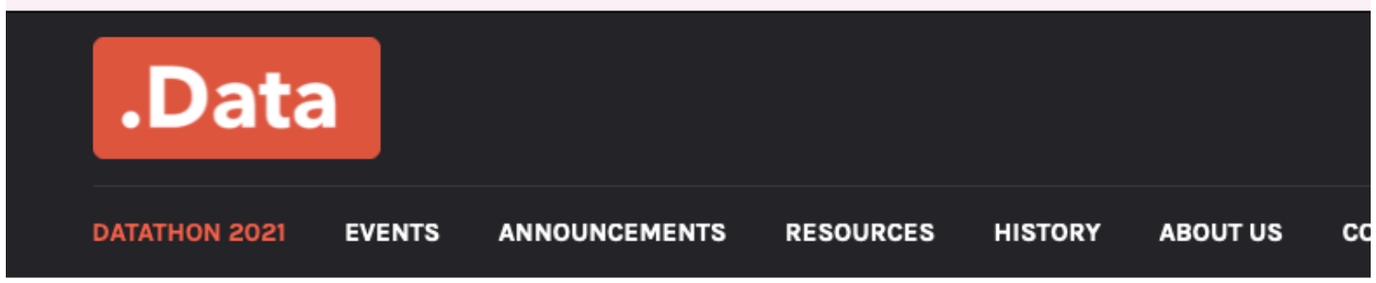


DATATHON 2021

http://dotdatascience.org/datathon-2021/

1. Food shall be provided throughout the duration of the hackathon
2. All participants shall receive 50$ worth Google Cloud credits.
3. All participants shall receive free Data Science Swag

Competition Schedule

October 4th – Team/Participant registration begins

October 14th – Team/Participant confirmations

October 15th – Teams for solo participants released

October 16th

9:00 am – In-person Checkin

10:00 am – Opening Ceremony

11:00 am – Hacking begins

October 17th

11:00 am – Project Submission

12:00 pm – Demo

2:00 pm – Winning teams are announced

External Events:- TBD

BLOG POST

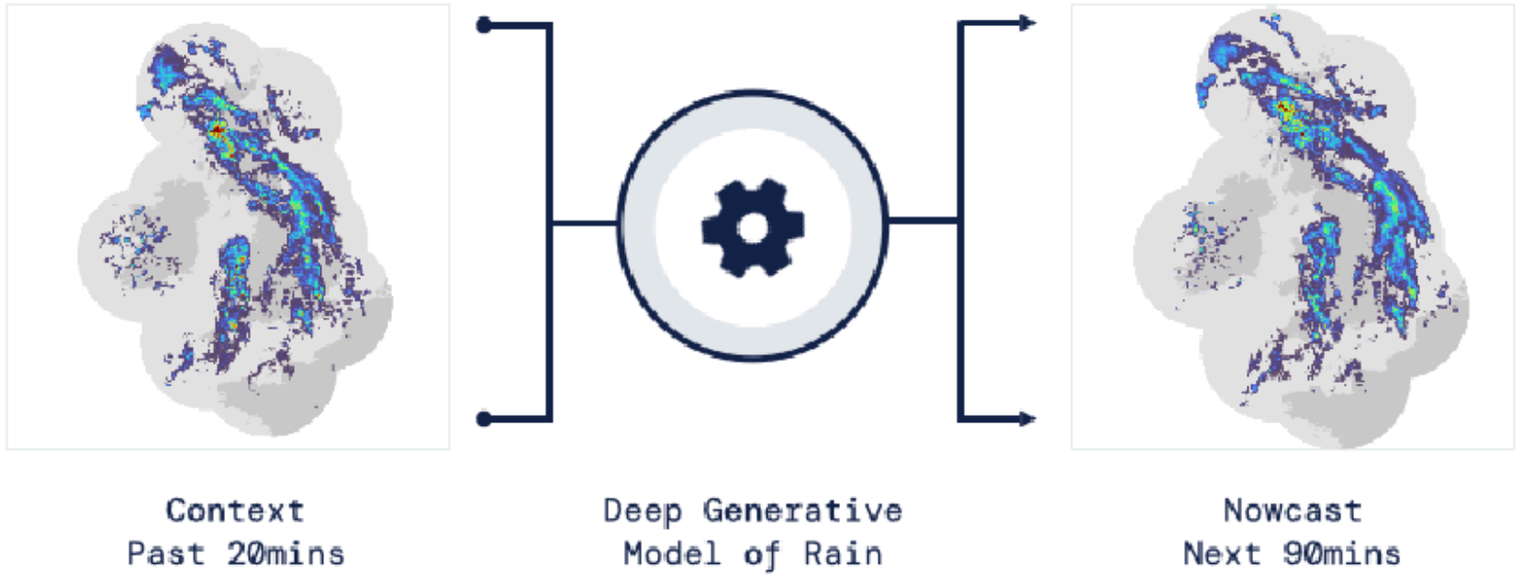
29 SEP 2021

Nowcasting the Next Hour of Rain

SHARE

AUTHORS

TNT The Nowcasting Team



https://deepmind.com/blog/article/nowcasting

https://www.nature.com/articles/s41586-021-03854-z

https://github.com/deepmind/deepmind-research/tree/master/nowcasting

Computer Science > Machine Learning

[Submitted on 5 Oct 2021]

Deep Neural Networks and Tabular Data: A Survey

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, Gjergji Kasneci

Heterogeneous tabular data are the most commonly used form of data and are essential for numerous critical and computationally demanding applications. On homogeneous data sets, deep neural networks have repeatedly shown excellent performance and have therefore been widely adopted. However, their application to modeling tabular data (inference or generation) remains highly challenging. This work provides an overview of state-of-the-art deep learning methods for tabular data. We start by categorizing them into three groups: data transformations, specialized architectures, and regularization models. We then provide a comprehensive overview of the main approaches in each group. A discussion of deep learning approaches for generating tabular data is complemented by strategies for explaining deep models on tabular data. Our primary contribution is to address the main research streams and existing methodologies in this area, while highlighting relevant challenges and open research questions. To the best of our knowledge, this is the first in-depth look at deep learning approaches for tabular data. This work can serve as a valuable starting point and guide for researchers and practitioners interested in deep learning with tabular data.

Subjects: **Machine Learning (cs.LG)**
Cite as: arXiv:2110.01889 **[cs.LG]**
(or arXiv:2110.01889v1 **[cs.LG]** for this version)

https://arxiv.org/abs/2110.01889

Computer Science > Machine Learning

[Submitted on 22 Jun 2021]

Revisiting Deep Learning Models for Tabular Data

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, Artem Babenko

Table 1: Results for single models. See supplementary for standard deviations. For each dataset, top results for baseline neural networks are in **bold**, top results for baseline neural networks and FT-Transformer are in **blue**, the overall top results are in **red**. "Top" means "the gap between this result and the result with the best mean score is not statistically significant". Datasets with heterogeneous features are underlined. FT-Transformer is introduced in section 3.6. Notation: "d" ~ "default", ↓ ~ lower is better, ↑ ~ higher is better.

	CA ↓	AD ↑	HE ↑	JA ↑	HI ↑	AL ↑	EP ↑	YE ↓	CO ↑	YA ↓	MI ↓
Baseline Neural Networks											
SNN	0.507	**0.816**	0.3728	0.718	0.721	0.954	0.8970	8.881	0.9465	0.769	0.7521
TabNet	0.513	0.796	0.3782	0.724	0.717	0.954	0.8902	9.032	0.9335	0.819	0.7565
GrowNet	0.500	0.793	–	–	0.724	–	**0.8977**	8.866	–	0.775	0.7549
DCN2	0.486	0.784	0.3853	0.714	0.720	0.955	**0.8975**	8.939	0.9491	0.766	0.7500
AutoInt	0.479	0.801	0.3722	0.716	**0.726**	0.945	0.8948	8.875	0.9312	0.795	0.7517
MLP	0.494	0.796	0.3832	0.719	0.721	0.954	0.8968	8.861	0.9499	0.776	0.7521
NODE	**0.464**	0.791	0.3593	**0.726**	0.724	0.918	0.8958	**8.774**	0.9436	**0.762**	**0.7474**
ResNet	0.487	**0.816**	**0.3960**	**0.727**	**0.727**	**0.963**	0.8971	8.845	**0.9560**	0.766	0.7493
FT-Transformer											
FT-Transformer$_d$	0.470	0.799	0.3812	0.725	0.723	0.953	0.8959	8.869	0.9617	**0.758**	**0.7475**
FT-Transformer	**0.464**	0.807	0.3913	**0.731**	**0.728**	0.960	**0.8982**	8.820	**0.9641**	**0.758**	**0.7469**
GBDT											
CatBoost$_d$	**0.430**	0.797	0.3814	0.721	0.724	0.946	0.8882	8.913	0.9076	0.751	0.7454
CatBoost	**0.431**	0.791	0.3853	0.723	0.725	–	0.8880	8.877	0.9658	0.743	0.7429
XGBoost$_d$	0.463	0.775	0.3502	0.721	0.705	0.925	0.8803	9.446	0.9640	0.773	0.7719
XGBoost	0.433	0.796	0.3755	0.724	0.725	–	0.8857	8.947	**0.9695**	**0.736**	**0.7424**

Table 2: Results for ensembles. Notation follows Table 1, but top results are now defined by the mean score. Standard deviations are reported in supplementary.

https://arxiv.org/abs/2106.11959

MAKING SENSE OF BIG DATA

pytorch-widedeep, deep learning for tabular data IV: Deep Learning vs LightGBM

A thorough comparison between DL algorithms and LightGBM for tabular data for classification and regression problems

Javier Rodriguez Zaurin Jun 13 · 30 min read

Here we go with yet another post in the series. This is the fourth of the series. The previous three posts, and the original version of this post are hosted in my own blog, just in case.

https://towardsdatascience.com/pytorch-widedeep-deep-learning-for-tabular-data-iv-deep-learning-vs-lightgbm-cadcbf571eaf

Lecture 07

Ensemble Methods Part 1/3

STAT 451: Machine Learning, Fall 2021

Sebastian Raschka

Part 1: Introduction

- L01 - Course overview, introduction to machine learning
- L02 - Introduction to Supervised Learning and k-Nearest Neighbors Classifiers

Part 2: Computational foundations

- L03 - Using Python
- L04 - Introduction to Python's scientific computing stack
- L05 - Data preprocessing and machine learning with scikit-learn

Part 3: Tree-based methods

- L06 - Decision trees
- L07 - Ensemble methods

Part 4: Model evaluation

- Midterm exam
- L08 - Model evaluation 1 – overfitting
- L09 - Model evaluation 2 – confidence intervals
- L10 - Model evaluation 3 – cross-validation and model selection
- L11 - Model evaluation 4 – algorithm selection
- L12 - Model evaluation 5 – evaluation and performance metrics

