DDA4210/AIR6002 Advanced Machine Learning
Lecture 02 Advanced Ensemble Learning

Tongxin Li

School of Data Science, CUHK-Shenzhen

Spring 2024

Overview

Ensemble Learning

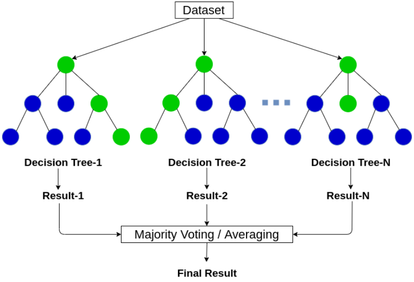
- Ensemble Learning is the process where multiple machine learning models are combined to get better results
  - Example: Suppose classifier 1 predicts Class A, classifier 2 predicts Class B, and classifier 3 predicts Class B, then the final prediction result is Class B. Aggregate(1, 2, 3) → B
- Ensemble learning methods achieved SOTA performance in many real cases (e.g. Kaggle competitions, Netflix competition, and many categorical datasets)

Outlook. (EL).

- works well for
  1. ranking
  2. regression.
  3. clustering.
  etc.
- Image

Ensemble Learning

- Ensemble Learning is the process where multiple machine learning models are combined to get better results
  - Example: Suppose classifier 1 predicts Class A, classifier 2 predicts Class B, and classifier 3 predicts Class B, then the final prediction result is Class B.
- Ensemble learning methods achieved SOTA performance in many real cases (e.g. Kaggle competitions, Netflix competition, and many categorical datasets)
- What have we learned in basic machine learning courses?
  - Bagging (bootstrap aggregation)
  - Random forest (right figure)
  * Connection and difference?
  * Strength and weakness?



Strength and weakness of random forest (RF)

Q: What are the key hyperparameters in a RF?

1. # of trees.
2. depth of a tree.
3. p: = # of features related to each tree.

- Strength of RF
  - Do not require a lot of tuning.
  - Typically very accurate.
  - Handle heterogeneous features well.
  - Implicitly select the most relevant features. Why?
- Weakness of RF vs DT
  - Less interpretable, slower to train (but parallelizable)
  - Do not work well on high dimensional sparse data (e.g. text)

Example of heterogeneous data.

students ID	Gender	Height	Age
1			
2			
3			
⋮			

Strength and weakness of random forest (RF)

- Strength of RF
  - Do not require a lot of tuning.
  - Typically very accurate.
  - Handle heterogeneous features well.
  - Implicitly select the most relevant features. Why?
- Weakness of RF
  - Less interpretable, slower to train (but parallelizable)
  - Do not work well on high dimensional sparse data (e.g. text)
- The view of bias-variance trade-off for bagging and RF
  - Recall: Expected test error = Bias^2 + Variance + Noise^2
  - Bagging reduces variance by averaging
  - Bagging has little effect on bias
  - How can we reduce bias?

1 Introduction

2 Boosting
- Gradient Boosting
- AdaBoost
- More boosting methods

3 Stacking

4 Summary

Boosting

Can <u>weak learners</u> be combined to generate a strong learner with low bias?

—-Michael Kearns,1988

- "weak learner" (also called base learner): a learner (e.g. classifier, predictor, etc) that performs relatively poorly–its accuracy is above chance. 0.55
  - e.g., shallow decision trees, small neural networks
- "strong learner": a learner (e.g. classifier, predictor, etc) that achieves arbitrarily good performance, much better than random guessing.
  0.95

Boosting

Can <u>weak learners</u> be combined to generate a strong learner with low bias?

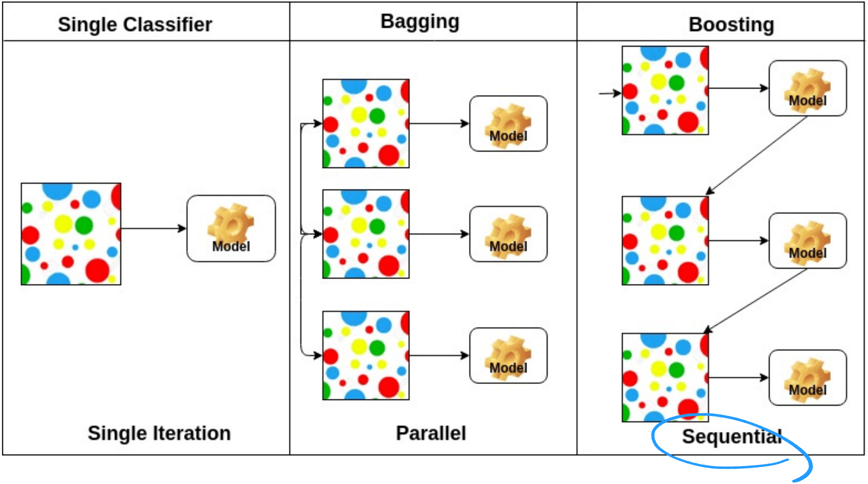
—-Michael Kearns,1988

- "weak learner" (also called base learner): a learner (e.g. classifier, predictor, etc) that performs relatively poorly–its accuracy is above chance.
  - e.g., shallow decision trees, small neural networks
- "strong learner": a learner (e.g. classifier, predictor, etc) that achieves arbitrarily good performance, much better than random guessing.
- Boosting can reduce bias!

  In machine learning, boosting is an ensemble meta-algorithm for primarily reducing bias, also variance in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones.—Wikipedia

Boosting: main idea

Single classifier vs bagging vs boosting



- Like bagging, boosting is a general approach that can be applied to many machine learning methods for regression or classification
- Main idea of boosting $h_1(x), h_2(x), \dots, h_T(x) \leftarrow$ weak learners
  - Ensemble model (weighted sum of weak learners): TBD

Training

$$\text{strong learner} \leftarrow H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \qquad (1)$$

Idea: train $h_1(x) \cdots h_T(x)$ sequentially.

  - This ensemble model is built in an iterative fashion.
  - In iteration t we add the weak learner $\alpha_t h_t(\mathbf{x})$ to the ensemble.

