

## Lecture 07

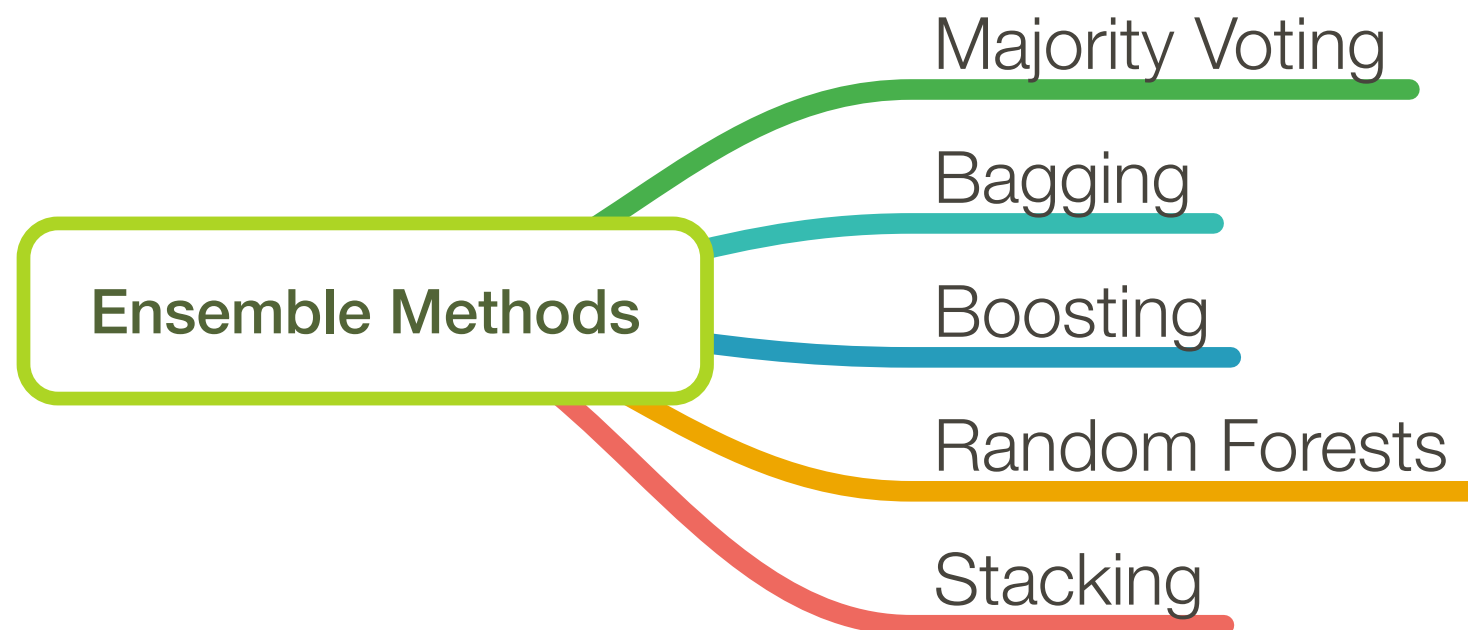
# Ensemble Methods

STAT 479: Machine Learning, Fall 2018

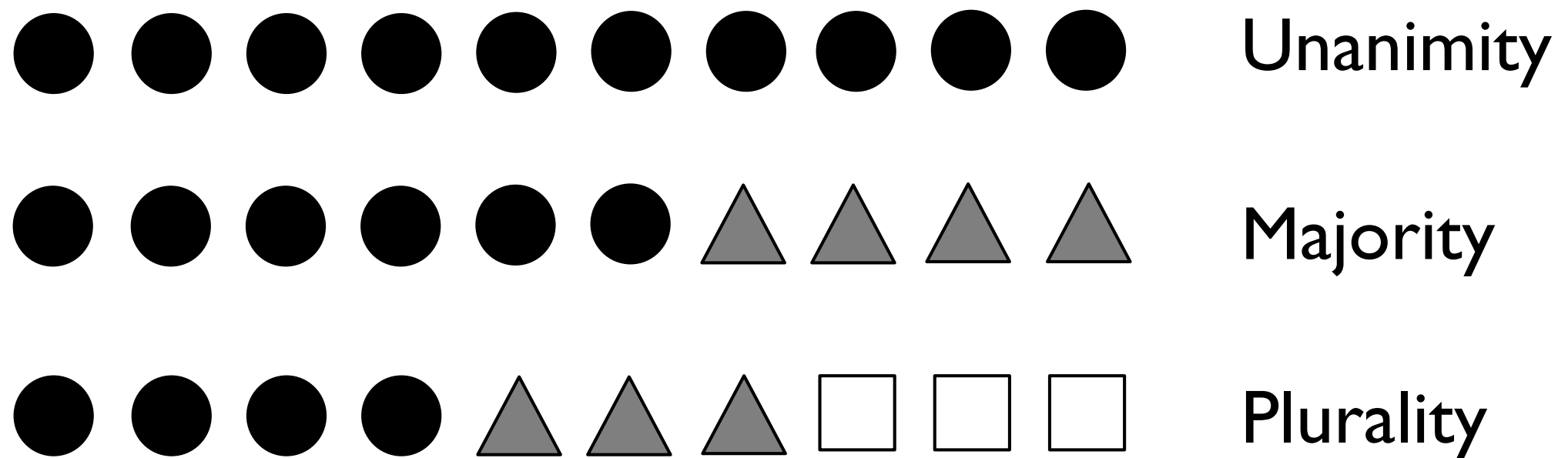
Sebastian Raschka

<http://stat.wisc.edu/~sraschka/teaching/stat479-fs2018/>

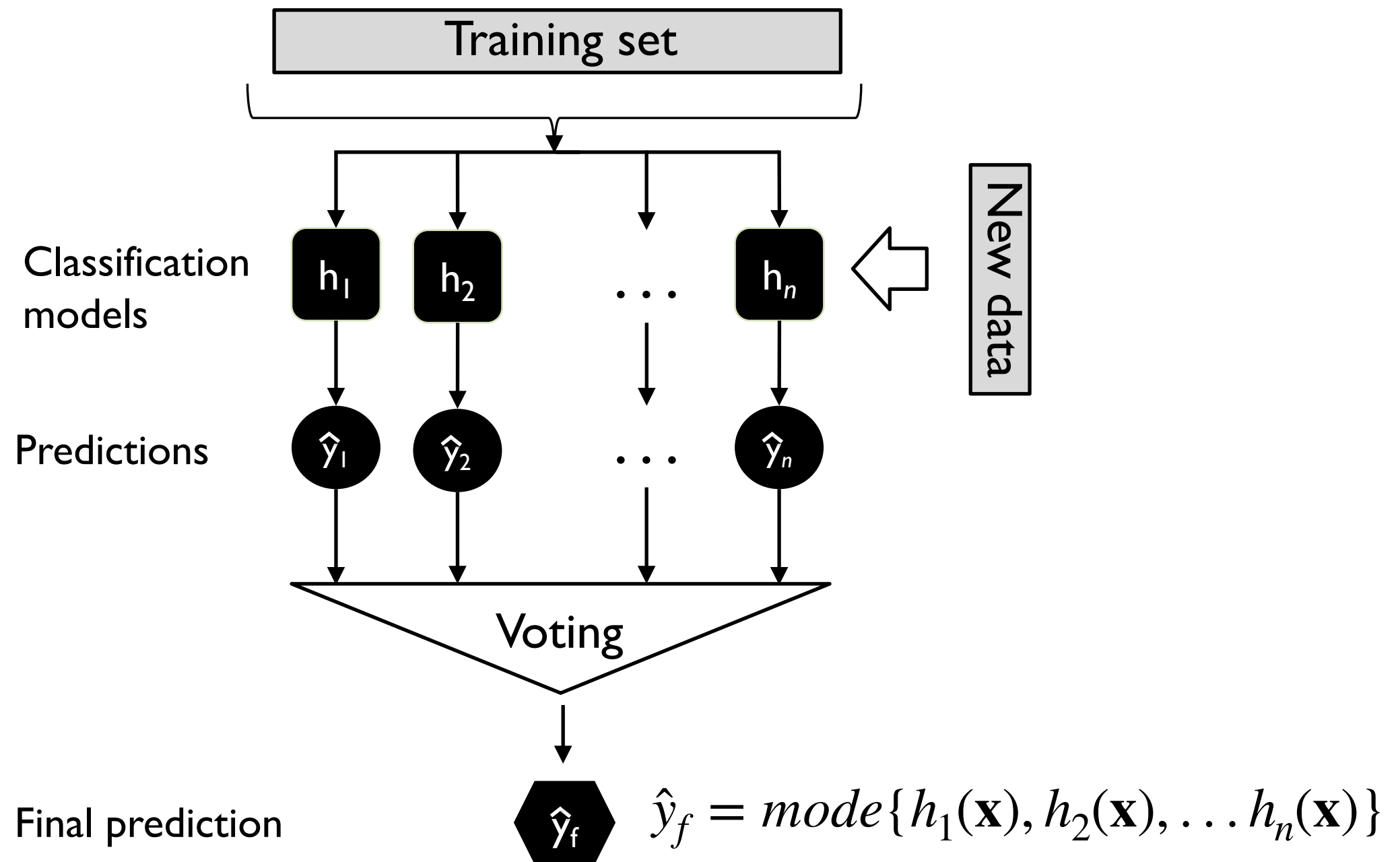
# Overview



# Majority Voting



# Majority Vote Classifier



where  $h_i(\mathbf{x}) = \hat{y}_i$

# Why Majority Vote?

- assume  $n$  independent classifiers with a base error rate  $\epsilon$
- here, independent means that the errors are uncorrelated
- assume a binary classification task
- assume the error rate is better than random guessing (i.e., lower than 0.5 for binary classification)

$$\forall \epsilon_i \in \{\epsilon_1, \epsilon_2, \dots, \epsilon_n\}, \epsilon_i < 0.5$$

# Why Majority Vote?

- assume  $n$  independent classifiers with a base error rate  $\epsilon$
- here, independent means that the errors are uncorrelated
- assume a binary classification task
- assume the error rate is better than random guessing (i.e., lower than 0.5 for binary classification)

$$\forall \epsilon_i \in \{\epsilon_1, \epsilon_2, \dots, \epsilon_n\}, \epsilon_i < 0.5$$

The probability that we make a wrong prediction via the ensemble if  $k$  classifiers predict the same class label

$$P(k) = \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k} \quad k > \lceil n/2 \rceil$$

# Why Majority Vote?

The probability that we make a wrong prediction via the ensemble if  $k$  classifiers predict the same class label

$$P(k) = \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k} \quad k > \lceil n/2 \rceil$$

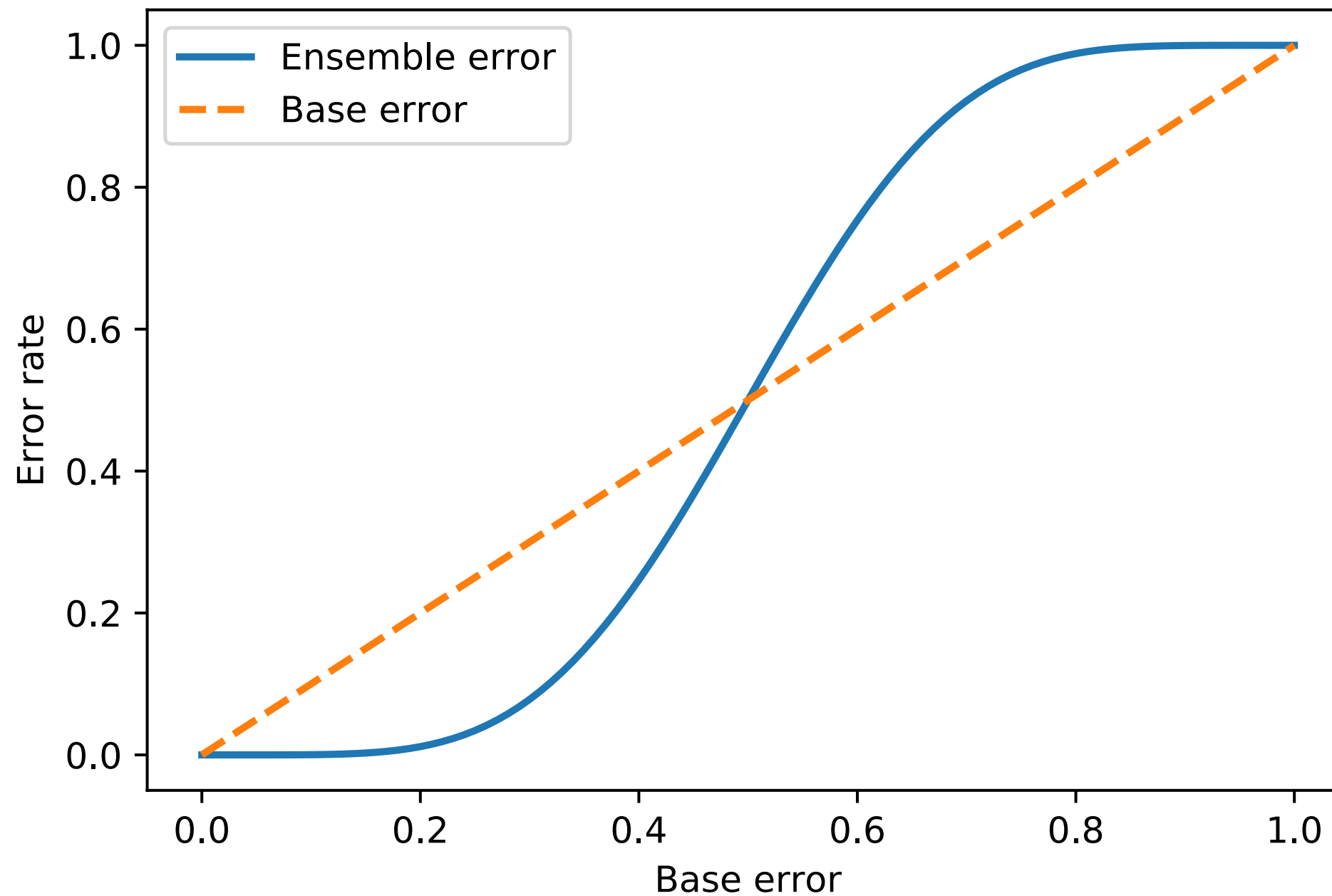
Ensemble error:

$$\epsilon_{ens} = \sum_k^n \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k}$$

$$\epsilon_{ens} = \sum_{k=6}^{11} \binom{11}{k} 0.25^k (1 - 0.25)^{11-k} = 0.034$$



$$\epsilon_{ens} = \sum_k^n \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k}$$



# "Soft" Voting

$$\hat{y} = \arg \max_j \sum_{i=1}^n w_i p_{i,j}$$

$p_{i,j}$ : predicted class membership  
probability of the  $i$ th classifier for  
class label  $j$

$w_j$ : optional weighting parameter, default  
 $w_i = 1/n, \forall w_i \in \{w_1, \dots, w_n\}$

# "Soft" Voting

Use only for well-calibrated  
classifiers!

$$\hat{y} = \arg \max_j \sum_{i=1}^n w_i p_{i,j}$$

$p_{i,j}$ : predicted class membership  
probability of the  $i$ th classifier for  
class label  $j$

$w_j$ : optional weighting parameter, default  
 $w_i = 1/n, \forall w_i \in \{w_1, \dots, w_n\}$

# "Soft" Voting

$$\hat{y} = \arg \max_j \sum_{i=1}^n w_i p_{i,j}$$

Binary classification example

$$j \in \{0,1\} \quad h_i(i \in \{1,2,3\})$$

$$h_1(\mathbf{x}) \rightarrow [0.9, 0.1]$$

$$h_2(\mathbf{x}) \rightarrow [0.8, 0.2]$$

$$h_3(\mathbf{x}) \rightarrow [0.4, 0.6]$$

# "Soft" Voting

$$\hat{y} = \arg \max_j \sum_{i=1}^n w_i p_{i,j}$$

Binary classification example

$$j \in \{0,1\} \quad h_i(i \in \{1,2,3\})$$

$$h_1(\mathbf{x}) \rightarrow [0.9, 0.1]$$

$$h_2(\mathbf{x}) \rightarrow [0.8, 0.2]$$

$$h_3(\mathbf{x}) \rightarrow [0.4, 0.6]$$

$$p(j = 0 \mid \mathbf{x}) = 0.2 \cdot 0.9 + 0.2 \cdot 0.8 + 0.6 \cdot 0.4 = 0.58$$

$$p(j = 1 \mid \mathbf{x}) = 0.2 \cdot 0.1 + 0.2 \cdot 0.2 + 0.6 \cdot 0.6 = 0.42$$

$$\hat{y} = \arg \max_j \left\{ p(j = 0 \mid \mathbf{x}), p(j = 1 \mid \mathbf{x}) \right\}$$

# Bagging

## (Bootstrap Aggregating)

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2), 123-140.