Part 5: Dimensionality reduction and unsupervised learning

7.1 Ensemble Methods -- Intro and Overview

7.2 Majority Voting

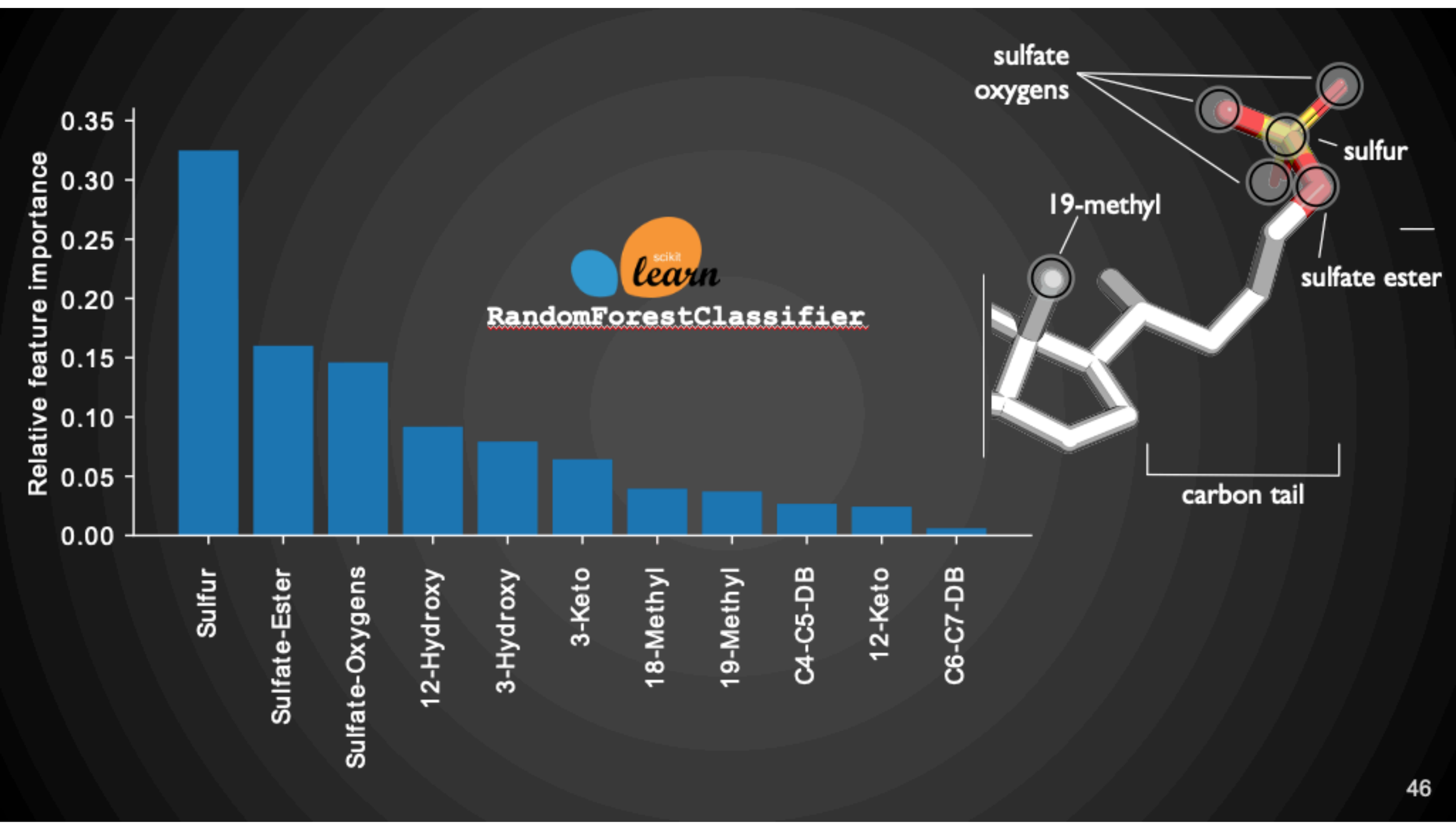
7.3 Bagging

7.4 Boosting

7.5 Gradient Boosting

7.6 Random Forests

7.7 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

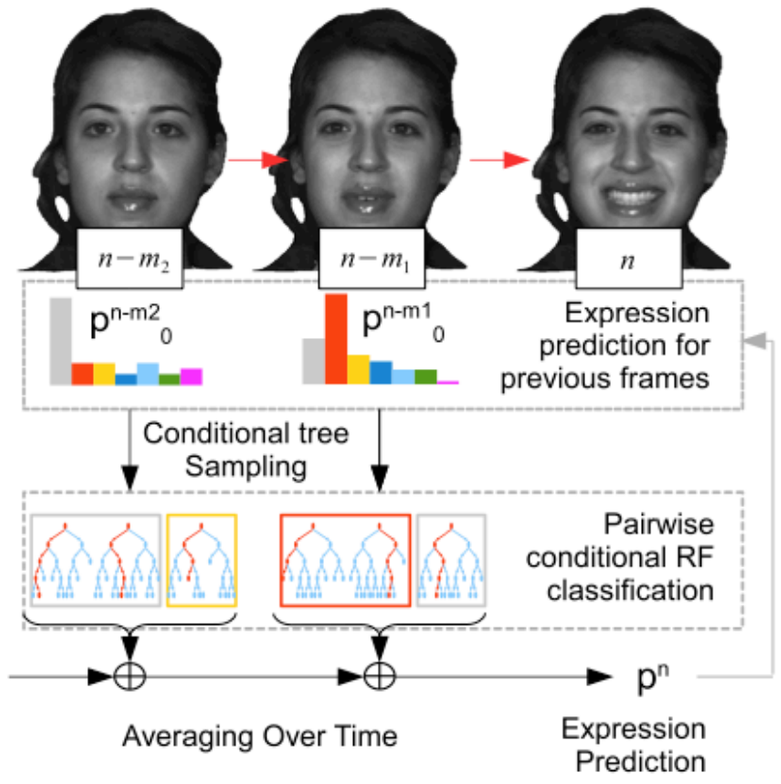
Pairwise Conditional Random Forests for Facial Expression Recognition

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."

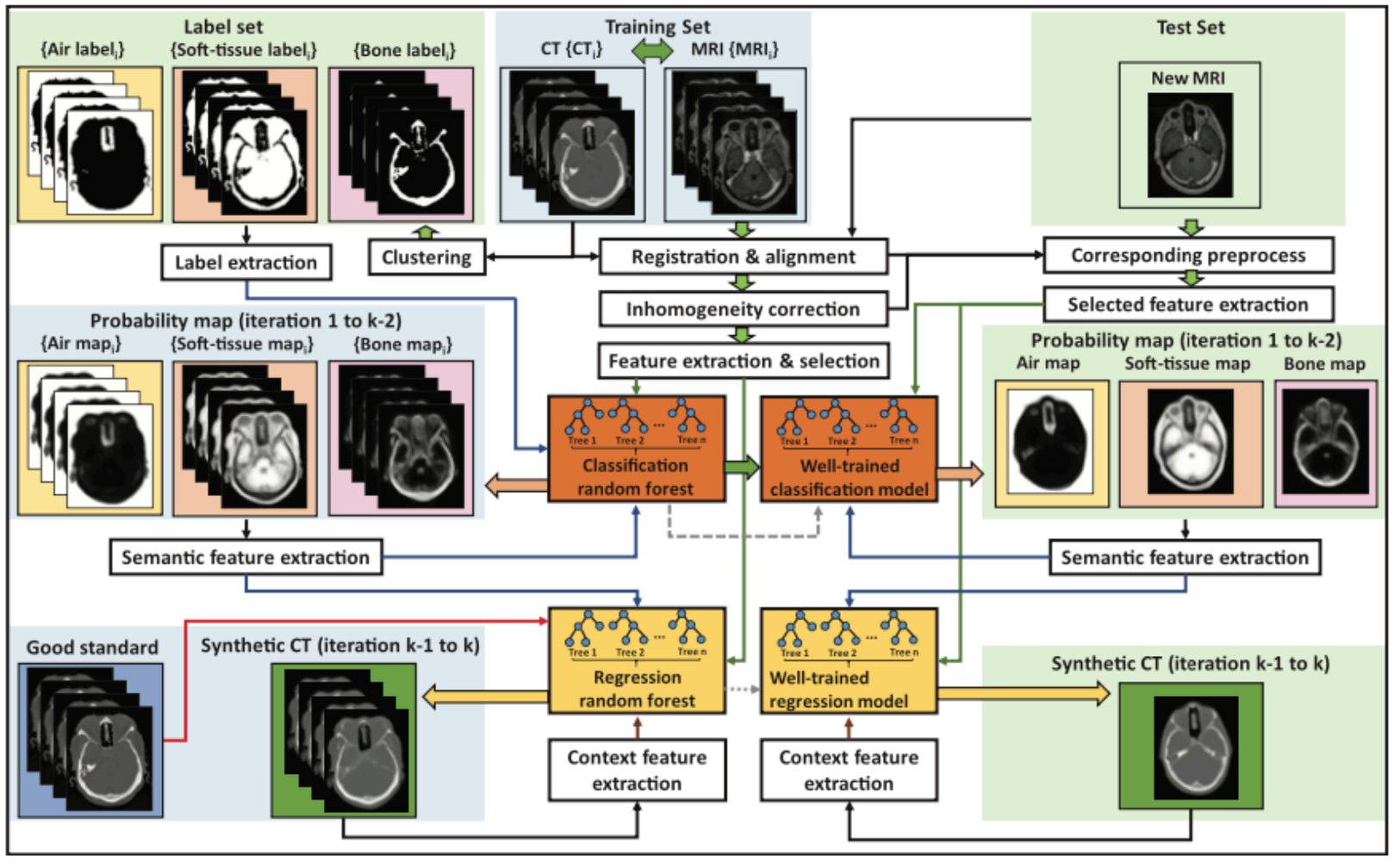


Figure 1. Schematic flow chart of the proposed algorithm for MRI-based sCT generation. The left part of this figure shows the training stage of our proposed method, which consists of classification random forest training, semantic feature extraction, context feature extraction, and regression random 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

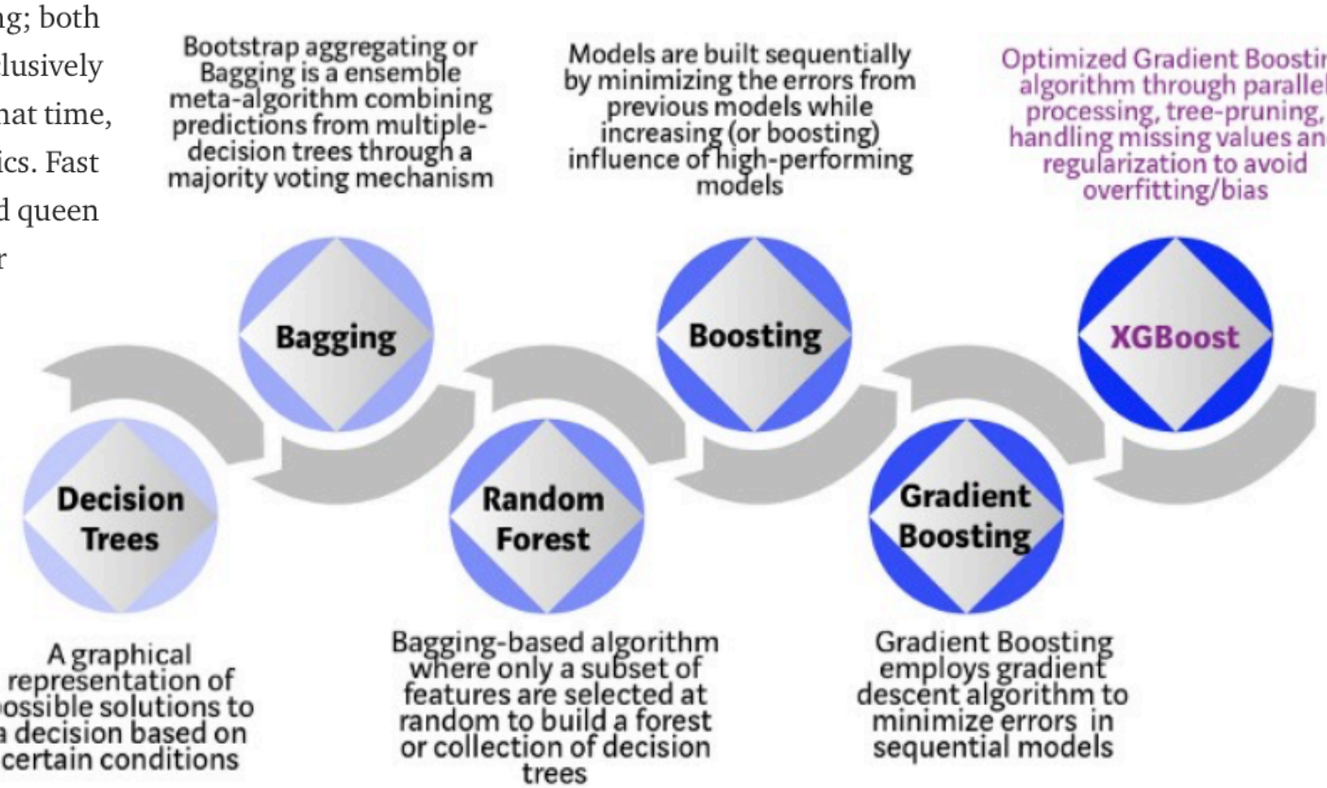
XGBoost Algorithm: Long May She Reign!

