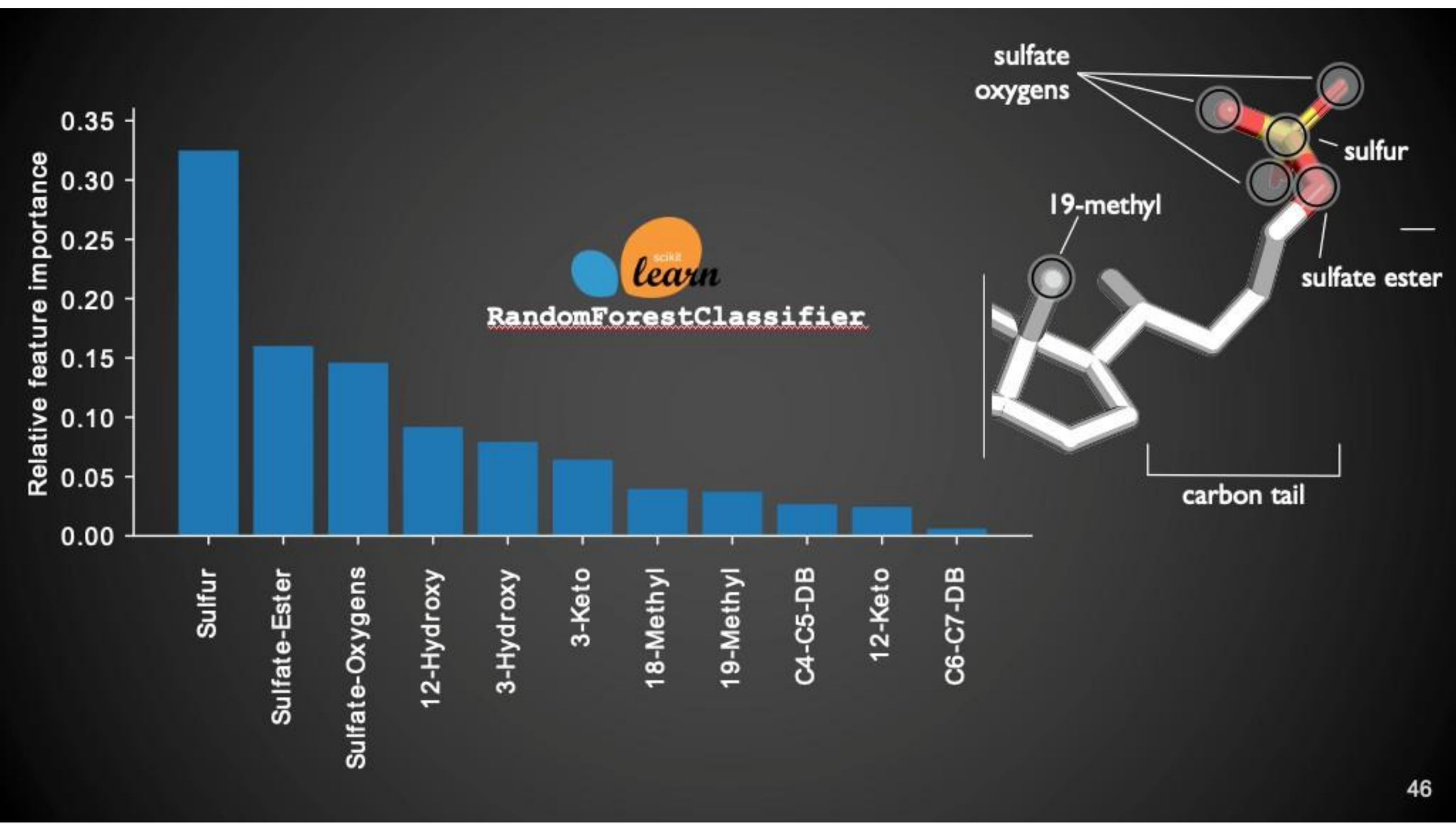
Ensemble Methods

1. **Ensemble Methods -- Intro and Overview**
2. Majority Voting
3. Bagging
4. Boosting
5. Gradient Boosting
6. Random Forests
7. Stacking



Sebastian Raschka, Leslie A. Kuhn, Anne M. Scott, and Weiming Li (2018) *Computational Drug Discovery and Design: Automated Inference of Chemical Group Discriminants of Biological Activity from Virtual Screening Data.* Springer. ISBN: 978-1-4939-7755-0

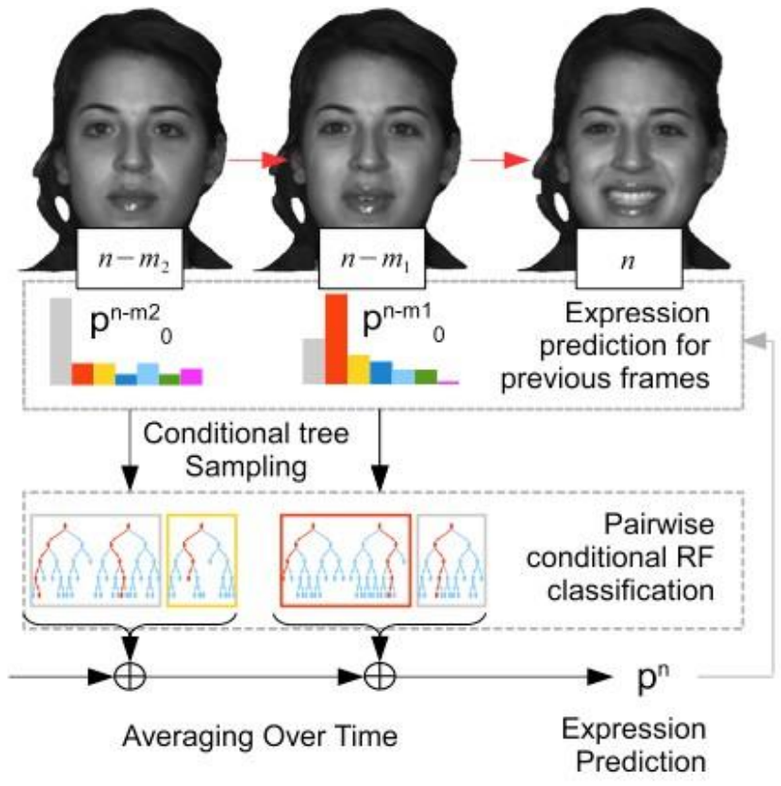
Pairwise Conditional Random Forests for Facial Expression Recognition

Arnaud Dapogny[1]
arnaud.dapogny@isir.upmc.fr

Kevin Bailly[1]
kevin.bailly@isir.upmc.fr

Séverine Dubuisson[1]
severine.dubuisson@isir.upmc.fr

[1] Sorbonne Universités, UPMC Univ Paris 06 CNRS, UMR 7222, F-75005, Paris, France



http://www.isir.upmc.fr/files/2015ACTI3549.pdf

Figure 1. Flowchart of our PCRF FER method. When evaluating a video frame indexed by n, dedicated pairwise trees are drawn conditionally to expression probability distributions from previous frames $n - m_1, n - m_2, ...$ The subsequent predictions are averaged over time to output an expression probability distribution for the current frame. This prediction can be used as a tree sampling distribution for subsequent frame classification.

"The purpose of this work is to develop a learning-based method to generate patient-specific synthetic CT (sCT) from a routine anatomical MRI for use in MRI-only radiotherapy treatment planning. An auto-context model with patch-based anatomical features was integrated into a classification random forest to generate and improve semantic information. The semantic information along with anatomical features was then used to train a series of regression random forests based on the auto-context model."

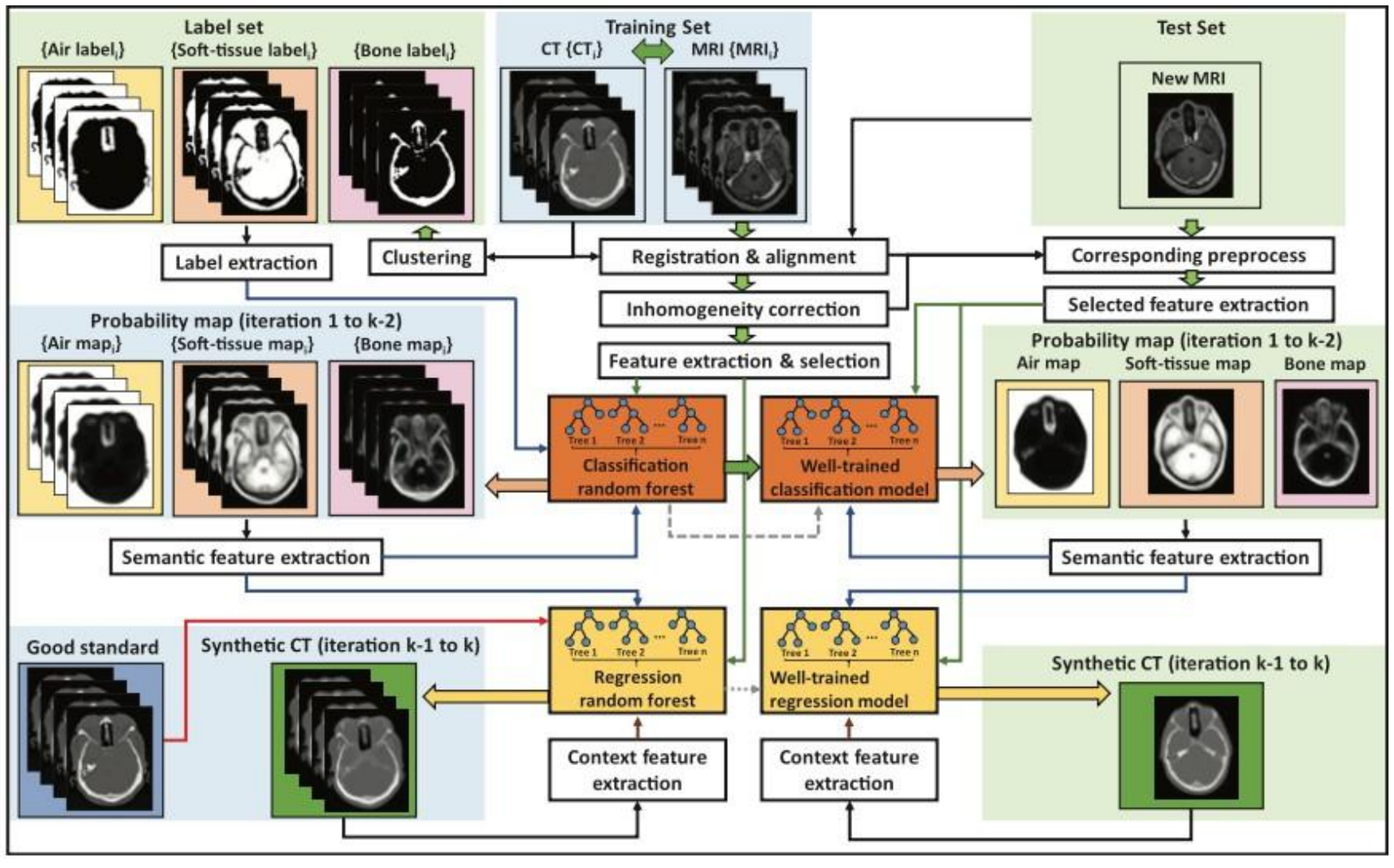


Figure 1. Schematic flow chart of the proposed algorithm for MRI-based sCT generation. The left part of this figure shows the training stage of our proposed method, which consists of classification random forest training, semantic feature extraction, context feature extraction, and regression random forest training. The right part of this figure shows the synthesizing stage, where a new MR image follows a similar sequence to the training stage to generate a sCT image.

Lei, Yang, Joseph Harms, Tonghe Wang, Sibo Tian, Jun Zhou, Hui-Kuo Shu, Jim Zhong et al. "MRI-based synthetic CT generation using semantic random forest with iterative refinement." *Physics in Medicine & Biology* 64, no. 8 (2019): 085001.

Motivations

Most widely used non-DL machine learning models

"More recently, gradient boosting machines (GBMs) have become a Swiss army knife in many a Kaggler’s toolbelt" [1]

[1] Sebastian Raschka, Joshua Patterson, and Corey Nolet (2020)
Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence
Information 2020, 11, 4

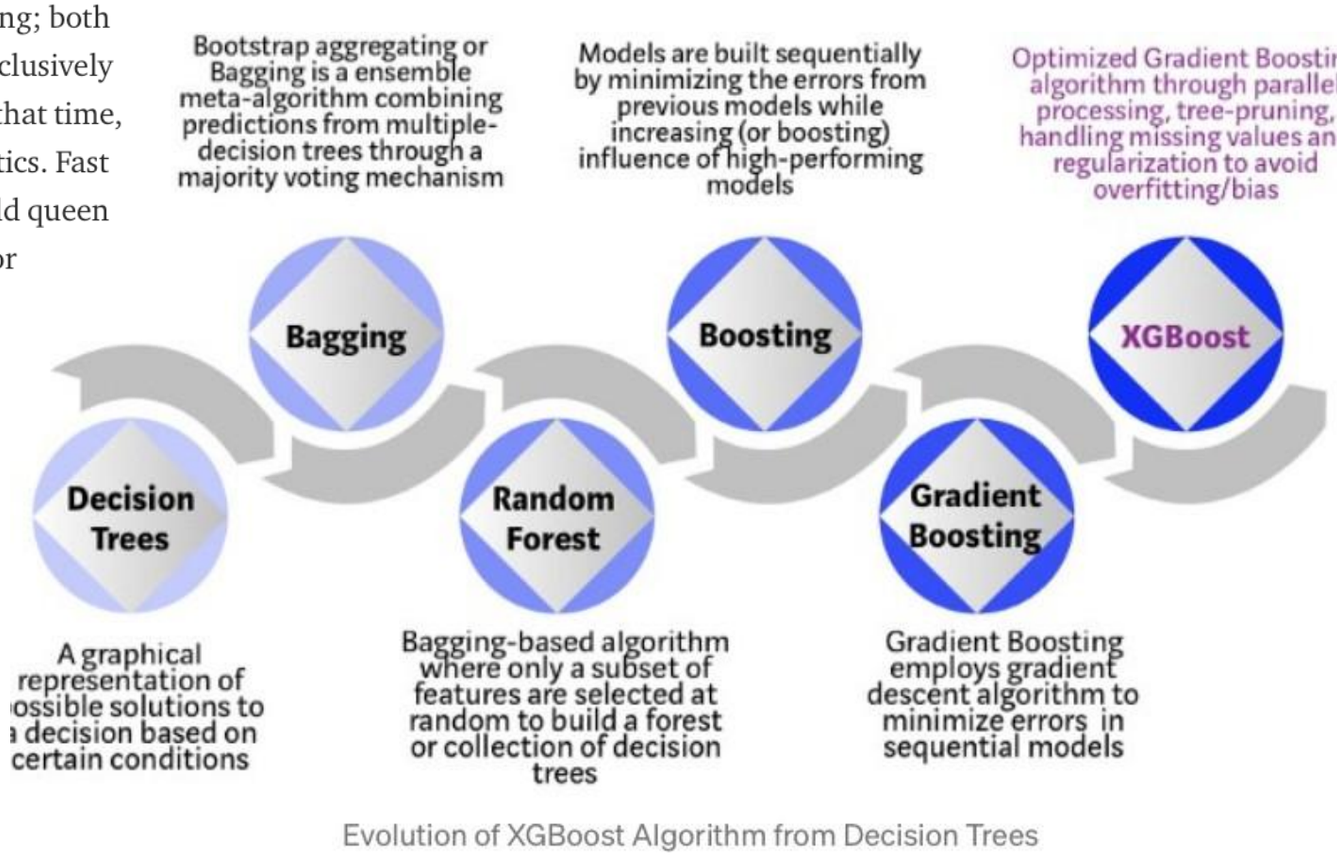
XGBoost Algorithm: Long May She Reign!

The new queen of Machine Learning algorithms taking over the world...

Vishal Morde Apr 7, 2019 · 7 min read

(This article was co-authored with Venkat Anurag Setty)

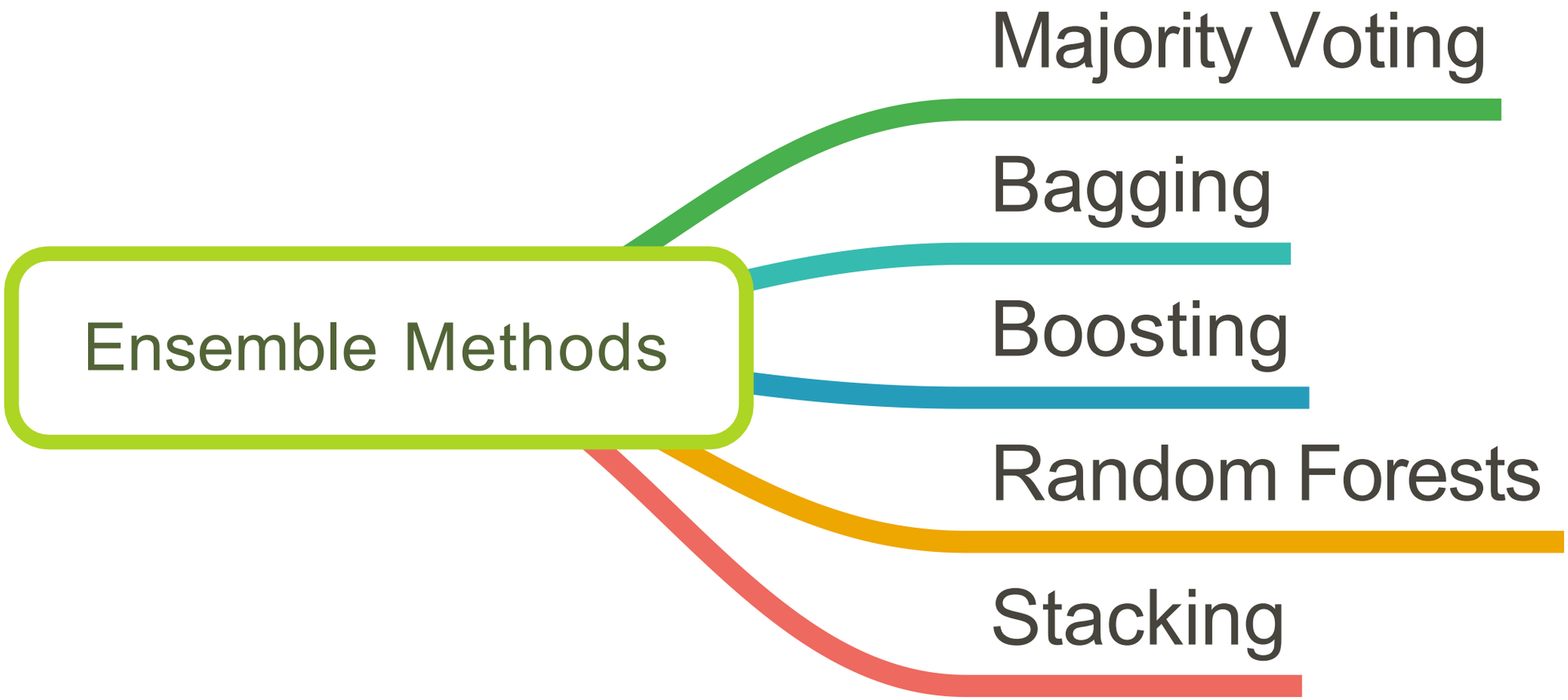
I remember thinking myself, “I got this!”. I knew regression modeling; both linear and logistic regression. My boss was right. In my tenure, I exclusively built regression-based statistical models. I wasn’t alone. In fact, at that time, regression modeling was the undisputed queen of predictive analytics. Fast forward fifteen years, the era of regression modeling is over. The old queen has passed. Long live the new queen with a funky name; XGBoost or Extreme Gradient Boosting!



Evolution of XGBoost Algorithm from Decision Trees

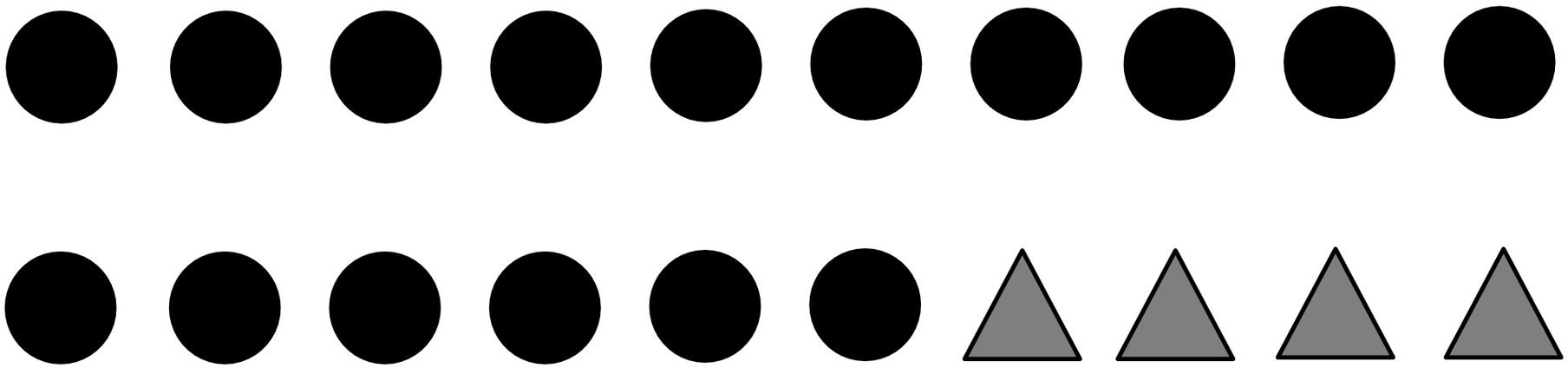
https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

Lecture Overview



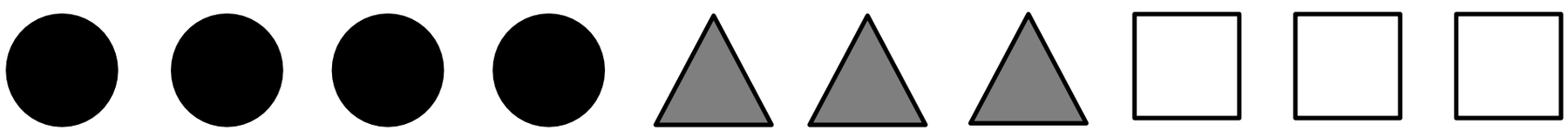
1. Ensemble Methods -- Intro and Overview
2. **Majority Voting**
3. Bagging
4. Boosting
5. Gradient Boosting
6. Random Forests
7. Stacking

