Introduction to Machine Learning (IML)

LECTURE #10: BAGGING AND BOOSTING

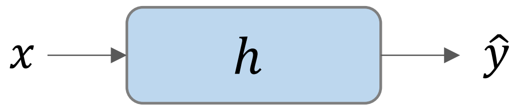
236756 – 2023-2024 WINTER – TECHNION

LECTURER: YONATAN BELINKOV

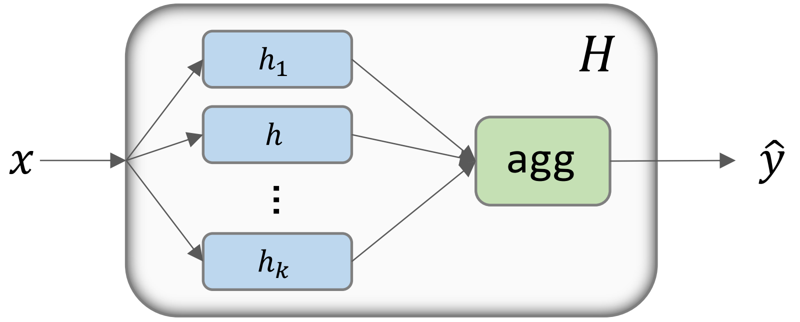
Today

- **Part III**: *more supervised learning*
  1. Regression
  2. Bagging and boosting (today)
  3. Generative models
  4. Deep learning
- Back to a discriminative frame-of-thought
- Will consider classification and regression interchangeably (i.e., use a different loss)

Ensemble methods


$$\begin{tikzcd} x \arrow[r] & \boxed{h} \arrow[r] & \hat{y} \end{tikzcd}$$

Ensemble (aka committee)



- **Ensemble methods:**

  aggregate multiple models into a single, (hopefully) more powerful model

- **Q**: are ensembles more powerful than any individual models?
- **A**: it's complicated
- Today's goal is to shed light on this question
- **Intuitively, ensembling works well when:**
  - Models differ in strengths and weaknesses (e.g., accurate on different parts of D)
  - Aggregation amplifies strengths and mitigates weaknesses
  - Optimization is likely to work reasonably well

Ensemble methods

- Template (and today):

1. Base-class learner

ID3 (decision trees)

2. Aggregation rule

linear: $H = \sum_i \alpha_i h_i$

3. Meta-learner

- *Bagging*
- *Boosting*

- **Base model class**: $h \in \mathcal{H}$ (classification, regression, …)
- **Base-class learner** (=algorithm): $h_S = A(S)$
- **Linear ensemble model**: $H(x) = \sum_i \alpha_i h_i(x), \quad \alpha_i \in \mathbb{R}, h_i \in \mathcal{H}$
- **Meta-learner** (=algorithm): $H_S = M(S)$ (learns α_i, h_i)

Ensemble methods

- Template (and today):

1. Base-class learner

ID3 (decision trees)

2. Aggregation rule

linear: $H = \sum_i \alpha_i h_i$

3. Meta-learner

- *Bagging*
- *Boosting*

regression:

$$\boxed{H(x) = \frac{1}{k}\sum_{i=1}^{k} h_i(x)}$$

will write this…

classification:

$$H(x) = \text{majority}(\{h_i(x)\})$$
$$= \boxed{\text{sign}\left(\frac{1}{k}\sum_{i=1}^{k} \text{sign}\big(h_i(x)\big)\right)}$$

…to also mean this

Ensemble methods

- Template (and today):

1. Base-class learner

ID3 (decision trees)

2. Aggregation rule

linear: $H = \sum_i \alpha_i h_i$

3. Meta-learner

- *Bagging*
- *Boosting*

- **Q**: Does it make sense to aggregate linear models?
- **A**: Question is ill-posed
  - Need to also say what base learner is (e.g., ERM, RLM, SVM)
  - For linear classes, ensembles H are also linear! so member of the base class
  - But this does <u>not</u> imply that learning H is the same as learning h
- We will return to this!

Bagging

Bagging

- **Recall**: bias-variance tradeoff

$$\underbrace{\mathbb{E}_{S\sim D^m}\left[L_D^{sqr}(h_S)\right]}_{\textit{expected error}} = \underbrace{\mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]}_{\textit{noise}} + \underbrace{\mathbb{E}_x\left[\left(\bar{h}(x) - \bar{y}(x)\right)^2\right]}_{\textit{bias}^2} + \underbrace{\mathbb{E}_{S,x}\left[\left(h_S(x) - \bar{h}(x)\right)^2\right]}_{\textit{variance}}$$

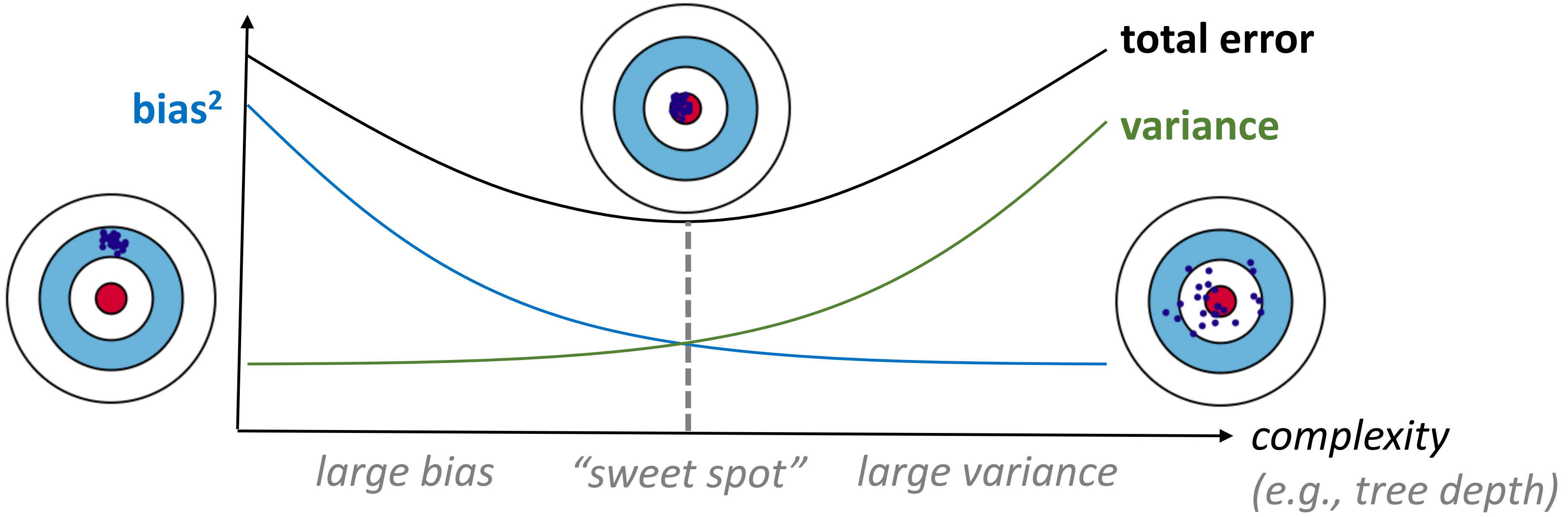
- where:
  - Expected label: $\bar{y}(x) = \mathbb{E}_{y\sim D_{Y|X=x}}[y] \in [0,1]$
  - Expected classifier: $\bar{h} = \mathbb{E}_{S\sim D^m}[h_S]$

Bagging

- **Recall**: bias-variance tradeoff

$$\mathbb{E}_{S\sim D^m}\left[L_D^{sqr}(h_S)\right] = \mathbb{E}_{x,y}\left[(\bar{y}(x)-y)^2\right] + \mathbb{E}_x\left[\left(\bar{h}(x)-\bar{y}(x)\right)^2\right] + \mathbb{E}_{S,x}\left[\left(h_S(x)-\bar{h}(x)\right)^2\right]$$

expected error ***noise*** ***bias²*** ***variance***

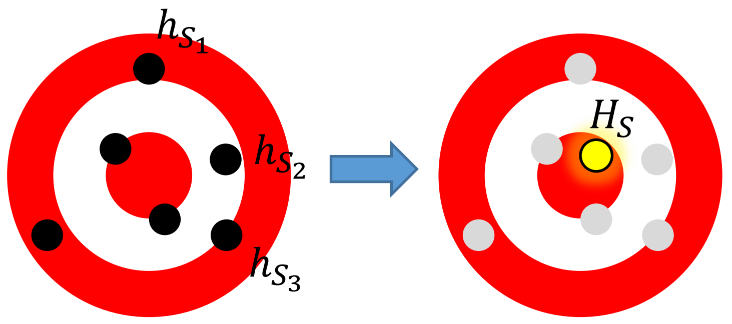


Bagging

- **Recall**: bias-variance tradeoff

$$\underbrace{\mathbb{E}_{S\sim D^m}\left[L_D^{sqr}(h_S)\right]}_{\textit{expected error}} = \underbrace{\mathbb{E}_{x,y}\left[(\bar{y}(x)-y)^2\right]}_{\textit{noise}} + \underbrace{\mathbb{E}_x\left[\left(\bar{h}(x)-\bar{y}(x)\right)^2\right]}_{\textit{bias}^2} + \boxed{\underbrace{\mathbb{E}_{S,x}\left[\left(\overset{H_S}{\cancel{h_S}}(x)-\bar{h}(x)\right)^2\right]}_{\textit{variance}}}$$

- Bagging is an ensembling approach aimed at **reducing variance** *without decreasing complexity*
- **Idea**: learn an H_S that tries to "approximate" $\bar{h}$
- Bagging works well for *high-variance, low-bias* base model classes
- For example, **deep decision trees**

