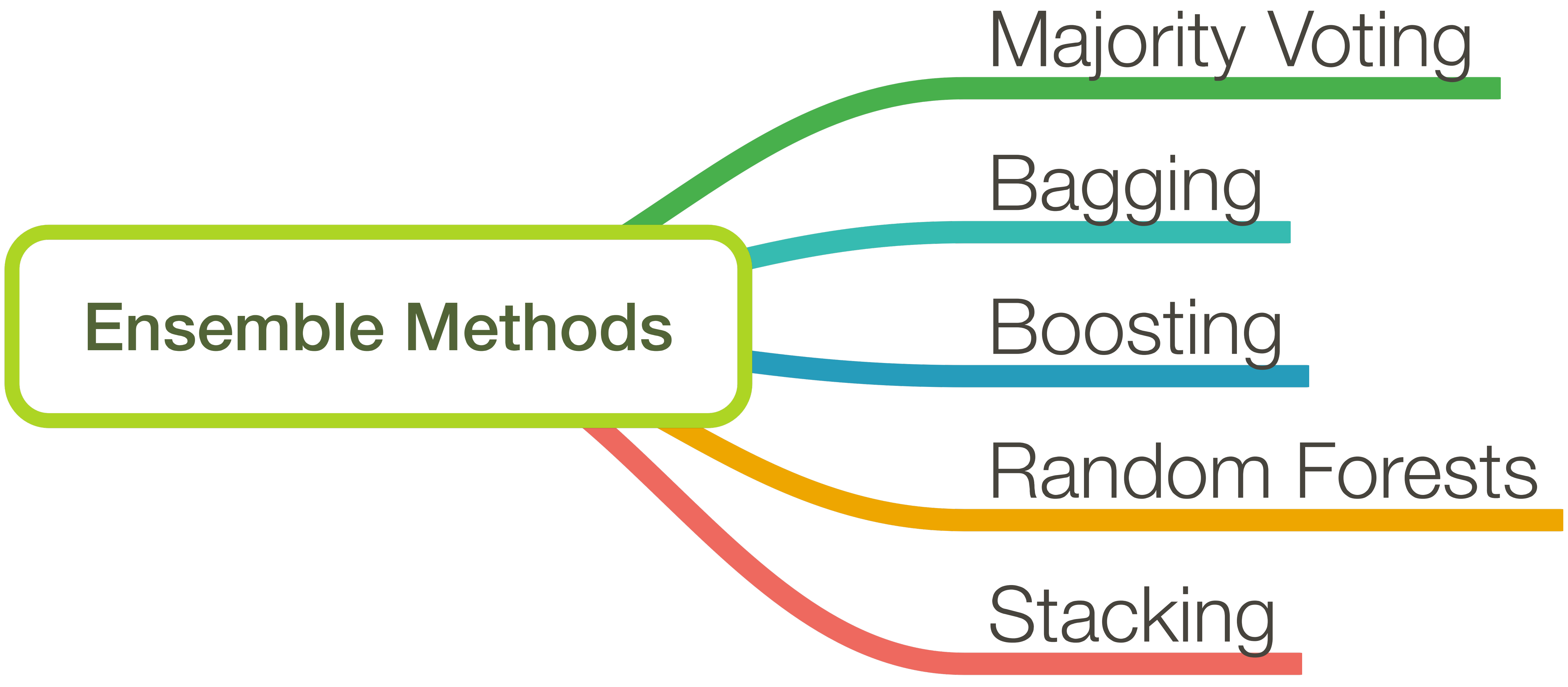
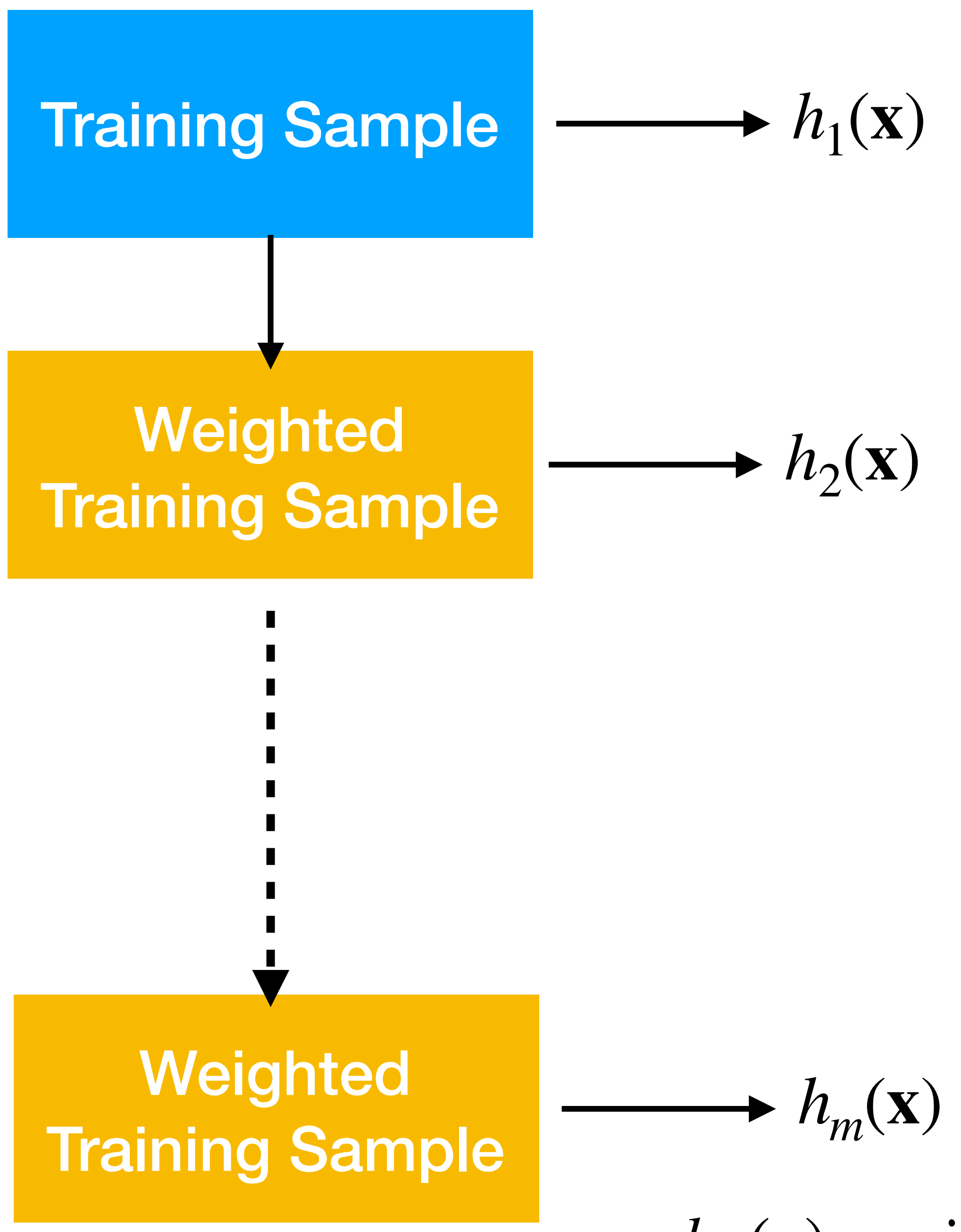
Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