Majority Voting



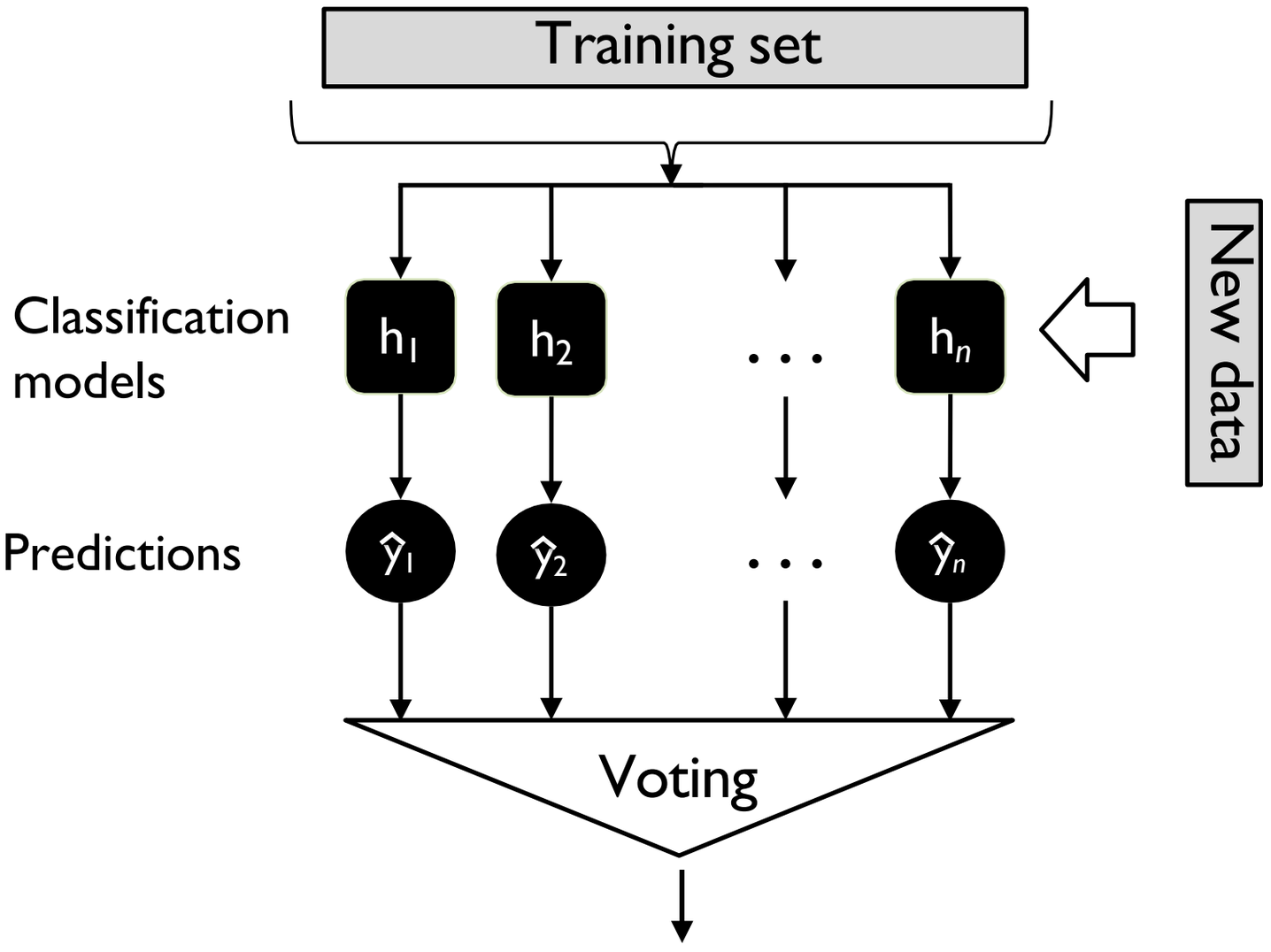
Unanimity

Majority



Plurality

Majority Voting 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

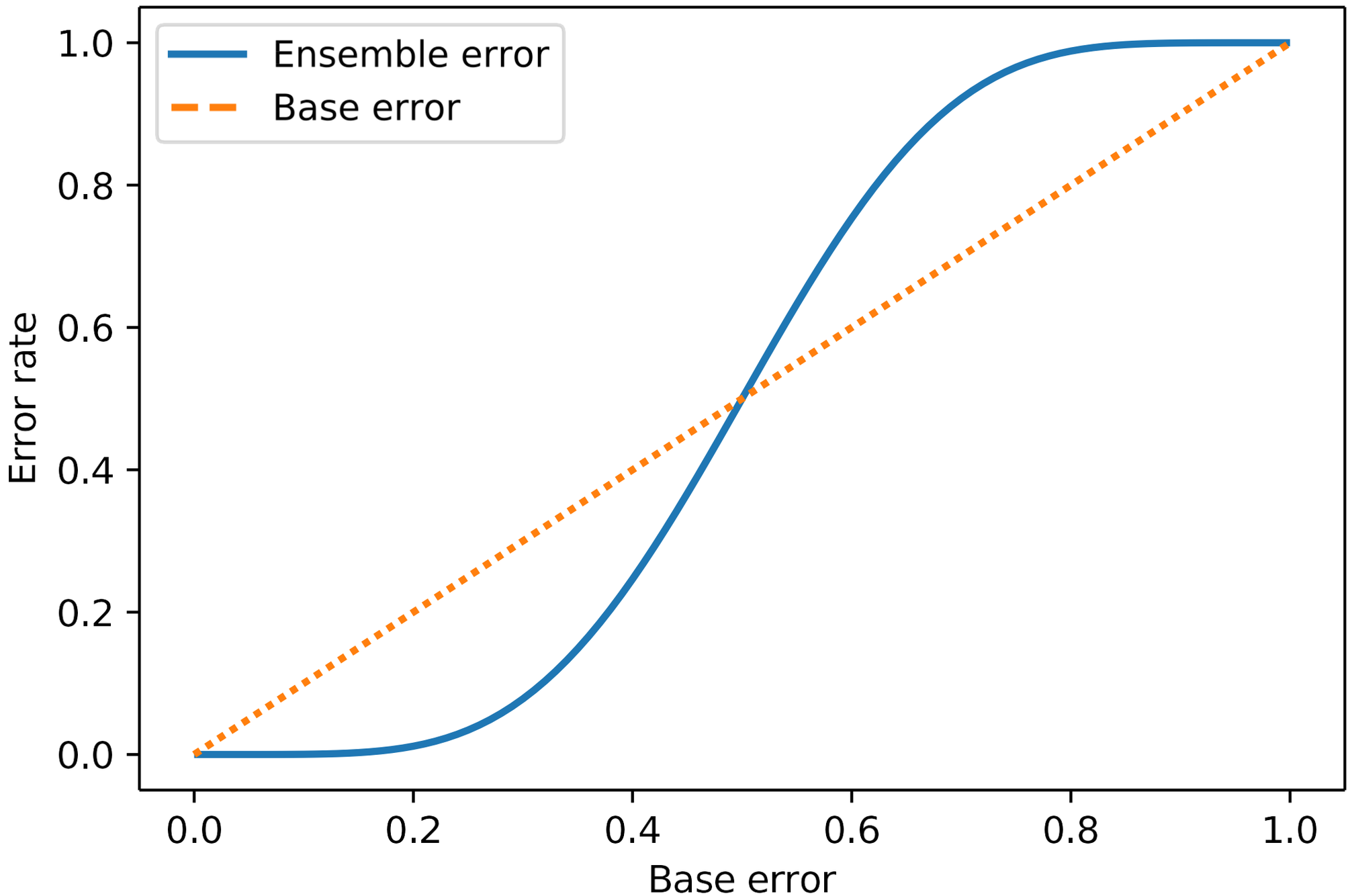
$$P(k) = \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k} \qquad k > \lceil n/2 \rceil$$

Ensemble error:

$$\epsilon_{ens} = \sum_{k}^{n} \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k}$$ (cumulative prob. distribution)

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

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



"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_i: 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

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

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

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

Binary classification example

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

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

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

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

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

Code

```
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
from mlxtend.classifier import EnsembleVoteClassifier

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])

clf1 = DecisionTreeClassifier(random_state=1)
clf2 = DecisionTreeClassifier(random_state=1, max_depth=1)
clf3 = DecisionTreeClassifier(random_state=1, max_depth=3)
eclf = EnsembleVoteClassifier(clfs=[clf1, clf2, clf3], weights=[1, 1, 1])

labels = ['Classifier 1', 'Classifier 2', 'Classifier 3', 'Ensemble']
for clf, label in zip([clf1, clf2, clf3, eclf], labels):

    clf.fit(X_train, y_train)
    print("Validation Accuracy: %0.2f [%s]" % (clf.score(X_val, y_val), label))

print("Test Accuracy: %0.2f" % eclf.score(X_test, y_test))
```

```
Train/Valid/Test sizes: 84 28 38
Validation Accuracy: 0.86 [Classifier 1]
Validation Accuracy: 0.82 [Classifier 2]
Validation Accuracy: 0.93 [Classifier 3]
Validation Accuracy: 0.93 [Ensemble]
Test Accuracy: 0.95
```

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

Code

```
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
from mlxtend.classifier import EnsembleVoteClassifier

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])

clf1 = DecisionTreeClassifier(random_state=1)
clf2 = DecisionTreeClassifier(random_state=1, max_depth=1)
clf3 = DecisionTreeClassifier(random_state=1, max_depth=3)
eclf = EnsembleVoteClassifier(clfs=[clf1, clf2, clf3], weights=[1, 1, 1])

labels = ['Classifier 1', 'Classifier 2', 'Classifier 3', 'Ensemble']
for clf, label in zip([clf1, clf2, clf3, eclf], labels):

    clf.fit(X_train, y_train)
    print("Validation Accuracy: %0.2f [%s]" % (clf.score(X_val, y_val), label))

print("Test Accuracy: %0.2f" % eclf.score(X_test, y_test))
```

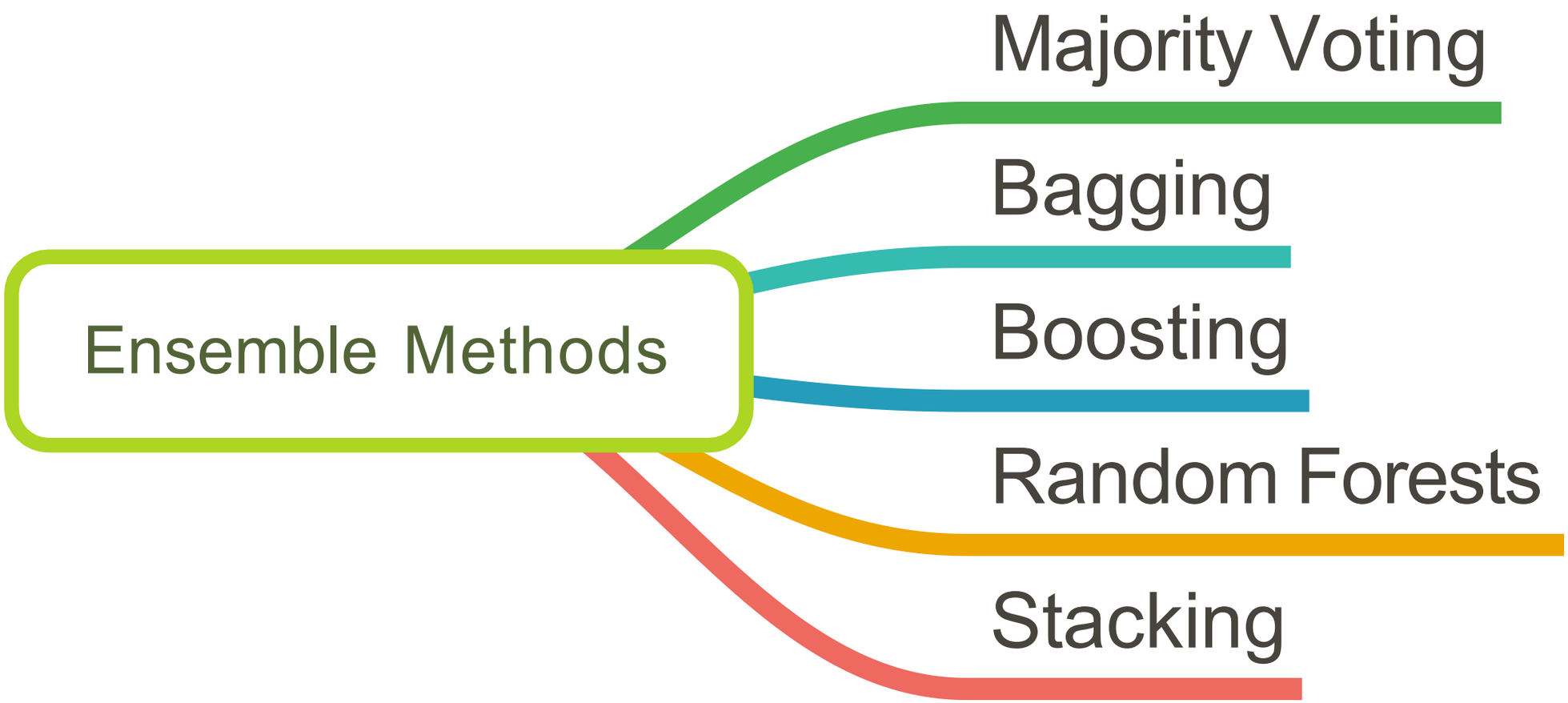
Now also available as:
sklearn.ensemble.**VotingClassifier**

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

```
Train/Valid/Test sizes: 84 28 38
Validation Accuracy: 0.86 [Classifier 1]
Validation Accuracy: 0.82 [Classifier 2]
Validation Accuracy: 0.93 [Classifier 3]
Validation Accuracy: 0.93 [Ensemble]
Test Accuracy: 0.95
```

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

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

1. Ensemble Methods -- Intro and Overview
2. Majority Voting
3. **Bagging**
4. Boosting
5. Gradient Boosting
6. Random Forests
7. Stacking

Bagging

(Bootstrap Aggregating)

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

Bagging

(Bootstrap Aggregating)

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, D_i
5: Train base classifier h_i on D_i
6: $\hat{y} = mode\{h_1(\mathbf{x}), \ldots, h_n(\mathbf{x})\}$

Bootstrap Sampling

Bootstrap Sampling

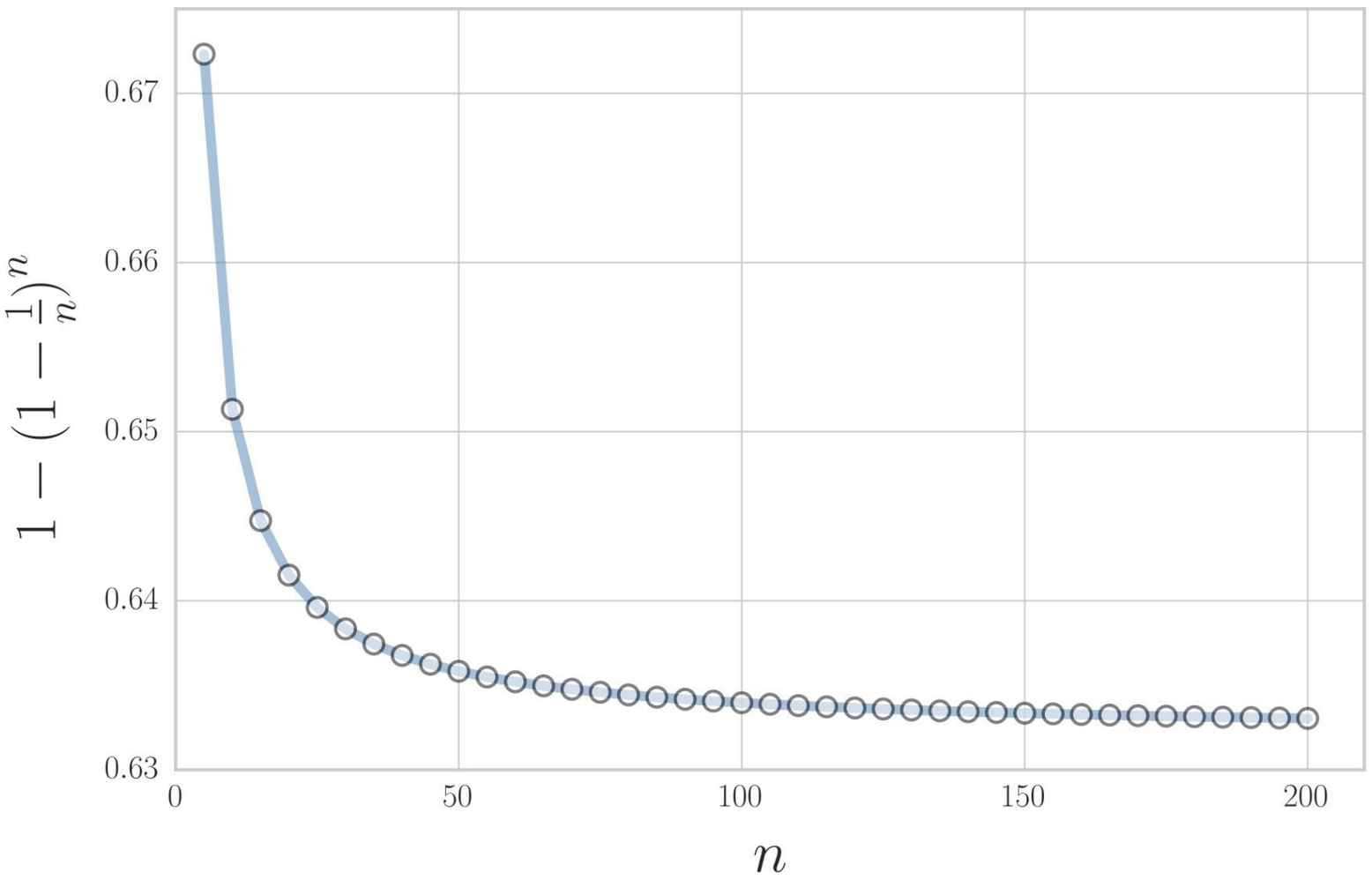
$$P(\text{not chosen}) = \left(1 - \frac{1}{m}\right)^m,$$

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

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

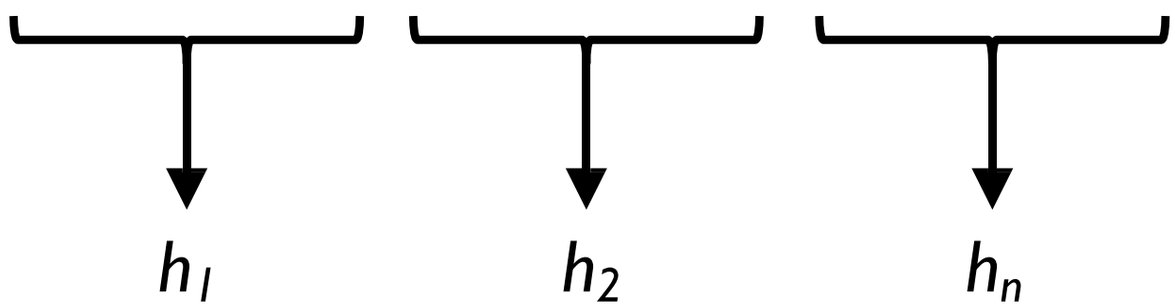
$$\frac{1}{e} \approx 0.368, \quad m \to \infty.$$

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

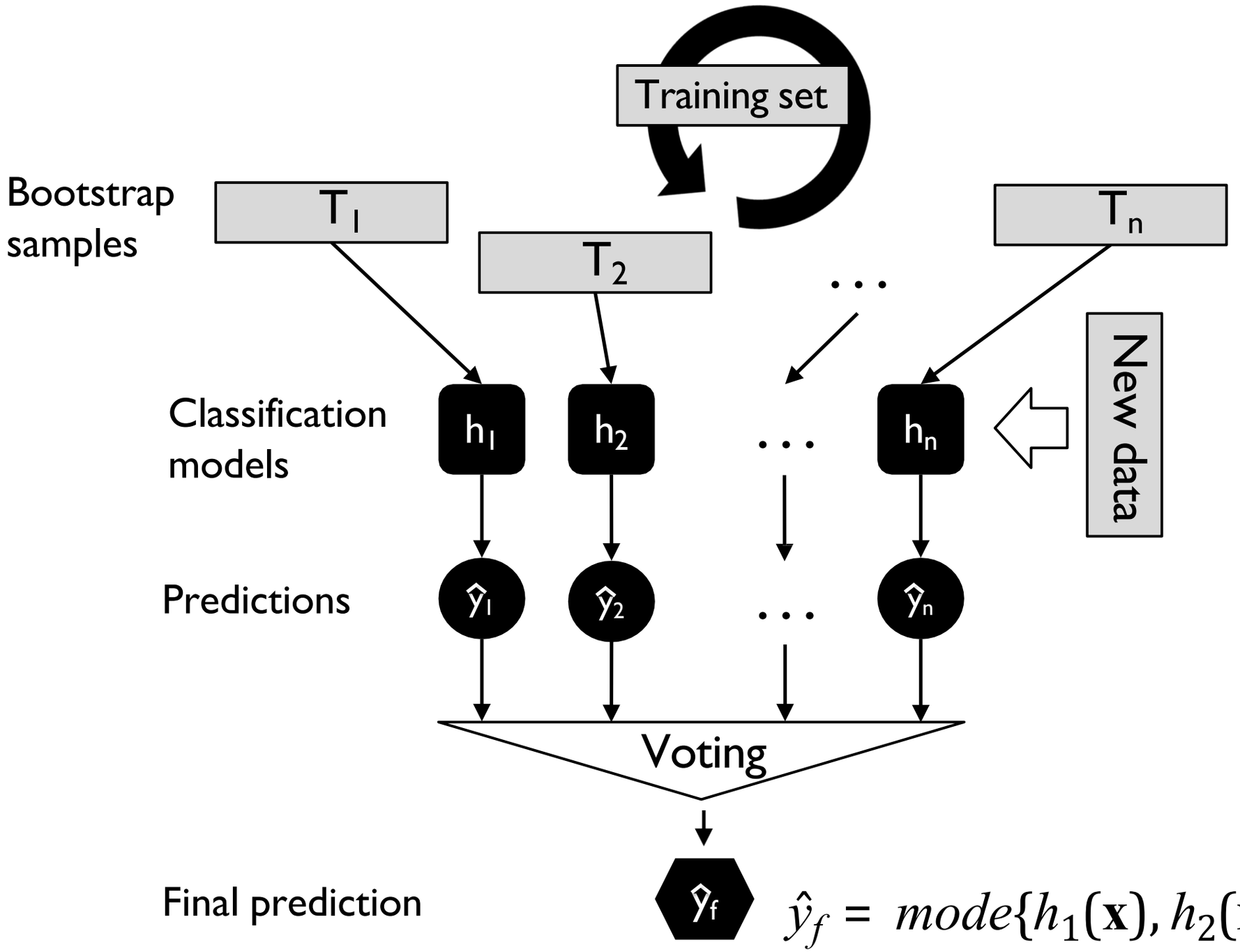


Bootstrap Sampling

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	…



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

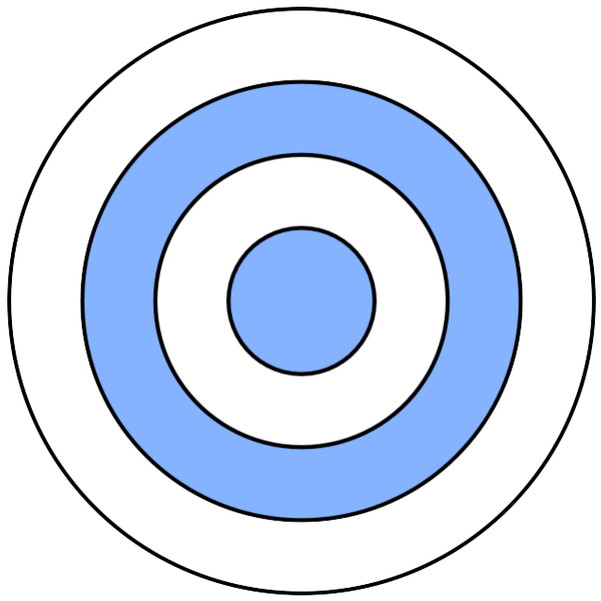
Loss = Bias + Variance + Noise

Bias-Variance Intuition

Low Variance
(Precise)

High Variance
(Not Precise)

Low Bias
(Accurate)



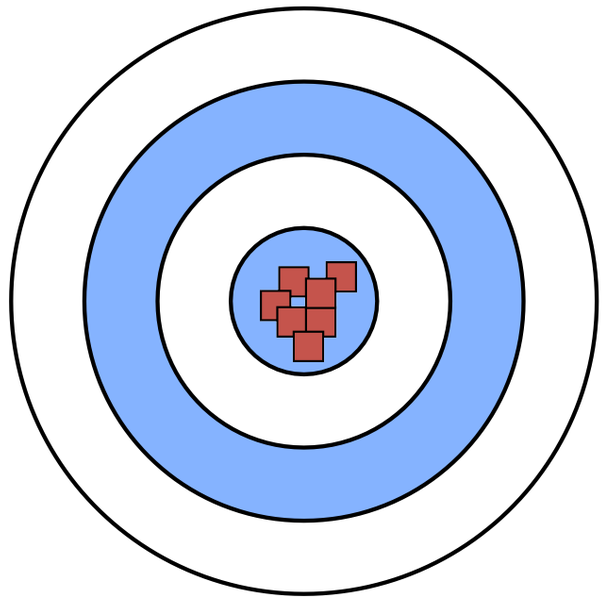
High Bias
(Not Accurate)