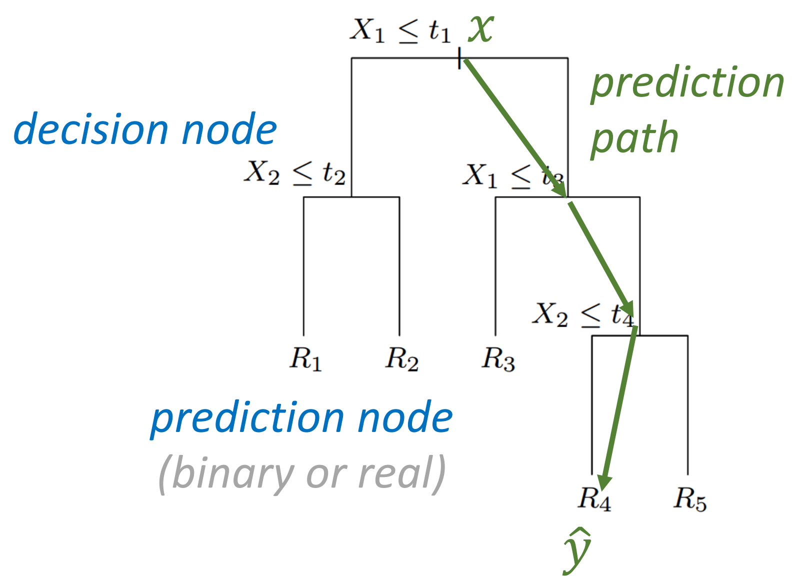


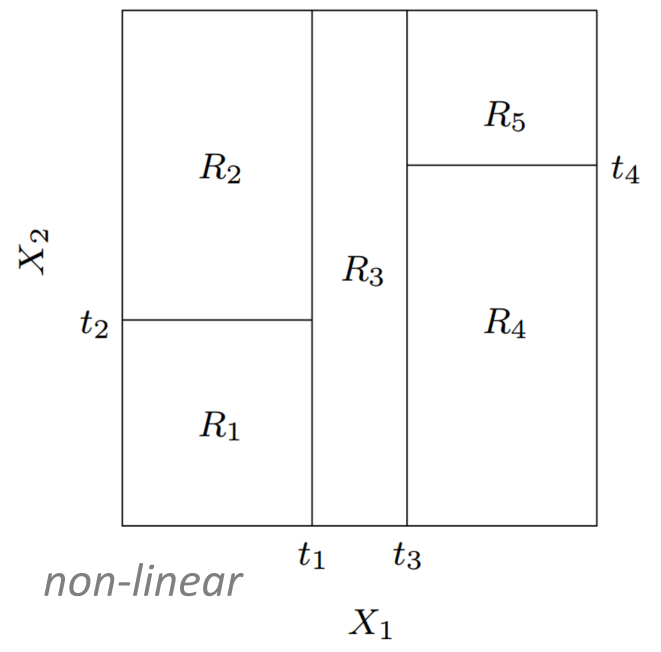
low bias
high variance

low bias
low variance

Decision trees

- **Recall**: *decision trees*
- **ID3 learning algorithm:**
  1. Start with root as leaf.
  2. For each feature, compute best possible split (using entropy).
  3. Create decision node (and split data) using best feature.
  4. Recurse.
- **Trees can be highly non-linear**
- Without restrictions, always exists a tree T having zero empirical error, $L_S(H_T) = 0$
- Generally, large depth implies **high variance**





Variance reduction via averaging

- **Bagging** reduces variance by **averaging**.
- **Recall**: law of large numbers roughly states that:

  If z_i are i.i.d. with mean μ and variance σ^2, then

  1. $\bar{z}_{(k)} = \frac{1}{k}\sum_{i=1}^{k} z_i \xrightarrow[k\to\infty]{} \mu$
  2. $\mathrm{Var}\left(\bar{z}_{(k)}\right) = \frac{\sigma^2}{k}$

- **Idea**: average multiple models

Variance reduction via averaging

bias²: $\mathbb{E}_x\left[\left(\bar{h}(x) - \bar{y}(x)\right)^2\right]$

variance: $\mathbb{E}_{S,x}\left[\left(h_S(x) - \bar{h}(x)\right)^2\right]$

- **Bagging** reduces variance by **averaging**.
- **Recall**: law of large numbers roughly states that:

  If h_i are i.i.d. with mean μ and variance σ^2, then*

  1. $\bar{h}_{(k)} = \frac{1}{k}\sum_{i=1}^{k} h_i \xrightarrow[k\to\infty]{} \mu$ $\rightarrow$ *bias remains roughly the same*
  2. $\mathrm{Var}\left(\bar{h}_{(k)}\right) = \frac{\sigma^2}{k}$ $\rightarrow$ *variance diminishes quickly! (compared to base models)*
- **Idea**: average multiple models
- Can hope for error (or at least its variance term) to decrease at rate $\approx \frac{1}{k}$

* model mean and variance are not well-defined here; a proper treatment requires more elaborate tools

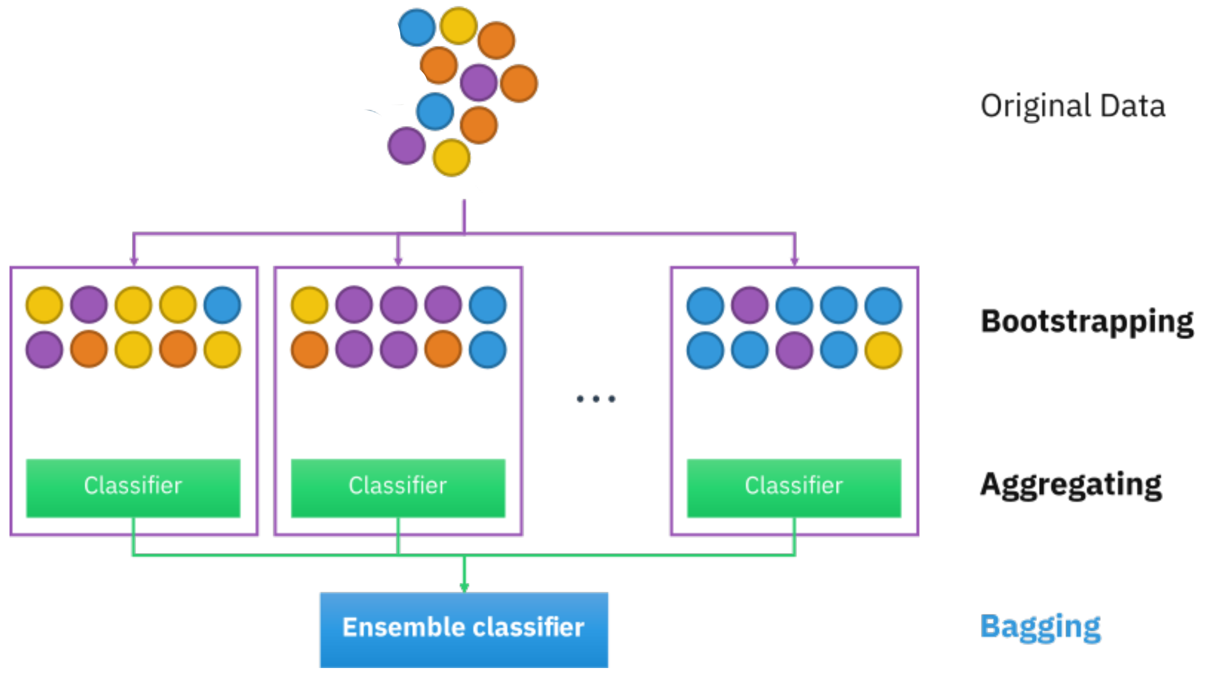
Bootstrapping

- **Q**: How can we obtain k different models $h_1, \dots, h_k$?
- **A**: (obvious) train on k different (iid) data sets $S_1, \dots, S_k$:
$$h_1 = A(S_1), \dots, h_k = A(S_k)$$
- (recall A is our *base learner*)

- **Problem**: we don't have multiple (iid) data sets $S_1, \dots, S_k$ – just one!
- **Naïve solution**: partition S into k sets of size m/k
- **Think**: can/should/will this lead to improved performance?
- **Better solution**: use single S to "simulate" multiple sets by repeated sub-sampling
  - ➢ Many ways to subsample
  - ➢ Bagging uses ***bootstrapping***

Boostrapping

- **Bootstrap** = sample uniformly with replacement: (="take out and put back in")
  - Define distribution $Q((x,y)|S) = \begin{cases} \frac{1}{m} & (x,y) \in S \\ 0 & o.w. \end{cases}$
  - Sample $S_i \sim Q^m$ iid
- Then:
  - Train $h_i = A(S_i)$
  - Aggregate $H = \text{agg}(h_1, \dots, h_k)$ (by averaging)
- Bagging = Bootstrap **Aggregating**

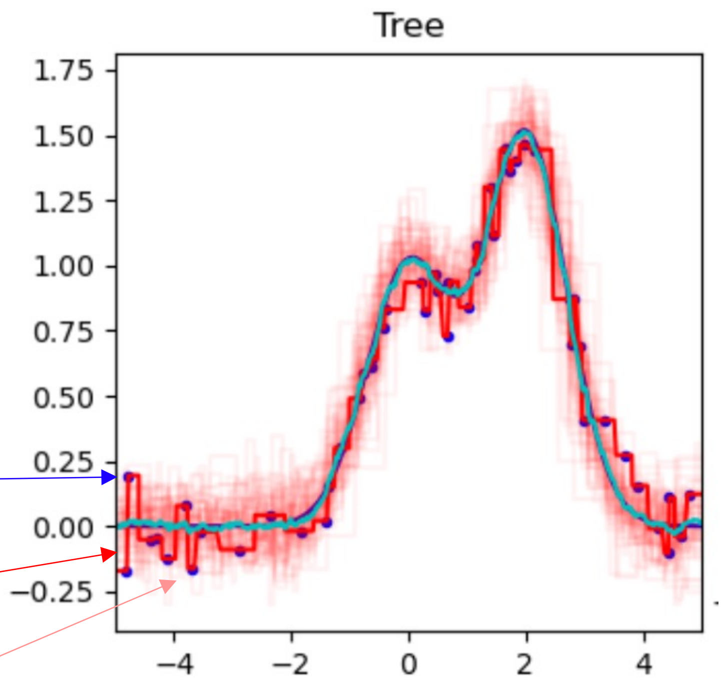


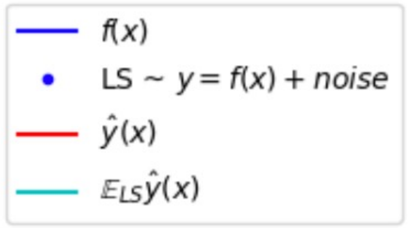
Example

single sample set S

single tree trained on S

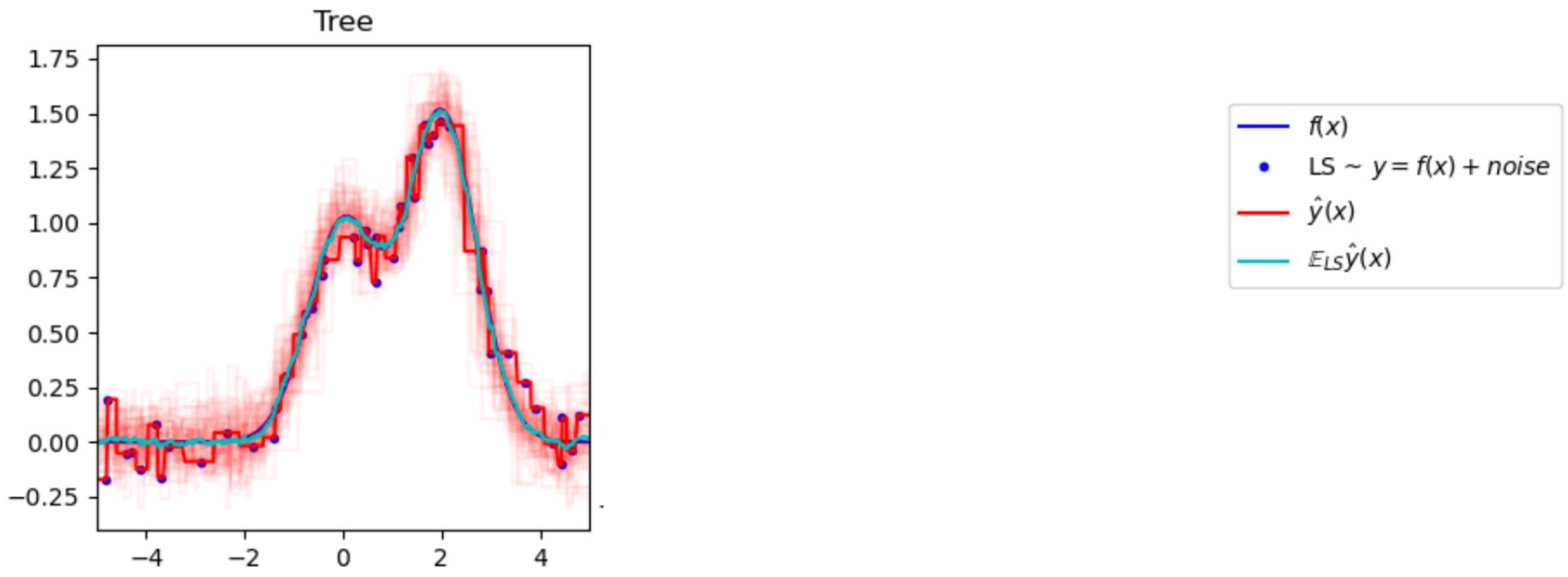
other trees trained on other sample sets S'