Boosting

General Boosting



$$h_m(\mathbf{x}) = sign\left(\sum_{j=1}^{m} w_j \, h_j(\mathbf{x})\right) \quad \text{for} \quad h(\mathbf{x}) \in \{-1,1\}$$

or $$h_m(\mathbf{x}) = \arg\max_i \left(\sum_{j=1}^{m} w_j \, \mathbf{1}[h_j(\mathbf{x}) = i]\right) \quad \text{for} \quad h(\mathbf{x}) = i, \qquad i \in \{1,\ldots,n\}$$

General Boosting

- Initialize a weight vector with uniform weights

- Loop:
  - Apply weak learner* to weighted training examples (instead of orig. training set, may draw bootstrap samples with weighted probability)

  - Increase weight for misclassified examples

- (Weighted) majority voting on trained classifiers

* a learner slightly better than random guessing

AdaBoost

Algorithm 1 AdaBoost

Initialize k: the number of AdaBoost rounds
Initialize $\mathcal{D}$: the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, ..., \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
Initialize $w_1(i) = 1/n, \quad i = 1, ..., n, \ \mathbf{w}_1 \in \mathbb{R}^n$

for r=1 to k **do**
    For all $i : \mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]
    $h_r := FitWeakLearner(\mathcal{D}, \mathbf{w}_r)$
    $\epsilon_r := \sum_i w_r(i) \, \mathbf{1}(h_r(i) \neq y_i)$ [compute error]
    if $\epsilon_r > 1/2$ then stop
    $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]
    $w_{r+1}(i) := w_r(i) \times \begin{cases} \mathrm{e}^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ \mathrm{e}^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$
Predict: $h_{ens}(\mathbf{x}) = \arg\max_j \sum_r^k \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$

AdaBoost

0/1 loss $\mathbf{1}\left(h_r(i) \neq y_i\right) = \begin{cases} 0 & \textbf{if } h_r(i) = y_i \\ 1 & \textbf{if } h_r(i) \neq y_i \end{cases}$

Algorithm 1 AdaBoost

Initialize k: the number of AdaBoost rounds

Initialize $\mathcal{D}$: the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]}\rangle, ..., \mathbf{x}^{[n]}, y^{[n]}\rangle\}$

Initialize $w_1(i) = 1/n, \quad i = 1, ..., n, \; \mathbf{w}_1 \in \mathbb{R}^n$

for r=1 to k **do**

$\quad$ For all i : $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ $\quad$ [normalize weights]

$\quad$ $h_r := FitWeakLearner(\mathcal{D}, \mathbf{w}_r)$

$\quad$ $\epsilon_r := \sum_i w_r(i)\, \mathbf{1}(h_r(i) \neq y_i)$ $\quad$ [compute error]

$\quad$ if $\epsilon_r > 1/2$ then stop

$\quad$ $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ $\quad$ [small if error is large and vice versa]

$\quad$ $w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$

Predict: $h_{ens}(\mathbf{x}) = \arg\max_j \sum_r^k \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$

Assumes binary classification problem

AdaBoost

Algorithm 1 AdaBoost

Initialize k: the number of AdaBoost rounds

Initialize $\mathcal{D}$: the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, ..., \mathbf{x}^{[n]}, y^{[n]} \rangle\}$

Initialize $w_1(i) = 1/n, \quad i = 1, ..., n, \ \mathbf{w}_1 \in \mathbb{R}^n$

Estimator weight

for r=1 to k **do**

  For all $i : \mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]

  $h_r := FitWeakLearner(\mathcal{D}, \mathbf{w}_r)$

  $\epsilon_r := \sum_i w_r(i) \, \mathbf{1}(h_r(i) \neq y_i)$ [compute error]

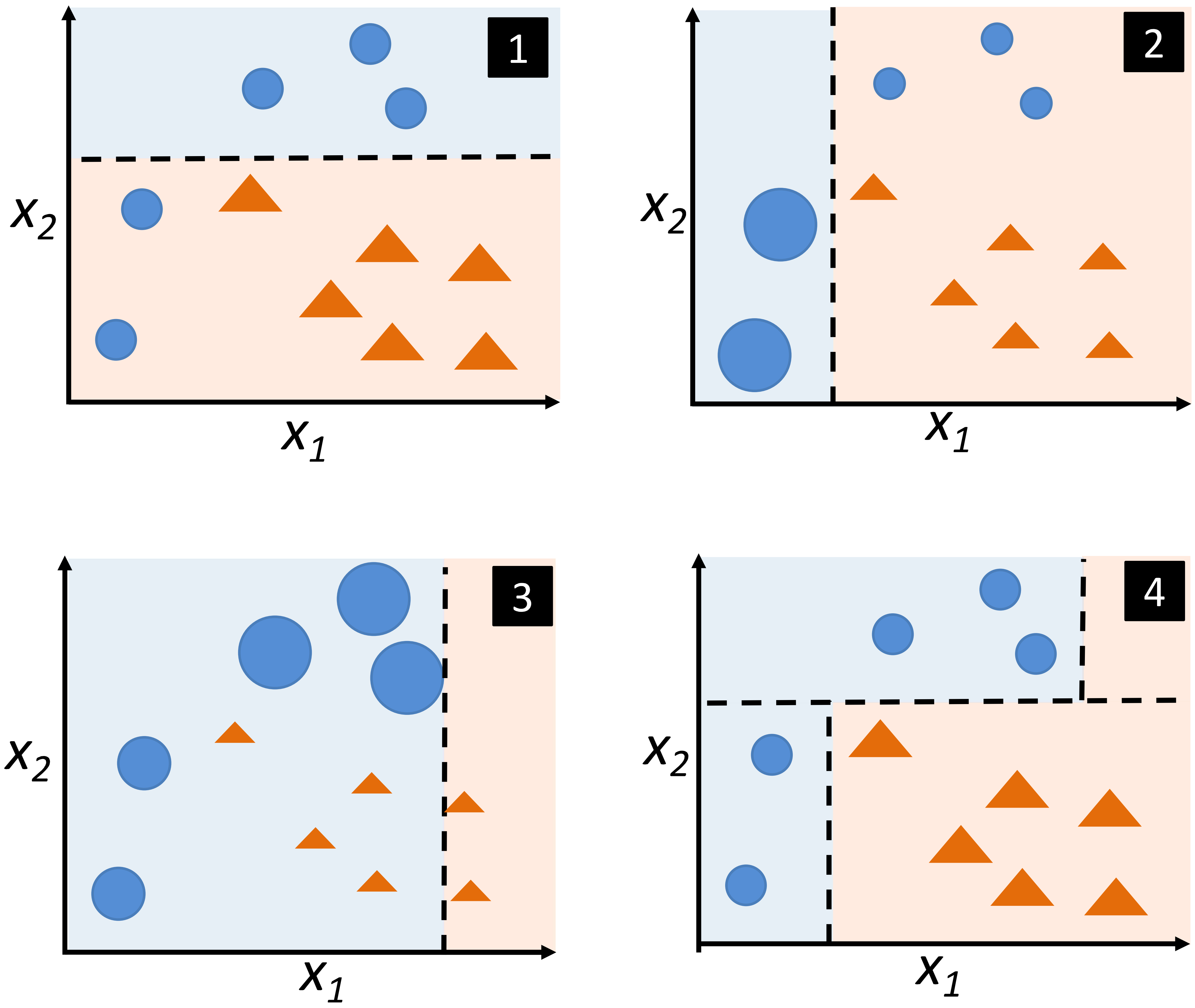
  if $\epsilon_r > 1/2$ then stop

  $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]

