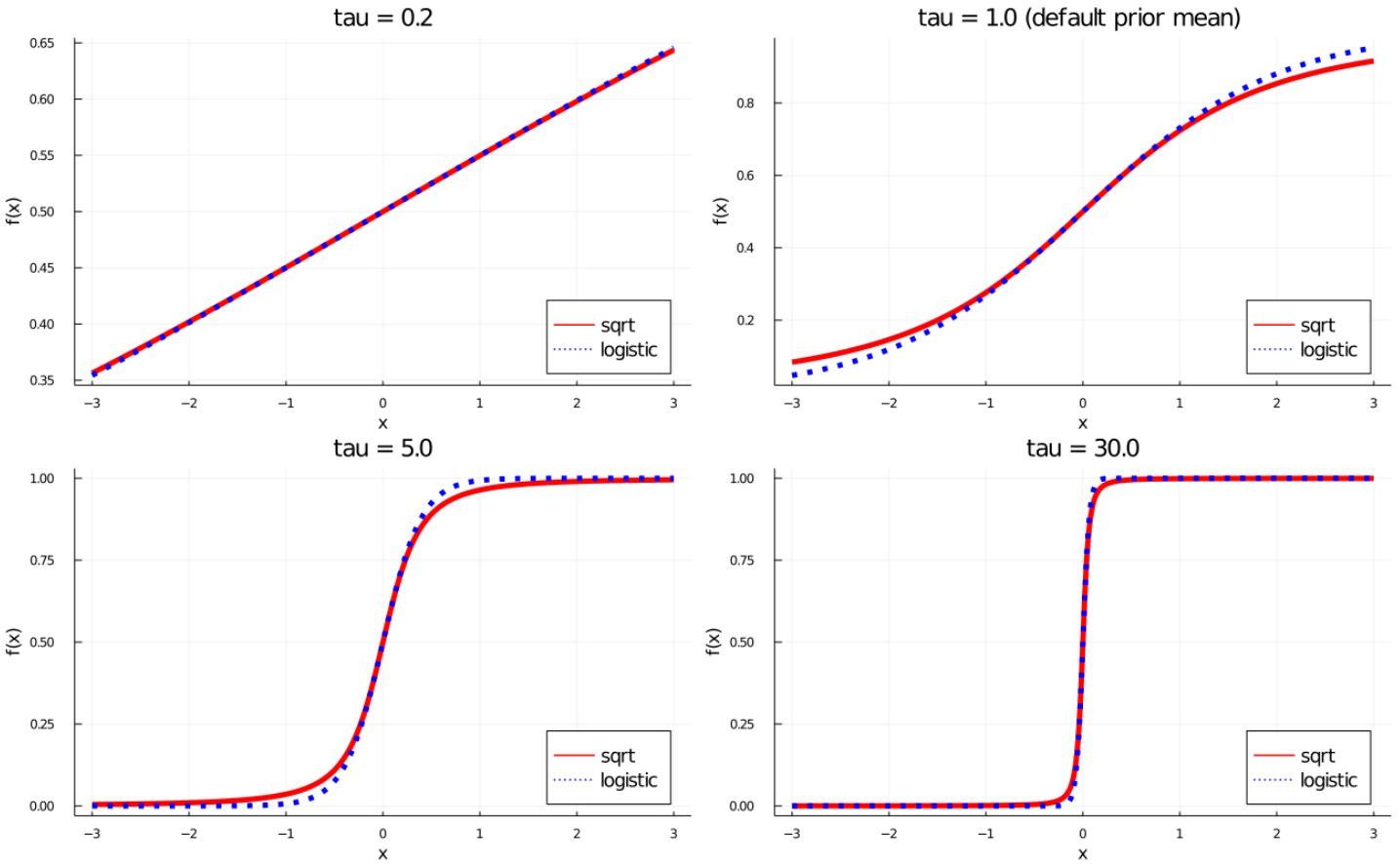
Understandable with binary features. Otherwise surprising that step functions should do so well ...



XGBoost wins forecasting competitions because ...

- Can capture additive nonlinearities and interaction effects in high dimensions (large p), with any combination of continuous and discrete variables.
- Automatic feature selection.
- Robust to leverage points, and messy, noisy, highly non-Gaussian features. Little affected by feature transformation.
- Model complexity is determined automatically by pseudo-out-of-sample fit.

Smooth Trees



Why aren't smooth trees used?

- The idea of a Fuzzy Tree goes back to Chang and Pavlidis (1977) and Jang (1994). Olaru and Wehenkel (2003), De Rosa and Medeiros (2008).
- Linero and Yang (2018) fully Bayesian (MCMC) estimation of an ensemble of probabilistic trees.
- More flexible. Should perform better in many instances. Yet never used.
- Computations are several orders of magnitude slower. Current solutions not practical.

Why are smooth trees so slow?

A single smooth tree can be written as

$$\boldsymbol{y} \sim N(\mathbf{G}\beta, \sigma^2\mathbf{I}), \tag{1}$$

where *at every node*, we need to: a) build the *full* $\boldsymbol{G(\iota, \tau, \mu)}$, which is $(n, 2^{depth})$, b) compute the *full* matrix $\mathbf{G'G}$.
Boosting calls for evaluating a split at every quantile of every feature, and now also to optimize over the smoothness parameter τ.

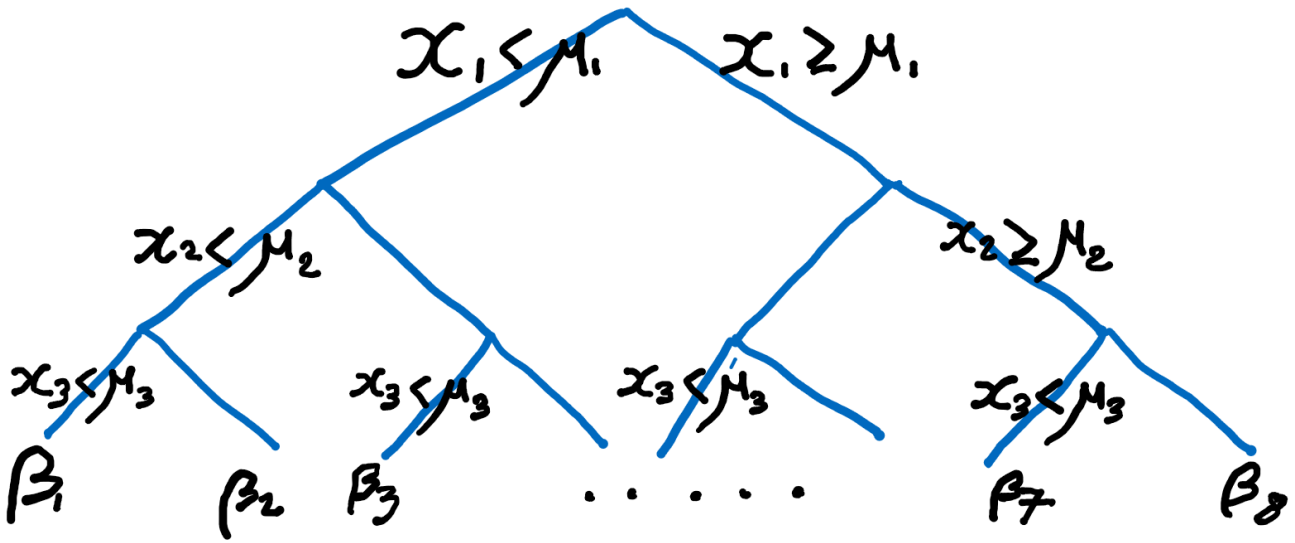
SMARTboost

1. A different base (Symmetric Trees) is roughly 10-20 times faster and often fits better, particularly at low SNR.
2. A set of algorithms and short-cuts to further increase speed with no or minimal loss of fit.
3. A set of Bayesian priors (MAP inference) to reduce the need for cv and hence computing time, and improve performance for low signal-to-noise.

Resulting in great performance even with low SNR and for many DGPs, and feasible for 'large' n (in the millions).

Symmetric trees

A Symmetric Tree (or Oblivious Tree) restricts the splitting variable and point to be the same for all nodes at a given depth. For standard trees, fitting is not any faster. Implemented in CatBoost because forecasting is faster, and at least as good on average. For smooth trees of depth 4 (5), it is 8 (17) times faster.



Symmetry is restrictive for one tree, but not for an ensemble of trees

- Imposing symmetry is restrictive for a single tree, but not for a tree ensemble $E(y|x) = \Sigma_{b=1}^{B} T_b(x; \theta_b)$, where it acts more like a prior/penalization.
- Ensembles of symmetric trees can approximate ensembles of (shallow) standard trees arbitrarily well asymptotically, because:

1. A shallow non-symmetric (standard) tree can be represented as a deeper symmetric tree.
2. A non-symmetric (standard) tree can be represented as a small collection of symmetric trees.