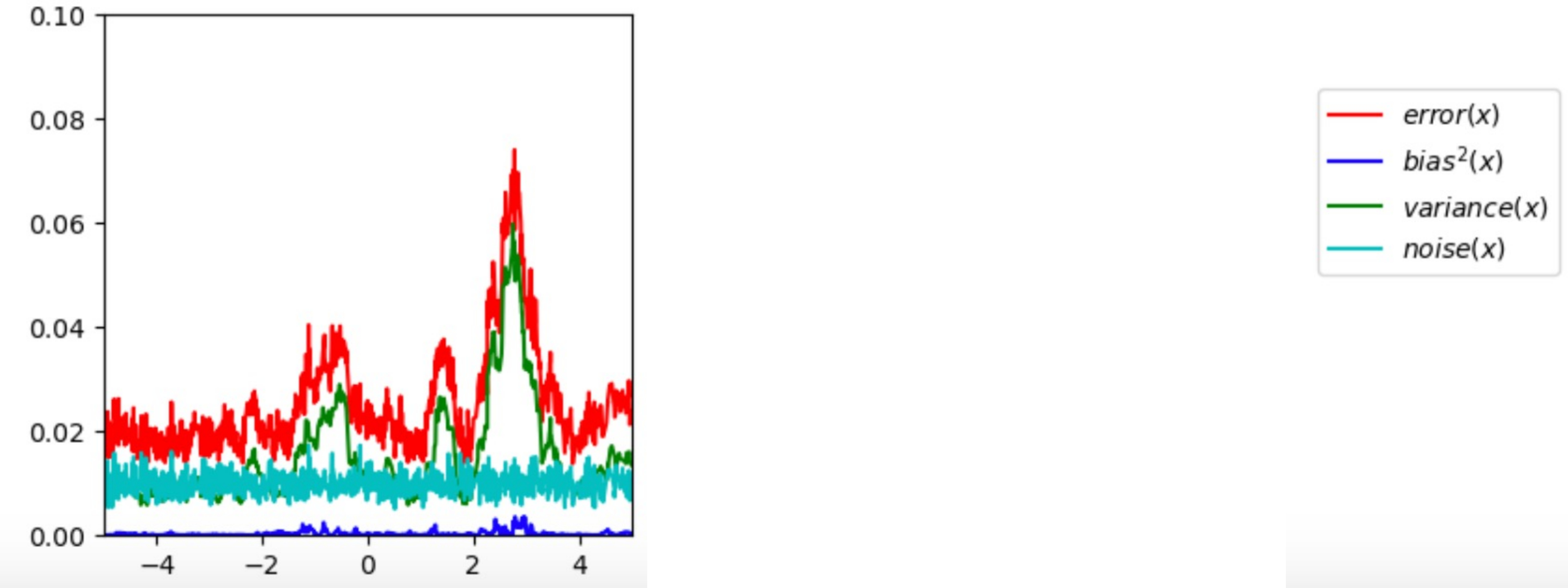




https://scikit-learn.org/stable/auto_examples/ensemble/plot_bias_variance.html

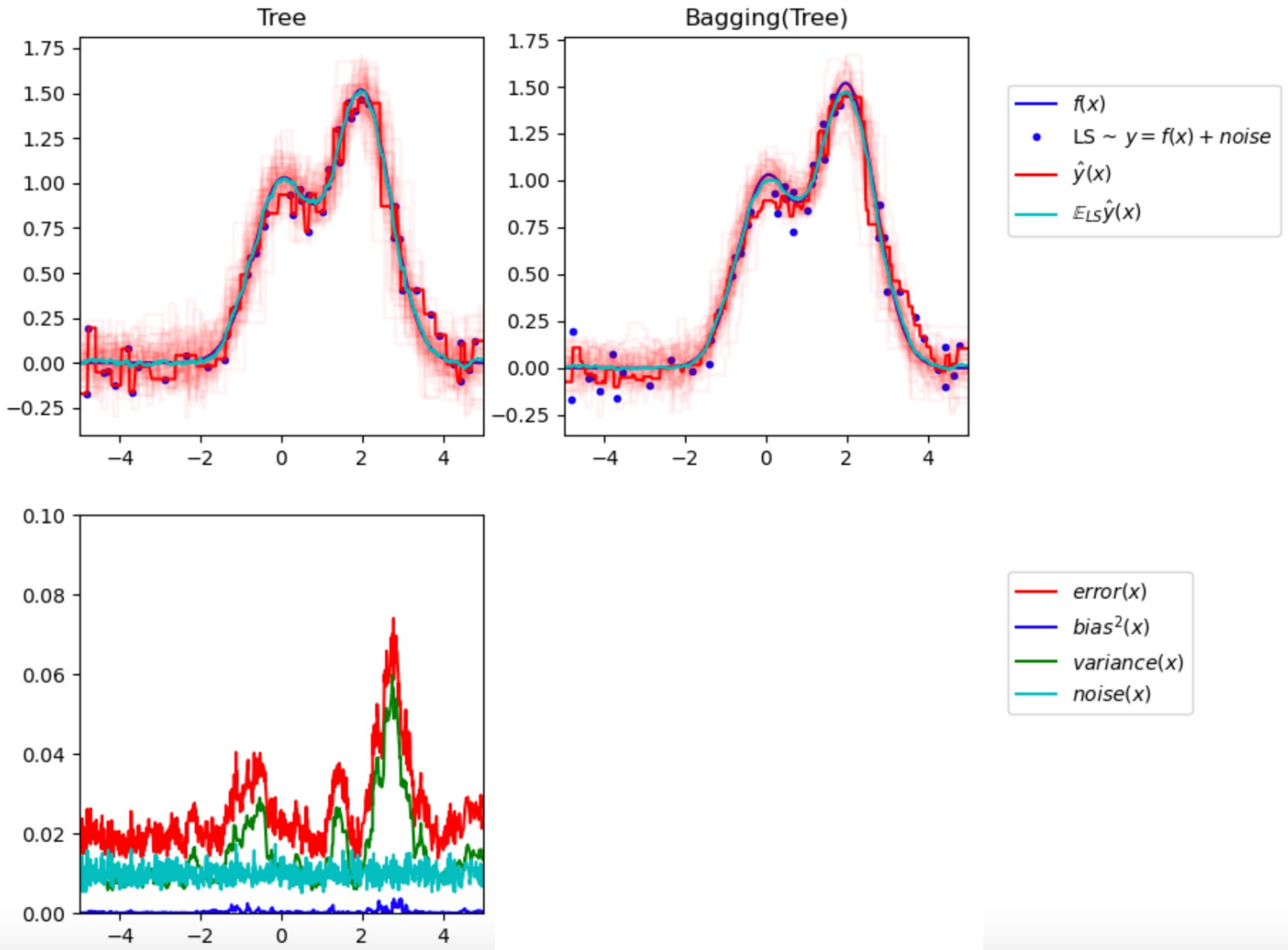
Example





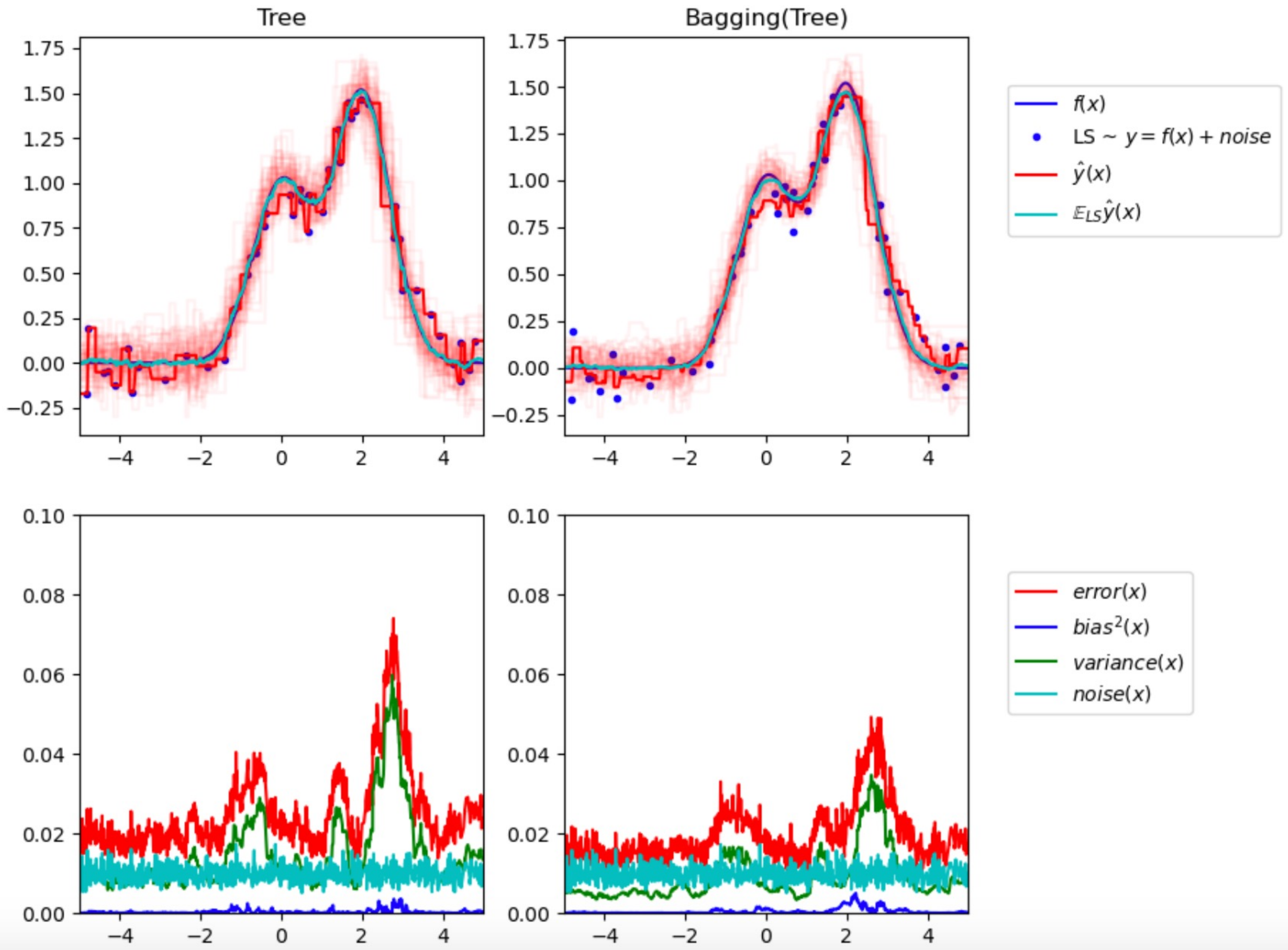
https://scikit-learn.org/stable/auto_examples/ensemble/plot_bias_variance.html

