Lecture 07

Ensemble Methods

STAT 479: Machine Learning, Fall 2019
Sebastian Raschka
http://stat.wisc.edu/~sraschka/teaching/stat479-fs2019/

Announcements First

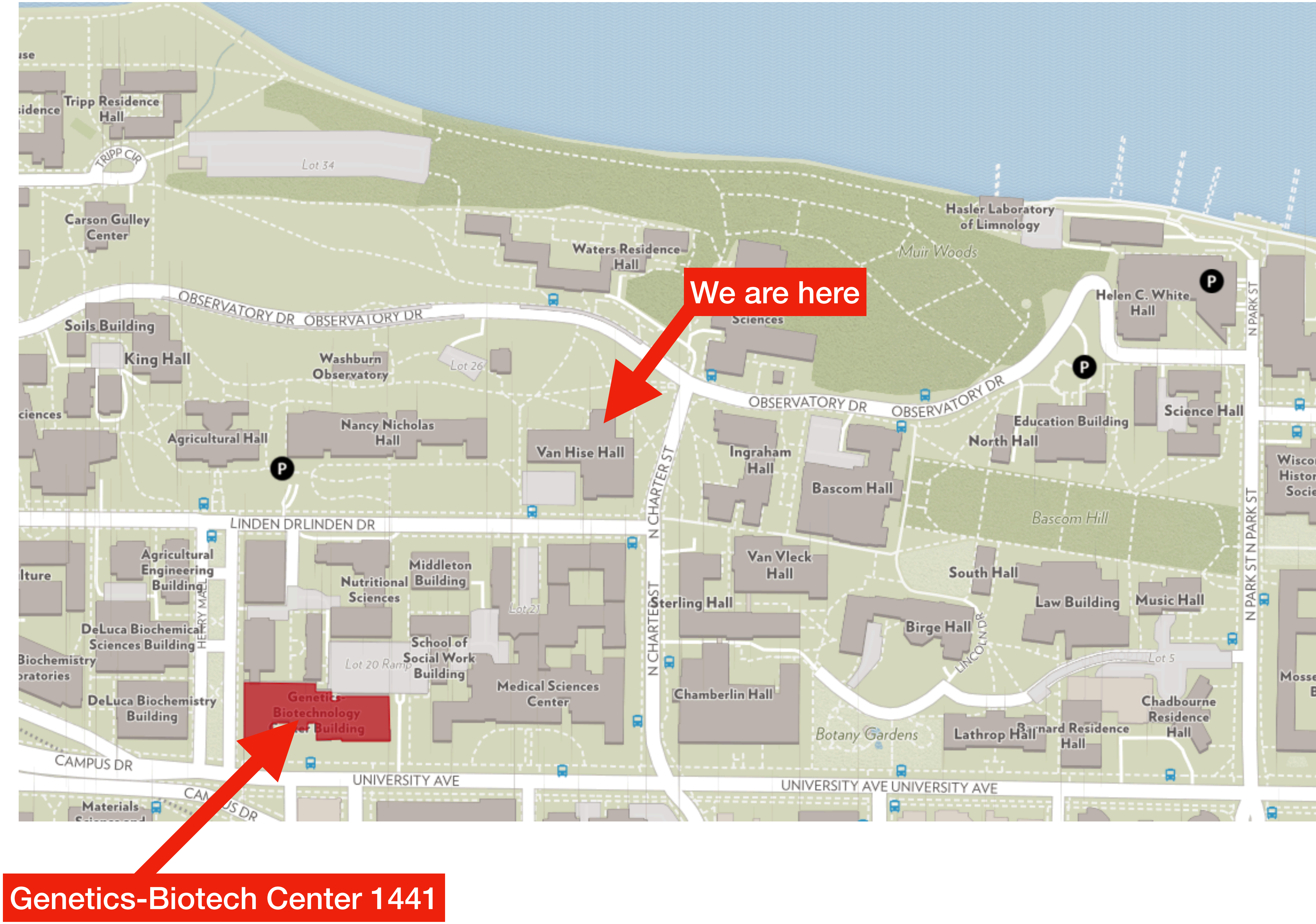
[DataSci] Machine Learning Series
Scheduled: Thursday Oct 10, 6:30 PM
Location: Genetics-Biotech Center 1441

Soft-biometric characteristics include a person's age, gender, race, and health status. As many Deep Learning-centric applications are developed in recent years, the automatic extraction of soft biometric attributes can happen without the user's agreement, thereby raising several privacy concerns. <u>This talk will introduce how to extract soft-biometric attributes from facial images, as well as how to conceal soft-biometric information for enhancing privacy.</u>

Don't worry if you do not have programming experience with Python! Dr. Rashka will also give a tutorial introducing PyTorch and how we can use it to train a simple gender classifier and ordinal regression model for estimating the apparent age from face images.

Best,

Lareina Liu
UW Data Science Club
.Data


We are here
Genetics-Biotech Center 1441

Example Exam Question

(6 points) Does the (computational) time complexity of a k-Nearest Neighbor classifier grow linearly, quadratically, or exponentially with the number of samples in the training dataset? Explain your answer in 1-2 sentences.

Example Exam Question

(6 points) Can you represent the following boolean function with a decision tree? If you answer "no," explain why in 1-2 sentences. Otherwise, draw a decision tree that separates the data records perfectly.

x_1	x_2	$f(x_1, x_2)$
1	1	0
0	0	0
1	0	1
0	1	0

Part I: Introduction

- Lecture 1: What is Machine Learning? An Overview.
- Lecture 2: Intro to Supervised Learning: KNN

Part II: Computational Foundations

- Lecture 3: Using Python, Anaconda, IPython, Jupyter Notebooks
- Lecture 4: Scientific Computing with NumPy, SciPy, and Matplotlib
- Lecture 5: Data Preprocessing and Machine Learning with Scikit-Learn

Part III: Tree-Based Methods

- Lecture 6: Decision Trees
- Lecture 7: Ensemble Methods

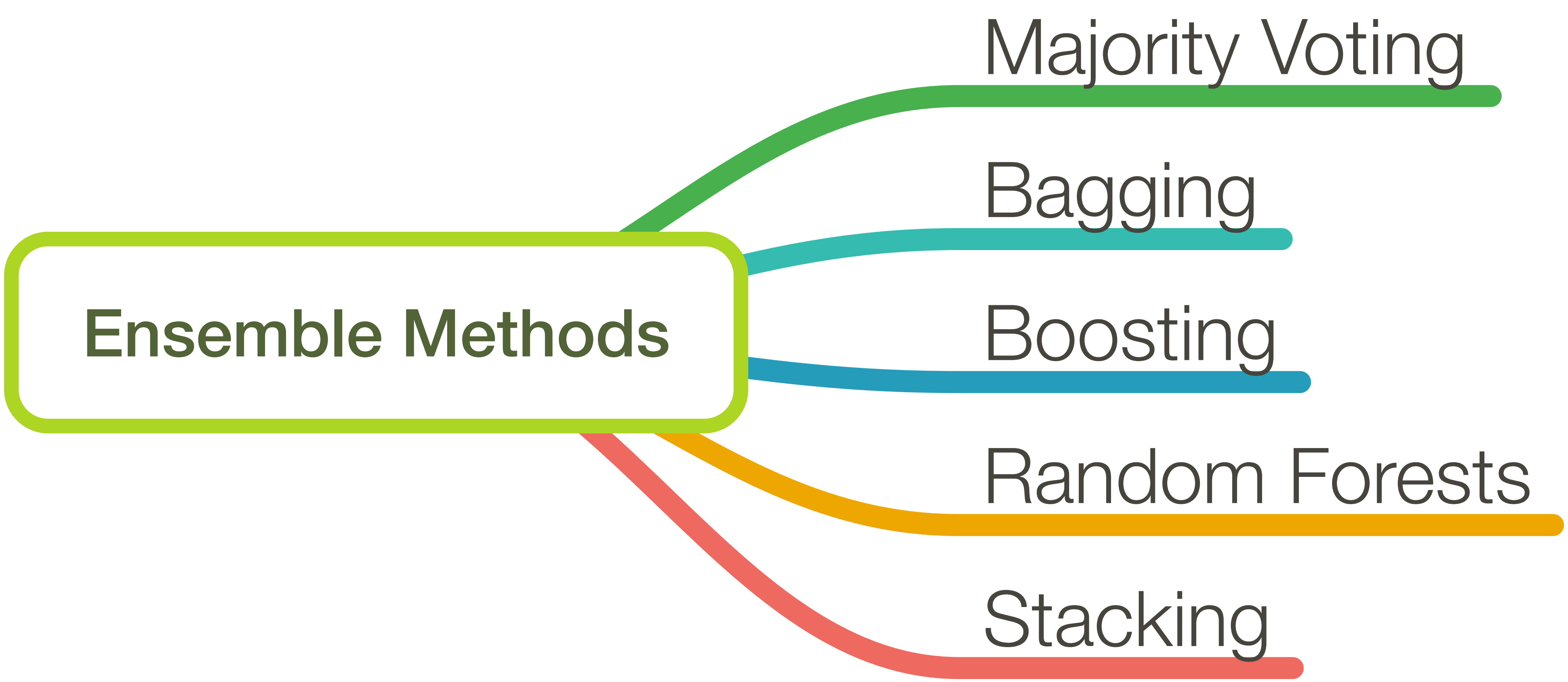
Part IV: Evaluation

- Lecture 8: Model Evaluation 1: Introduction to Overfitting and Underfitting
- Lecture 9: Model Evaluation 2: Uncertainty Estimates and Resampling
- Lecture 10: Model Evaluation 3: Model Selection and Cross-Validation
- Lecture 11: Model Evaluation 4: Algorithm Selection and Statistical Tests
- Lecture 12: Model Evaluation 5: Performance Metrics

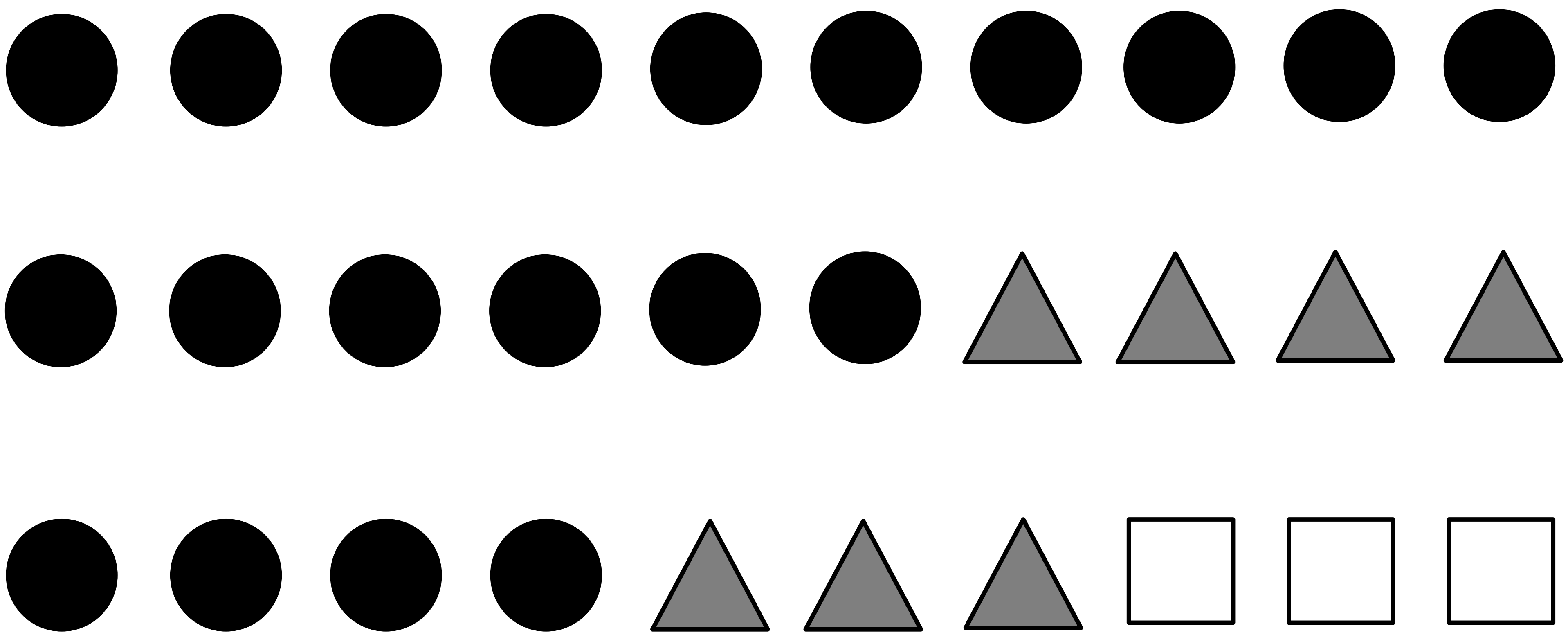
Part V: Dimensionality Reduction

- Lecture 13: Feature Selection
- Lecture 14: Feature Extraction

Lecture Overview



Majority Voting

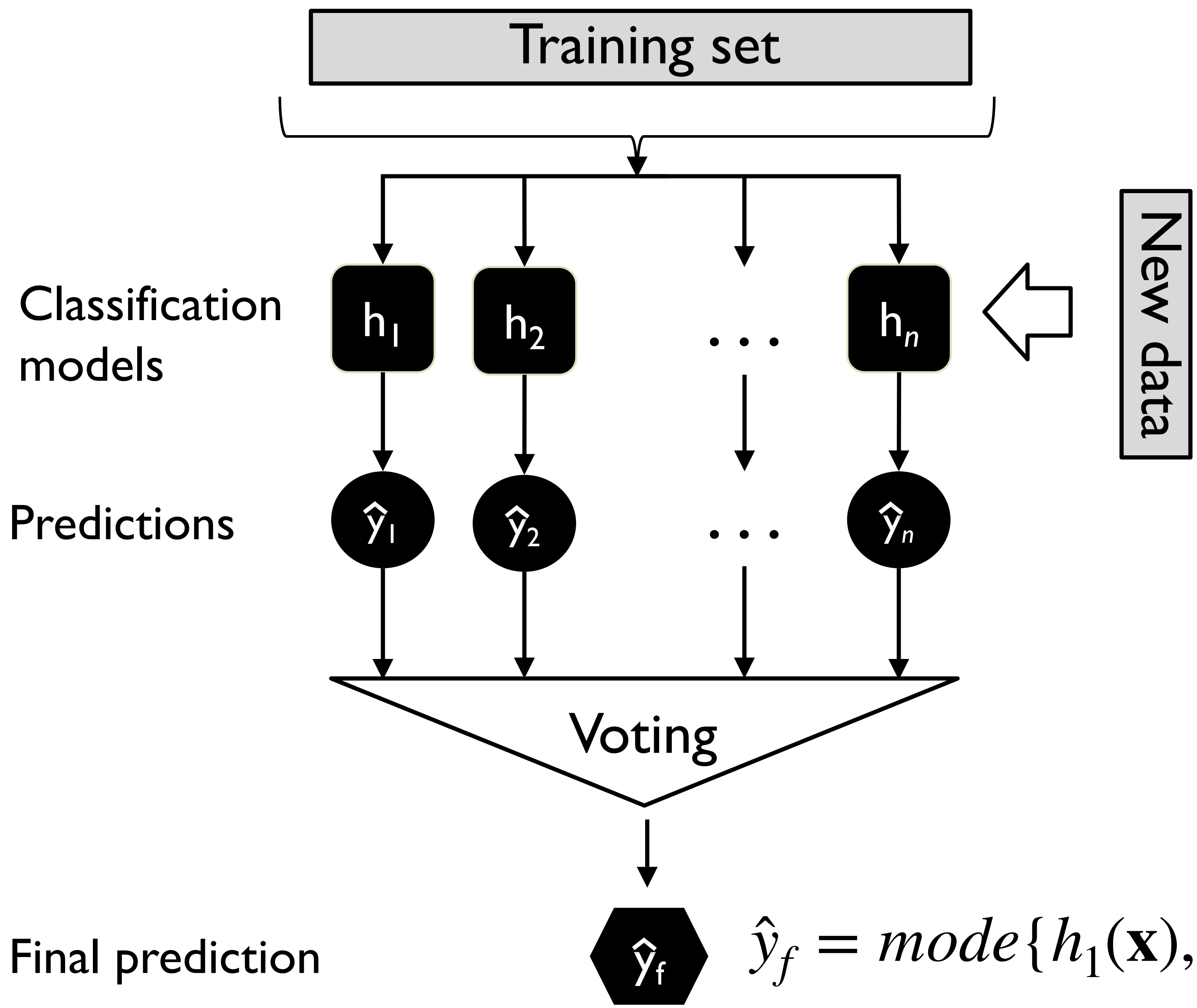


Unanimity

Majority

Plurality

Majority Vote Classifier



$$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots h_n(\mathbf{x})\}$$

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

Why Majority Vote?

- assume *n* independent classifiers with a base error rate ϵ
- 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)

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

Why Majority Vote?

- assume *n* independent classifiers with a base error rate ϵ
- 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)

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

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

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

(Probability mass func. of a binomial distr.)

Why Majority Vote?