Bias-Variance Intuition

Low Variance
(Precise)

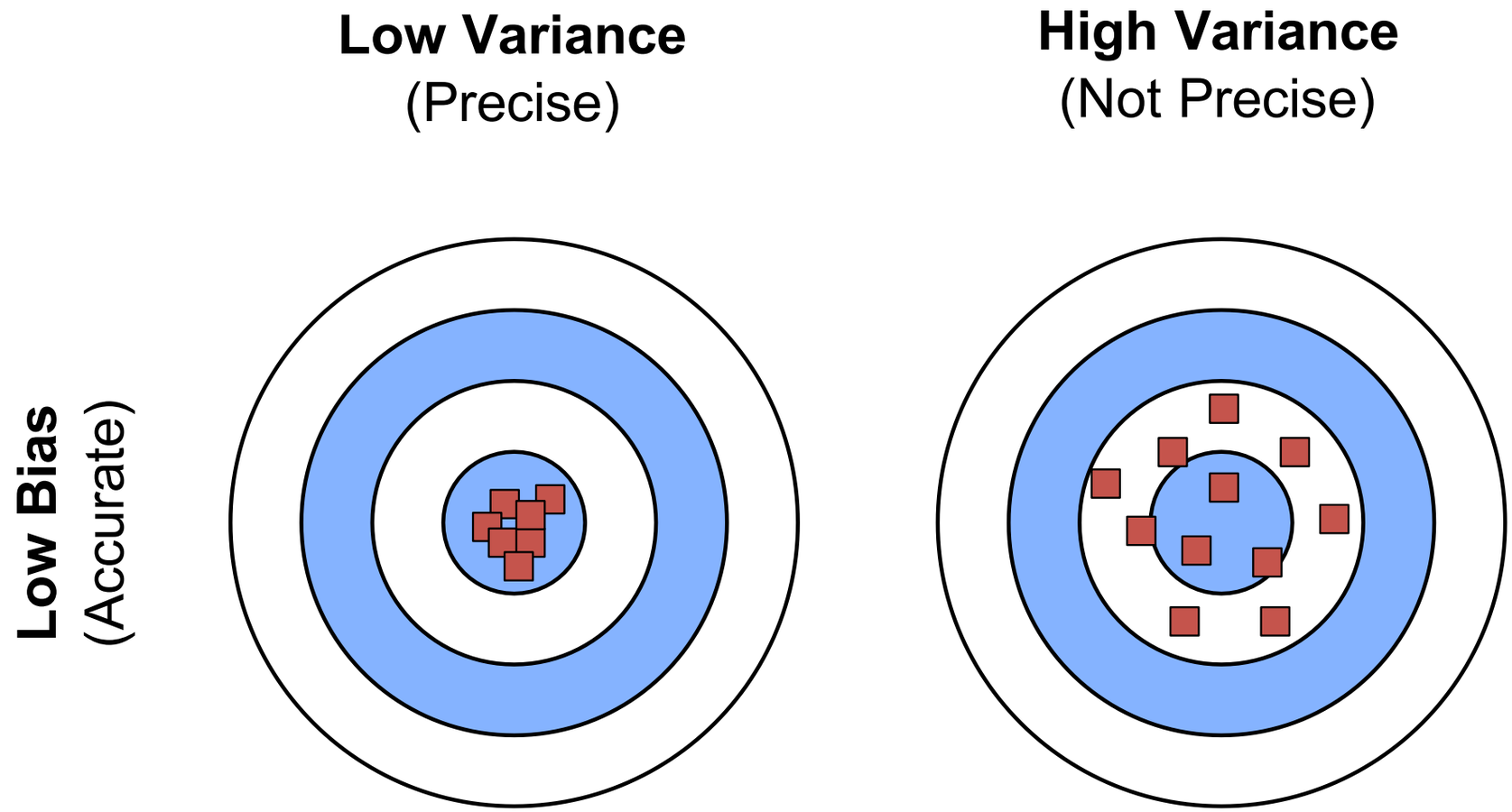
High Variance
(Not Precise)

Low Bias
(Accurate)

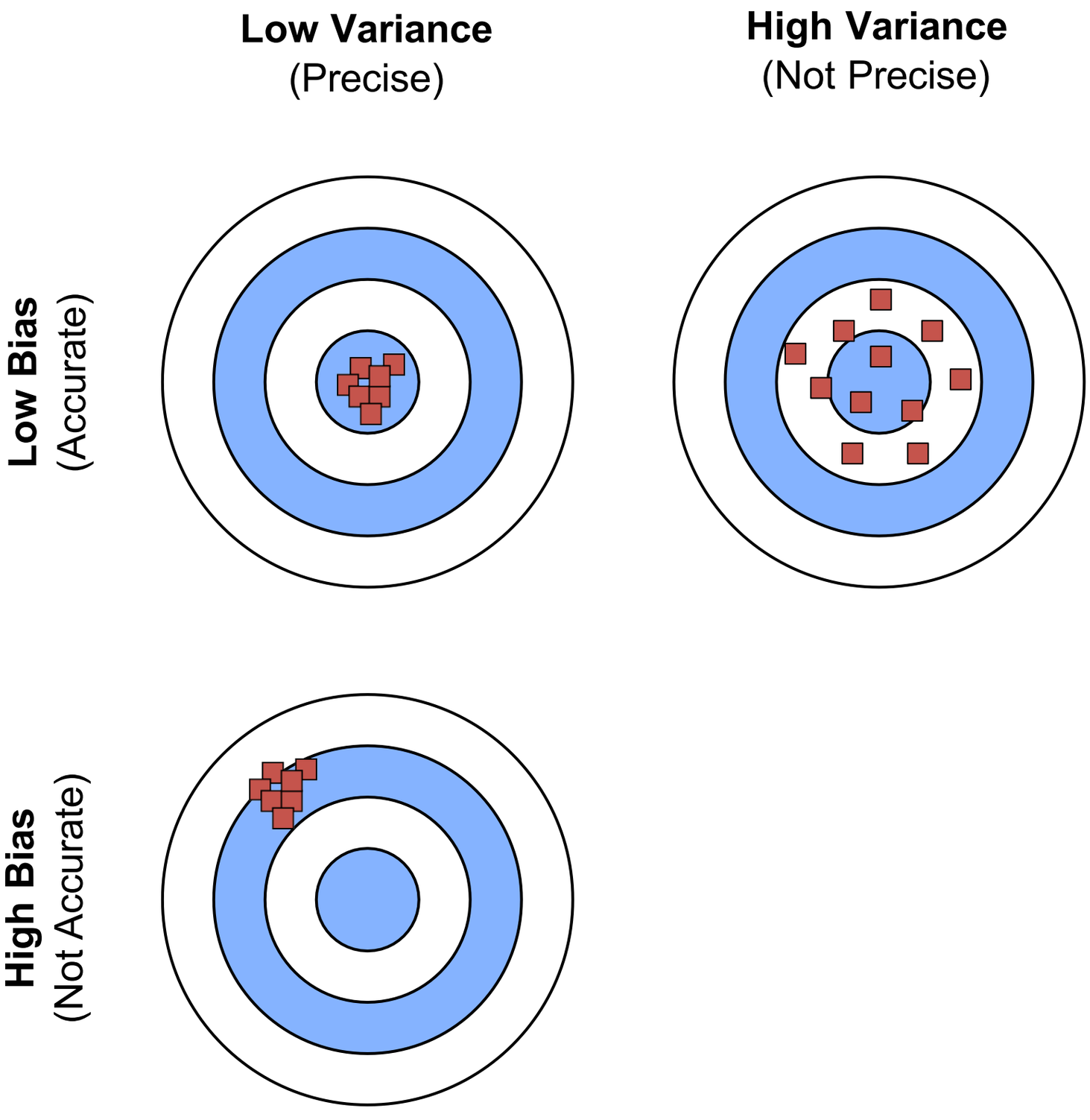


High Bias
(Not Accurate)