Example



https://scikit-learn.org/stable/auto_examples/ensemble/plot_bias_variance.html

Example



https://scikit-learn.org/stable/auto_examples/ensemble/plot_bias_variance.html

Why it works #1

- Assume for simplicity $y = h^*(x)$ (i.e., no noise)
- Each base learner can be written as: $h_i(x) = h^*(x) + \epsilon_i(x)$

Error specific to each base learner

- Expected (squared-loss) error of each **base learner**:

$$\text{err}_i = \mathbb{E}_x\left[\left(h_i(x) - h^*(x)\right)^2\right] = \mathbb{E}_x\left[\left(h^*(x) + \epsilon_i(x) - h^*(x)\right)^2\right] = \mathbb{E}_x[\epsilon_i(x)^2]$$

- Denote "average error" of base learners, ***had they been acting individually***:

$$\text{err}_{avg} = \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}_x[\epsilon_i(x)^2]$$

- Decmoposition of error of the **ensemble**:

$$\text{err}_H = \mathbb{E}_x\left[\left(H(x) - h^*(x)\right)^2\right] = \mathbb{E}_x\left[\left(\frac{1}{k}\sum_{i=1}^{k} h_i(x) - h^*(x)\right)^2\right] = \mathbb{E}_x\left[\left(\frac{1}{k}\sum_{i=1}^{k} \epsilon_i(x)\right)^2\right]$$

$$= \mathbb{E}_x\left[\frac{1}{k^2}\left(\sum_{i=1}^{k} \epsilon_i(x)^2 + \sum_{i \neq j} \epsilon_i(x)\epsilon_j(x)\right)\right] = \frac{1}{k}\text{err}_{avg} + \frac{1}{k^2}\mathbb{E}_x\left[\sum_{i \neq j} \epsilon_i(x)\epsilon_j(x)\right]$$

Why it works #1

- Decmoposition of error of the **ensemble**:

  $$\mathrm{err}_H = \frac{1}{k}\mathrm{err}_{avg} + \frac{1}{k^2}\mathbb{E}_x\left[\sum_{i\neq j}\epsilon_i(x)\epsilon_j(x)\right]$$

- If errors ϵ_i:

  - have zero mean: $\mathbb{E}_x[\epsilon_i(x)] = 0$

  - are uncorrelated: $\mathbb{E}_x[\epsilon_i(x)\epsilon_i(x)] = \mathbb{E}_x[\epsilon_i(x)]\mathbb{E}_x[\epsilon_i(x)]$,

  then: $\mathrm{err}_H = \frac{1}{k}\mathrm{err}_{avg}$

- **Dramatic reduction in error just by averaging!**
- In practice, errors are usually correlated
- Suffer additional “correlation errors” $\frac{1}{k^2}\mathbb{E}_x\left[\sum_{i\neq j}\epsilon_i(x)\epsilon_j(x)\right]$

Why it works #2

- **Intuition**: “more” variation in input ⇒ less variation in output (through averaging)
- Note that examples are no longer independent (they are linked by S)
- This means $H = \frac{1}{k}\sum_{i=1}^{k} h_i \cancel{\xrightarrow[m\to\infty]{}} \mu$ (S_i are not iid and so LLN does not kick in)
- Nonetheless, their probability under D is preserved
- **Claim**: $P_Q(x, y) = P_D(x, y)$ (that is, sampling from D is like sampling S and then from Q)
- To simplify, consider finite $x \in \{x_1, \ldots, x_n\}$ with $P(x_i) = p_i$, and discard y
- **Proof**:

Why it works #2

- **Intuition**: "more" variation in input ⇒ less variation in output (through averaging)
- Note that examples are no longer independent (they are linked by S)
- This means $H = \frac{1}{k}\sum_{i=1}^{k} h_i \cancel{\xrightarrow[m\to\infty]{}} \mu$ (S_i are not iid and so LLN does not kick in)
- Nonetheless, their probability under D is preserved
- **Claim**: $P_Q(x, y) = P_D(x, y)$ (that is, sampling from D is like sampling S and then from Q)
- To simplify, consider finite $x \in \{x_1, \dots, x_n\}$ with $P(x_i) = p_i$, and discard y
- **Proof**:

$$P_Q(x_i) = P(x_i \in S \text{ and was picked})$$

Why it works #2

- **Intuition**: "more" variation in input ⇒ less variation in output (through averaging)
- Note that examples are no longer independent (they are linked by S)
- This means $H = \frac{1}{k}\sum_{i=1}^{k} h_i \cancel{\xrightarrow[m\to\infty]{}} \mu$ (S_i are not iid and so LLN does not kick in)
- Nonetheless, their probability under D is preserved
- **Claim**: $P_Q(x, y) = P_D(x, y)$ (that is, sampling from D is like sampling S and then from Q)
- To simplify, consider finite $x \in \{x_1, \dots, x_n\}$ with $P(x_i) = p_i$, and discard y
- **Proof**:

prob. of having n copies of x_i in S

$$P_Q(x_i) = P(x_i \in S \text{ and was picked}) = \sum_{n=1}^{m} \binom{m}{n} p_i^n (1 - p_i)^{m-n} \cdot \frac{n}{m}$$

probability of picking one of those copies

Why it works #2

- **Intuition**: “more” variation in input ⇒ less variation in output (through averaging)
- Note that examples are no longer independent (they are linked by S)
- This means $H = \frac{1}{k}\sum_{i=1}^{k} h_i \xrightarrow[m\to\infty]{} \mu$ (S_i are not iid and so LLN does not kick in)
- Nonetheless, their probability under D is preserved
- **Claim**: $P_Q(x,y) = P_D(x,y)$ (that is, sampling from D is like sampling S and then from Q)
- To simplify, consider finite $x \in \{x_1, \dots, x_n\}$ with $P(x_i) = p_i$, and discard y
- **Proof**:

prob. of having n copies of x_i in S

$$P_Q(x_i) = P(x_i \in S \text{ and was picked}) = \sum_{n=1}^{m} \binom{m}{n} p_i^n (1-p_i)^{m-n} \cdot \frac{n}{m}$$

probability of picking one of those copies

expected value of binomial distribution

$$= \frac{1}{m}\sum_{n=1}^{m} \binom{m}{n} p_i^n (1-p_i)^{m-n}\, n$$

$\mathbb{E}[\text{Binomial}(p_i, m)] = mp_i$

Why it works #2

- **Intuition**: "more" variation in input ⇒ less variation in output (through averaging)
- Note that examples are no longer independent (they are linked by S)
- This means $H = \frac{1}{k}\sum_{i=1}^{k} h_i \cancel{\xrightarrow[m\to\infty]{}} \mu$ (S_i are not iid and so LLN does not kick in)
- Nonetheless, their probability under D is preserved
- **Claim**: $P_Q(x, y) = P_D(x, y)$ (that is, sampling from D is like sampling S and then from Q)
- To simplify, consider finite $x \in \{x_1, \dots, x_n\}$ with $P(x_i) = p_i$, and discard y
- **Proof**:

$$P_Q(x_i) = P(x_i \in S \text{ and was picked}) = \sum_{n=1}^{m} \overbrace{\binom{m}{n} p_i^n (1-p_i)^{m-n}}^{\text{prob. of having n copies of } x_i \text{ in } S} \cdot \underbrace{\frac{n}{m}}_{\text{probability of picking one of those copies}}$$

$$= \frac{1}{m} \overbrace{\sum_{n=1}^{m} \binom{m}{n} p_i^n (1-p_i)^{m-n}\, n}^{\text{expected value of binomial distribution}} = \frac{1}{m} m p_i = p_i = P_D(x_i)$$

$$\mathbb{E}[\text{Binomial}(p_i, m)] = m p_i$$

0.632 bootstrapping

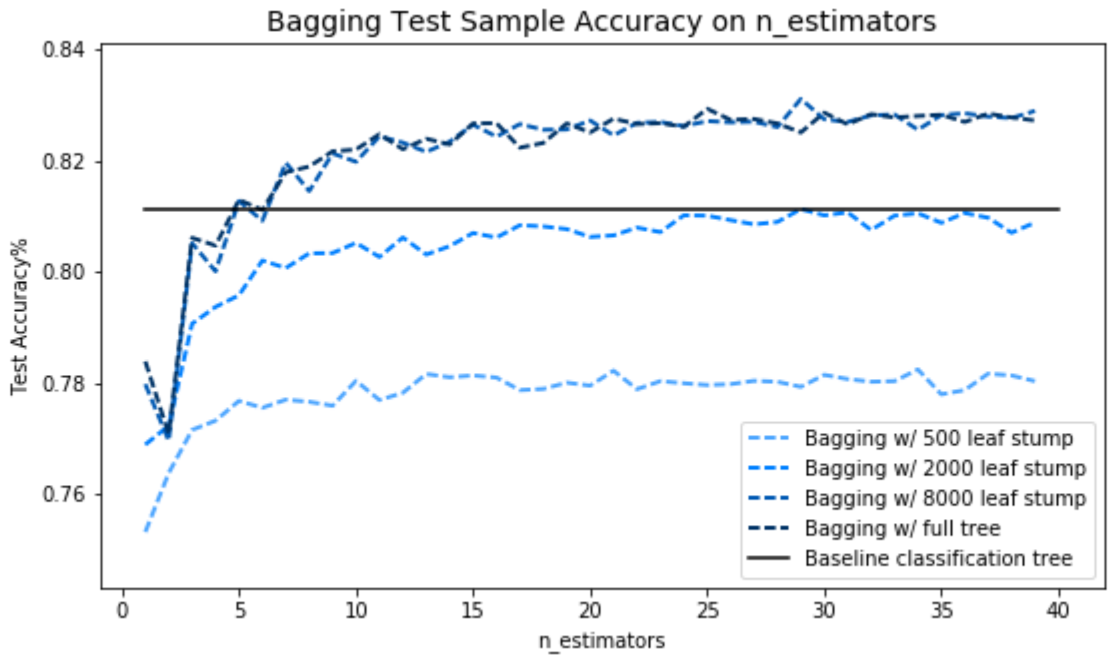
- For any x_i and S_j:
  - At each step, $P(\text{choose } x_i) = \frac{1}{m}$
  - Overall, $P\left(x_i \notin S_j\right) = \left(1 - \frac{1}{m}\right)^m \xrightarrow[m\to\infty]{} e^{-1} = 0.368$
- This means each S_j includes roughly 2/3 of the data, *regardless of m*
- Hence, each h_j is effectively trained on a different, random 1/3-2/3 "split" of the data
- Performance on non-selected points provides unbiased estimate of test error
- Sounds like cross-validation, but free! (i.e., didn't "throw away" validation examples)
- Gives insight into how variance reduction works

