Ensemble Learning

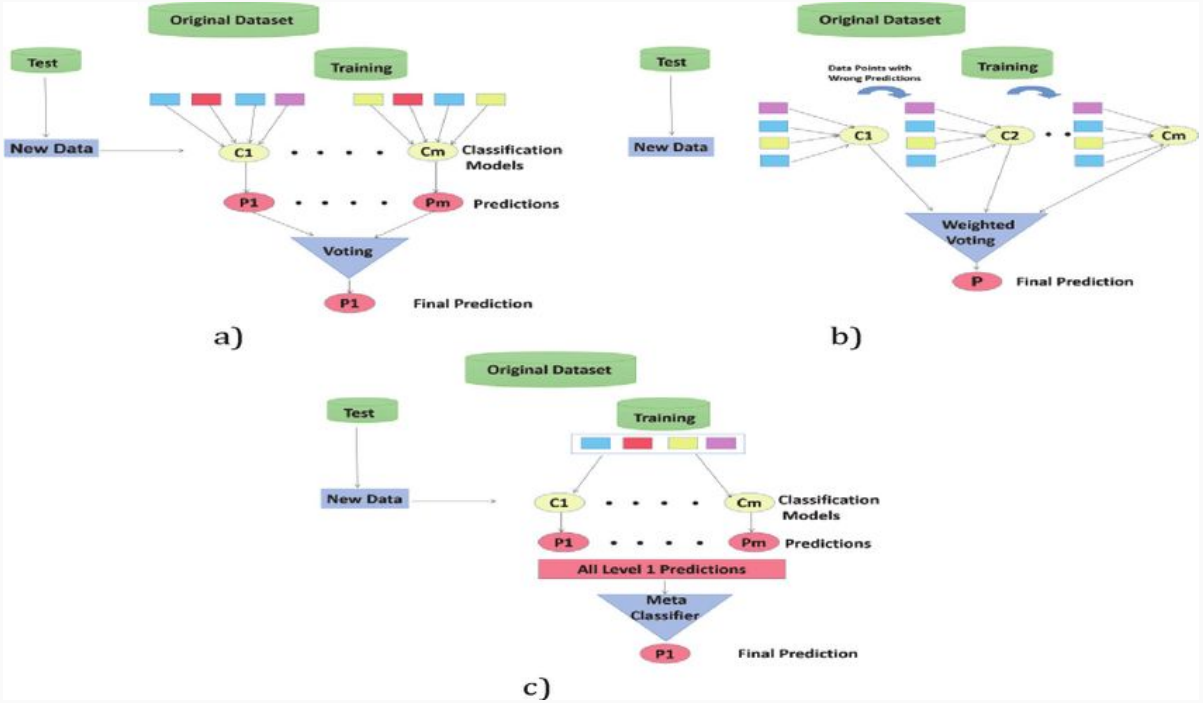
Jakub Kuciński

Introduction

The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models.

Main types of ensemble learning:

a) bagging
b) boosting
c) stacking



Early example

Multi-class classification using *error-correcting output codes.*

Digit	C_1	C_2	C_3	C_4	C_5	C_6	$\cdots$	C_{15}
0	1	1	0	0	0	0	$\cdots$	1
1	0	0	1	1	1	1	$\cdots$	0
2	1	0	0	1	0	0	$\cdots$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
8	1	1	0	1	0	1	$\cdots$	1
9	0	1	1	1	0	0	$\cdots$	0

1. Learn a separate classifier for each of the $L = 15$ two class problems defined by the columns of the coding matrix.
2. At a test point x, let $\hat{p}_\ell(x)$ be the predicted probability of a one for the ℓth response.
3. Define $\delta_k(x) = \sum_{\ell=1}^{L} |C_{k\ell} - \hat{p}_\ell(x)|$, the discriminant function for the kth class, where $C_{k\ell}$ is the entry for row k and column ℓ in Table 16.1.

Penalized Regression

J-terminal node regression trees $\mathcal{T} = \{T_k\}$

$$f(x) = \sum_{k=1}^{K} \alpha_k T_k(x)$$

Since the number of basis trees will be very large we need to add regularization.

$$\min_{\alpha} \left\{ \sum_{i=1}^{N} \left(y_i - \sum_{k=1}^{K} \alpha_k T_k(x_i) \right)^2 + \lambda \cdot J(\alpha) \right\}$$