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

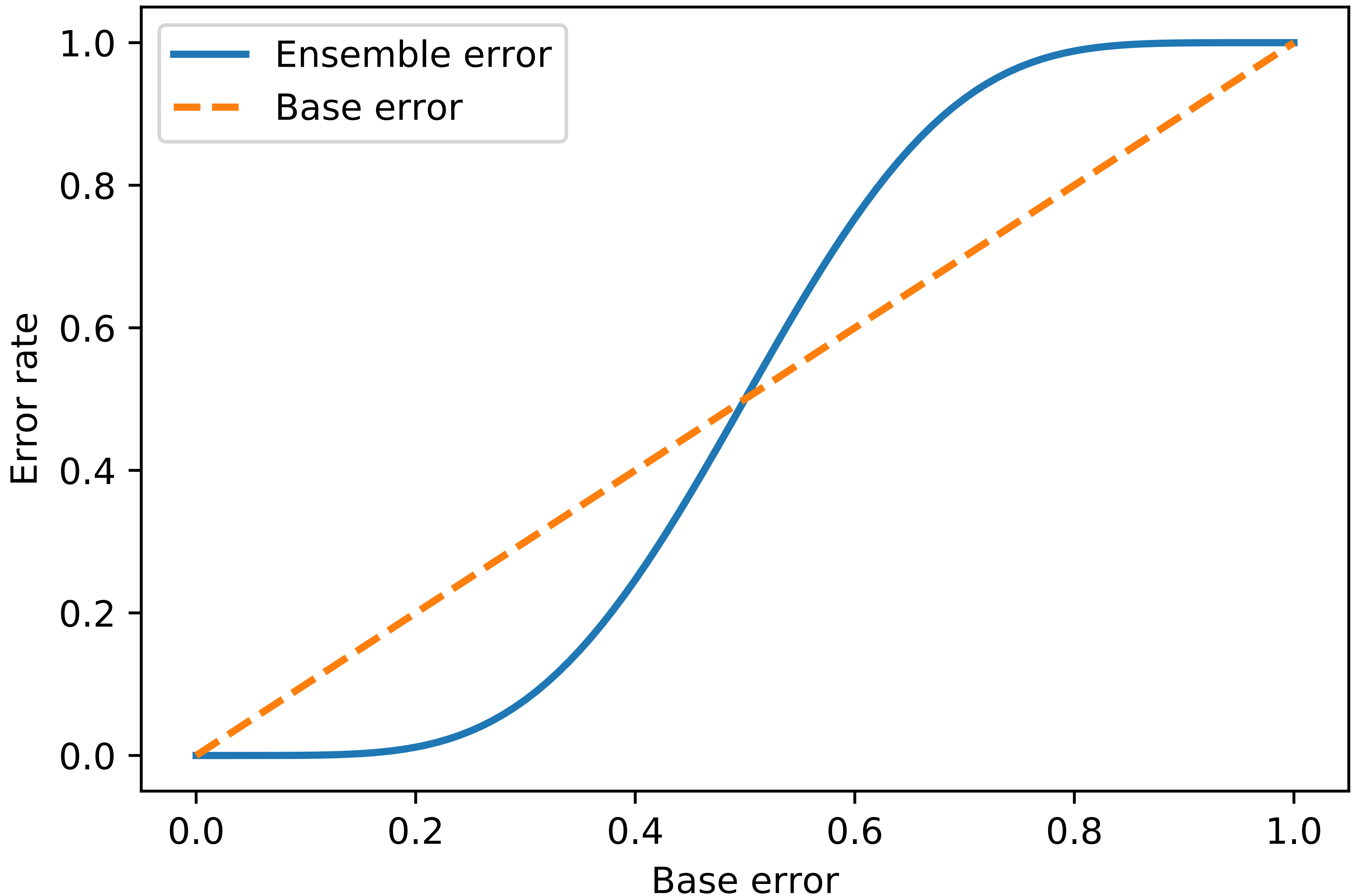
$$P(k) = \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k} \qquad k > \lceil n/2 \rceil$$

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 $w_i = 1/n, \forall w_i \in \{w_1, \ldots, w_n\}$

"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 $w_i = 1/n, \forall w_i \in \{w_1, \ldots, w_n\}$

"Soft" Voting

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

Binary classification example

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

$$h_1(\mathbf{x}) \rightarrow [0.9, 0.1]$$
$$h_2(\mathbf{x}) \rightarrow [0.8, 0.2]$$
$$h_3(\mathbf{x}) \rightarrow [0.4, 0.6]$$

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

Binary classification example

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

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

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

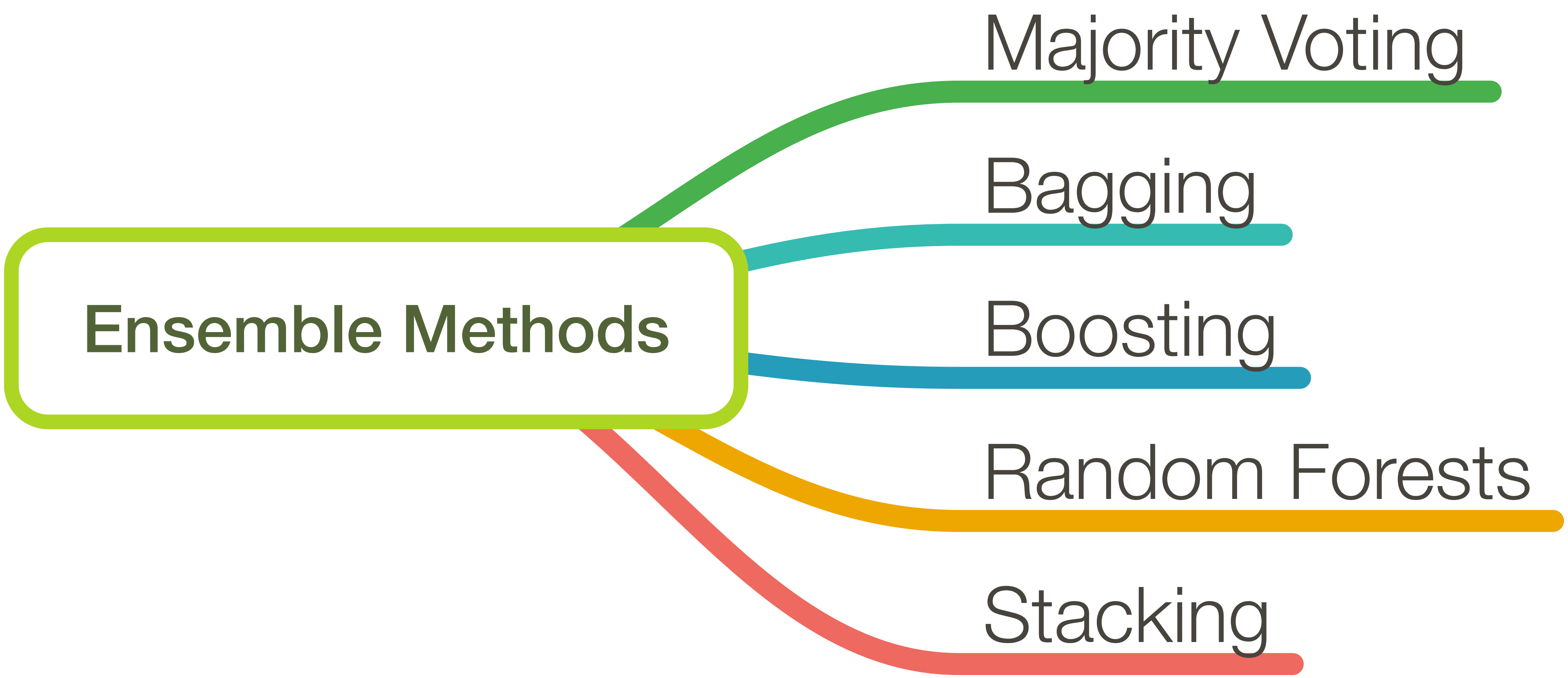
$$h_3(\mathbf{x}) \rightarrow [0.4, 0.6]$$

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

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

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

Overview



Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

Bagging

(Bootstrap Aggregating)

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

Bagging
(Bootstrap Aggregating)

Algorithm 1 Bagging

Let n be the number of bootstrap samples

for i=1 to n **do**

    Draw bootstrap sample of size m, $\mathcal{D}_i$

    Train base classifier h_i on $\mathcal{D}_i$

$\hat{y} = mode\{h_1(\mathbf{x}), ..., h_n(\mathbf{x})\}$

Bootstrap Sampling

Bootstrap Sampling

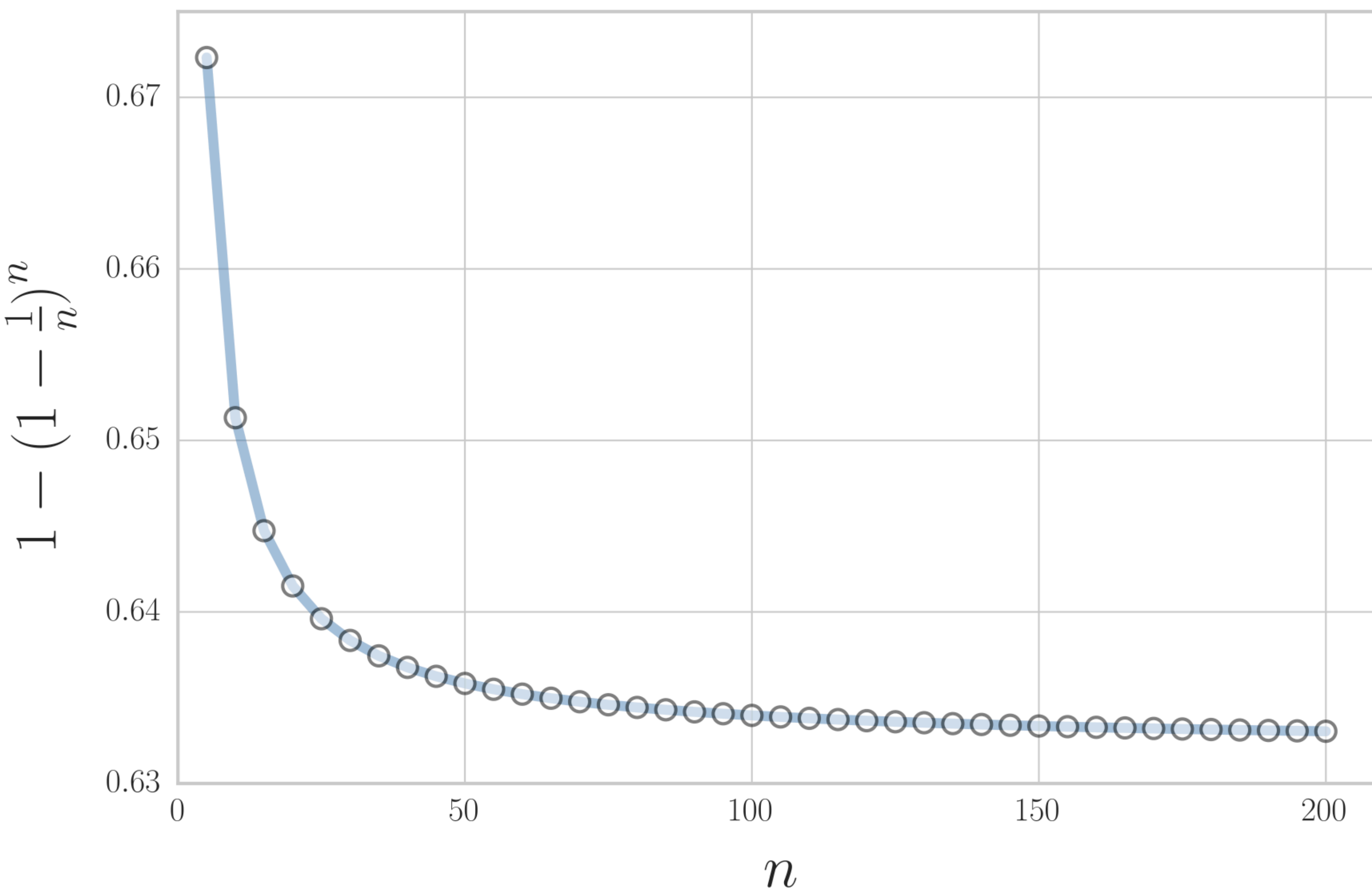
$$P(\textbf{not chosen}) = \left(1 - \frac{1}{n}\right)^n,$$

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

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

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

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

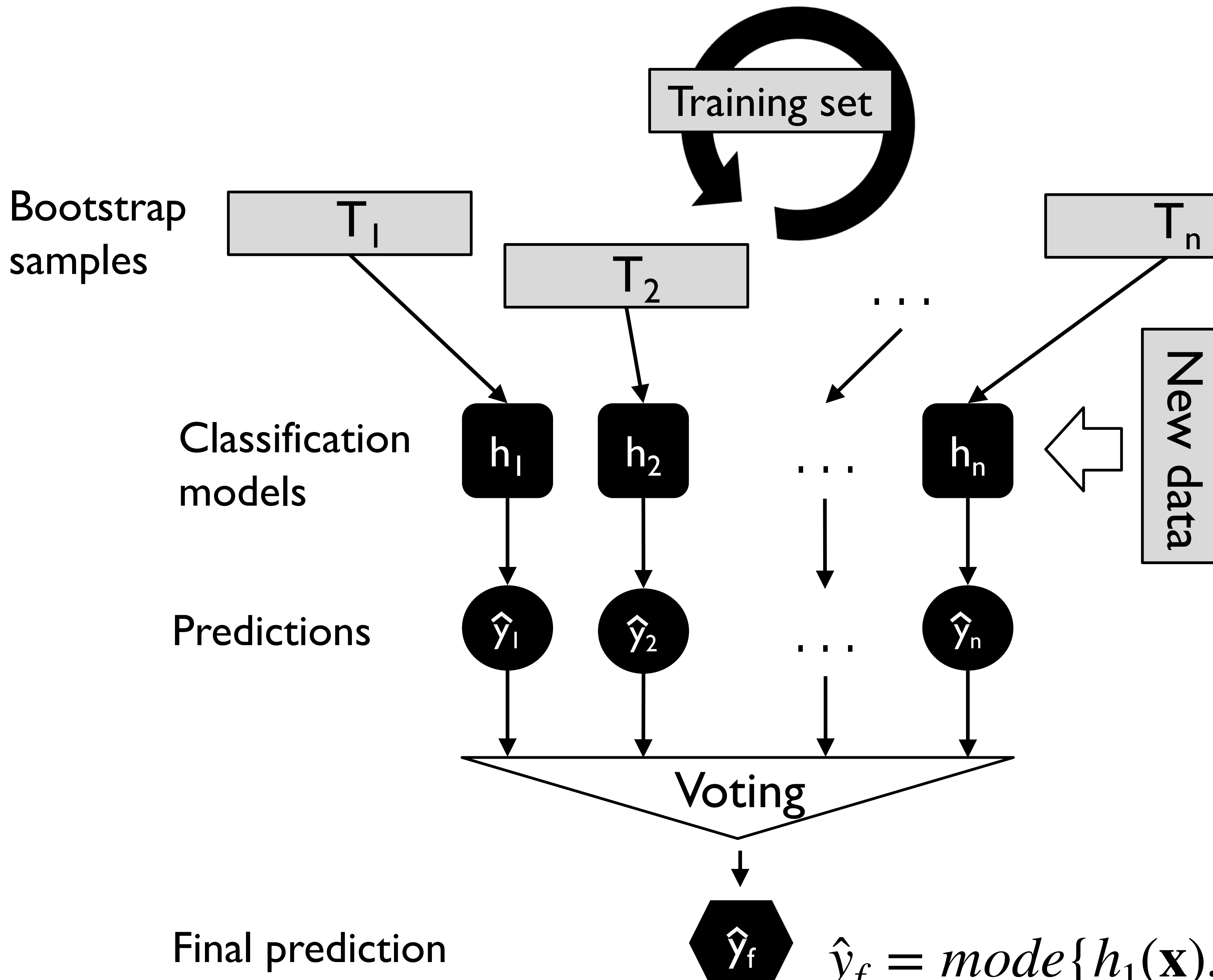


Bootstrap Sampling

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



$$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots h_n(\mathbf{x})\}$$

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

Bias-Variance Decomposition

Loss = Bias + Variance + Noise

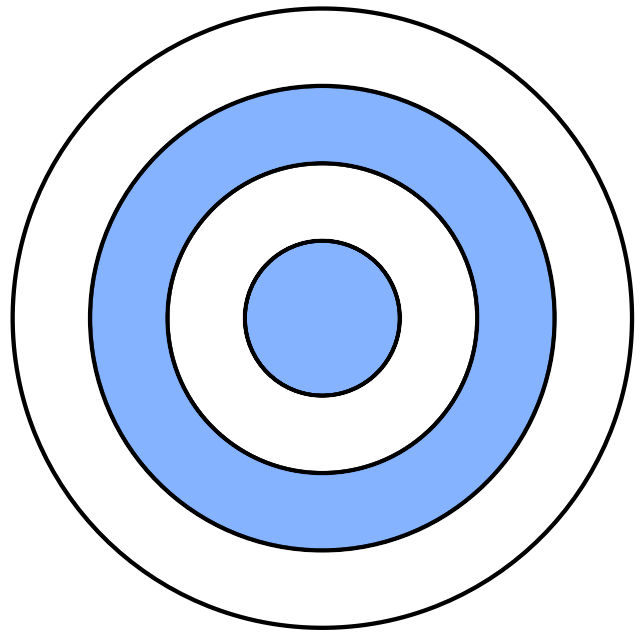
(more technical details in next lecture on model evaluation)

Bias-Variance Intuition

Low Variance
(Precise)

High Variance
(Not Precise)

Low Bias
(Accurate)