---

**Algorithm 1** Bagging

---

- 1: Let  $n$  be the number of bootstrap samples
  - 2:
  - 3: **for**  $i=1$  to  $n$  **do**
  - 4:     Draw bootstrap sample of size  $m$ ,  $\mathcal{D}_i$
  - 5:     Train base classifier  $h_i$  on  $\mathcal{D}_i$
  - 6:  $\hat{y} = \text{mode}\{h_1(\mathbf{x}), \dots, h_n(\mathbf{x})\}$
-

# Bootstrap Sampling





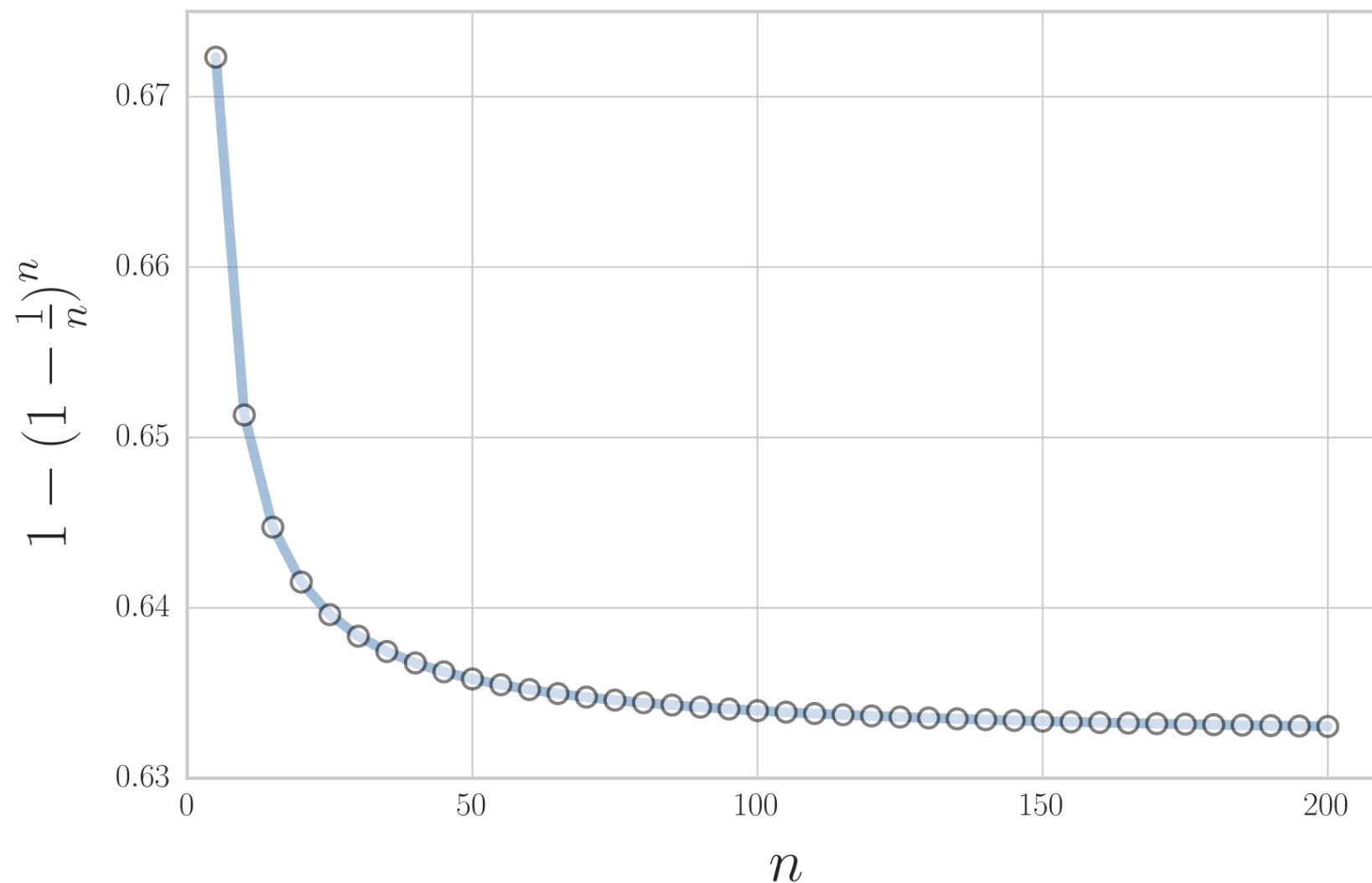
# Bootstrap Sampling

$$P(\text{not chosen}) = \left(1 - \frac{1}{n}\right)^n,$$
$$\frac{1}{e} \approx 0.368, \quad n \rightarrow \infty.$$

$$P(\text{not chosen}) = \left(1 - \frac{1}{n}\right)^n,$$

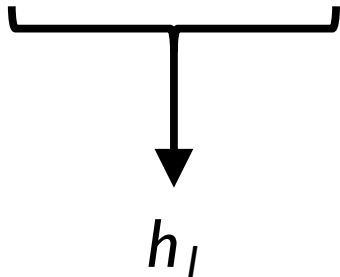
$$\frac{1}{e} \approx 0.368, \quad n \rightarrow \infty.$$

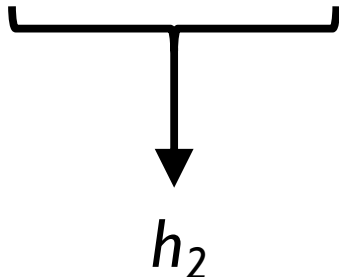
$$P(\text{chosen}) = 1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632$$

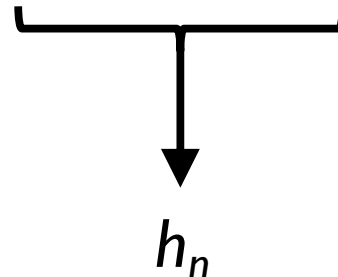


# Bootstrap Sampling

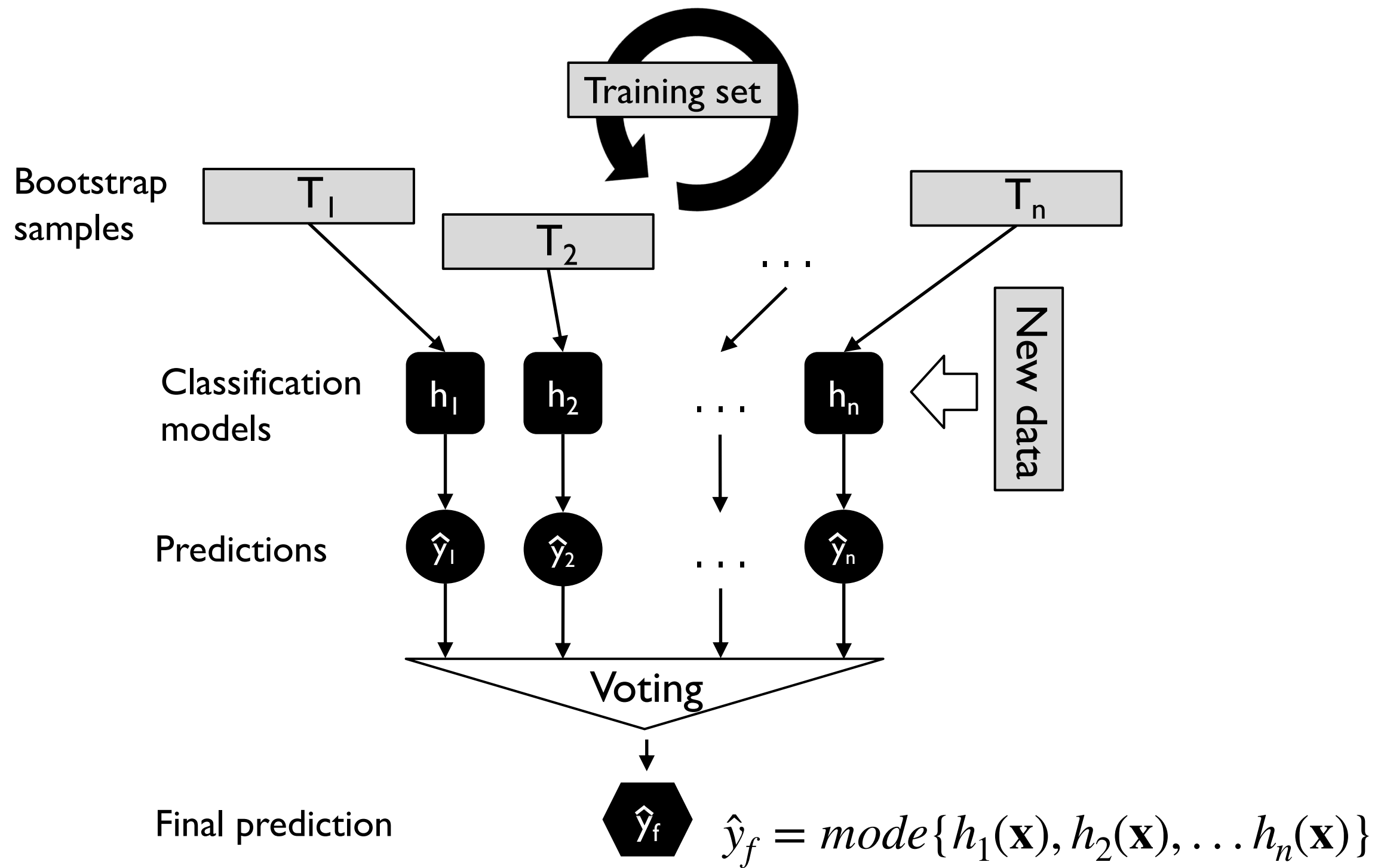
Training example indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...