$$J(\alpha) = \sum_{k=1}^{K} |\alpha_k|^2 \qquad \text{ridge regression,}$$

$$J(\alpha) = \sum_{k=1}^{K} |\alpha_k| \qquad \text{lasso,}$$

Forward Stagewise Linear Regression

Algorithm 16.1 *Forward Stagewise Linear Regression.*

1. Initialize $\check{\alpha}_k = 0,\ k = 1, \ldots, K$. Set $\varepsilon > 0$ to some small constant, and M large.

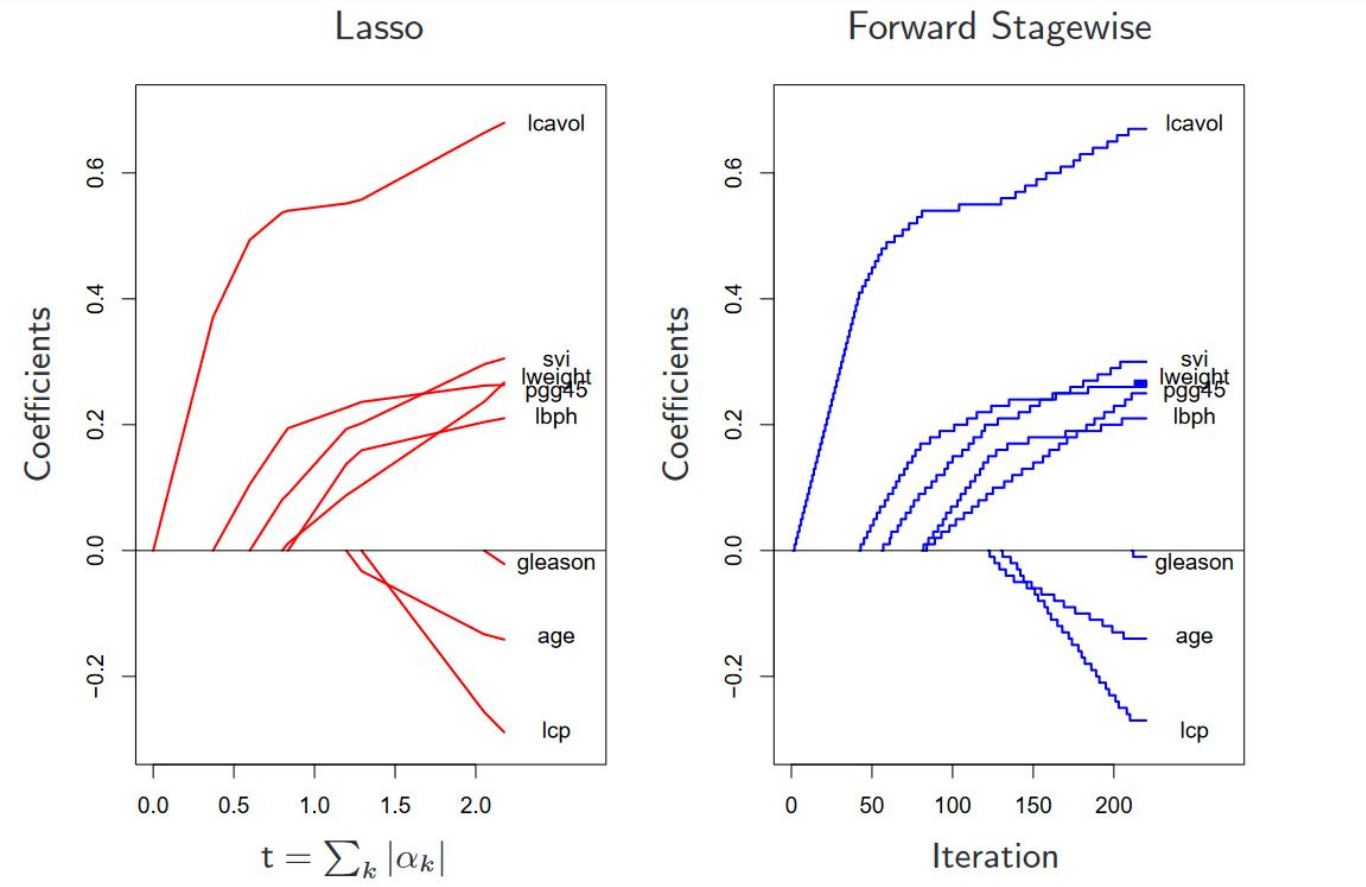
2. For $m = 1$ to M:

   (a) $(\beta^*, k^*) = \arg\min_{\beta,k} \sum_{i=1}^{N} \left(y_i - \sum_{l=1}^{K} \check{\alpha}_l T_l(x_i) - \beta T_k(x_i) \right)^2$.

   (b) $\check{\alpha}_{k^*} \leftarrow \check{\alpha}_{k^*} + \varepsilon \cdot \text{sign}(\beta^*)$.

3. Output $f_M(x) = \sum_{k=1}^{K} \check{\alpha}_k T_k(x)$.

Forward Stagewise Linear Regression



Forward Stagewise Linear Regression

- If basis functions are mutually uncorrelated or $\alpha_k(\lambda)$ in lasso are all monotone functions of λ then FSLR yields same solution as lasso.
- This is often the case when the correlation between the variables is low.
- When the $\alpha_k(\lambda)$ are not monotone in λ, then the solution sets are not identical.
- Coefficient paths are piece-wise linear functions, both for the lasso and forward stagewise hence they can be calculated with same cost as a single least-squares fit.
- Tree boosting with shrinkage closely resembles FSLR with the learning rate parameter ν corresponding to ε. Thus, one can view tree boosting with shrinkage as a form of monotone ill-posed regression on all possible (J-terminal node) trees, with the lasso penalty as a regularizer.