Testing

  - At the test stage we evaluate all weak learners and return the weighted sum.

$t=1$: $\rightarrow h_1(x) \xrightarrow{t=2} H_2(x) \xrightarrow{t=3} H_3(x) \rightarrow \cdots \rightarrow H_T(x)$

$H_2(x) = \alpha_1 h_1 + \alpha_2 h_2$

$H_3(x) = H_2(x) + \alpha_3 h_3$

$H_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$

error loss $\rightarrow \alpha_2, h_2$

error loss $\rightarrow \alpha_3, h_3$

error loss $\rightarrow \sum_{t=1}^{T} \alpha_t h_t(x).$

Boosting: main idea

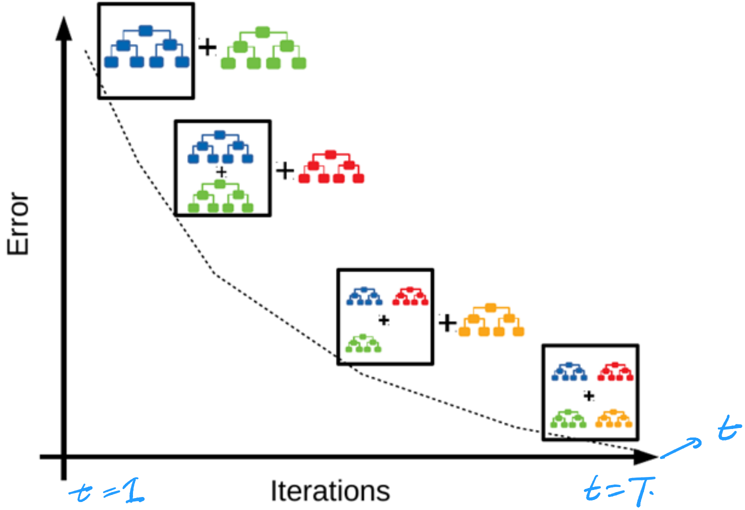
- Like bagging, boosting is a general approach that can be applied to many machine learning methods for regression or classification
- Main idea of boosting
  - Ensemble model (weighted sum of weak learners):

$$H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \tag{1}$$

  - This ensemble model is built in an iterative fashion.
  - In iteration t we add the weak learner $\alpha_t h_t(\mathbf{x})$ to the ensemble.
  - At the test stage we evaluate all weak learners and return the weighted sum.
- Comparing boosting with bagging (suppose h is a tree)
  - Bagging constructs all trees independently
  - Boosting constructs all trees sequentially

Boosting: main idea

An intuitive example



Boosting: main idea

- Ensemble model: $H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$; data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$
- Details of constructing the ensemble model
  - Very similar to gradient descent.
  - However, instead of updating the model parameters in each iteration, we add functions to our ensemble.

Goal: Find / optimize $\{\alpha_t, h_t\}_{t=1}^{T}$ sequentially!

Boosting: main idea

- Ensemble model: $H_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$; data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$
- Details of constructing the ensemble model
  - Very similar to gradient descent.
  - However, instead of updating the model parameters in each iteration, we add functions to our ensemble.
  - Let ℓ denote a loss function and write

e.g. $\ell(x,y) = (x-y)^2$

square loss. Then

$$\mathcal{L}(H) := \frac{1}{n} \sum_{i=1}^{n} \ell(H(\mathbf{x}_i), y_i) \tag{2}$$

$\mathcal{L}(H) = \mathcal{L}_{MSE}(H)$

↓

mean square error

Current time t. I have $\{h_t\}_{t=1}^{t}$.

  - We compute h_{t+1} (as well as α_{t+1}) via

At time t. I know H_t. → current model.

$$h_{t+1}, \alpha_{t+1} = \operatorname*{argmin}_{h \in \mathbb{H},\ \alpha} \mathcal{L}(H_t + \alpha h) \tag{3}$$

- $\mathbb{H}$: hypothesis set → I can customize

$\alpha_{t+1} \cdot h_{t+1}$

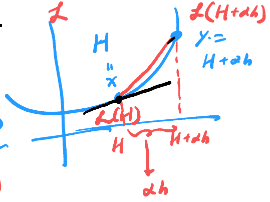
- Once (3) is solved, we add h_{t+1} to our ensemble, i.e., $H_{t+1} := H_t + \alpha_{t+1} h_{t+1}$.
- How to solve (3)?

Gradient Boosting

- Taylor approximation on $\mathcal{L}(H + \alpha h)$

$$\mathcal{L}(H + \alpha h) \approx \mathcal{L}(H) + \alpha \langle \nabla \mathcal{L}(H), h \rangle \tag{4}$$

* α should be small enough; suppose α is fixed.



Remark:

1. We only focus on optimizing h.
   (α is more like a step size)

2. The learning rate (weight) α can also be optimized similarly. (details omitted in gradient boosting).

Gradient Boosting

- Taylor approximation on $\mathcal{L}(H + \alpha h)$

$$\mathcal{L}(H + \alpha h) \approx \mathcal{L}(H) + \alpha \langle \nabla \mathcal{L}(H), h \rangle \tag{4}$$

doesn't depend on h

fixed.

  - * α should be small enough; suppose α is fixed.

- Find h via

$$\begin{aligned} h &= \mathrm{argmin}_{h \in \mathbb{H}} \mathcal{L}(H + \alpha h) \\ &\approx \mathrm{argmin}_{h \in \mathbb{H}} \langle \nabla \mathcal{L}(H), h \rangle \\ &= \mathrm{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]} h(\mathbf{x}_i) \quad \text{(Gradient Boosting)} \end{aligned} \tag{5}$$

# of data samples.

$$\nabla \mathcal{L}(H) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial [H(x_1)]} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial [H(x_n)]} \end{bmatrix} \qquad h = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_n) \end{bmatrix}$$

column vector

Gradient Boosting

- Taylor approximation on $\mathcal{L}(H+\alpha h)$

next loss current loss < 0

$$\mathcal{L}(H+\alpha h) \approx \mathcal{L}(H) + \alpha\langle\nabla\mathcal{L}(H), h\rangle \tag{4}$$

  * α should be small enough; suppose α is fixed.