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)

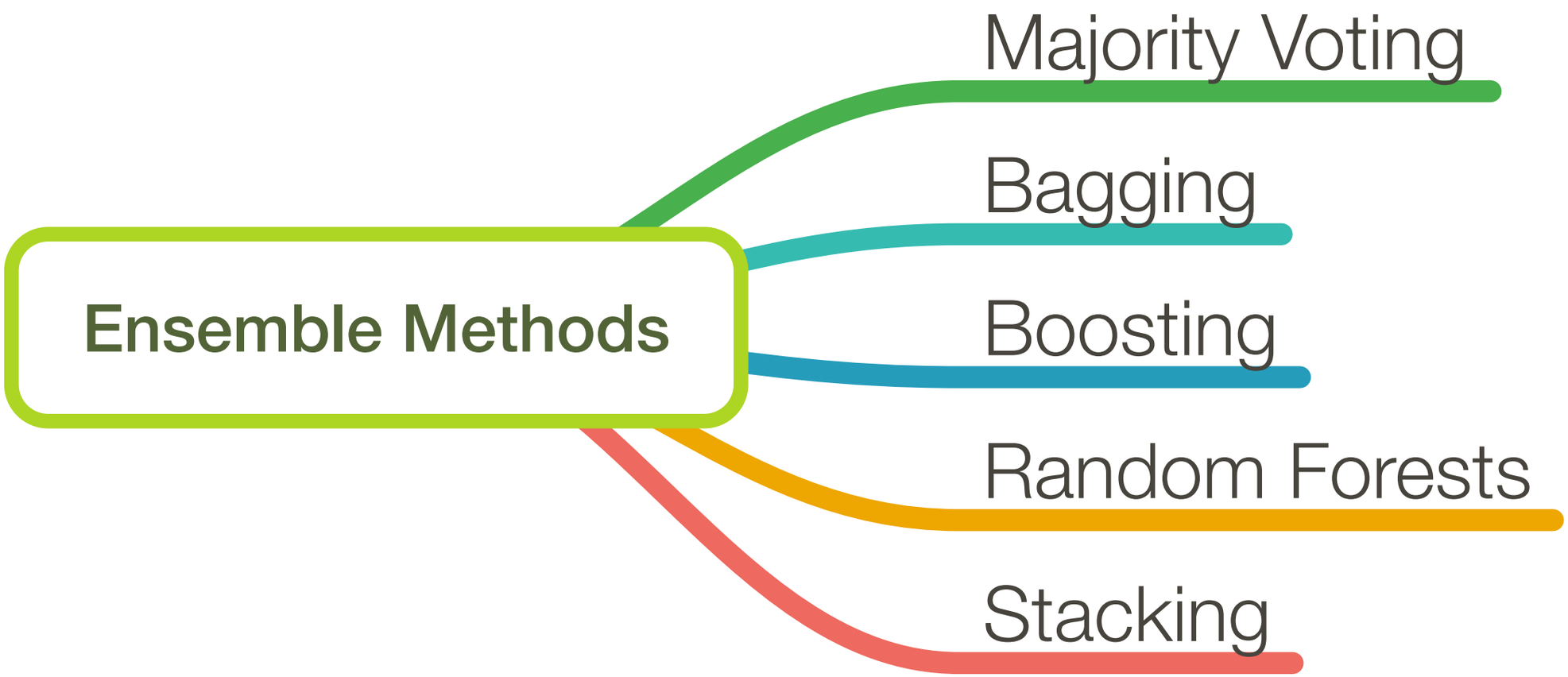
I remember thinking myself, “I got this!”. I knew regression modeling; both linear and logistic regression. My boss was right. In my tenure, I exclusively built regression-based statistical models. I wasn’t alone. In fact, at that time, regression modeling was the undisputed queen of predictive analytics. 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