 $h_1$

 $h_2$

 $h_n$

# Bagging Classifier



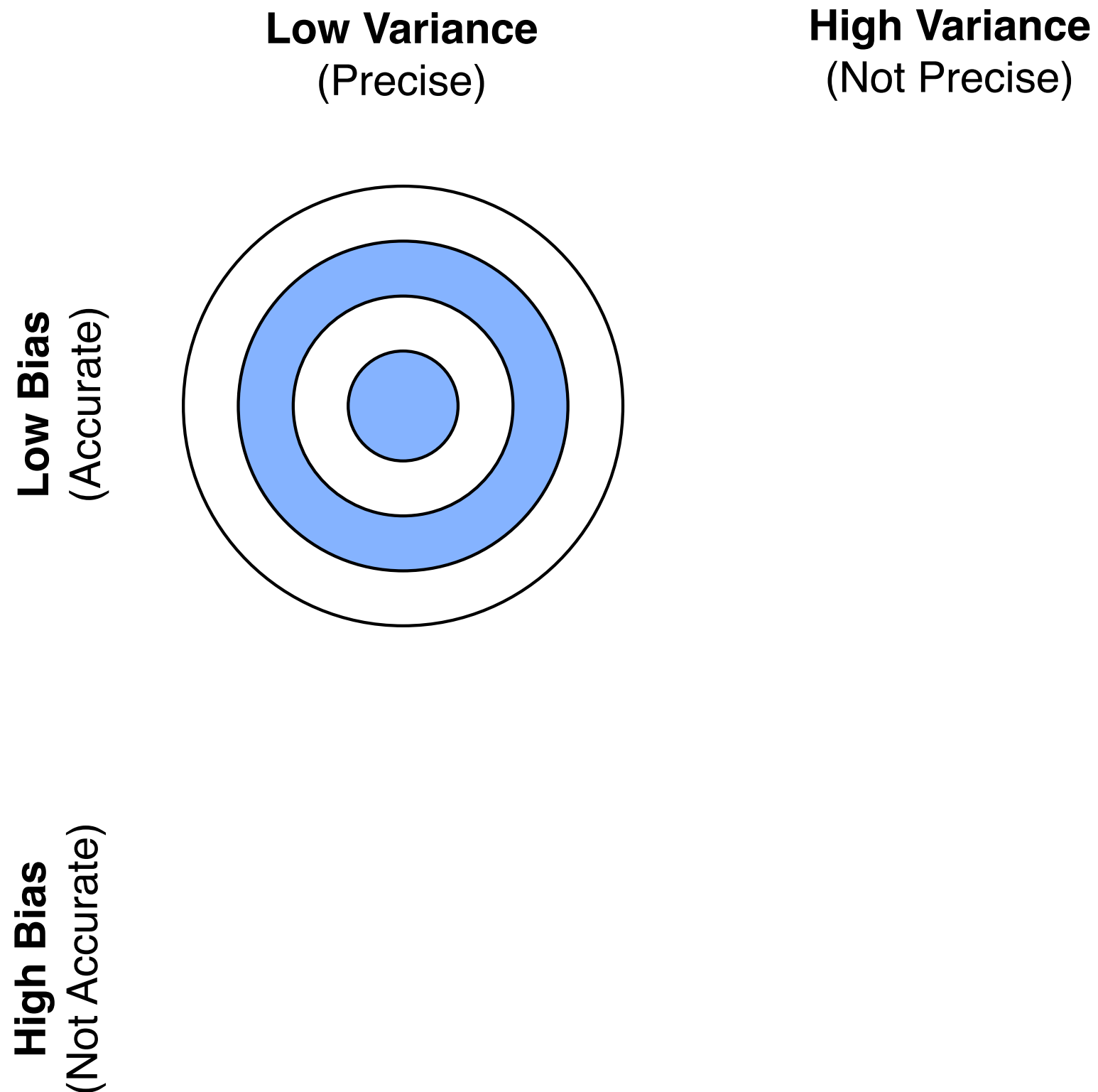
where  $h_i(\mathbf{x}) = \hat{y}_i$

# Bias-Variance Decomposition

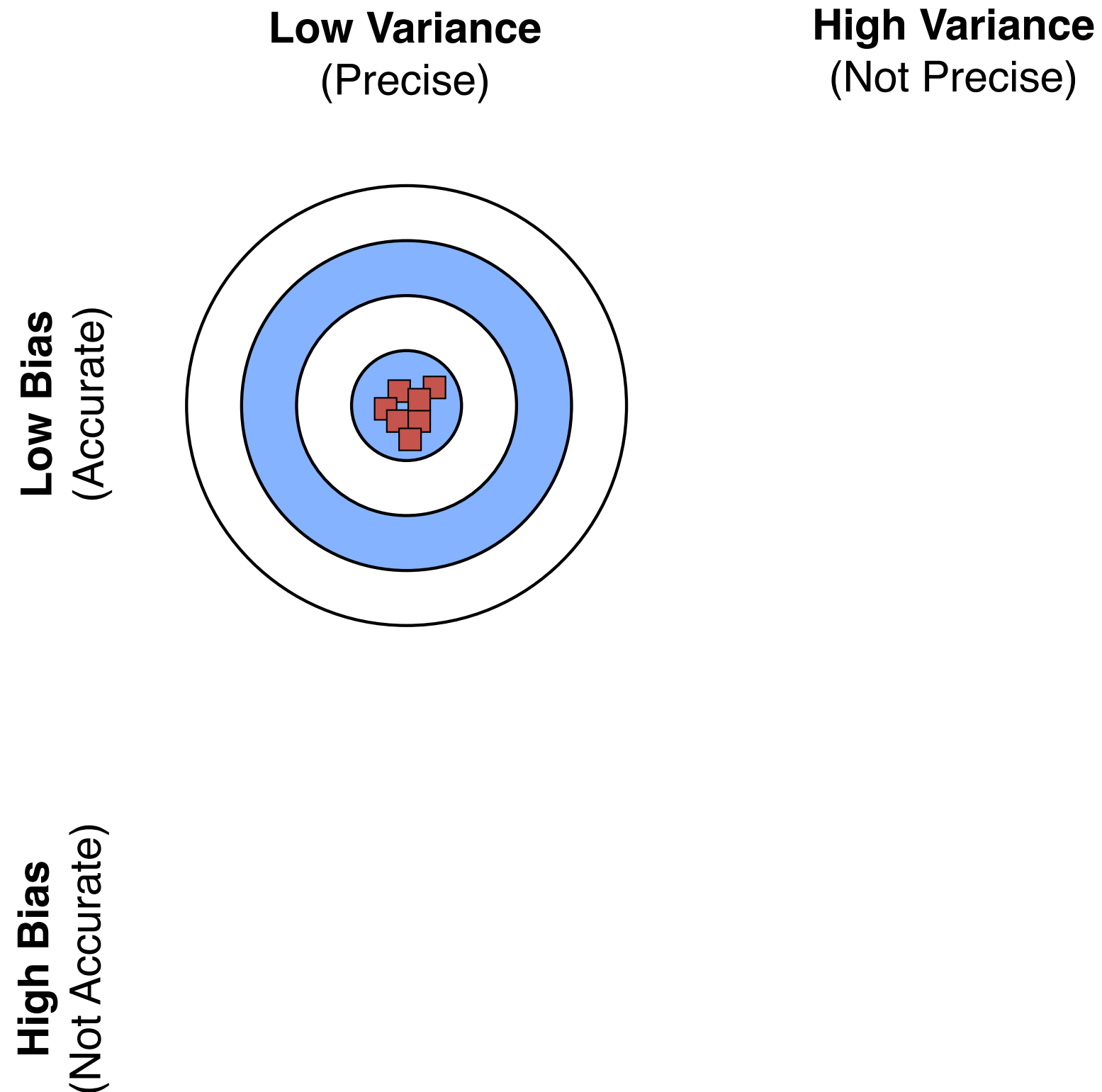
$$\text{Loss} = \text{Bias} + \text{Variance} + \text{Noise}$$

(more technical details in next lecture on model evaluation)