The “Bet on Sparsity” Principle

L2 norm is computionaly easier than L1. Why use L1 then?

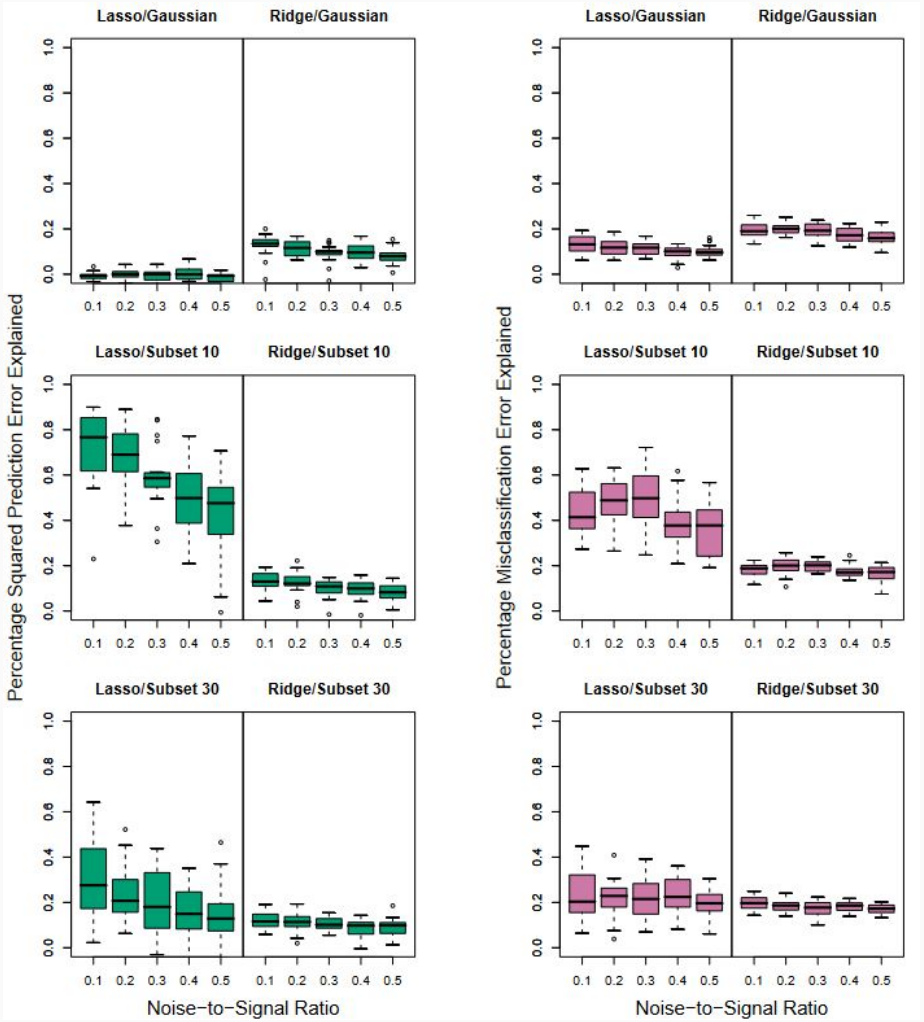
Example: Consider we have 10’000 points and our model is a linear combination of 1’000’000 trees.

- If small number (e.g. 1000) of trees’ coefficients should be nonzero then lasso will work better.
- If coefficients arose from a Gaussian distribution, then best predictor is ridge regression. However in this scenario neither method does very well since there is too little data from which to estimate such a large number of nonzero coefficient.