- Find h via

$H_{t+1} = H + \alpha h$ new model.

$H_t := H$ old model

$\mathcal{L}(\alpha h + H) \leq \mathcal{L}(H)$.

$$\begin{aligned} h &= \operatorname{argmin}_{h\in\mathbb{H}}\mathcal{L}(H+\alpha h) \\ &\approx \operatorname{argmin}_{h\in\mathbb{H}}\langle\nabla\mathcal{L}(H), h\rangle \\ &= \operatorname{argmin}_{h\in\mathbb{H}}\sum_{i=1}^{n}\frac{\partial\mathcal{L}}{\partial[H(\mathbf{x}_i)]}h(\mathbf{x}_i) \quad \text{(Gradient Boosting)} \end{aligned} \tag{5}$$

- Thus we can do boosting if we have an algorithm to compute $h_{t+1} = \operatorname{argmin}_{h\in\mathbb{H}}\sum_{i=1}^{n} q_i h(\mathbf{x}_i)$, where $q_i = \frac{\partial\mathcal{L}}{\partial[H_t(\mathbf{x}_i)]}$, irrelevant to h.
  - q_i is the weight for the training data i.
  - h does not need to be perfect.
  - We make progress as long as $\sum_{i=1}^{n} q_i h(\mathbf{x}_i) < 0$. Why?

$\Rightarrow \mathcal{L}(H+\alpha h) \leq \mathcal{L}(H)$.

Gradient Boosting

- Suppose $\mathcal{F}$ is the objective function we want to minimize to find the parameters Θ for a model (e.g., linear regression, neural network).
- We compare gradient boosting with gradient descent.

gradient boosting	gradient descent
$h = \mathrm{argmin}_{h\in\mathbb{H}}\mathcal{L}(H+\alpha h)$ $\approx \mathrm{argmin}_{h\in\mathbb{H}}\langle\nabla\mathcal{L}(H), h\rangle$	$\theta = \mathrm{argmin}_{\theta}\mathcal{F}(\Theta+\eta\theta)$ $\approx \mathrm{argmin}_{\theta}\langle\nabla\mathcal{F}(\Theta), \theta\rangle$
$h = -\nabla\mathcal{L}(H)$ (ideal)	$\theta = -\nabla\mathcal{F}(\Theta)$
learn h on $-\frac{\partial\mathcal{L}}{\partial[H(\mathbf{x}_i)]}$, $i = 1, \ldots, n$	θ is the negative gradient

a parametrized function (weak learner) in $\mathbb{H}$.

[$h: \mathcal{X} \to \mathcal{Y}$ $x_i \in \mathcal{X}$ $i=1..n$
$y_i \in \mathcal{Y}$ $i=1...n$.]

Gradient Boosting

- Suppose $\mathcal{F}$ is the objective function we want to minimize to find the parameters Θ for a model (e.g., linear regression, neural network).
- We compare gradient boosting with gradient descent.

gradient boosting	gradient descent
$h = \operatorname{argmin}_{h \in \mathbb{H}} \mathcal{L}(H + \alpha h)$ $\approx \operatorname{argmin}_{h \in \mathbb{H}} \langle \nabla \mathcal{L}(H), h \rangle$	$\theta = \operatorname{argmin}_{\theta} \mathcal{F}(\Theta + \eta\theta)$ $\approx \operatorname{argmin}_{\theta} \langle \nabla \mathcal{F}(\Theta), \theta \rangle$
$h = -\nabla \mathcal{L}(H)$ (ideal)	$\theta = -\nabla \mathcal{F}(\Theta)$
learn h on $-\frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]}$, $i = 1, \ldots, n$	θ is the negative gradient

- In gradient boosting, we require $h(\mathbf{x}_i) \approx -\frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]}$ such that $\sum_{i=1}^{n} q_i h(\mathbf{x}_i) < 0$. For instance, when $h(\mathbf{x}_i) = -\frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]}$, $\mathcal{L}(H + \alpha h) \approx \mathcal{L}(H) - \alpha \|\nabla \mathcal{L}(H)\|^2$, which ensures a decreasing $\mathcal{L}$.

Generic algorithm of Gradient Boosting

Big picture.

Also called AnyBoost [Llew Mason et al. 2000]

Input: ℓ, α, T, $\{(\mathbf{x}_i, y_i)\}$, $\mathbb{H}$, $\mathbb{A}$(an algorithm)

$H_0 = 0$

for $t = 0 : T - 1$ **do**

  $q_i = \frac{\partial \mathcal{L}}{\partial H_t(\mathbf{x}_i)}$, $i = 1, 2, \ldots, n$

Taylor apr...

gradient boosting.

  $h_{t+1} = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{n} q_i h(\mathbf{x}_i)$ via Algorithm $\mathbb{A}$

  if $\sum_{i=1}^{n} q_i h_{t+1}(\mathbf{x}_i) < 0$ **then**

    $H_{t+1} = H_t + \alpha h_{t+1}$ (α may also be optimized)

  else

    return (H_t)

  end if

end for

Output: H_T

A brief history of Gradient Boosting

- [Schapire,1989] first provable boosting algorithm
- [Freund, 1990] “optimal" algorithm that “boosts by majority"
- [Drucker, Schapire & Simard, 1992] first experiments using boosting, limited by practical drawbacks
- [Freund et al., 1996, Freund and Schapire, 1997] invent AdaBoost (to be introduced later), the first successful boosting algorithm
- [Breiman et al., 1998, Breiman, 1999] formulate AdaBoost as gradient descent with a special loss function
- [Friedman et al., 2000, Friedman, 2001] generalize Adaboost to Gradient Boosting in order to handle a variety of loss functions

Gradient Boosting for Regression

- Solve $h_{t+1} = \mathrm{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{n} q_i h(\mathbf{x}_i)$, where $q_i = \frac{\partial \mathcal{L}}{\partial [H(\mathbf{x}_i)]}$