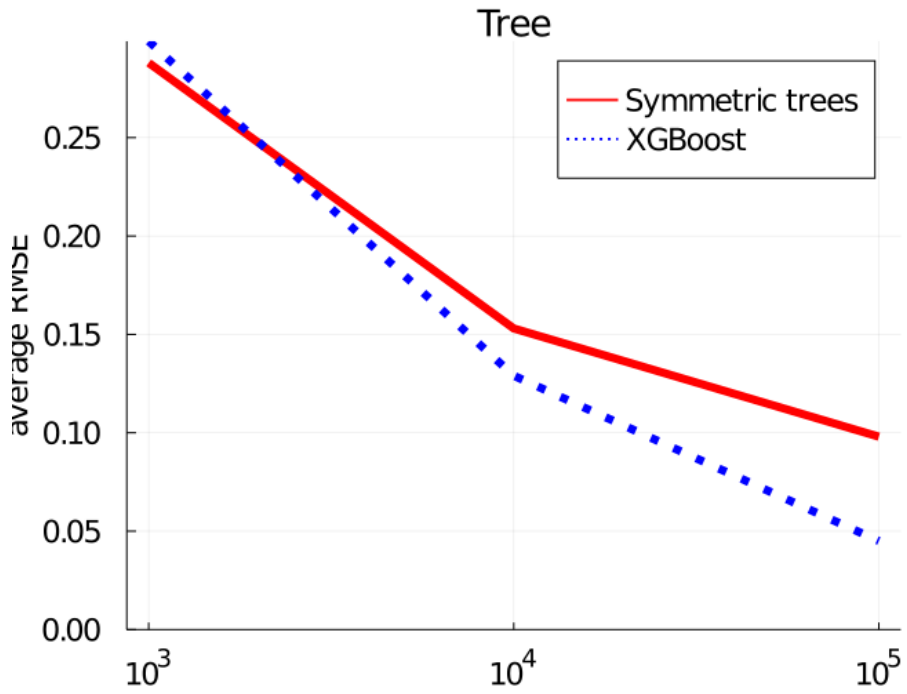
Symmetry is restrictive for one tree, but not for an ensemble of trees

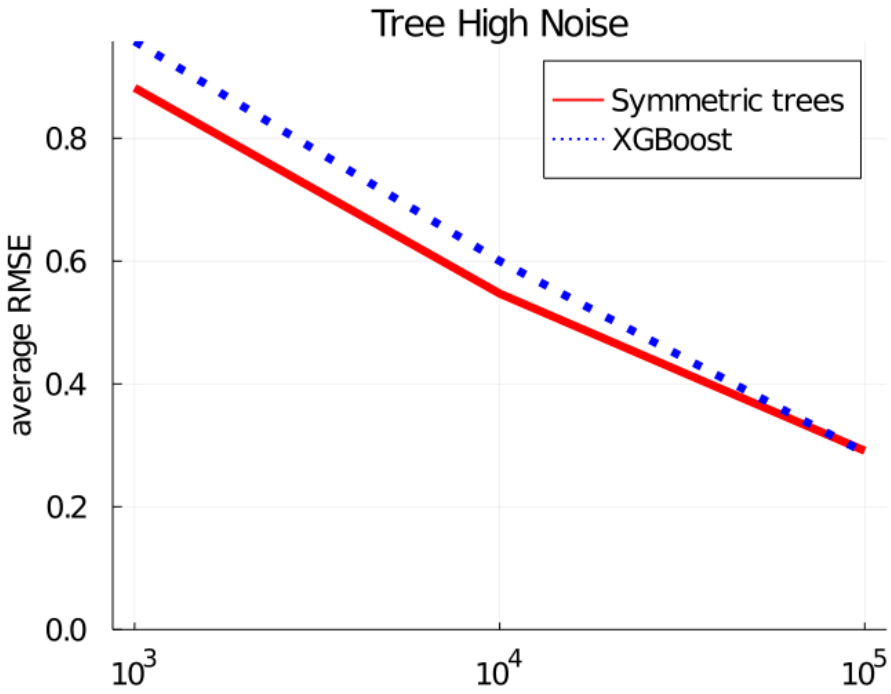
- Imposing symmetry is restrictive for a single tree, but not for a tree ensemble $E(y|x) = \Sigma_{b=1}^{B} T_b(x; \theta_b)$, where it acts more like a prior/penalization.
- Ensembles of symmetric trees can approximate ensembles of (shallow) standard trees arbitrarily well asymptotically, because:

1. A shallow non-symmetric (standard) tree can be represented as a deeper symmetric tree.
2. A non-symmetric (standard) tree can be represented as a small collection of symmetric trees.

Symmetric trees outperform for low n and/or low SNR

DGP is a tree with no symmetry (all splits on different features, split points and coefficients random).





Speeding up smooth trees

1. Symmetric trees are 8 (17) times faster at depth $= 4$ (5).
2. Two phases: i) faster feature selection, with a very rough optimization, ii) refine optimization only for the selected feature. 3-5 times faster.
3. Refinement: univariate optimization of μ on a grid of τ. Very robust and can be parallelized.
4. Square root sigmoid instead of exponential (10 times faster).

50-100 times faster. Still 10-30 times slower than XGBoost for $10k \leq n \leq 1m$.

SMARTboost prior on β (regularization) requires no cv

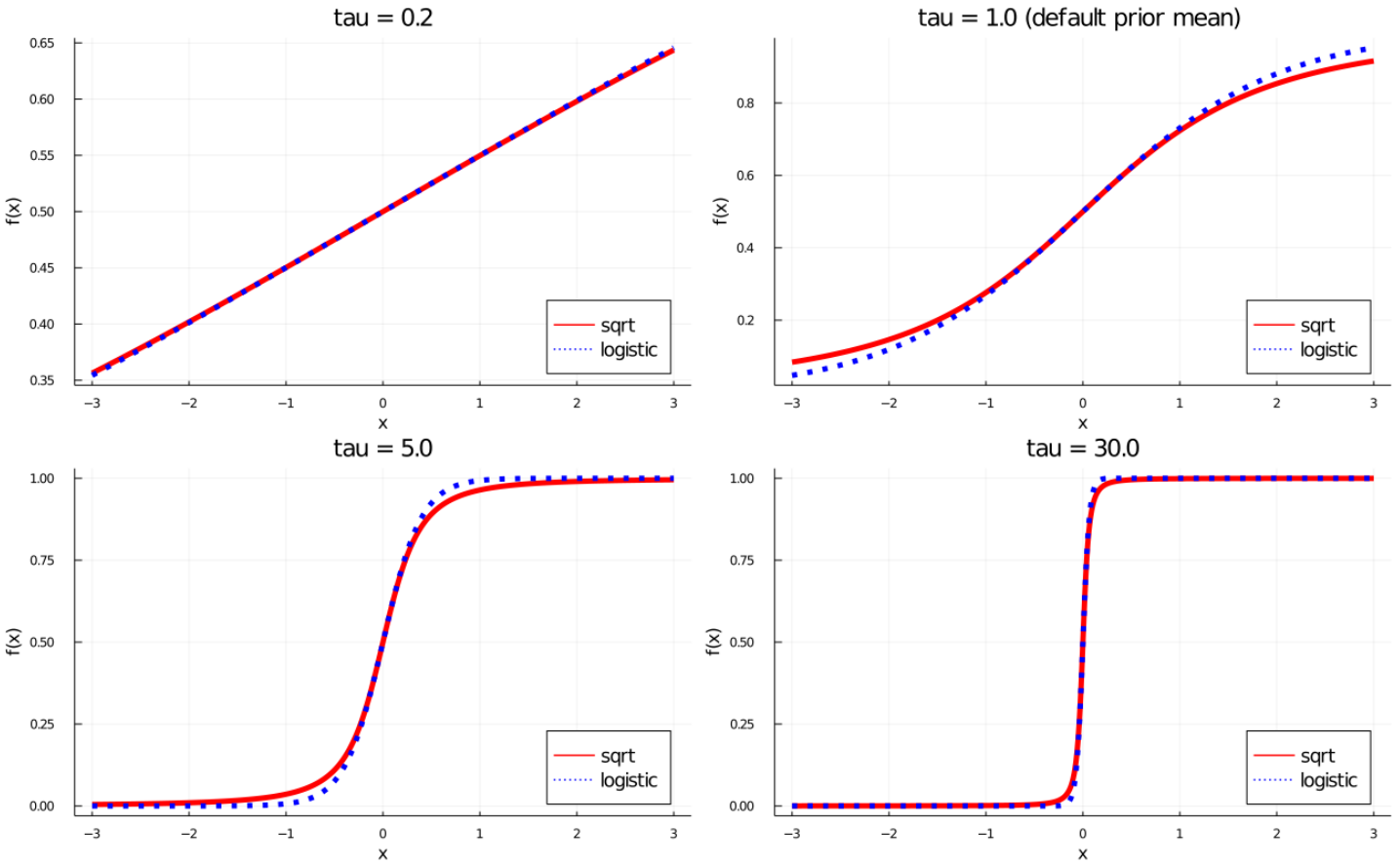
Instead of regularization in the form $\eta\beta'\beta$, prior on β formulated in terms of R^2 of the entire tree (automatically calibrated on the first tree)

$$\beta \sim N(\mathbf{0}, \mathbf{P}_\beta^{-1})$$

$$\mathbf{P}_\beta = \left(\frac{\mathrm{trace}(\mathbf{G'G})/\mathrm{n}}{\mathrm{var}(\mathbf{r}) \times \mathrm{R}_\mathrm{p}^2} \right) \times \mathbf{I},$$

adapts automatically to SNR, depth, and smoothness. 'Coherent' prior: β drawn from the prior imply $\boldsymbol{G}\beta$ that, given $\boldsymbol{r}$ and $\boldsymbol{G}$, result in $E(R^2) = R_p^2$).