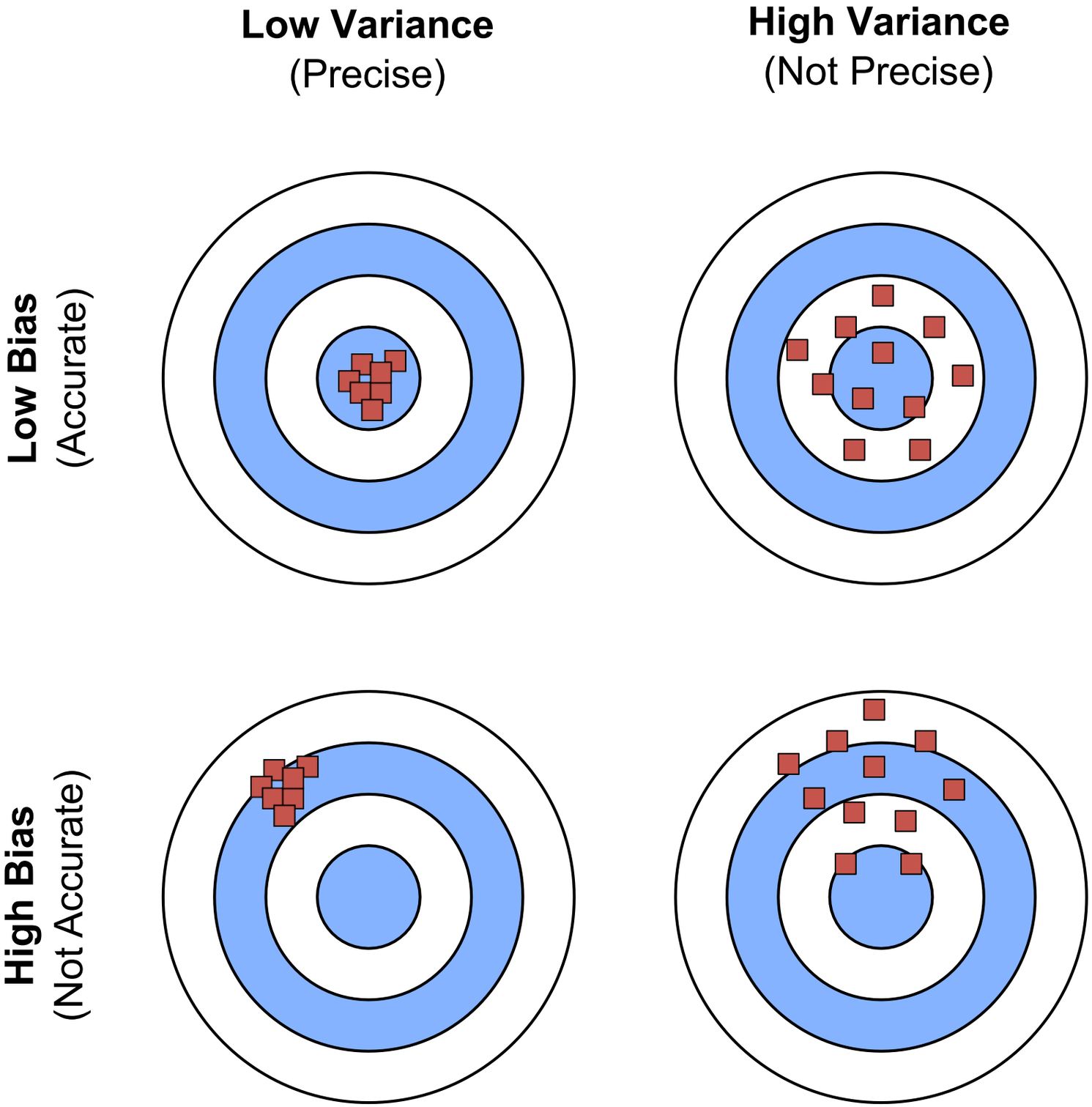
Bias-Variance Intuition



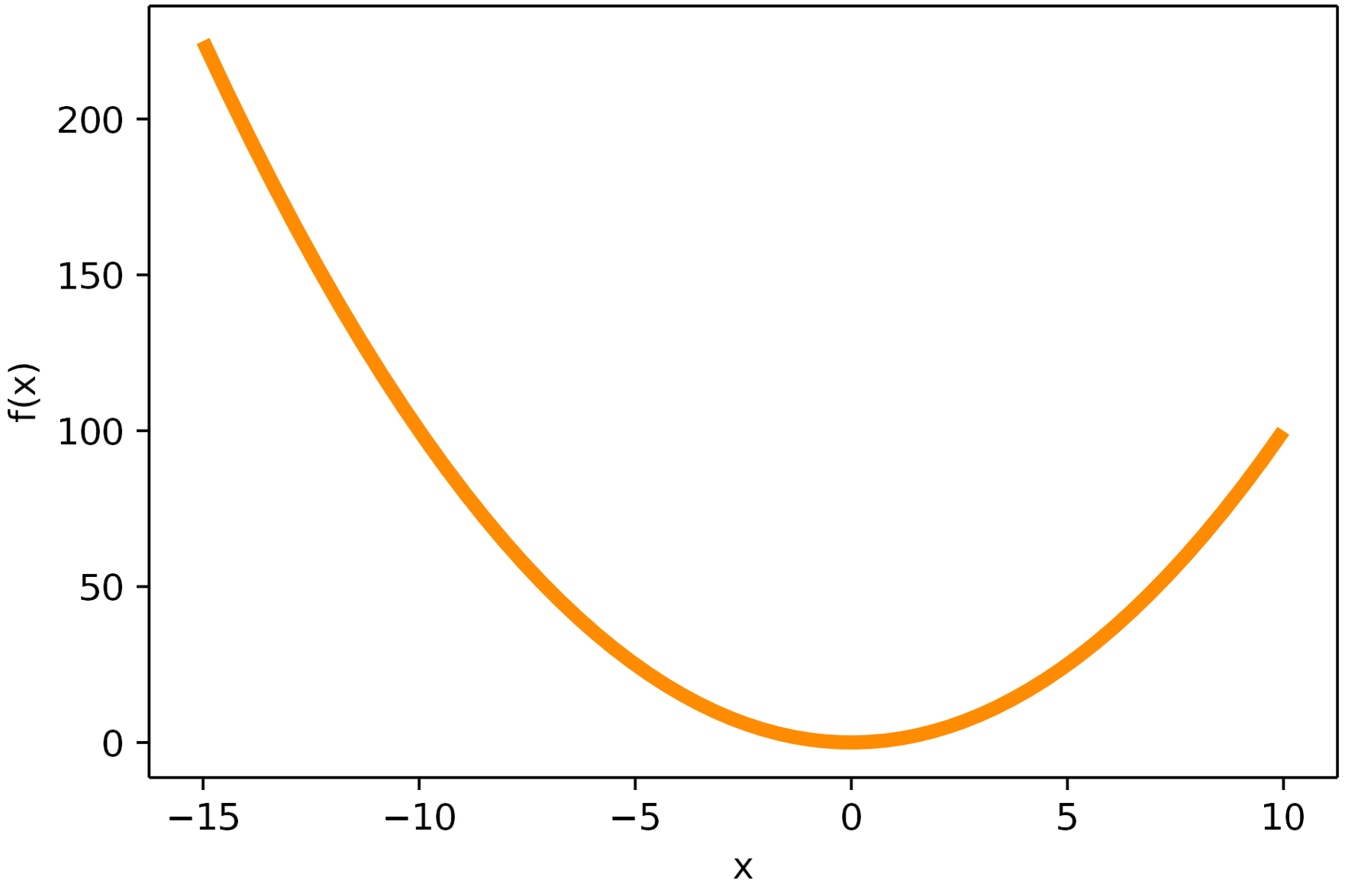
Bias-Variance Intuition



Bias-Variance Intuition

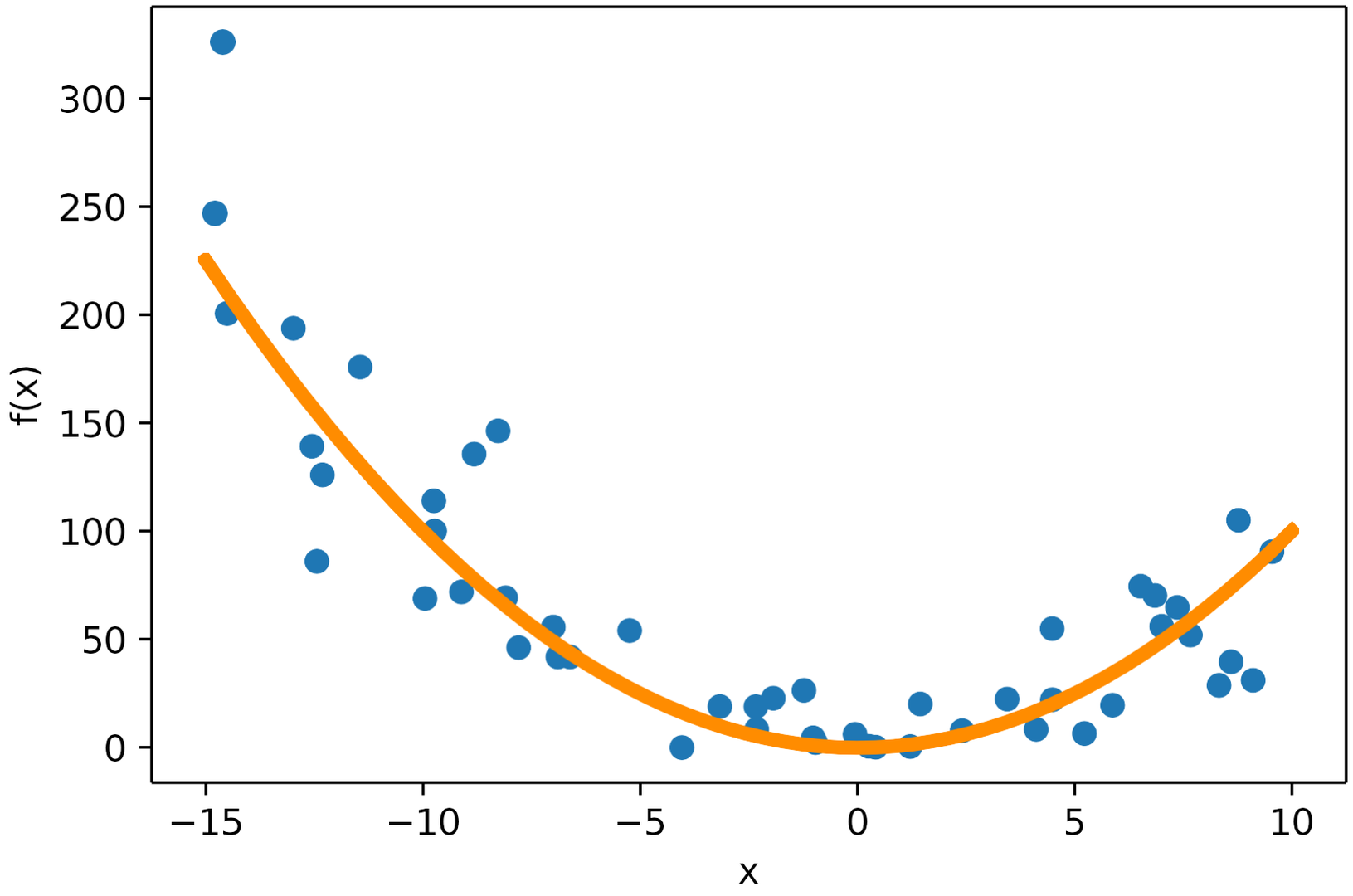


Bias and Variance Example



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

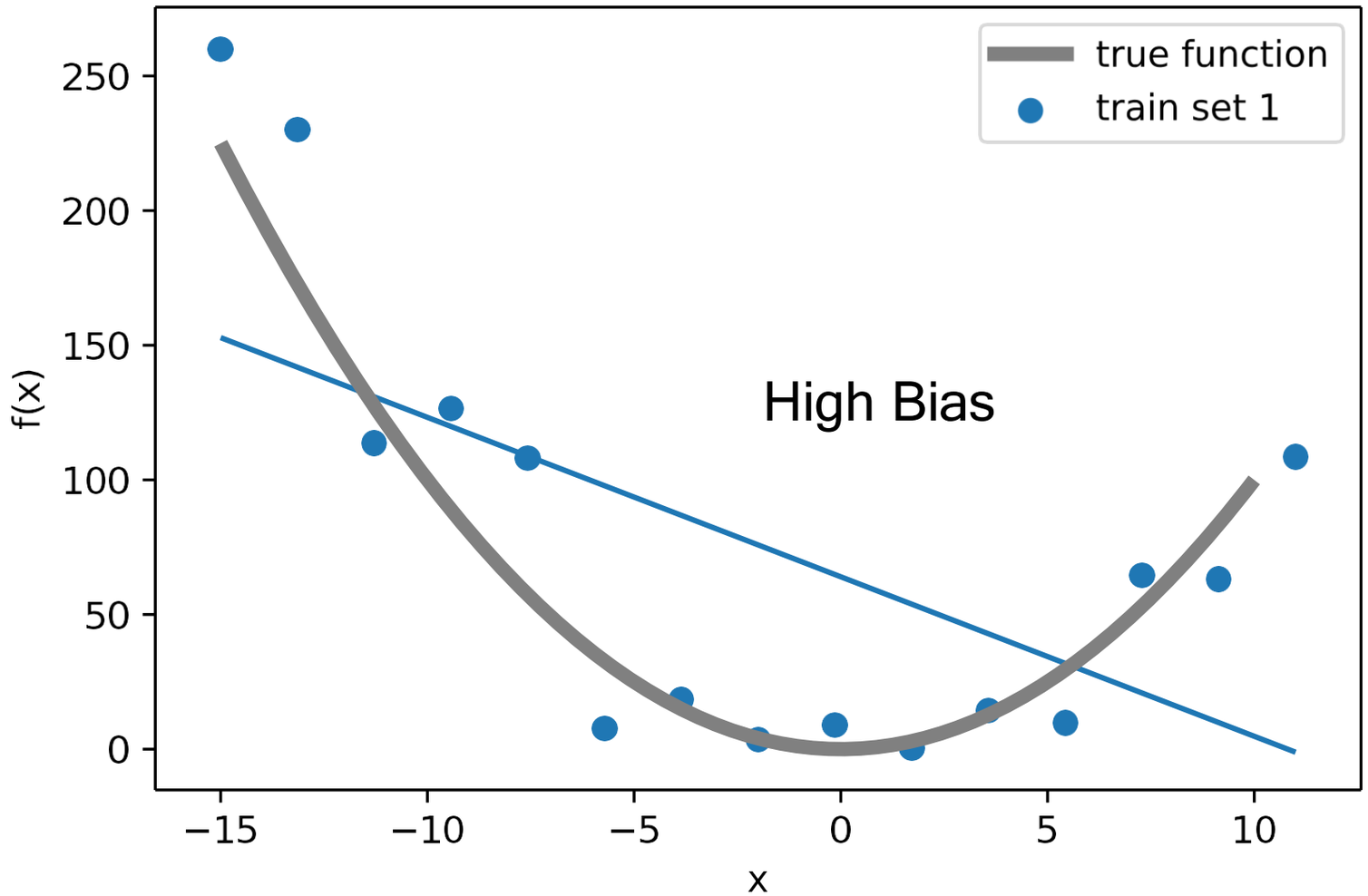
Bias and Variance Example



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

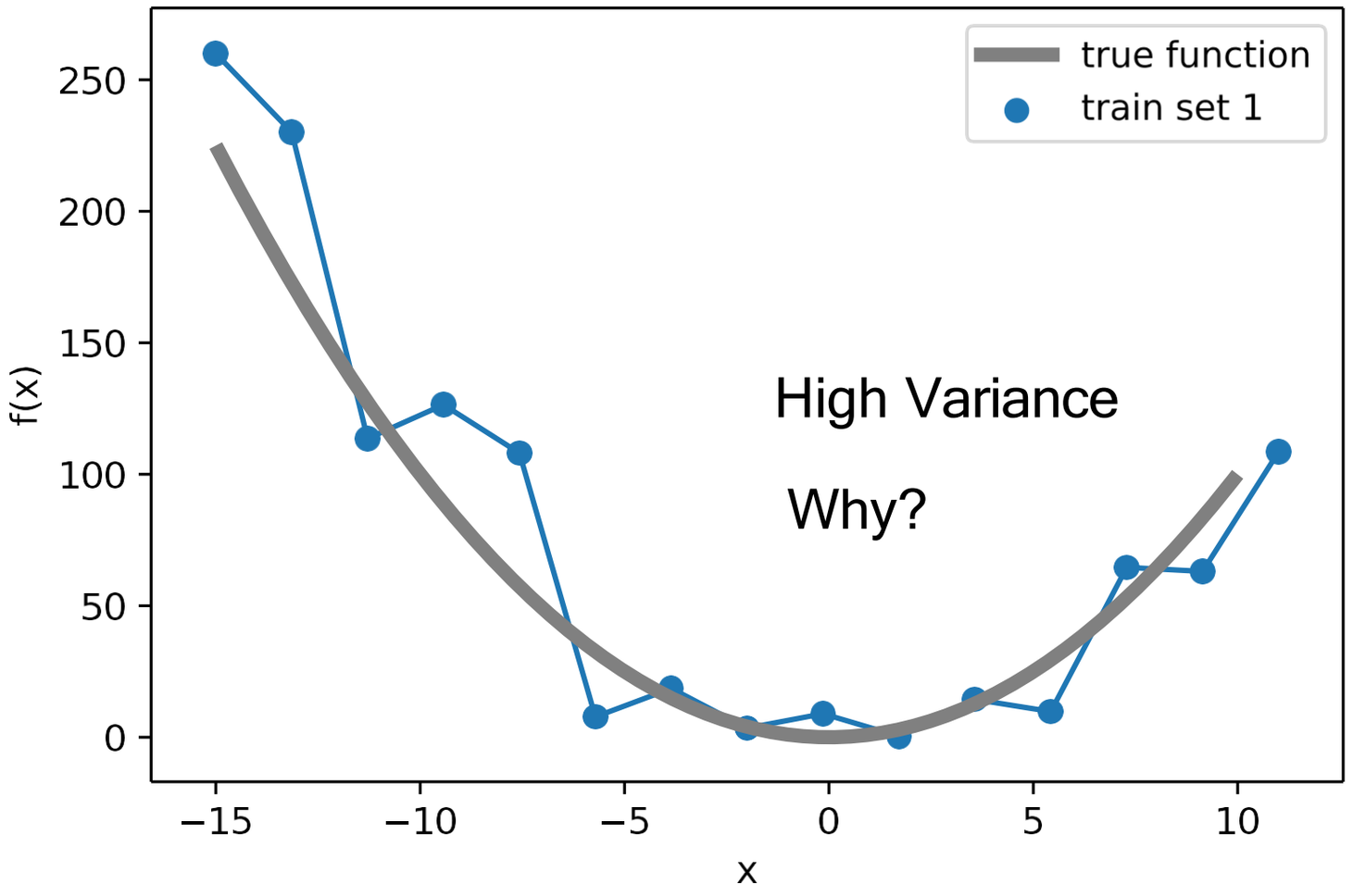


where f(x) is some true (target) function

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

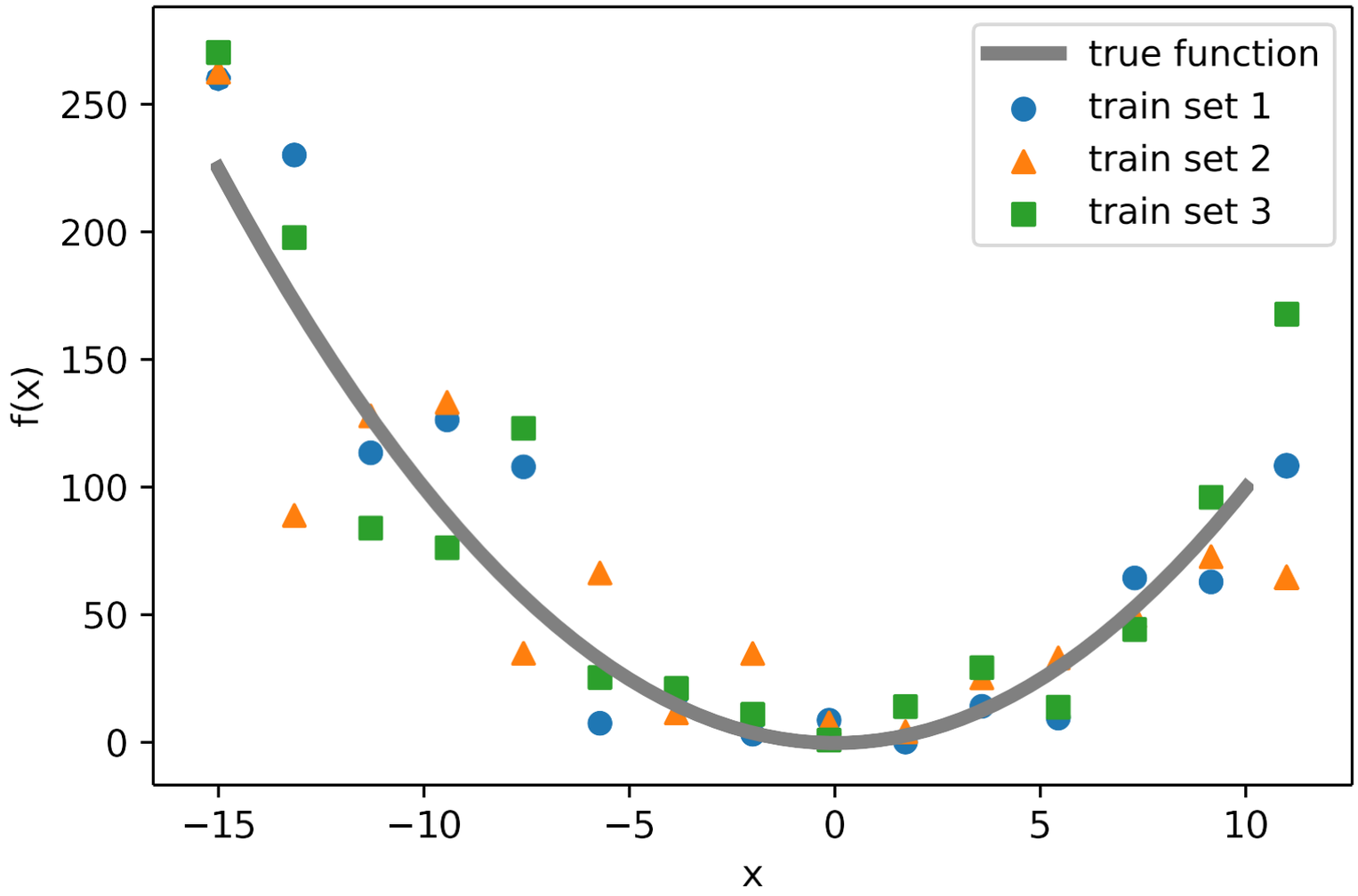


where f(x) is some true (target) function

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

here, suppose I fit an unpruned decision tree

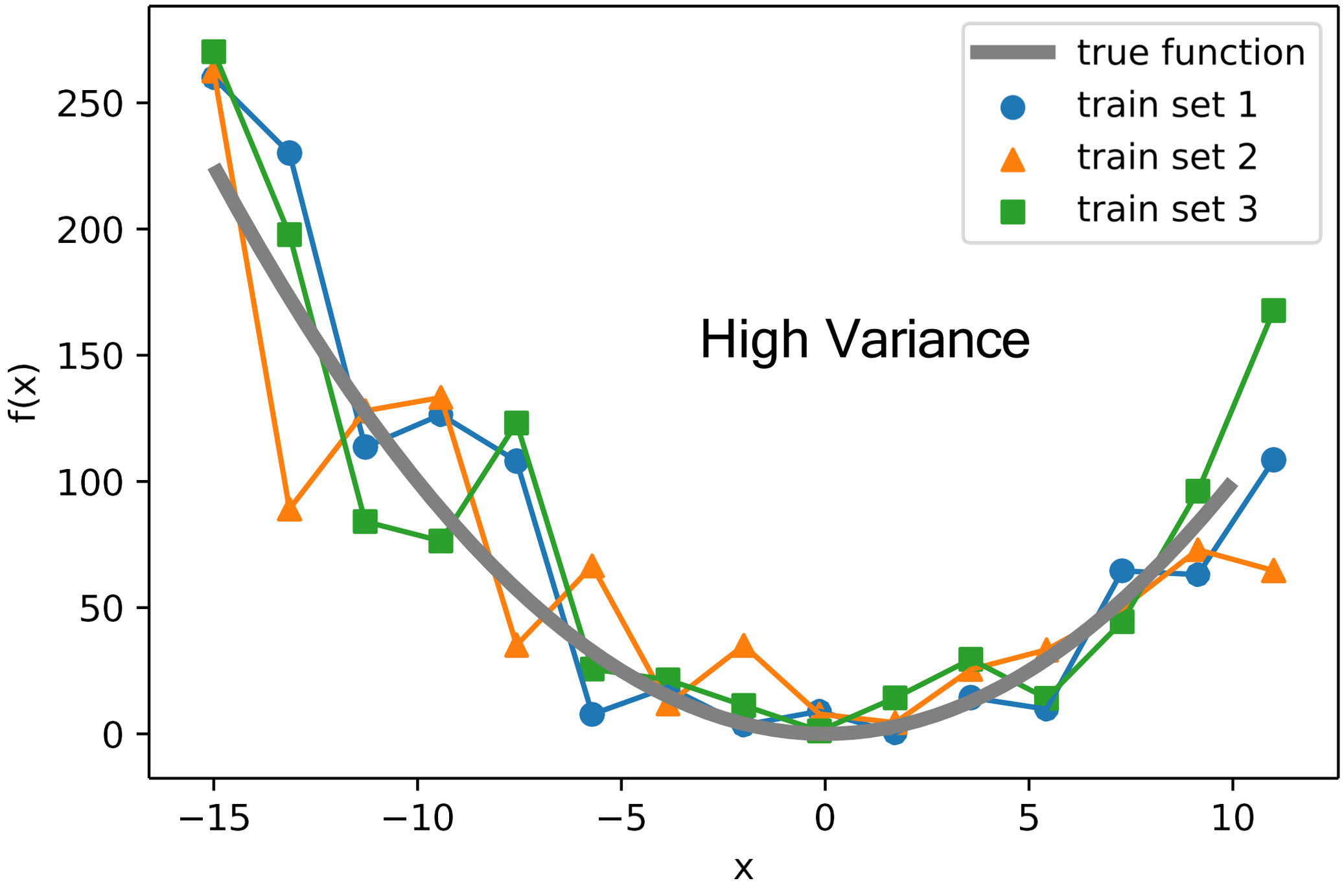
Bias and Variance Example



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

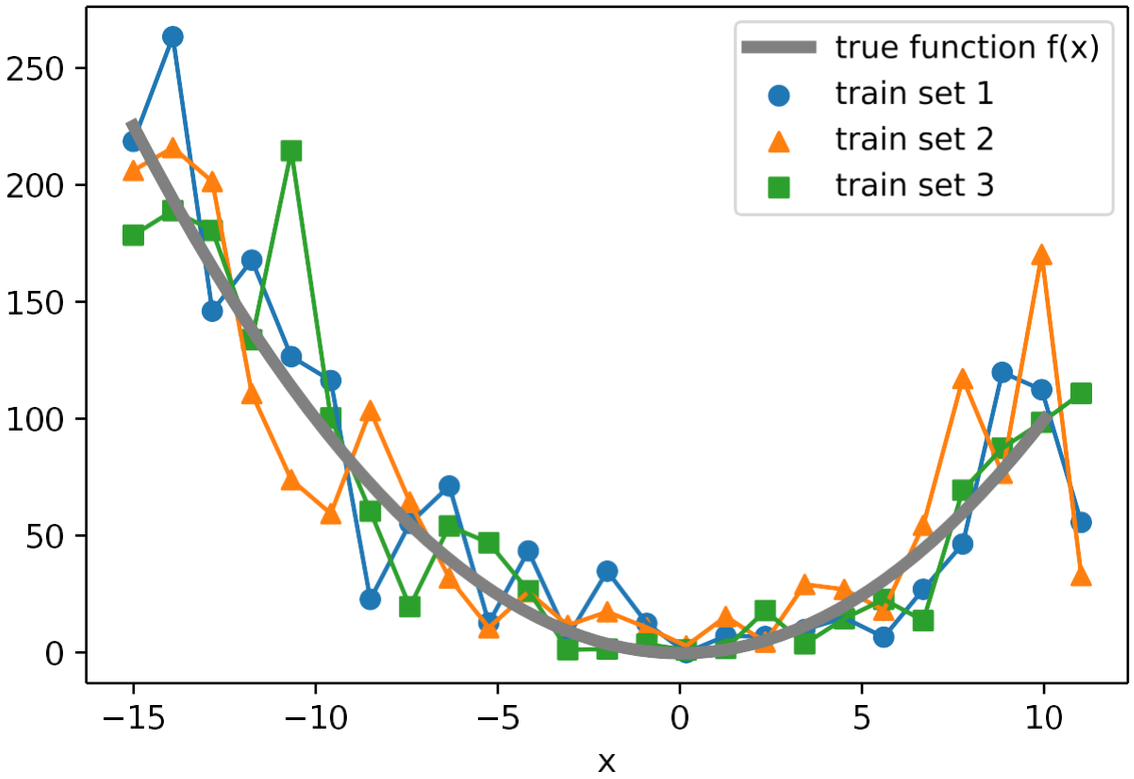
suppose we have multiple training sets

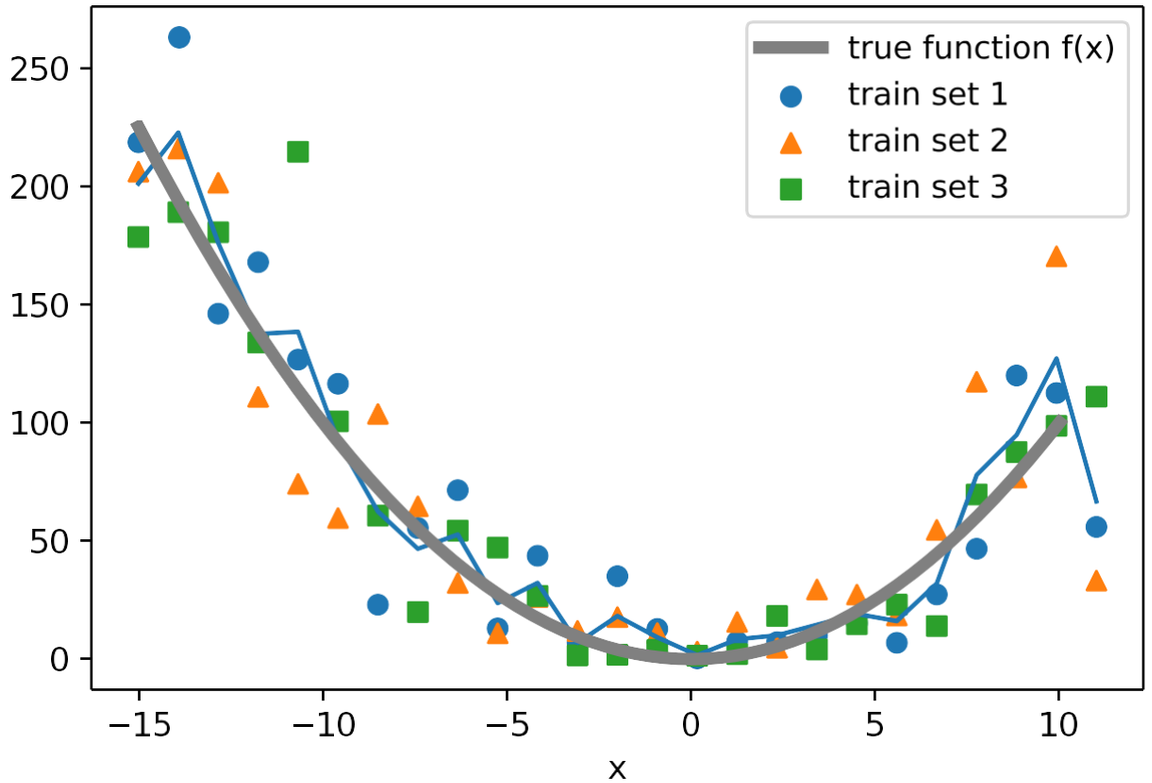
Bias and Variance Example



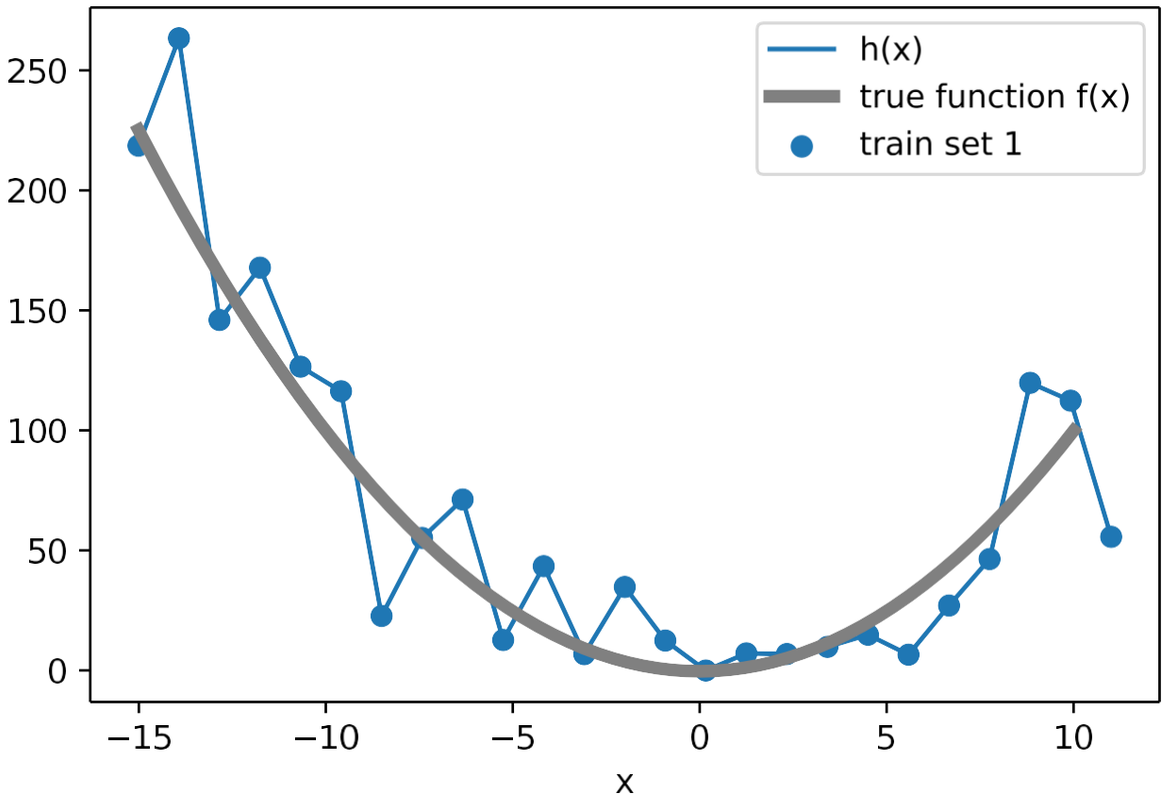
So, why does bagging work/what does it do?

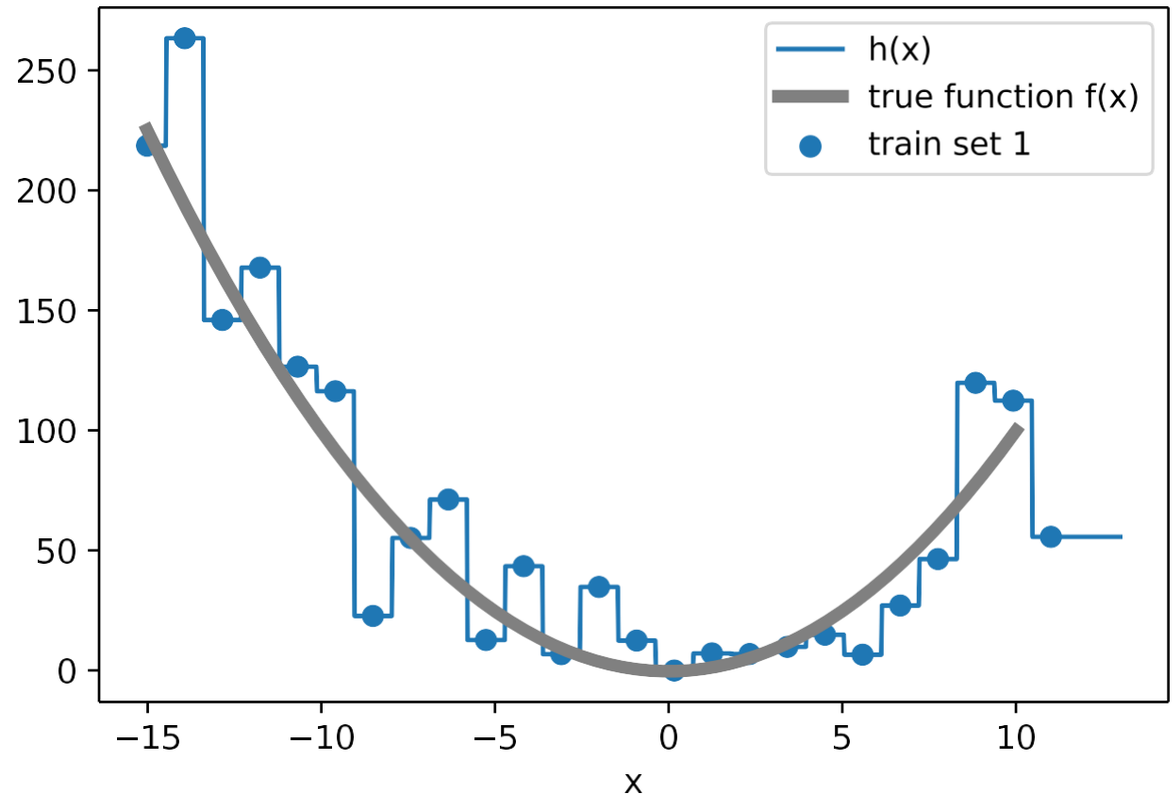
Clarifications 1





Clarifications 2



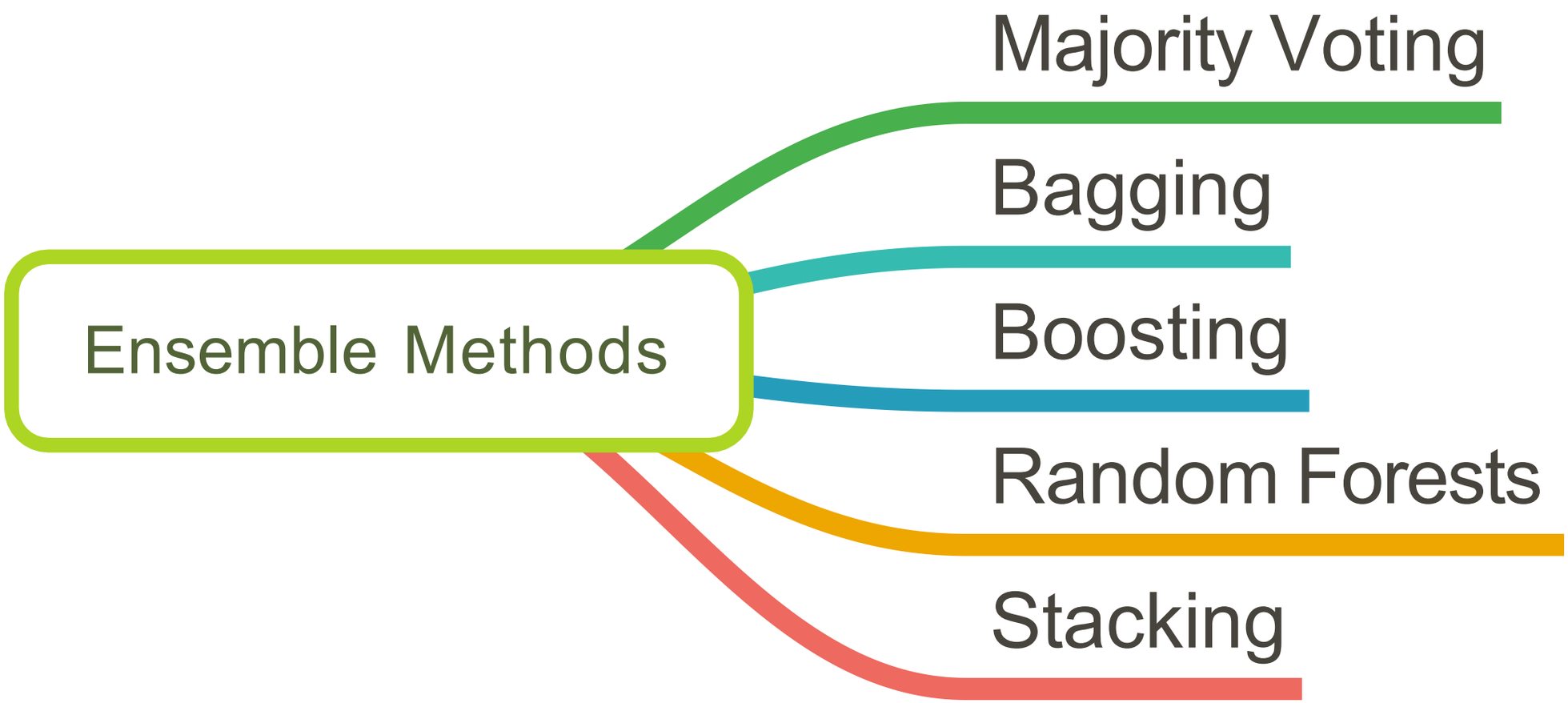


Code

```
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.ensemble import BaggingClassifier

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])

tree = DecisionTreeClassifier(criterion='entropy',
                              random_state=1,
                              max_depth=None)


bag = BaggingClassifier(base_estimator=tree,
                        n_estimators=500,
                        oob_score=True,
                        bootstrap=True,
                        bootstrap_features=False,
                        n_jobs=1,
                        random_state=1)

bag.fit(X_train, y_train)

print("OOB Accuracy: %0.2f" % bag.oob_score_)
print("Test Accuracy: %0.2f" % bag.score(X_test, y_test))
```

```
Train/Valid/Test sizes: 84 28 38
OOB Accuracy: 0.93
Test Accuracy: 0.95
```