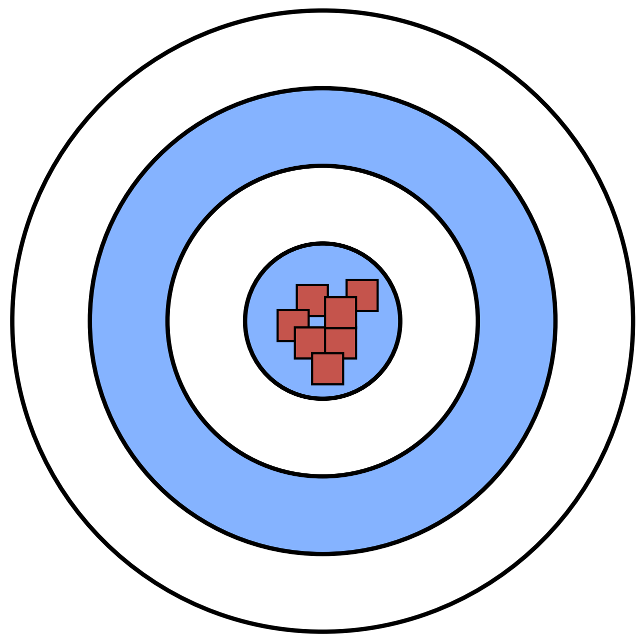
High Bias
(Not Accurate)

Bias-Variance Intuition

Low Variance
(Precise)

High Variance
(Not Precise)

Low Bias
(Accurate)



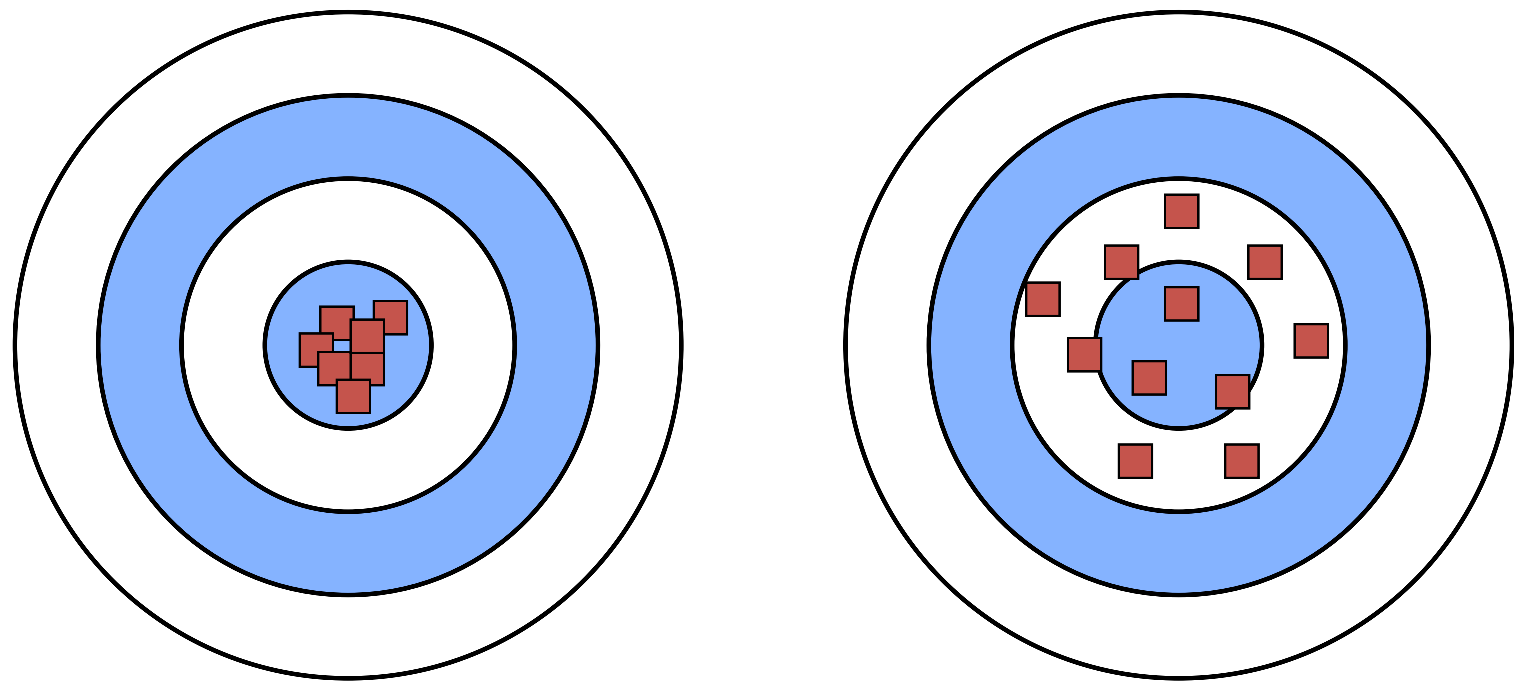
High Bias
(Not Accurate)

Bias-Variance Intuition

Low Variance
(Precise)

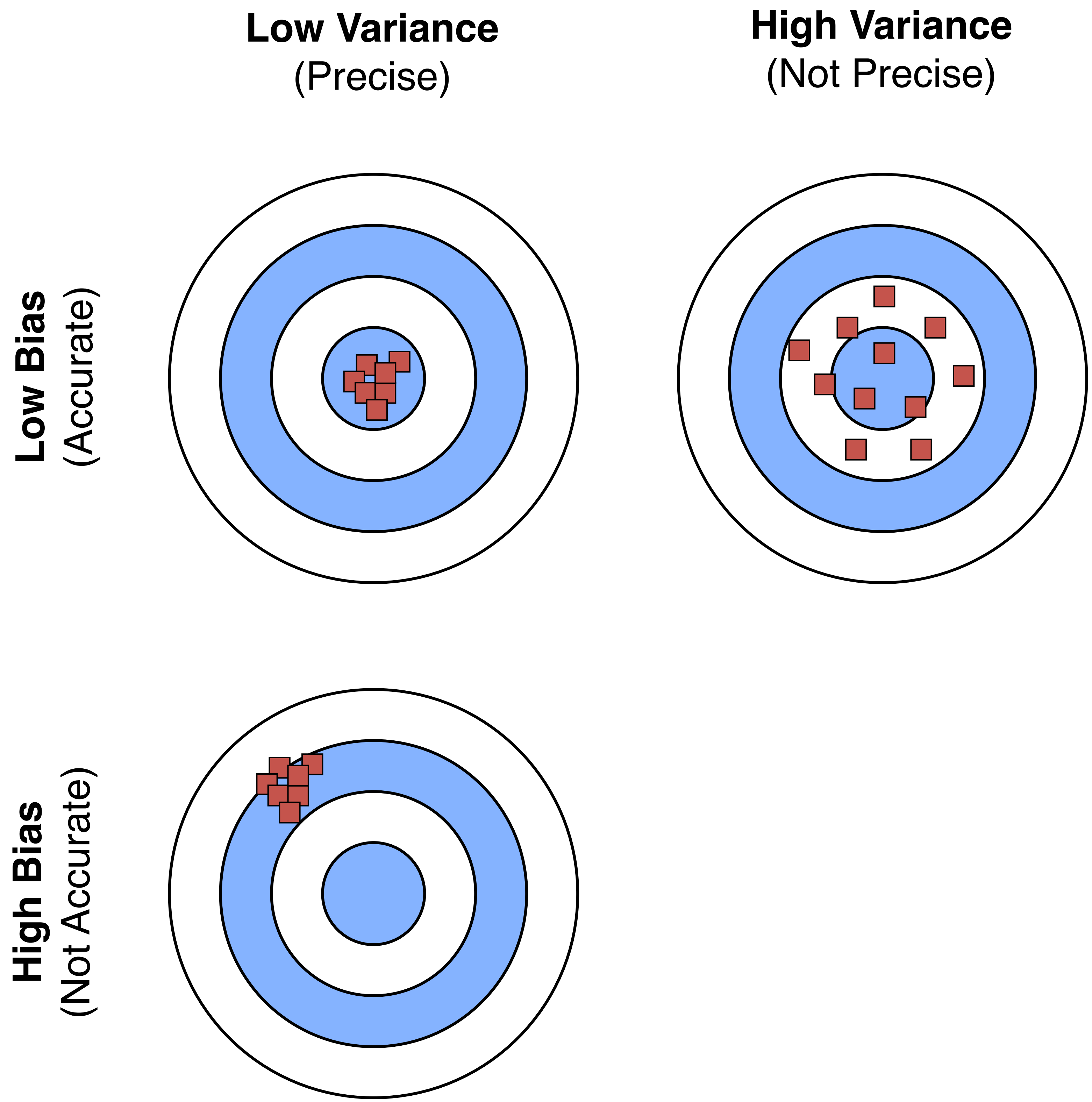
High Variance
(Not Precise)

Low Bias
(Accurate)

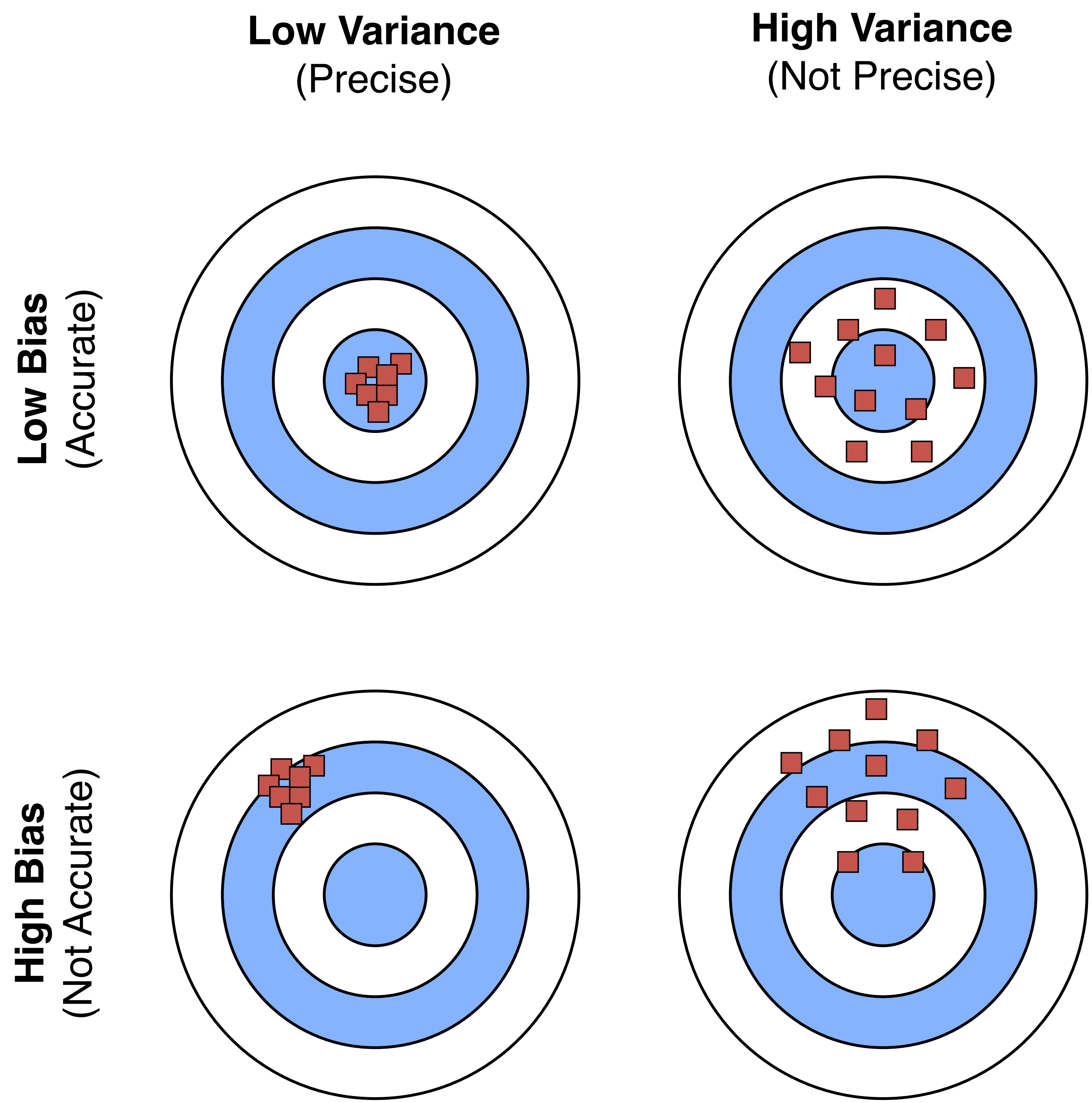


High Bias
(Not Accurate)