$$w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$$

Predict: $h_{ens}(\mathbf{x}) = \arg\max_j \sum_r^k \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$

Sample weight


1
2
3
4
x_1
x_2

Gradient Boosting

Gradient Boosting

Gradient boosting is somewhat similar to AdaBoost:

- trees are fit sequentially to improve error of previous trees
- boost weak learners to a strong learner

The way how the trees are fit sequentially differs in AdaBoost and Gradient Boosting, though ...

Gradient Boosting -- Conceptual Overview

- **Step 1:** Construct a base tree (just the root node)
- **Step 2:** Build next tree based on errors of the previous tree
- **Step 3:** Combine tree from step 1 with trees from step 2. Go back to step 2.

Algorithm 1 Gradient Boosting

Initialize T: the number of trees for gradient boosting rounds

Initialize $\mathcal{D}$: the training dataset, $\{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{n}$

Choose $L\big(y^{(i)}, h(\mathbf{x}^{(i)})\big)$, a differentiable loss function

Step 1: Initialize model $h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L\big(y^{(i)}, \hat{y}\big)$ [root node]

Step 2:

for t=1 to T **do**

    A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x}) = h_{t-1}(\mathbf{x})}$, for $i = 1$ to n

    B. Fit tree to r_{i_t} values, and create terminal nodes $R_{j,t}$ for $j = 1, ..., J_t$.

    C.

    for j=1 to J_t **do**

        $\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\big(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y}\big)$

    D. Update $h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t}\, \mathbb{I}\big(\mathbf{x} \in R_{j,t}\big)$

Step 3: Return $h_t(\mathbf{x})$

Gradient Boosting -- Conceptual Overview
--> A Regression-based Example

In million US Dollars

x1# Rooms	x2=City	x3=Age	y=Price
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunakee	10	0.1

- **Step 1:** Construct a base tree (just the root node)

$$\hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875$$

Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **<u>Step 2:</u>** Build next tree based on errors of the previous tree

First, compute (pseudo) residuals: $r_1 = y_1 - \hat{y}_1$

In million US Dollars

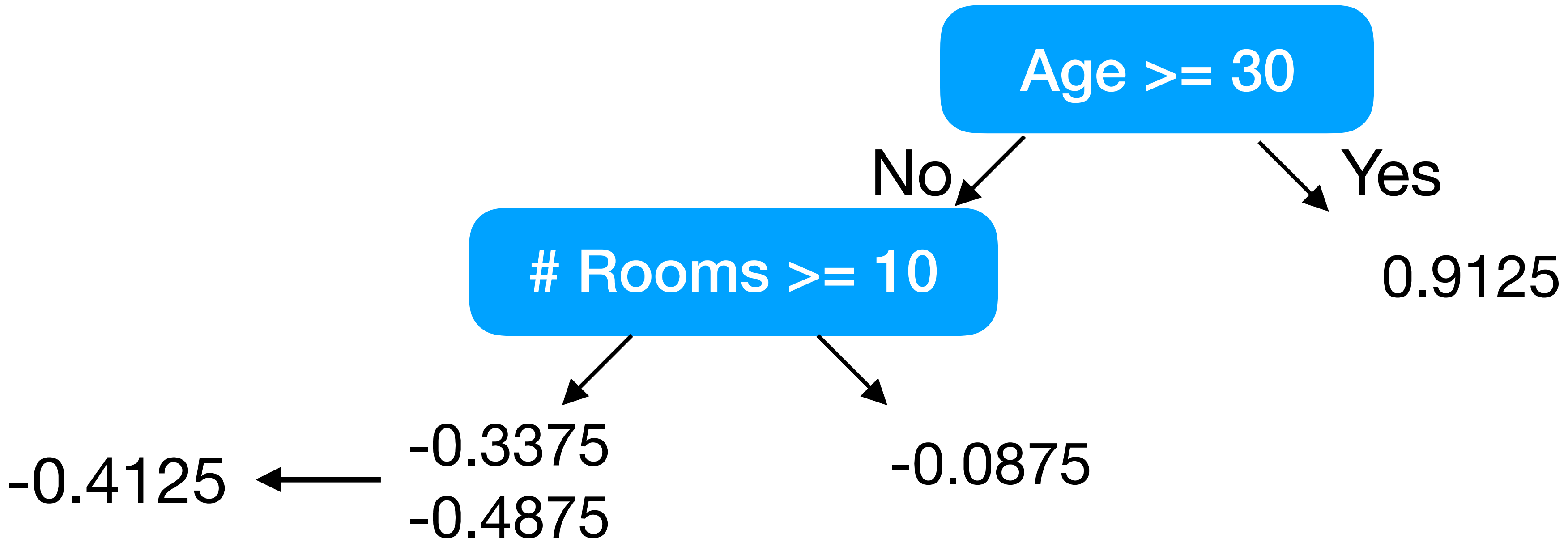
x1#	x2=City	x3=Age	y=Price	r1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **Step 2:** Build next tree based on errors of the previous tree

Then, create a tree based on $x_1, \ldots, x_m$ to fit the residuals

x1#	x2=City	x3=Age	y=Price	r1=Residual
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