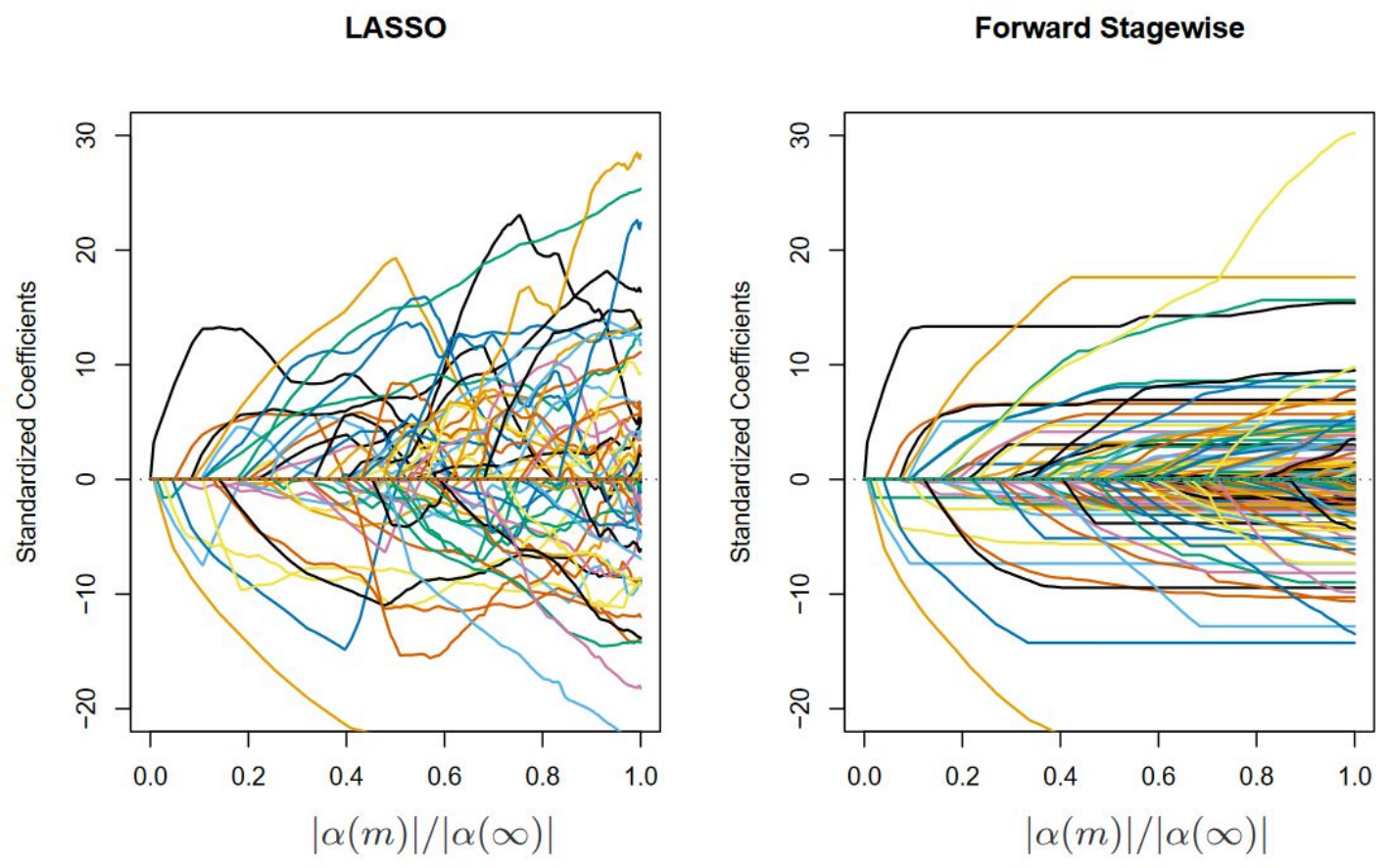
“Use a procedure that does well in sparse problems,
since no procedure does well in dense problems.”