7.1 Ensemble Methods -- Intro and Overview

7.2 Majority Voting

7.3 Bagging

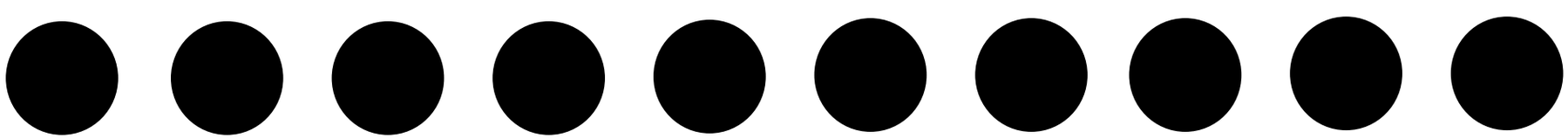
7.4 Boosting

7.5 Gradient Boosting

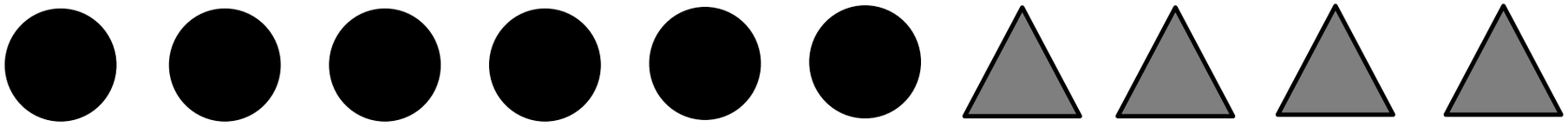
7.6 Random Forests

7.7 Stacking

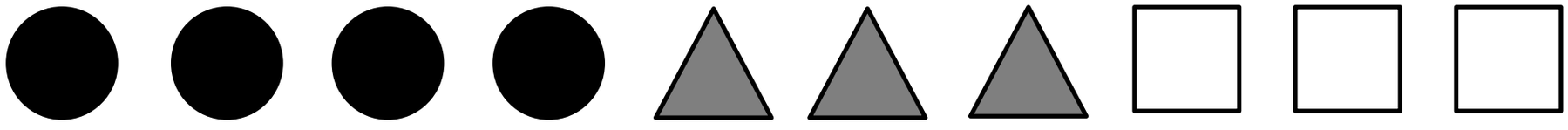
Majority Voting



Unanimity

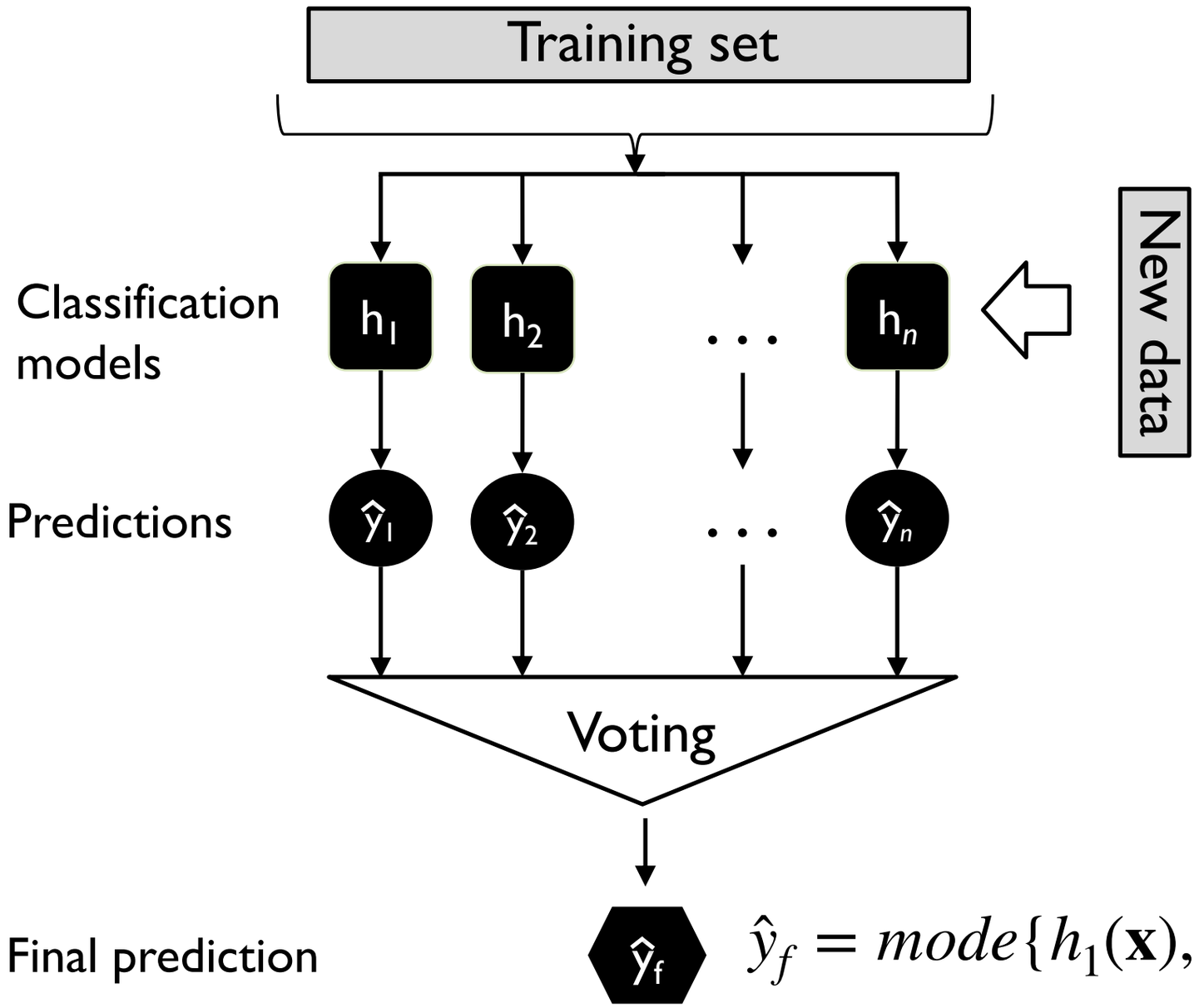


Majority



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

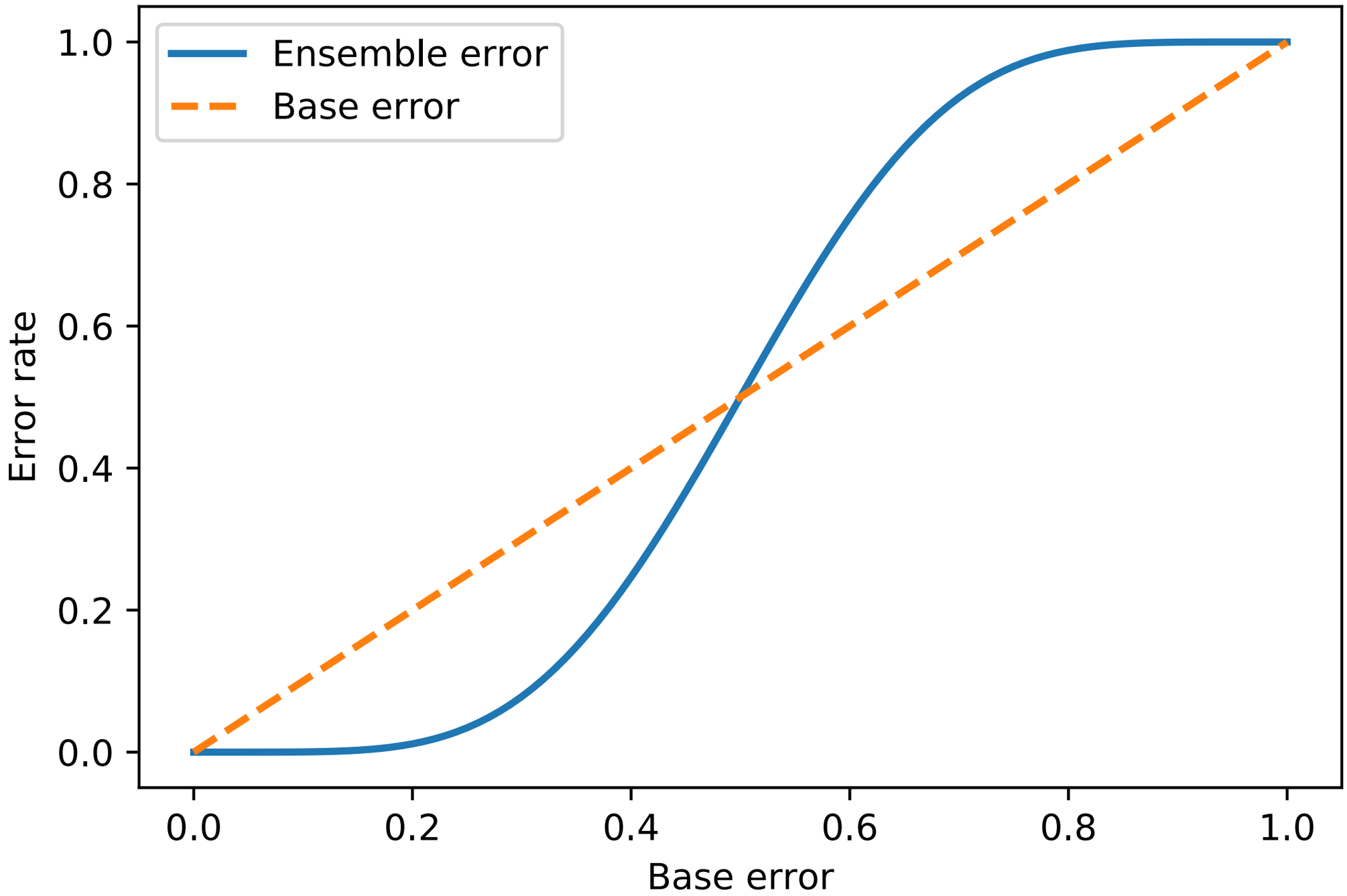
$$P(k) = \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k} \qquad k > \lfloor n/2 \rfloor$$

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

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

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

"Soft" Voting

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

Binary classification example

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

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

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

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

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

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

data = datasets.load_breast_cancer()
X, y = data.data, data.target

X_temp, X_test, y_temp, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=123, stratify=y)

X_train, X_valid, y_train, y_valid = \
    train_test_split(X_temp, y_temp, test_size=0.2, random_state=123, stratify=y_temp)

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

```
Train/Valid/Test sizes: 318 80 171
```

```
clf1 = DecisionTreeClassifier(random_state=1, max_depth=None)
clf2 = DecisionTreeClassifier(random_state=1, max_depth=1)
clf3 = DecisionTreeClassifier(random_state=1, max_depth=2)
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_valid, y_valid), label))

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

```
Validation Accuracy: 0.91 [Classifier 1]
Validation Accuracy: 0.89 [Classifier 2]
Validation Accuracy: 0.91 [Classifier 3]
Validation Accuracy: 0.91 [Ensemble]
Test Accuracy: 0.93
```

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

data = datasets.load_breast_cancer()
X, y = data.data, data.target

X_temp, X_test, y_temp, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=123, stratify=y)

X_train, X_valid, y_train, y_valid = \
    train_test_split(X_temp, y_temp, test_size=0.2, random_state=123, stratify=y_temp)

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

```
Train/Valid/Test sizes: 318 80 171
```

Now also available as:
sklearn.ensemble.VotingClassifier

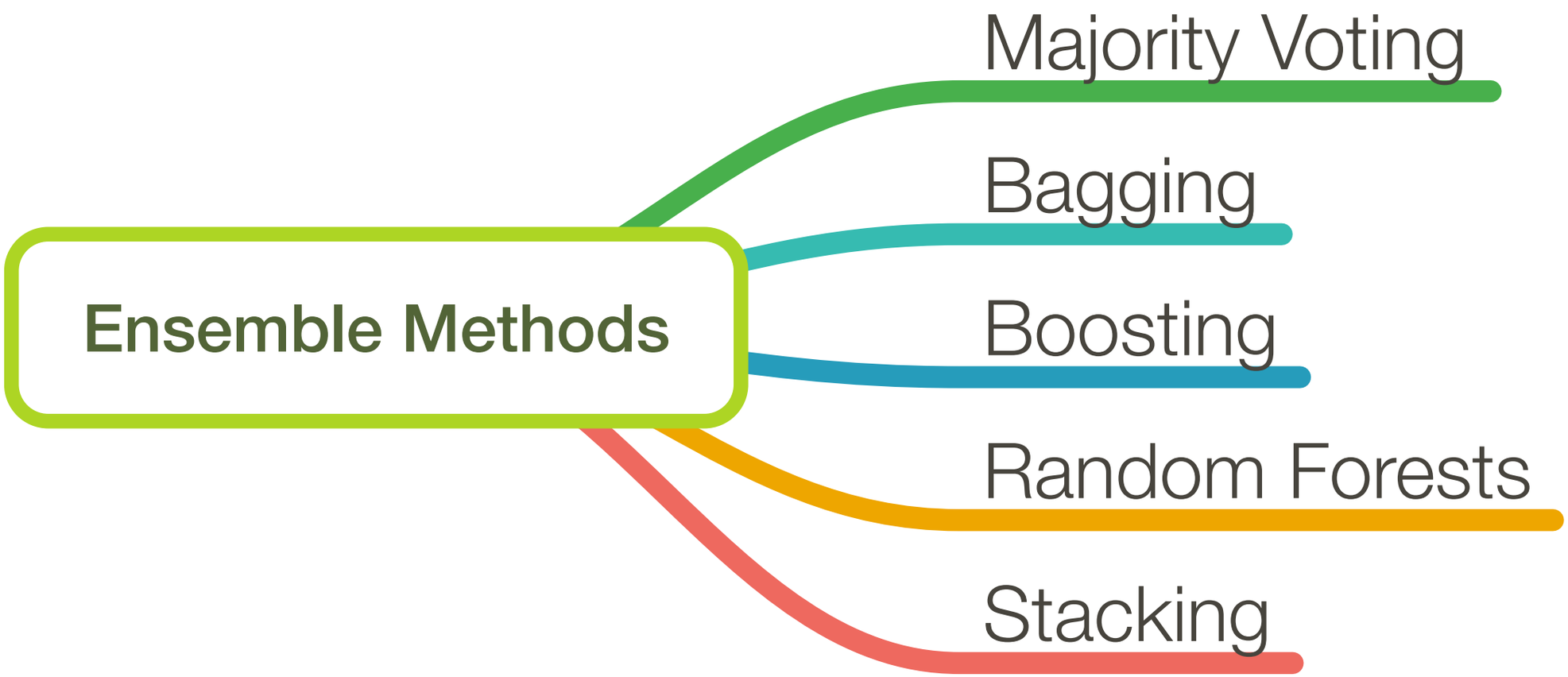
see https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

```
clf1 = DecisionTreeClassifier(random_state=1, max_depth=None)
clf2 = DecisionTreeClassifier(random_state=1, max_depth=1)
clf3 = DecisionTreeClassifier(random_state=1, max_depth=2)
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_valid, y_valid), label))

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

```
Validation Accuracy: 0.91 [Classifier 1]
Validation Accuracy: 0.89 [Classifier 2]
Validation Accuracy: 0.91 [Classifier 3]
Validation Accuracy: 0.91 [Ensemble]
Test Accuracy: 0.93
```

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

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

7.1 Ensemble Methods -- Intro and Overview

7.2 Majority Voting

7.3 Bagging

7.4 Boosting

7.5 Gradient Boosting

7.6 Random Forests

7.7 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, $\mathcal{D}_i$
5: Train base classifier h_i on $\mathcal{D}_i$
6: $\hat{y} = mode\{h_1(\mathbf{x}), ..., h_n(\mathbf{x})\}$

Bootstrap Sampling

Original Dataset | x_1 | x_2 | x_3 | x_4 | x_5 | x_6 | x_7 | x_8 | x_9 | x_{10} |

Bootstrap 1 | x_8 | x_6 | x_2 | x_9 | x_5 | x_8 | x_1 | x_4 | x_8 | x_2 | x_3 | x_7 | x_{10} |

Bootstrap 2 | x_{10} | x_1 | x_3 | x_5 | x_1 | x_7 | x_4 | x_2 | x_1 | x_8 | x_6 | x_9 |

Bootstrap 3 | x_6 | x_5 | x_4 | x_1 | x_2 | x_4 | x_2 | x_6 | x_9 | x_2 | x_3 | x_7 | x_8 | x_{10} |