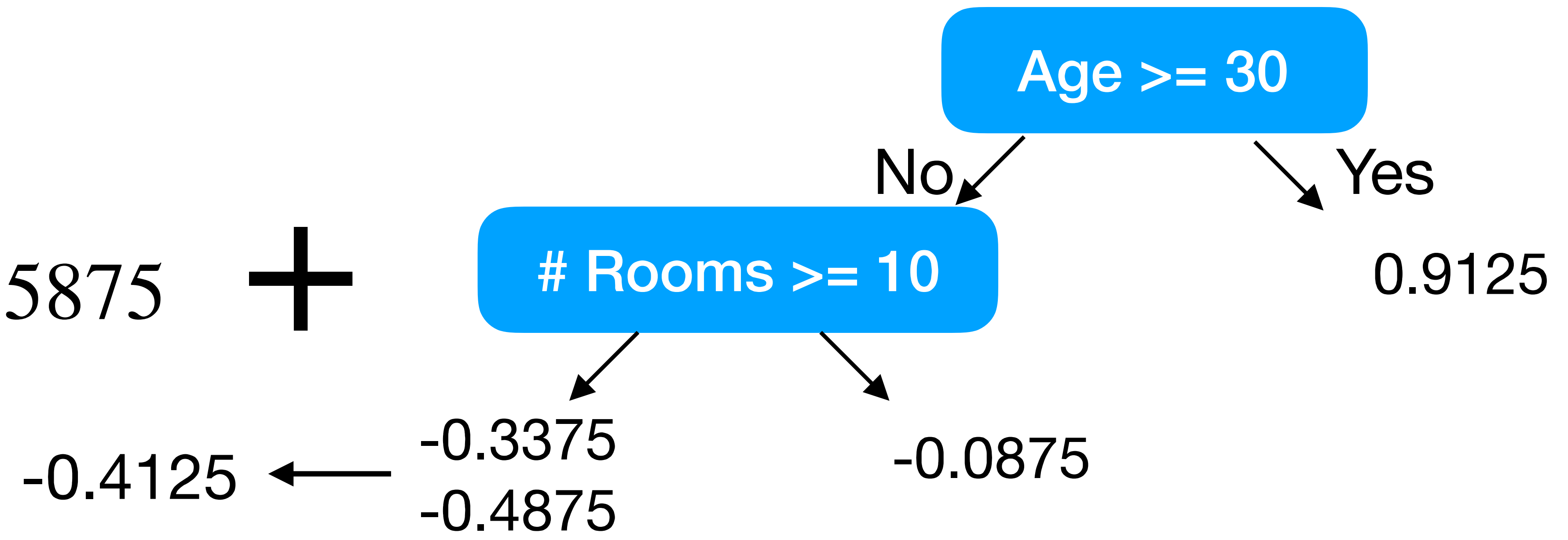


Gradient Boosting -- Conceptual Overview
--> A Regression-based Example

- **Step 3:** Combine tree from step 1 with trees from step 2

x1#	x2=City	x3=Age	y=Price	r=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

$$\hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875 \quad +$$



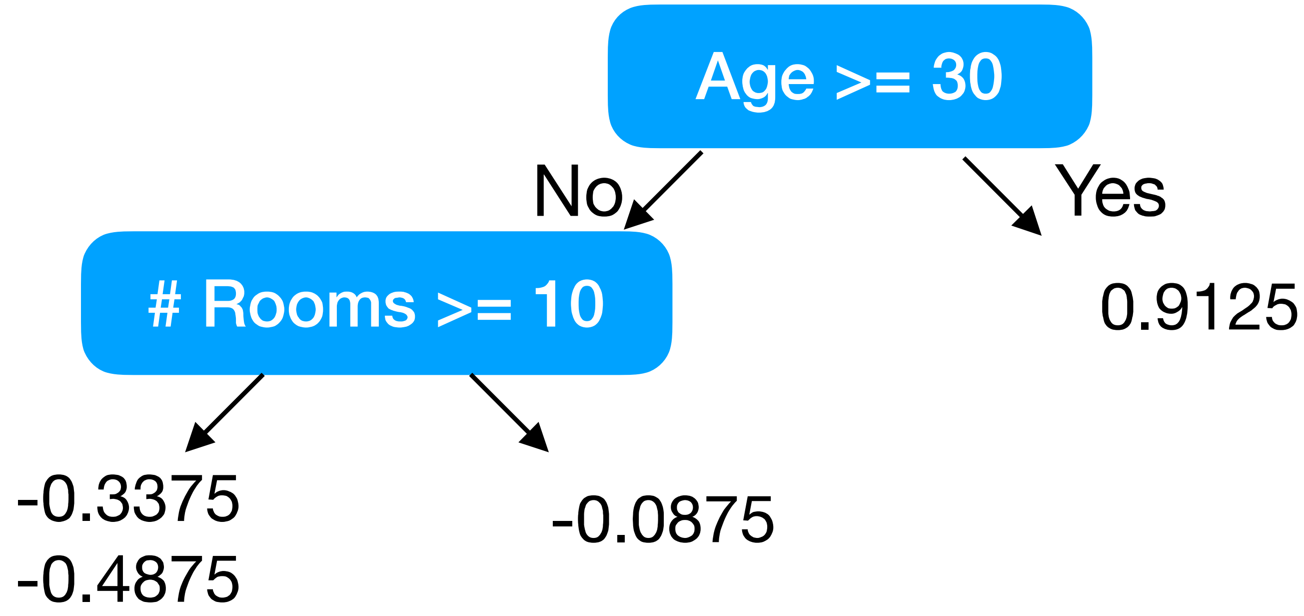
Gradient Boosting -- Conceptual Overview
--> A Regression-based Example

- **<u>Step 3:</u>** Combine tree from step 1 with trees from step 2

E.g., predict Lansing

x1#	x2=City	x3=Age	y=Price	r=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunakee	10	0.1	0.1 - 0.5875 = -0.4875

$$\hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875 \quad +$$



-0.4125 ←

E.g., predict Lansing

$$0.5875 + \alpha \times (-0.4125)$$

where α learning rate between 0 and 1 (if $\alpha = 1$, low bias but high variance)

Gradient Boosting -- Algorithm Overview

<u>Step 0:</u> Input data $\{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{n}$

Differentiable Loss function $L\big(y^{(i)}, h(\mathbf{x}^{(i)})\big)$

<u>Step 1:</u> Initialize model $h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L\big(y^{(i)}, \hat{y}\big)$

<u>Step 2:</u> for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = - \left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})} \right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

...

Gradient Boosting -- Algorithm Overview

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\big(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y}\big)$$

D. Update $h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t}\, \mathbb{I}\big(\mathbf{x} \in R_{j,t}\big)$

Step 3: Return $h_t(\mathbf{x})$

Gradient Boosting -- Algorithm Overview Discussion

Step 0: Input data $\{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{n}$

Differentiable Loss function $L\left(y^{(i)}, h(\mathbf{x}^{(i)})\right)$

E.g., Sum-squared error in regression

$$SSE' = \frac{1}{2}\left(y^{(i)} - h(\mathbf{x}^{(i)})\right)^2$$

$$\frac{\partial}{\partial h(\mathbf{x}^{(i)})} \frac{1}{2}\left(y^{(i)} - h(\mathbf{x}^{(i)})\right)^2 \quad \text{[chain rule]}$$

$$= 2 \times \frac{1}{2}\left(y^{(i)} - h(\mathbf{x}^{(i)})\right) \times (0 - 1) = -\left(y^{(i)} - h(\mathbf{x}^{(i)})\right)$$

[neg. residual]