The “Bet on Sparsity” Principle

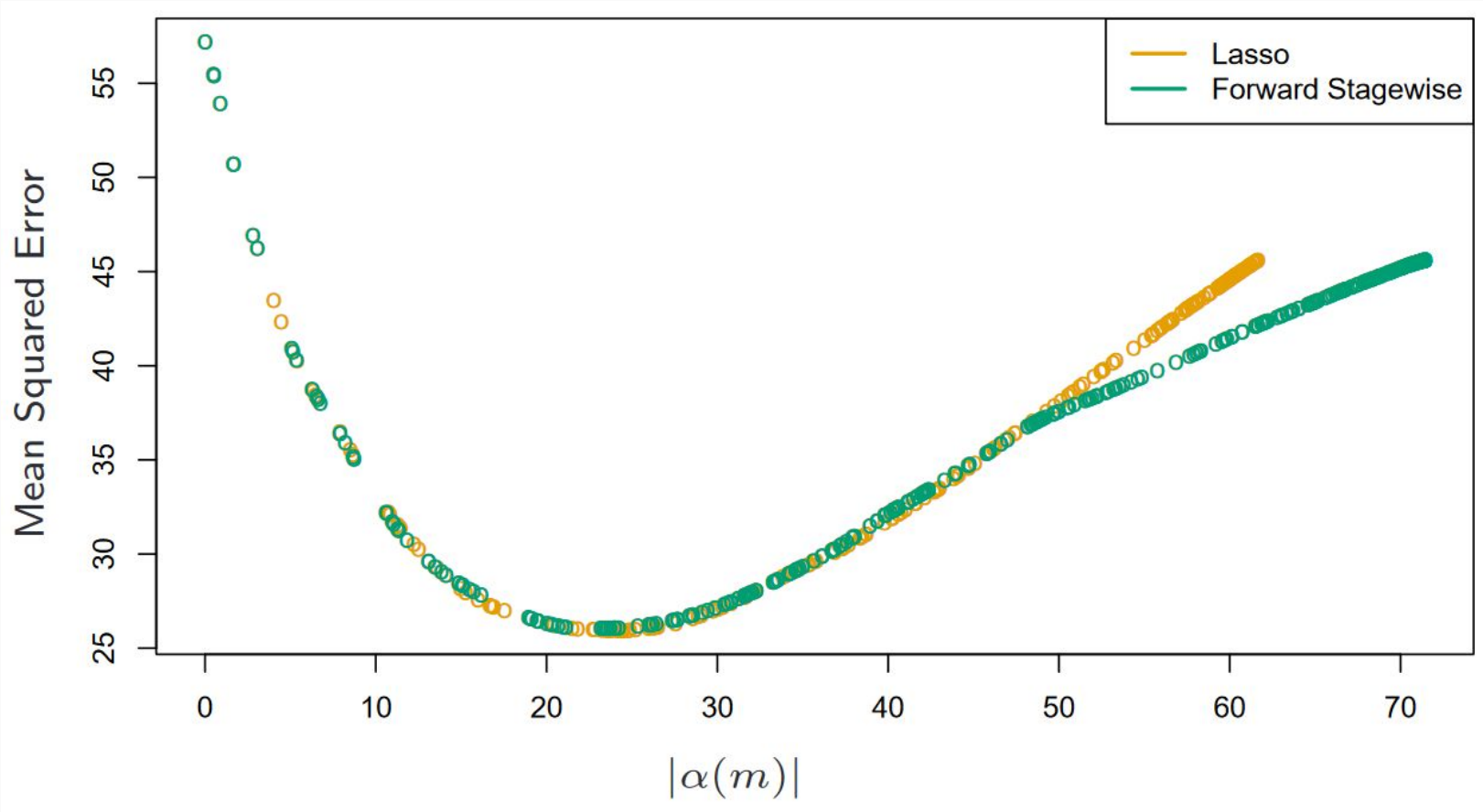
Simulations that show the superiority of the L1 (lasso) penalty over L2 (ridge) in regression and classification



Lasso and infinitesimal forward stagewise paths



Lasso and infinitesimal forward stagewise paths



Margin

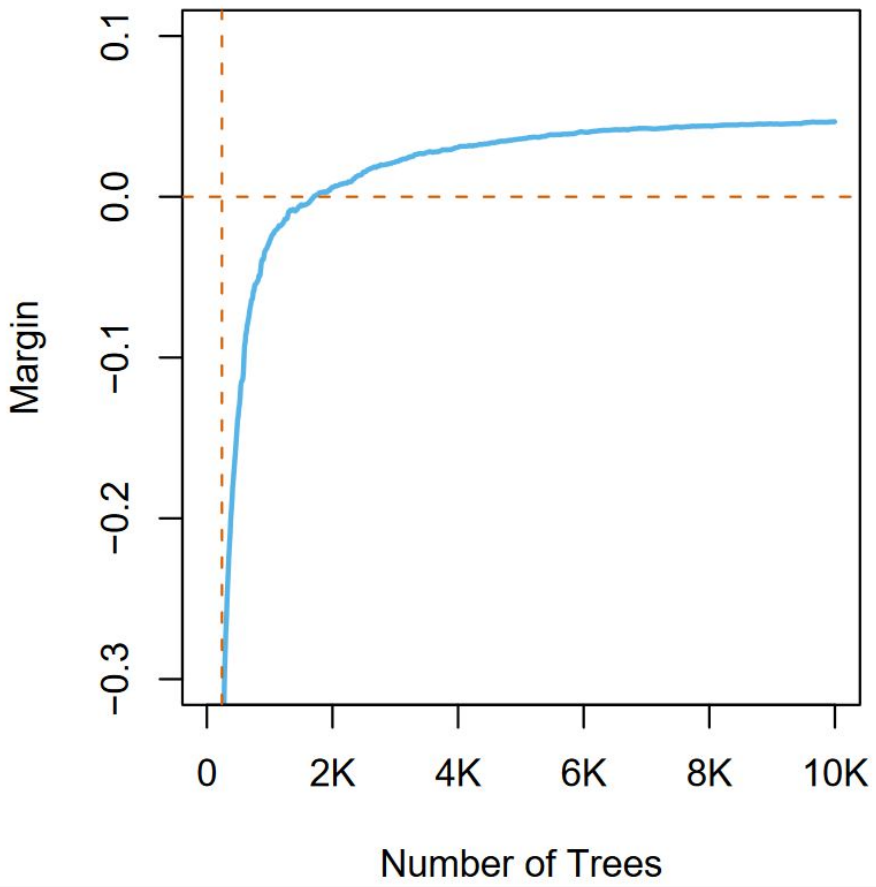
There have been suggestions that boosting performs well (for two-class classification) because it exhibits maximal-margin properties, much like the support-vector machines.