Random Forests

- An extremely popular and practically useful instantiation of Bagging
- Uses decision trees, but not using ID3 – rather, uses a different base learner (having additional randomization)

```
Random Forest($S, k, d'$)
1.  Sub-sample $k$ sample sets $S_1, \dots, S_k$ of size $m = |S|$ with replacement from $S$
2.  For each $S_i$, learn an unbounded-depth decision tree $h_i$, but with a twist:
    at each decision node, use only a randomly sampled subset of $d' < d$ features
    (without replacement! i.e., each split criterion is based on different features)
3.  Return $H(x) = \frac{1}{k}\sum_{i=1}^{k} h_i(x)$
```

- Usually fairly insensitive to choices of hyper-parameters k, d' (often set $d' = \sqrt{d}$)


Bagging Test Sample Accuracy on n_estimators
Test Accuracy%
n_estimators
0.84
0.82
0.80
0.78
0.76
0
5
10
15
20
25
30
35
40
Bagging w/ 500 leaf stump
Bagging w/ 2000 leaf stump
Bagging w/ 8000 leaf stump
Bagging w/ full tree
Baseline classification tree

Boosting

Bagging:

- mostly aimed at reducing variance
- targets high-variance models
  e.g., deep trees
- linear aggregation, uniform weights:

$$H = \sum_{i=1}^{k} \frac{1}{k} h_i$$

- parallel training
  (h_i independent given S)

Boosting:

- mostly aimed at reducing bias
- targets high-bias models
  e.g., stumps (depth-one trees)
- linear aggregation, varying weights:

$$H = \sum_{i=1}^{k} \alpha_i h_i$$

- incremental (greedy) training
  (h_i depends on $h_1, \dots, h_{i-1}$)

Weak learners

- Boosting works well for **weak base learners**
  (this is how we can operationalize the notion of "high-bias")
- **Definition**: (for binary classification)

  The base class $\mathcal{H} = \{h: \mathcal{X} \to \{\pm 1\}\}$ is a γ-weakly learnable if exists A and $m(\delta)$ s.t. for all $\delta \in [0,1]$ and all D, for $S \sim D^m$,

$$P_D\left(\mathrm{err}\left(A(S)\right) \leq 0.5 - \gamma\right) \geq 1 - \delta$$

- Intuitively, means learned models perform at least better than random
- A fairly minimal requirement…
  notice that if both $h, -h \in \mathcal{H}$, then at least one has $\mathrm{err} < 0.5$

Weak learners

- Boosting works well for **weak base learners**
  (this is how we can operationalize the notion of "high-bias")
- **Definition**: (for binary classification)

  The base class $\mathcal{H} = \{h: \mathcal{X} \to \{\pm 1\}\}$ is a γ-weakly learnable if exists A and $m(\delta)$ s.t. for all $\delta \in [0,1]$ and all D, for $S \sim D^m$,

$$P_D\big(\mathrm{err}\big(A(S)\big) \le 0.5 - \gamma\big) \ge 1 - \delta$$

Vs. PAC-learnable:

… and $m(\delta, \epsilon)$ s.t. $\forall\, \delta, \epsilon \in [0,1]$

$$P_D\big(\mathrm{err}\big(A(S)\big) \le \epsilon\big) \ge 1 - \delta$$

- Intuitively, means learned models perform at least better than random
- A fairly minimal requirement…
  notice that if both $h, -h \in \mathcal{H}$, then at least one has $\mathrm{err} < 0.5$
- Compare to PAC; in weak learning, m does not depending on error (ϵ)

Weak learners

- Boosting works well for **weak base learners**
  (this is how we can operationalize the notion of "high-bias")
- **Definition**: (for binary classification)

  The base class $\mathcal{H} = \{h: \mathcal{X} \to \{\pm 1\}\}$ is a γ-weakly learnable if exists A and $m(\delta)$ s.t. for all $\delta \in [0,1]$ and all D, for $S \sim D^m$,

  $$P_D\big(\mathrm{err}(A(S)) \leq 0.5 - \gamma\big) \geq 1 - \delta$$

- We call such A a **weak learner**,
  and the learned model h_S a **weak model**
- Simply averaging weakly-learned models (i.e., bagging) isn't going to work
- Enter **Boosting**

Boosting

- **Boosting** uses a weak learner to form a strong learner (i.e., that PAC-learns with any ϵ)
- **Main idea** – iteratively build aggregate model:

```
for $t = 1 \dots T$
    learn $h_t$ with weak learner $A$
    set $\alpha_t$
    update $H_t = H_{t-1} + \alpha_t h_t$
return $H = H_T = \sum_{t=1}^{T} \alpha_t h_t$
```

- α_t can be either pre-determined (constant, diminishing, etc) or optimized (also greedily) (this depends on the particular learning setting)
- **Intuition**: each h_t "compensates" for errors made by its predecessors in H_{t-1}
- Let's start with an example, and then proceed to a more fundamental understanding

AdaBoost

- **AdaBoost** = **Ada**ptive **Boost**ing
- Requires weak learner $A(S; D)$ that takes as input:
  - data $S = \{(x_i, y_i)\}_{i=1}^m$
  - vector $D \in [0,1]^m$ having $\sum_i D_i = 1$ that encodes per-example importance weights
- Each weak model h_t minimizes $\sum_{i=1}^m D_i \mathbb{1}\{y_i \neq h_t(x_i)\}$

```
• initialize $D^{(1)} = (1/m, \dots, 1/m)$
    • for $t = 1 \dots T$
        • Learn weak model $h_t = A(S; D^{(t)})$
        • set $\alpha_t$
        • update $D^{(t+1)}$ based on $D^{(t)}$
    • return $H = H_T = \sum_{t=1}^T \alpha_t h_t$
```

Intuition:
- α_t should be high if h_t is "good"
- $D^{(t+1)}$ should up-weight examples that h_t misclassified

For example (not real derivation)**:**
- $\alpha_t \approx 1/\text{err}(h_t)$
- $D_i^{(t+1)} \approx 1/y_i h_t(x_i)$

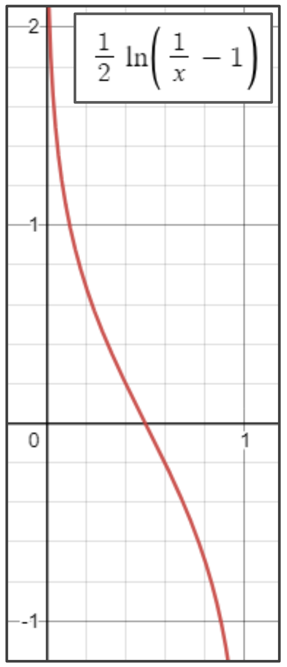
```
AdaBoost$(S, A, T)$
• initialize $D^{(1)} = (1/m, \ldots, 1/m)$
• for $t = 1 \ldots T$:
    1. $h_t = A(S; D^{(t)})$                                              # train with weak learner
    2. $\boxed{\varepsilon_t} = \sum_{i=1}^{m} D_i^{(t)} \boxed{\mathbb{1}\{y_i \neq h_t(x_i)\}}$      # compute weighted 0/1 error
    3. $\boxed{\alpha_t} = {}^1/_2 \log(\boxed{{}^1/_{\varepsilon_t}} - 1)$                          # set model coefficient
    4. $\forall i, D_i^{(t+1)} \propto D_i^{(t)} \boxed{\exp\{-\boxed{\alpha_t} y_i h_t(x_i)\}}$      # update normalized weights
• return $H = \mathrm{sign}(\sum_{t=1}^{T} \boxed{\alpha_t} h_t)$
```

0-1 loss

guess?

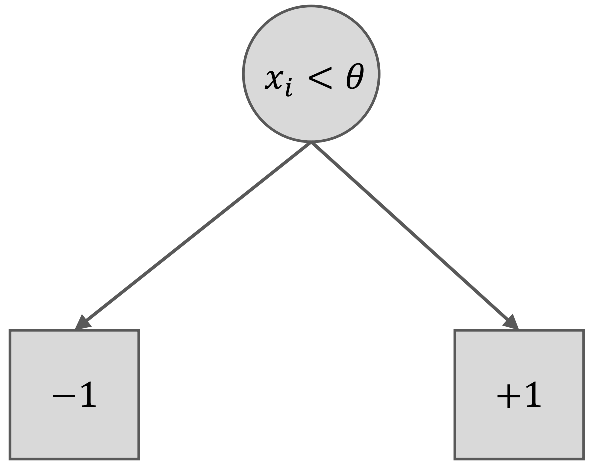
“more” wrong => more importance

learned (=optimized) weights



- More to follow!

most popular base-class for boosting:



stumps

= trees of depth 0

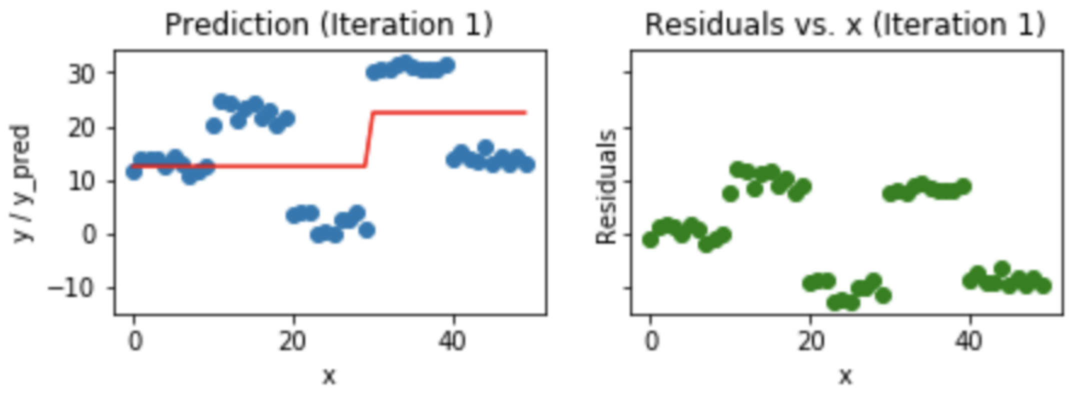
(vs. deep trees in bagging)

Boosting vs. Bagging

- Boosting trains each h_t on a "reweighted" sample, with importance scores $D^{(t)}$
- Each $D^{(t)}$ is updated according to $D^{(t-1)}$
- Can think of Bagging as special case where $\alpha_t = \frac{1}{T}$ and each $D^{(t)}$:
  - ➢ is set independently
  - ➢ and at random
  - ➢ to include "count" entries $D_i \in \{0,1,2,\dots\}$ (and then normalize)
  - ➢ (by sampling indices with replacement)
- **Conclusion**: boosting generalizes bagging
- **Q**: What generalizes boosting?