Training Sets

Bootstrap Sampling

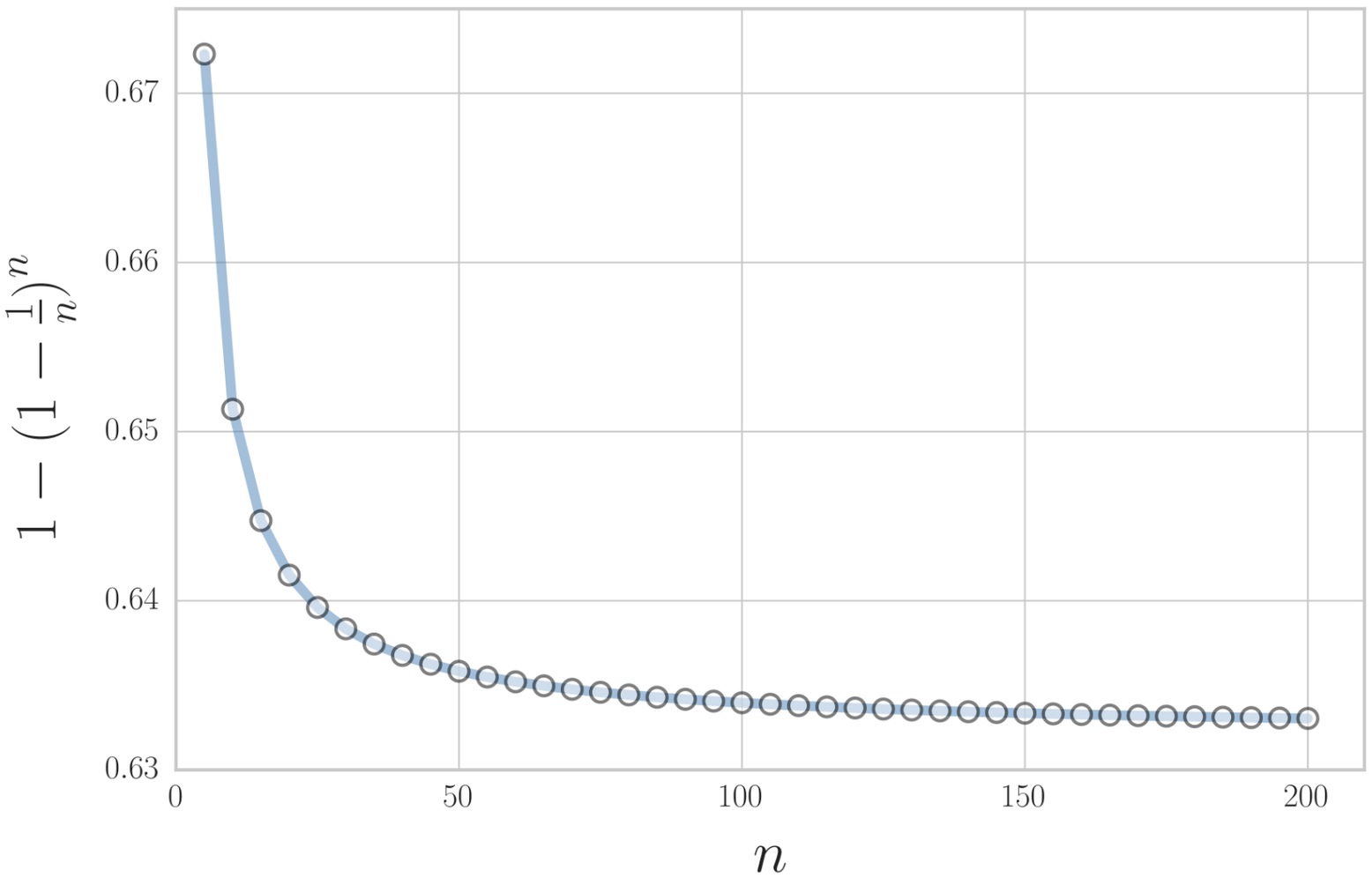
$$P(\text{not chosen}) = \left(1 - \frac{1}{m}\right)^m,$$

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

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

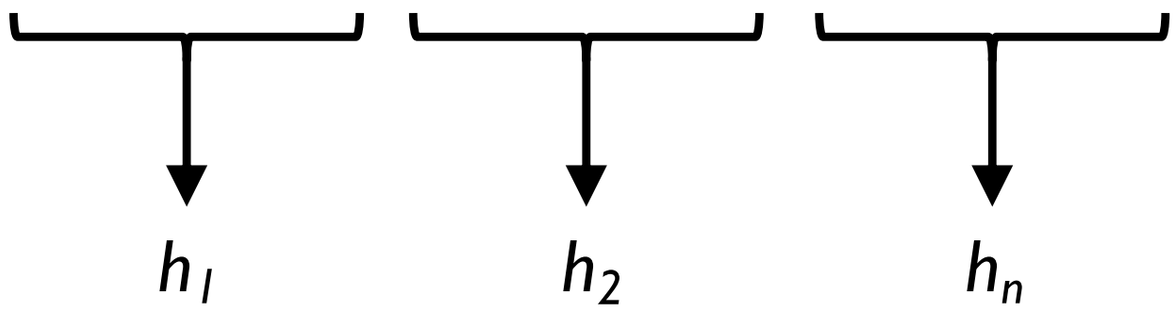
$$\frac{1}{e} \approx 0.368, \quad m \to \infty\,.$$

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

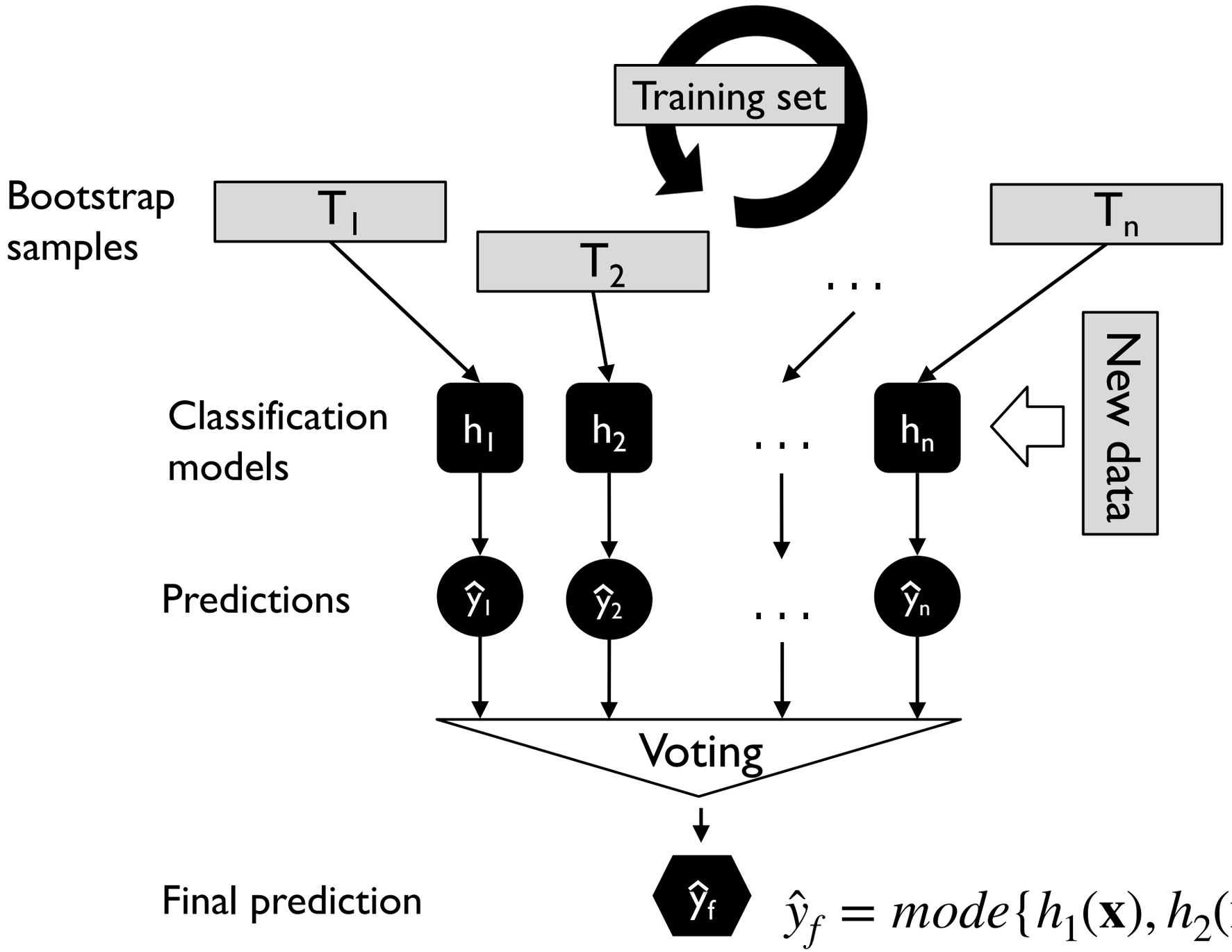


Bootstrap Sampling

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

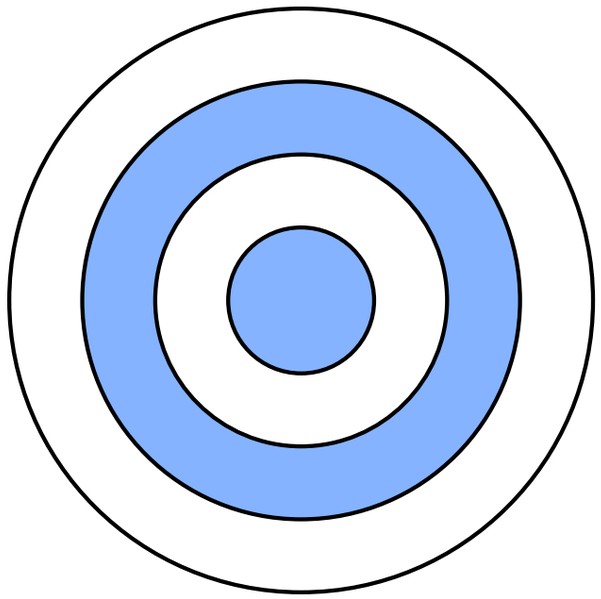
Loss = Bias + Variance + Noise

Bias-Variance Intuition

Low Variance
(Precise)

High Variance
(Not Precise)

Low Bias
(Accurate)



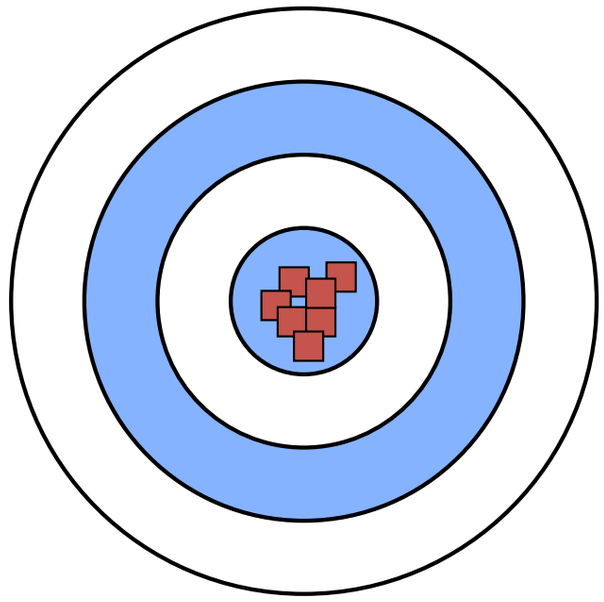
High Bias
(Not Accurate)

Bias-Variance Intuition

Low Variance
(Precise)