Gradient Boosting -- Algorithm Overview Discussion

Step 1: Initialize model $h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}\right)$

pred. target

turns out to be the average (in regression)

$$\frac{1}{n} \sum_{i=1}^{n} y^{(i)}$$

Gradient Boosting -- Algorithm Overview Discussion

Loop to make T trees (e.g., $T=100$)

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

pseudo residual of the t-th tree and i-th example

Derivative of the loss function

Gradient Boosting -- Algorithm Overview Discussion

Loop to make T trees (e.g., $T=100$)

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = - \left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})} \right]_{h(\mathbf{x}) = h_{t-1}(\mathbf{x})}$

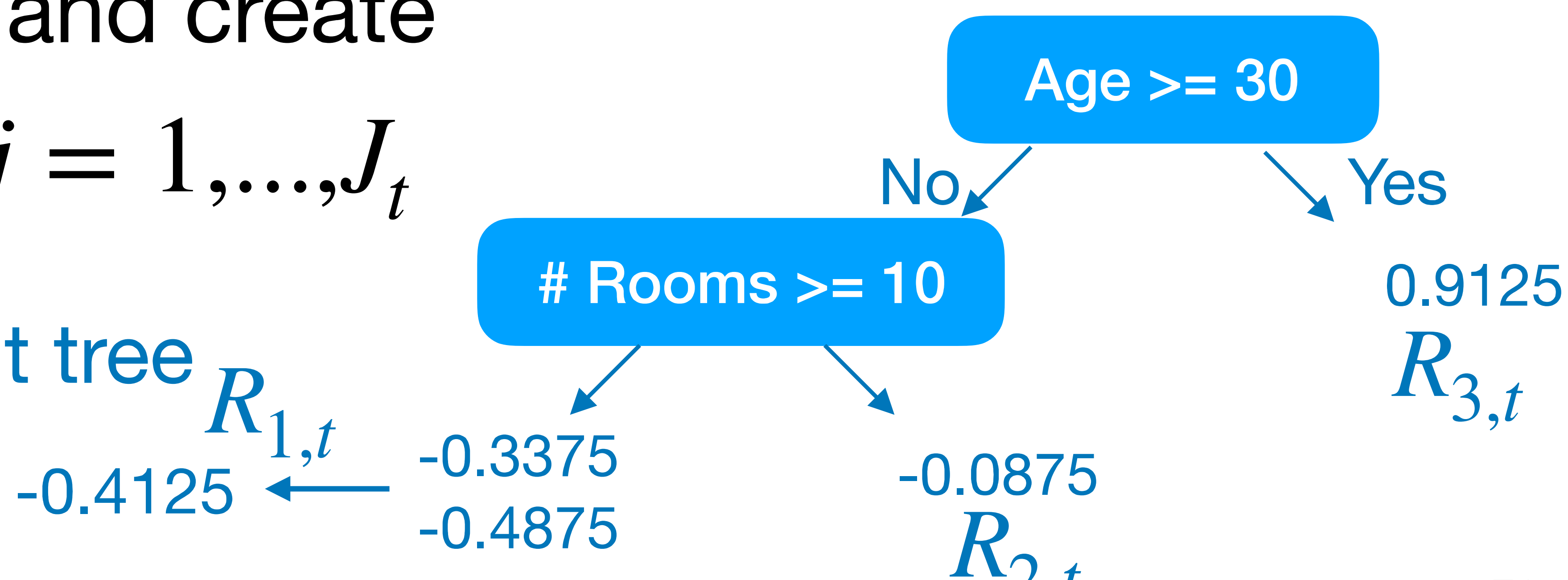
for $i = 1$ to n

pseudo residual of the t-th tree and i-th example

Derivative of the loss function

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1, \ldots, J_t$

Use features in dataset to fit tree



Gradient Boosting -- Algorithm Overview Discussion

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\left(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y}\right)$$

Compute the residual for each leaf node

Only consider examples at that leaf node

Like step 1 but add previous prediction

Gradient Boosting -- Algorithm Overview Discussion

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\left(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y}\right)$$

D. Update $h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t}\, \mathbb{I}\left(\mathbf{x} \in R_{j,t}\right)$

learning rate between 0 and 1 (usually 0.1)

Summation just in case examples end up in multiple nodes

Gradient Boosting -- Algorithm Overview Discussion

For prediction, combine all T trees, e.g.,

$$h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}\right)$$

$$+\alpha\ \hat{y}_{j,t=1} \quad = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\left(y^{(i)}, h_{(t=1)-1}(\mathbf{x}^{(i)}) + \hat{y}\right)$$

...

$$+\alpha\ \hat{y}_{j,T} \quad = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\left(y^{(i)}, h_{T-1}(\mathbf{x}^{(i)}) + \hat{y}\right)$$

Gradient Boosting -- Algorithm Overview Discussion

For prediction, combine all *T* trees, e.g.,

$$h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}\right)$$

$$+\alpha \ \hat{y}_{j,t=1}$$

The idea is that we decrease the pseudo residuals by a small amount at each step

$$\ldots$$

$$+\alpha \ \hat{y}_{j,T}$$

XGBoost

Summary and Main Points:

- scalable implementation of gradient boosting
- Improvements include: regularized loss, sparsity-aware algorithm, weighted quantile sketch for approximate tree learning, caching of access patterns, data compression, sharding
- Decision trees based on CART
- Regularization term for penalizing model (tree) complexity
- Uses second order approximation for optimizing the objective
- Options for column-based and row-based subsampling
- Single-machine version of XGBoost supports the exact greedy algorithm

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM.

Stacking

Stacking Algorithm

Wolpert, David H. "Stacked generalization." Neural networks 5.2 (1992): 241-259.