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

1. Ensemble Methods -- Intro and Overview
2. Majority Voting
3. Bagging
4. **Boosting**
5. Gradient Boosting
6. Random Forests
7. 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

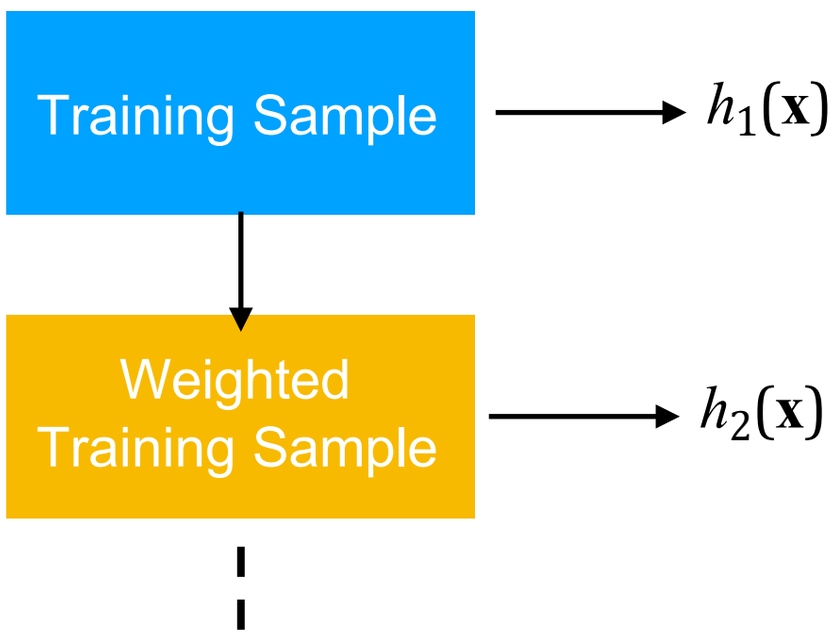
Gradient Boosting

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



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

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

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

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

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

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

Estimator weight

Sample weight

Decision Tree Stumps

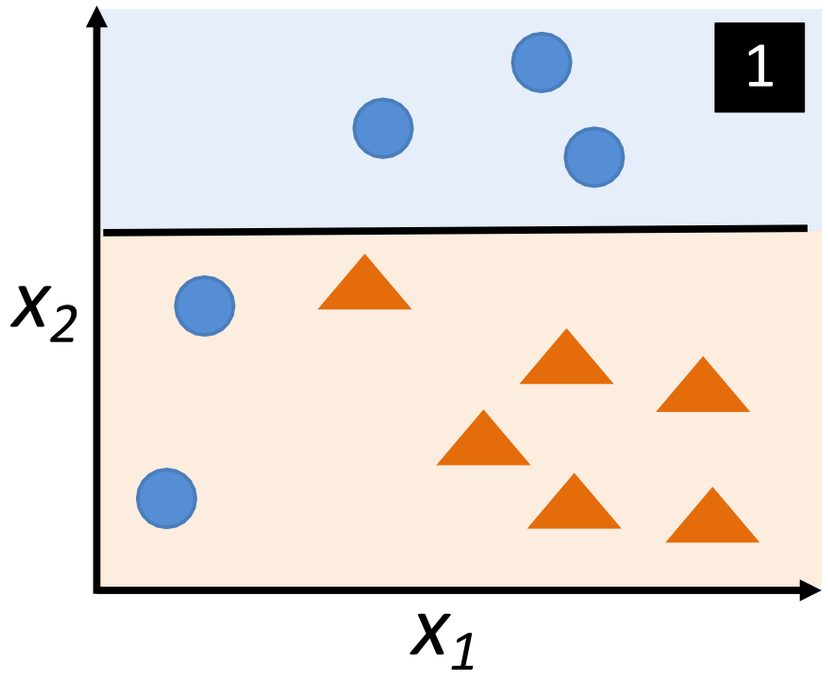
Weak classifier, here: decision tree stump for binary classification problem with labels -1, 1

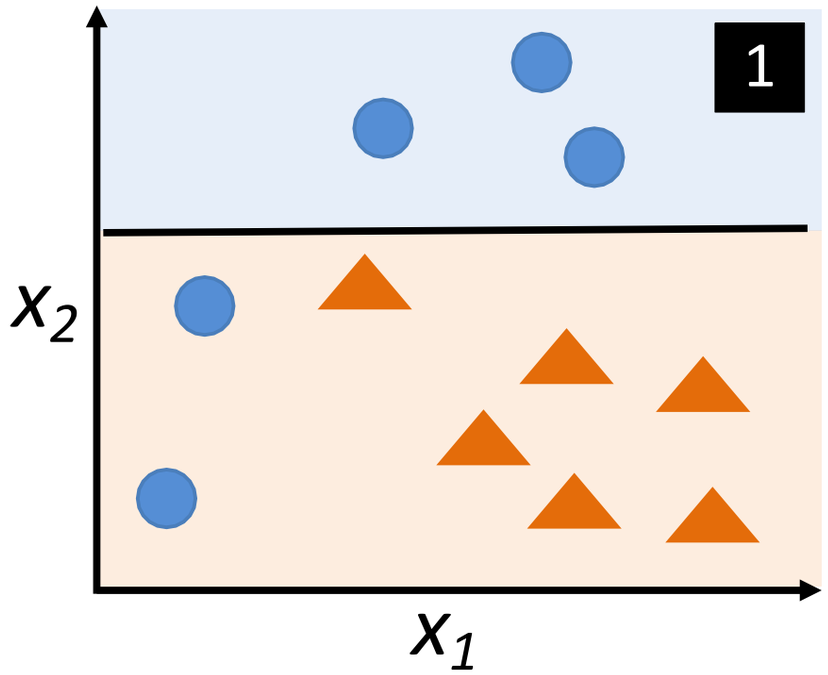
$$h(\mathbf{x}) = s(\mathbf{1}(x_k \geq t))$$

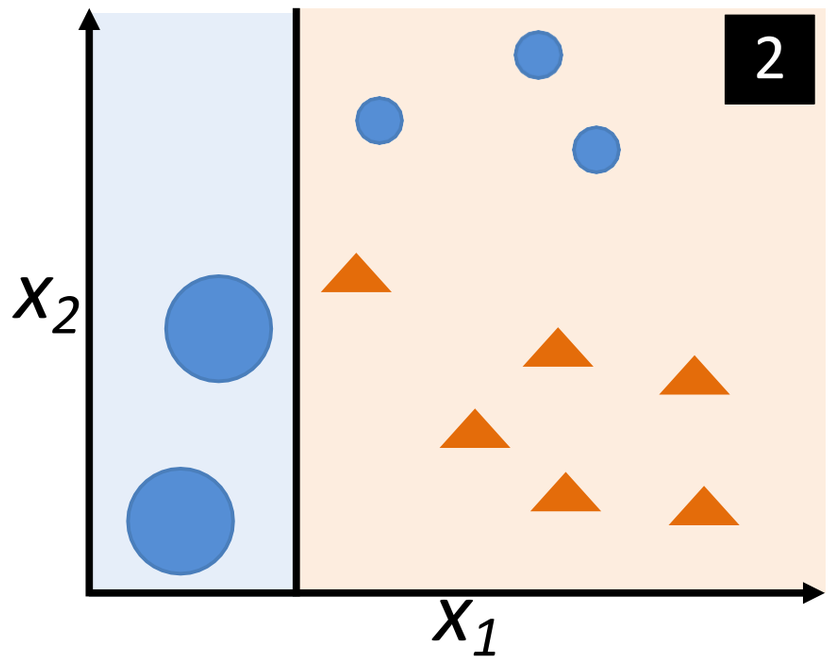
where

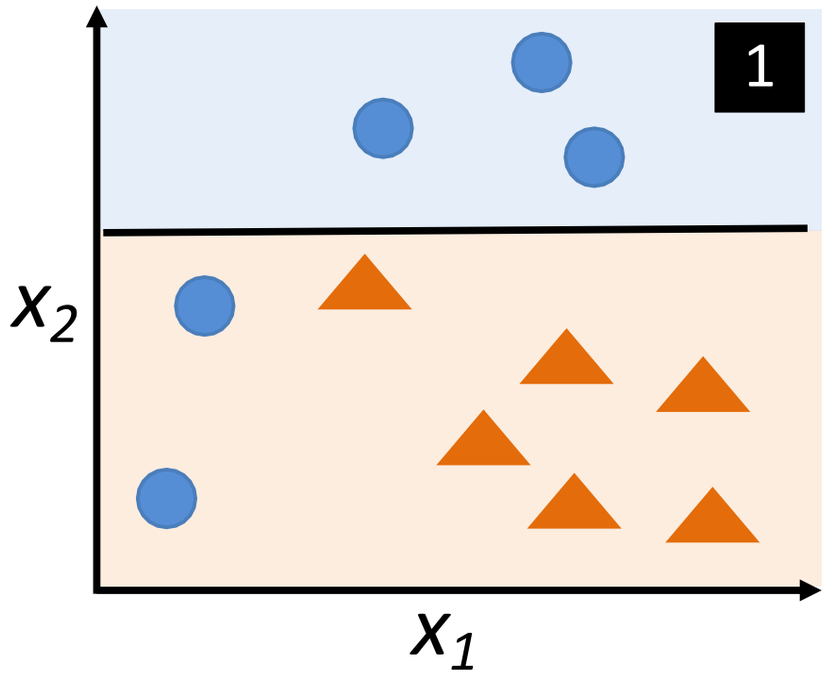
$s(x) \in \{-1, 1\}$

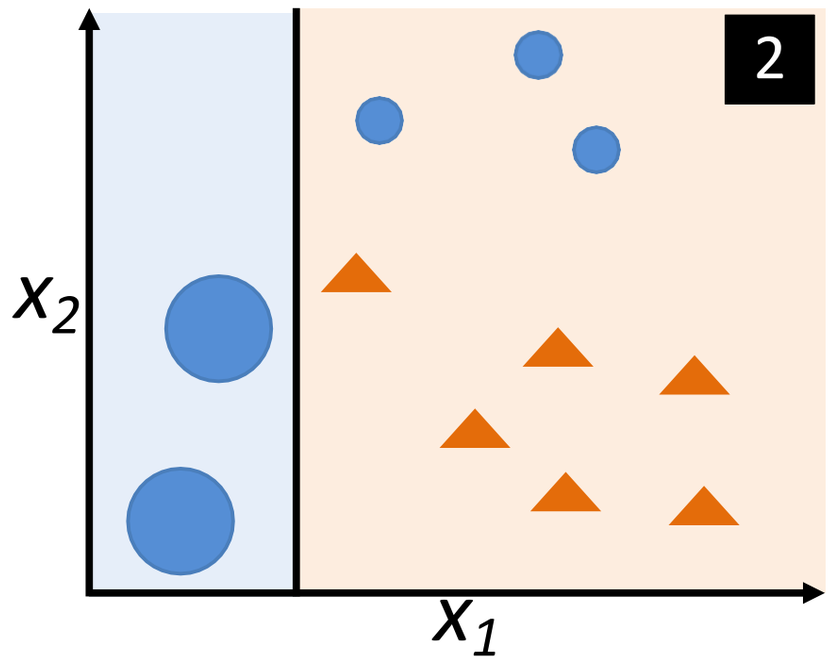
$k \in \{1, \ldots, K\}$ (is the number of features)

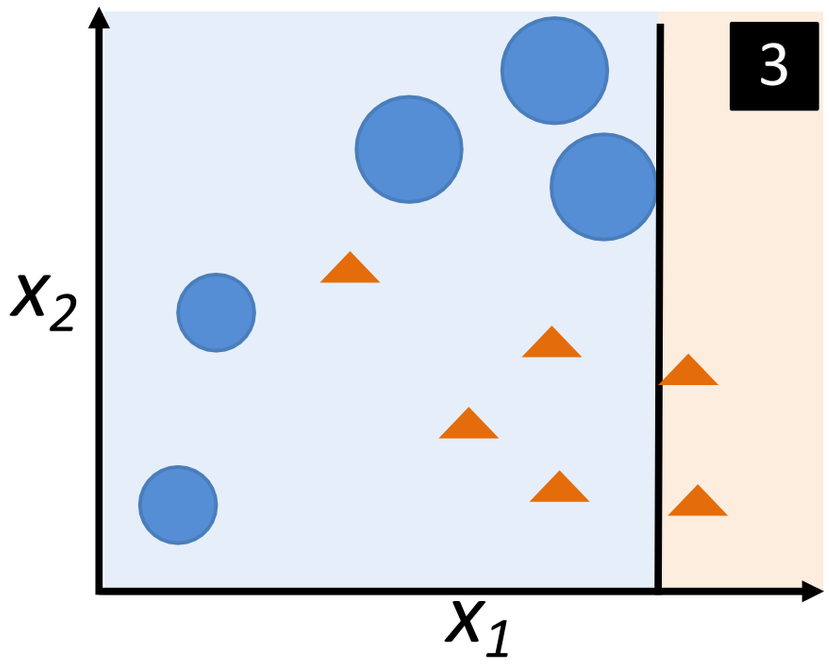

1
x_2
x_1

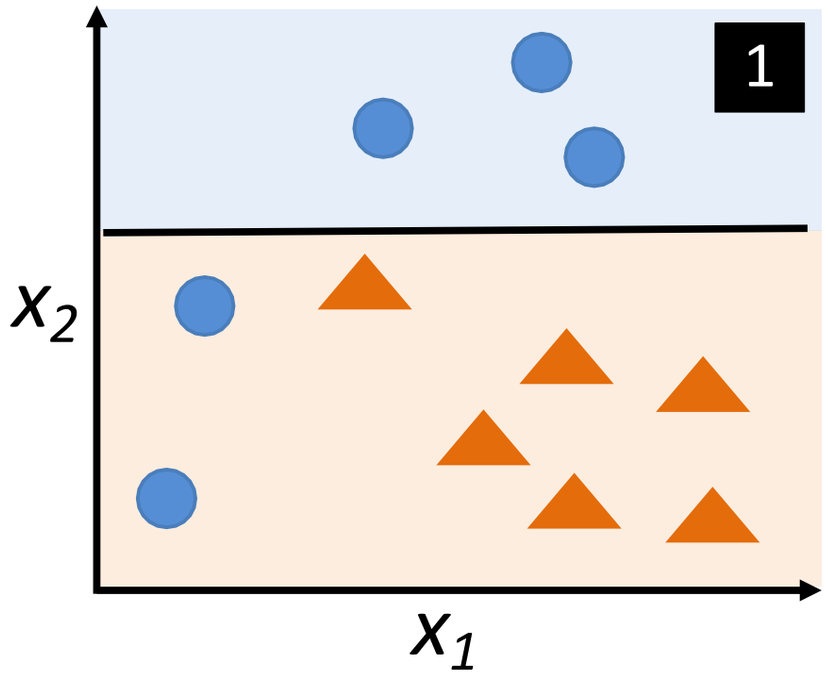

1
x_2
x_1

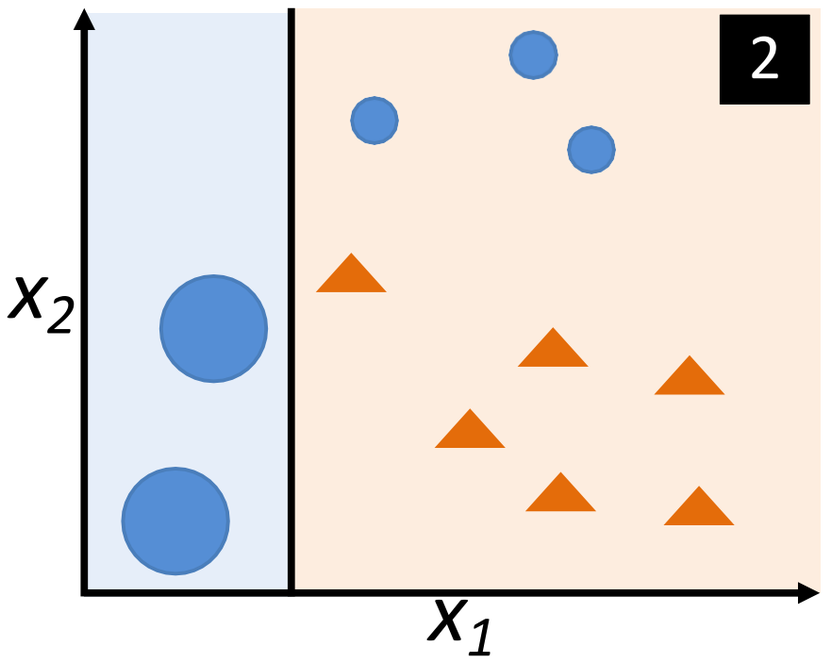

2
x_2
x_1

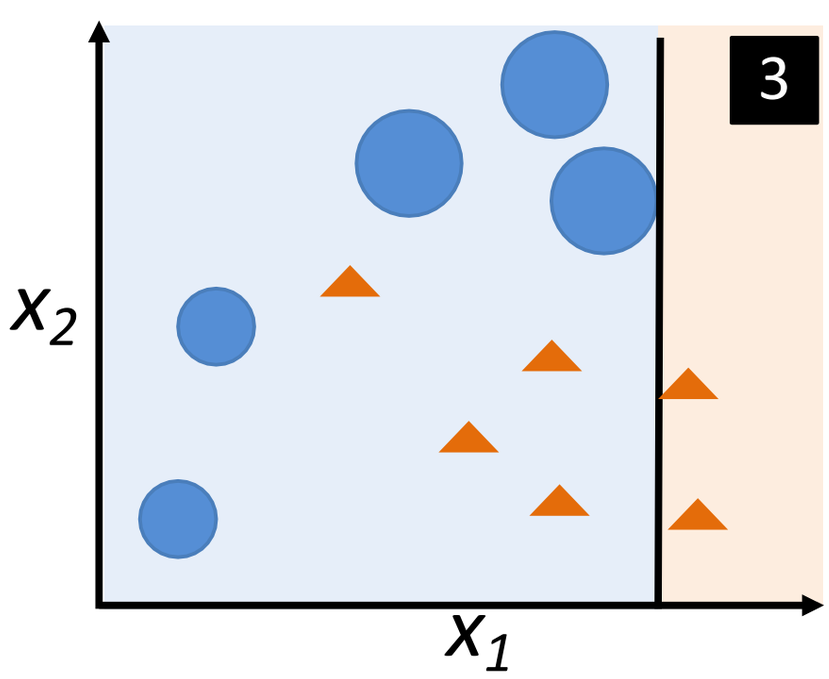

1
x_2
x_1

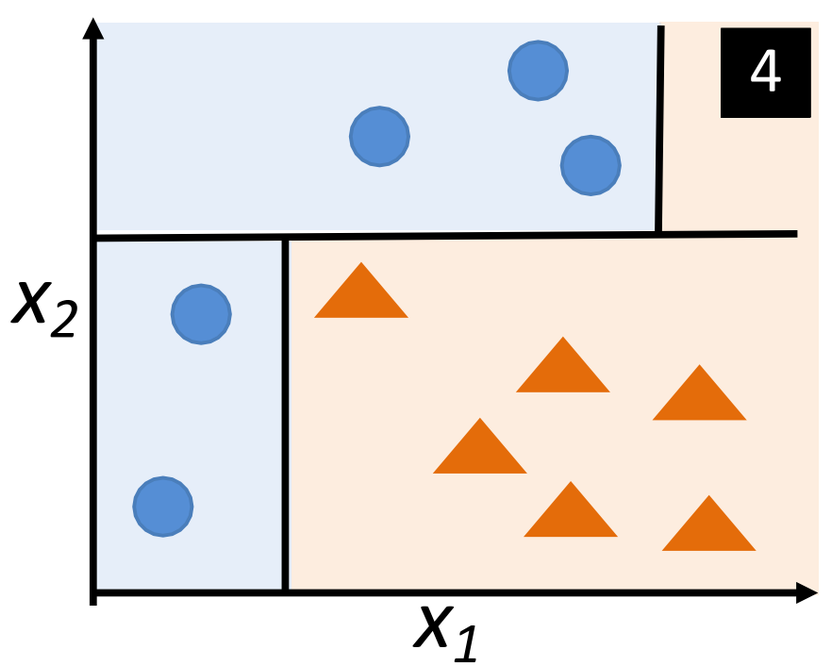

2
x_2
x_1


3
x_2
x_1


1
x_2
x_1


2
x_2
x_1


3
x_2
x_1


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

Statistics and Its Interface
Volume 2 (2009)
Number 3

Multi-class AdaBoost

Pages: 349 – 360

DOI: https://dx.doi.org/10.4310/SII.2009.v2.n3.a8

Authors
Trevor Hastie (Department of Statistics, Stanford University, Stanford, Calif., U.S.A.)
Saharon Rosset (Department of Statistics, Tel Aviv University, Tel Aviv, Israel)
Ji Zhu (Department of Statistics, University of Michigan, Ann Arbor, Mich., U.S.A.)
Hui Zou (School of Statistics, University of Minnesota, Minneapolis, Minn., U.S.A.)