Gradient Boosting for Regression

- Solve $h_{t+1} = \operatorname{argmin}_{h\in\mathbb{H}} \sum_{i=1}^n q_i h(\mathbf{x}_i)$, where $q_i = \frac{\partial \mathcal{L}}{\partial[H(\mathbf{x}_i)]}$
- Let $\sum_{i=1}^n h^2(\mathbf{x}_i) = $ constant (we can always normalize the predictions) and replace q_i with $-2r_i$. We have

Original Data $\{x_i, y_i\}_{i=1}^n$ → H_0 → weighted Data → h_1 → H_1 → weighted Data

$$
\begin{aligned}
h_{t+1} &= \operatorname{argmin}_{h\in\mathbb{H}} \sum_{i=1}^n q_i h(\mathbf{x}_i) \\
&= \operatorname{argmin}_{h\in\mathbb{H}} -2\sum_{i=1}^n r_i h(\mathbf{x}_i) \\
&= \operatorname{argmin}_{h\in\mathbb{H}} \sum_{i=1}^n \left(r_i^2 - 2r_i h(\mathbf{x}_i) + (h(\mathbf{x}_i))^2\right) \\
&= \operatorname{argmin}_{h\in\mathbb{H}} \sum_{i=1}^n (h(\mathbf{x}_i) - r_i)^2
\end{aligned}
\tag{6}
$$

: = train h on new weightrece data $\{(x_i, r_i)\}_{i=1}^n$.

- We train h_{t+1} to predict r_i, which are from the old model H_t.

Gradient Boosting for Regression

Special Case: MSE

$$\mathcal{L}(H) = \sum_{i=1}^{n} (H(x_i) - y_i)^2$$

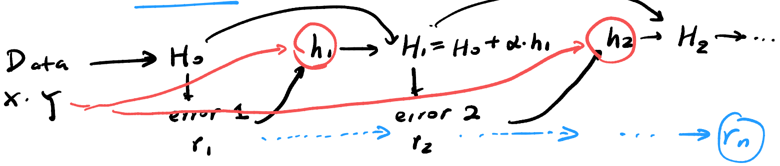
what is $\longrightarrow$ $\frac{\partial \mathcal{L}}{\partial[H(x_i)]} = 2(H(x_i) - y_i)$

- Note that $h_{t+1} = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{n} (h(\mathbf{x}_i) - r_i)^2$, where $r_i = -q_i/2$.
- Let ℓ be the squared loss, i.e., $\mathcal{L}(H) = \sum_{i=1}^{n} (H(\mathbf{x}_i) - y_i)^2$, then

$$r_i = -\frac{1}{2} \times \frac{\partial \mathcal{L}}{\partial[H(\mathbf{x}_i)]} = y_i - H(\mathbf{x}_i)$$

called "pseudo-residuals".

  - r_i are just the residuals given by the old model.



Gradient Boosting for Regression

- Pseudo code [A special instance using MSE]

Require: $\ell, \; T, \; \alpha, \; \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \; \mathbb{H}$

step size (can be optimized. later!)

$H = 0$

for $t = 0 : T - 1$ **do**

$\quad r_i = y_i - H(\mathbf{x}_i), \; i = 1, 2, \ldots, n$

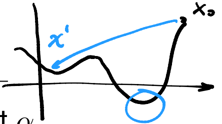
$\quad h = \text{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{n} (h(\mathbf{x}_i) - r_i)^2$

$\quad H \leftarrow H + \alpha h$

end for

Ensure: H

" Trade-off of selecting the step-size α "



- Characteristic: squared loss and fixed weight α

Gradient Boosting for Regression

- Square loss is easy to deal with mathematically but not robust to outliers.
- Absolute loss (more robust to outliers)

y_1 $h(x_1)$

$\downarrow$

outlier

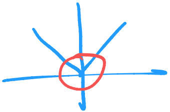
$$\ell(y, \hat{y}) = |y - \hat{y}|$$

Gradient Boosting for Regression

- Square loss is easy to deal with mathematically but not robust to outliers.
- Absolute loss (more robust to outliers)

$$\ell(y, \hat{y}) = |y - \hat{y}|$$

"sub-gradient"



GBR with absolute loss:

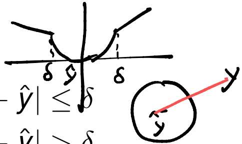
- The gradient is $q_i = \frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = -\mathrm{sign}(y_i - H(\mathbf{x}_i))$.
- Then fit h on $-q_i$, $i = 1, 2, \ldots, n$. (no longer the residuals, different from using the squared loss)

- Square loss is easy to deal with mathematically but not robust to outliers.
- Absolute loss (more robust to outliers)

$$\ell(y, \hat{y}) = |y - \hat{y}|$$

GBR with absolute loss:

- The gradient is $q_i = \frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = -\text{sign}(y_i - H(\mathbf{x}_i))$.
- Then fit h on $-q_i$, $i = 1, 2, \ldots, n$. (no longer the residuals, different from using the squared loss)



- Huber loss (more robust to outliers)

$$\ell(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & |y - \hat{y}| \leq \delta \\ \delta(|y - \hat{y}| - \delta/2) & |y - \hat{y}| > \delta \end{cases}$$

sub

The gradient is

$$\frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = \begin{cases} -(y_i - H(\mathbf{x}_i)) & |y_i - H(\mathbf{x}_i)| \leq \delta \\ -\delta \text{sign}(y_i - H(\mathbf{x}_i)) & |y_i - H(\mathbf{x}_i|) > \delta \end{cases}$$

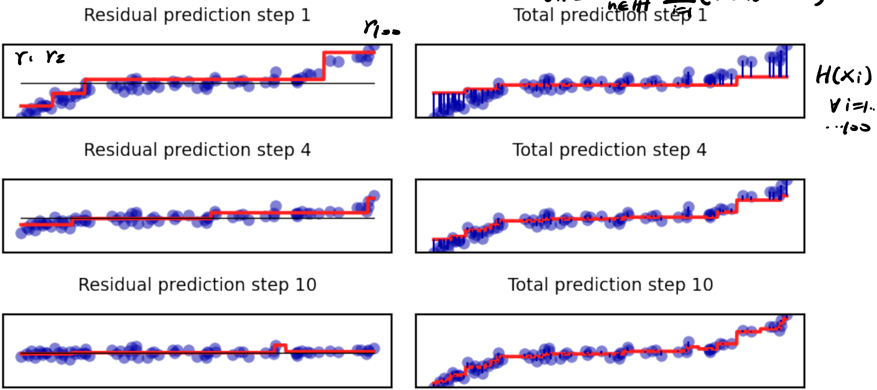
GBR with h being a decision tree

$\longrightarrow$ Gradient Boosted Regression Tree (GBRT)

Remarks:

1. Regression w.r.t. ℓ_2-norm.

$$h_{t+1} = \min_{h \in \mathcal{H}} \sum_{i=1}^{n} (h(x_i) - r_i)^2$$



https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html

Gradient Boosting for Classification

soft max.