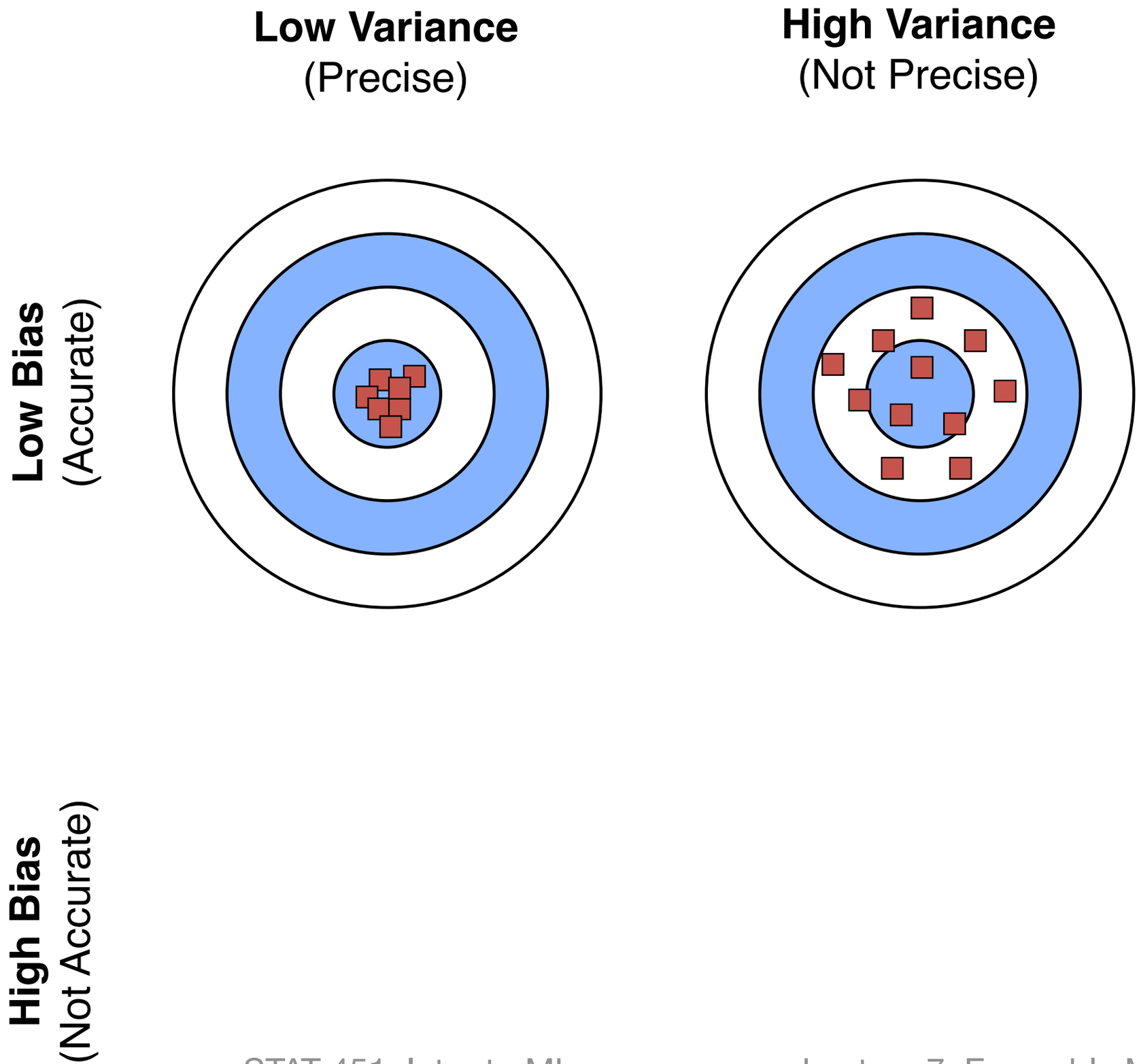
High Variance
(Not Precise)

Low Bias
(Accurate)

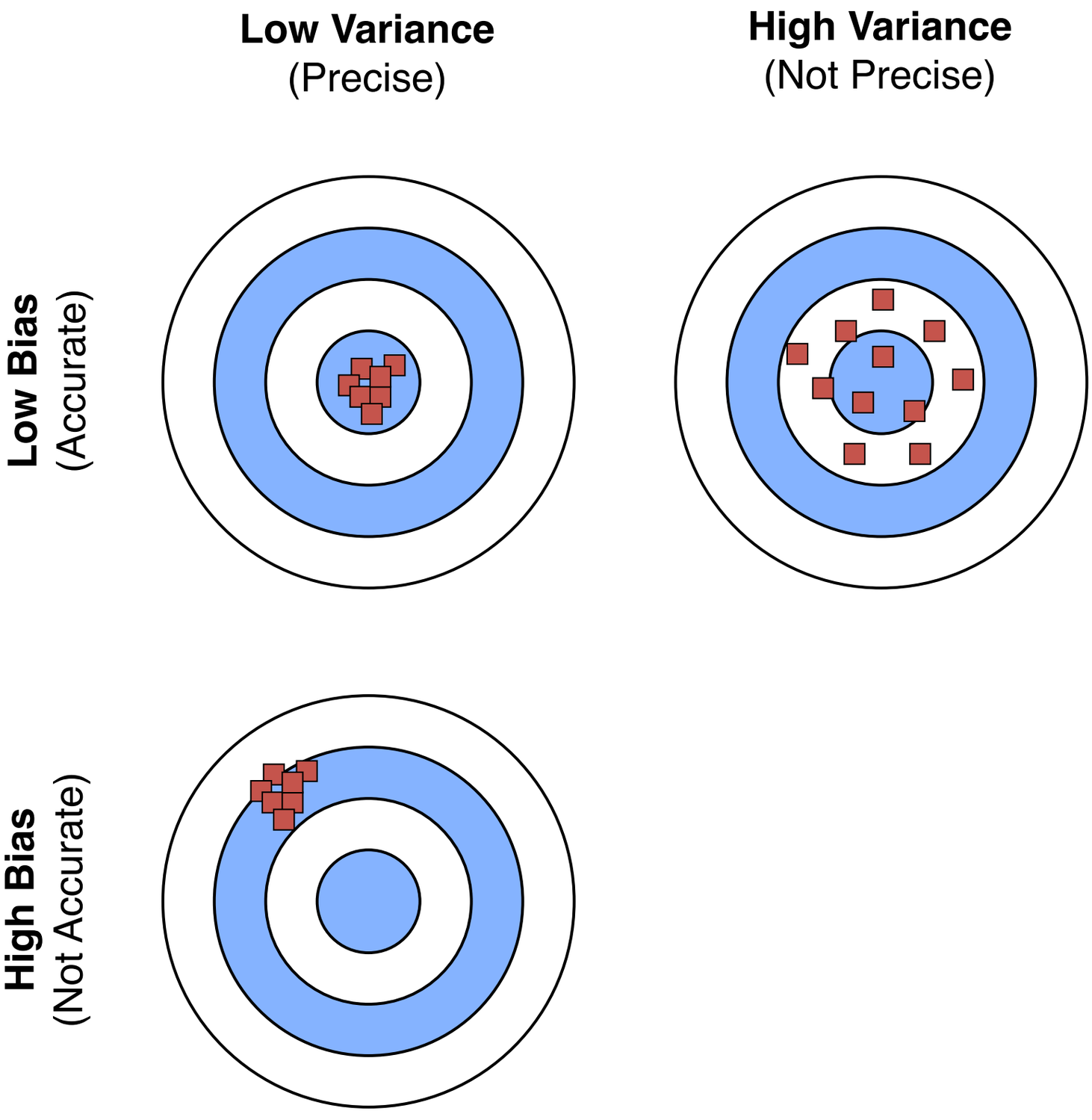


High Bias
(Not Accurate)