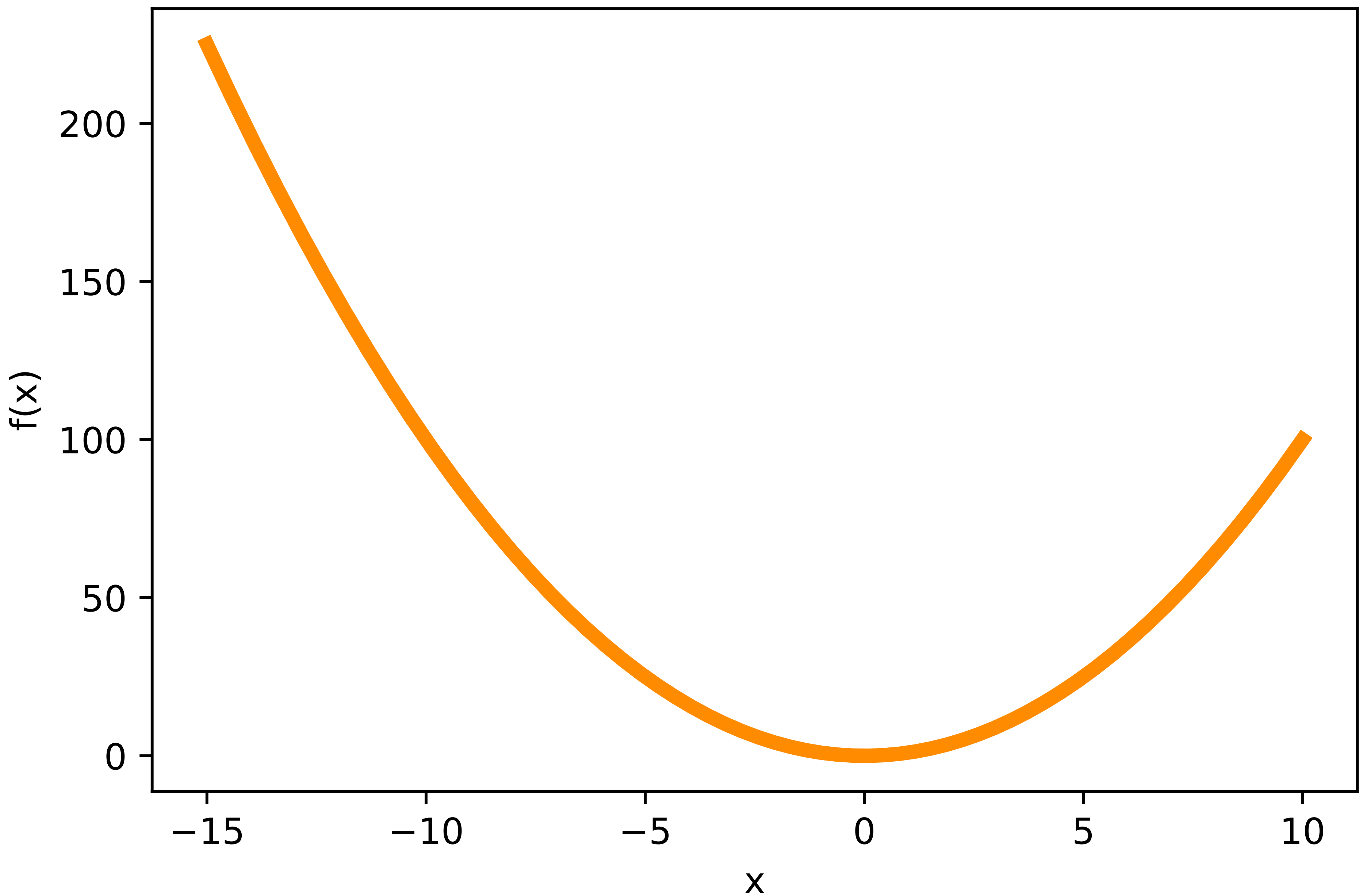
Bias-Variance Intuition



Bias-Variance Intuition

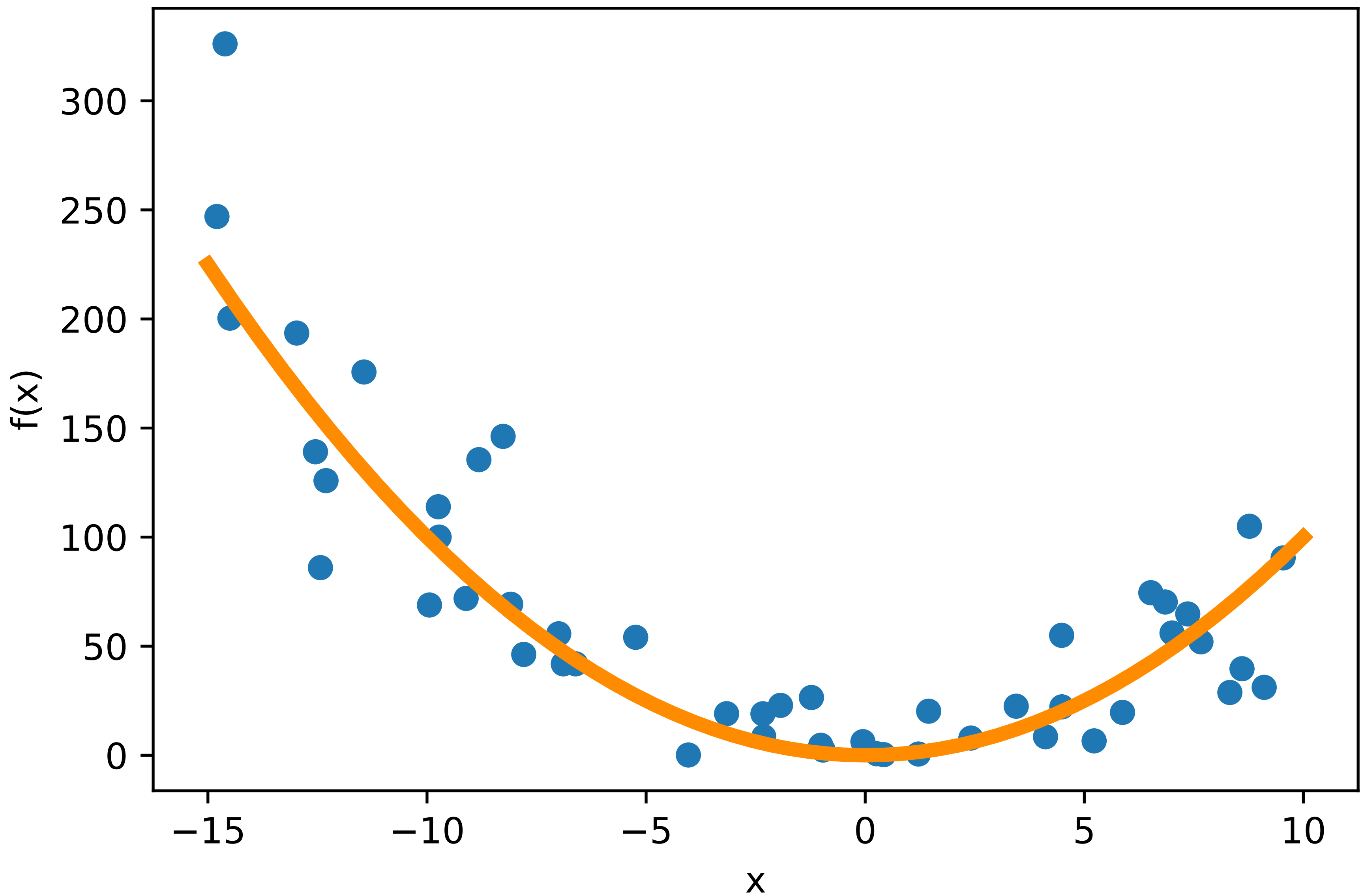


Bias and Variance Example



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

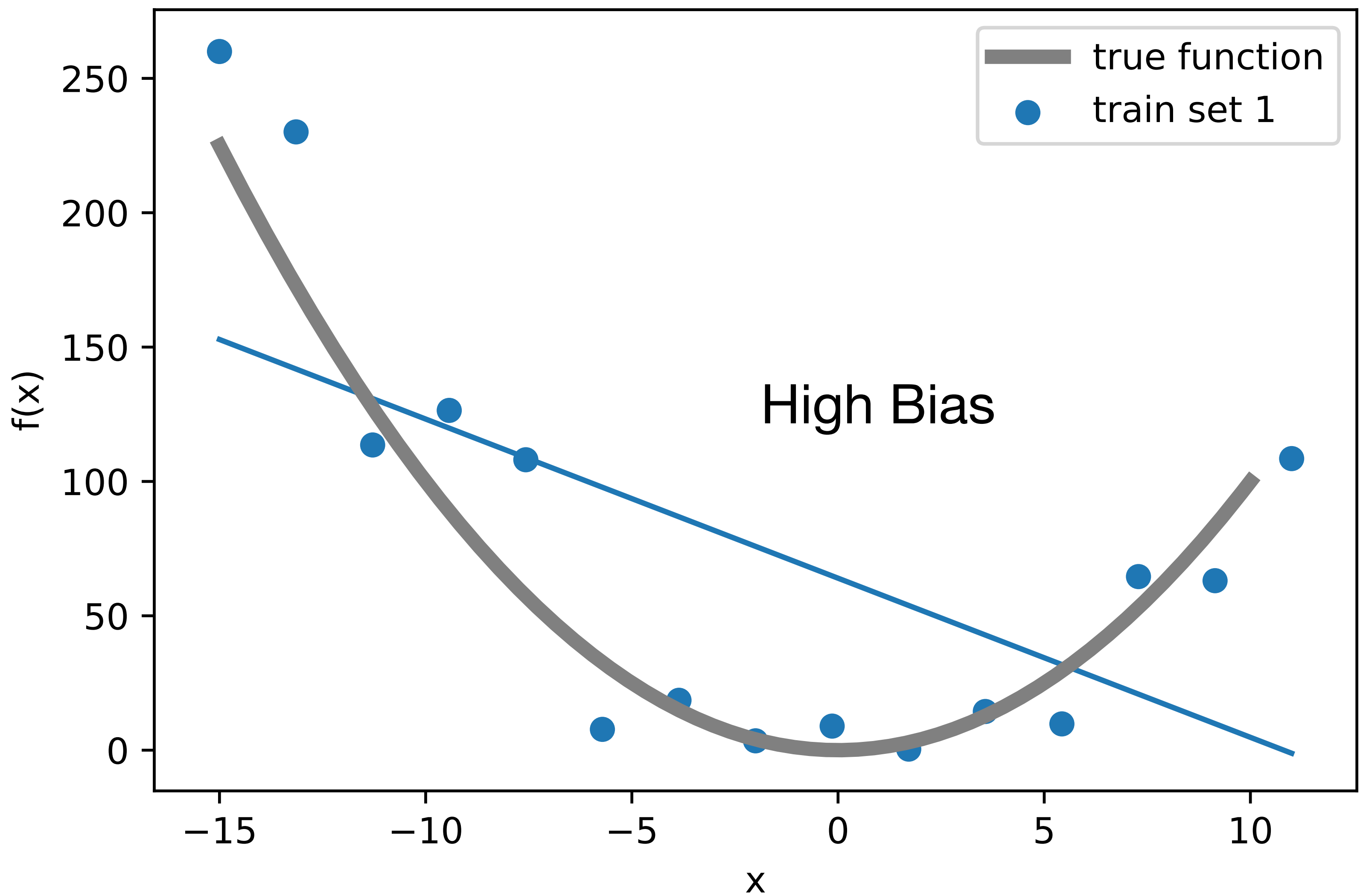
Bias and Variance Example



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

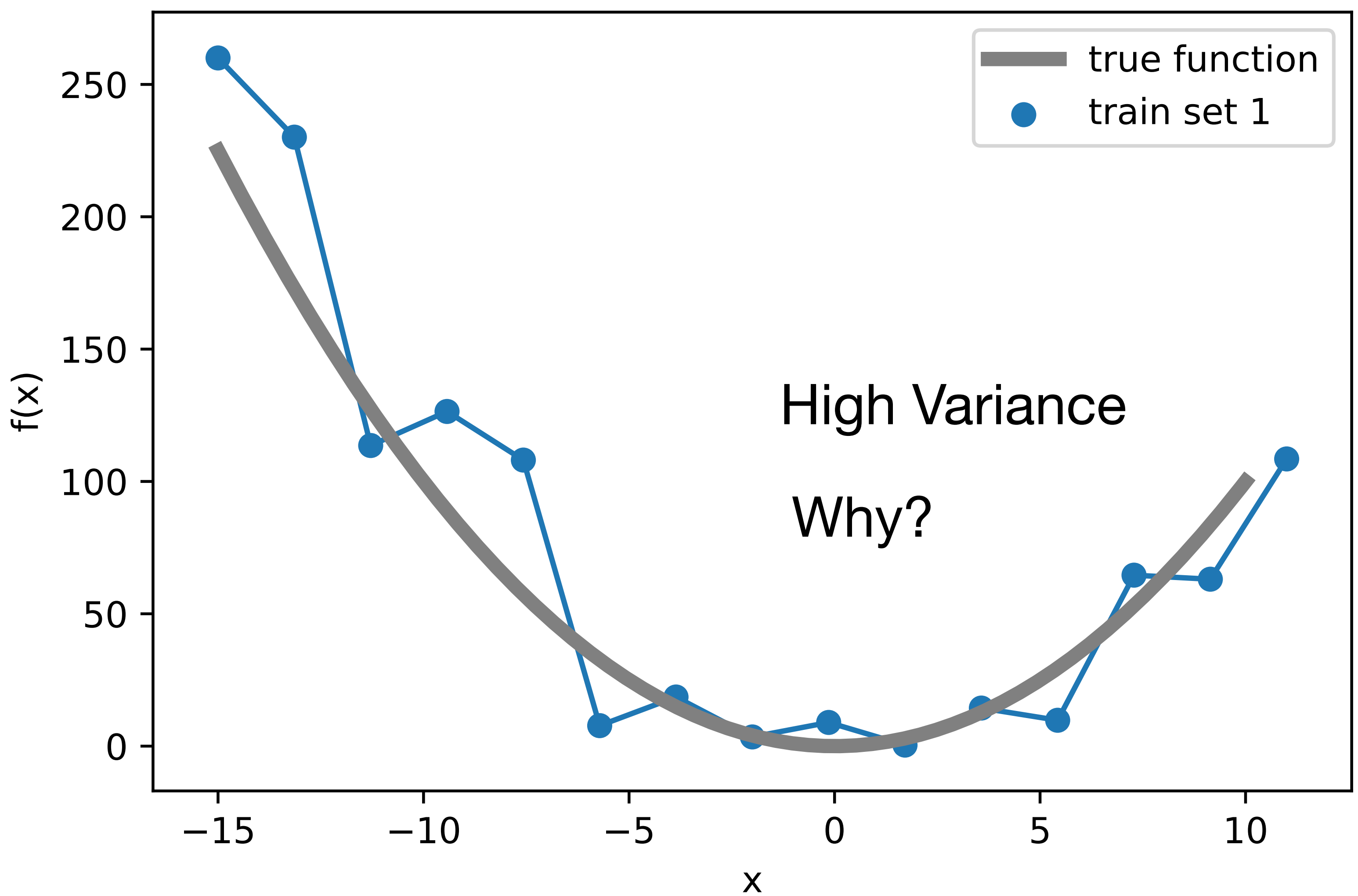


where f(x) is some true (target) function

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

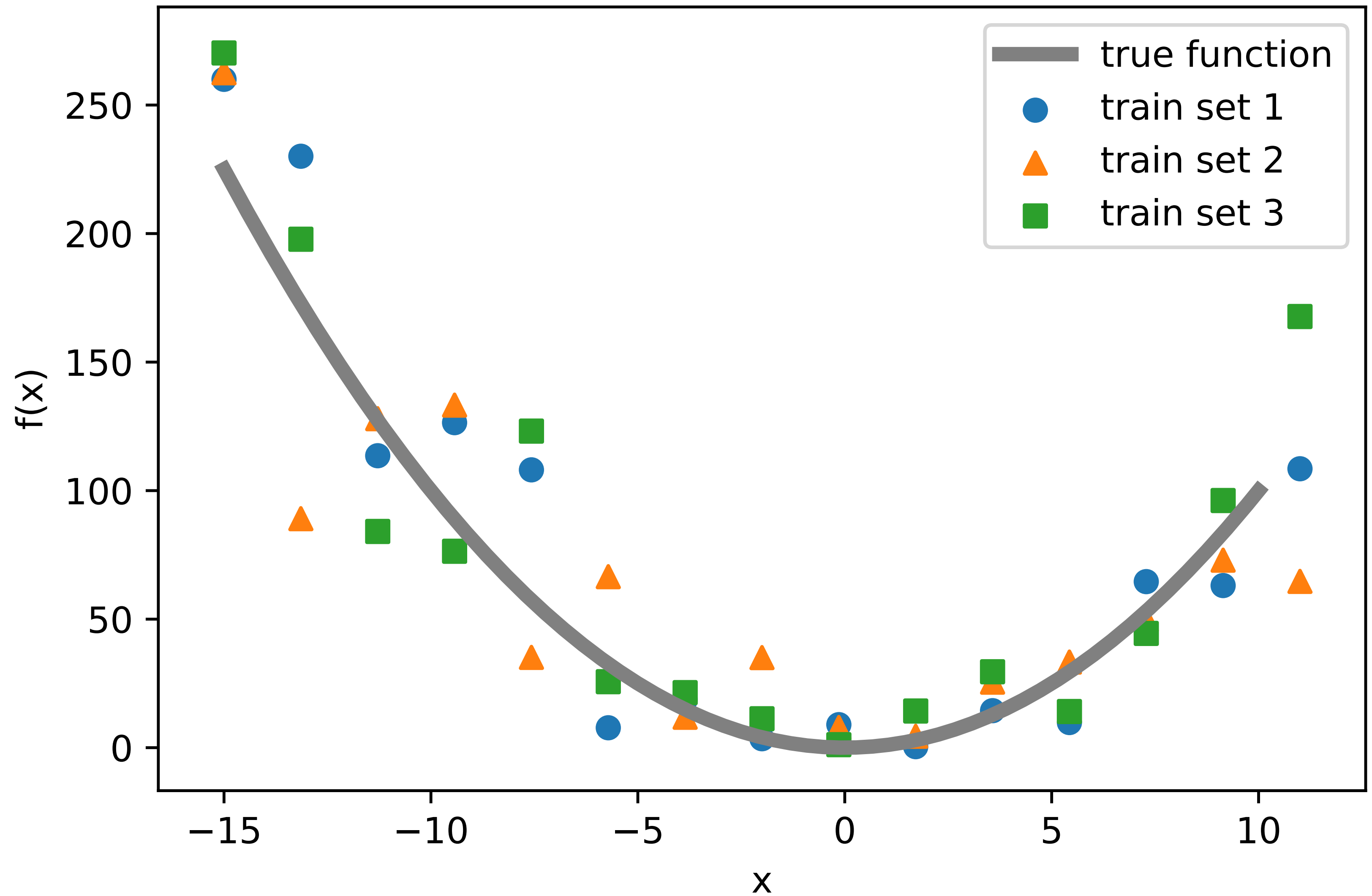


where f(x) is some true (target) function

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

here, suppose I fit an unpruned decision tree

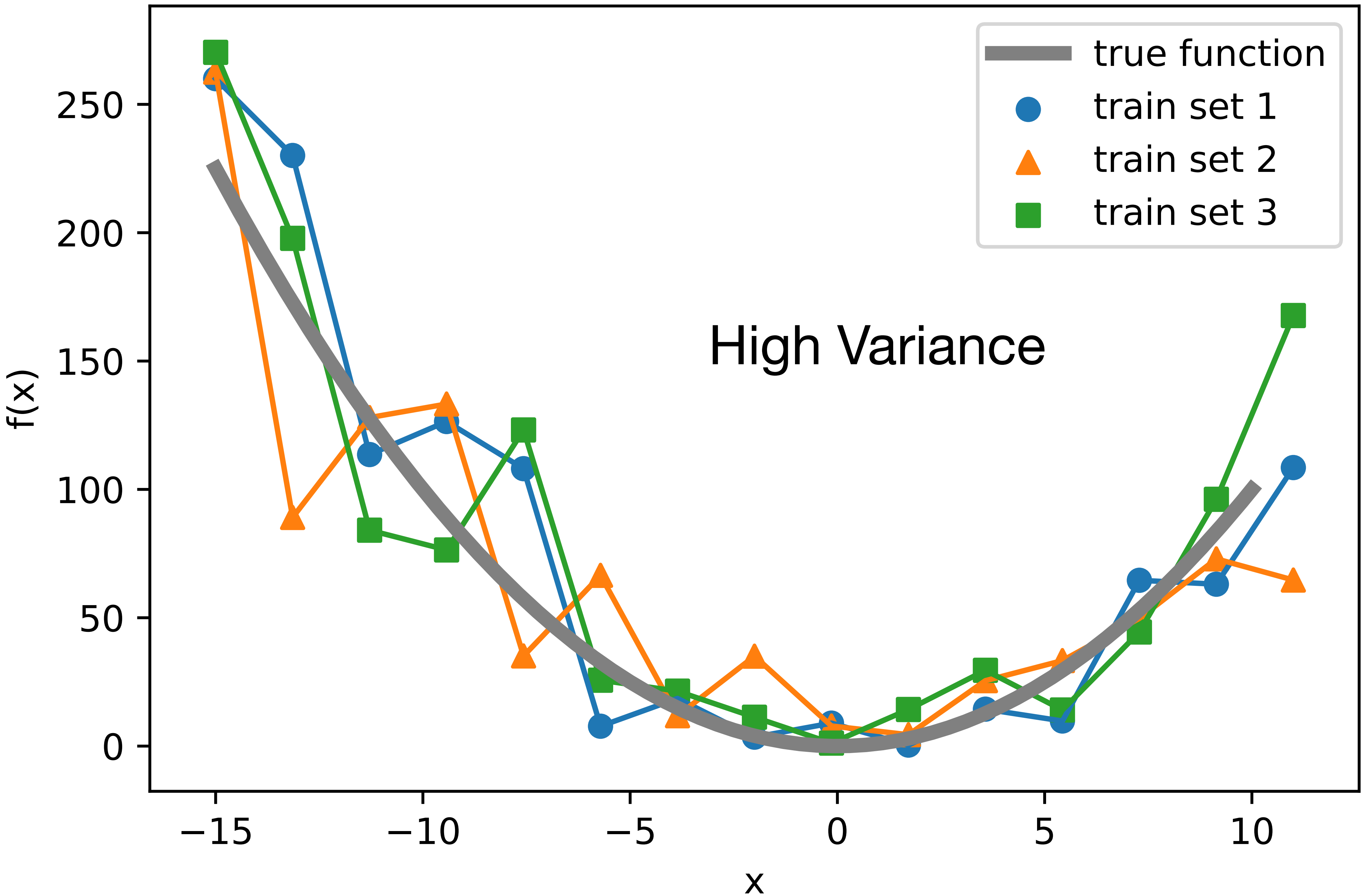
Bias and Variance Example



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

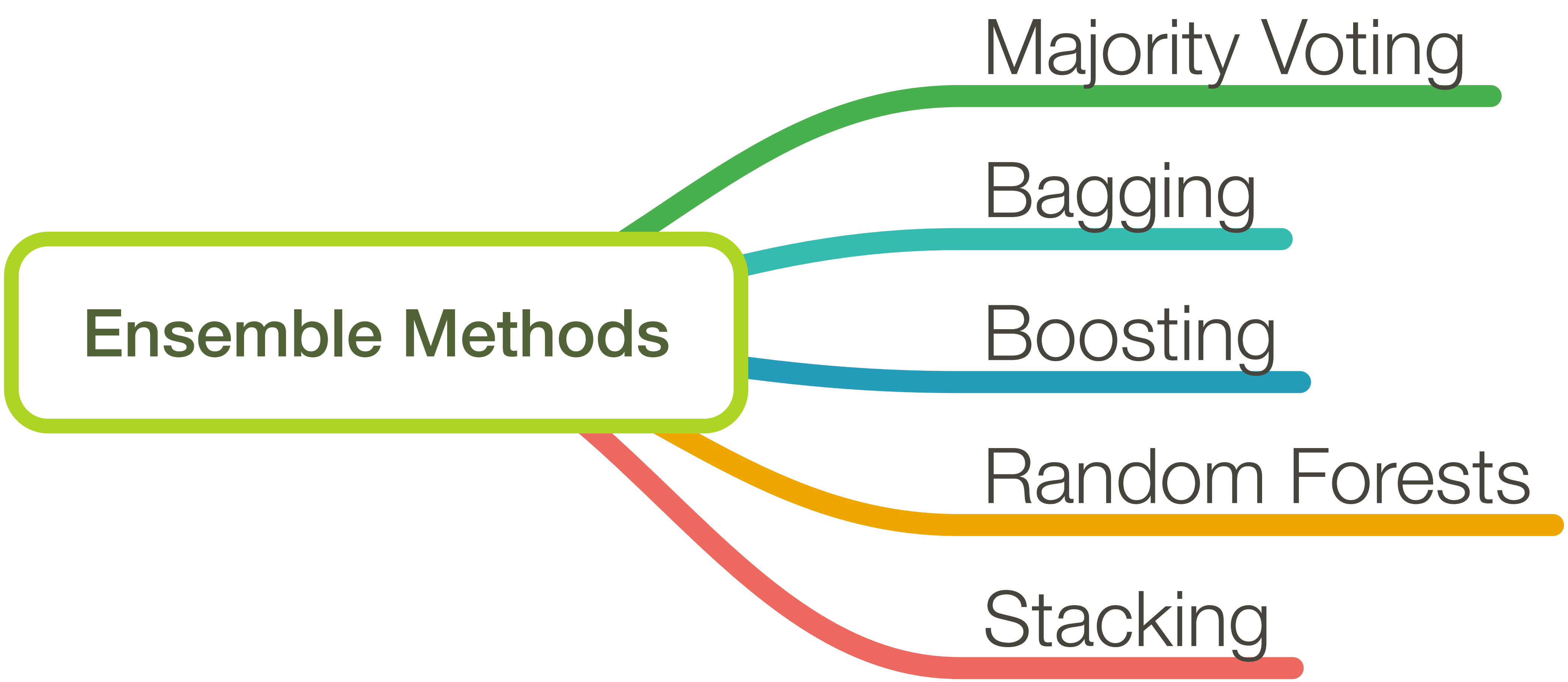
suppose we have multiple training sets

Bias and Variance Example



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

Overview



Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

Boosting

Adaptive Boosting

e.g., AdaBoost

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., LightGBM, XGBoost, scikit-learn's GradientBoostingClassifier

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

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

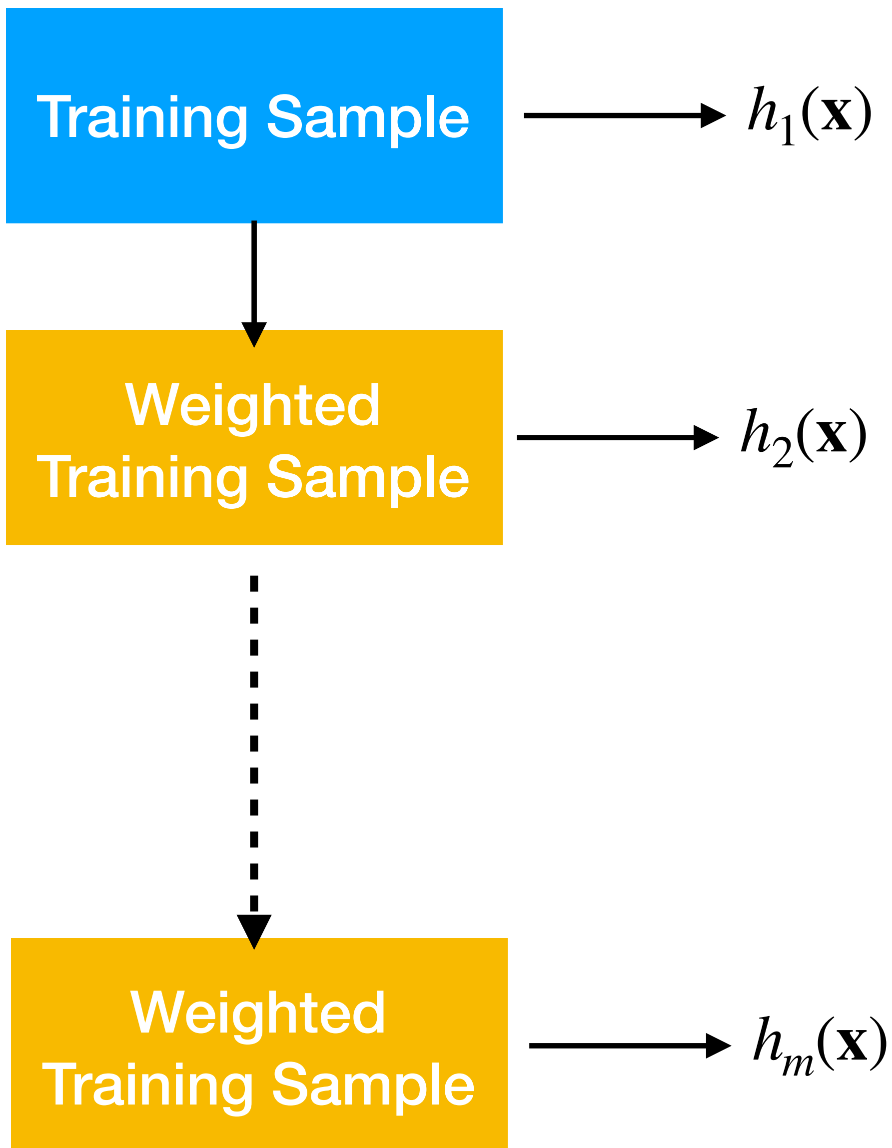
Gradient Boosting

e.g., LightGBM, XGBoost, scikit-learn's GradientBoostingClassifier

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



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

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

Assumes binary classification problem

Stopping Criteria

https://github.com/scikit-learn/scikit-learn/blob/1495f6924/sklearn/ensemble/weight_boosting.py#L569

```
        # Stop if classification is perfect
        if estimator_error <= 0:
            return sample_weight, 1., 0.

        n_classes = self.n_classes_