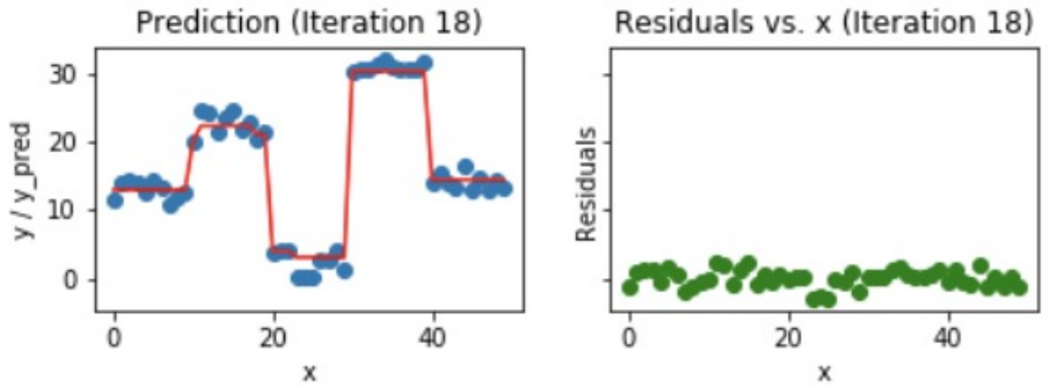
Gradient boosting

A gentle start



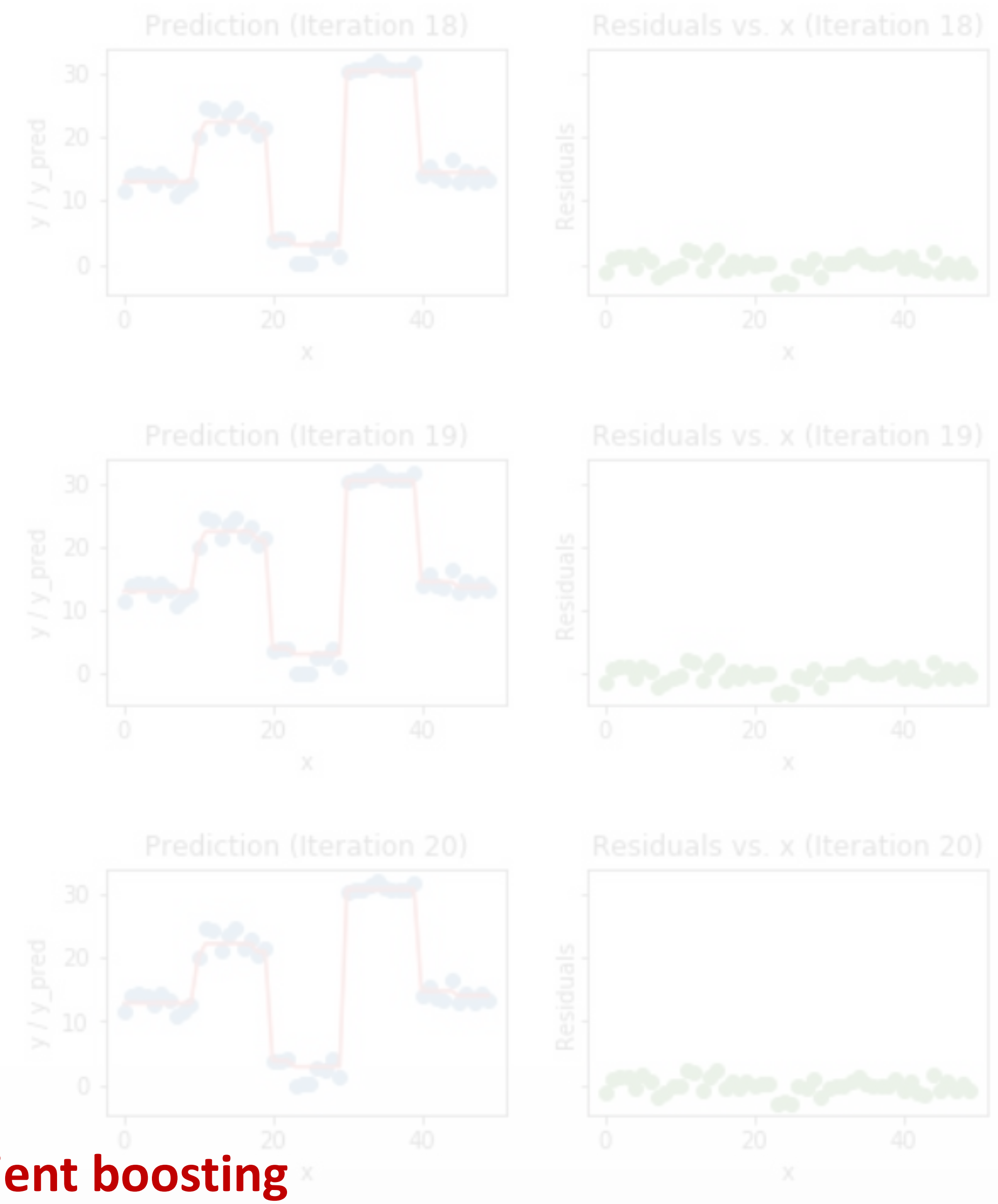
- Consider regression with least squares as a loss
- Ideally, we want $\hat{y} = y$.
- In practice, get residual errors: $r_i = y_i - \hat{y}_i$
- **Idea:**
  - Train sequence of models $h_1, h_2, \ldots$
  - Train each h_t to fit the **residuals** of h_{t-1}
  - Hope that each h_t "compensates" for previous errors
  - Output $H = \sum_t h_t$

A gentle start



- Consider regression with least squares as a loss
- Ideally, we want $\hat{y} = y$.
- In practice, get residual errors: $r_i = y_i - \hat{y}_i$
- **Idea:**
  - Train sequence of models $h_1, h_2, \ldots$
  - Train each h_t to fit the **residuals** of h_{t-1}
  - Hope that each h_t "compensates" for previous errors
  - Output $H = \sum_t h_t$

A gentle start



- Consider regression with least squares as a loss
- Ideally, we want $\hat{y} = y$.
- In practice, get residual errors: $r_i = y_i - \hat{y}_i$
- **Idea:**
    - ➢ Train sequence of models $h_1, h_2, \ldots$
    - ➢ Train each h_t to fit the **residuals** of h_{t-1}
    - ➢ Hope that each h_t “compensates” for previous errors
    - ➢ Output $H = \sum_t h_t$
- But we’d like a more general solution; enter **gradient boosting**

Boosted least squares

- At step t, say we want to minimize $L_S(H_{t-1}) = \sum_i \frac{1}{2}(H_{t-1}(x_i) - y_i)^2$, by adjusting prior predictions $H(x_i)$
- Think of $H_{t-1}(x_i)$ as parameters and take the gradient:

$$g_i = [\nabla L_S(H_{t-1})]_{x_i} = \frac{\partial L_S(H_{t-1})}{\partial H_{t-1}(x_i)} = \frac{\partial \frac{1}{2}(H_{t-1}(x_i) - y_i)^2}{\partial H_{t-1}(x_i)} = H_{t-1}(x_i) - y_i$$

- Residuals are negative gradients(!): $\underbrace{y_i - H_{t-1}(x_i)}_{\boldsymbol{r_i}} = -g_i$

- h fits the residuals and thus approximates the negative gradient

$$h(x_i) \approx -g_i$$

Boosted least squares

- **Gradient-Boosted Least Squares:**

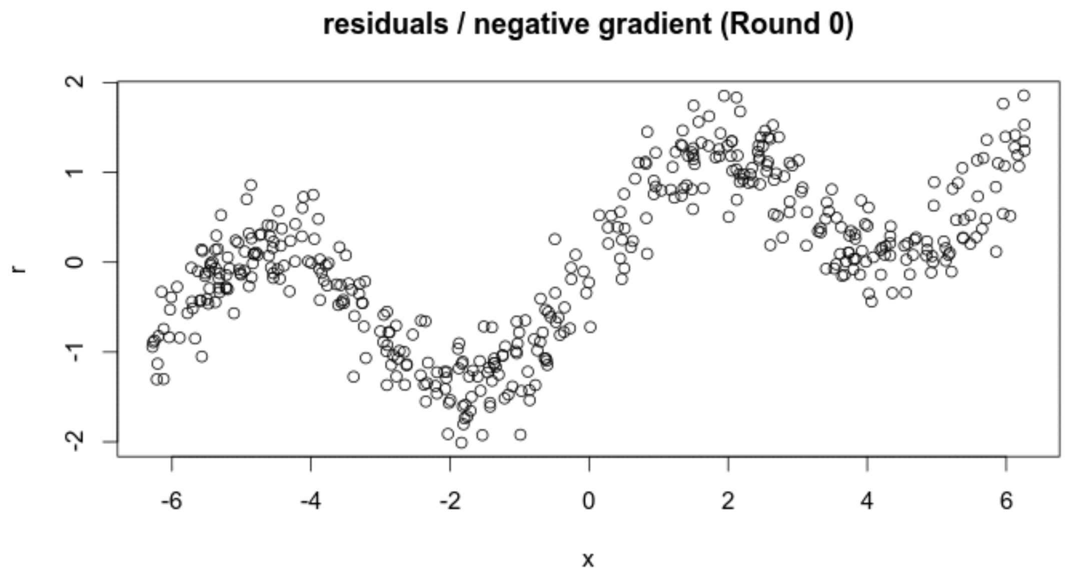
  Start with initial model, say $H_0 = \frac{1}{m}\sum_i y_i$