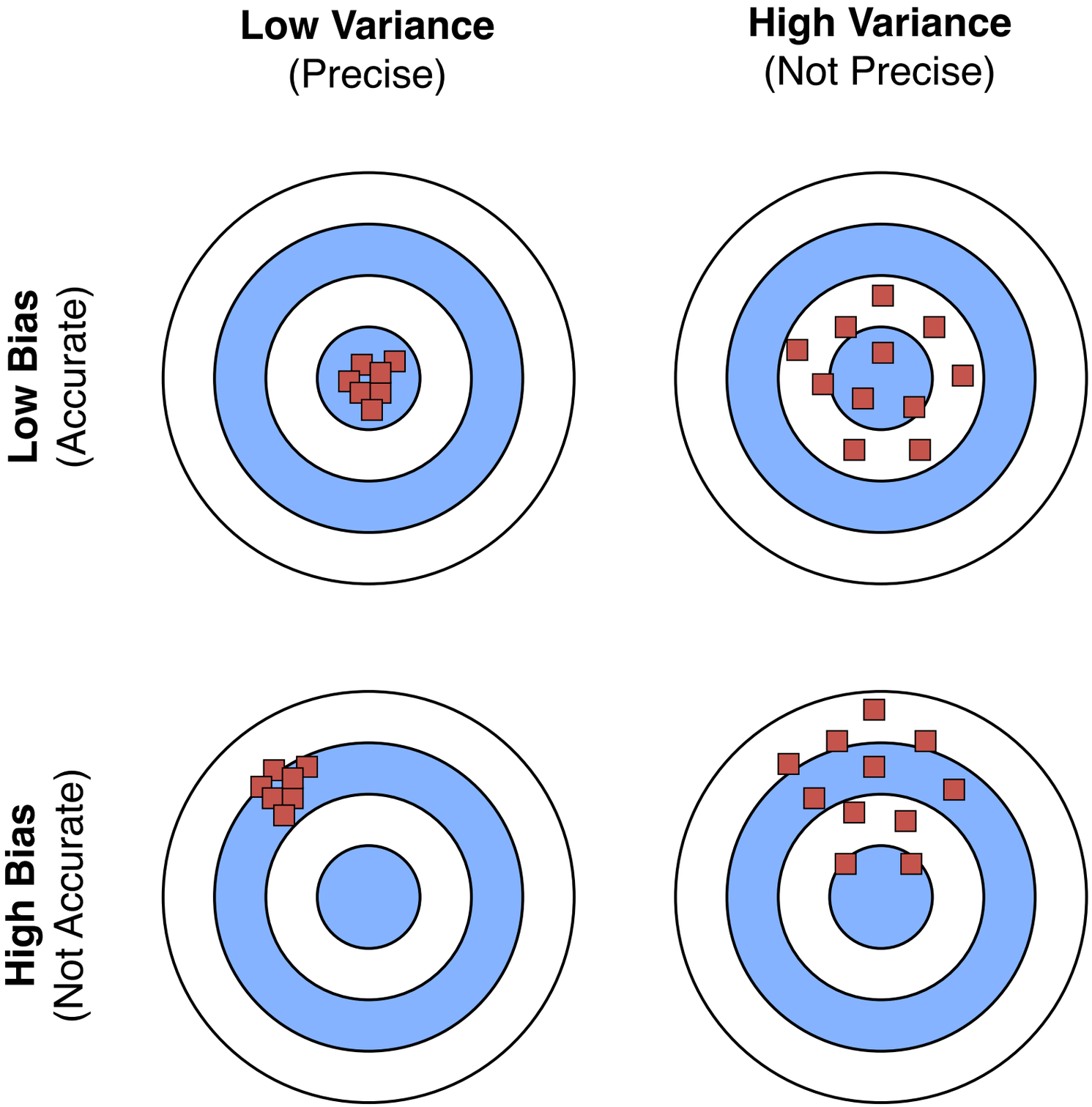
Bias-Variance Intuition



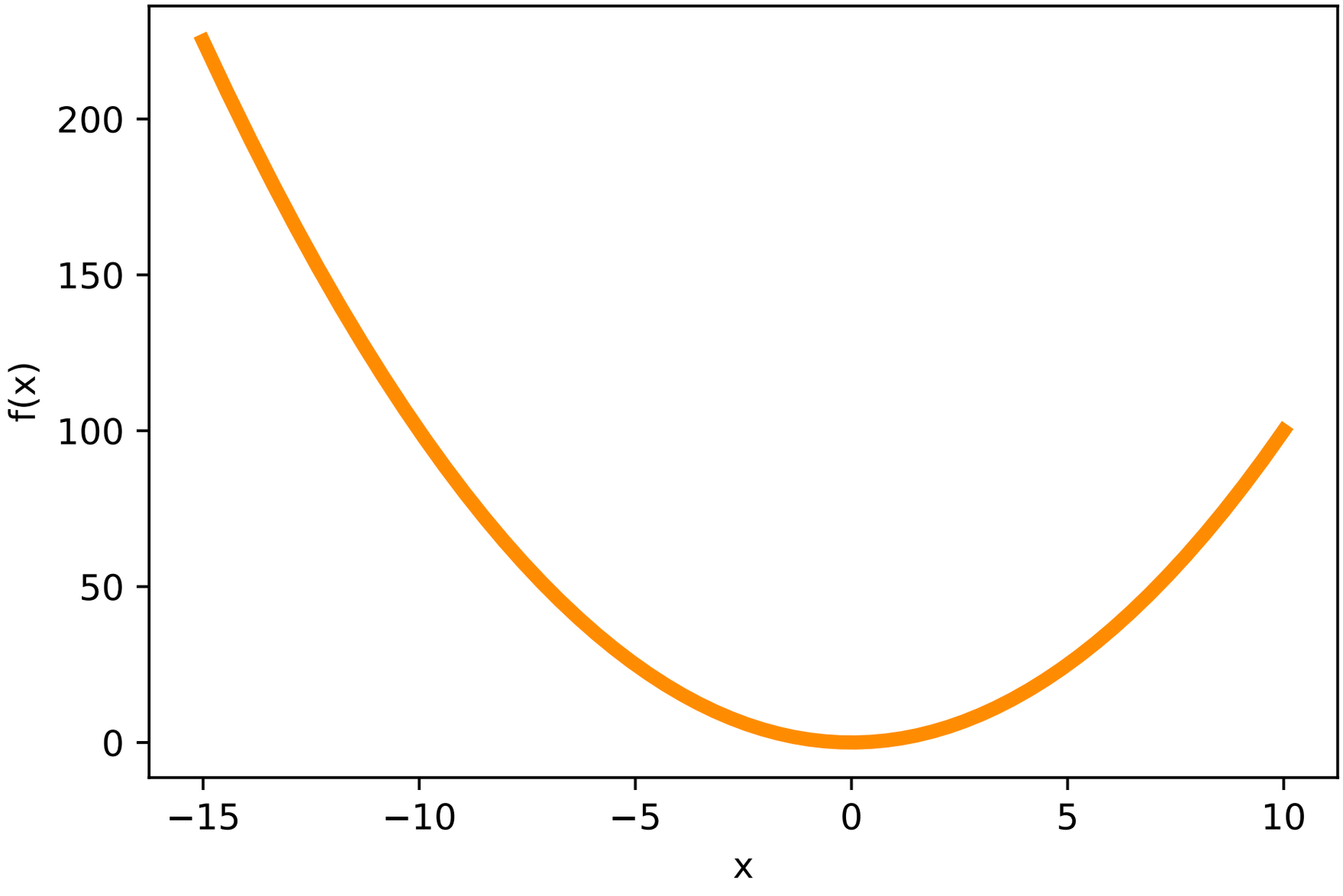
Bias-Variance Intuition



Bias-Variance Intuition

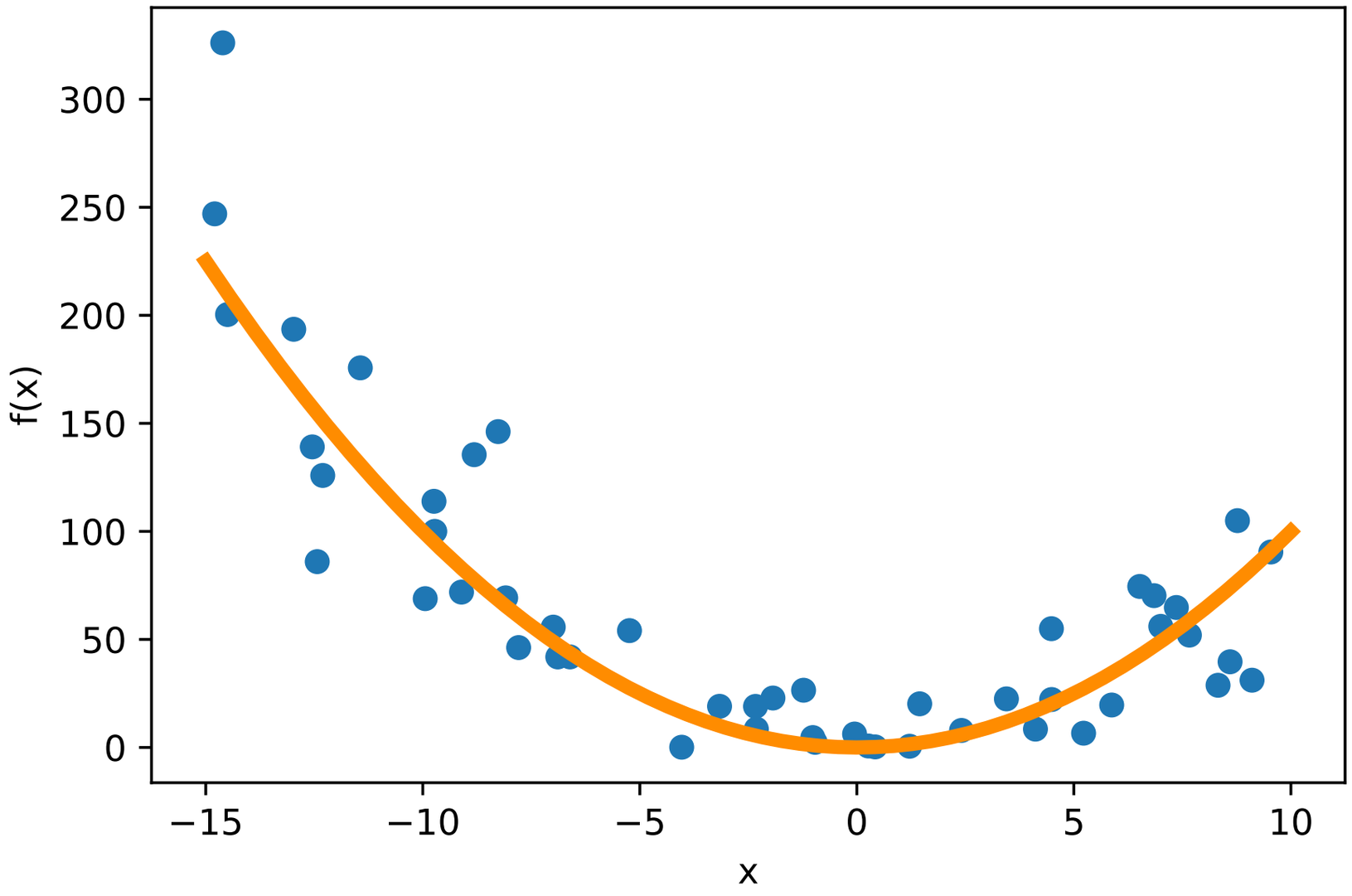


Bias and Variance Example



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

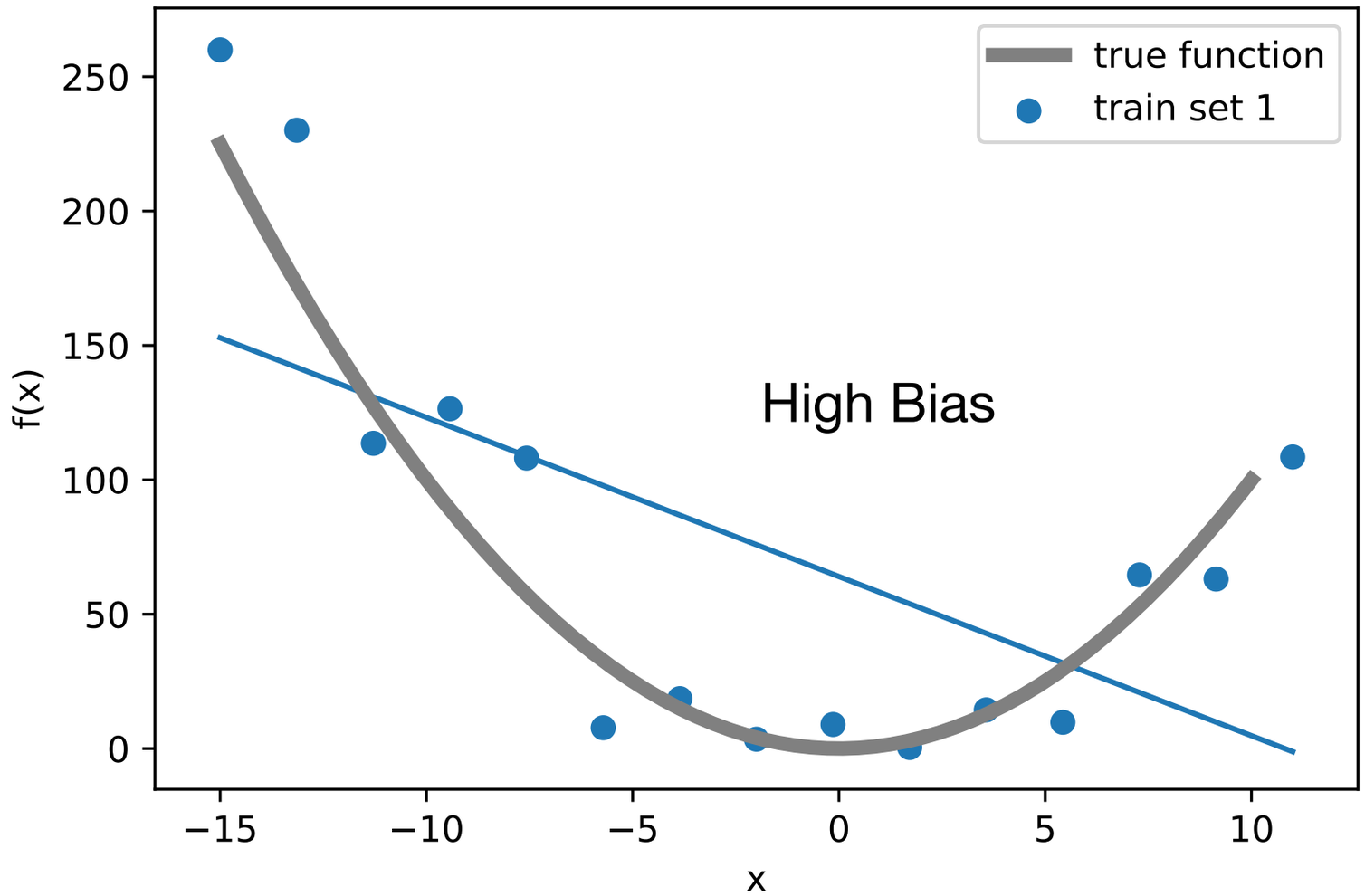
Bias and Variance Example



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

the blue dots is the dataset sampled from the data-generating distribution; here, I added some random Gaussian noise