Default prior centered on very smooth f(x)



SMARTboost prior encourages smoothness without imposing it

Default prior centered on near-linearity, informative but not dogmatic:

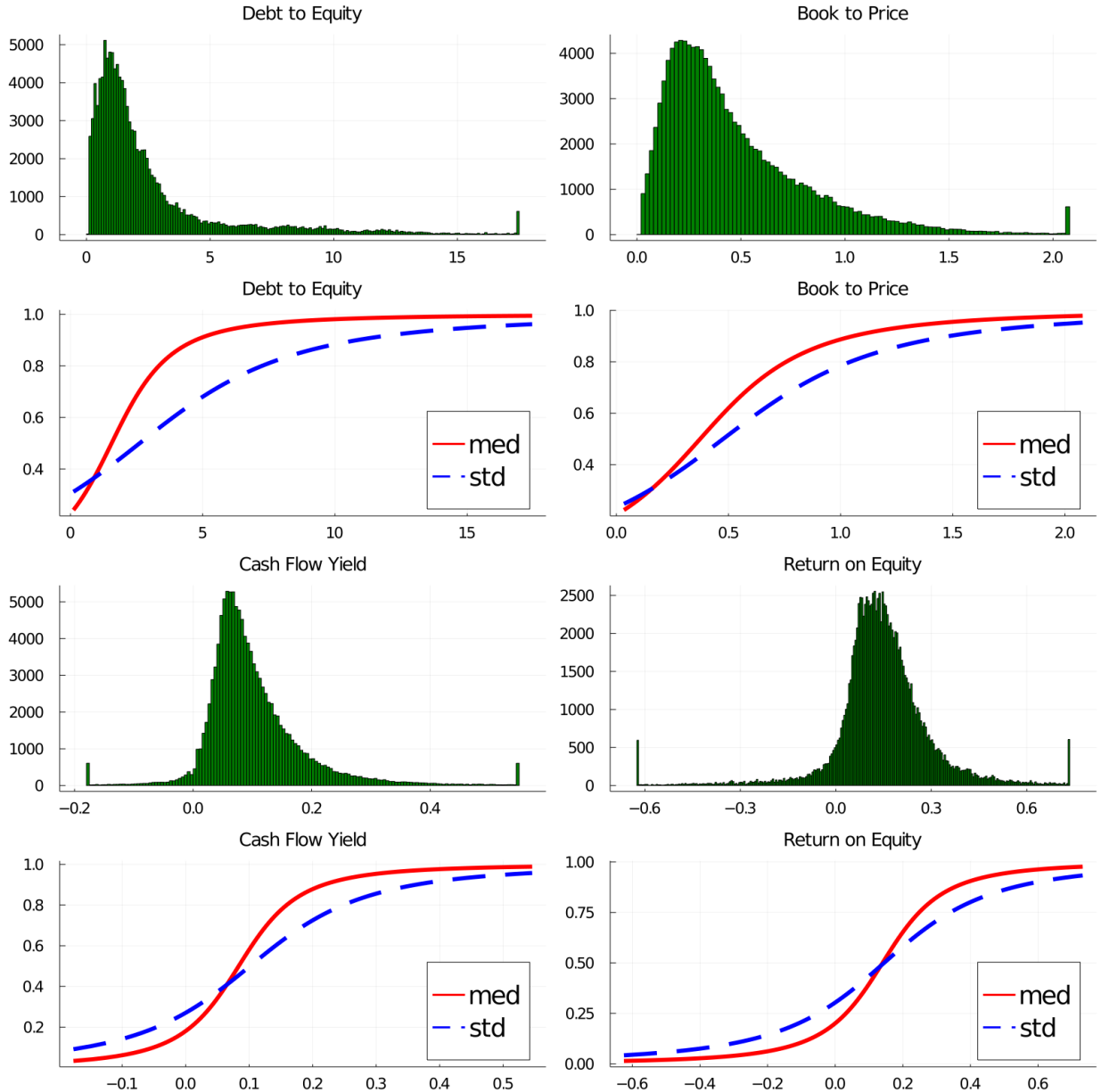
$$\log(\tau) \sim t(0, 1/d, 5), \tag{2}$$

where we scale each continuous feature as

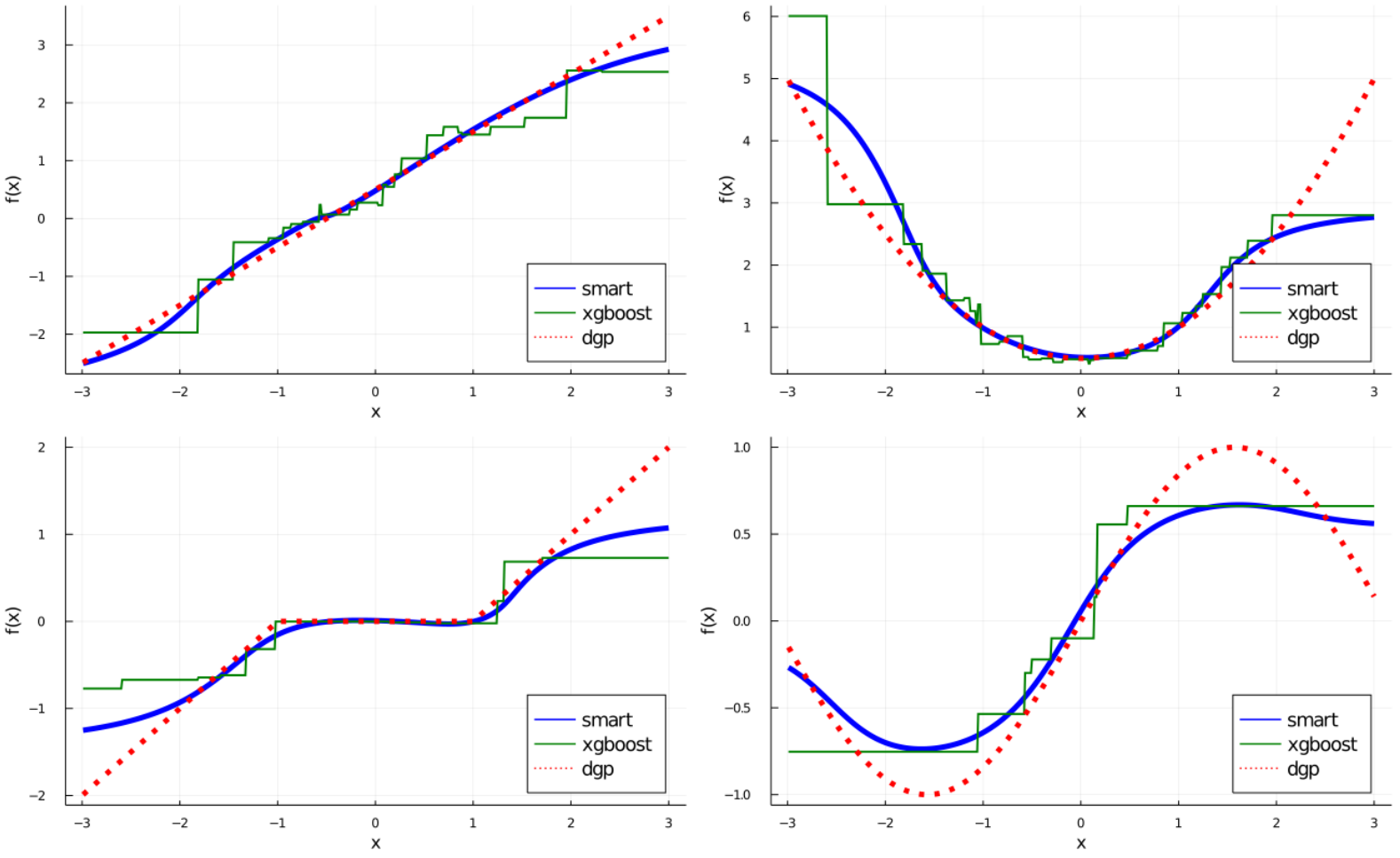
$$x \leftarrow \frac{x - median(x)}{1.25 \times mean\left(|x - median(x)|\right)}. \tag{3}$$

to suggest that near linear functions are less likely for highly skewed and leptokurtic features.