Stagewise Additive Modeling using a Multi-class Exponential loss function (SAMME)

https://www.intlpress.com/site/pub/pages/journals/items/sii/content/vols/0002/0003/a008/

Algorithm 1. *AdaBoost [8]*

1. *Initialize the observation weights* $w_i = 1/n$, $i = 1, 2, \ldots, n$.
2. *For* $m = 1$ *to* M*:*
   (a) *Fit a classifier* $T^{(m)}(\boldsymbol{x})$ *to the training data using weights* w_i.
   (b) *Compute*

$$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$

   (c) *Compute*

$$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$$

   (d) *Set*

$$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),$$

   for $i = 1, 2, \ldots, n$.
   (e) *Re-normalize* w_i.
3. *Output*

$$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$

Algorithm 2. *SAMME*

1. *Initialize the observation weights* $w_i = 1/n$, $i = 1, 2, \ldots, n$.
2. *For* $m = 1$ *to* M*:*
   (a) *Fit a classifier* $T^{(m)}(\boldsymbol{x})$ *to the training data using weights* w_i.
   (b) *Compute*

$$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$

   (c) *Compute*

$$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K - 1). \tag{1}$$

   (d) *Set*

$$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),$$

   for $i = 1, \ldots, n$.
   (e) *Re-normalize* w_i.
3. *Output*

$$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$

Note that Algorithm 2 (SAMME) shares the same simple modular structure of AdaBoost with a *simple but subtle* difference in (1), specifically, the extra term $\log(K-1)$. Obviously, when $K = 2$, SAMME reduces to AdaBoost. However, the term $\log(K-1)$ in (1) is critical in the multi-class case ($K > 2$). One immediate consequence is that now in order

Code 1

```
tree = DecisionTreeClassifier(criterion='entropy',
                              random_state=1,
                              max_depth=None)


bag = BaggingClassifier(base_estimator=tree,
                        n_estimators=500,
                        oob_score=True,
                        bootstrap=True,
                        bootstrap_features=False,
                        n_jobs=1,
                        random_state=1)
```

```
tree = DecisionTreeClassifier(criterion='entropy',
                              random_state=1,
                              max_depth=1)


boost = AdaBoostClassifier(base_estimator=tree,
                           n_estimators=500,
                           #n_jobs=1,
                           random_state=1)
```

Code 2

```
tree = DecisionTreeClassifier(criterion='entropy',
                              random_state=1,
                              max_depth=None)

tree.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % tree.score(X_test, y_test))
```

```
Test Accuracy: 0.92
```

```
tree = DecisionTreeClassifier(criterion='entropy',
                              random_state=1,
                              max_depth=1)

tree.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % tree.score(X_test, y_test))
```

```
Test Accuracy: 0.58
```

Code 3

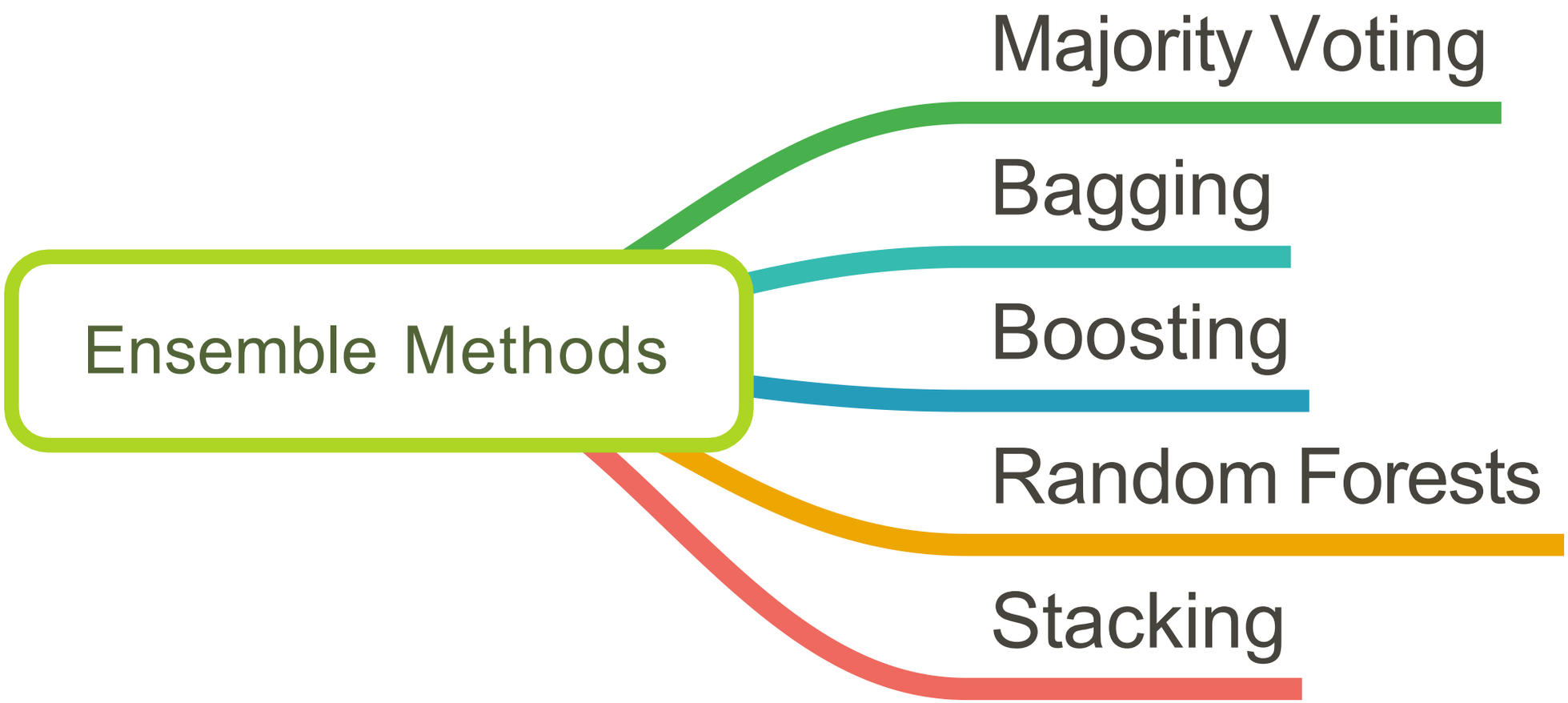
```
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.ensemble import AdaBoostClassifier

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])

tree = DecisionTreeClassifier(criterion='entropy',
                              random_state=1,
                              max_depth=1)


boost = AdaBoostClassifier(base_estimator=tree,
                           n_estimators=500,
                           algorithm='SAMME',
                           #n_jobs=1,
                           random_state=1)

boost.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
Train/Valid/Test sizes: 84 28 38
Test Accuracy: 0.97
```

```
boost.estimator_weights_
```

```
array([1.33318454, 2.19722458, 2.72852209, 2.40539213, 2.12261206,
       2.08078997, 1.64903181, 2.05956682, 1.54835435, 2.11511154,
       1.65838342, 1.95728572, 1.61295958, 1.55963465, 1.57714424,
       1.47449946, 1.47584541, 1.63354308, 1.47219613, 1.43615879,
```

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

1. Ensemble Methods -- Intro and Overview
2. Majority Voting
3. Bagging
4. Boosting
5. **Gradient Boosting**
6. Random Forests
7. Stacking

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 ...

Friedman, J. H. (1999). "Greedy Function Approximation: A Gradient Boosting Machine".

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.

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)

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

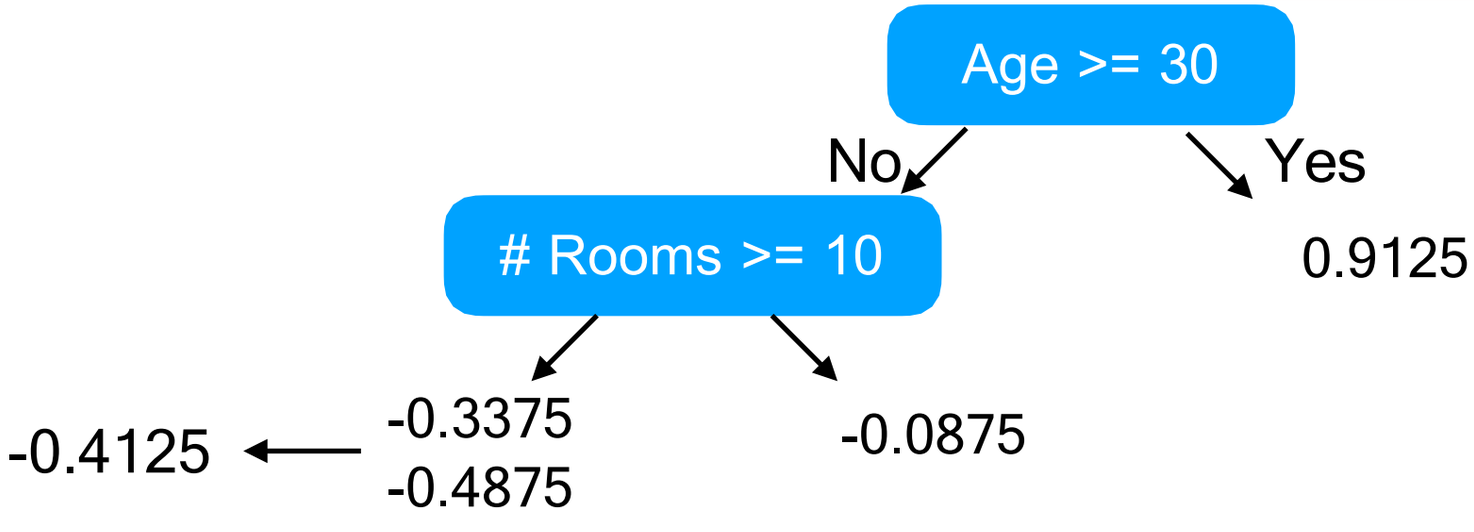
x1#	x2=City	x3=Age	y=Price	r1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

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

- **<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



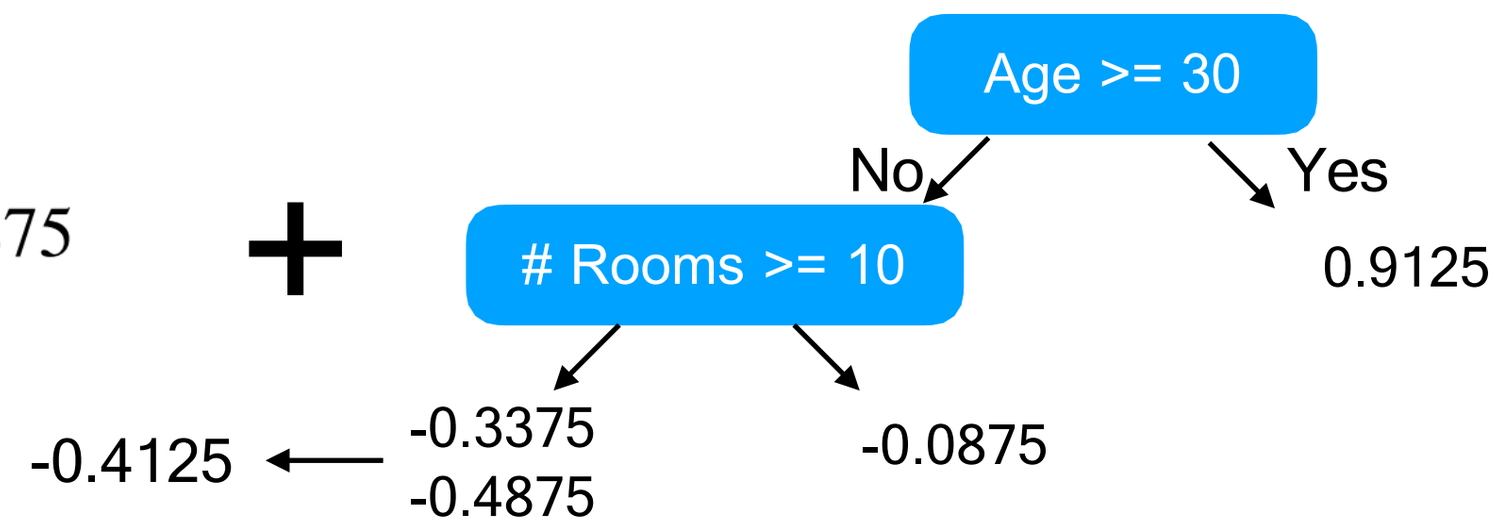
Gradient Boosting -- Conceptual Overview
--> A Regression-based Example

- **<u>Step 3:</u>** 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

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

+



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

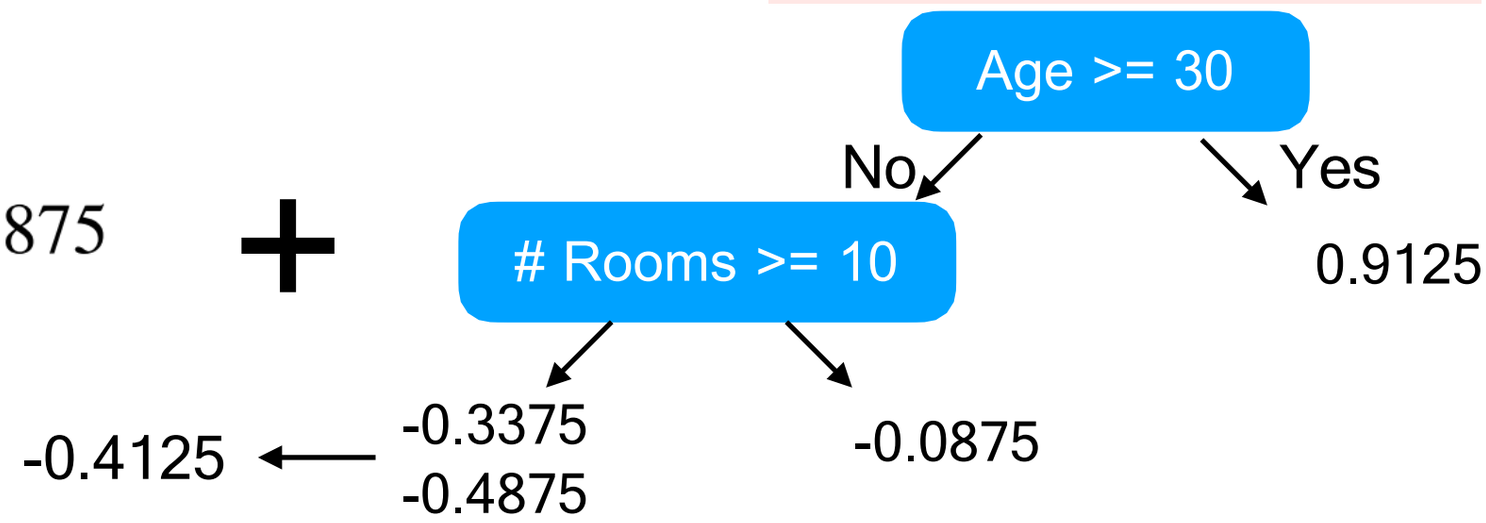
- **Step 3:** Combine tree from step 1 with trees from step 2

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

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

+



E.g., predict Lansing

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

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

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

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 $j = 1,\ldots,J_t$

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

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

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

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

E.g., Sum-squared error in regression

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

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

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

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

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

for $i = 1$ to n

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

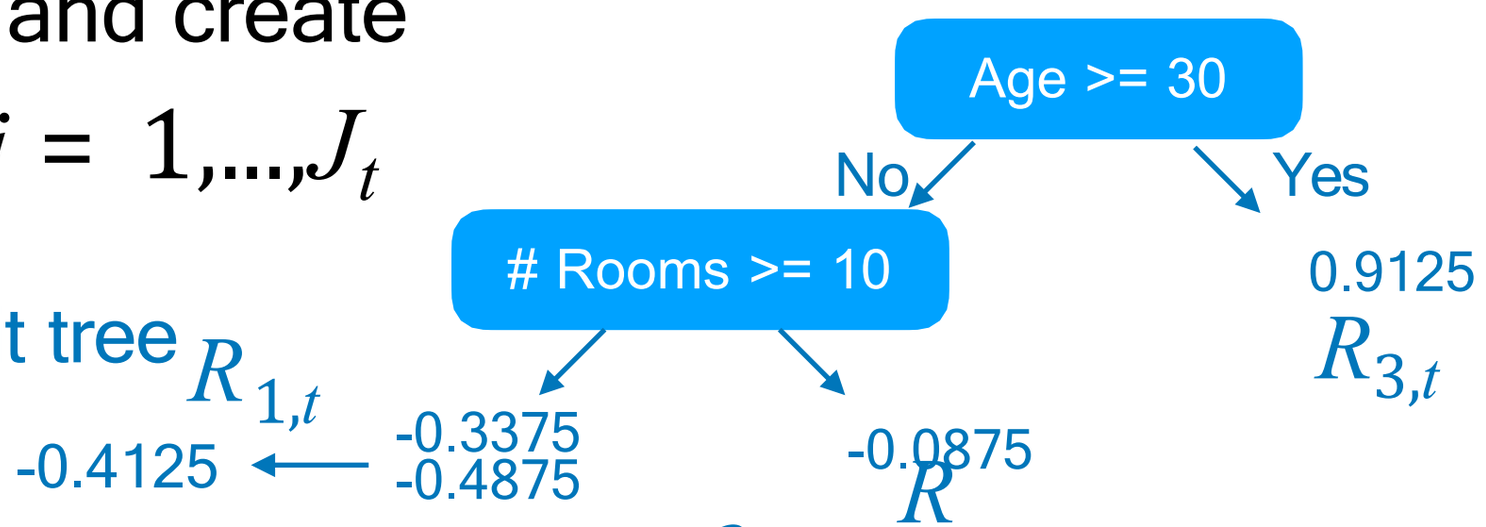
A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$ for $i = 1$ to n

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 $j = 1,...,J_t$

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 $_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,...,J_t$

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

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

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 $_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,...,J_t$

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

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

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

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.,

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

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

...

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

Gradient Boosting -- Algorithm Overview Discussion

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

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

$$+ \alpha \; y_{j,t=1}$$

...

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

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

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 International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM.

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 International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM.

Speed

We compared speed using only the training task without any test or metric output. We didn't count the time for IO. For the ranking tasks, since XGBoost and LightGBM implement different ranking objective functions, we used `regression` objective for speed benchmark, for the fair comparison.

The following table is the comparison of time cost:

Data	xgboost	xgboost_hist	LightGBM
Higgs	3794.34 s	165.575 s	**130.094 s**
Yahoo LTR	674.322 s	131.462 s	**76.229 s**
MS LTR	1251.27 s	98.386 s	**70.417 s**
Expo	1607.35 s	137.65 s	**62.607 s**
Allstate	2867.22 s	315.256 s	**148.231 s**

LightGBM ran faster than xgboost on all experiment data sets.

Accuracy

We computed all accuracy metrics only on the test data set.

Data	Metric	xgboost	xgboost_hist	LightGBM
Higgs	AUC	0.839593	0.845314	**0.845724**
Yahoo LTR	$NDCG_1$	0.719748	0.720049	**0.732981**
	$NDCG_3$	0.717813	0.722573	**0.735689**
	$NDCG_5$	0.737849	0.740899	**0.75352**
	$NDCG_{10}$	0.78089	0.782957	**0.793498**
MS LTR	$NDCG_1$	0.483956	0.485115	**0.517767**
	$NDCG_3$	0.467951	0.47313	**0.501063**
	$NDCG_5$	0.472476	0.476375	**0.504648**
	$NDCG_{10}$	0.492429	0.496553	**0.524252**
Expo	AUC	0.756713	0.776224	**0.776935**
Allstate	AUC	0.607201	**0.609465**	0.609072

https://lightgbm.readthedocs.io/en/latest/Experiments.html

2.2 Histogram-based Split Finding

Both XGBoost and LightGBM support **histogram-based** algorithm for split finding. As mentioned in XGBoost paper, the exact-greedy (brute-force) split find algorithm is time consuming: for current feature to search, need to sort feature values and iterate through. For faster training, histogram-based algorithm is used, which bucket continuous feature into discrete bins. This speeds up training and reduces memory usage.

LightGBM is using histogram-based algorithm. Related parameters are:

- `max_bin` : max number of bins that feature values will be bucketed in.
- `min_data_in_bin` : minimal number of data inside one bin.
- `bin_construct_sample_cnt` : number of data that sampled to construct histogram bins.

XGBoost has options to choose histogram-based algorithm, it is specified by `tree_method` with options:

- `auto` : (default) use heuristic to choose the fastest method.
- `exact` : exact greedy algorithm.
- `approx` : approximate greedy algorithm using quantile sketch and gradient histogram.
- `hist` : fast histogram optimized approximate greedy algorithm, with this option enabled, `max_bin` (default 256) could be tuned

https://bangdasun.github.io/2019/03/21/38-practical-comparison-xgboost-lightgbm/

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.

Code 1

```
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])
```

```
Train/Valid/Test sizes: 84 28 38
```

```
from sklearn.ensemble import GradientBoostingClassifier


boost = GradientBoostingClassifier(
    learning_rate=0.1,
    n_estimators=100,
    max_depth=8,
    random_state=1)

boost.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
Test Accuracy: 0.95
```

Code 2

```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier


boost = HistGradientBoostingClassifier(
    learning_rate=0.1,
    #n_estimators=100,
    max_depth=8,
    random_state=1)

boost.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
Test Accuracy: 0.97
```

Code 3

```
# https://xgboost.readthedocs.io/en/latest/build.html
```

```
#!pip install xgboost
```

```
import numpy as np
import xgboost as xgb


dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)


param = {
    'max_depth': 8,
    'eta': 0.1,  # learning rate
    'objective': 'multi:softprob',  # loss function for multiclass
    'num_class': 3}  # number of classes

boost = xgb.train(param, dtrain, num_boost_round=100)

y_pred = boost.predict(dtest)
y_labels = np.argmax(y_pred, axis=1)


print("Test Accuracy: %0.2f" % (y_labels == y_test).mean())
```

```
Test Accuracy: 0.97
```

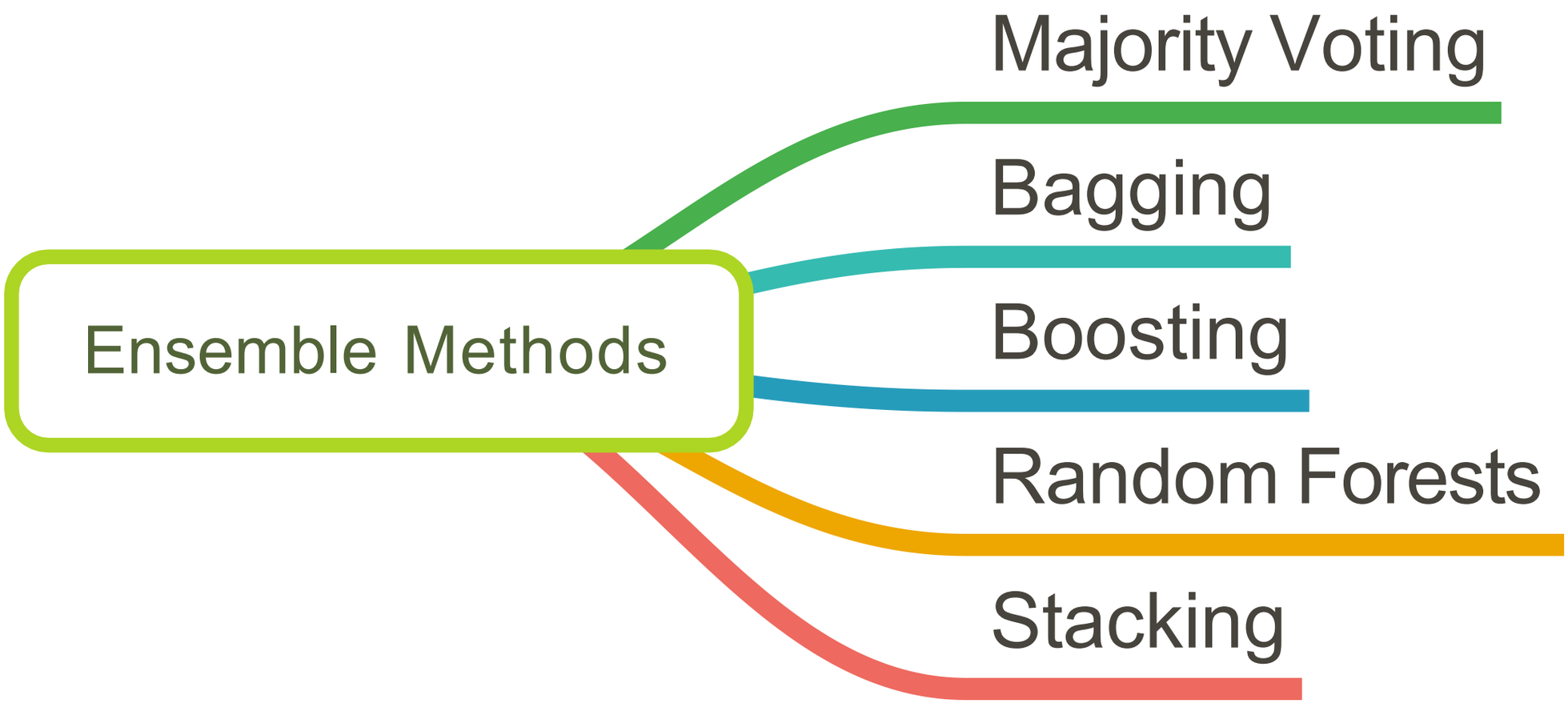
Code 4

```
# https://lightgbm.readthedocs.io/en/latest/Installation-Guide.html
# conda install -c conda-forge lightgbm


import lightgbm as lgb

boost = lgb.LGBMClassifier(n_estimators=100,
                           max_depth=8,
                           random_state=1,
                           learning_rate=0.1)

boost.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
Test Accuracy: 1.00
```