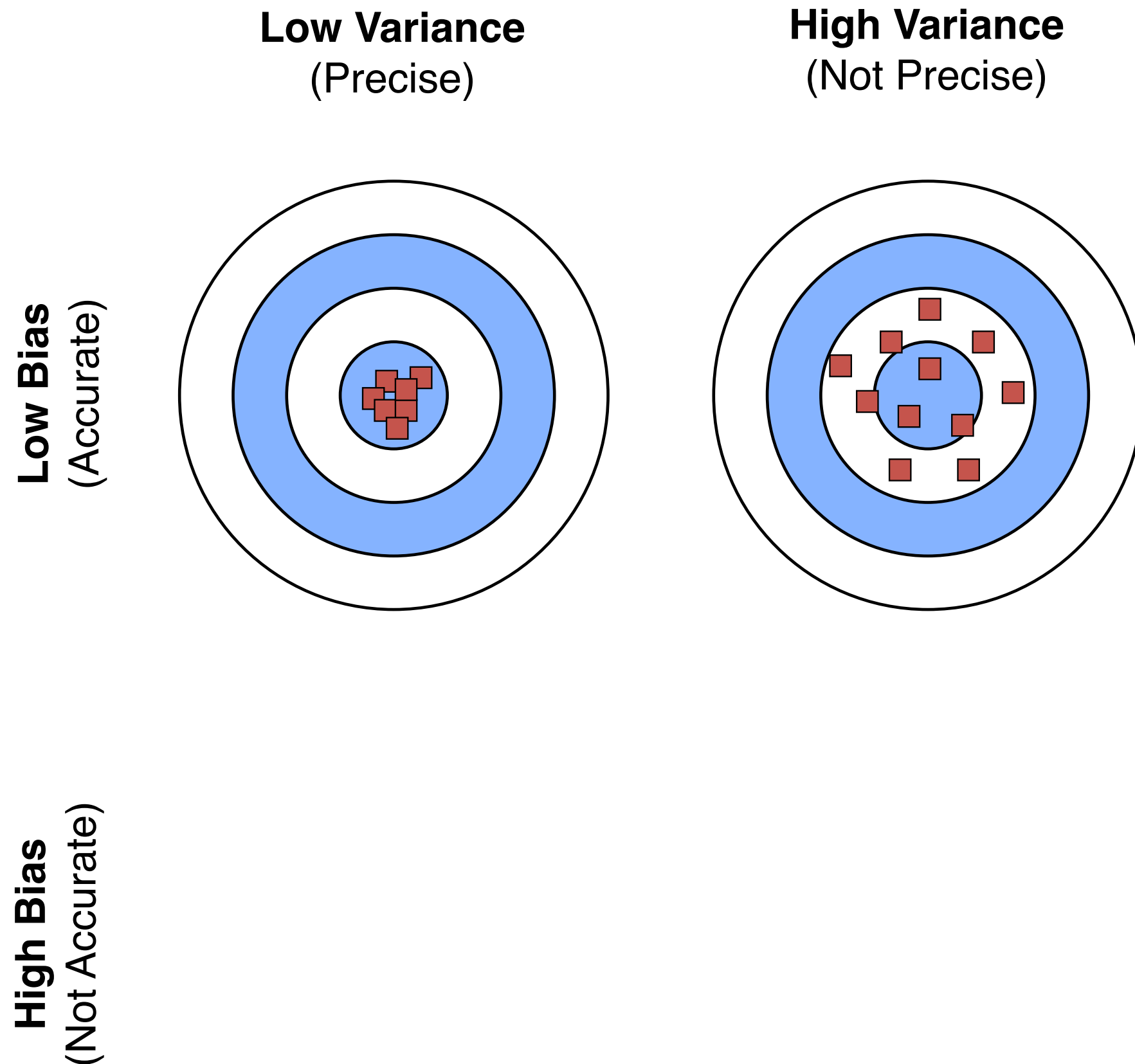
# Bias-Variance Intuition



# Bias-Variance Intuition

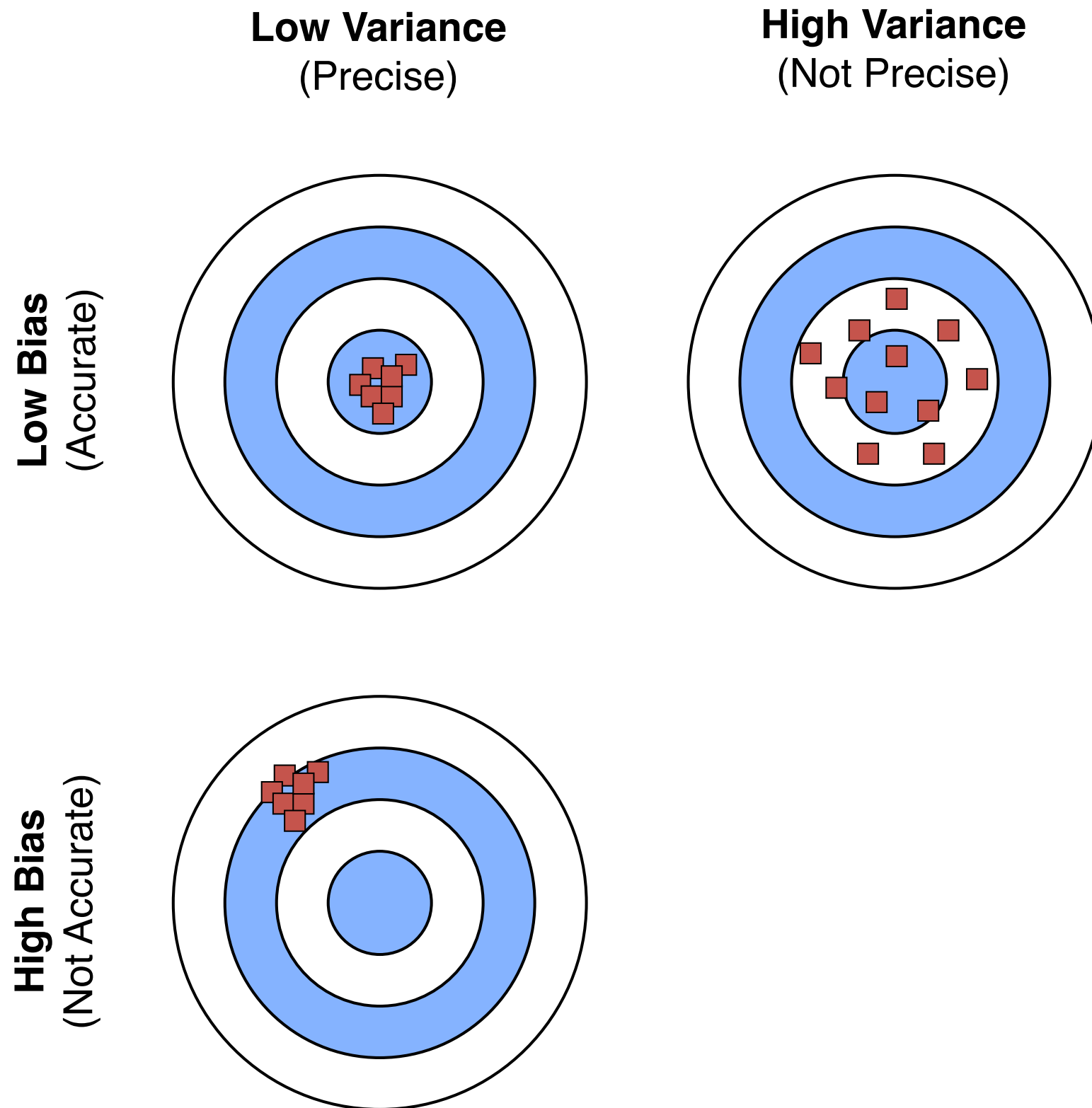


# Bias-Variance Intuition

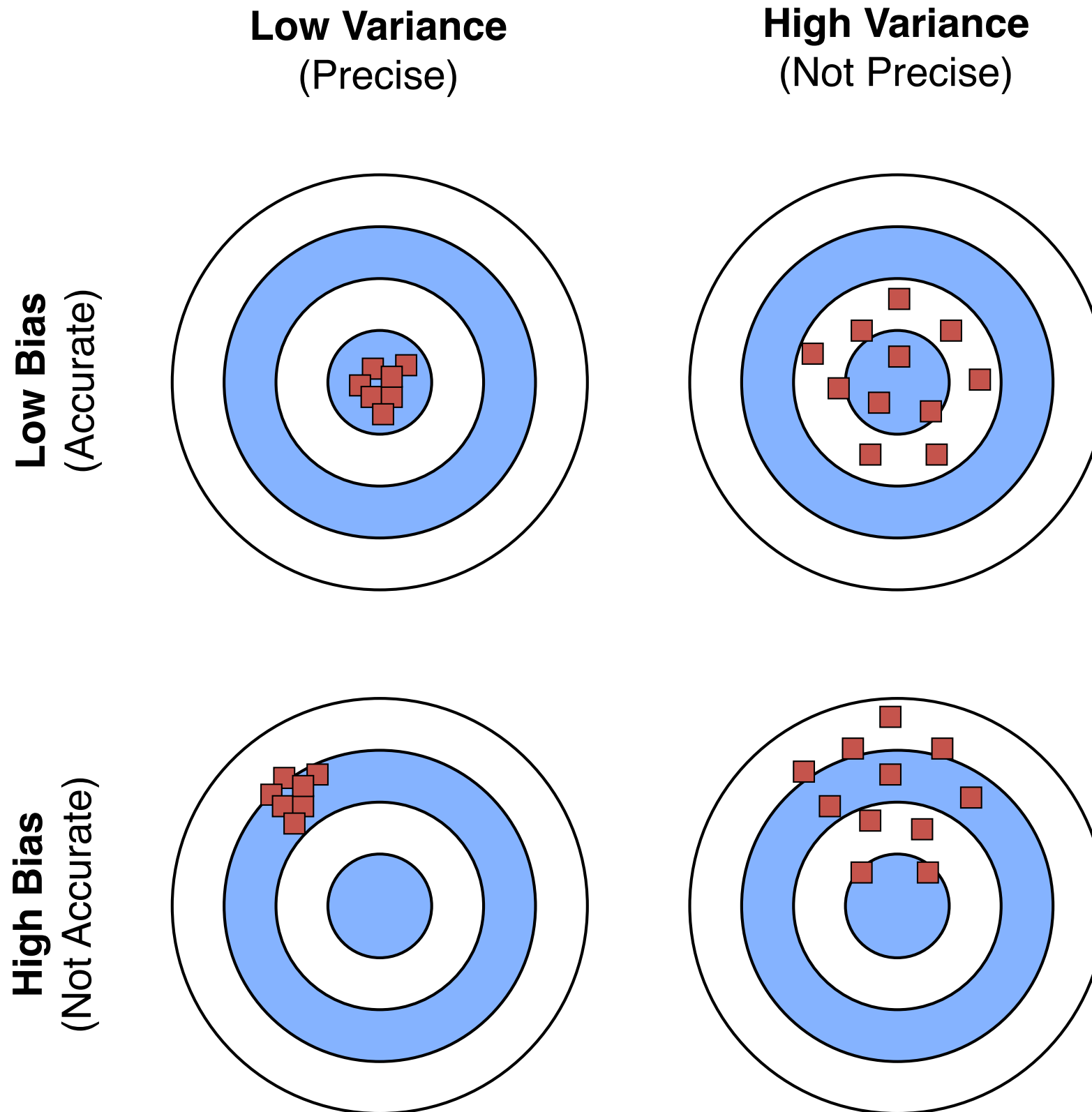




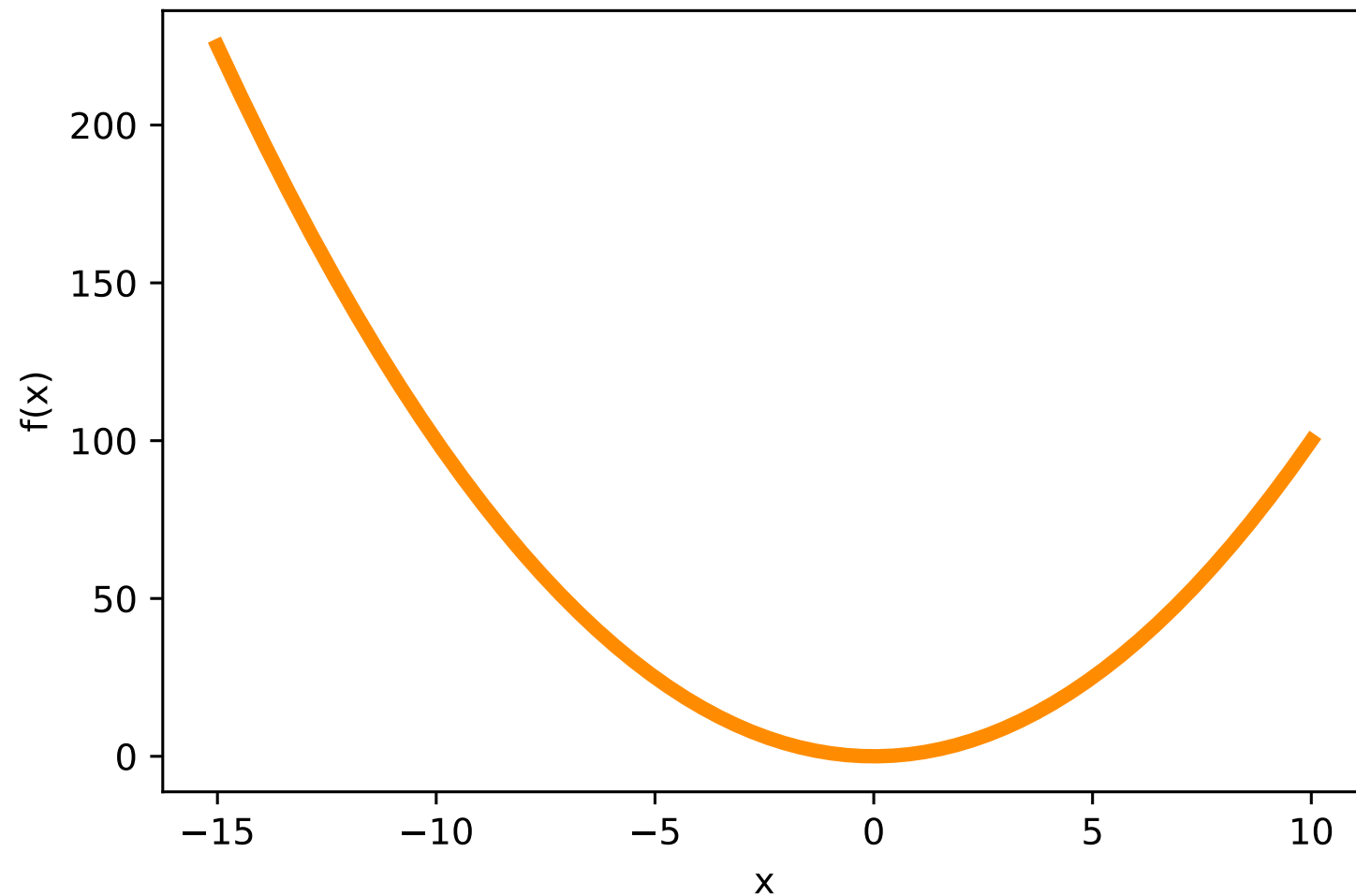
# Bias-Variance Intuition



# Bias-Variance Intuition

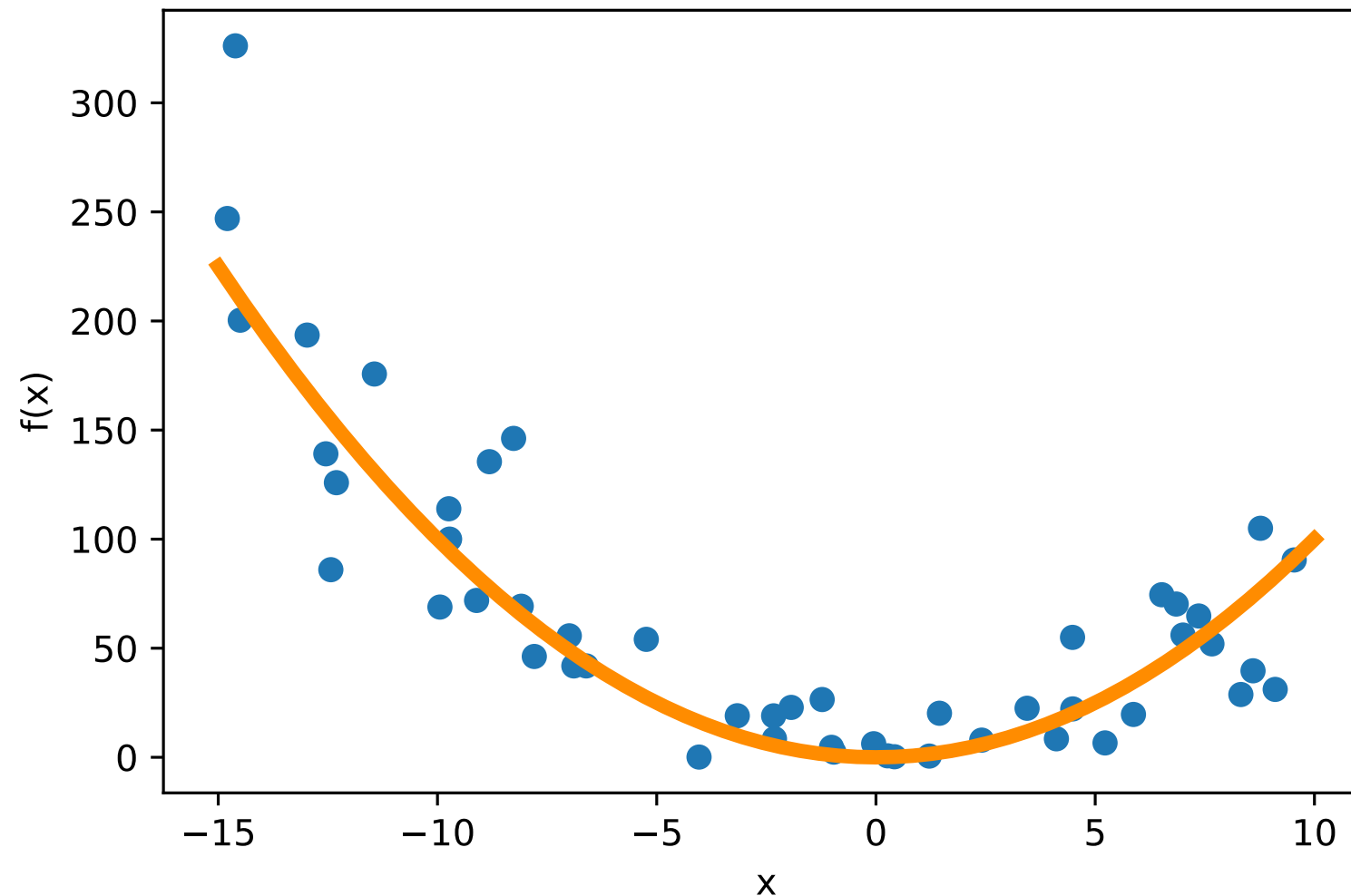


# Bias and Variance Example



where  $f(x)$  is some true (target) function

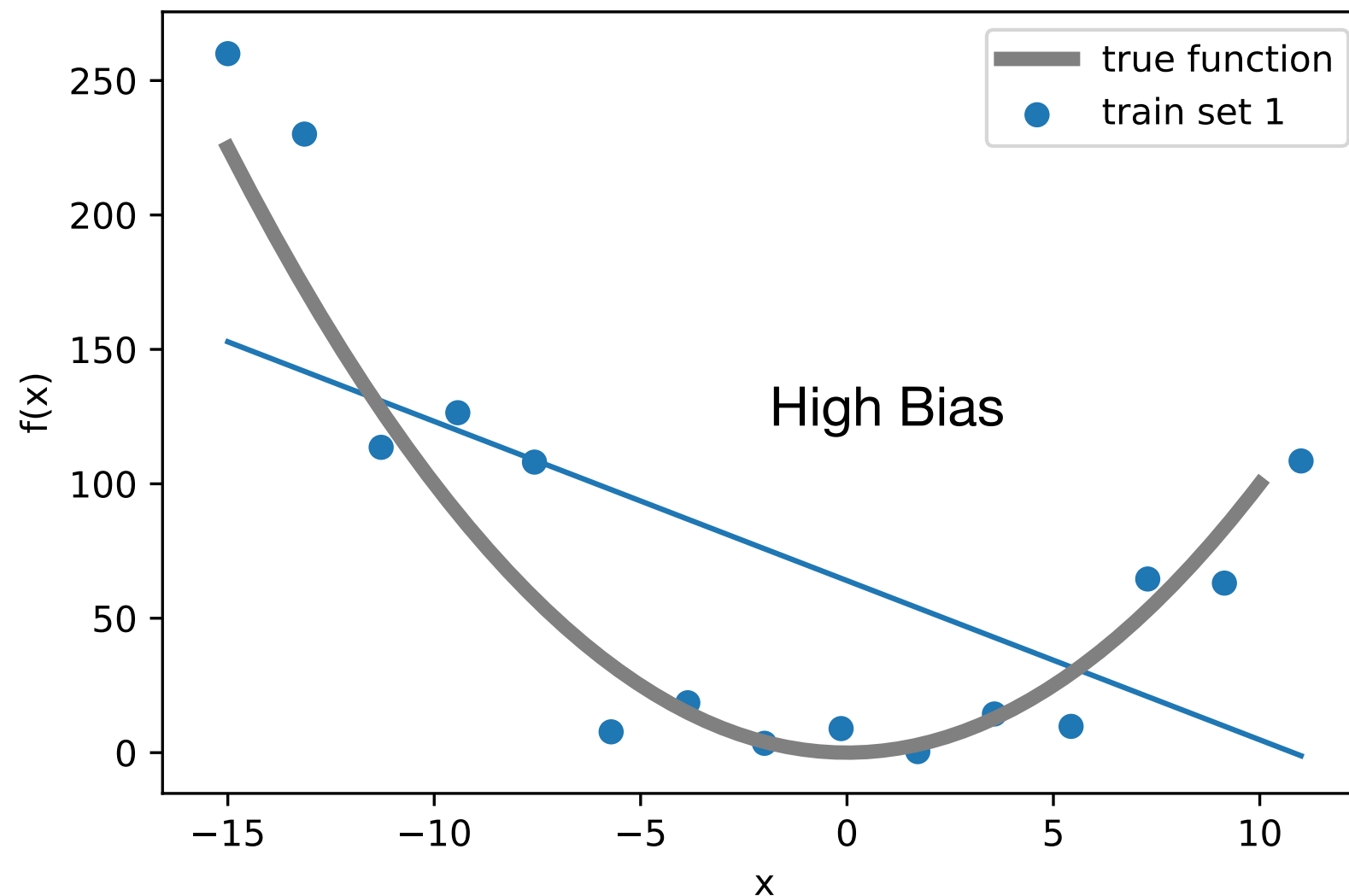
# Bias and Variance Example



where  $f(x)$  is some true (target) function

the blue dots are a training dataset;  
here, I added some random Gaussian noise

# Bias and Variance Example

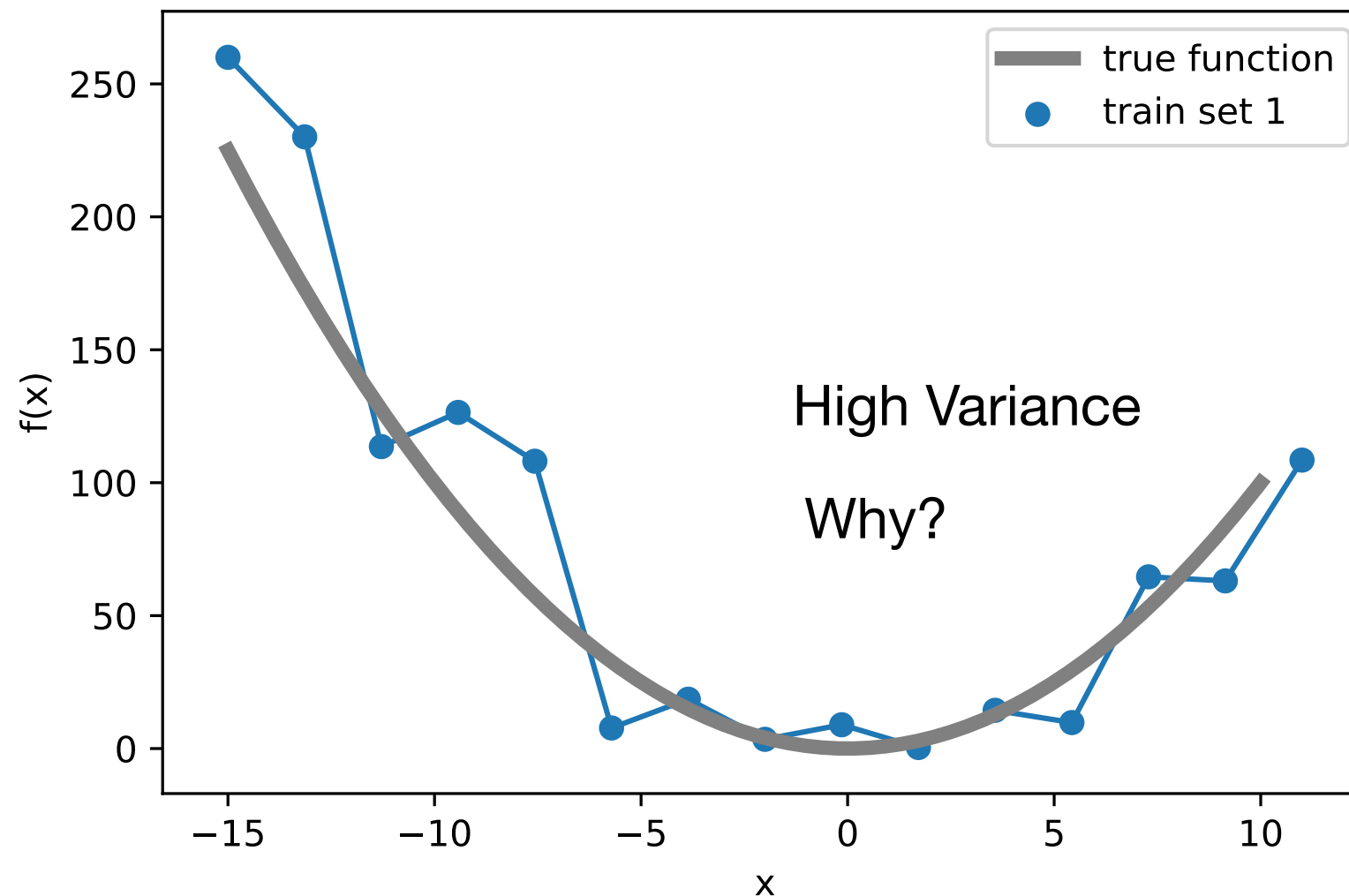


where  $f(x)$  is some true (target) function

the blue dots are a training dataset;  
here, I added some random Gaussian noise

here, suppose I fit a simple linear model (linear regression)  
or a decision tree stump