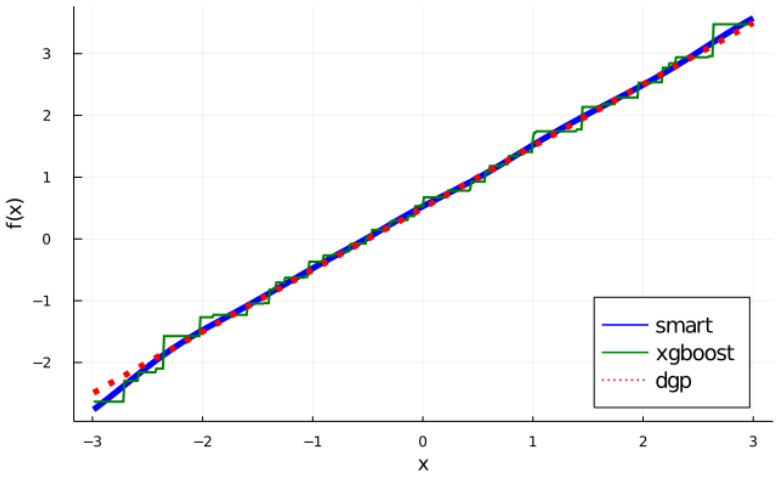
Linearity less likely a-priori if features are highly non-Gaussian