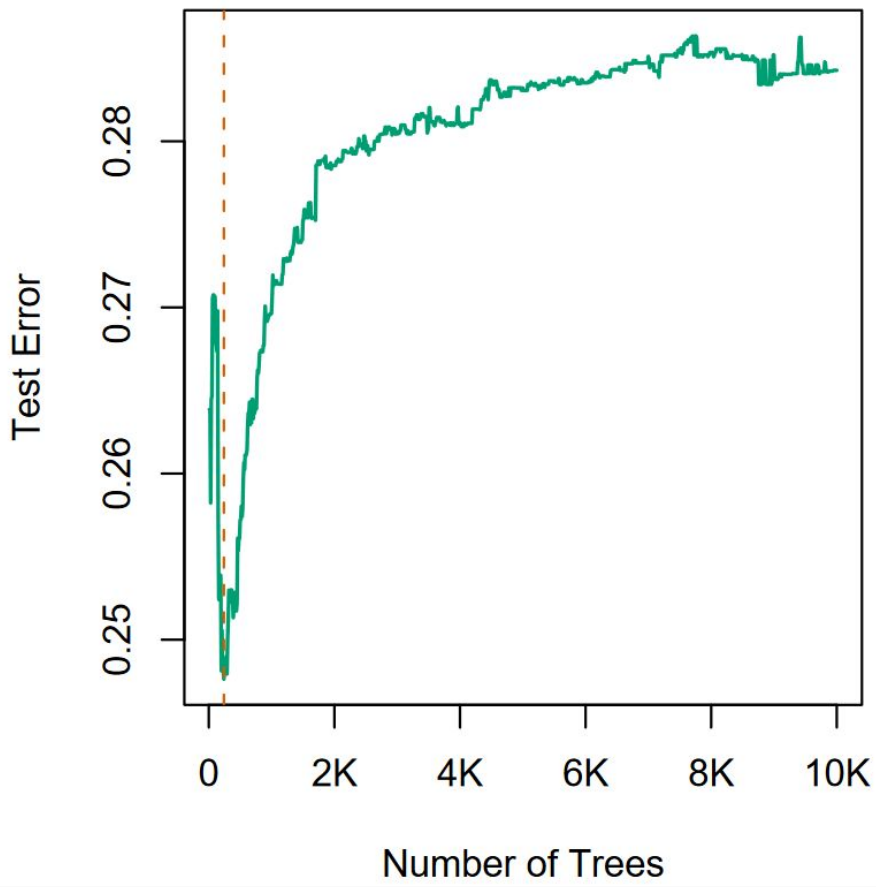
define the *normalized* L_1 margin of a fitted model $f(x) = \sum_k \alpha_k T_k(x)$ as

$$m(f) = \min_i \frac{y_i f(x_i)}{\sum_{k=1}^{K} |\alpha_k|}. \qquad (16.7)$$

where $y_i \in \{-1, +1\}$. L1 margin m(f) measures the distance to the closest training point in L_∞ units (maximum coordinate distance).

Margin


Margin
0.1
0.0
−0.1
−0.2
−0.3
0
2K
4K
6K
8K
10K
Number of Trees


Test Error
0.28
0.27
0.26
0.25
0
2K
4K
6K
8K
10K
Number of Trees

Margin

- Adaboost increases m(f) with each iteration, converging to a margin-symmetric solution.
- Adaboost with shrinkage converges asymptotically to a L1-margin-maximizing solution.
- As $\lambda \downarrow 0$, for particular loss functions the solution converges to a margin-maximizing configuration. In particular this is the case for the exponential loss of Adaboost and binomial deviance.

"The sequence of boosted classifiers form an L1-regularized monotone path to a margin-maximizing solution."

The margin-maximizing end of the path can be a very poor, overfit solution. One should use early stopping with validation set to get best performing solution*.

*Sometimes model generalizes to data much later then the validation dataset loss starts raising (e.g. with heavily overparametriezed models like NN https://mathai-iclr.github.io/papers/papers/MATHAI_29_paper.pdf, https://arxiv.org/abs/2108.12284)

Learning Ensembles

Again we consider functions of the form: $$f(x) = \alpha_0 + \sum_{T_k \in \mathcal{T}} \alpha_k T_k(x)$$