- Predicting probability of each of K classes, namely

$$p_k(\mathbf{x}) = \frac{\exp\left(h^{(k)}(\mathbf{x})\right)}{\sum_{c=1}^{K} \exp\left(h^{(c)}(\mathbf{x})\right)} \triangleq \hat{y}^{(k)}, \quad \underline{k = 1, 2, \ldots, K}.$$

- Consider the loss $\mathcal{L}(H) = \sum_{i=1}^{n} \ell(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ (e.g. multi-class cross-entropy or KL divergence)

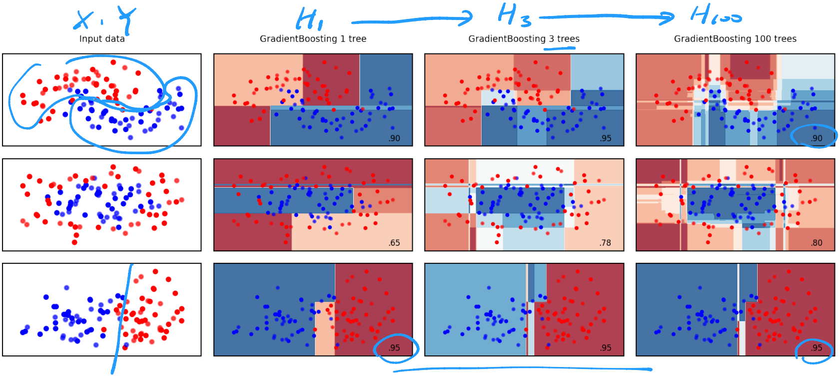
e.g. $y_1 = \left.\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}\right\} K \qquad \hat{y}_1 = \left.\begin{bmatrix} 0 \\ 0.9 \\ 0.05 \\ 0 \\ \vdots \\ 0.05 \end{bmatrix}\right\} K$

Gradient Boosting for Classification

- Predicting probability of each of K classes, namely
$p_k(\mathbf{x}) = \frac{\exp\left(h^{(k)}(\mathbf{x})\right)}{\sum_{c=1}^{K}\exp\left(h^{(c)}(\mathbf{x})\right)} \triangleq \hat{y}^{(k)}, \quad k = 1, 2, \ldots, K.$
- Consider the loss $\mathcal{L}(H) = \sum_{i=1}^{n} \ell(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ (e.g. multi-class cross-entropy or KL divergence)
- Model initialization $H^{(1)}, H^{(2)}, \ldots, H^{(K)}$ and iterate until converge or reach maximum T
  1. Calculate negative gradients for every class:
  $$-g_k(\mathbf{x}_i) = -\frac{\partial \mathcal{L}}{\partial [H^{(k)}(\mathbf{x}_i)]}, \; i = 1, 2, \ldots, n, \; k = 1, 2, \ldots, K$$
  2. Fit $h^{(k)}$ to the negative gradients $-g_k(\mathbf{x}_i)$, $k = 1, 2, \ldots, K$.
  3. Update the models via $H^{(k)} \leftarrow H^{(k)} + \alpha h^{(k)}$, $k = 1, 2, \ldots, K$.

Numerical example of GBC

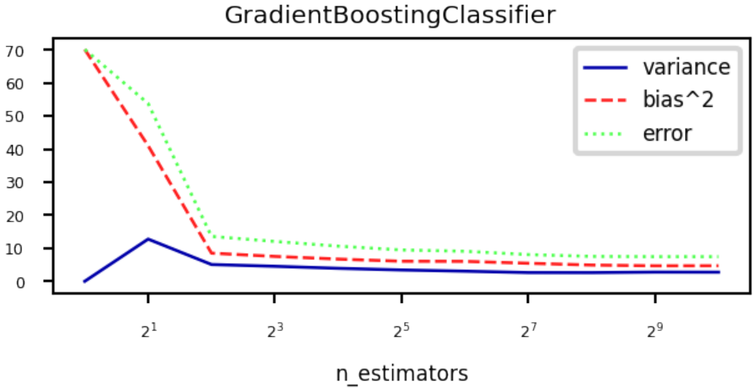
GBC with *h* being a decision tree
$\longrightarrow$ Gradient Boosted Classification Tree (GBCT)



https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html

Numerical example of GBC

- Bias-Variance analysis
  - Gradient Boosting reduces bias (and a little variance)
  - Boosting too much will eventually increase variance



https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html

History of Boosting

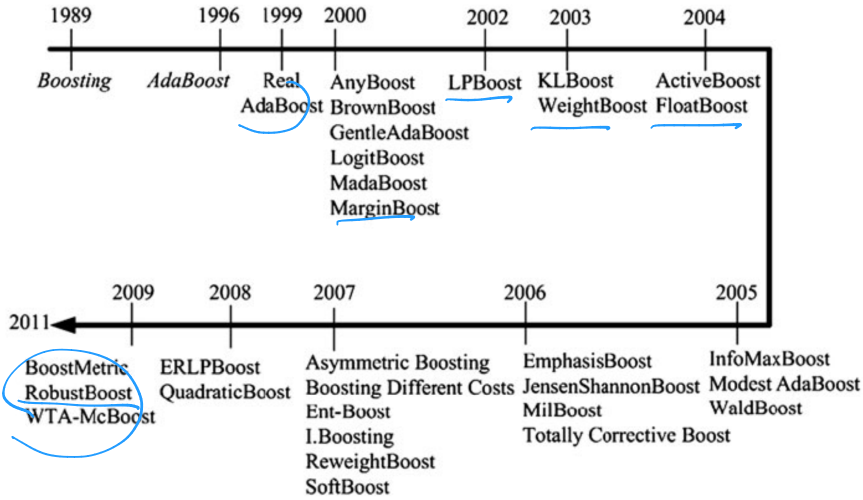


Image from A.J. Ferreira and M.A.T. Figueiredo, in Ensemble Machine Learning: Methods and Applications. 2012.