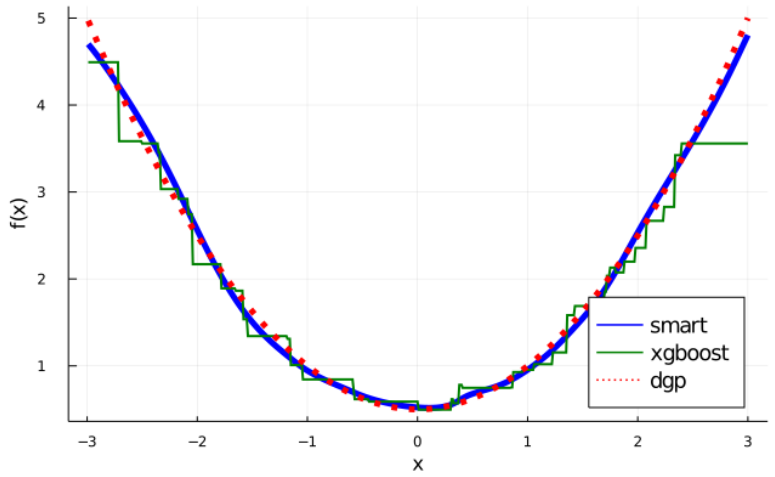


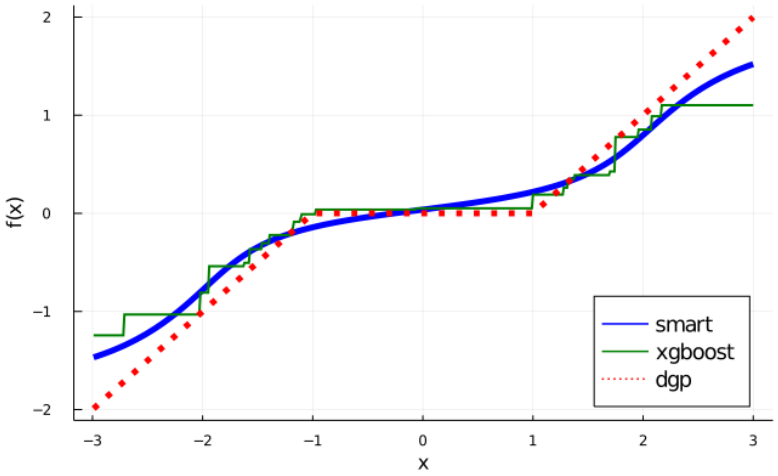
SMARTboost univariate examples

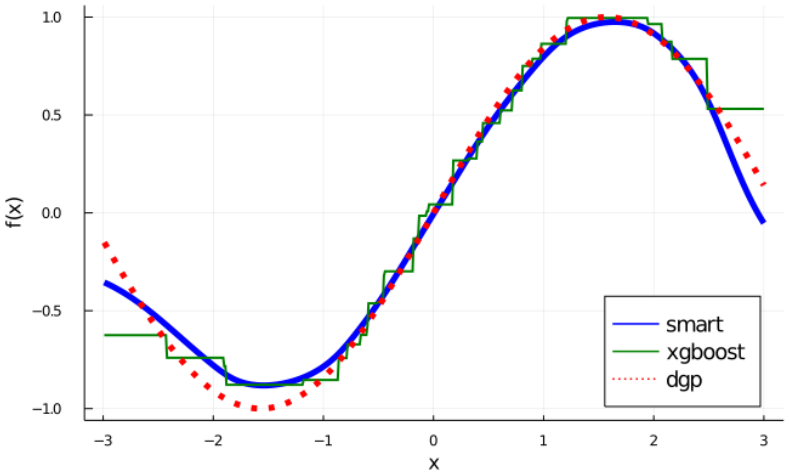


SMARTboost univariate examples

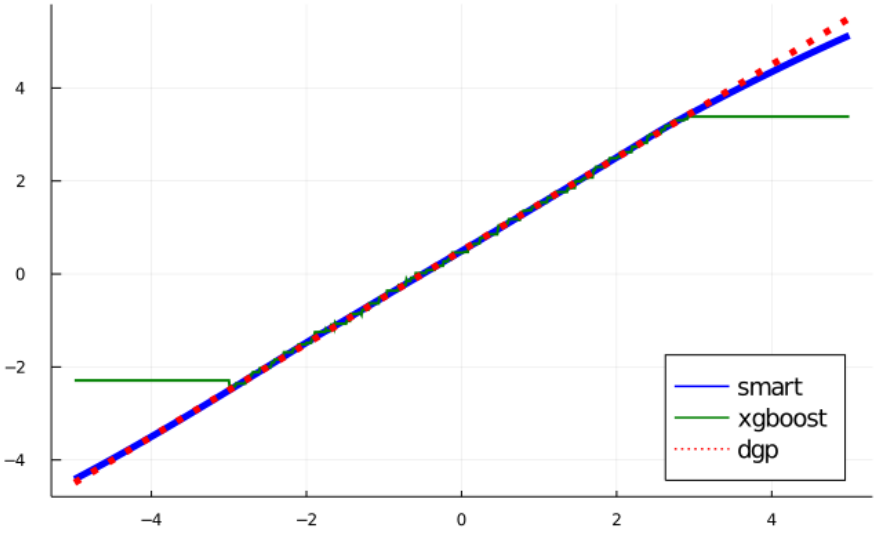


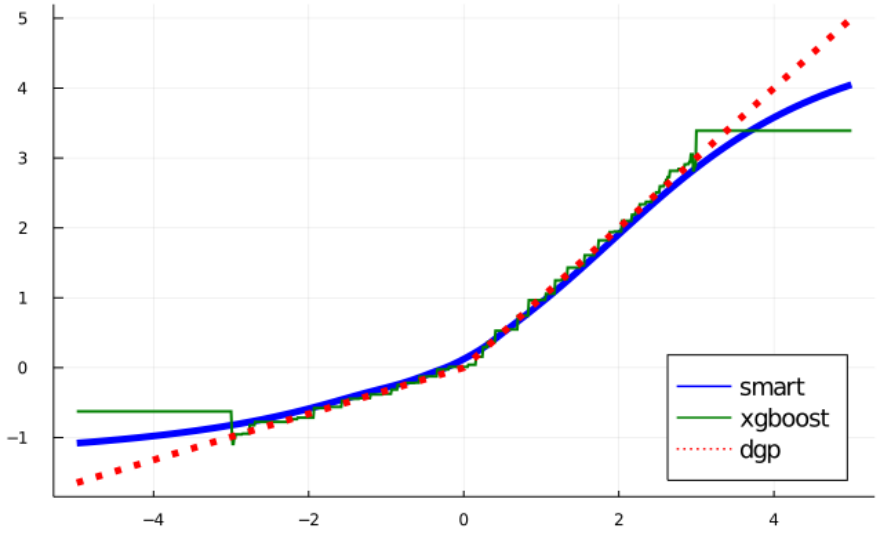


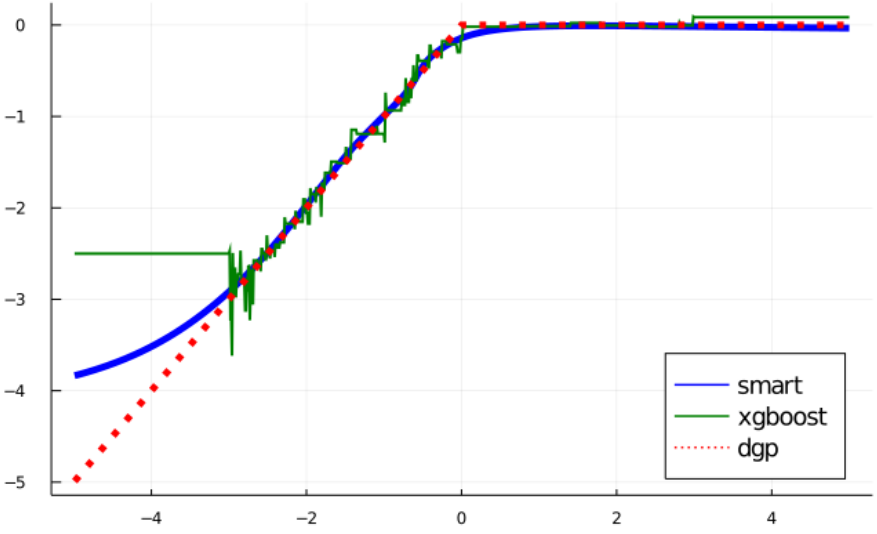


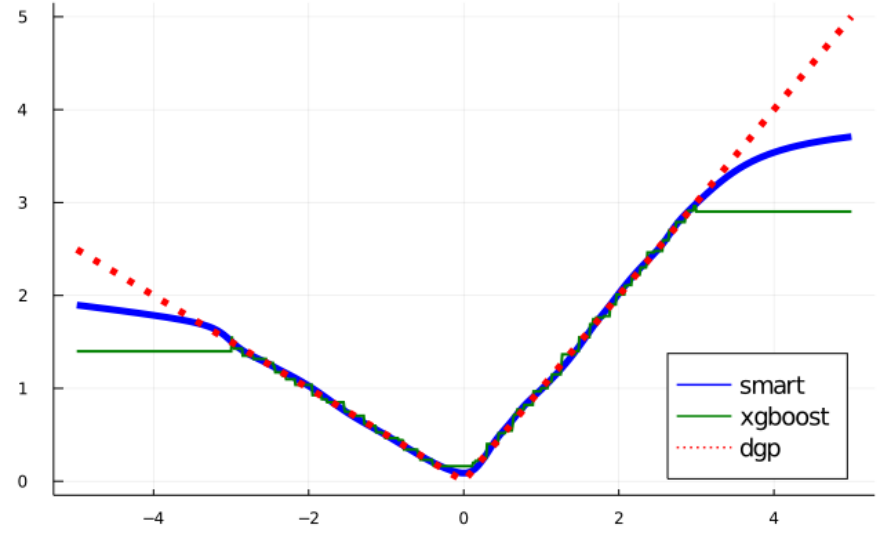


More intuitive extrapolation









Default CV settings assume a time series or panel structure

- The most common, default CV settings in ML are not appropriate for time series and panels.
- The default in SMARTboost is Purged CV (Purged CV of De Prado), adapted to panels.
- Cross-correlation and auto-correlation of panel data taken into account in setting default priors.

SMARTboost outperforms in a large number of settings familiar to econometrics and ML

Linear:

$$f(x) = 2x_1 + 1.5x_2 + x_3 + 0.5x_4,$$

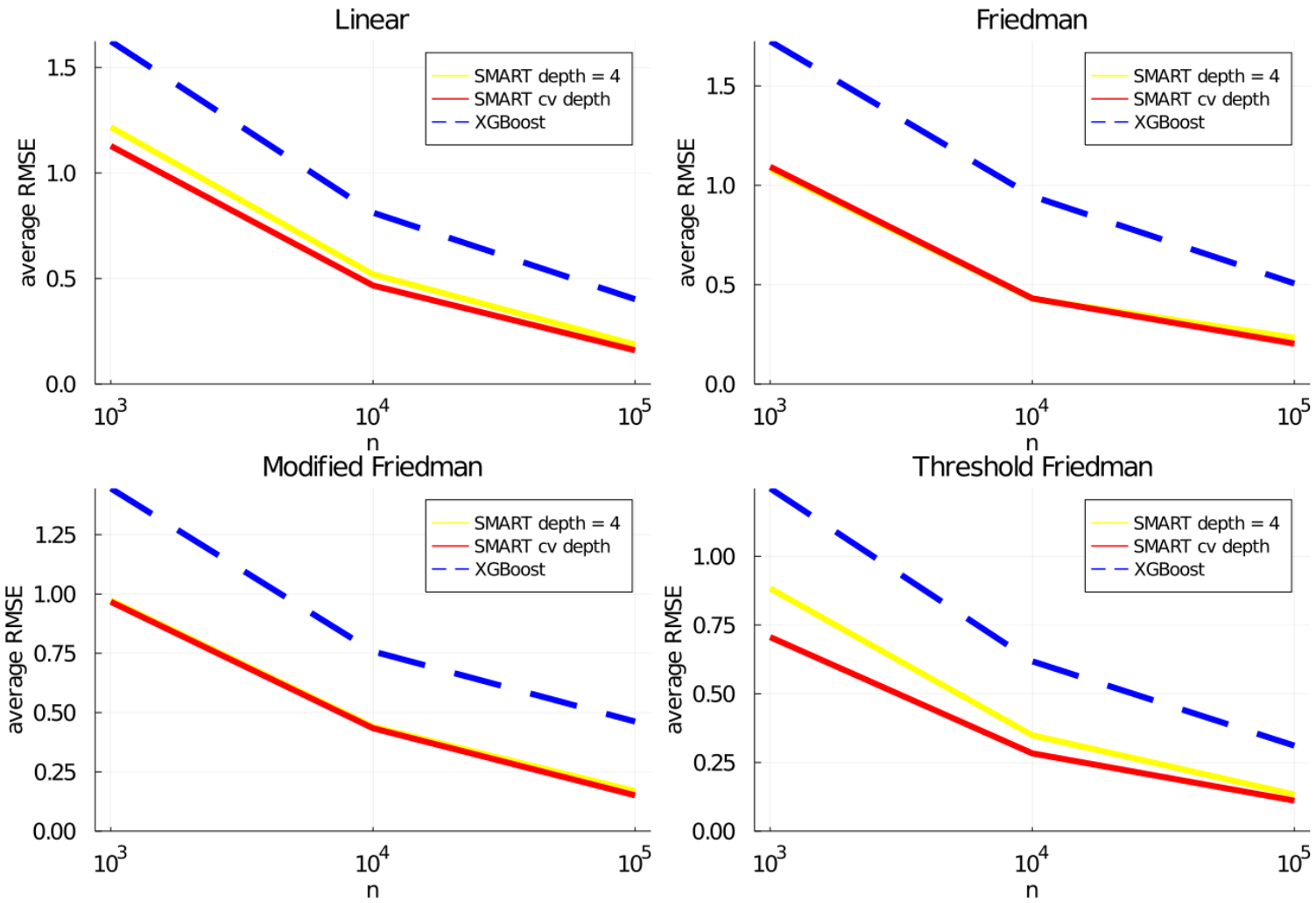
Friedman:

$$f(x) = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5,$$

Threshold Friedman:

$$f_1(x) = 10sin(\pi x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5$$
$$f_2(x) = 2sin(\pi x_2) + 10(x_3 - 0.5)^2 + 20x_4 + 10x_5$$
$$f(x) = f_1(x)I(x_1 < 0) + f_2(x)I(x_1 \geq 0).$$

SMARTboost outperforms in a large number of settings familiar to econometrics and ML



SMARTboost outperform in a large number of settings familiar to econometrics and ML

Linear and non-linear factor models:

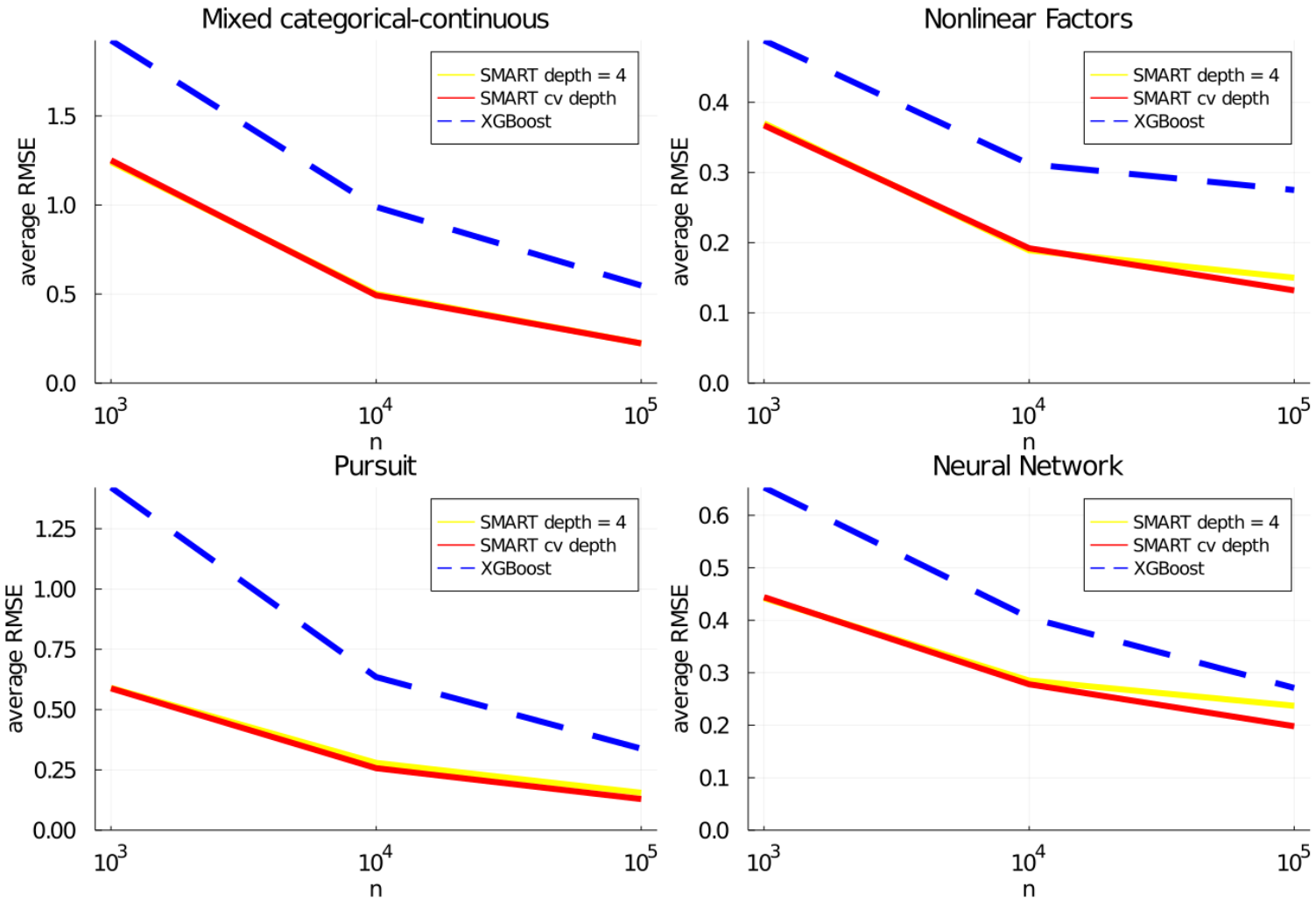
$$y \sim N\left(\overline{x}_1 + (\overline{x}_2 + 1)I(\overline{x}_2 < -1) + (\overline{x}_2 - 1)I(\overline{x}_2 > 1) + \overline{x}_3 I(\overline{x}_3 > 0), 1\right),$$

where $\overline{x}_i = mean(\mathbf{x_i})$, $\boldsymbol{x}_i$ is a ten-dimensional vector of multivariate N(0,1), and all cross-correlations 0.5.

Neural network:

$$\begin{aligned} z_1 = x_1 + x_2 + x_3,\ f_1 = z_1 I(z_1 > 0) \\ z_2 = x_4 + x_5 + x_6,\ f_2 = (z_2 - 1)I(z_2 > 1) \\ z_3 = x_7 + x_8 + x_9,\ f_3 = -z_3 I(z_3 > 0) \\ z_4 = 1.5 f_1 + f_2 + 0.5 f_3 \\ y \sim N\left(z_4 I(z_4 > 0), 1.0\right). \end{aligned}$$

SMARTboost outperform in a large number of settings familiar to econometrics and ML



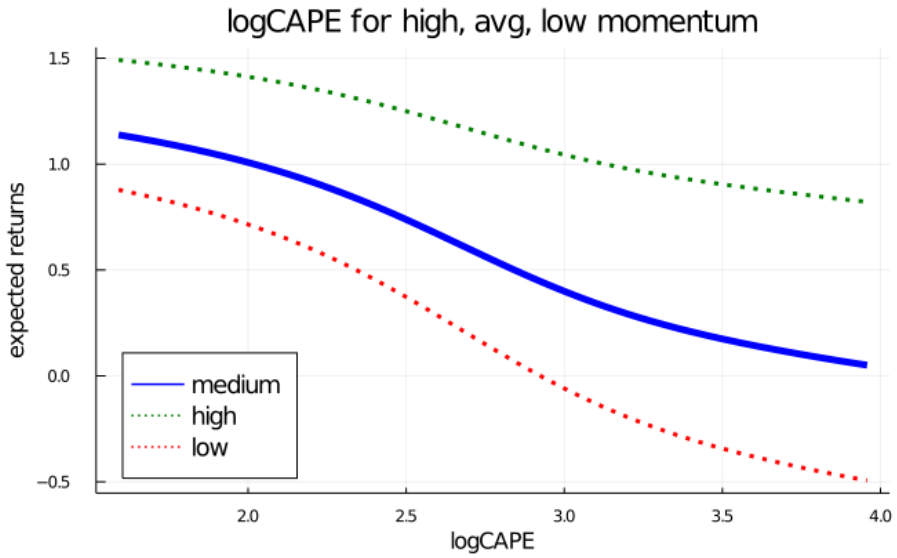
Global Equity Indexes

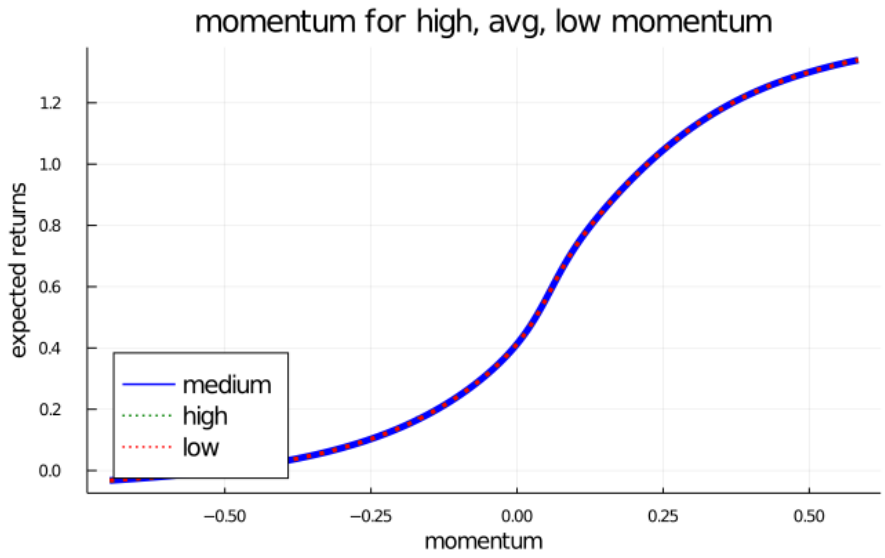
- Monthly total returns of global equity indexes. n = 13k.
- Low n, low SNR, few well-behaved variables: XGBoost does poorly, SMARTboost well, OLS can do well but is sensitive to data-transformation (showing OLS best results here).
- It's all in the interaction: SMARTboost performance drops if depth = 1 is forced.