Algorithm 19.7 Stacking

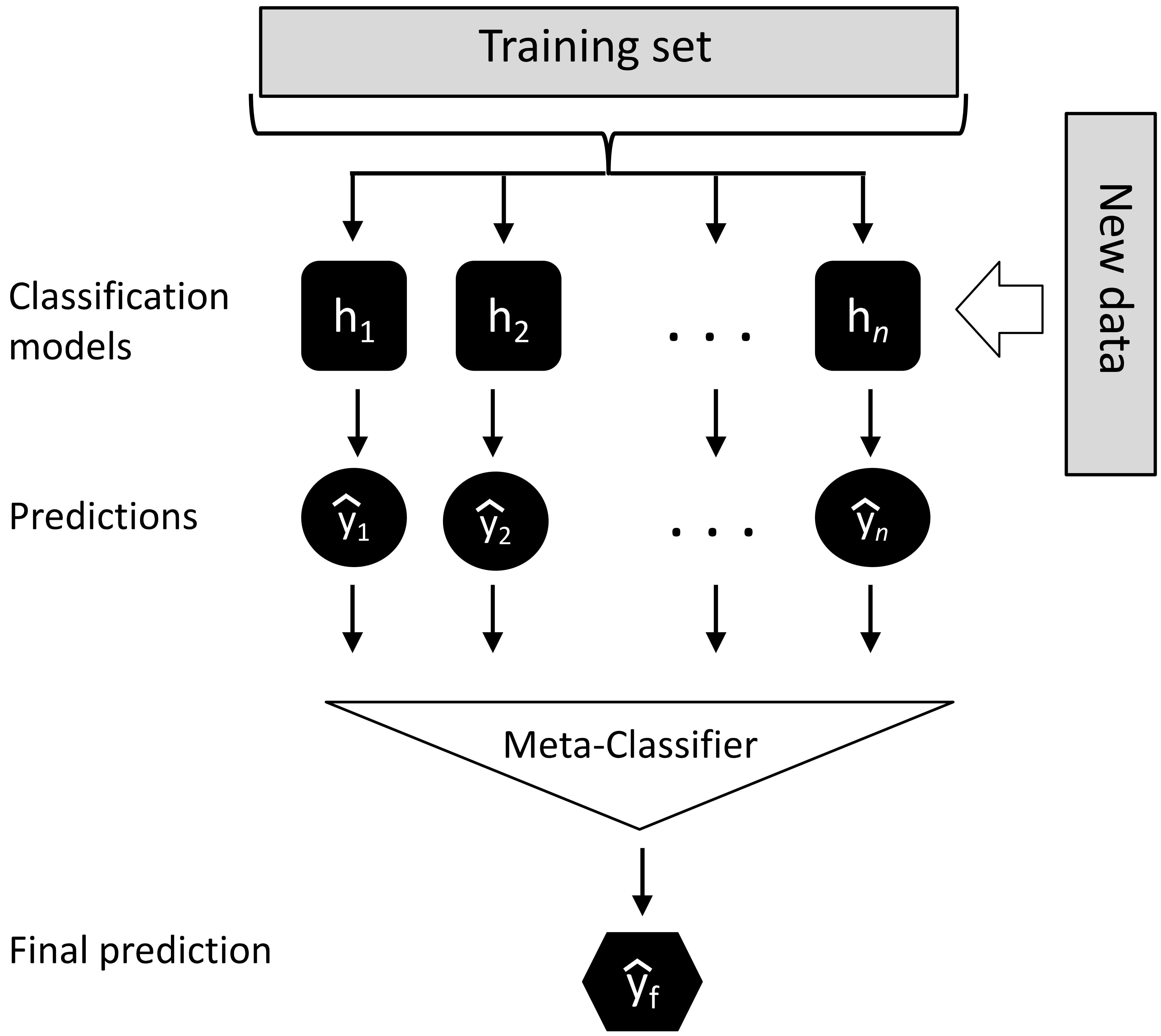
Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{m}$ ($\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathcal{Y}$)

Output: An ensemble classifier H

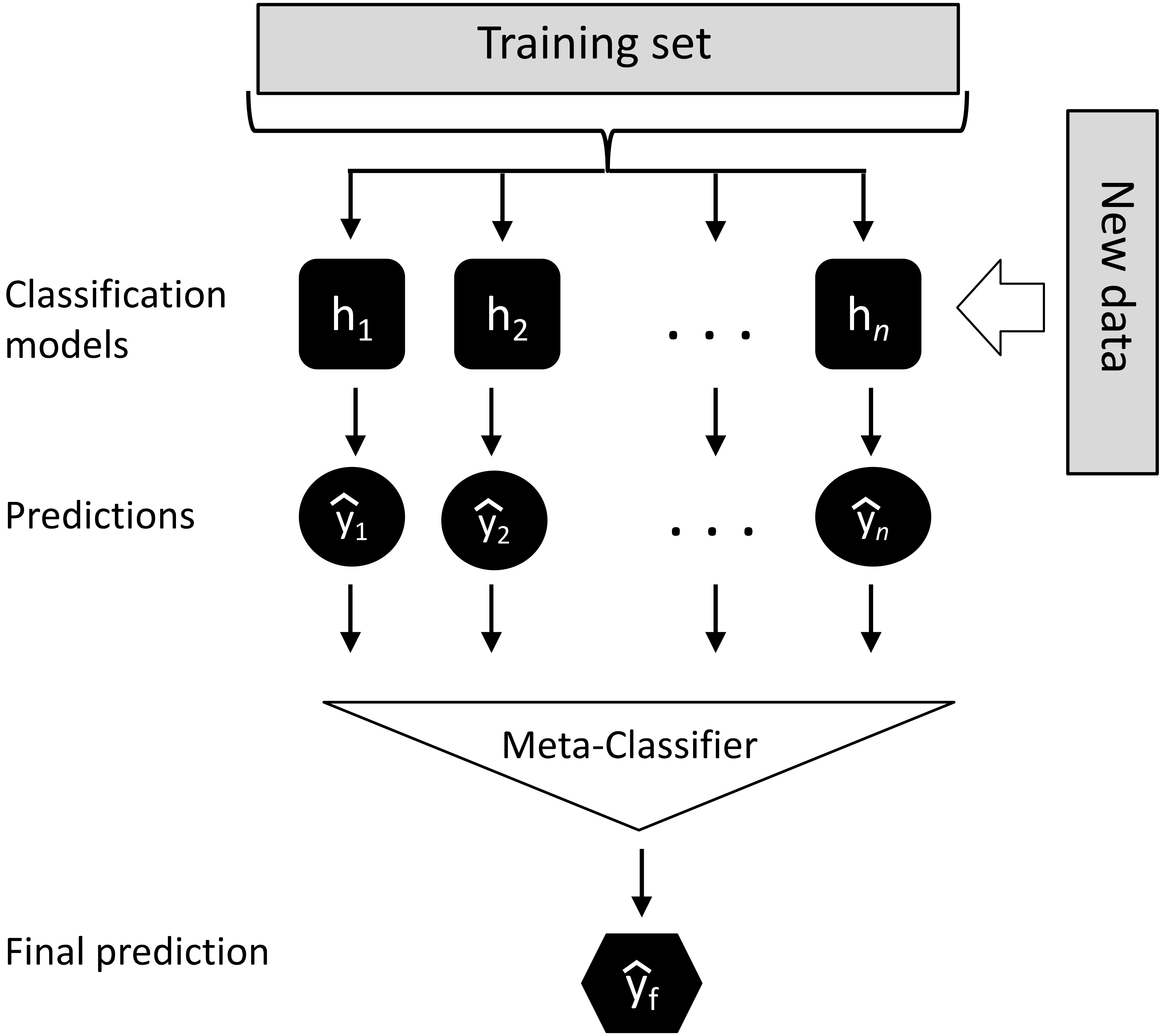
```
Step 1: Learn first-level classifiers
for t ← 1 to T do
    Learn a base classifier h_t based on D
end for
Step 2: Construct new data sets from D
for i ← 1 to m do
    Construct a new data set that contains {x'_i, y_i}, where x'_i = {h_1(x_i), h_2(x_i), ..., h_T(x_i)}
end for
Step 3: Learn a second-level classifier
Learn a new classifier h' based on the newly constructed data set
return H(x) = h'(h_1(x), h_2(x), ..., h_T(x))
```

Tang, J., S. Alelyani, and H. Liu. "Data Classification: Algorithms and Applications." Data Mining and Knowledge Discovery Series, CRC Press (2015): pp. 498-500.