# Bias and Variance Example

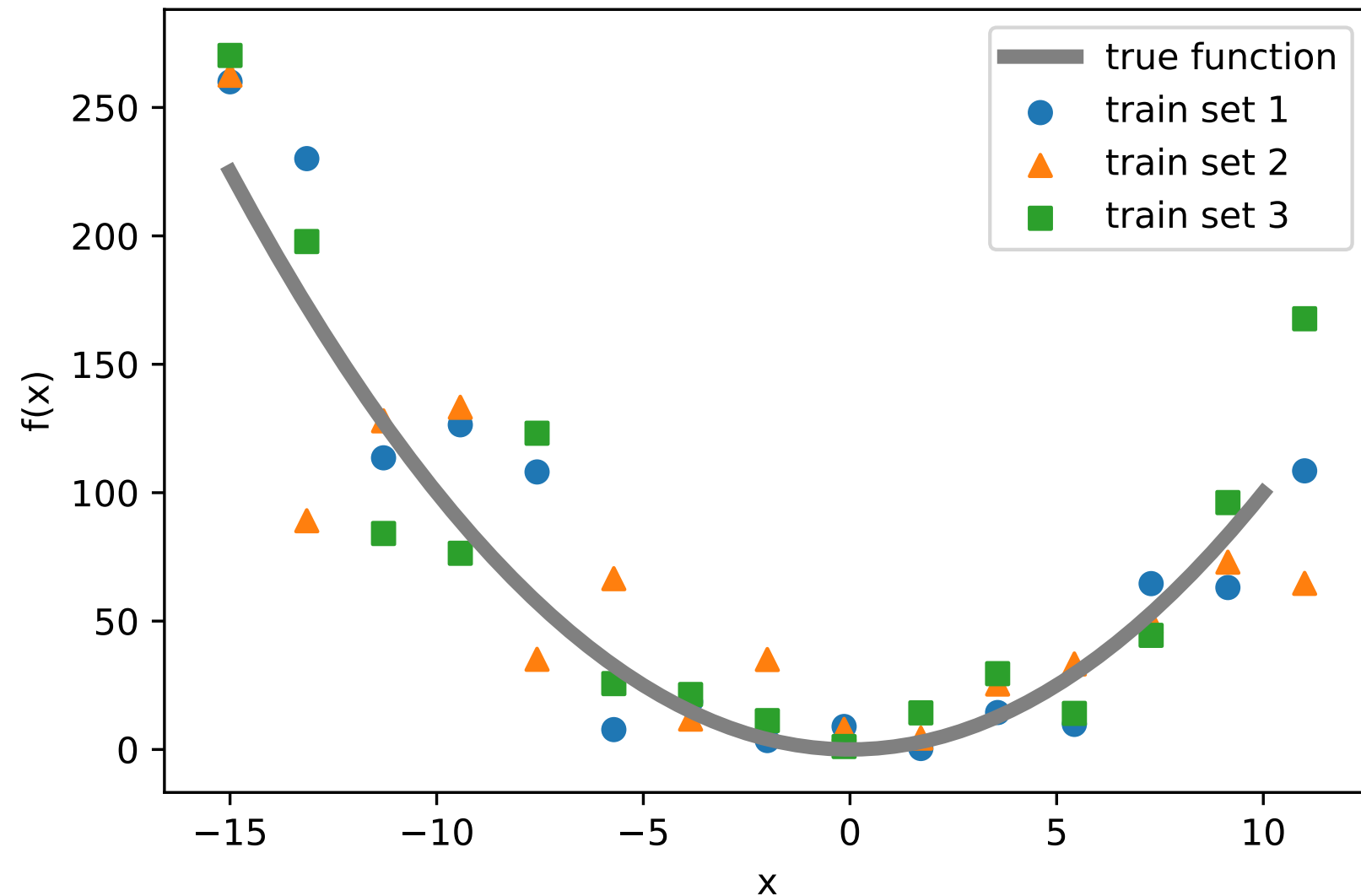


where  $f(x)$  is some true (target) function

the blue dots are a training dataset;  
here, I added some random Gaussian noise

here, suppose I fit an unpruned decision tree

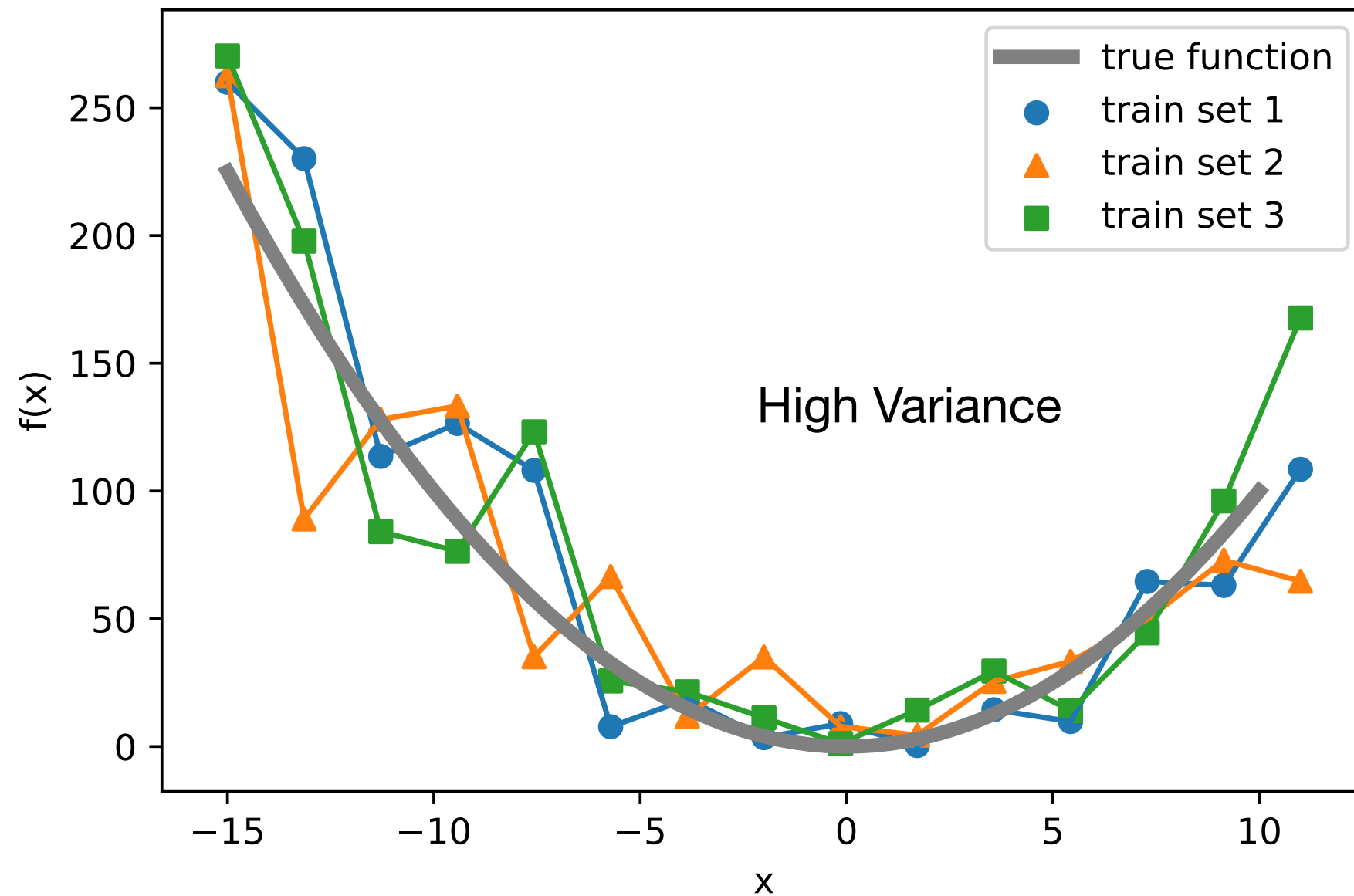
# Bias and Variance Example



where  $f(x)$  is some true (target) function

suppose we have multiple training sets

# Bias and Variance Example





So, why does bagging work/what does it do?

# Boosting

# Adaptive Boosting

e.g., AdaBoost (here!)

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), 119-139.

# Gradient Boosting

e.g., XGBoost

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.

Chen, T., & Guestrin, C. (2016). 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.

# Adaptive Boosting

e.g., AdaBoost (here!)

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), 119-139.

Differ mainly in terms of how

- weights are updated
- classifiers are combined

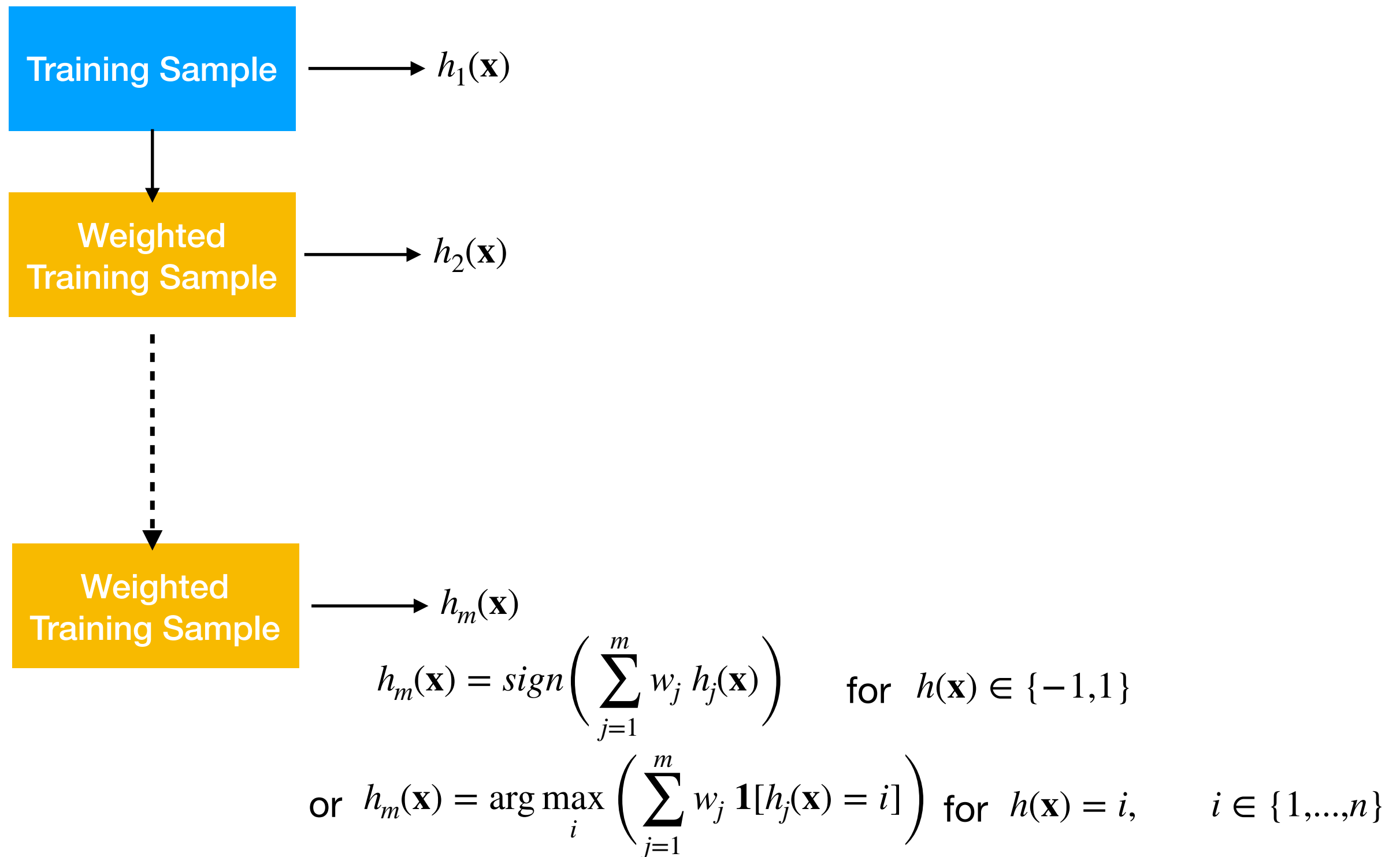
# Gradient Boosting

e.g., XGBoost

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.

Chen, T., & Guestrin, C. (2016). 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.

# General Boosting



# 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

---

- 1: Initialize  $k$ : the number of AdaBoost rounds
  - 2: Initialize  $\mathcal{D}$ : the training dataset,  $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \dots, \langle \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
  - 3: Initialize  $w_1(i) = 1/n$ ,  $i = 1, \dots, n$ ,  $\mathbf{w}_1 \in \mathbb{R}^n$
  - 4:
  - 5: **for**  $r=1$  to  $k$  **do**
  - 6:     For all  $i$  :  $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$      [normalize weights]
  - 7:      $h_r := \text{FitWeakLearner}(\mathcal{D}, \mathbf{w}_r)$
  - 8:      $\epsilon_r := \sum_i w_r(i) \mathbf{1}(h_r(i) \neq y_i)$      [compute error]
  - 9:     if  $\epsilon_r > 1/2$  then stop
  - 10:      $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$      [small if error is large and vice versa]
  - 11:      $w_{r+1}(i) := w_r(i) / z_r \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$
  - 12: Predict:  $h_k(\mathbf{x}) = \arg \max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$
  - 13:
-