        # Stop if the error is at least as bad as random guessing
        if estimator_error >= 1. - (1. / n_classes):
            self.estimators_.pop(-1)
            if len(self.estimators_) == 0:
                raise ValueError('BaseClassifier in AdaBoostClassifier '
                                 'ensemble is worse than random, ensemble '
                                 'can not be fit.')
            return None, None, None
```

Combining the Classifiers (Weak Learners)

https://github.com/scikit-learn/scikit-learn/blob/1495f6924/sklearn/ensemble/weight_boosting.py#L617

```
        pred = self.decision_function(X)

        if self.n_classes_ == 2:
            return self.classes_.take(pred > 0, axis=0)

        return self.classes_.take(np.argmax(pred, axis=1), axis=0)
```

where def decision_function(self, X):

```
        if self.algorithm == 'SAMME.R':
            # The weights are all 1. for SAMME.R
            pred = sum(_samme_proba(estimator, n_classes, X)
                       for estimator in self.estimators_)
        else:  # self.algorithm == "SAMME"
            pred = sum((estimator.predict(X) == classes).T * w
                       for estimator, w in zip(self.estimators_,
                                               self.estimator_weights_))

        pred /= self.estimator_weights_.sum()
        if n_classes == 2:
            pred[:, 0] *= -1
            return pred.sum(axis=1)
        return pred
```

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

Decision Tree Stumps

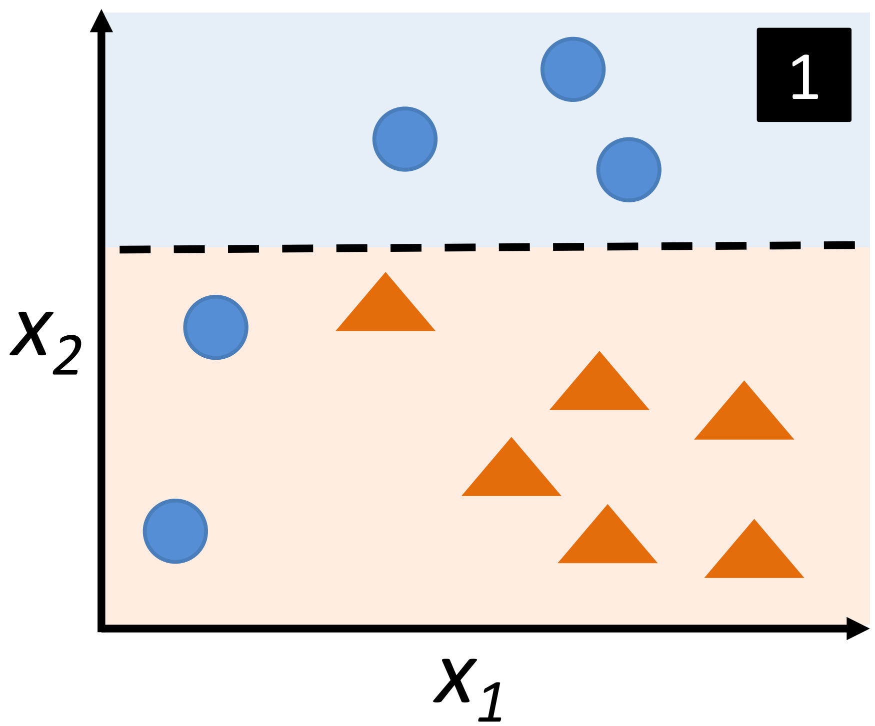
Weak classifier, here: decision tree stump for binary classification problem with labels -1, 1

$$h(\mathbf{x}) = s\Big(\mathbf{1}(x_k \geq t)\Big)$$

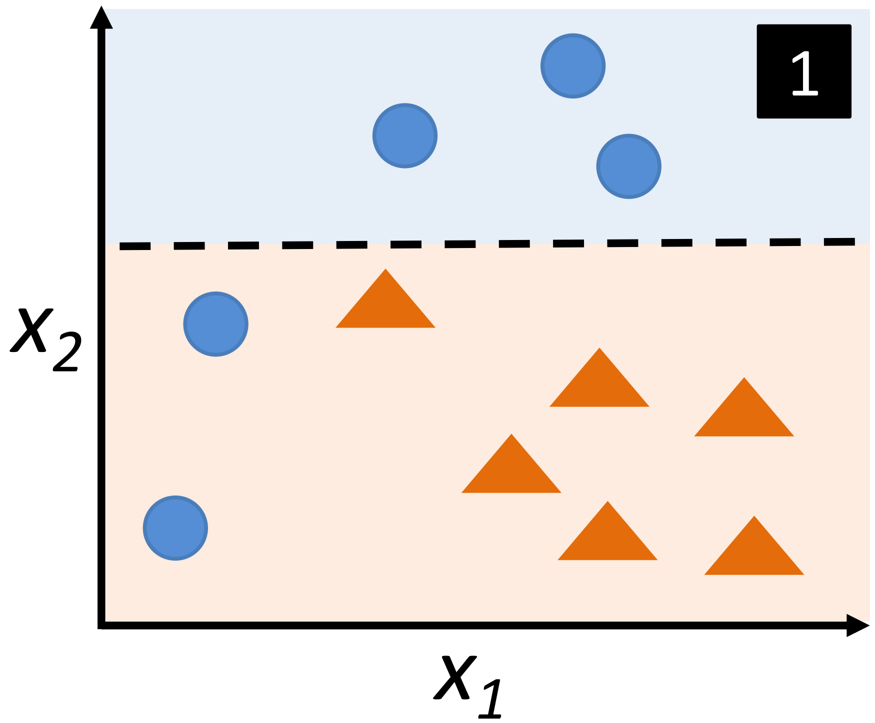
where

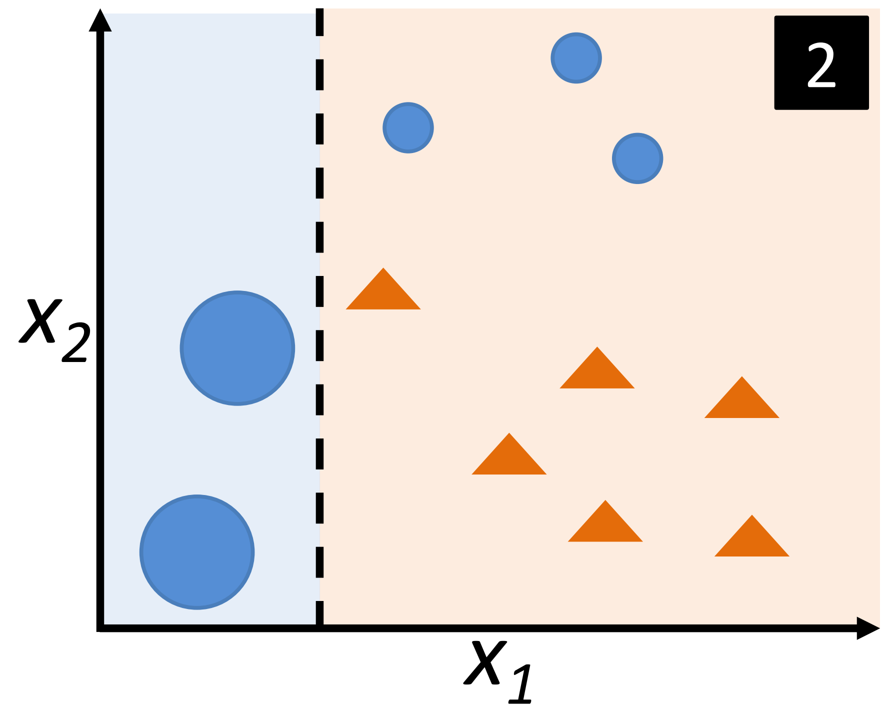
$s(x) \in \{-1, 1\}$

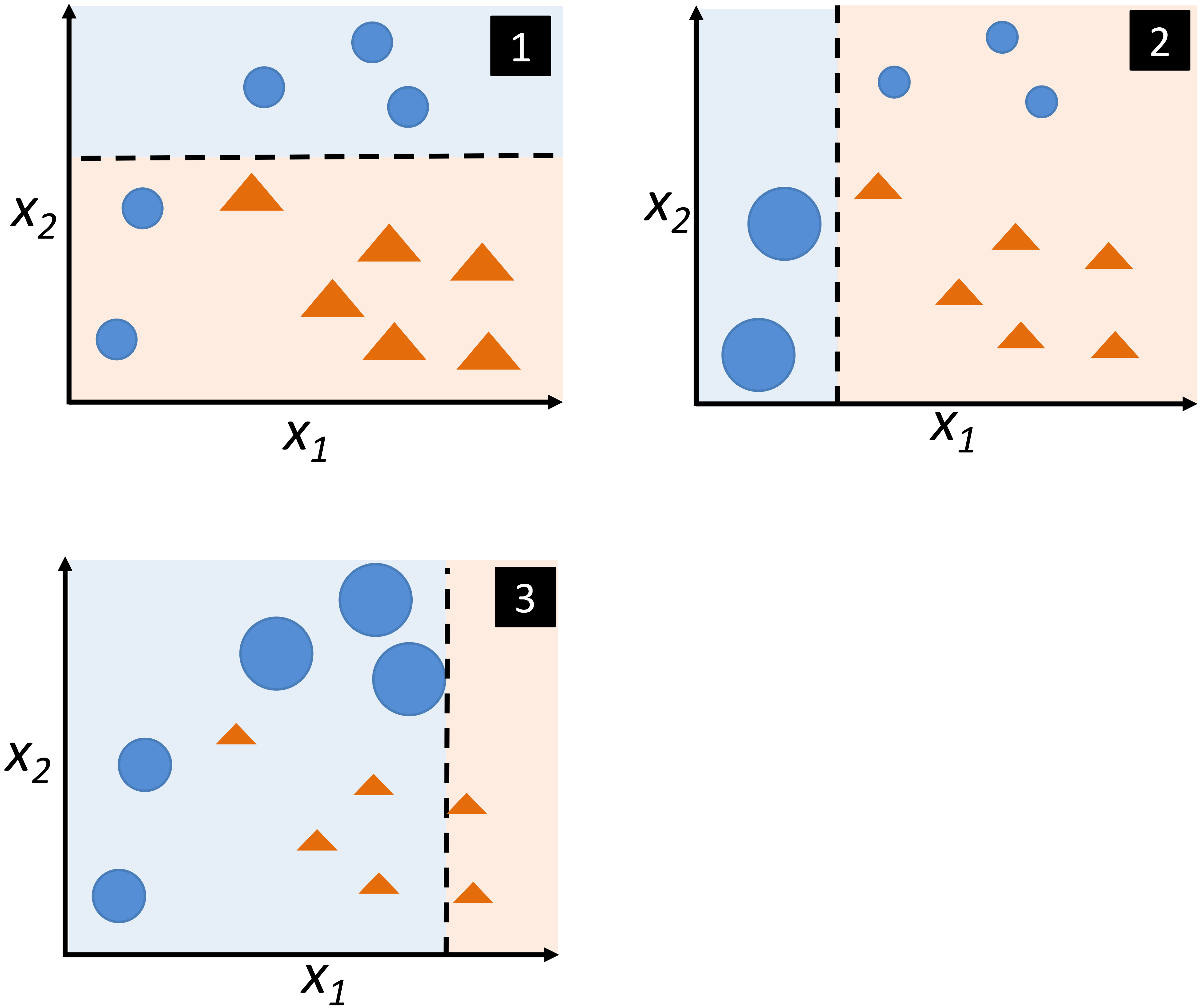
$k \in \{1, \ldots, K\}$ (K is the number of features)