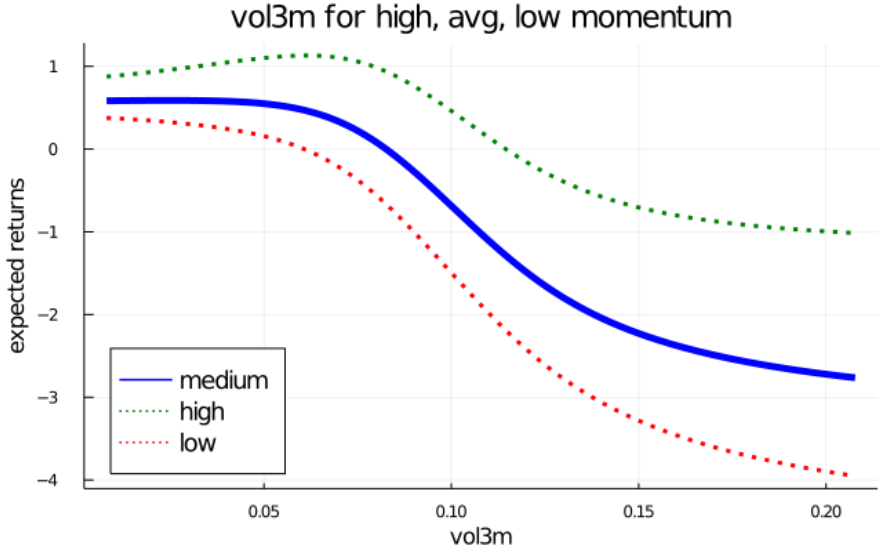
oos R^2 in %	log returns	returns
SMART depth = 4	2.16	1.86
SMART depth CV	2.15	1.57
XGBoost	0.84	0.21
OLS	1.16	0.48

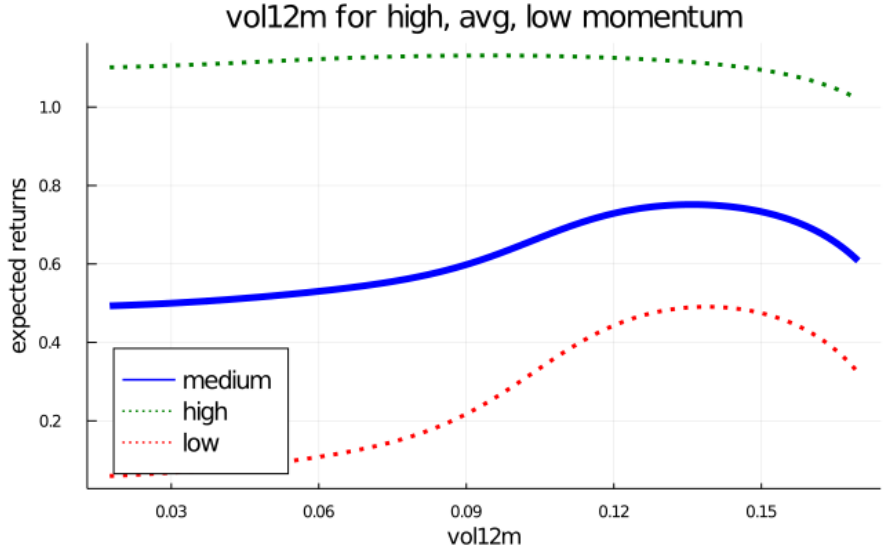
Table: Cross-validated out-of-sample R^2 in % on global equity indexes