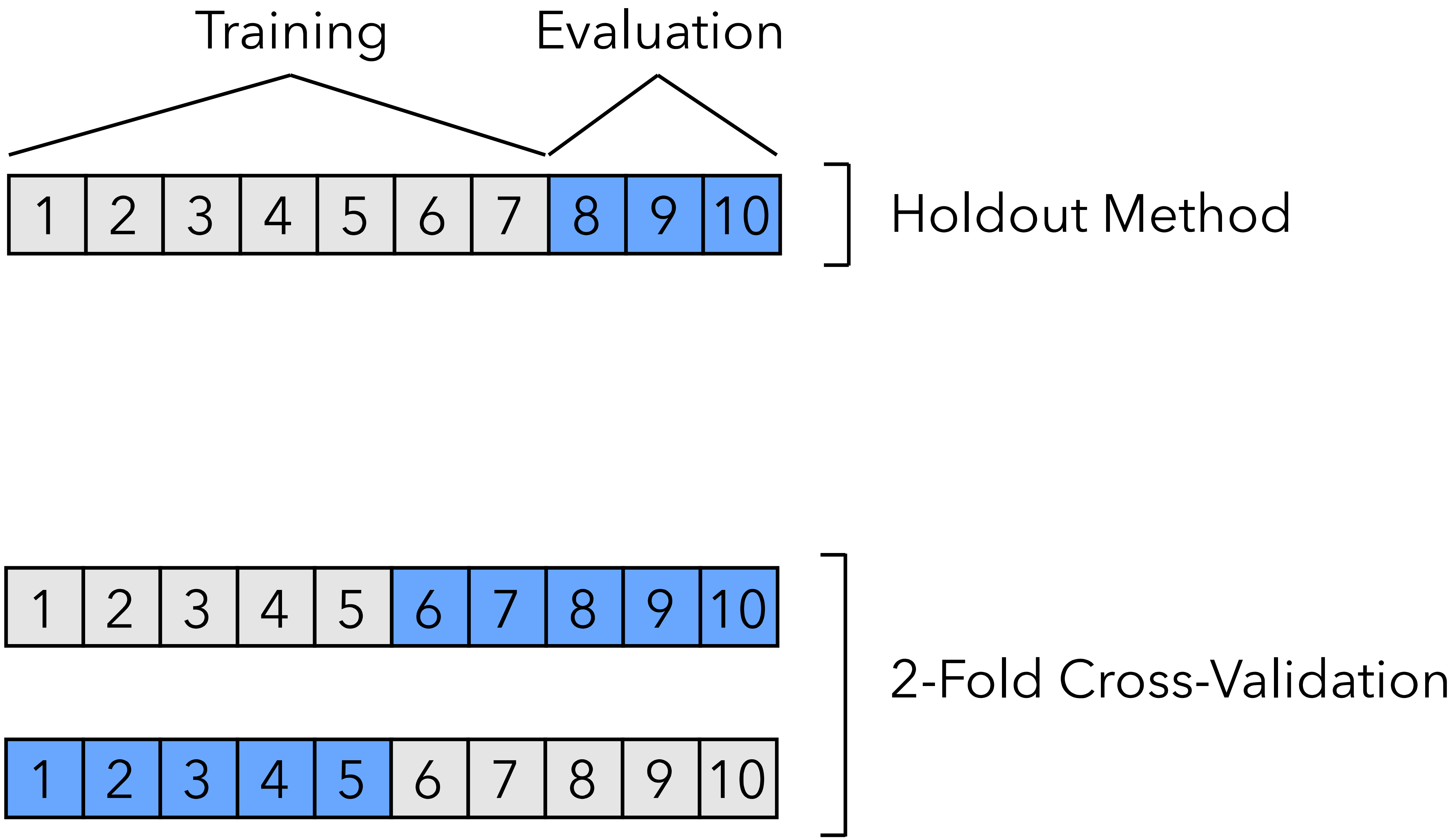
Stacking Algorithm



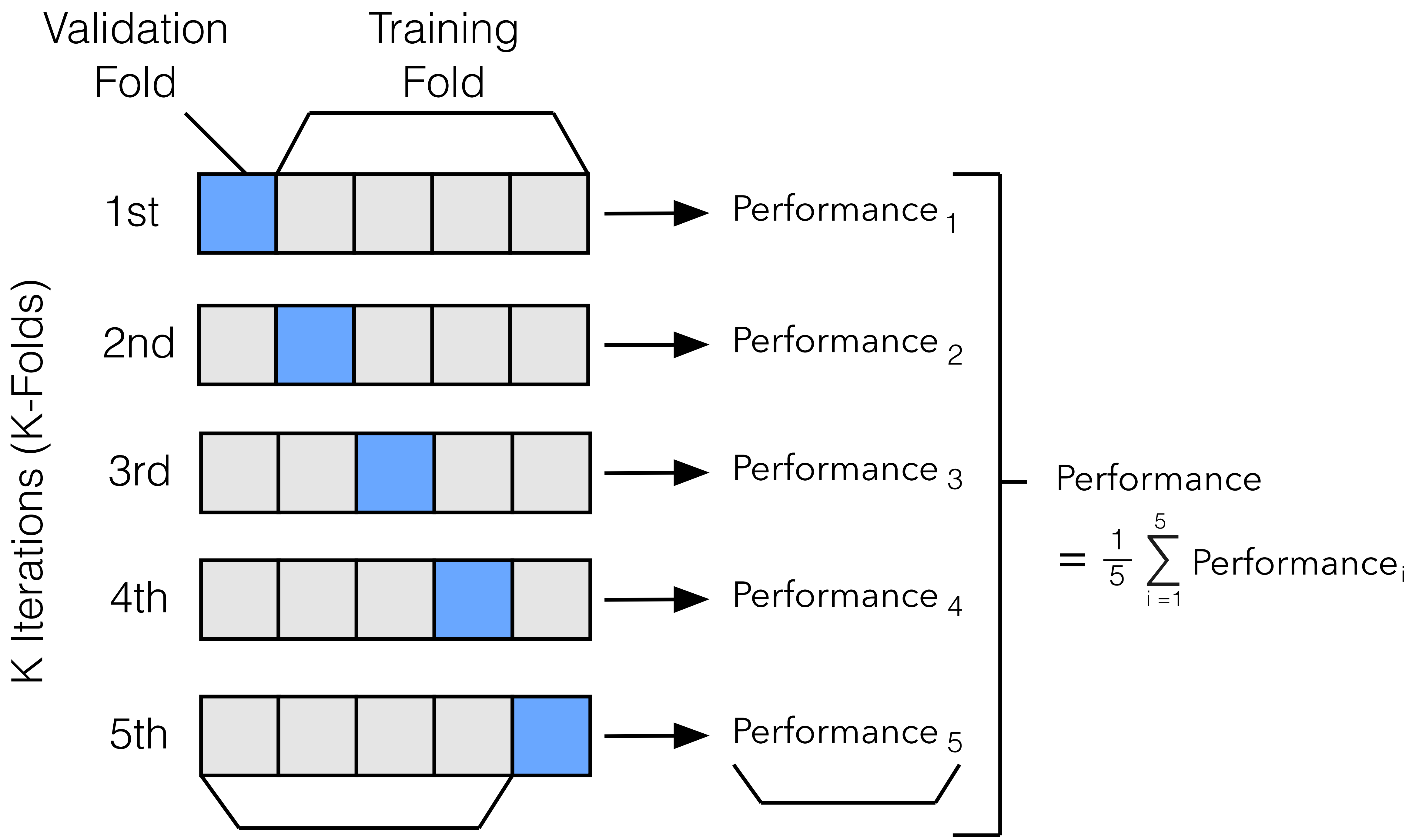
What is the problem with this stacking procedure?