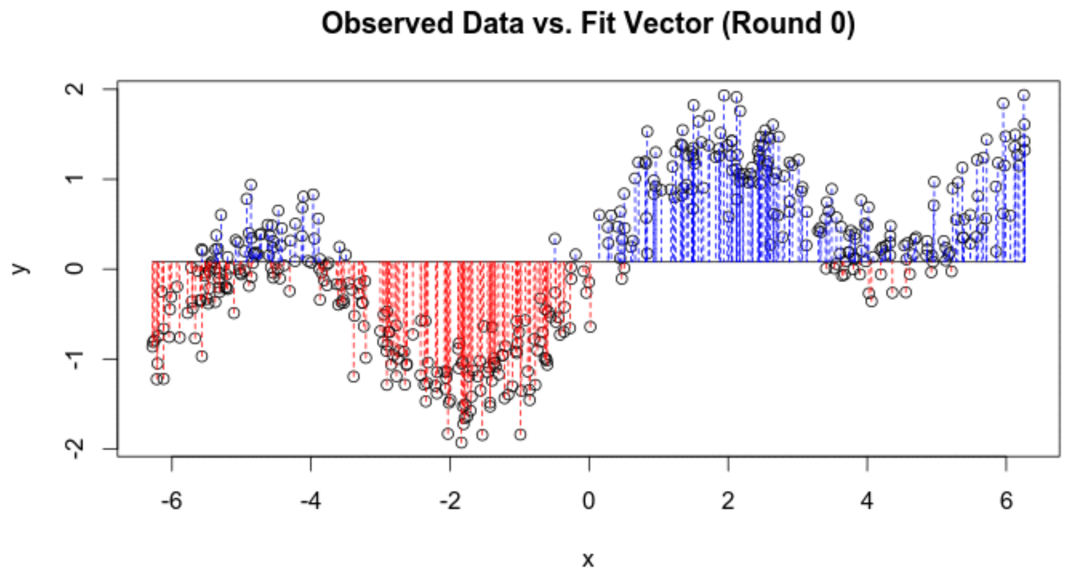
  at each step t,

  1. Compute negative gradient $r_i = y_i - H_{t-1}(x_i)$
  2. Learn $h_t \in \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{m} (h(x_i) - r_i)^2$
  3. Take optimal local step: $H_t = H_{t-1} + h_t$

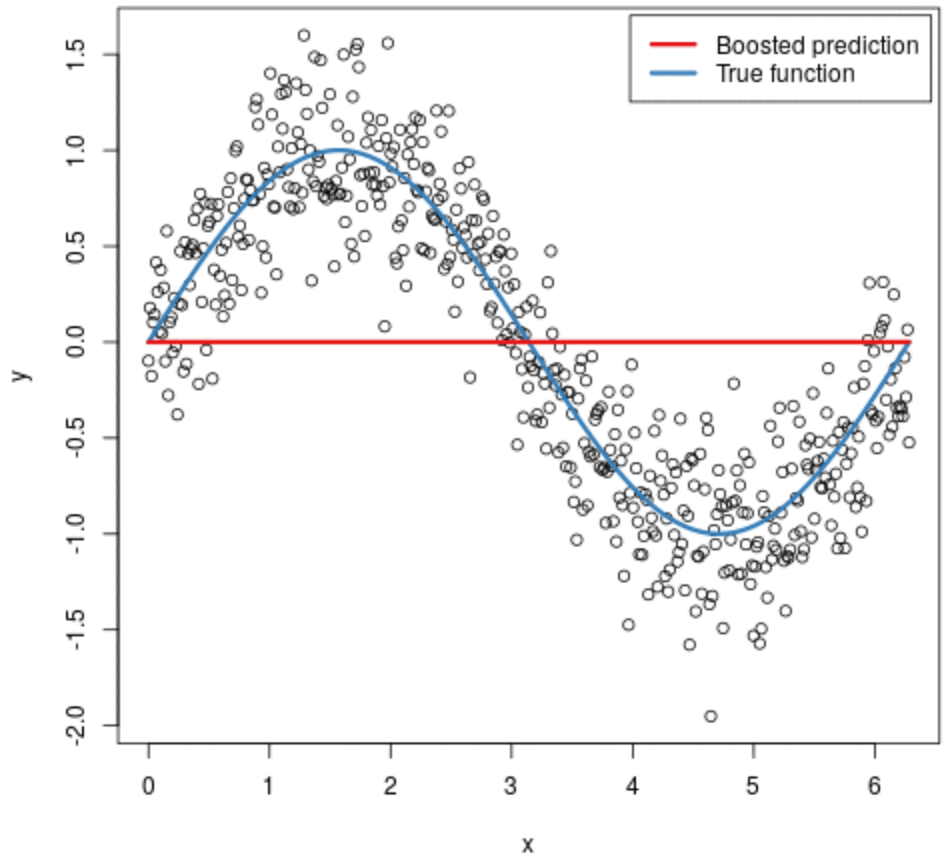
- Notice the update rule

$$H_t(x_i) = H_{t-1}(x_i) + h_t(x_i) \approx H_{t-1}(x_i) - g_i = H_{t-1}(x_i) - 1[\nabla L_S(H_{t-1})]_{x_i}$$





http://colugos.blogspot.com/2018/10/



http://uc-r.github.io/gbm_regression

Functional gradient descent

- Boosted least squares approximates the following update:

$$H_t = H_{t-1} + h_t \approx H_{t-1} - 1[\nabla L_S(H_{t-1})]$$

- Generally, we *want*: $H_t = H_{t-1} - \eta \nabla L_S(H_{t-1})$
- What does this remind you of?
- Recall gradient descent in *parameter space* (e.g., for linear $h_w(x) = w^\top x$):

$$w_t = w_{t-1} - \eta \nabla L_S(w_t)$$

- We can now think of GD as learning an "ensemble":

$$\hat{h} = h_{\hat{w}}, \quad \hat{w} = \sum_t \alpha_t h_t, \quad h_t = \nabla L_S(w_t), \quad \alpha_t = -\eta$$

- Gradient boosting performs gradient descent in function space

Functional gradient descent

- Gradient descent updates: $H_t = H_{t-1} - \eta \nabla L_S(H_{t-1})$
- Great! But – two problems:
  1. no one said $\nabla L_S(H_{t-1}) \in \mathcal{H}$ (recall we want to learn an ensemble)
  2. we can't compute $\nabla L_S(H_{t-1})$ for all x – we only have S!
- Fortunately, one solution solves both problems at once
- **Idea**:
  1. Compute "empirical" gradient, with entries only for observed data points:
$$g_i = [\nabla L_S(H_{t-1})]_{x_i} = \frac{\partial \ell\big(y_i, H_{t-1}(x_i)\big)}{\partial H_{t-1}(x_i)}$$
  2. Find $h_t \in \mathcal{H}$ that best fits empirical gradient, i.e., $h_t(x_i) \approx g_i \;\; \forall i \in [m]$
  3. Update with inferred model: $H_t = H_{t-1} - \eta h_t$

Functional gradient descent

- **Gradient Boosting:** at each step t,

  1. Compute “empirical” gradient $g_i = [\nabla L_S(H_{t-1})]_{x_i}$
  2. Find $h_t \in \mathcal{H}$ that best fits empirical gradient, i.e., $h_t(x_i) \approx g_i \;\; \forall i \in [m]$
  3. Update with inferred model: $H_t \; = H_{t-1} - \eta h_t$

Functional gradient descent

- **Gradient Boosting:** at each step t,

  1. Compute "empirical" gradient $g_i = [\nabla L_S(H_{t-1})]_{x_i}$

  2. Learn $h_t \in \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{m} \ell\big(g_i, h(x_i)\big)$

     (can use *any* loss function; should choose according to what g_i are)

  3. Update with inferred model: $H_t = H_{t-1} - \eta h_t$

Functional gradient descent

- **Gradient Boosting:** at each step t,

  1. Compute "empirical" gradient $g_i = [\nabla L_S(H_{t-1})]_{x_i}$

  2. Learn $h_t \in \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{m} \ell\big(g_i, h(x_i)\big)$

     (can use *any* loss function; should choose according to what g_i are)

  3. Take optimal local step: $H_t = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}}\, H_{t-1} + \alpha h_t$

     (aka "line search"; when possible; sometimes has closed-form solution)

- Together, can think of as approximating $\underset{h \in \mathcal{H}, \alpha \in \mathbb{R}}{\operatorname{argmin}} L_S(H_{t-1} + \alpha h)$
- This approach is called **forward stage-wise optimization**
- Gradient-Boosted least squares is one example

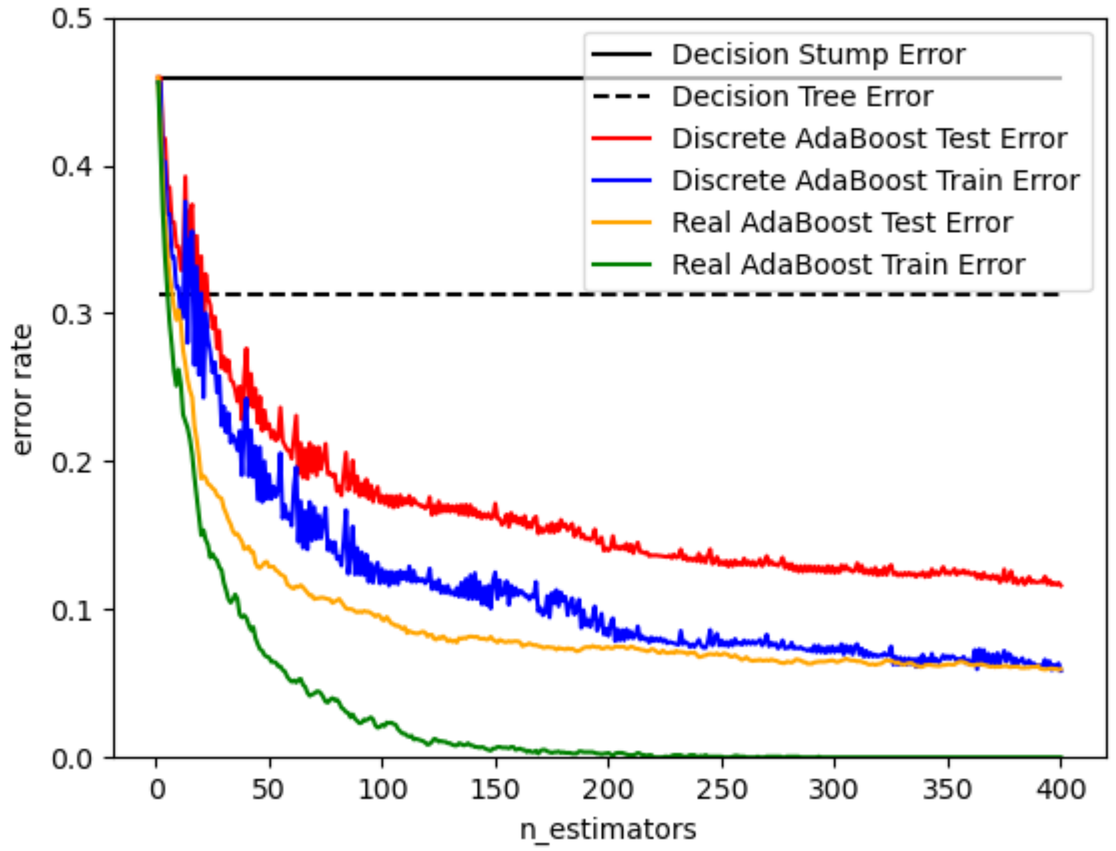
AdaBoost, revisited

- **Claim**:
  AdaBoost is an instance of Gradient Boosting
- **Loss**: exponential, $\ell^{\exp}(y, \hat{y}) = e^{-y\hat{y}}$
- **Gradient**:

$$g_i = \frac{\partial \ell(y_i, H(x_i))}{\partial H(x_i)} = -y_i e^{-y_i H(x_i)}$$