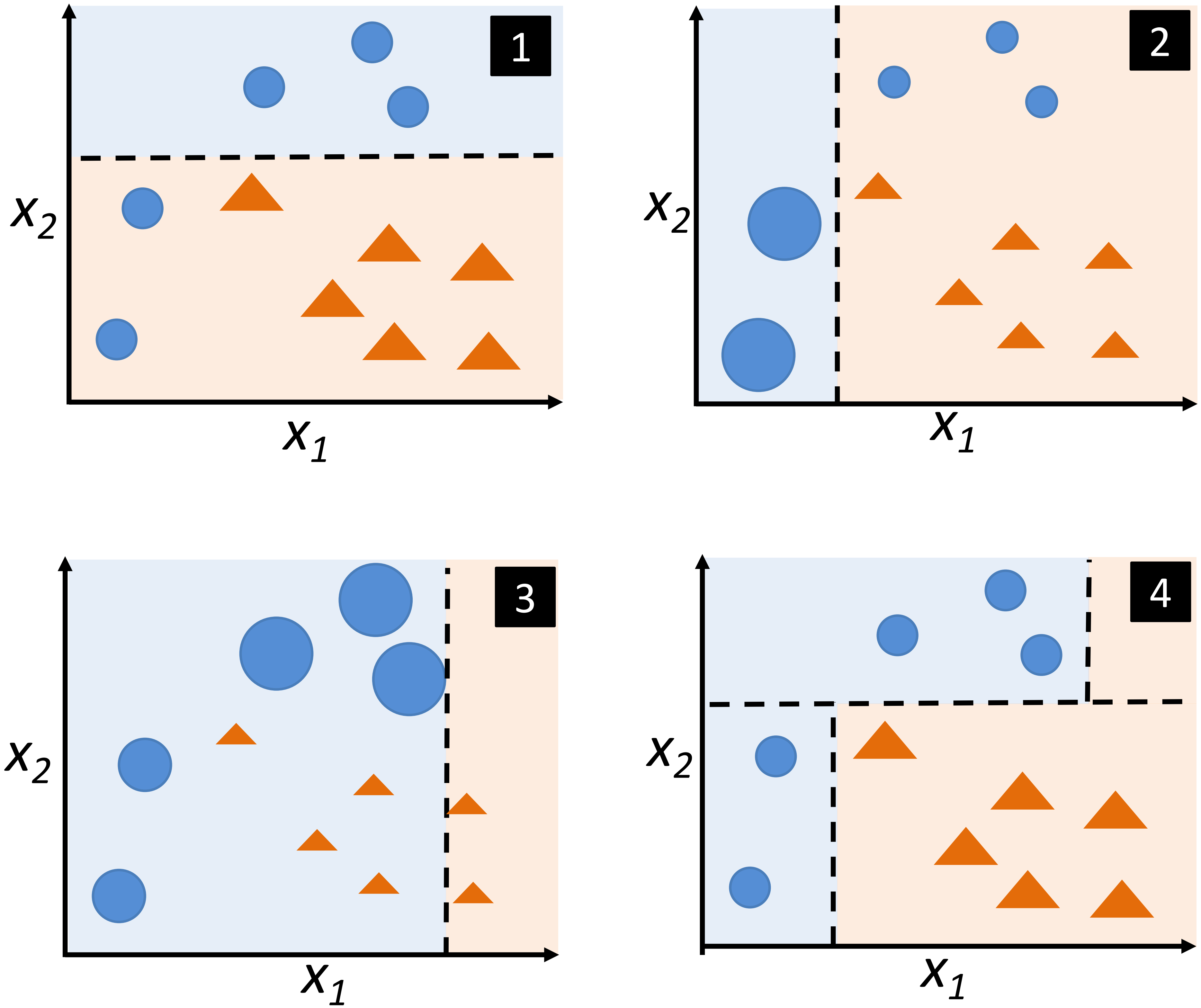


1
x_2
x_1


1
x_2
x_1


2
x_2
x_1


1
2
3
x_1
x_2



1
2
3
4
x_2
x_1

AdaBoost resources

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. Journal of Computer and System Sciences 55(1), 119–139 (1997)

https://pdf.sciencedirectassets.com/272574/1-s2.0-S0022000000X00384/1-s2.0-S002200009791504X/main.pdf

----

Explaining AdaBoost
Robert E. Schapire

http://rob.schapire.net/papers/explaining-adaboost.pdf

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

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

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

- **<u>Step 1:</u>** 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

- **Step 2:** 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

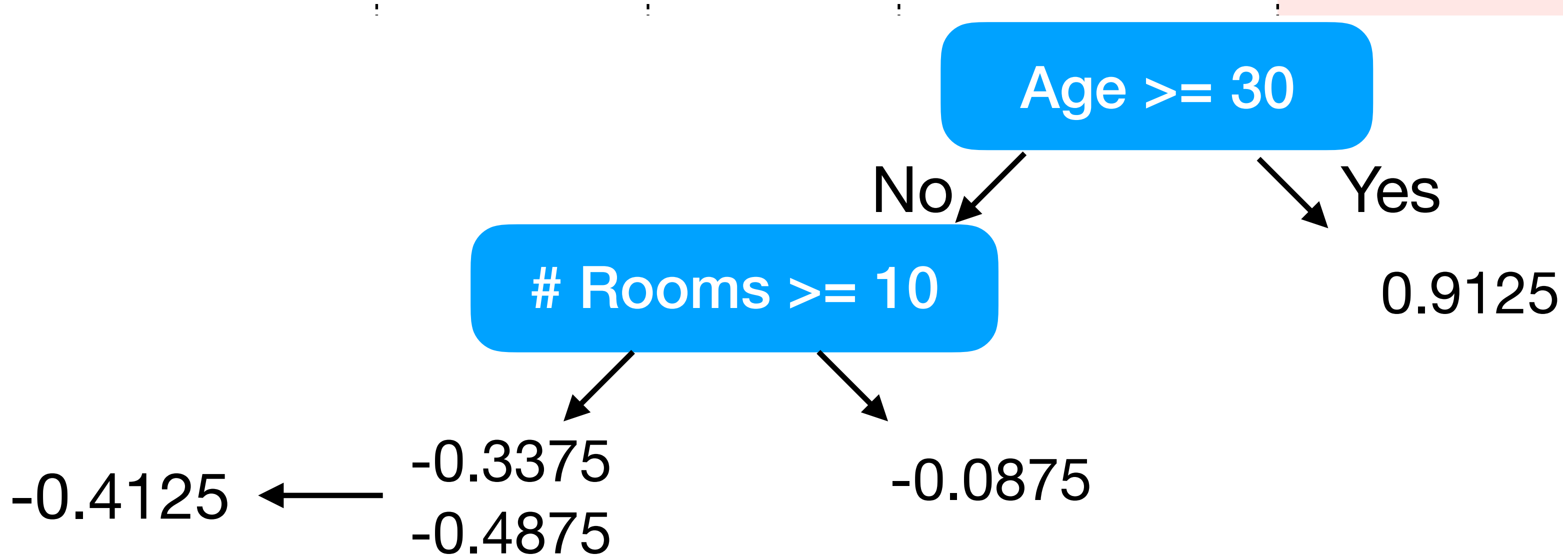
x1#	x2=City	x3=Age	y=Price	r1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

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

- **<u>Step 2:</u>** 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

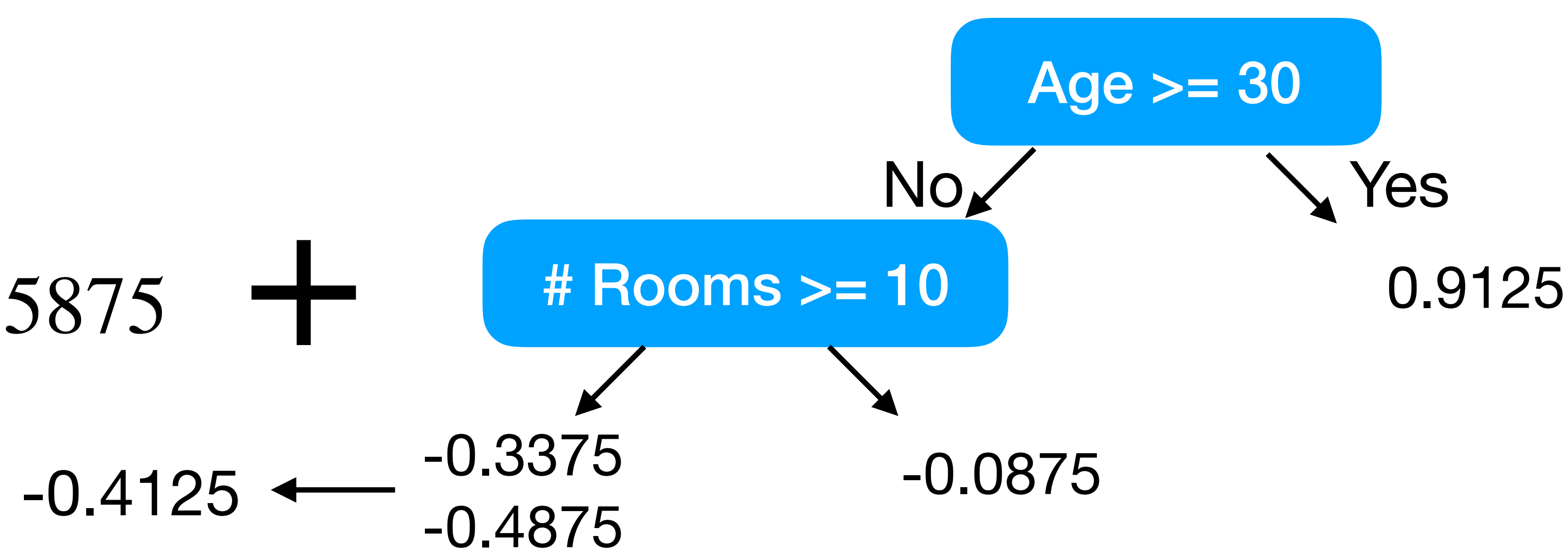


Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **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

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



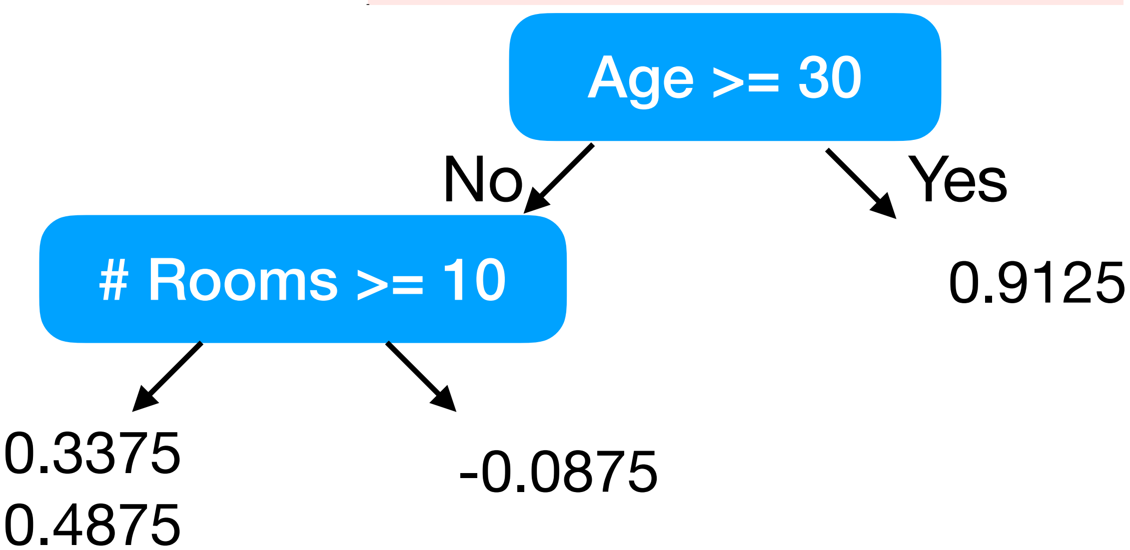
Gradient Boosting -- Conceptual Overview
--> A Regression-based Example

- **<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 $\leftarrow$ -0.3375, -0.4875

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

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

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

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

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 $j = 1,...,J_t$

...

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 $L\Big(y^{(i)}, h(\mathbf{x}^{(i)})\Big)$

E.g., Sum-squared error in regression

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

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

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

[neg. residual]

Gradient Boosting -- Algorithm Overview Discussion

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

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

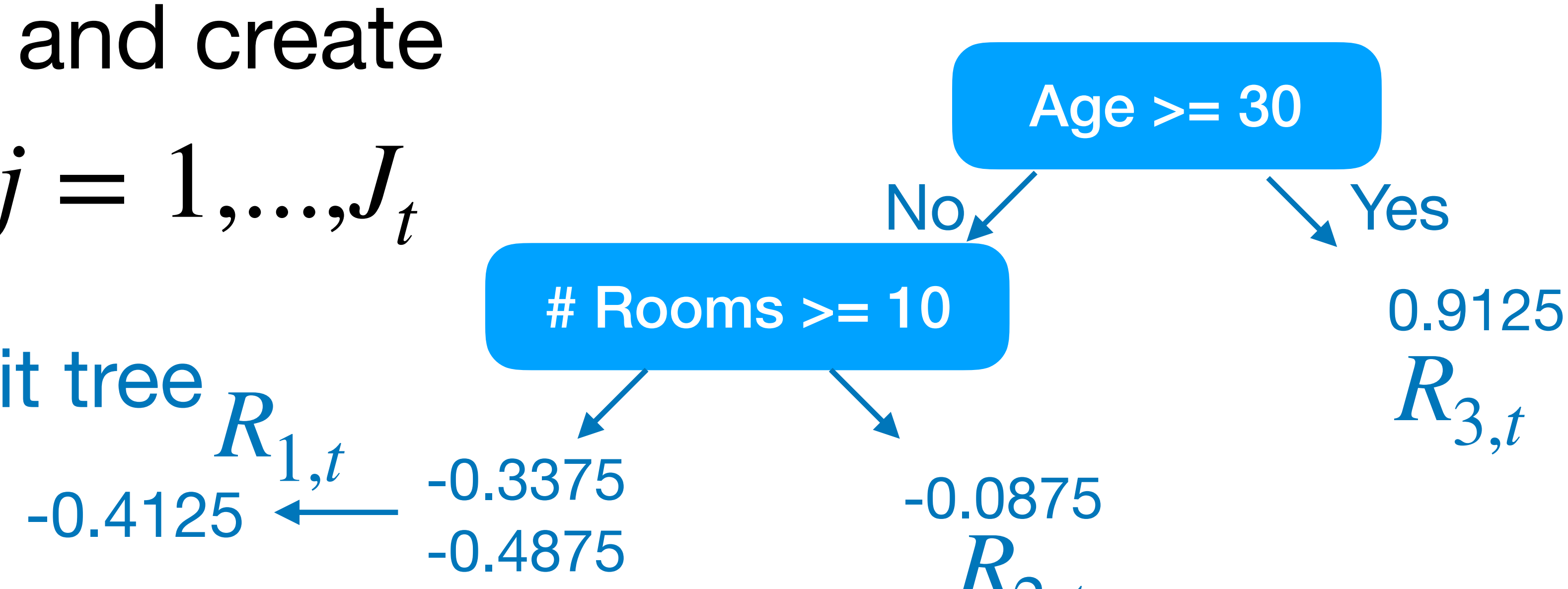
A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$ for $i = 1$ to n

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.,

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

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

$$\ldots$$

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

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

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.

XGBoost

learning system for tree boosting. The system is available as an open source package[2]. The impact of the system has been widely recognized in a number of machine learning and data mining challenges. Take the challenges hosted by the machine learning competition site Kaggle for example. Among the 29 challenge winning solutions [3] published at Kaggle's blog during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. For comparison, the second most popular method, deep neural nets, was used in 11 solutions. The success of the system was also witnessed in KDDCup 2015, where XGBoost was used by every winning team in the top-10. Moreover, the winning teams reported that ensemble

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.

XGBoost

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

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.