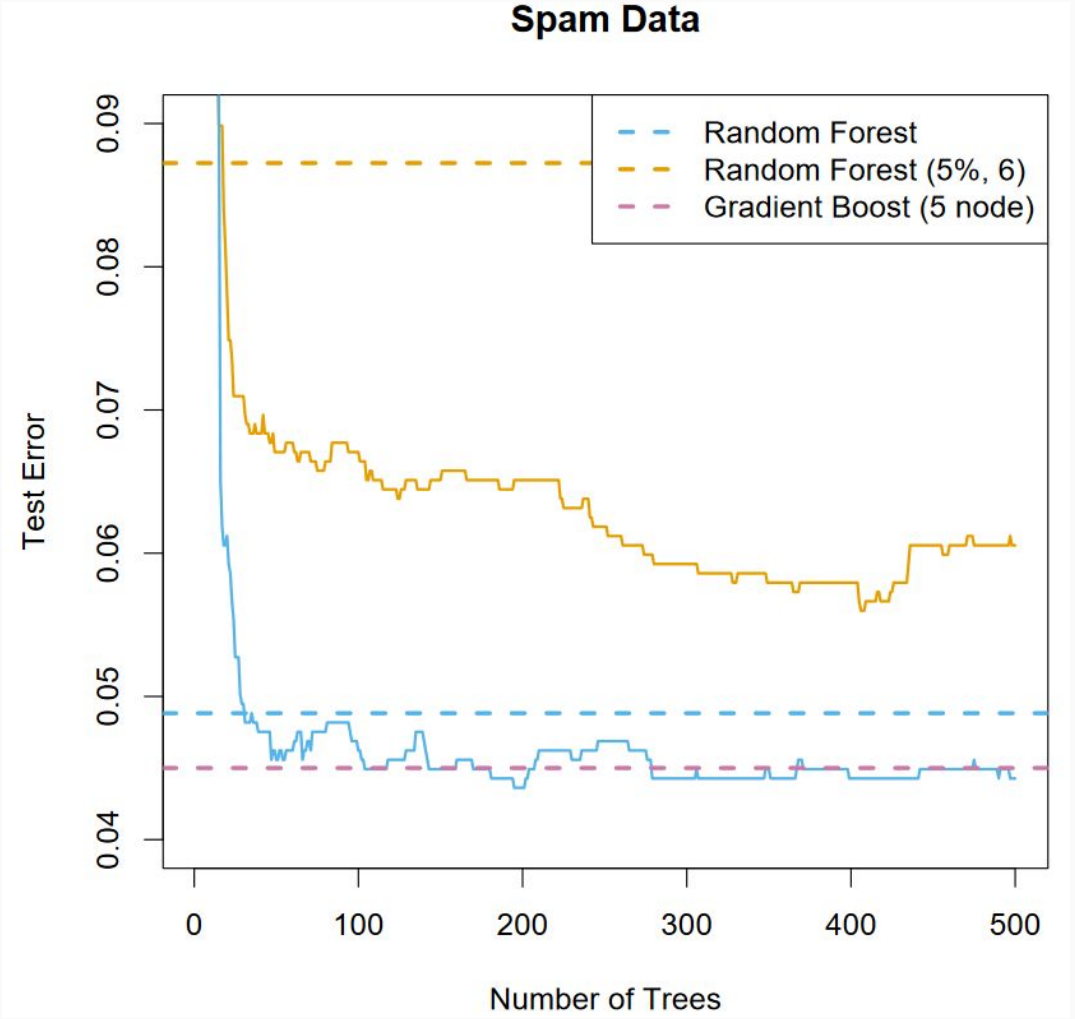
- A finite dictionary $\mathcal{T}_L = \{T_1(x), T_2(x), \ldots, T_M(x)\}$ of basis functions is induced from the training data;
- A family of functions $f_\lambda(x)$ is built by fitting a lasso path in this dictionary:

$$\alpha(\lambda) = \arg\min_{\alpha} \sum_{i=1}^{N} L[y_i, \alpha_0 + \sum_{m=1}^{M} \alpha_m T_m(x_i)] + \lambda \sum_{m=1}^{M} |\alpha_m|. \quad (16.9)$$

Learning Ensembles

Horizontal lines represents the test errors of the baseline models.

The orange and blue curves are the test errors after post-processing baseline models.



Learning a Good Ensemble

For the post-processor to be effective we want basis functions that covers the space well in places where they are needed and are sufficiently different from each other.

We want to find a set of M evaluation points $\gamma_m \in \Gamma$ and corresponding weights α_m so that $f_M(x) = \alpha_0 + \sum_{m=1}^{M} \alpha_m b(x; \gamma_m)$ approximates $f(x)$ well over the domain of x where $\gamma \in \Gamma$ indexes the basis functions $b(x; \gamma)$.

We want to introduce randomness in the selection of γ to make them more diverse, but give more weight to relevant regions of the space Γ.

Importance sampled learning ensemble

Algorithm 16.2 *ISLE Ensemble Generation.*

1. $f_0(x) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$
2. For $m = 1$ to M do
   (a) $\gamma_m = \arg\min_\gamma \sum_{i \in S_m(\eta)} L(y_i, f_{m-1}(x_i) + b(x_i; \gamma))$
   (b) $f_m(x) = f_{m-1}(x) + \nu b(x; \gamma_m)$
3. $\mathcal{T}_{ISLE} = \{b(x; \gamma_1), b(x; \gamma_2), \ldots, b(x; \gamma_M)\}$.