# Modify AdaBoost w. Bootstrap

---

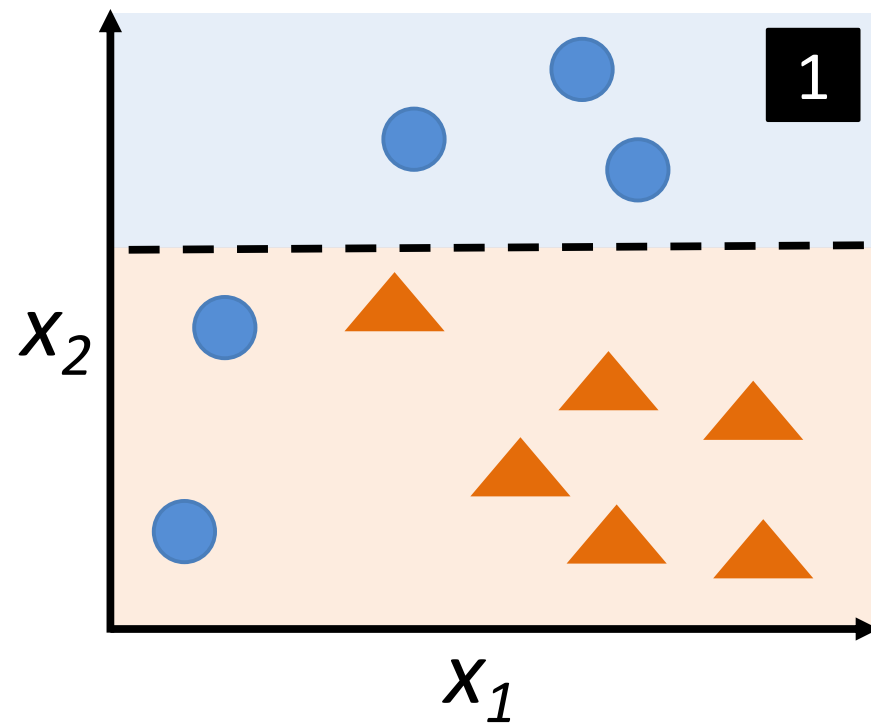
**Algorithm 1** AdaBoost

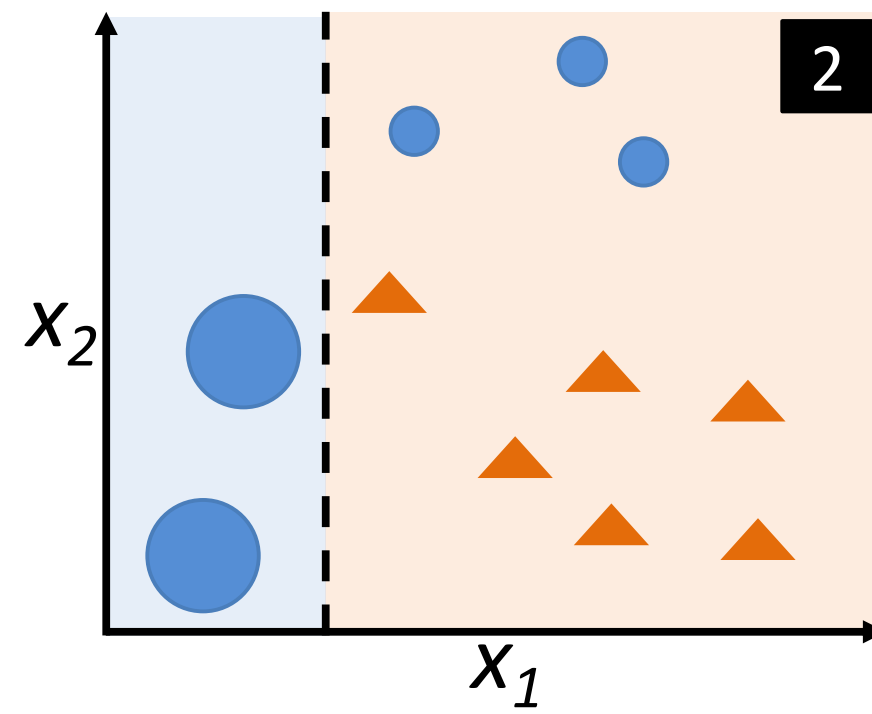
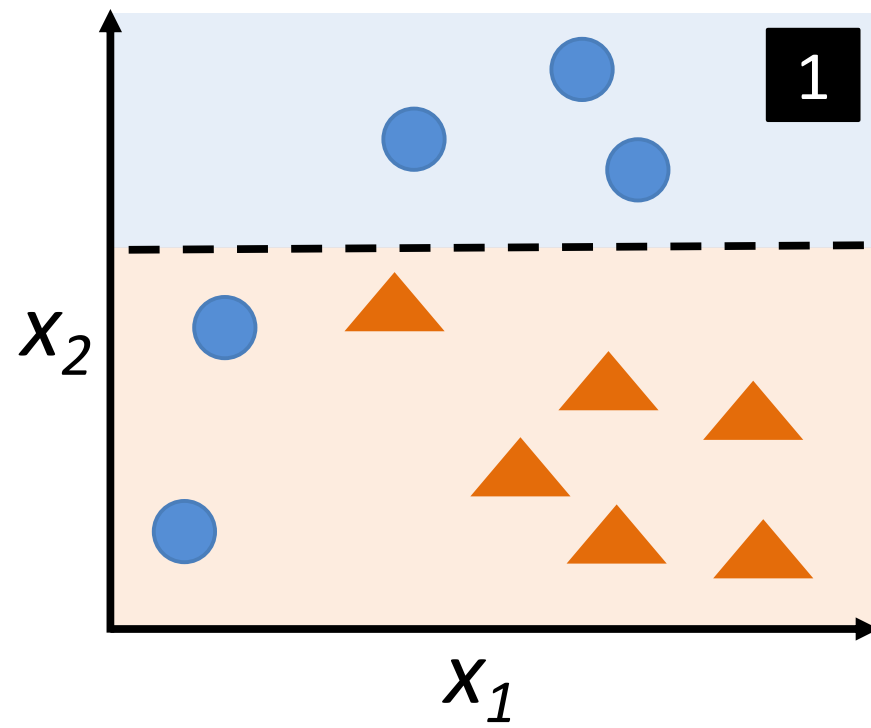
---

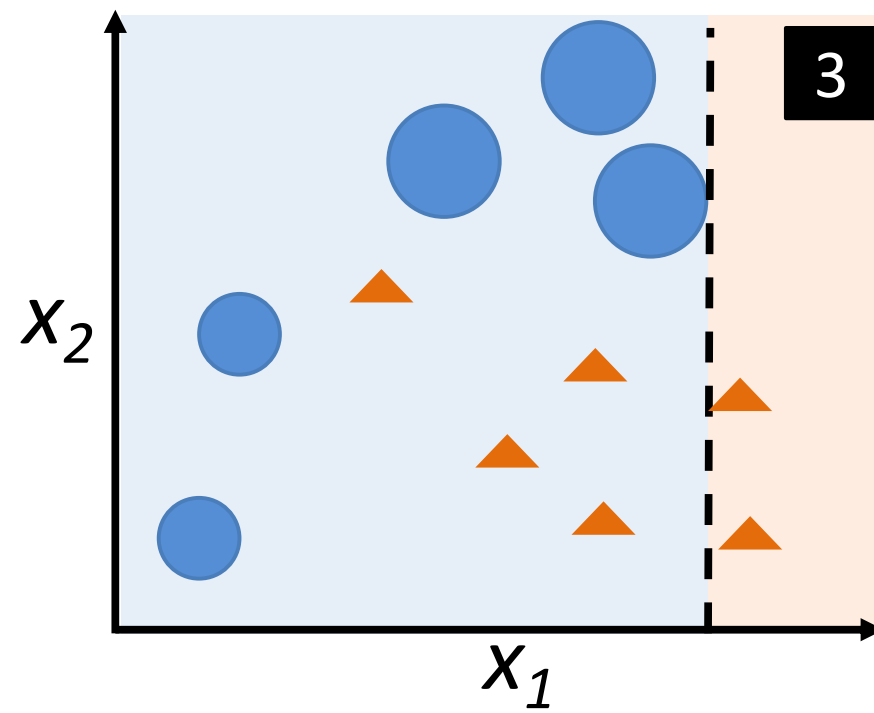
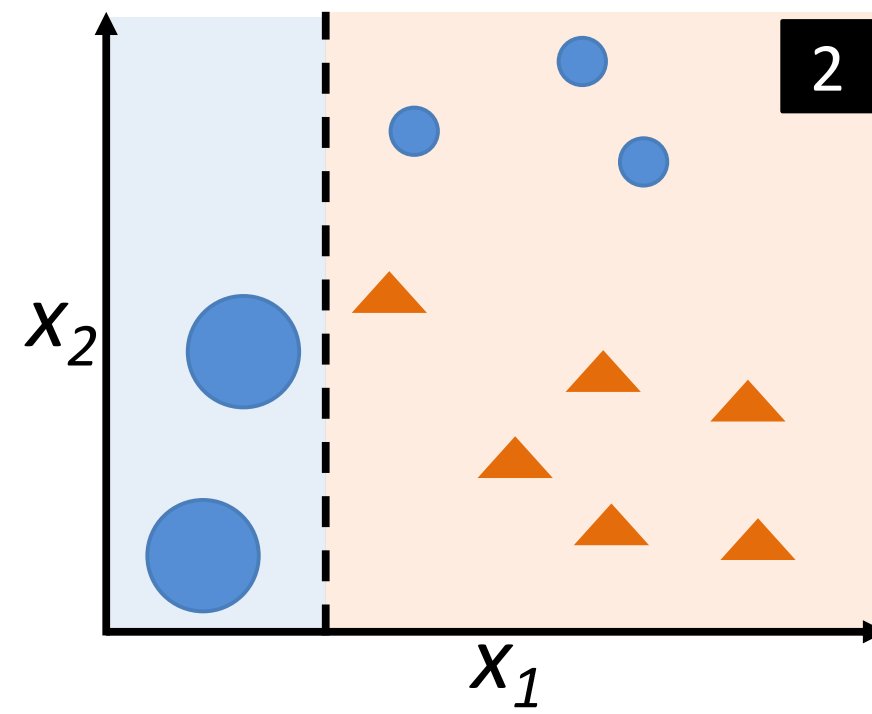
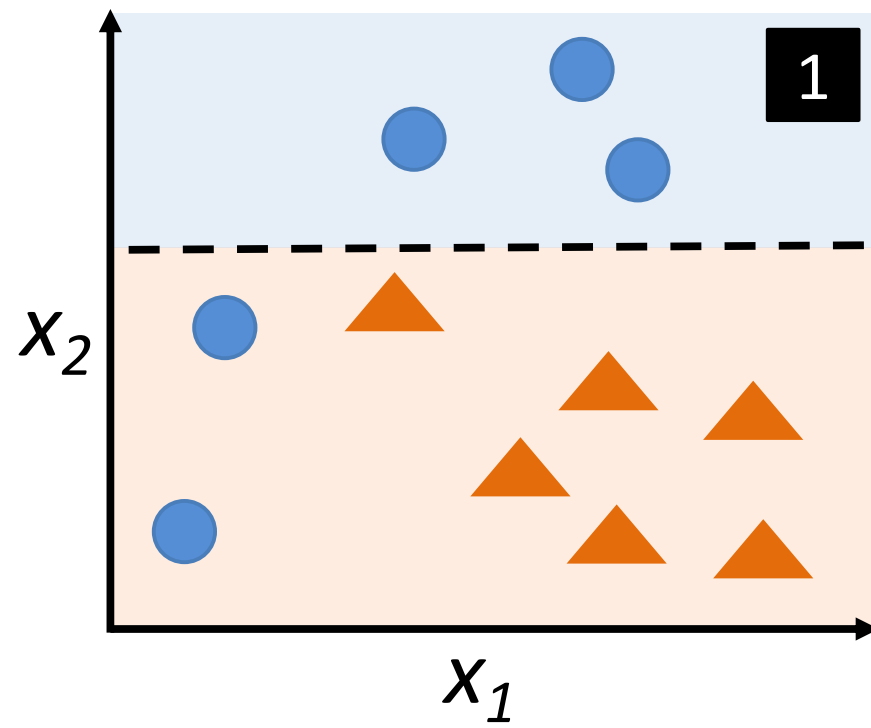
- 1: Initialize  $k$ : the number of AdaBoost rounds
  - 2: Initialize  $\mathcal{D}$ : the training dataset,  $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \dots, \langle \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
  - 3: Initialize  $w_1(i) = 1/n$ ,  $i = 1, \dots, n$ ,  $\mathbf{w}_1 \in \mathbb{R}^n$
  - 4:
  - 5: **for**  $r=1$  to  $k$  **do**
  - 6:   For all  $i$  :  $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$    [normalize weights]
  - 7:    $h_r := \text{FitWeakLearner}(\mathcal{D}, \mathbf{w}_r)$
  - 8:    $\epsilon_r := \sum_i w_r(i) \mathbf{1}(h_r(i) \neq y_i)$    [compute error]
  - 9:   if  $\epsilon_r > 1/2$  then stop
  - 10:    $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r) / \epsilon_r]$    [small if error is large and vice versa]
  - 11:    $w_{r+1}(i) := w_r(i) / z_r \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$
  - 12: Predict:  $h_k(\mathbf{x}) = \arg \max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$
  - 13:
-