- **Will see in tirgul:**
  1. updating D approximates gradient step
  2. α_t is the optimal step size
- AdaBoost converges *exponentially fast* (won't show)

```
AdaBoost($S, A, T$)
• initialize $D^{(1)} = (1/m, \dots, 1/m)$
• for $t = 1 \dots T$:
      # train with weak learner
      1. $h_t = A(S; D^{(t)})$
      # compute weighted 0/1 error
      2. $\varepsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}\{y_i \neq h_t(x_i)\}$
      # set model coefficient
      3. $\alpha_t = {}^1/_2 \log({}^1/_{\varepsilon_t} - 1)$
      # update normalized weights
      4. $\forall i, D_i^{(t+1)} \propto D_i^{(t)} \exp\{-\alpha_t y_i h_t(x_i)\}$
• return $H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t)$
```



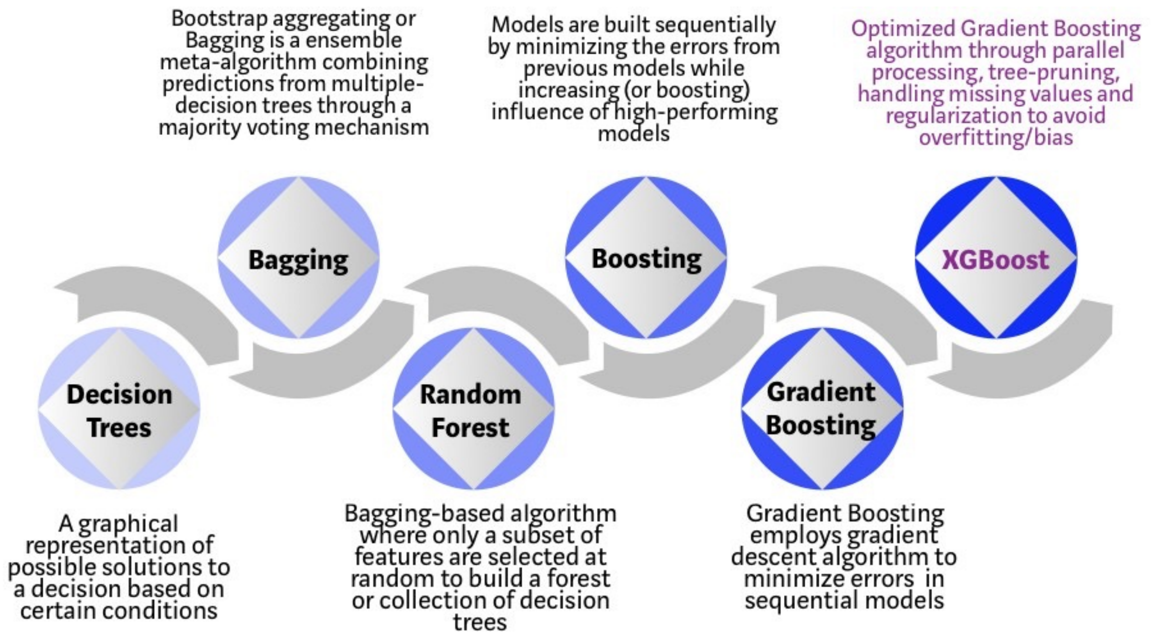
https://scikit-learn.org/stable/modules/ensemble.html

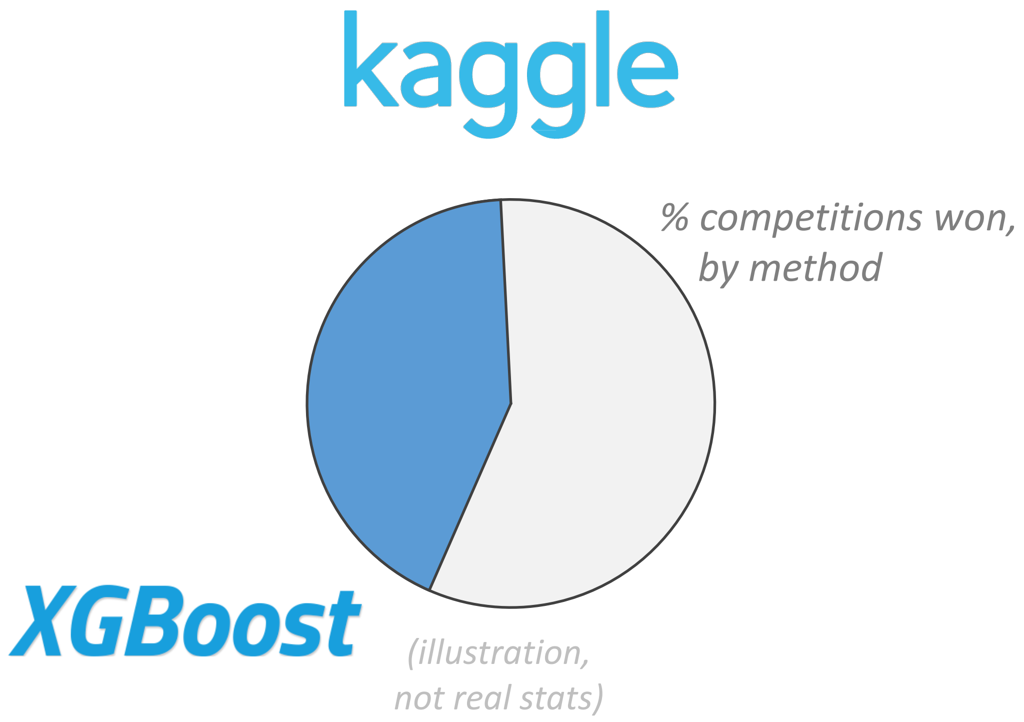
Discussion

- Ensembling is an approach to combine multiple models into a single, better model
  - **Bagging** works well when base learner has **high variance**
  - **Boosting** works well when base learner has **high bias**
- Both propose ways to optimize over a class of linear ensemble models, $H = \sum_i \alpha_i h_i$ (helpful when direct optimization is hard)
- **Cons**: runtime (training, prediction)
- **Pros**:
  - expressivity
  - works very, very well

Discussion

XGBoost: highly-optimized flexible gradient boosting tool

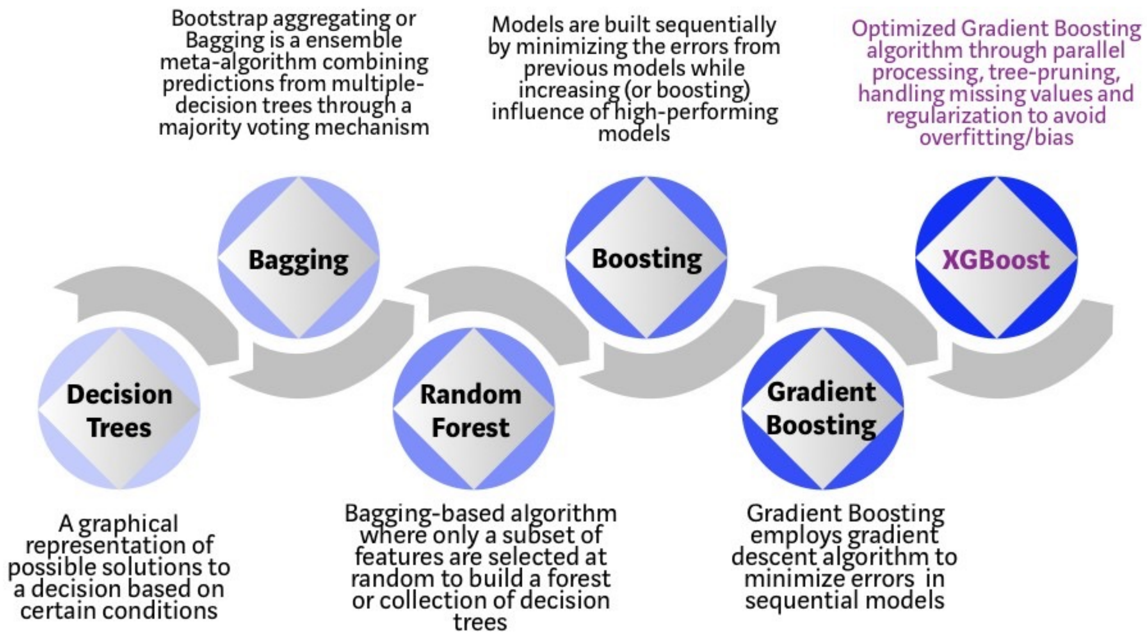

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models



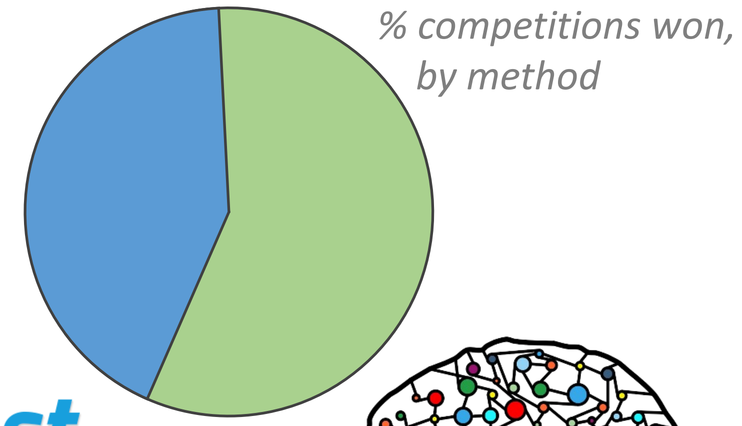
https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

Discussion

XGBoost: highly-optimized flexible gradient boosting tool


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

kaggle



XGBoost

(illustration, not real stats)

https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

Discussion

- **Interpretation**:
  ensembles are linear models over a representation consisting of weak model predictions:

$$H(x) = \sum_i \alpha_i h_i\,(x) \quad \equiv \quad H(x) = w^\top \phi(x), \;\; \big(\phi(x)\big)_i = h_i(x)$$

- Can think of learning as done in two steps:
  1. learn representation mapping ϕ
  2. learn linear model w over representation ϕ
- We will see that deep learning can be thought of as jointly learning ϕ and w
  (and so entries in ϕ will now longer correspond to predictions of some h)

Next week

- **Part III**: *more supervised learning*
  1. Regression
  2. Bagging and boosting (today)
  3. Generative models
  4. Deep learning