Adaptive Boosting (AdaBoost)

consider the update of the learning rate α.

$X \cdot Y$ $\quad X := (x_1 \cdots x_n)$ $\quad Y := (y_1 \cdots y_n)$. $\quad y_i \in \{-1, +1\}$ classification.

- AdaBoost (a special case of gradient boosting) uses the exponential loss, i.e.,

strong model. $\quad$ outputs: $H(x_1) \cdots H(x_n)$.

$$\mathcal{L}(H) = \sum_{i=1}^{n} \underline{e}^{-y_i H(\mathbf{x}_i)}$$

and learns α adaptively.

- The gradient is then $q_i = \frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = -y_i e^{-y_i H(\mathbf{x}_i)}$.

$H(x_i) = 0.9 \qquad y_i = 1 \qquad e^{-0.9}$

$\wedge$

$\underline{H(x_j) = 0.1} \qquad \underline{y_j = 1}. \qquad \underline{e^{-0.1}}.$

Adaptive Boosting (AdaBoost)

- AdaBoost (a special case of gradient boosting) uses the exponential loss, i.e.,

$$\mathcal{L}(H) = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$$

  and learns α adaptively.
- The gradient is then $q_i = \frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = -y_i e^{-y_i H(\mathbf{x}_i)}$.
- For convenience, let $\boxed{w_i = \frac{1}{Z} e^{-y_i H(\mathbf{x}_i)}}$, where $Z = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$ (such that $\sum_{i=1}^{n} w_i = 1$). Then w_i is the relative contribution of the training point $(\mathbf{x}_i, y_i)$ to the overall loss.

sample weight.

- We here consider a binary classification task, i.e., $y \in \{-1, +1\}$, and let $h(\mathbf{x}) \in \{-1, +1\}, \forall \mathbf{x}$.

Q: How to get h based on $\{q_i\}_{i=1}^{n}$?

Adaptive Boosting (AdaBoost)

- Note that $\mathcal{L}(H) = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$, $q_i = \frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)} = -y_i e^{-y_i H(\mathbf{x}_i)}$, $w_i = \frac{1}{Z} e^{-y_i H(\mathbf{x}_i)}$, $Z = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$ (a constant w.r.t h), $\sum_{i=1}^{n} w_i = 1$, $y \in \{-1, +1\}$, and $h(\mathbf{x}) \in \{-1, +1\}$.
- We have

Taylor approximation.

$$
\begin{aligned}
h &= \text{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{n} q_i h(\mathbf{x}_i) \\
&= \text{argmin}_{h \in \mathbb{H}} - \sum_{i=1}^{n} w_i \boxed{y_i h(\mathbf{x}_i)} \\
&= \text{argmin}_{h \in \mathbb{H}} \sum_{i: h(\mathbf{x}_i) \neq y_i} w_i - \sum_{i: h(\mathbf{x}_i) = y_i} w_i \\
&= \text{argmin}_{h \in \mathbb{H}} \sum_{i: h(\mathbf{x}_i) \neq y_i} w_i
\end{aligned}
\tag{7}
$$

$q_i = -y_i w_i \cdot Z$

$\left\{ \begin{array}{l} 1 \cdot 1 = 1 \\ 1 \cdot -1 = -1 \\ -1 \cdot 1 = -1 \\ -1 \cdot -1 = 1 \end{array} \right.$

$(h(x) \in \{-1, +1\})$

$2 \cdot (*) - 1$

$\sum_{i=1}^{n} w_i = 1 \Rightarrow$

$\boxed{\sum_{i: h(x_i) = y_i} w_i = 1 - \sum_{i: h(x_i) \neq y_i} w_i}$

$(*)$

* The last equality holds because $\sum_{i=1}^{n} w_i = 1$.

Adaptive Boosting (AdaBoost)

$$w_i = \frac{1}{Z} e^{-\overbrace{\underbrace{y_i}_{-1}}^{-1} \overbrace{\underbrace{H(x_i)}_{-1}}^{+1}} = \frac{1}{Z} e^{1}$$

$$= \frac{1}{Z} e^{-1}$$

- We have got

$$h = \mathrm{argmin}_{h \in \mathbb{H}} \sum_{i: h(\mathbf{x}_i) \neq y_i} w_i$$

$\epsilon := \sum_{i: h(\mathbf{x}_i) y_i = -1} w_i$ can be regarded as a weighted classification error, where $w_i = \frac{1}{Z} e^{-y_i H(\mathbf{x}_i)}$.
Note that a miss-classified point by H gets a larger weight.

Adaptive Boosting (AdaBoost)

- Given h, we find α via
$$\alpha = \operatorname{argmin}_{\alpha} \mathcal{L}(H + \alpha h) = \operatorname{argmin}_{\alpha} \sum_{i=1}^{n} e^{-y_i(H(\mathbf{x}_i)+\alpha h(\mathbf{x}_i))} \tag{8}$$
- Differentiate w.r.t α and equate with zero:
$$\sum_{i=1}^{n} y_i h(\mathbf{x}_i) e^{-(y_i H(\mathbf{x}_i)+\alpha y_i h(\mathbf{x}_i))} = 0.$$

Adaptive Boosting (AdaBoost)

- Given h, we find α via

$$\alpha = \text{argmin}_\alpha \mathcal{L}(H + \alpha h) = \text{argmin}_\alpha \sum_{i=1}^{n} e^{-y_i(H(\mathbf{x}_i)+\alpha h(\mathbf{x}_i))} \tag{8}$$