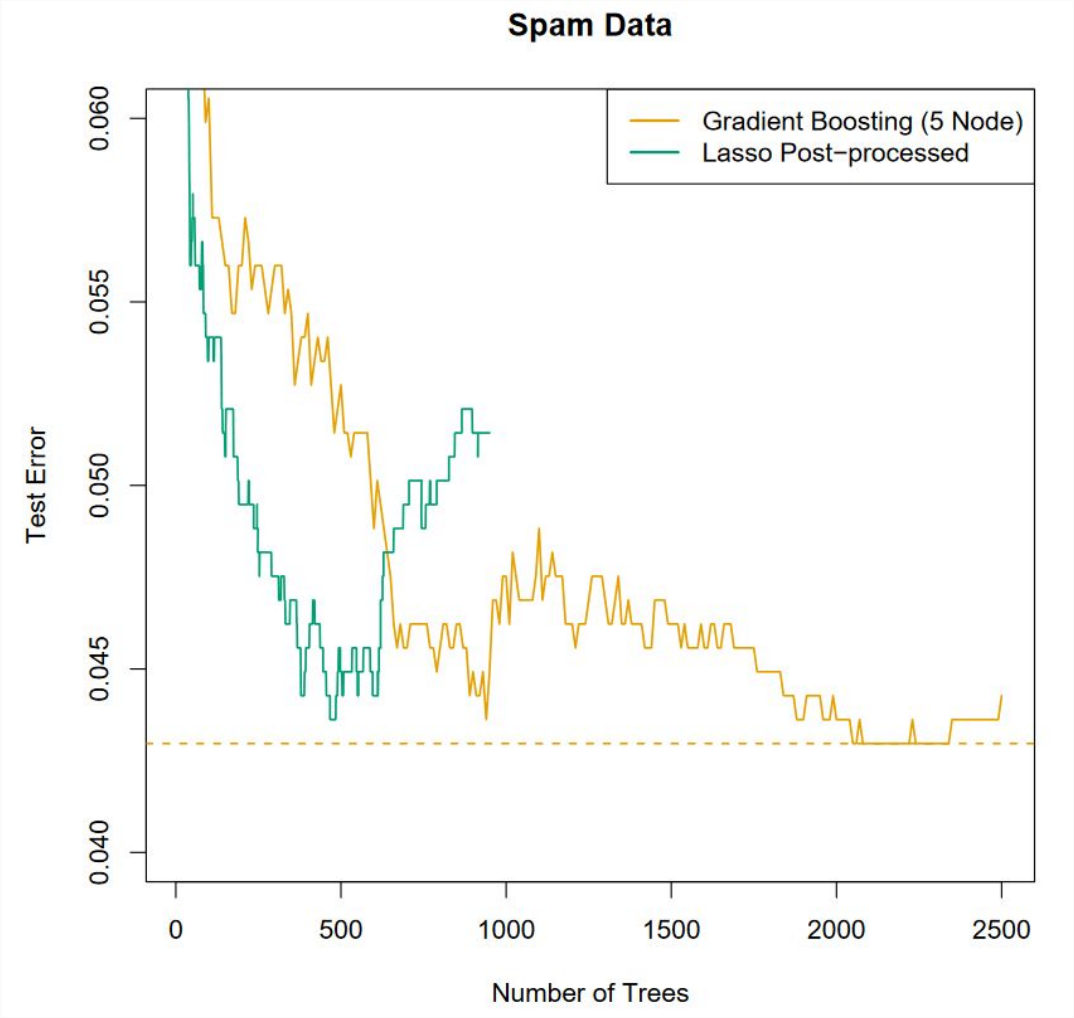
$S_m(\eta)$ refers to a subsample of $N \cdot \eta$ ($\eta \in (0, 1]$) of the training observations.
On the obtained new set of basis functions we can perform lasso post-processing.

Special cases of ISLE

- Bagging has $\eta = 1$, but samples with replacement, and has $v = 0$.
- Random forest sampling is similar, with more randomness introduced by the selection of the splitting variable. Reducing $\eta < 1/2$ in ISLE has a similar effect to reducing m in random forests.
- Gradient boosting with shrinkage uses $\eta = 1$, but typically does not produce sufficient spread of basis functions.
- Stochastic gradient boosting follows the recipe exactly.

Importance sampled learning ensemble

Gradient boosting model trained with $\eta = 1/2$, $\nu = 0.05$ and trees with five terminal nodes.



Importance sampled learning ensemble

GBM (0.1, 0.01) refers to a gradient boosted model, with parameters (η, ν), RF to random forest.