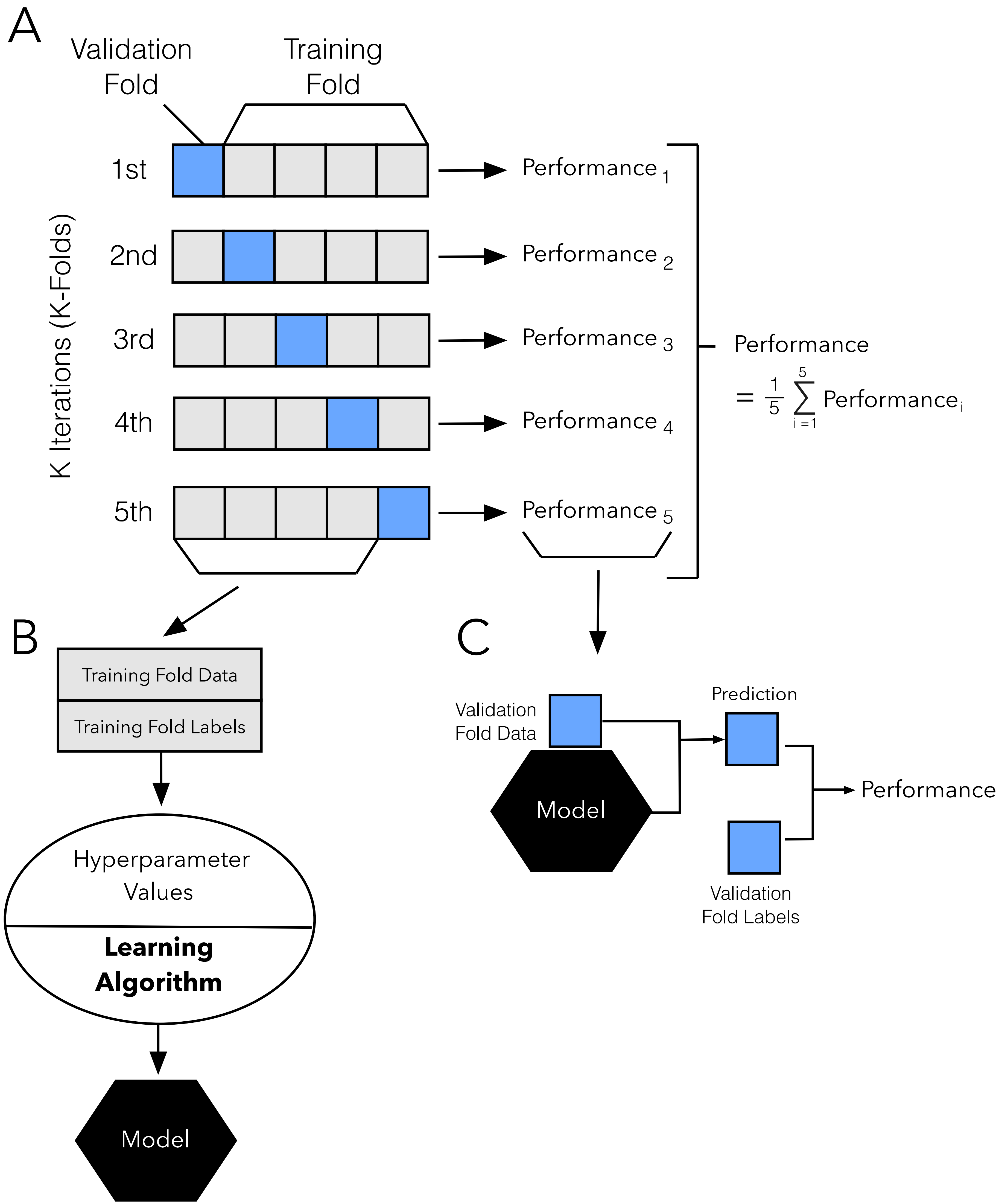


Cross-Validation



k-fold Cross-Validation




A
Validation Fold
Training Fold
K Iterations (K-Folds)
1st
2nd
3rd
4th
5th
Performance 1
Performance 2
Performance 3
Performance 4
Performance 5
Performance
$= \frac{1}{5} \sum_{i=1}^{5} \text{Performance}_i$
B
Training Fold Data
Training Fold Labels
Hyperparameter Values
Learning Algorithm
Model
C
Validation Fold Data
Model
Prediction
Performance
Validation Fold Labels

Stacking Algorithm with Cross-Validation

Wolpert, David H. "Stacked generalization." Neural networks 5.2 (1992): 241-259.

Algorithm 19.8 Stacking with K-fold Cross Validation

Input: Training data $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^m$ ($\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathcal{Y}$)
Output: An ensemble classifier H

Step 1: Adopt cross validation approach in preparing a training set for second-level classifier
Randomly split $\mathcal{D}$ into K equal-size subsets: $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K\}$
for $k \leftarrow 1$ to K **do**
    Step 1.1: Learn first-level classifiers
    for $t \leftarrow 1$ to T **do**
        Learn a classifier h_{kt} from $\mathcal{D} \setminus \mathcal{D}_k$
    end for
    Step 1.2: Construct a training set for second-level classifier
    for $\mathbf{x}_i \in \mathcal{D}_k$ **do**
        Get a record $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_{k1}(\mathbf{x}_i), h_{k2}(\mathbf{x}_i), \ldots, h_{kT}(\mathbf{x}_i)\}$
    end for
end for
Step 2: Learn a second-level classifier
Learn a new classifier h' from the collection of $\{\mathbf{x}'_i, y_i\}$
Step 3: Re-learn first-level classifiers
for $t \leftarrow 1$ to T **do**
    Learn a classifier h_t based on $\mathcal{D}$
end for
return $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$

Tang, J., S. Alelyani, and H. Liu. "Data Classification: Algorithms and Applications." Data Mining and Knowledge Discovery Series, CRC Press (2015): pp. 498-500.

Stacking Algorithm with Cross-Validation