1. Ensemble Methods -- Intro and Overview
2. Majority Voting
3. Bagging
4. Boosting
5. Gradient Boosting
6. **Random Forests**
7. Stacking

Overview

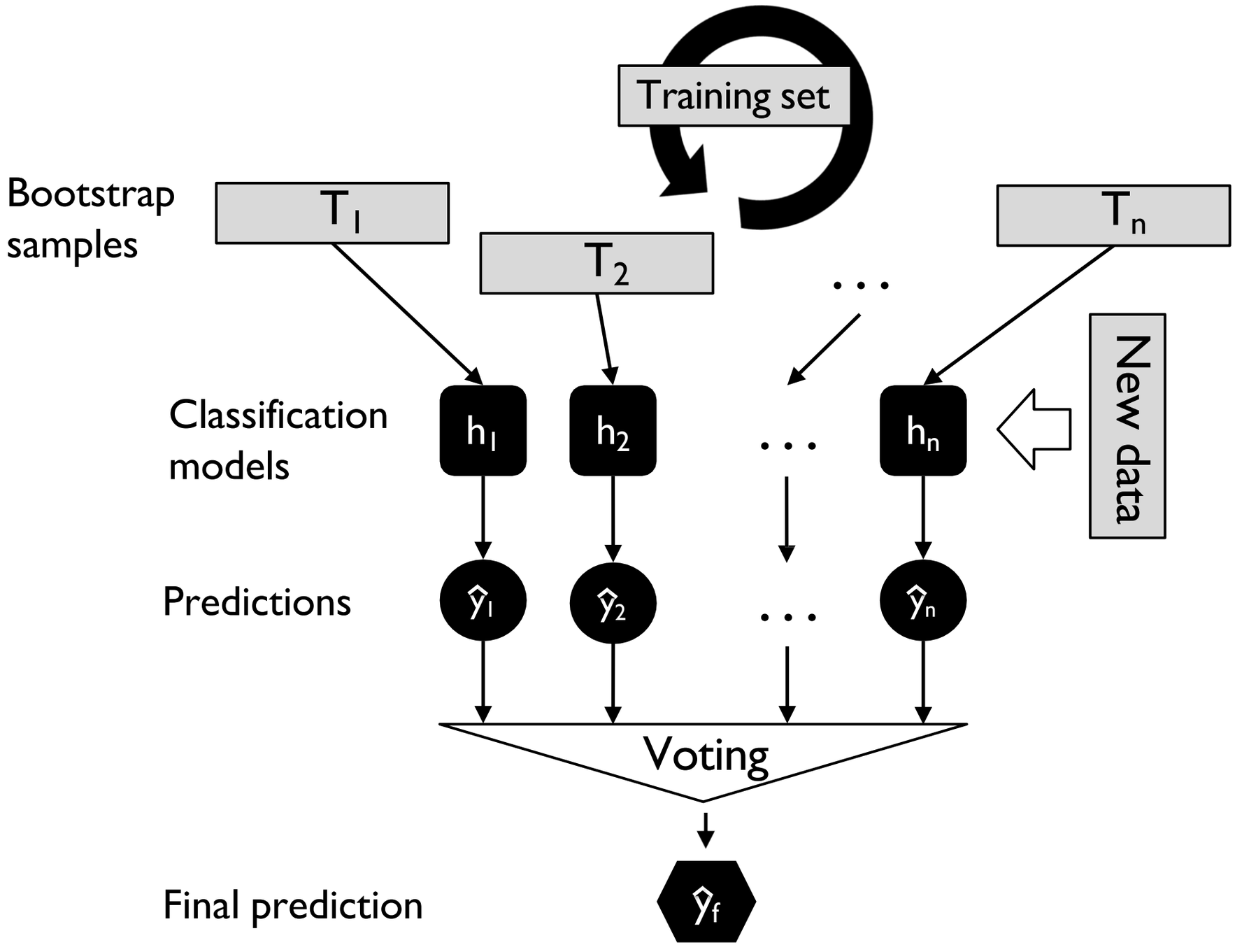

Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

Random Forests

Random Forests

= Bagging w. trees + random feature subsets

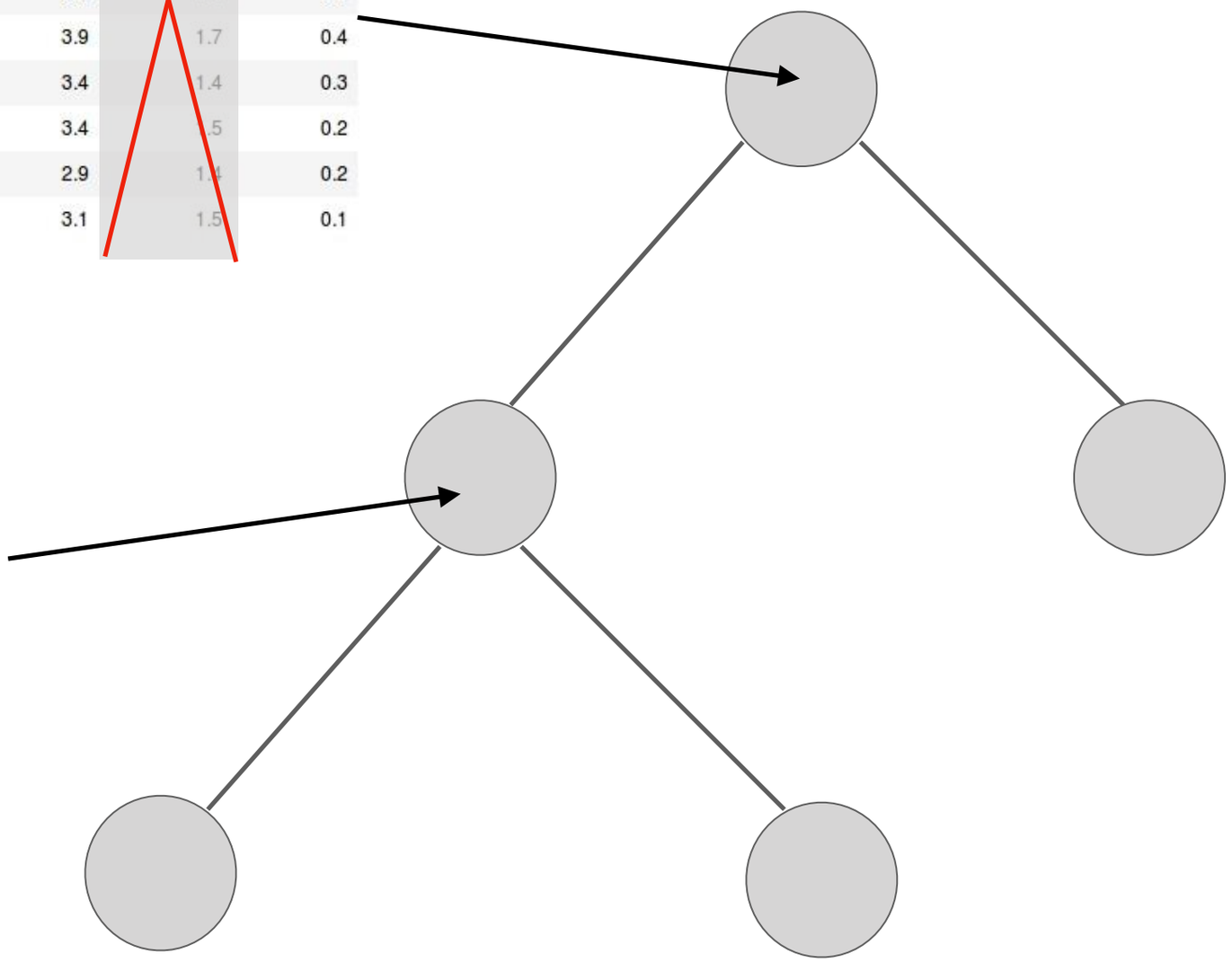
Bagging Classifier



Random Feature Subsets

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
2	4.7	3.2	1.3	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
9	4.9	3.1	1.5	0.1



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.

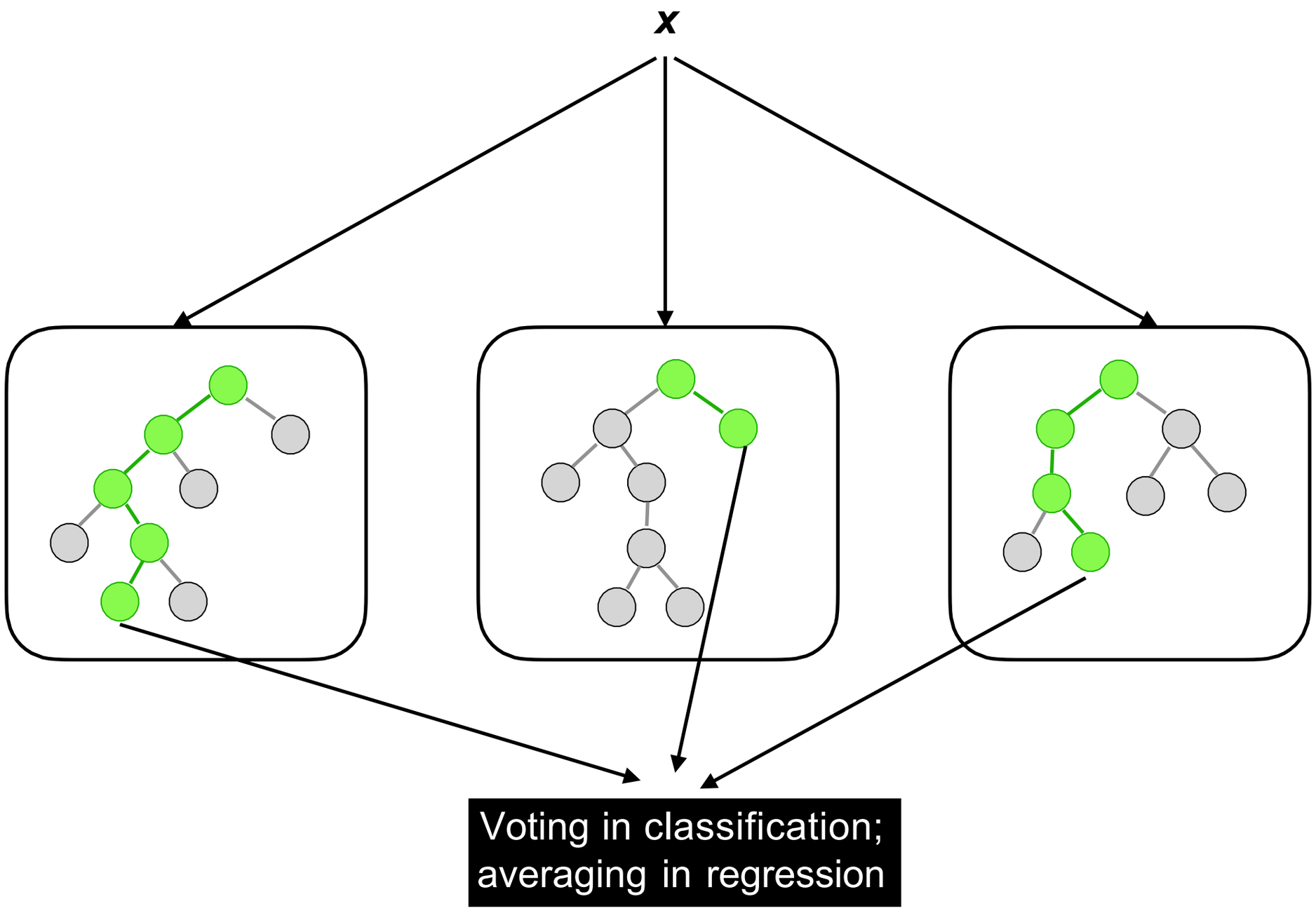
"Trademark" random forest:

"… random forest with random feature … andom, at each node, a small group of input variab …

- Breiman, Leo. "Random Forests" M … 2.

num features = $\log_2 m + 1$

where *m* is the number of input features


x
Voting in classification;
averaging in regression

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"

"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_i: 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

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

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

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

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

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

(Loose) Upper Bound for the Generalization Error

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

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

$\bar{\rho}$: 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) _____

Code

```
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn import datasets

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])
```

```
Train/Valid/Test sizes: 84 28 38
```

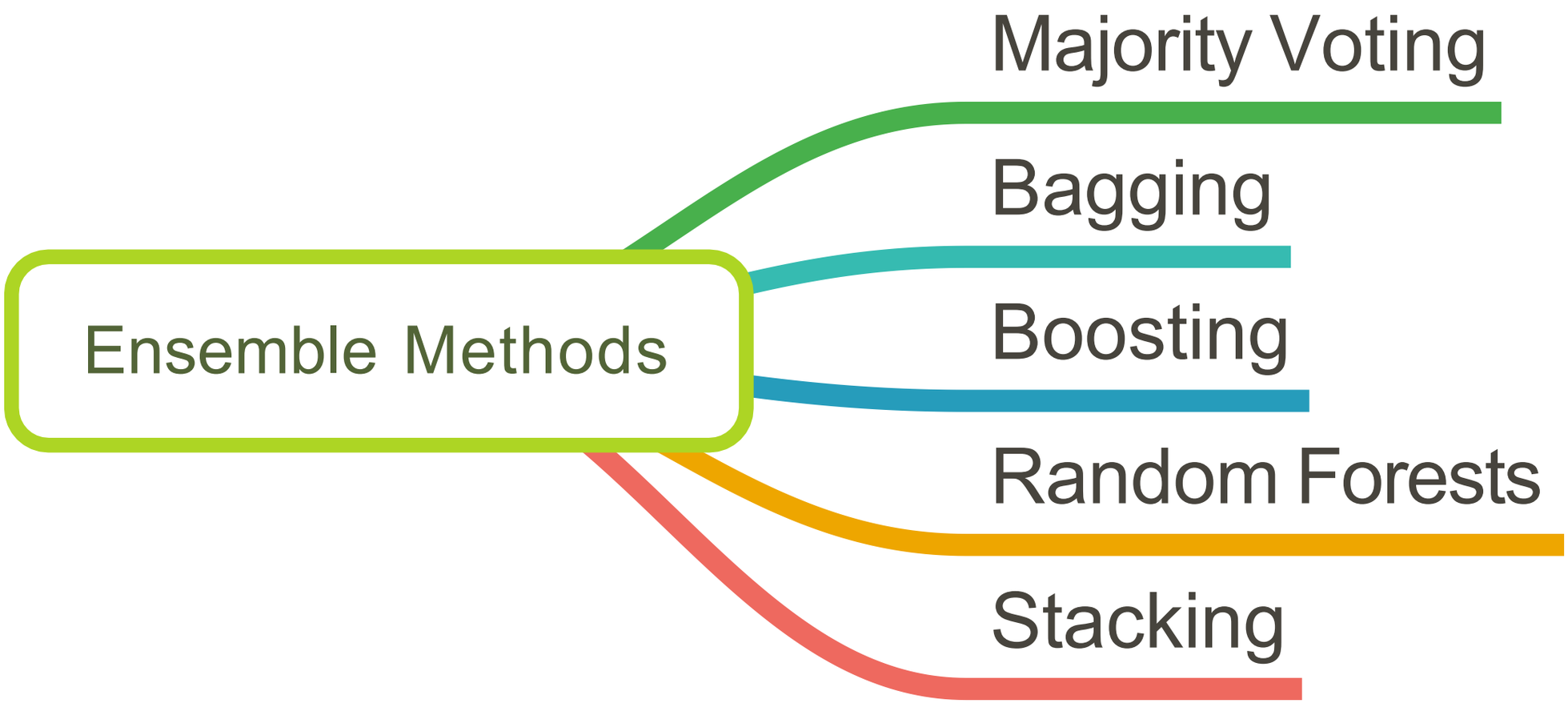
```
from sklearn.ensemble import RandomForestClassifier


forest = RandomForestClassifier(n_estimators=100,
                                random_state=1)

forest.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % forest.score(X_test, y_test))
```

```
Test Accuracy: 0.95
```

```
from sklearn.ensemble import ExtraTreesClassifier


forest = ExtraTreesClassifier(n_estimators=100,
                              random_state=1)

forest.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % forest.score(X_test, y_test))
```

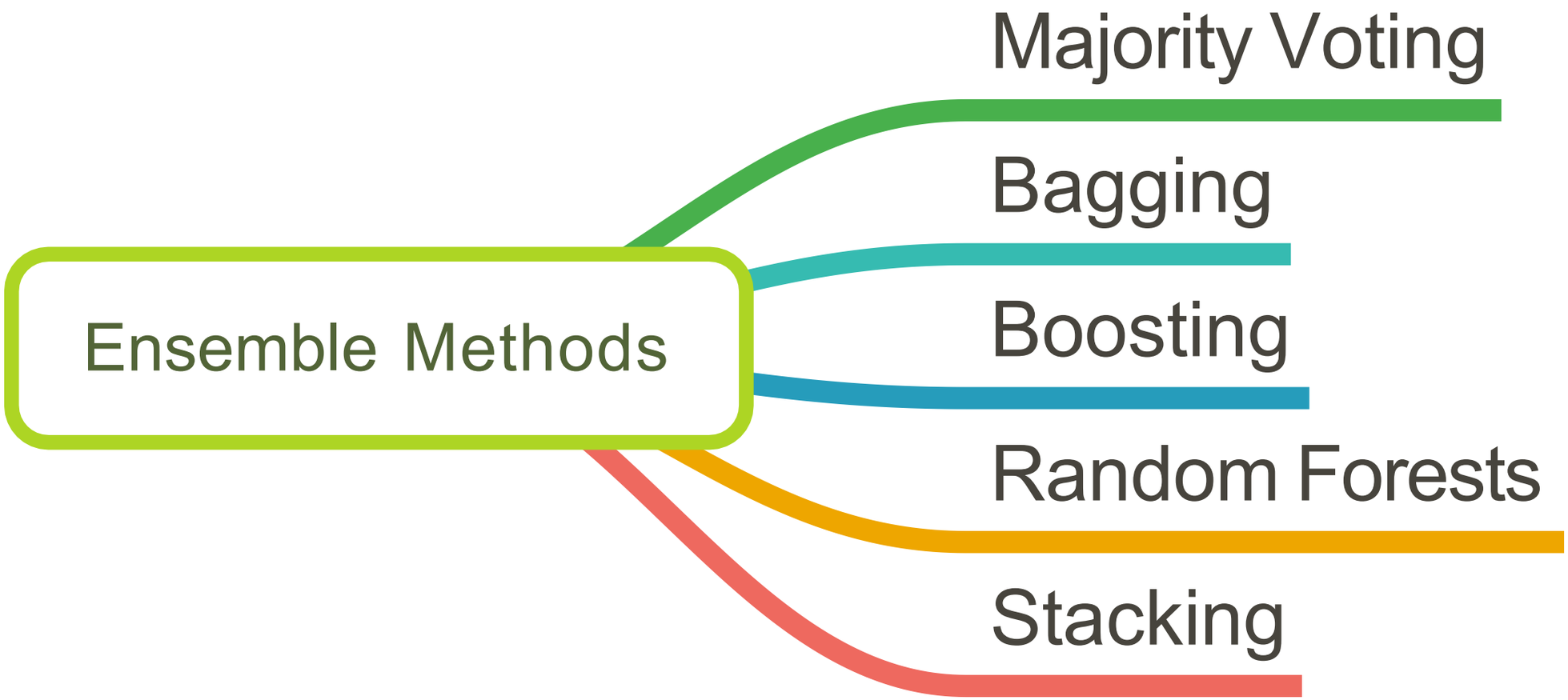
```
Test Accuracy: 0.95
```

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

1. Ensemble Methods -- Intro and Overview
2. Majority Voting
3. Bagging
4. Boosting
5. Gradient Boosting
6. Random Forests
7. **Stacking**

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.