Exploring interactions and the fragility of high valuations

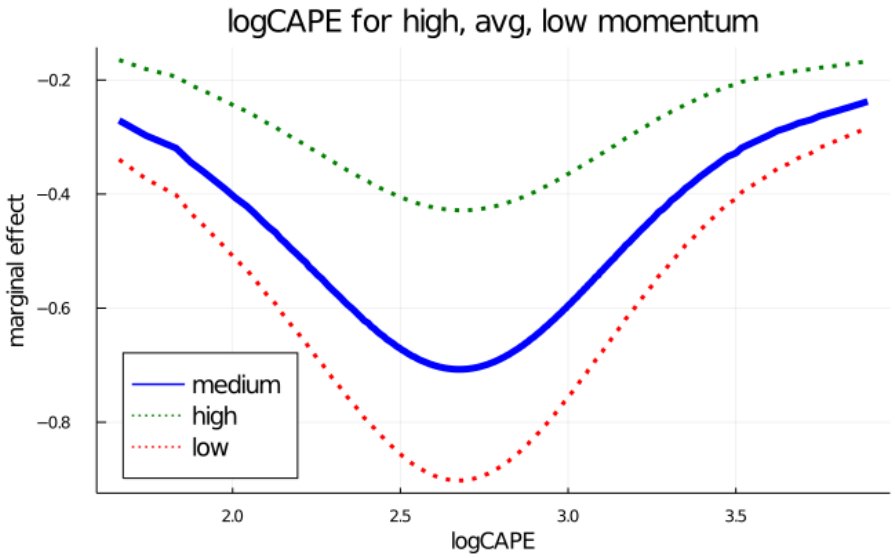


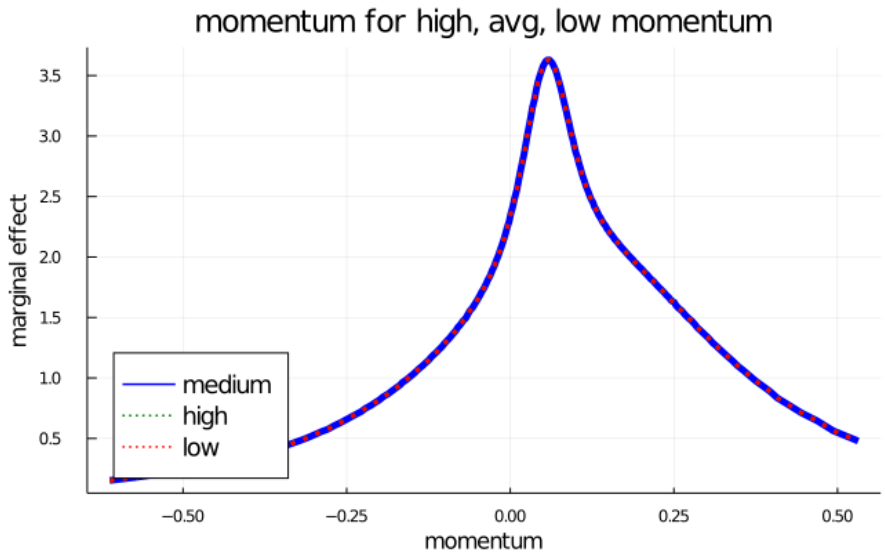


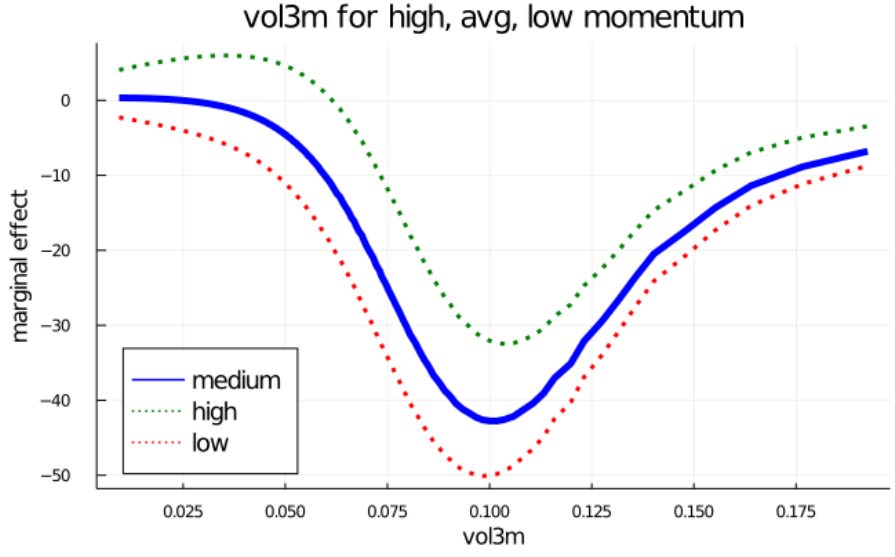


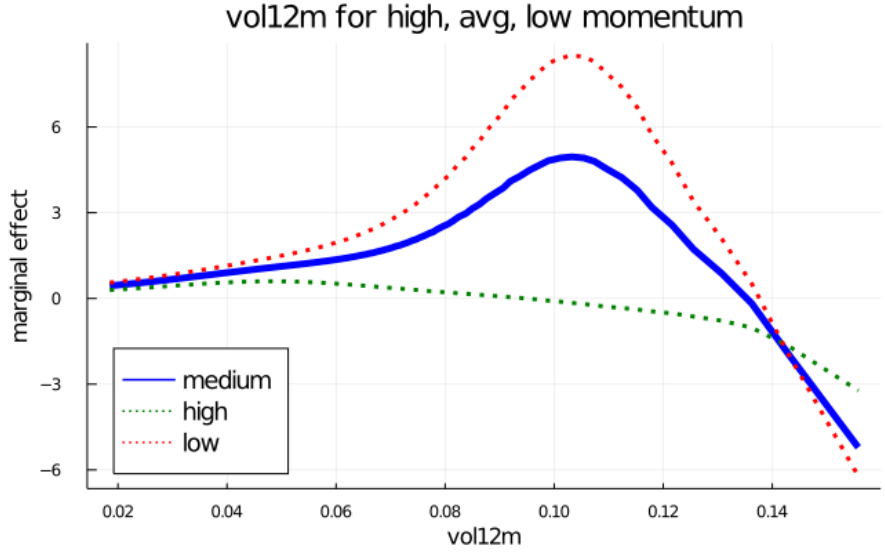


Marginal effects

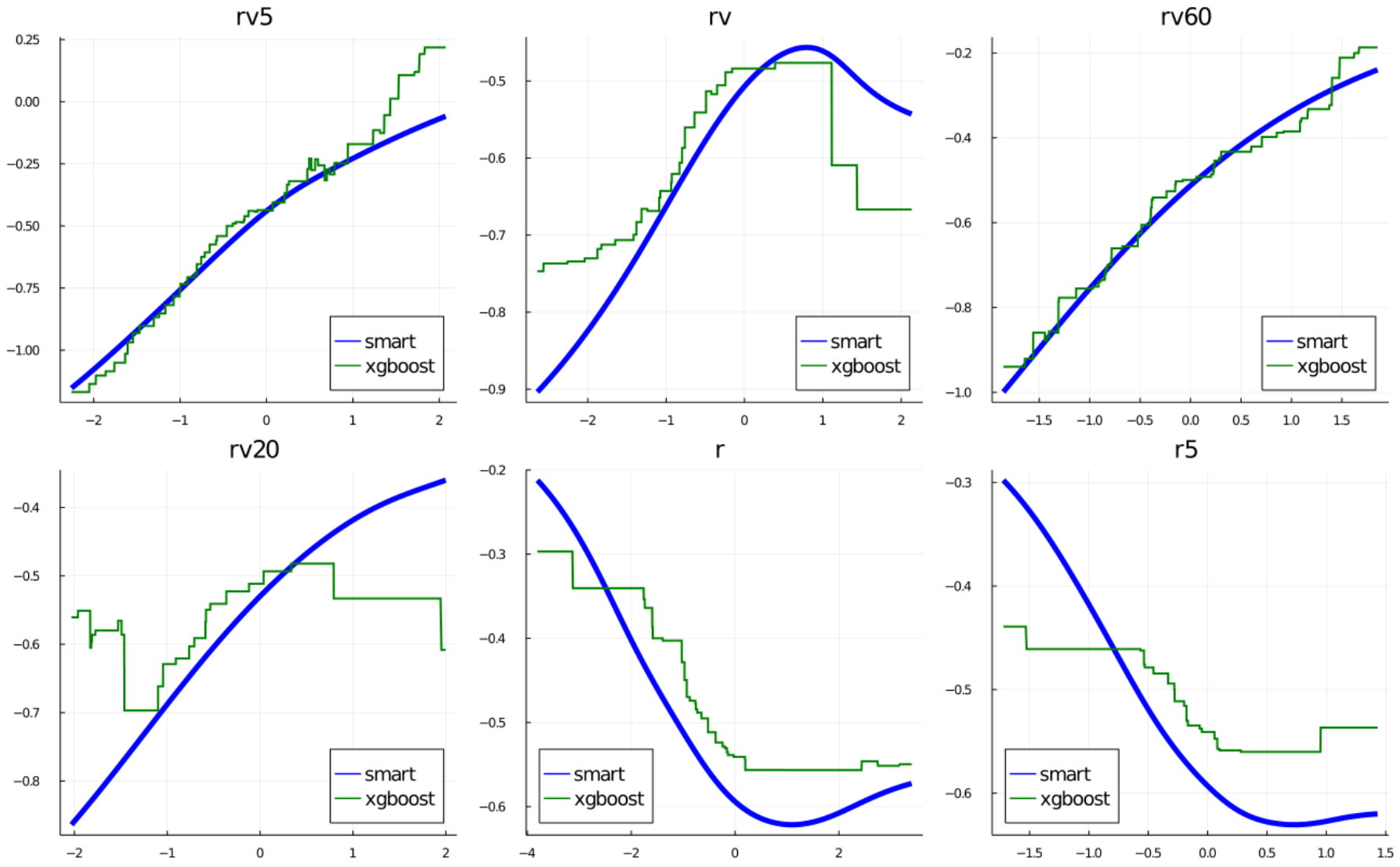








Realized Volatility in Equity Indexes

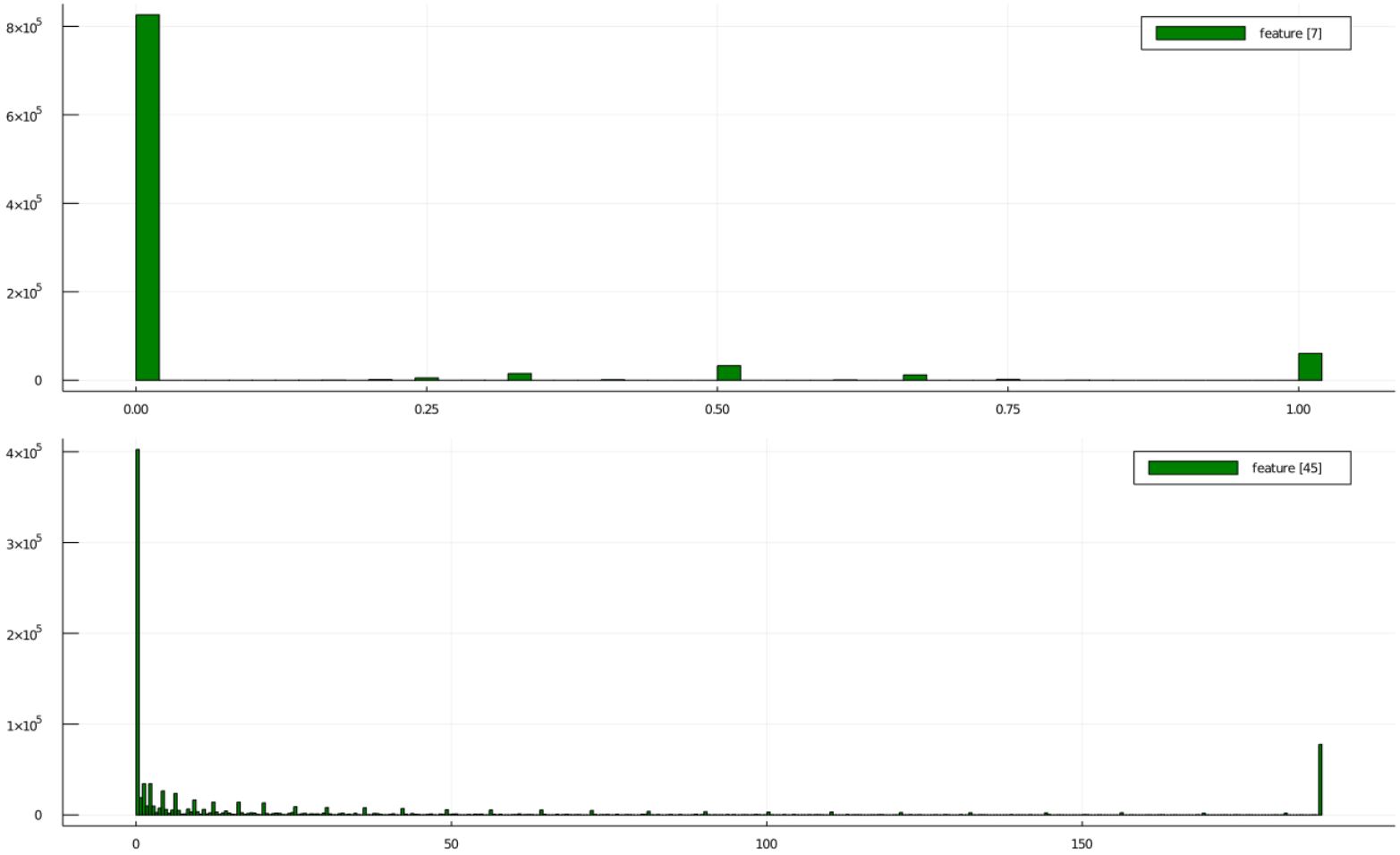


Some limitations of SMARTboost

- Computationally inefficient for sparse features.
- Truly large datasets will be challenging.
- In small samples, reasonably insensitive but not invariant to monotonic transformations of the features.

When is SMARTboost unlikely to beat other GBM?

Preliminary and vague but intuitive: when most relevant features have highly non-continuous distributions. Example from the Microsoft dataset:



What next?

- General log-likelihood/loss function.
- Improve priors to be able to handle highly non-standard features.
- ML applications, including large n and p: understand when SMARTboost is likely to perform well.