More GBM Implementations

LightGBM, Light Gradient Boosting Machine

From https://github.com/Microsoft/LightGBM:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Support of parallel and GPU learning
- Capable of handling large-scale data

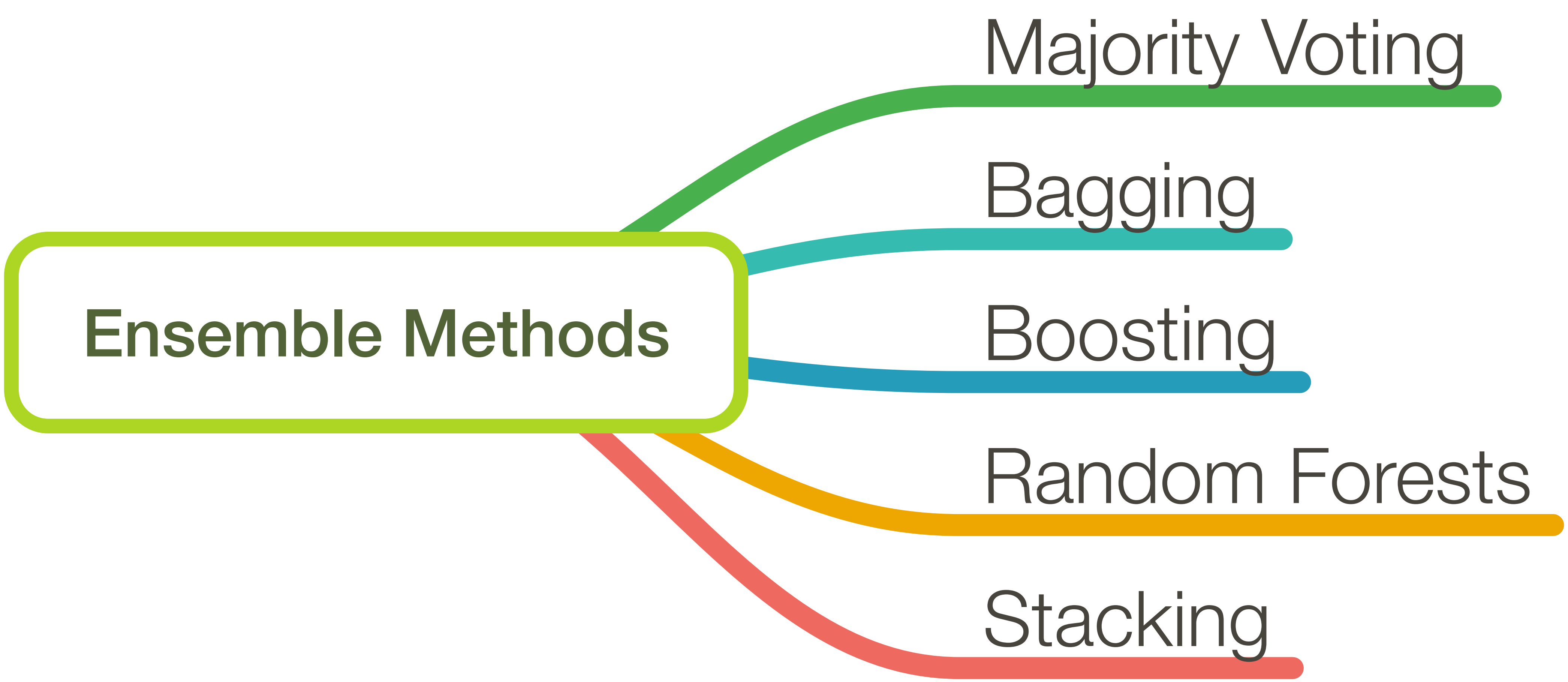
Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems* (pp. 3146-3154).

https://scikit-learn.org/stable/whats_new.html#version-0-21-0

sklearn.ensemble ¶

- **Major Feature** Add two new implementations of gradient boosting trees: `ensemble.HistGradientBoostingClassifier` and `ensemble.HistGradientBoostingRegressor` . The implementation of these estimators is inspired by LightGBM and can be orders of magnitude faster than `ensemble.GradientBoostingRegressor` and `ensemble.GradientBoostingClassifier` when the number of samples is larger than tens of thousands of samples. The API of these new estimators is slightly different, and some of the features from `ensemble.GradientBoostingClassifier` and `ensemble.GradientBoostingRegressor` are not yet supported.

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

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 … andom, at each node, a small group of input variab…

- Breiman, Leo. “Random Forests” Ma… 2.

num features $= \log_2 m + 1$

where *m* is the number of input features

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

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

$\bar{\rho}$: 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:

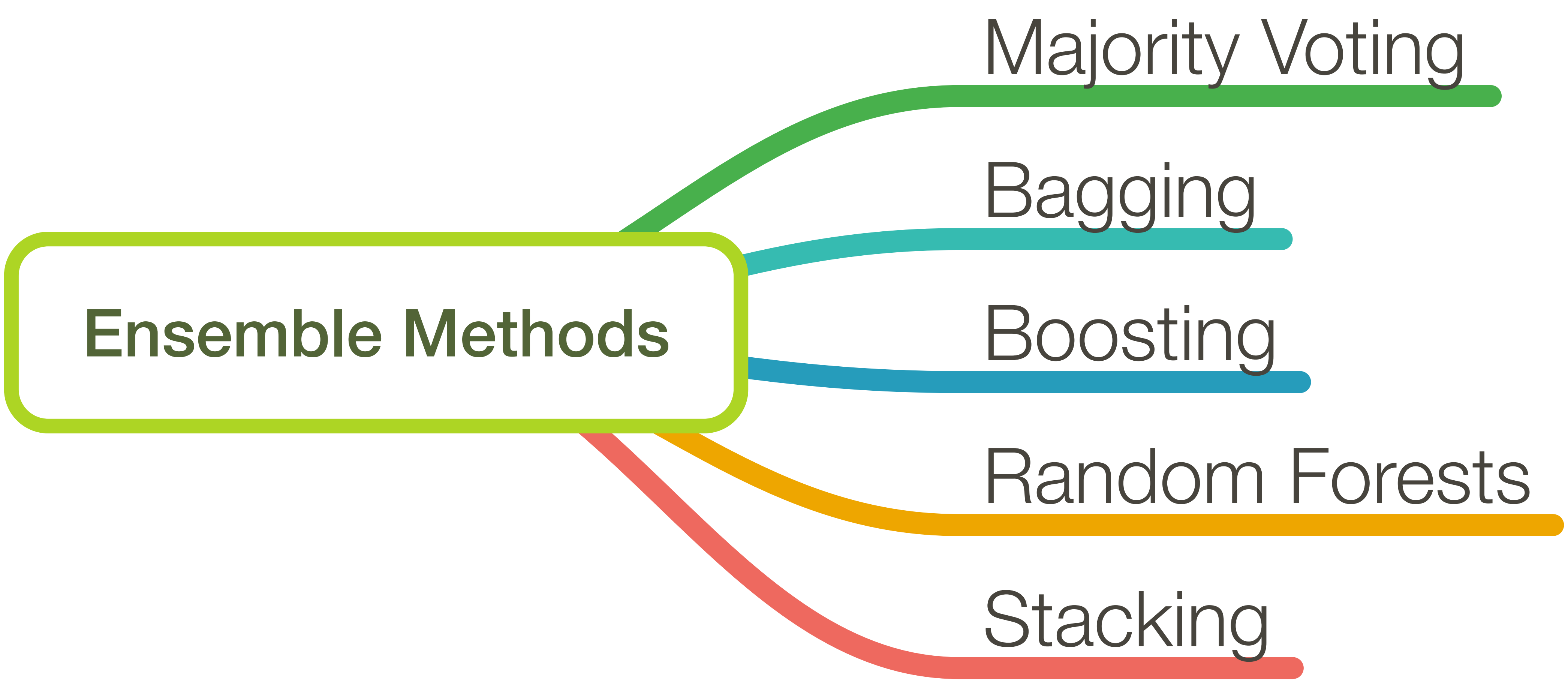
1) ______________

2) ______________

ExtraTrees algorithm adds one more random component

3) ______________

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

Stacking

Stacking Algorithm

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

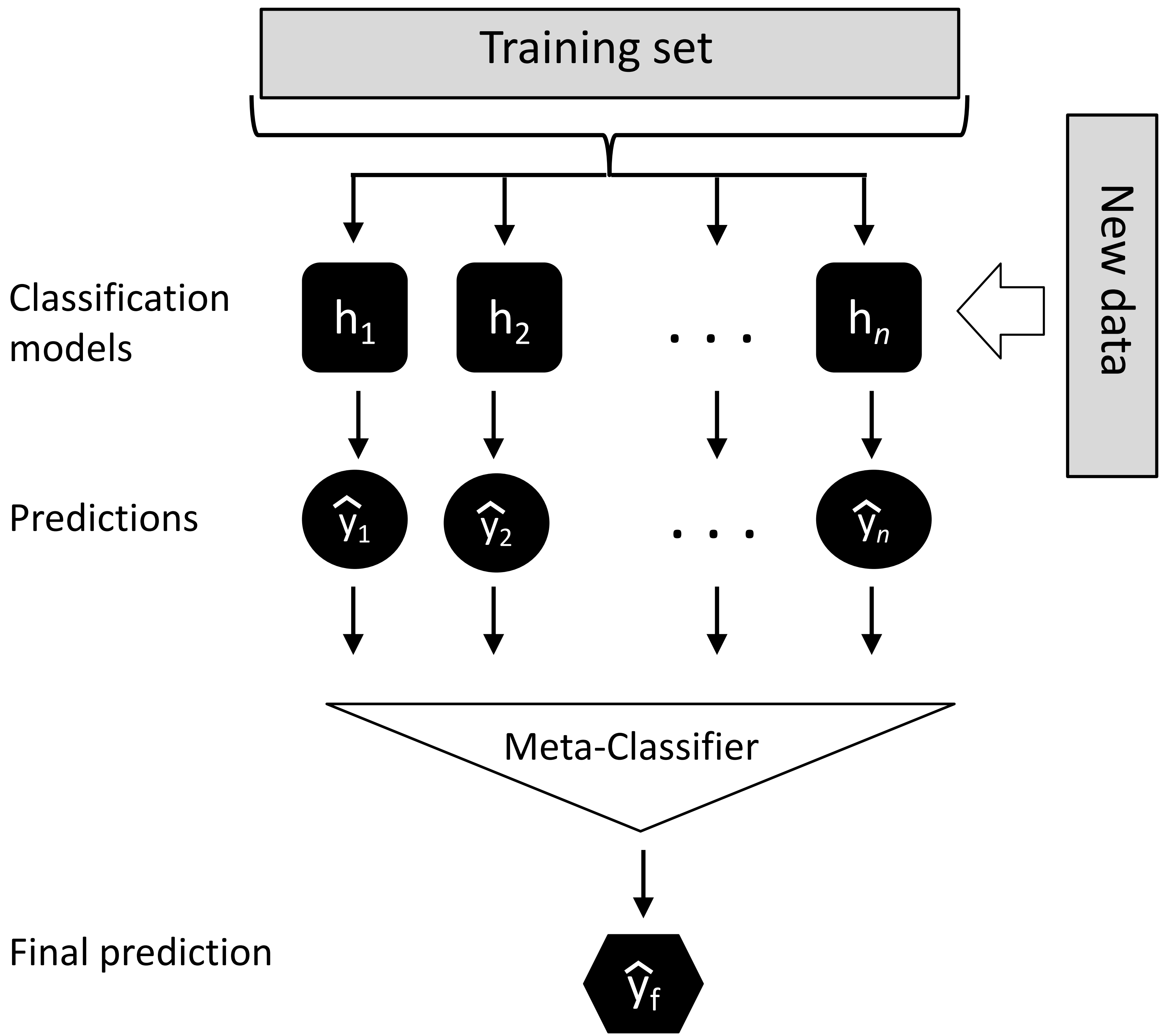
Algorithm 19.7 Stacking

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

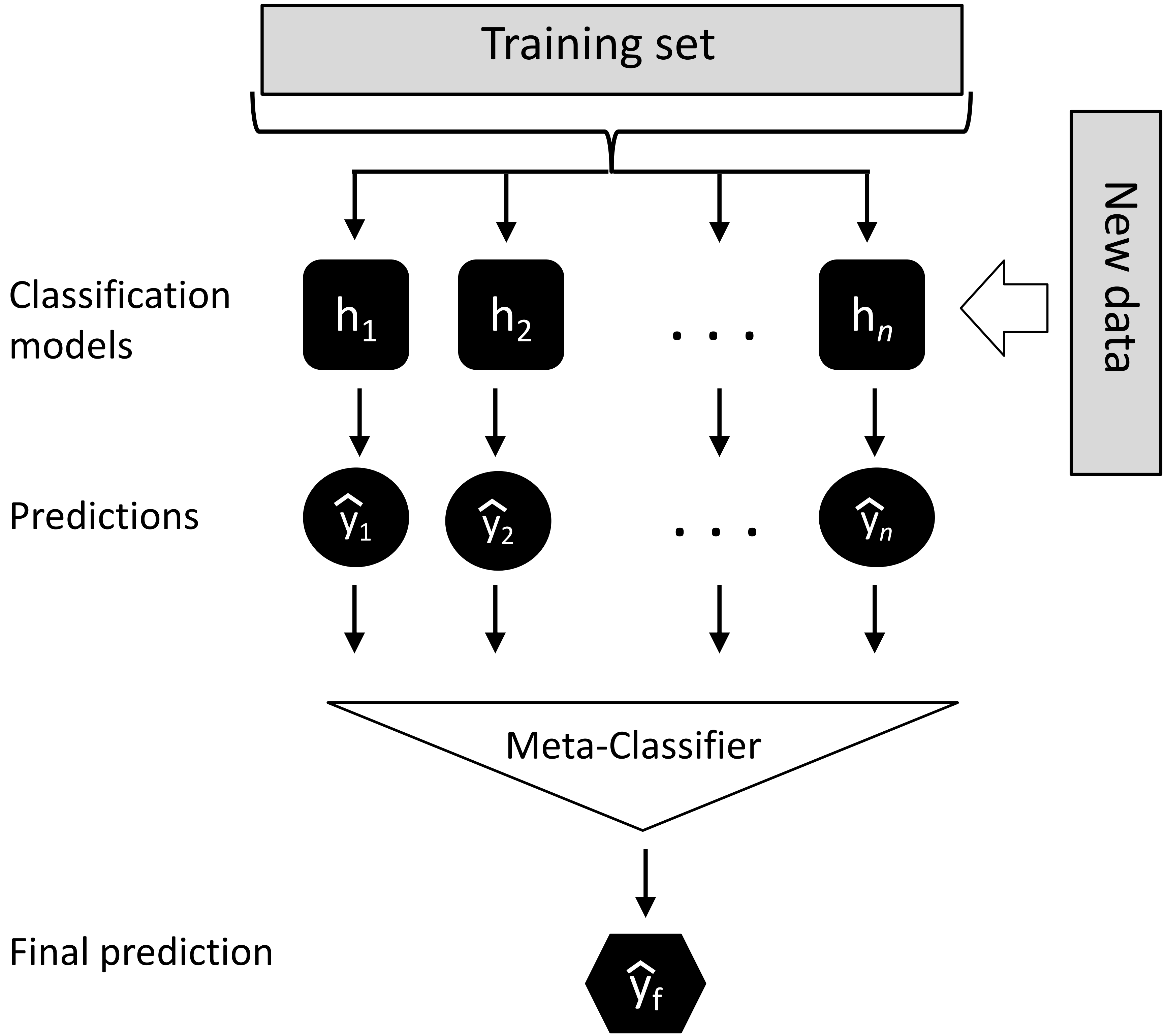
```
Step 1: Learn first-level classifiers
for t ← 1 to T do
    Learn a base classifier h_t based on D
end for
Step 2: Construct new data sets from D
for i ← 1 to m do
    Construct a new data set that contains {x'_i, y_i}, where x'_i = {h_1(x_i), h_2(x_i), ..., h_T(x_i)}
end for
Step 3: Learn a second-level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

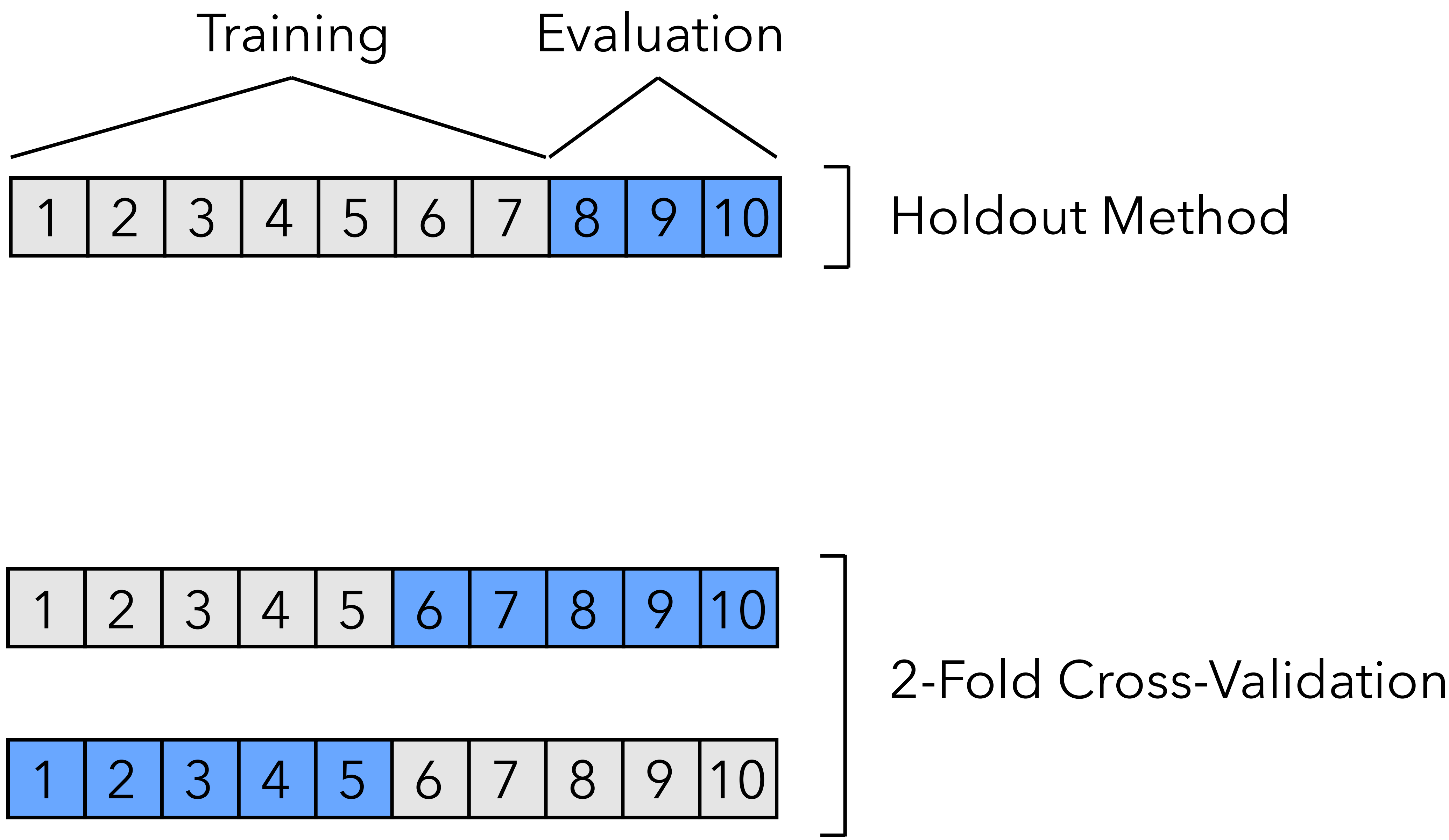
Stacking Algorithm



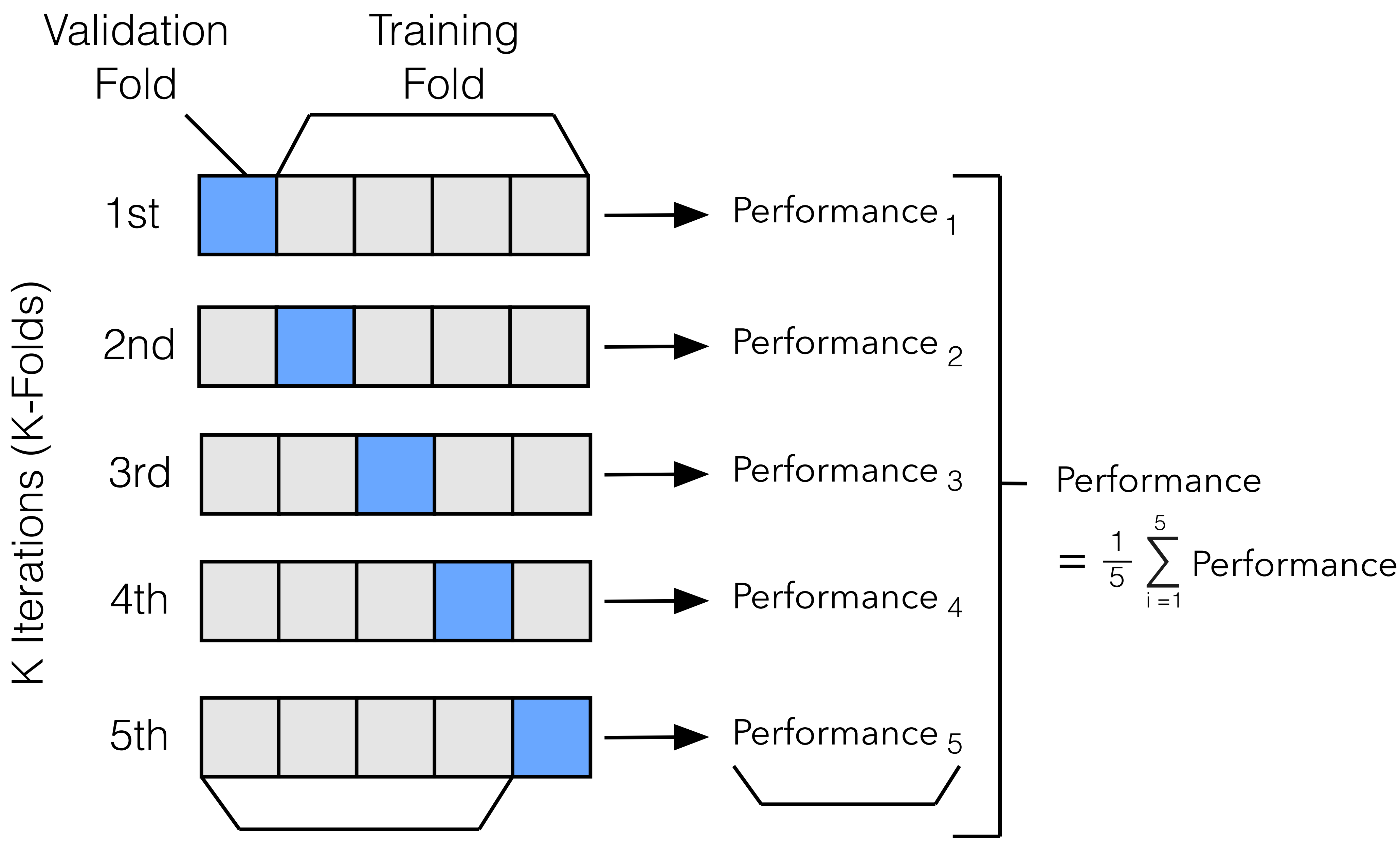
What is the problem with this stacking procedure?

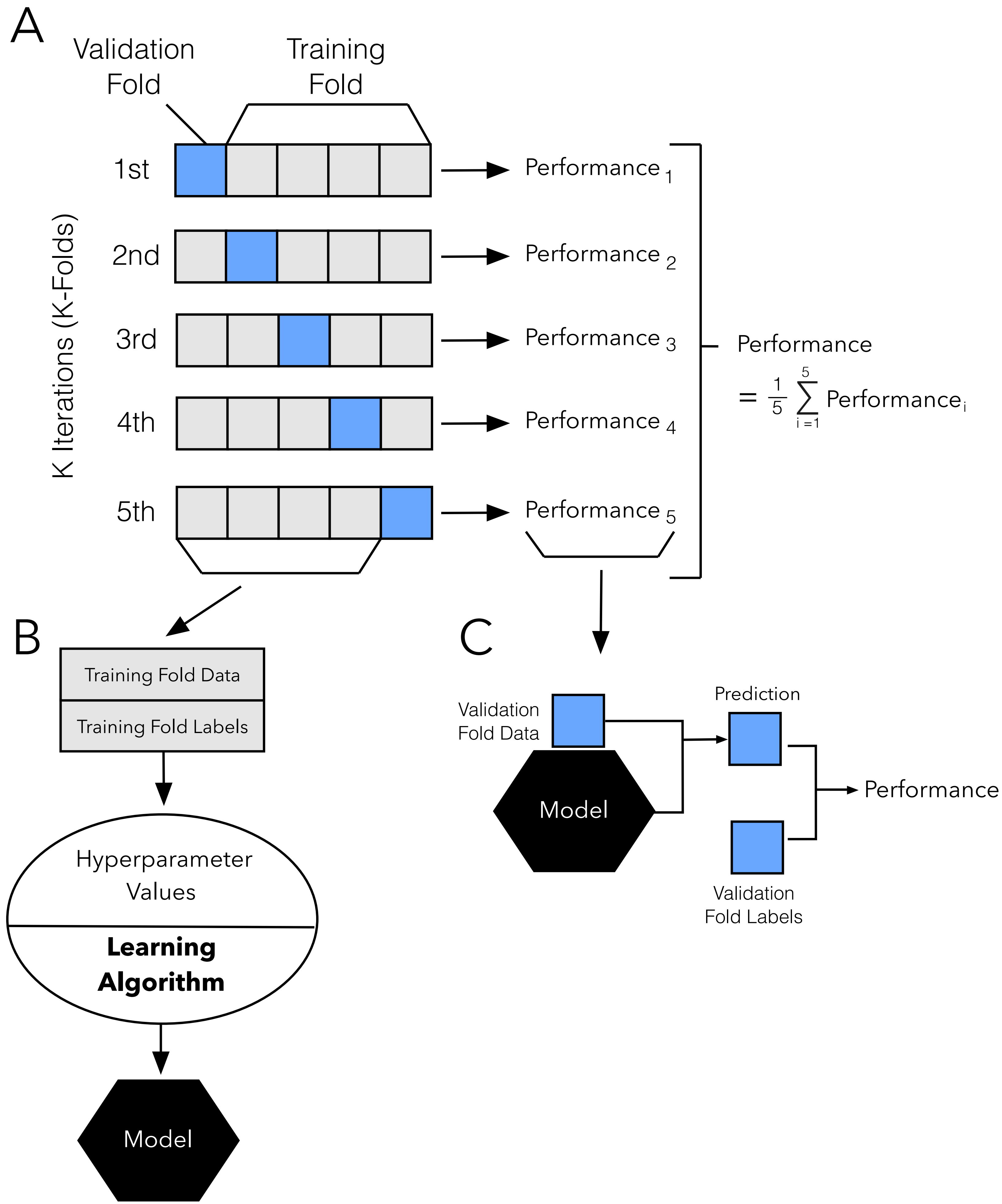


Cross-Validation



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.

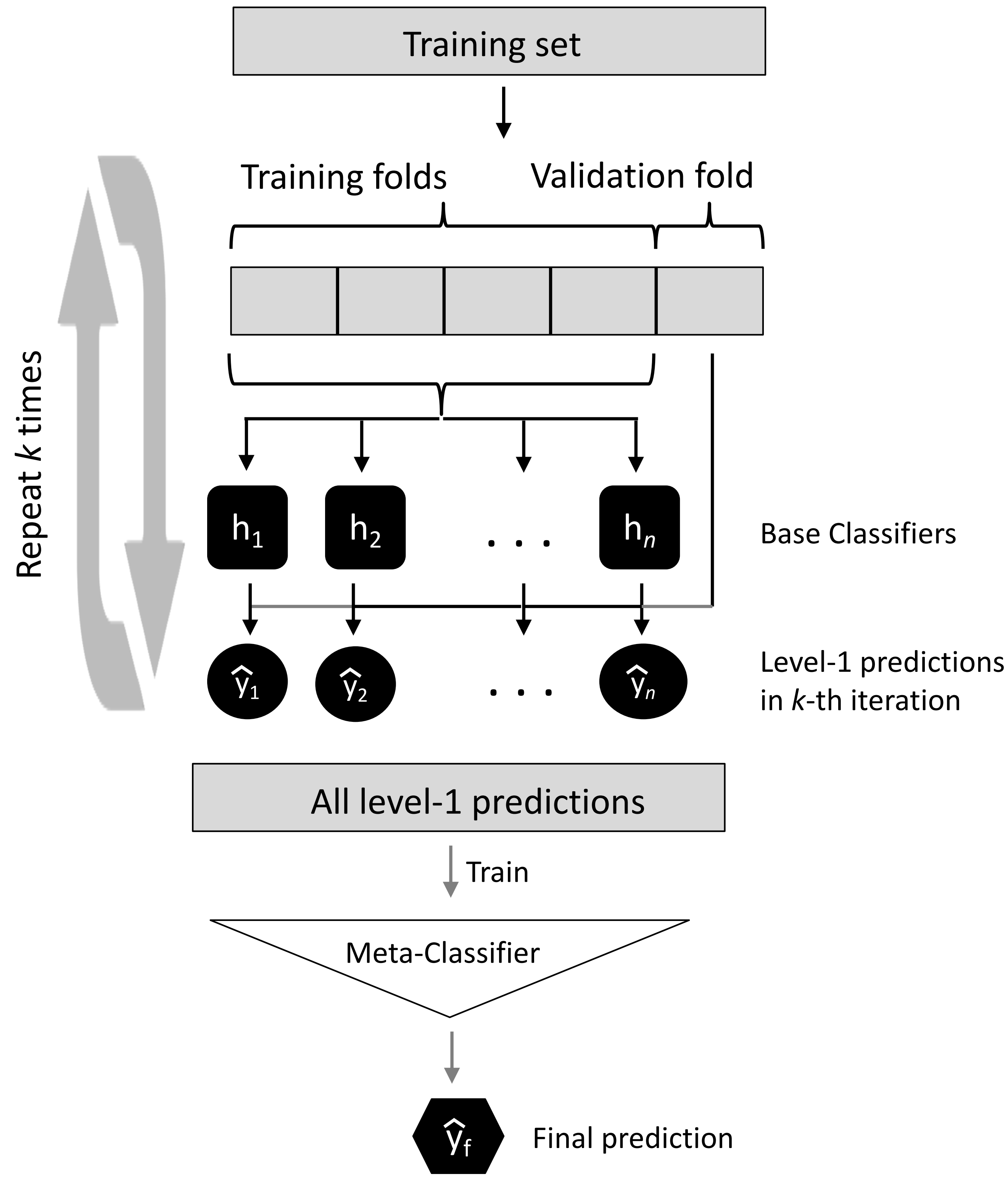
Algorithm 19.8 Stacking with K-fold Cross Validation

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

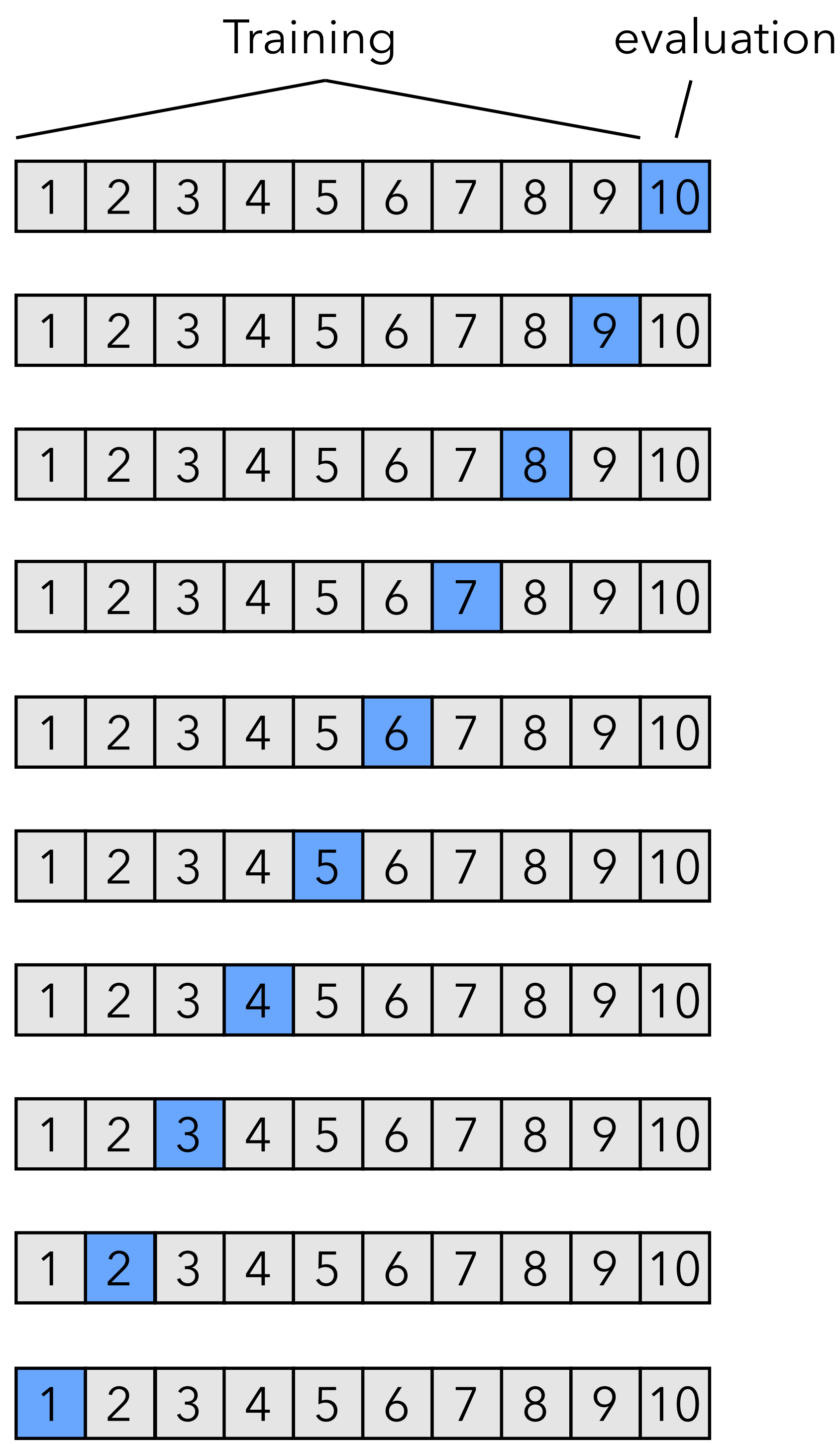
Step 1: Adopt cross validation approach in preparing a training set for second-level classifier
Randomly split $\mathcal{D}$ into K equal-size subsets: $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K\}$
for $k \leftarrow 1$ to K **do**
    Step 1.1: Learn first-level classifiers
    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



Leave-One-Out CV



Demos

http://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

http://scikit-learn.org/stable/auto_examples/ensemble/plot_bias_variance.html#sphx-glr-auto-examples-ensemble-plot-bias-variance-py

http://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_hastie_10_2.html#sphx-glr-auto-examples-ensemble-plot-adaboost-hastie-10-2-py

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

http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

http://rasbt.github.io/mlxtend/user_guide/classifier/StackingCVClassifier/

Reading Assignments

Python Machine Learning, 2nd Ed., Ch07