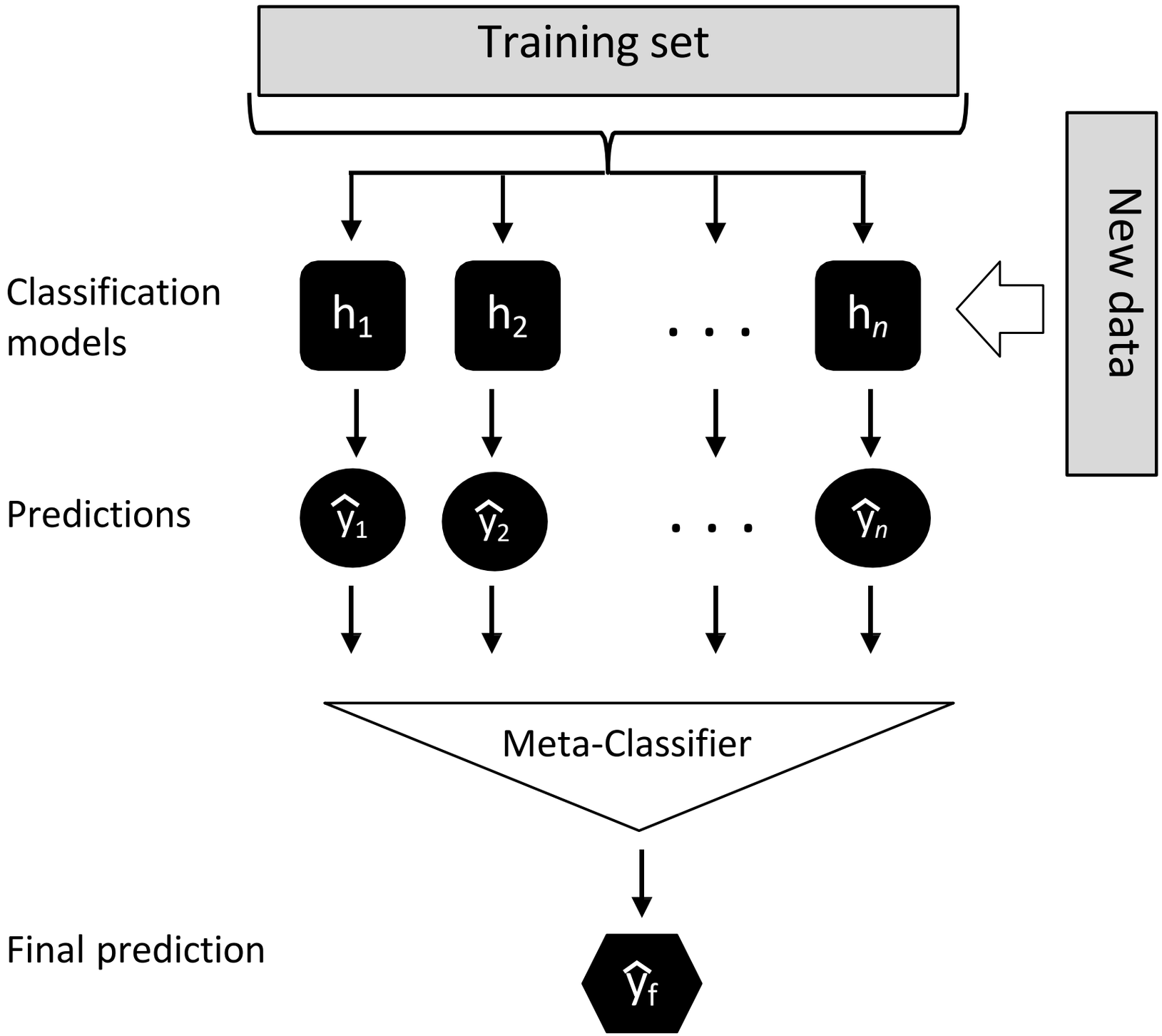
Algorithm 19.7 Stacking

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

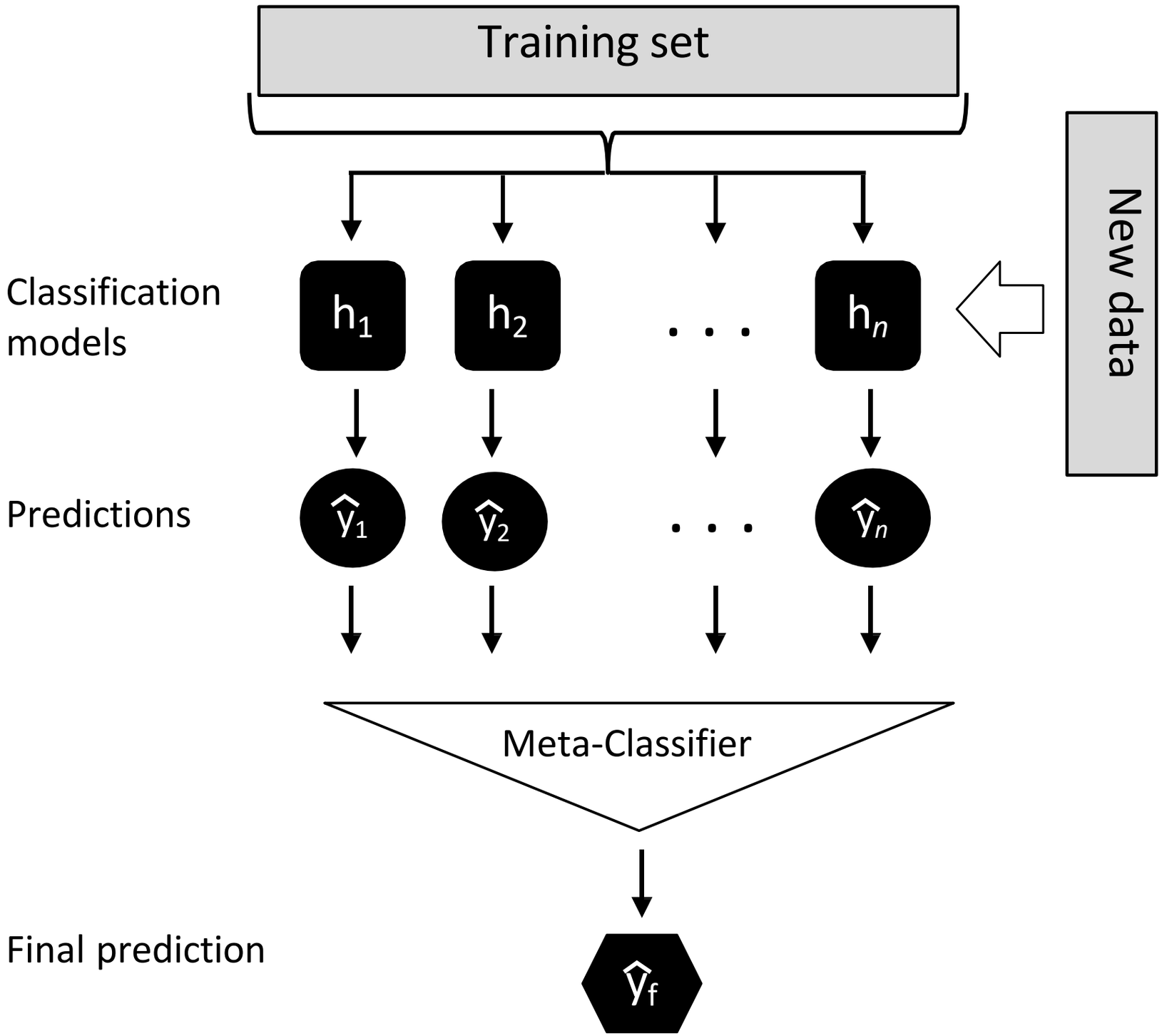
Step 1: Learn first-level classifiers
for $t \leftarrow 1$ to T **do**
    Learn a base classifier h_t based on $\mathcal{D}$
end for
Step 2: Construct new data sets from $\mathcal{D}$
for $i \leftarrow 1$ to m **do**
    Construct a new data set that contains $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i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

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

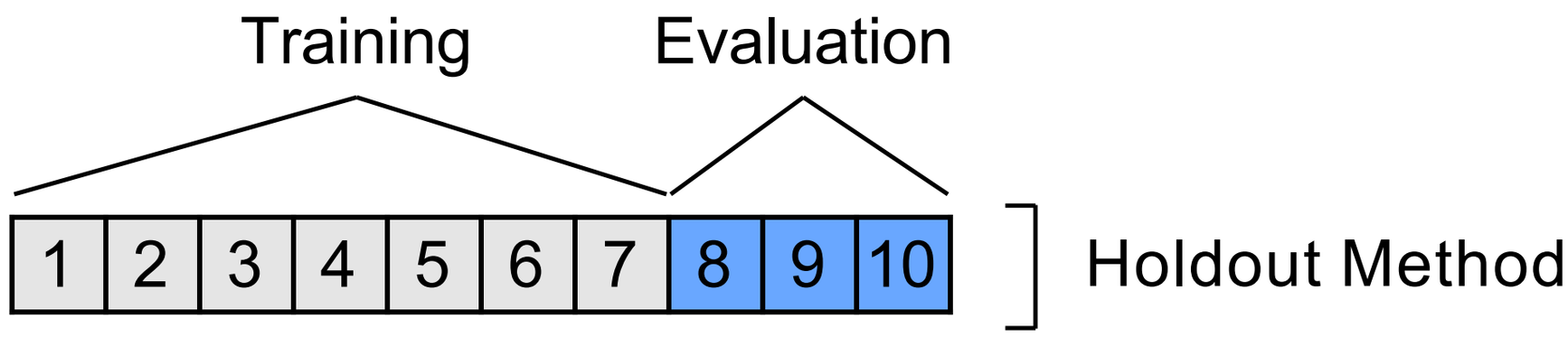
Stacking Algorithm

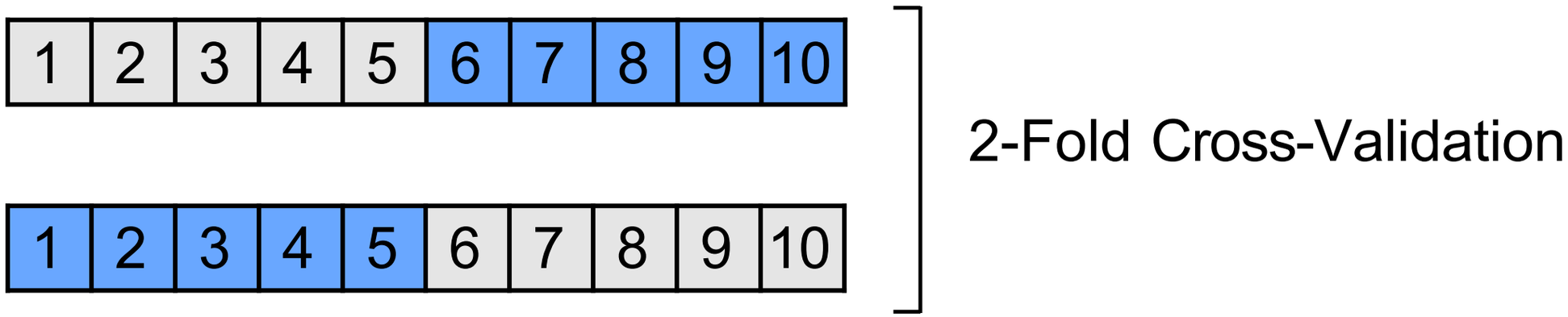


What is the problem with this stacking procedure?

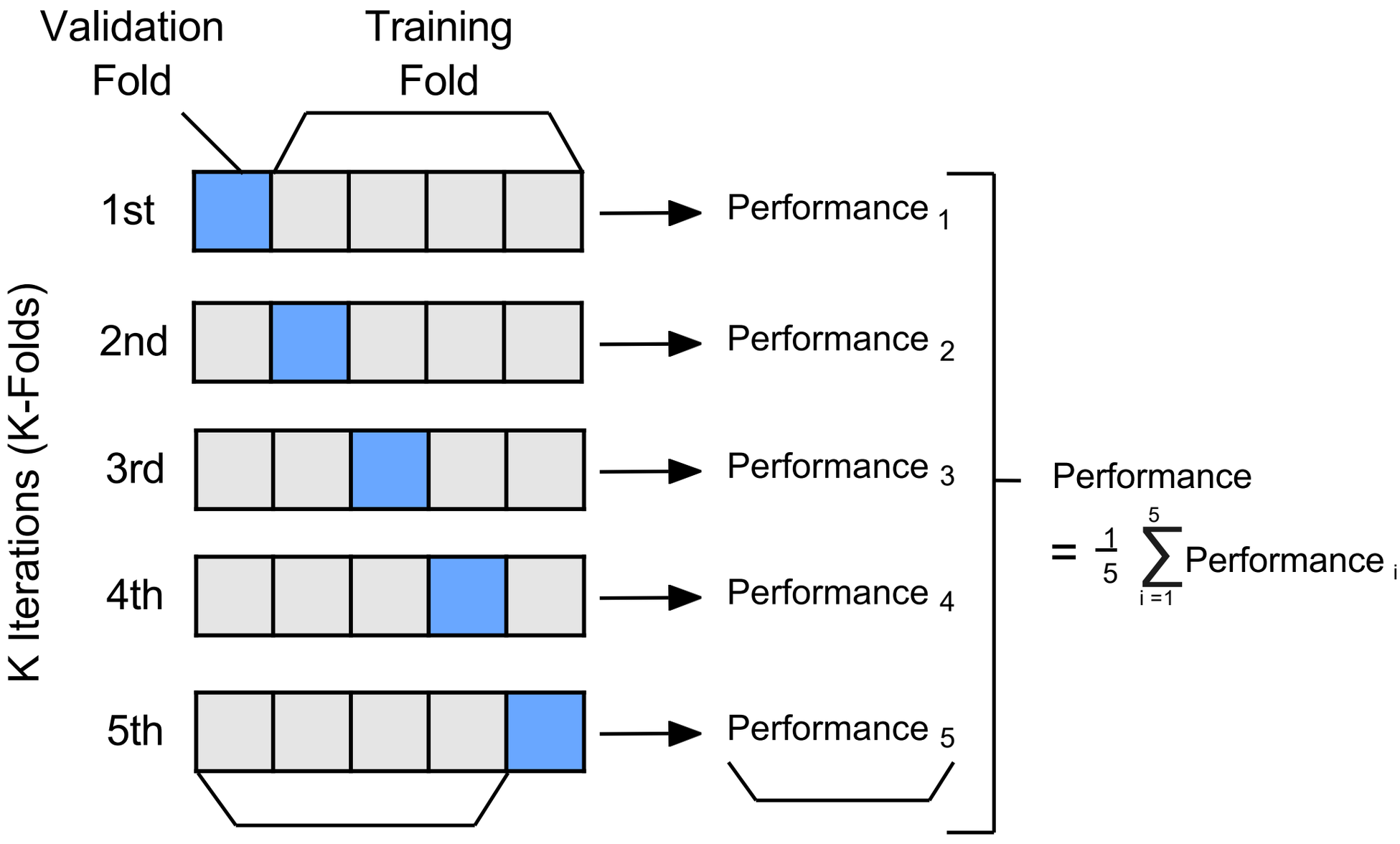


Cross-Validation





k-fold Cross-Validation



Stacking Algorithm with Cross-Validation

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

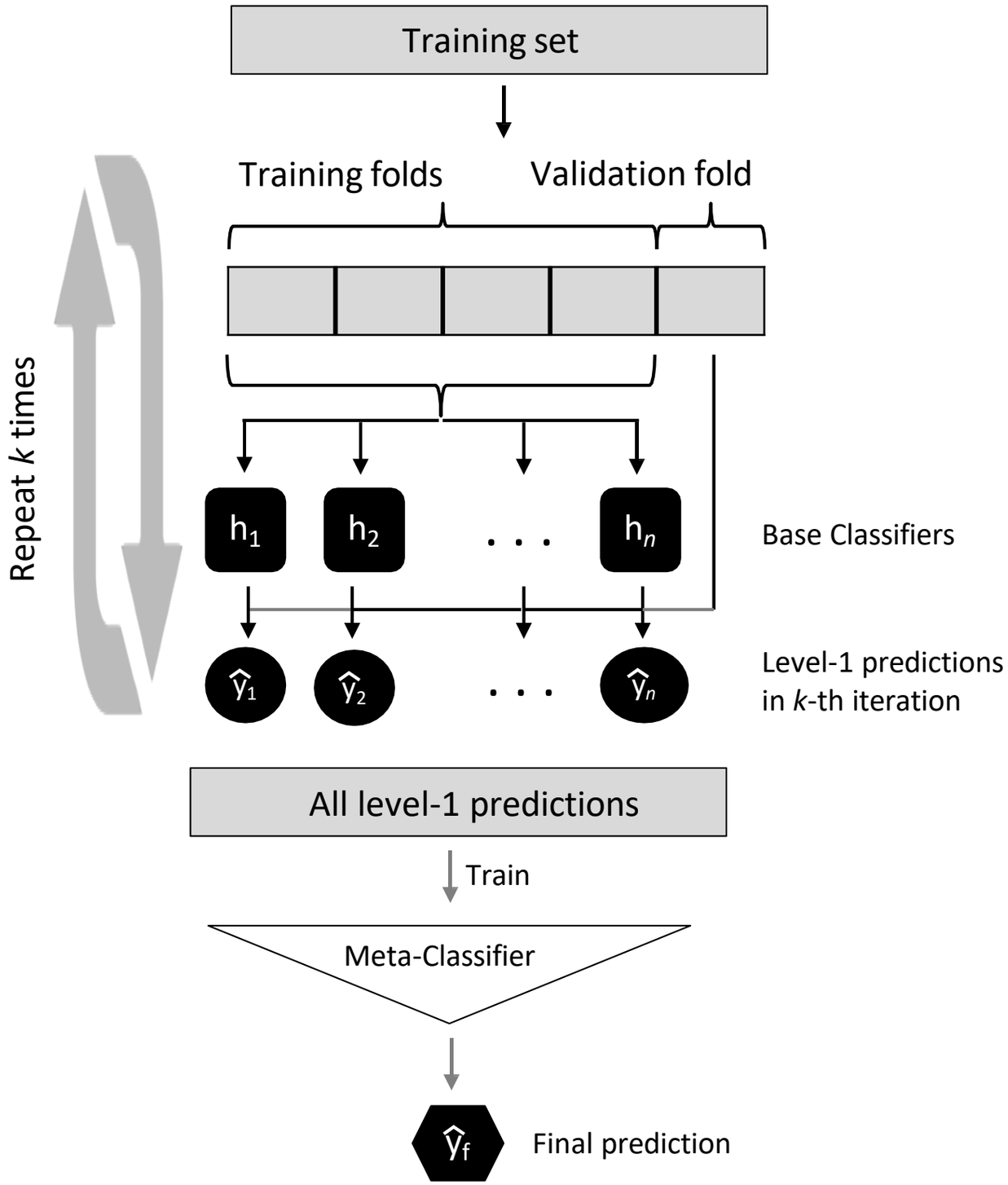
Algorithm 19.8 Stacking with K-fold Cross Validation

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

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



Code 1

```
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn import datasets

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])
```

```
Train/Valid/Test sizes: 84 28 38
```

Code 2

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from mlxtend.classifier import StackingClassifier


clf1 = KNeighborsClassifier(n_neighbors=5)
clf2 = RandomForestClassifier(random_state=1)
clf3 = HistGradientBoostingClassifier(random_state=1)
clf4 = AdaBoostClassifier(random_state=1)
clf5 = DecisionTreeClassifier(random_state=1,
                              max_depth=None)

lr = LogisticRegression(random_state=1)

sclf = StackingClassifier(classifiers=[clf1, clf2, clf3, clf4, clf5],
                          meta_classifier=lr)


sclf.fit(X_train, y_train)
print("Train Accuracy: %0.2f" % sclf.score(X_train, y_train))
print("Test Accuracy: %0.2f" % sclf.score(X_test, y_test))
```

```
Train Accuracy: 0.98
Test Accuracy: 0.95
```

Code 3

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from mlxtend.classifier import StackingCVClassifier


clf1 = KNeighborsClassifier(n_neighbors=5)
clf2 = RandomForestClassifier(random_state=1)
clf3 = HistGradientBoostingClassifier(random_state=1)
clf4 = AdaBoostClassifier(random_state=1)
clf5 = DecisionTreeClassifier(random_state=1,
                              max_depth=None)

lr = LogisticRegression(random_state=1)

sclf = StackingCVClassifier(classifiers=[clf1, clf2, clf3, clf4, clf5],
                            meta_classifier=lr,
                            cv=10,
                            random_state=1)


sclf.fit(X_train, y_train)
print("Train Accuracy: %0.2f" % sclf.score(X_train, y_train))
print("Test Accuracy: %0.2f" % sclf.score(X_test, y_test))
```

```
Train Accuracy: 0.96
Test Accuracy: 0.97
```

Code 4

New in version 0.22.

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from mlxtend.classifier import StackingCVClassifier


clf1 = KNeighborsClassifier(n_neighbors=5)
clf2 = RandomForestClassifier(random_state=1)
clf3 = HistGradientBoostingClassifier(random_state=1)
clf4 = AdaBoostClassifier(random_state=1)
clf5 = DecisionTreeClassifier(random_state=1,
                              max_depth=None)

lr = LogisticRegression(random_state=1)

sclf = StackingCVClassifier(classifiers=[clf1, clf2, clf3, clf4, clf5],
                            meta_classifier=lr,
                            cv=10,
                            random_state=1)


sclf.fit(X_train, y_train)
print("Train Accuracy: %0.2f" % sclf.score(X_train, y_train))
print("Test Accuracy: %0.2f" % sclf.score(X_test, y_test))
```

```
Train Accuracy: 0.96
Test Accuracy: 0.97
```

Code 5

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.ensemble import StackingClassifier
```

New in version 0.22.

```
clf1 = KNeighborsClassifier(n_neighbors=5)
clf2 = RandomForestClassifier(random_state=1)
clf3 = HistGradientBoostingClassifier(random_state=1)
clf4 = AdaBoostClassifier(random_state=1)
clf5 = DecisionTreeClassifier(random_state=1,
                              max_depth=None)

lr = LogisticRegression(random_state=1)

estimators = [('clf1', clf1),
              ('clf2', clf2),
              ('clf3', clf3),
              ('clf4', clf4),
              ('clf5', clf5)]

sclf = StackingClassifier(estimators=estimators,
                          final_estimator=lr,
                          cv=10)


sclf.fit(X_train, y_train)
print("Train Accuracy: %0.2f" % sclf.score(X_train, y_train))
print("Test Accuracy: %0.2f" % sclf.score(X_test, y_test))
```

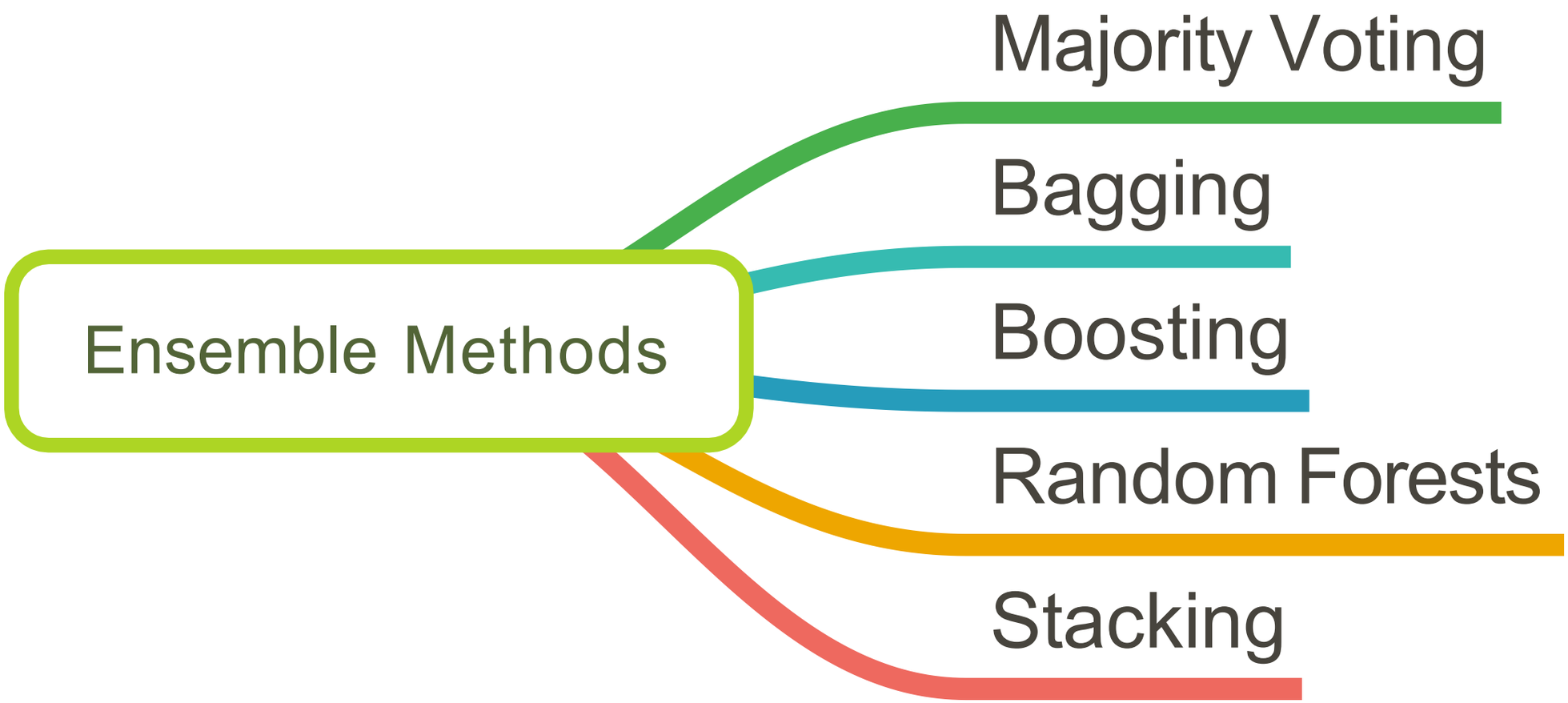
```
Train Accuracy: 0.98
Test Accuracy: 0.95
```

Code 6

```
# stack_method{'auto', 'predict_proba', 'decision_function', 'predict'}, default='auto'


from mlxtend.classifier import StackingCVClassifier


sclf = StackingCVClassifier(classifiers=[clf1, clf2, clf3, clf4, clf5],
                            meta_classifier=lr,
                            use_probas=True,
                            drop_proba_col='last',
                            #use_features_in_secondary=True,
                            cv=10,
                            random_state=1)


sclf.fit(X_train, y_train)
print("Train Accuracy: %0.2f" % sclf.score(X_train, y_train))
print("Test Accuracy: %0.2f" % sclf.score(X_test, y_test))
```

```
Train Accuracy: 0.98
Test Accuracy: 0.95
```

The End



http://stat.wisc.edu/~sraschka/teaching/stat451-fs2020/