- Differentiate w.r.t α and equate with zero:

$$\sum_{i=1}^{n} \boxed{y_i h(\mathbf{x}_i)} e^{-(y_i H(\mathbf{x}_i)+\alpha y_i h(\mathbf{x}_i))} = 0.$$

$$\begin{cases} w_i = \frac{1}{Z} e^{-y_i h(x_i)} \\ \varepsilon := \sum_{i: h(x_i) \neq y_i} w_i \end{cases}$$

It follows that

$$\sum_{i:h(\mathbf{x}_i)y_i=1} \boxed{e^{-(y_i H(\mathbf{x}_i)}} {}^{+\alpha y_i h(\mathbf{x}_i))} - \sum_{i:h(\mathbf{x}_i)y_i=-1} e^{-(y_i H(\mathbf{x}_i)+\alpha y_i h(\mathbf{x}_i))} = 0$$

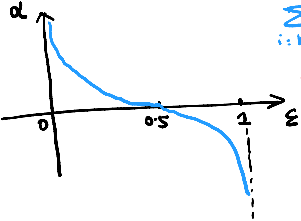
$w_i \cdot Z$

$$\sum_{i:h(\mathbf{x}_i)y_i=1} w_i e^{-\alpha} - \sum_{i:h(\mathbf{x}_i)y_i=-1} w_i e^{\alpha} = 0$$

We have $\boxed{(1-\epsilon)e^{-\alpha} - \epsilon e^{\alpha} = 0}$, $e^{2\alpha} = \frac{1-\epsilon}{\epsilon}$, and get

$$\alpha = \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}. \tag{9}$$

$$\alpha = \frac{1}{2}\log\frac{1-\varepsilon}{\varepsilon}$$



$$1-\varepsilon := \sum_{i:\, h(x_i)=y_i} w_i$$

i.e. the number of correctly predicted labels

$\varepsilon > 0.5$:= the new model is worse than random guessing.

Adaptive Boosting (AdaBoost)

- Pseudo code of AdaBoost for binary classification

Require: $\ell, \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, $\mathbb{H}$

$H_0 = 0$

$w_i = 1/n,\ i = 1, 2, , \ldots, n$

for $t = 0 : T - 1$ **do**

  $h_{t+1} = \mathrm{argmin}_{h \in \mathbb{H}} \sum_{i: h(\mathbf{x}_i) \neq y_i} w_i$

  $\epsilon = \sum_{i: h(\mathbf{x}_i) \neq y_i} w_i$

  if $\epsilon < 1/2$ **then**

    $\alpha = \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}$

    $H_{t+1} \leftarrow H_t + \alpha h_{t+1}$

    $w_i = \frac{1}{Z} \exp\left(-y_i H_{t+1}(\mathbf{x}_i)\right),\ i = 1, 2, \ldots, n$

  else

    Return H_t

  end if

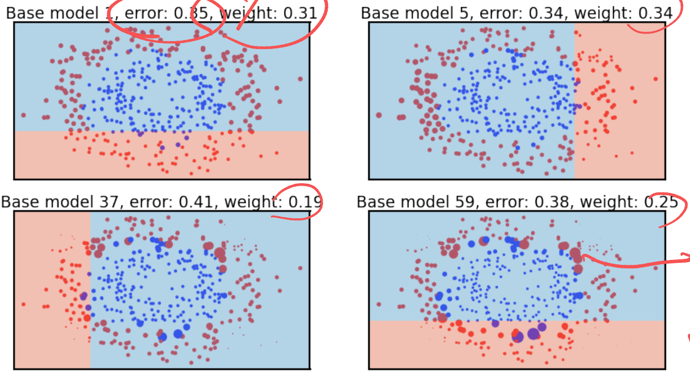
end for

Ensure: H_T

Numerical example of AdaBoost

- Binary classification on synthetic data: performance of the base models

step size. $\to \alpha_t$ $\quad W_t :=$ sample weight.



i

w_i is high.

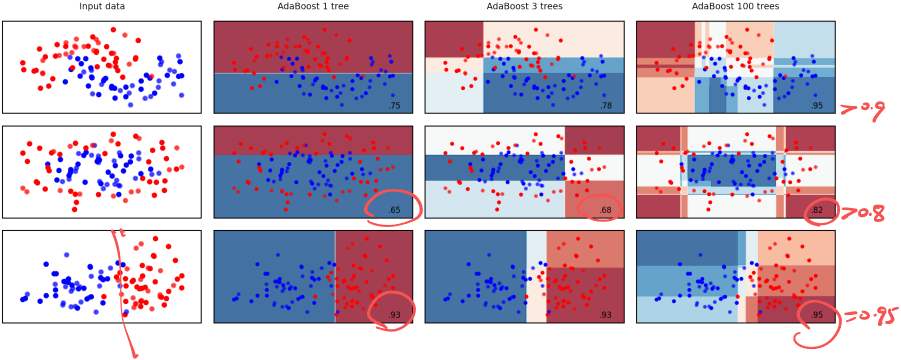
w_i

The size of dot indicates the weight of the data point.

https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html

Numerical example of AdaBoost

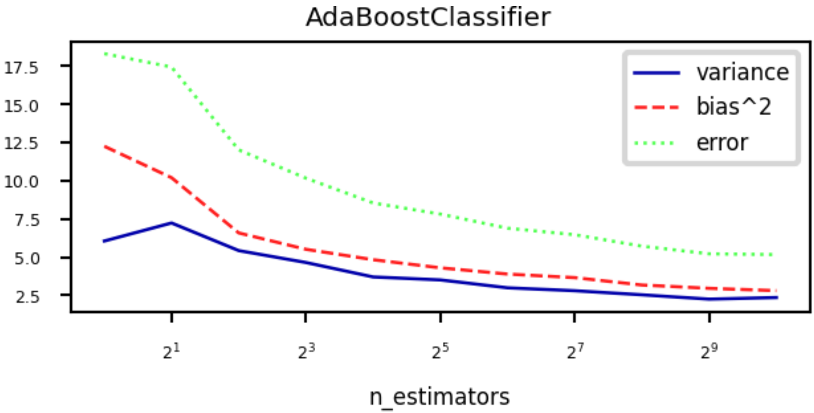
- Binary classification on synthetic data: performance of the ensemble classifier



https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html

Numerical example of AdaBoost

- Bias-Variance analysis
  - AdaBoost reduces bias (and a little variance)
  - Boosting too much will eventually increase variance



https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html

Comparison between GradientBoost and AdaBoost

special instance.