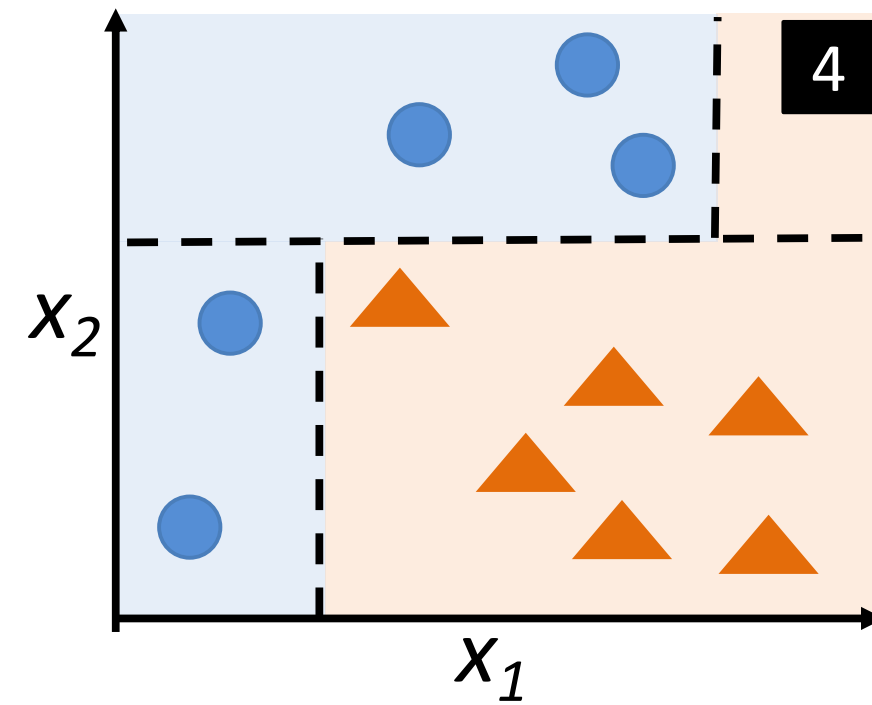
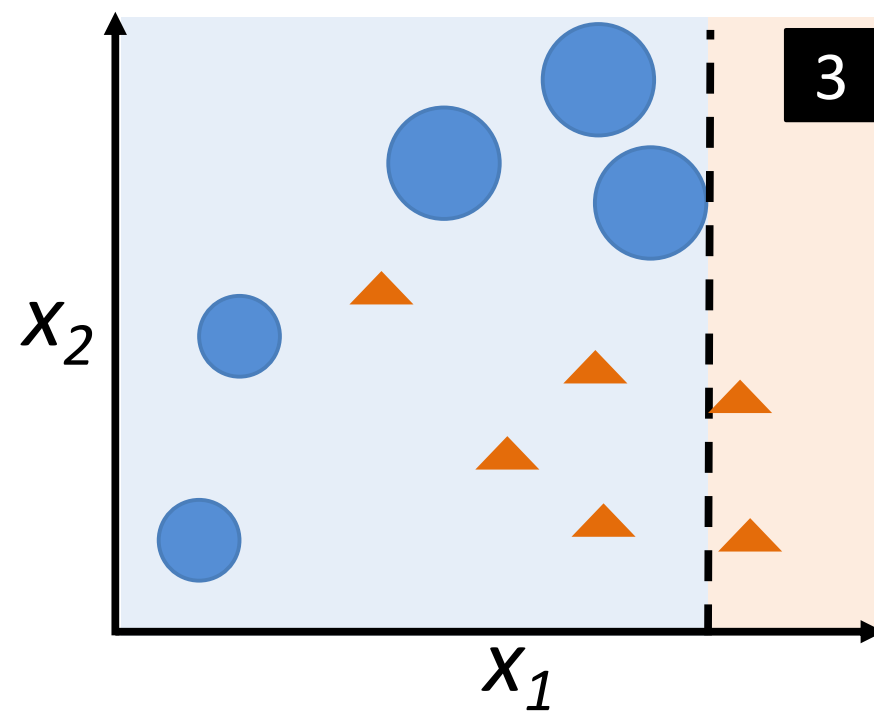
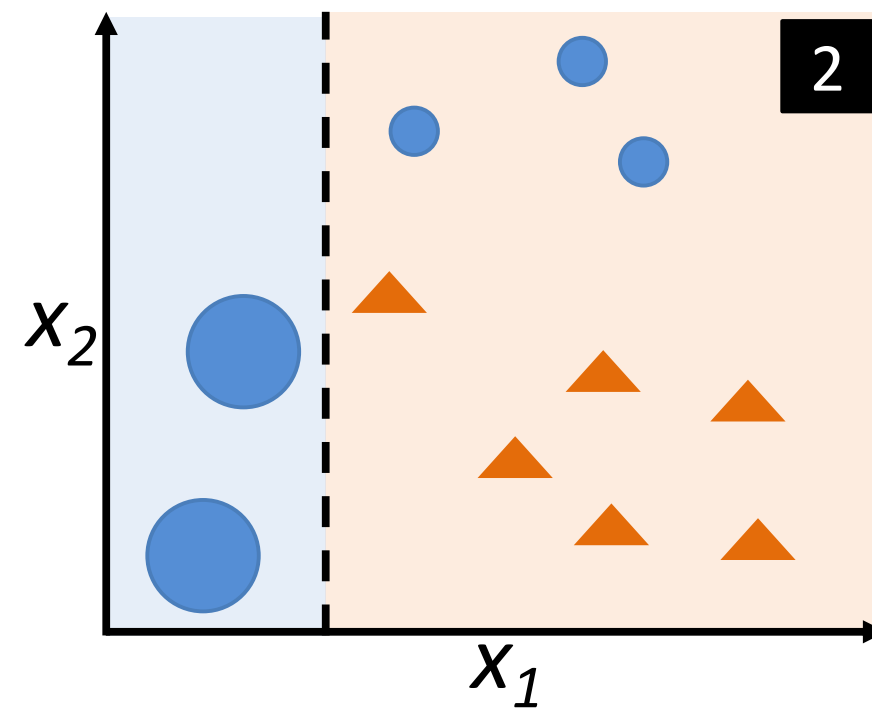
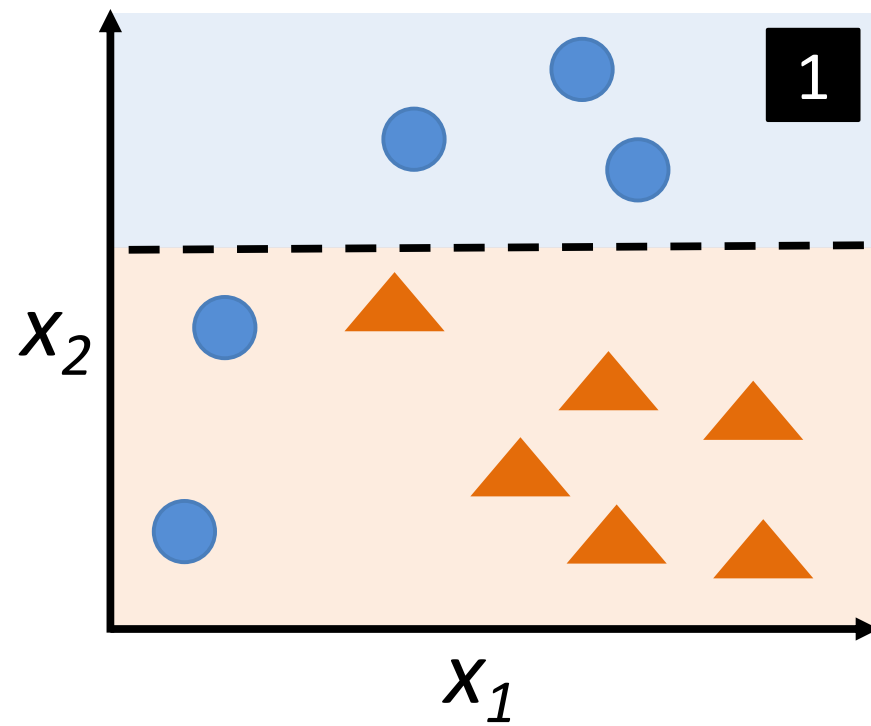


# Decision Tree Stumps









# Random Forests

# Random Forests

= Bagging w. trees + random feature subsets

# Random Feature Subset for each Tree or Node?

**Tin Kam Ho** used the “**random subspace method**,” where each tree got a random subset of features.

“Our method relies on an autonomous, pseudo-random procedure to select a small number of dimensions from a given feature space ...”

- Ho, Tin Kam. “The random subspace method for constructing decision forests.” IEEE transactions on pattern analysis and machine intelligence 20.8 (1998): 832-844.

**“Trademark” random forest:**

*“... random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on.”*

- **Breiman**, Leo. “Random Forests” Machine learning 45.1 (2001): 5-32.



# Random Feature Subset for each Tree or Node?

Tin Kam Ho used the “random subspace method,” where each tree got a random subset of features.

“Our method relies on an autonomous, pseudo-random procedure to select a small number of dimensions from a given feature space ...”

- Ho, Tin Kam. “The random subspace method for constructing decision forests.” IEEE transactions on pattern analysis and machine intelligence 20.8 (1998): 832-844.

“Trademark” random forest:

*“... random forest with random feature selection, at each node, a small group of input variables*

$$\text{num features} = \log_2 m + 1$$

where  $m$  is the number of input features

*random, at*

- Breiman, Leo. “Random Forests” Ma

2.

In contrast to the original publication  
[Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001]  
the scikit-learn implementation combines classifiers by averaging their  
probabilistic prediction, instead of letting each classifier vote for a single  
class.

"Soft Voting"

**Will discuss Random Forests  
and feature importance in  
*Feature Selection* lecture**

# (Loose) Upper Bound for the Generalization Error

Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 2001

$$\mathbf{PE} \leq \frac{\bar{\rho} \cdot (1 - s^2)}{s^2}$$

$\bar{\rho}$  : Average correlation among trees

$s$  : "Strength" of the ensemble

# Extremely Randomized Trees (ExtraTrees)

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63(1), 3-42.

Random Forest random components:

1) \_\_\_\_\_

2) \_\_\_\_\_

ExtraTrees algorithm adds one more random component

3) \_\_\_\_\_

# Stacking

# Stacking Algorithm

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

---

**Algorithm 19.7 Stacking**

---

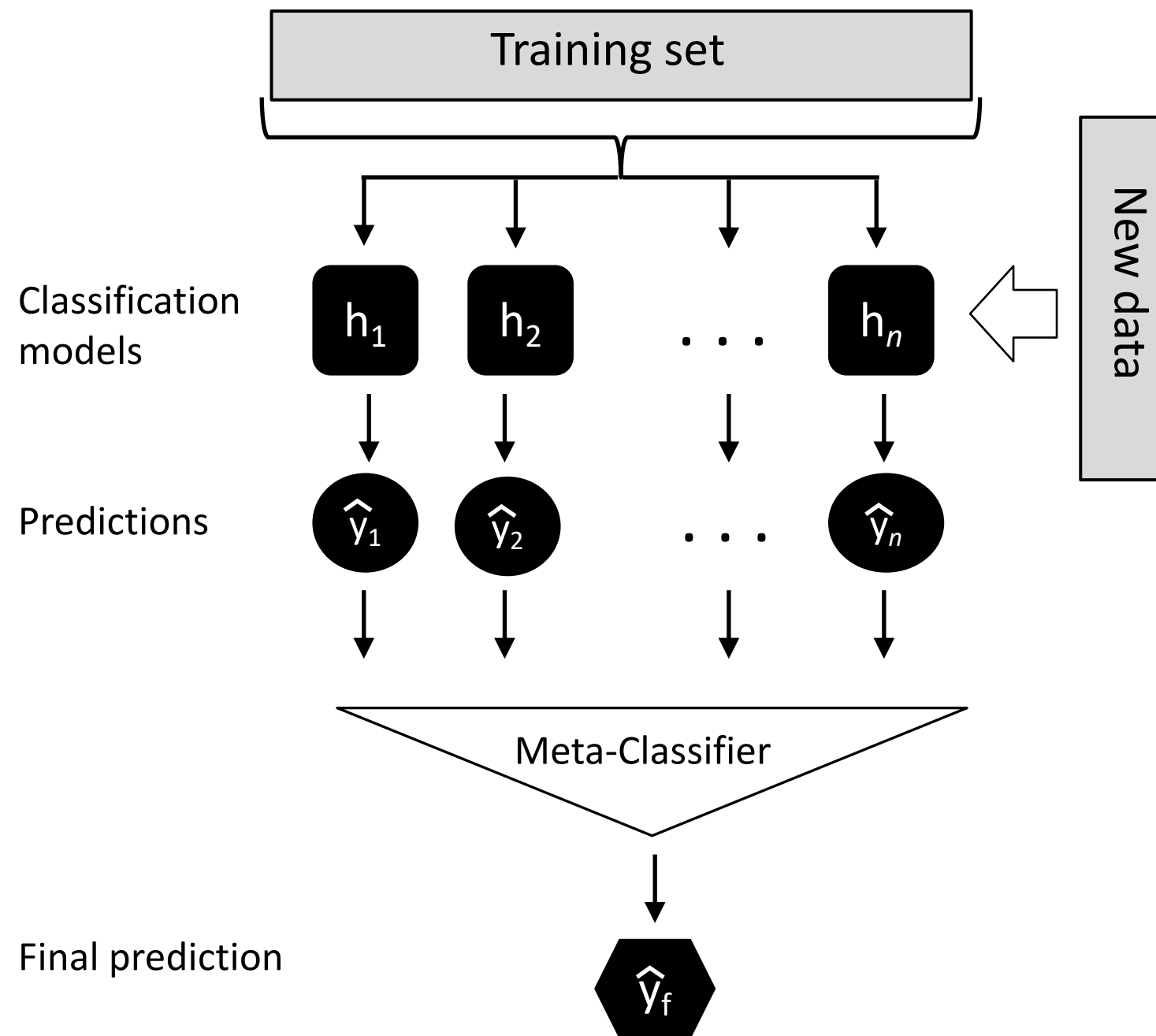
**Input:** Training data  $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^m$  ( $\mathbf{x}_i \in \mathbb{R}^n$ ,  $y_i \in \mathcal{Y}$ )

**Output:** An ensemble classifier  $H$

- 1: Step 1: Learn first-level classifiers
  - 2: **for**  $t \leftarrow 1$  to  $T$  **do**
  - 3:     Learn a base classifier  $h_t$  based on  $\mathcal{D}$
  - 4: **end for**
  - 5: Step 2: Construct new data sets from  $\mathcal{D}$
  - 6: **for**  $i \leftarrow 1$  to  $m$  **do**
  - 7:     Construct a new data set that contains  $\{\mathbf{x}'_i, y_i\}$ , where  $\mathbf{x}'_i = \{h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \dots, h_T(\mathbf{x}_i)\}$
  - 8: **end for**
  - 9: Step 3: Learn a second-level classifier
  - 10: Learn a new classifier  $h'$  based on the newly constructed data set
  - 11: **return**  $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_T(\mathbf{x}))$
- 

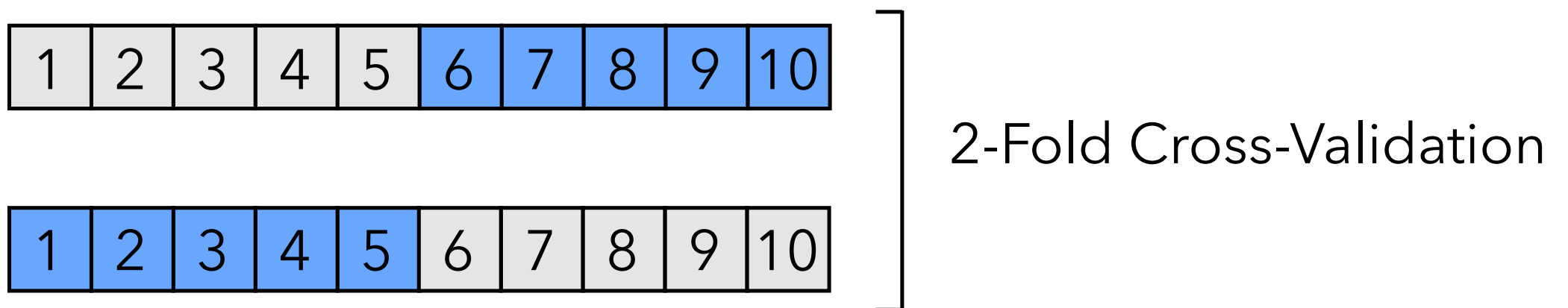
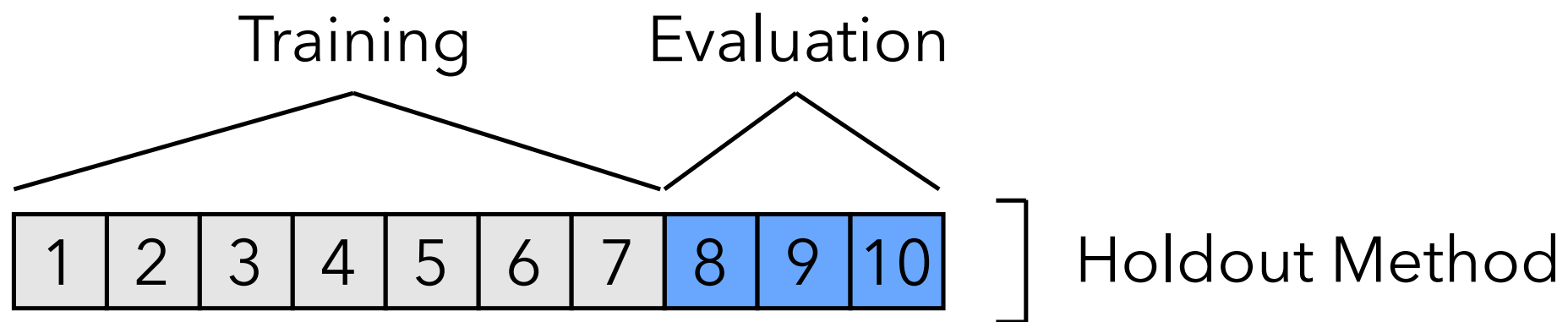
Tang, J., S. Alelyani, and H. Liu. "Data Classification: Algorithms and Applications." Data Mining and Knowledge Discovery Series, CRC Press (2015): pp. 498-500.