- AdaBoost minimizes the exponential loss function that can make the algorithm sensitive to outliers. With Gradient Boosting, any differentiable loss function can be utilized. Gradient Boosting is more robust to outliers than AdaBoost.
- AdaBoost up-weights observations that were misclassified before. Gradient boosting identifies difficult observations by large residuals computed in the previous iterations. ℓ_2 - norm loss
- AdaBoost was mainly designed for binary classification problems and can be utilized to boost the performance of decision trees. Gradient Boosting is used to solve the differentiable loss function problem. The technique can be used for both classification and regression problems.
- AdaBoost can be regarded as a special case of Gradient Boosting.

.

Discussion on boosting

- Boosting is a great way to turn a week learner into a strong leaner.
- It defines a whole family of algorithms, including Gradient Boosting, AdaBoost, LogitBoost, and many others.
- Gradient boosted decision tree is very useful for learning to rank (to be introduced in future).
- AdaBoost is an extremely powerful algorithm, that turns any weak learner that can classify any weighted version of the training set with below 0.5 error into a strong learner whose training error decreases exponentially.

A few slides are adapted from `https://www.cs.cornell.edu/courses/cs4780/2022fa/lectures/lecturenote19.html`

Extreme Gradient Boosting (XGBoost)[1,2] (optional)

"Scalable, Portable and Distributed Gradient Boosting (GBDT, GBRT or GBM) Library, for Python, R, Java, Scala, C++ and more. Runs on single machine, Hadoop, Spark, Dask, Flink, and DataFlow."

- A faster version of gradient boosting.
- Normal regression trees: split to minimize squared loss of leaf predictions.
- XGBoost trees only fit residuals: split so that residuals in leaf are more similar.
- Gradient descent sped up by using the second derivative of the loss function.
- Strong regularization by pre-pruning the trees.
- Column and row are randomly subsampled when computing splits.
- ...

[1] https://github.com/dmlc/xgboost

[2] https://xgboost.readthedocs.io/en/stable/

LightGBM[3][4] (optional)

"A fast, distributed, high performance gradient boosting (GBT, GBDT, GBRT, GBM or MART) framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks."

- Uses gradient-based sampling
- Use all instances with large gradients/residuals
- Randomly sample instances with small gradients, ignore the rest
- Intuition: samples with small gradients are already well-trained.
- Requires adapted information gain criterion
- Does smarter encoding of categorical features

[3] https://lightgbm.readthedocs.io/en/v3.3.2/

[4] https://github.com/microsoft/LightGBM

CatBoost[5][6] (optional)

"A fast, scalable, high performance Gradient Boosting on Decision Trees library, used for ranking, classification, regression and other machine learning tasks for Python, R, Java, C++. Supports computation on CPU and GPU."

- Another fast boosting technique
- Optimized for categorical variables
- Uses bagged and smoothed version of target encoding
- Uses symmetric trees: same split for all nodes on a given level aka
- Allows monotonicity constraints for numeric features
- Model must be be a non-decreasing function of these features
- Lots of tooling (e.g. GPU training)
- Focusing on optimizing decision trees for categorical variables

[5] https://catboost.ai/

[6] https://github.com/catboost/catboost

Stacking

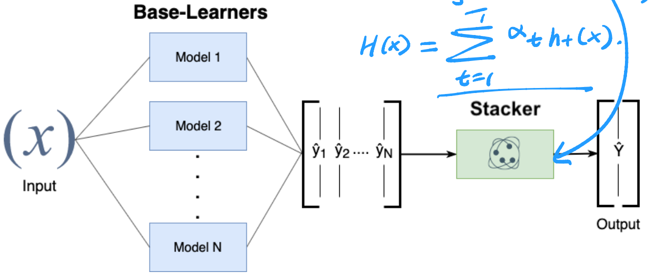
- Choose *M* different base-models, generate predictions
- Stacker (meta-model) learns mapping between predictions and correct label

function,

- Stacking is a type of meta-learning[7]

A generalization of

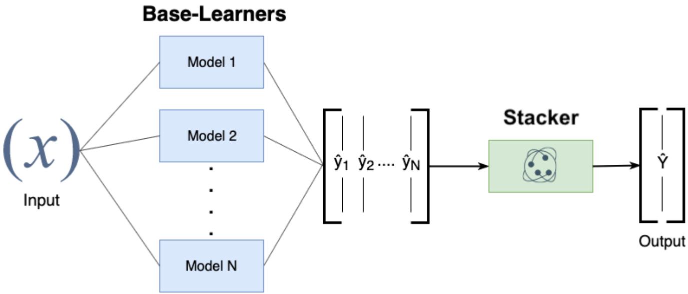
$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$



https://ml-course.github.io/master/notebooks/05%20-%20Ensemble%20Learning.html

[7] "meta-learning" is also referred as "learning to learn". See the following materials: [1] Hospedales et al. 2020. Meta-Learning in Neural Networks: A Survey; [2] a blog `https://lilianweng.github.io/posts/2018-11-30-meta-learning/`

Stacking



- Stacking can be repeated: multi-level stacking
- Popular stackers: linear models (fast) and gradient boosting (accurate)
- Models need to be sufficiently different, be experts at different parts of the data
- Can be very accurate, but also very slow to predict

Summary for ensemble learning

- Bagging
  - Vanilla bagging
  - Random forest
- Boosting
  - Gradient boosting
  - AdaBoost
  - ...
- Stacking
- Other ensemble learning techniques
  - Bayes optimal classifier
  - Bayesian model averaging
  - Any combination of different ensembling techniques
  - Mixture of experts
  - ...

Learning outcomes

- Understand the main ideas of bagging, boosting, and stacking
- Understand the algorithms of gradient boosting and AdaBoost
- Know the connection and difference between gradient boosting and AdaBoost
- Be able to utilize some packages of ensemble learning such as XGBoost to solve real problems