# Stacking Algorithm

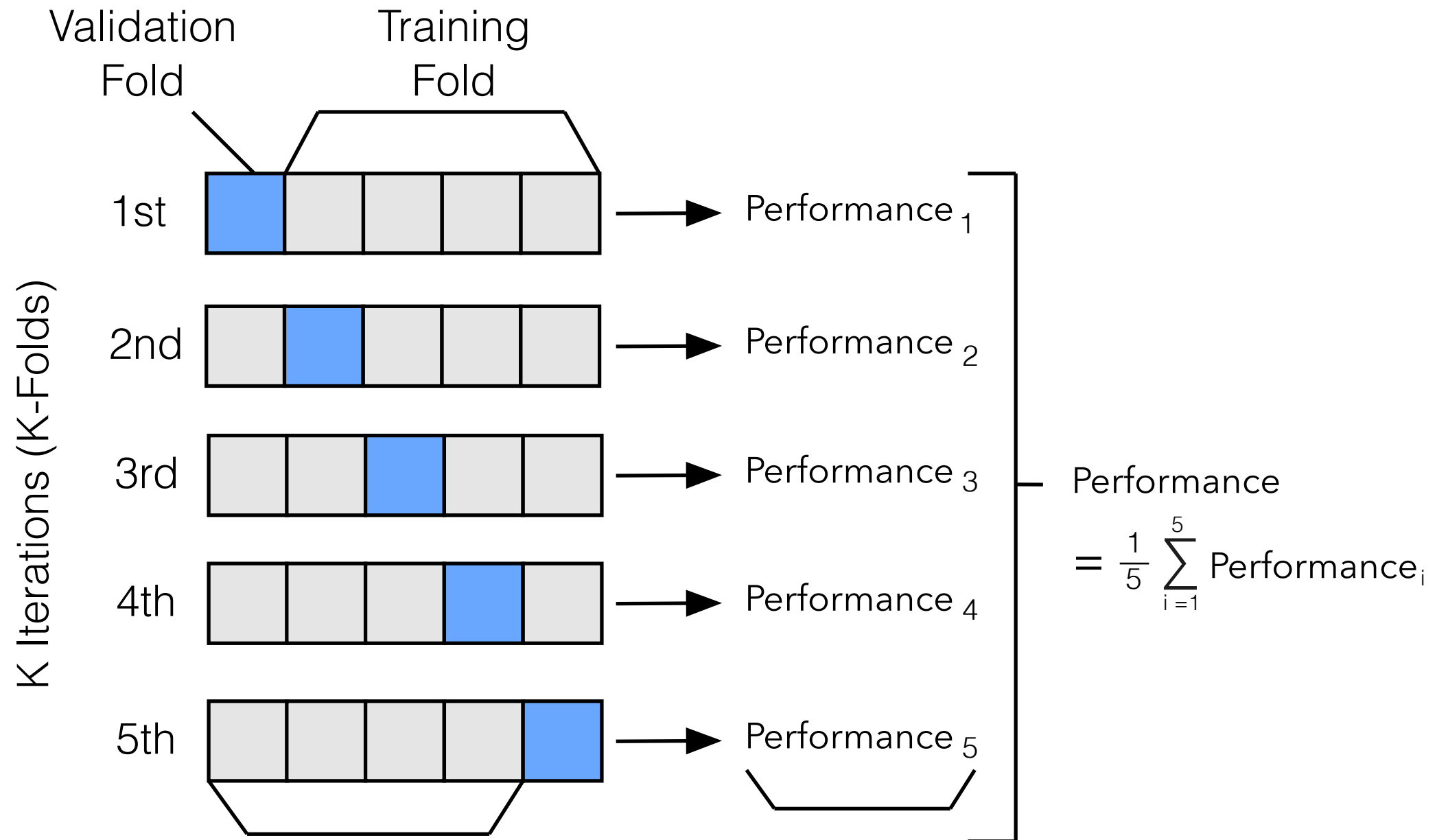


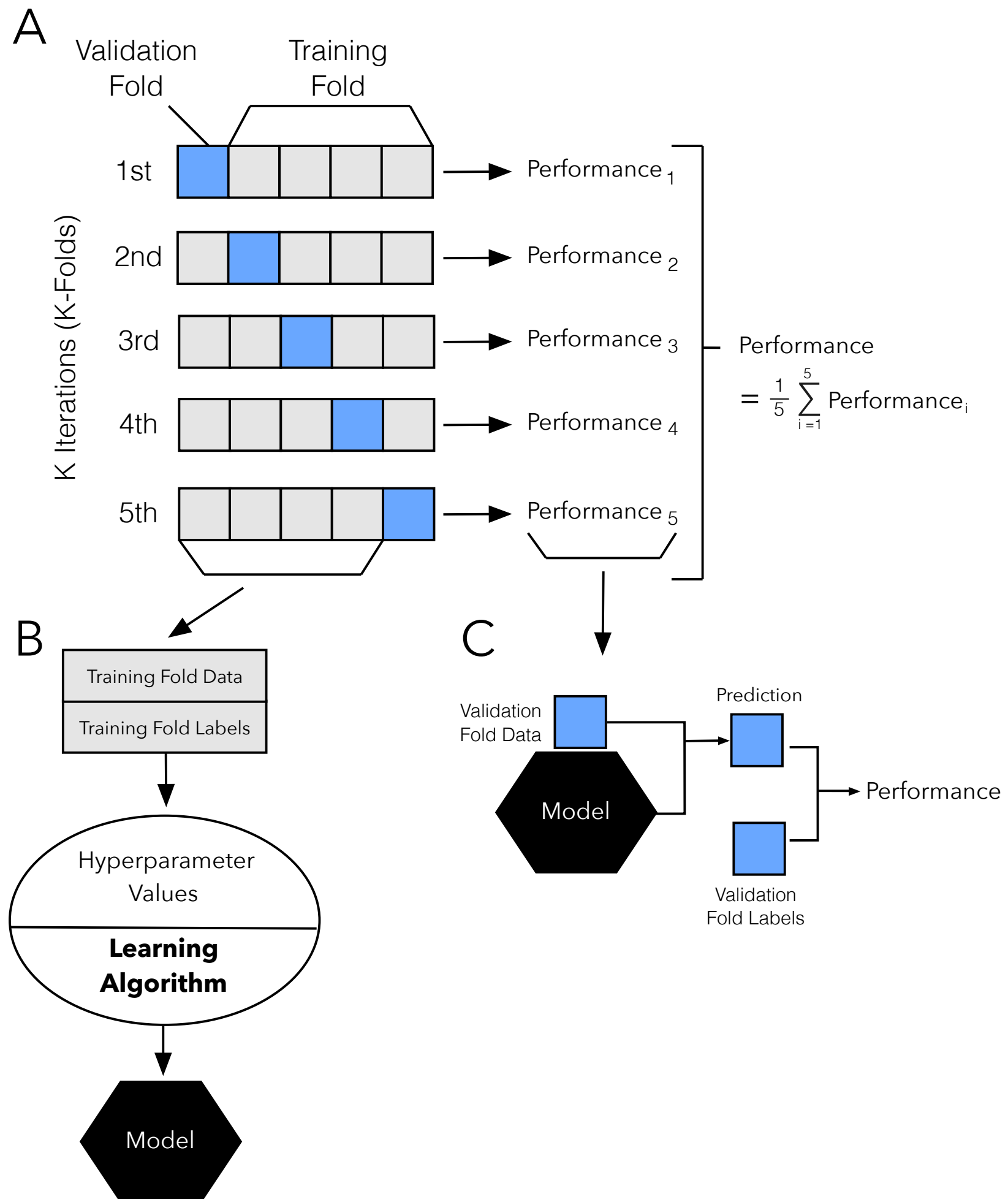


# Cross-Validation



# *k*-fold Cross-Validation





# 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**

---

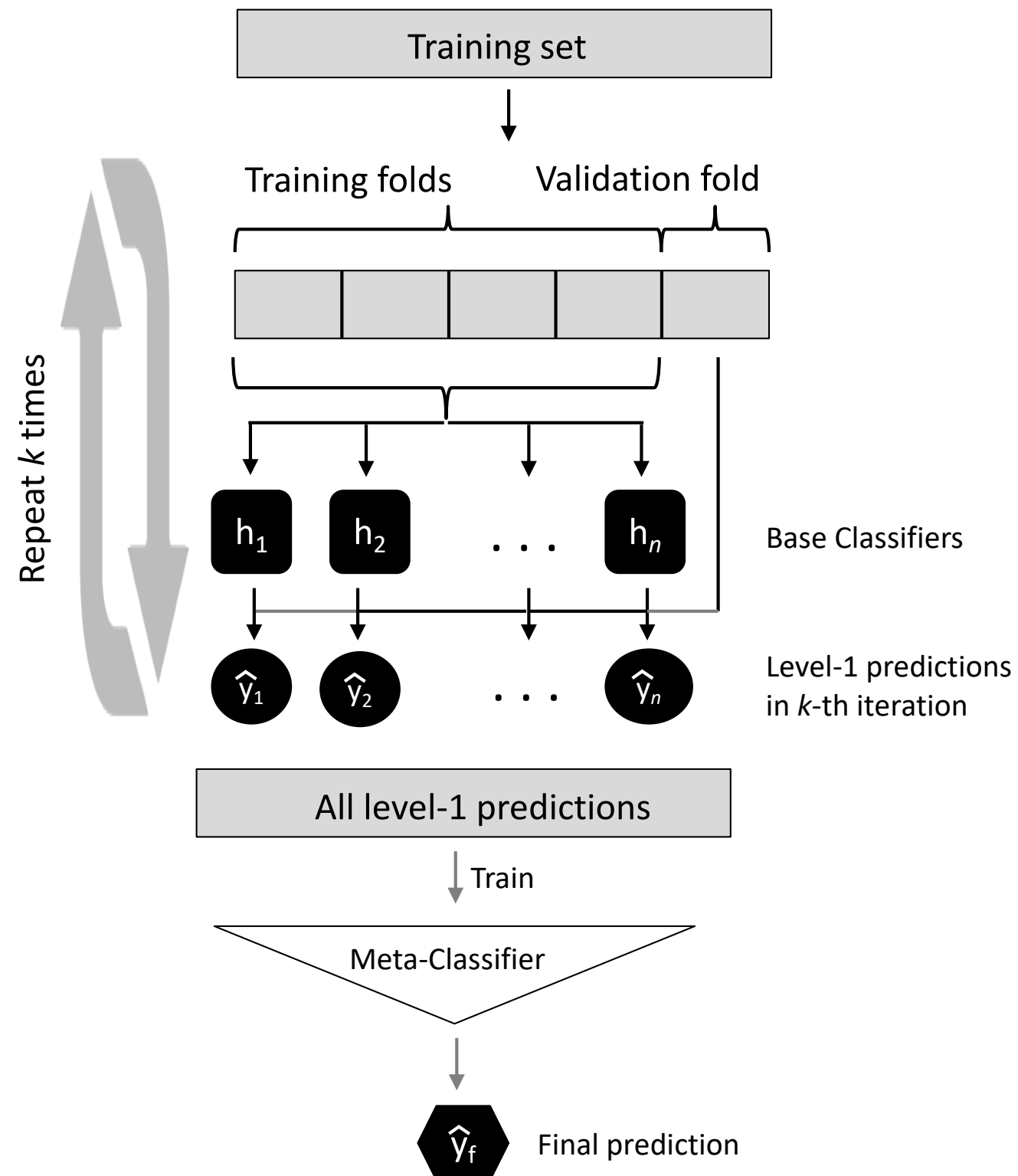
**Input:** Training data  $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^m$  ( $\mathbf{x}_i \in \mathbb{R}^n$ ,  $y_i \in \mathcal{Y}$ )

**Output:** An ensemble classifier  $H$

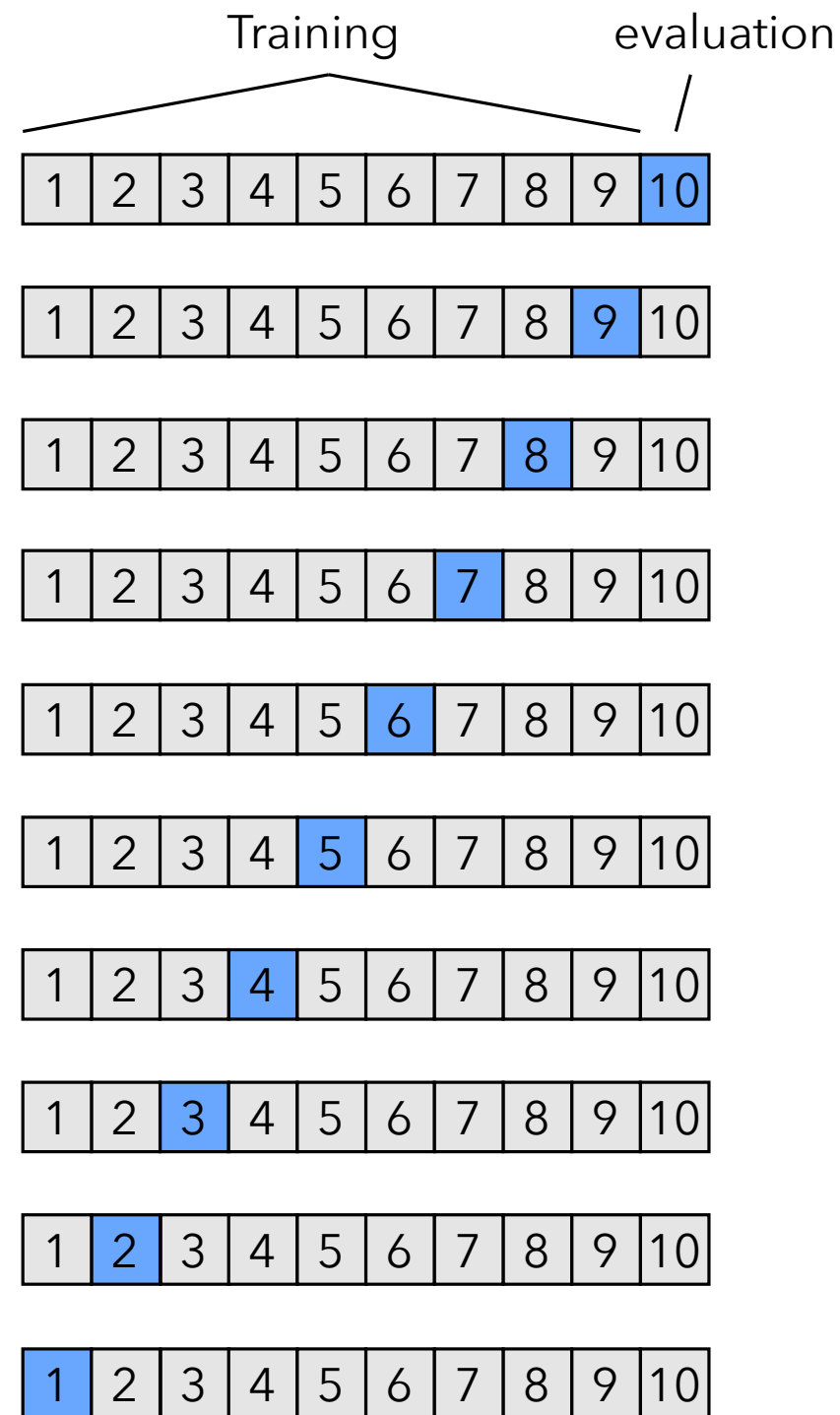
- 1: Step 1: Adopt cross validation approach in preparing a training set for second-level classifier
  - 2: Randomly split  $\mathcal{D}$  into  $K$  equal-size subsets:  $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_K\}$
  - 3: **for**  $k \leftarrow 1$  to  $K$  **do**
  - 4:  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

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



# Leave-One-Out CV



# Demo

07\_ensembles\_demo.ipynb

# Reading Assignments

Python Machine Learning, 2nd Ed., Ch07