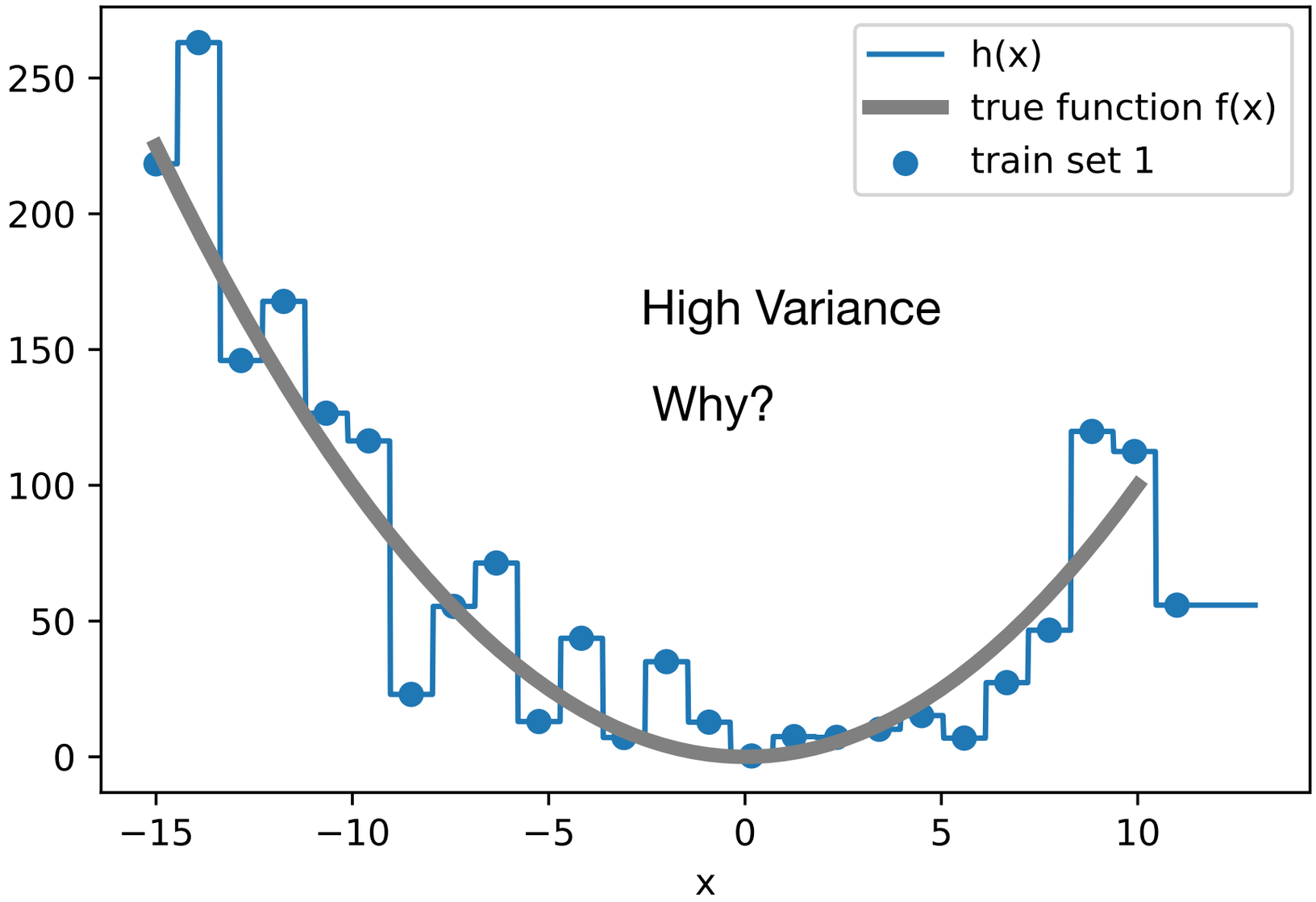
Bias and Variance Example



Suppose we obtain a training set "train set 1".

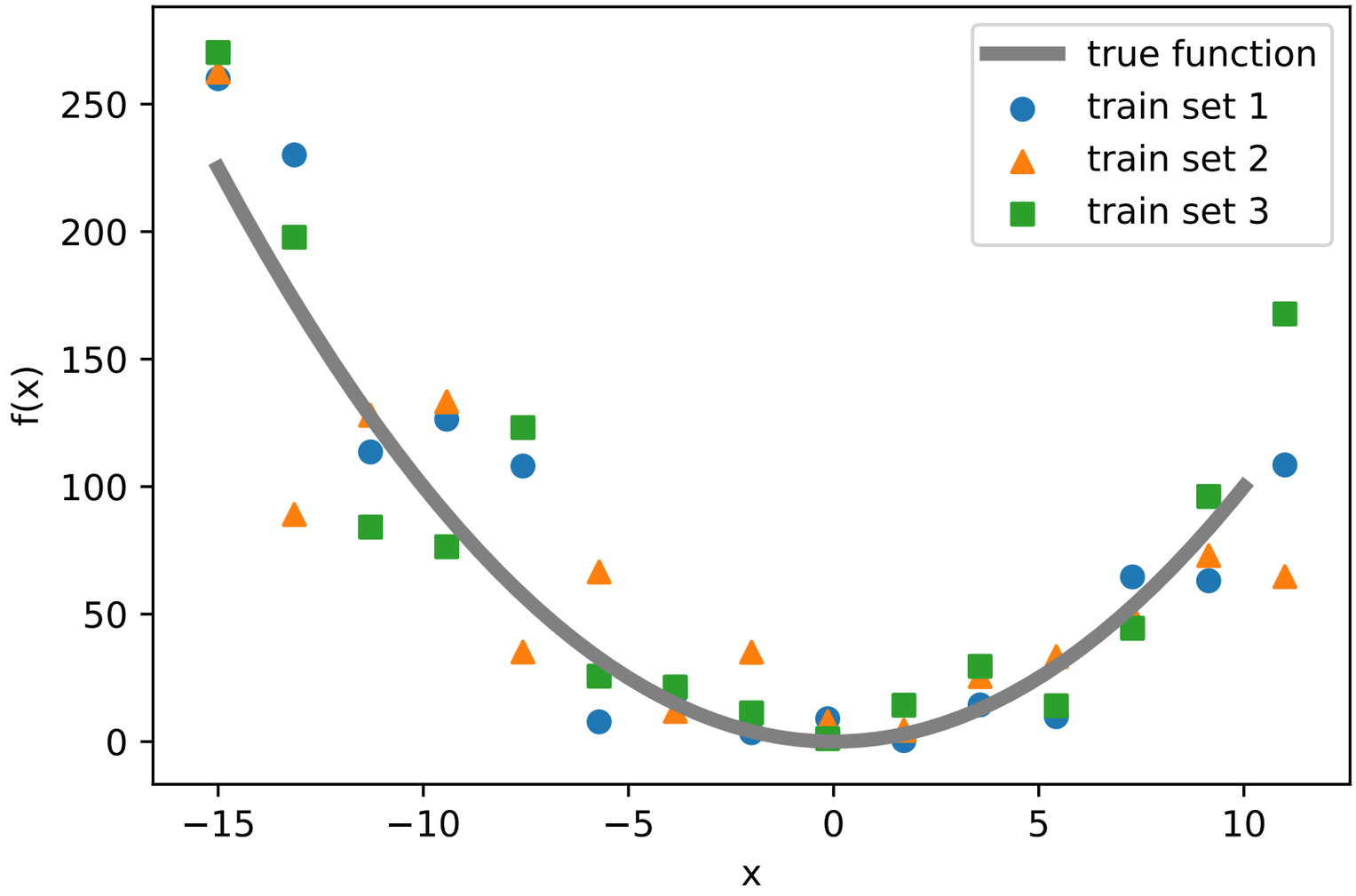
Also support we fit a simple linear model like least-squares linear regression (or a decision tree stump)

Bias and Variance Example



Now, suppose I fit an unpruned decision tree

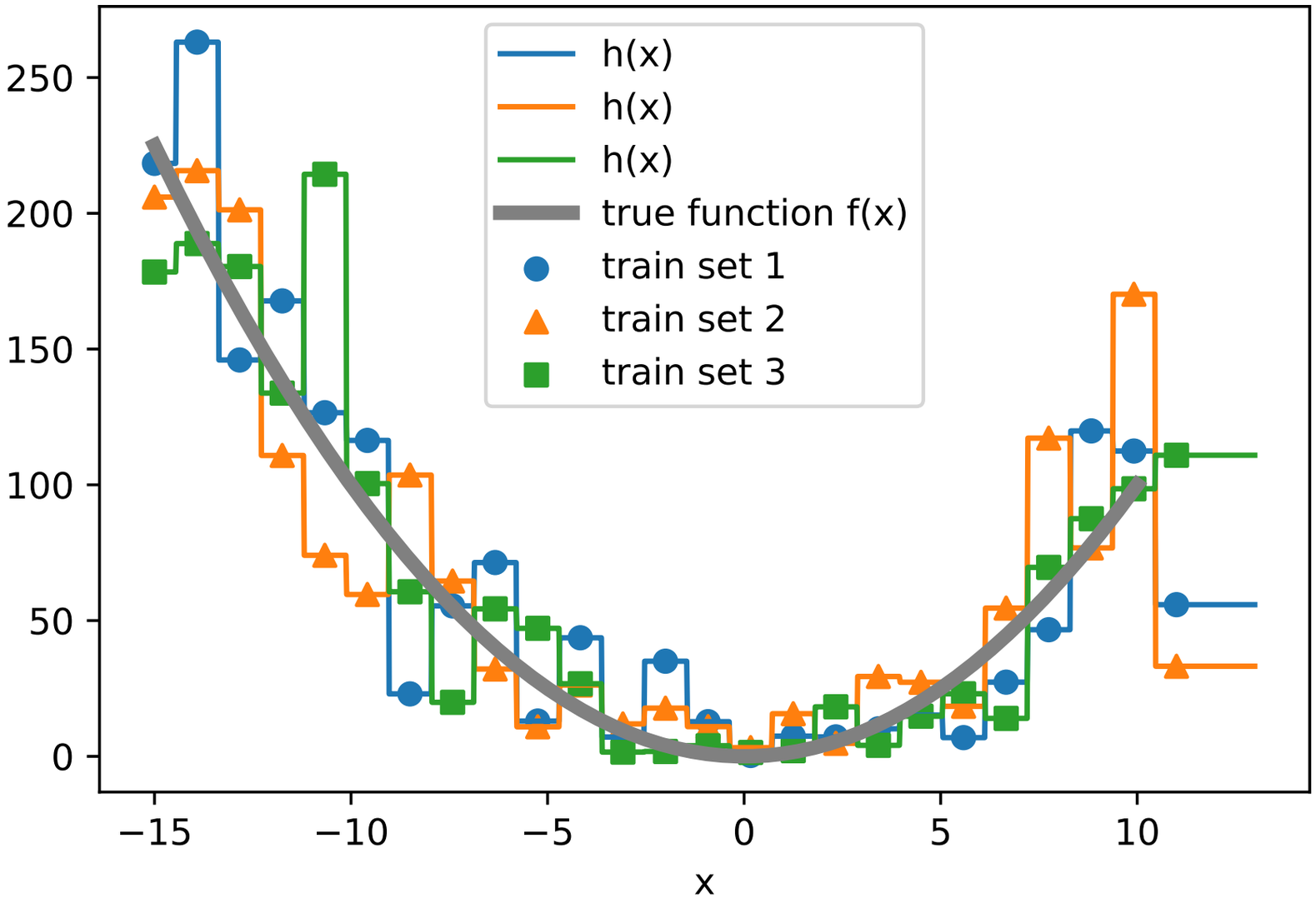
Bias and Variance Example



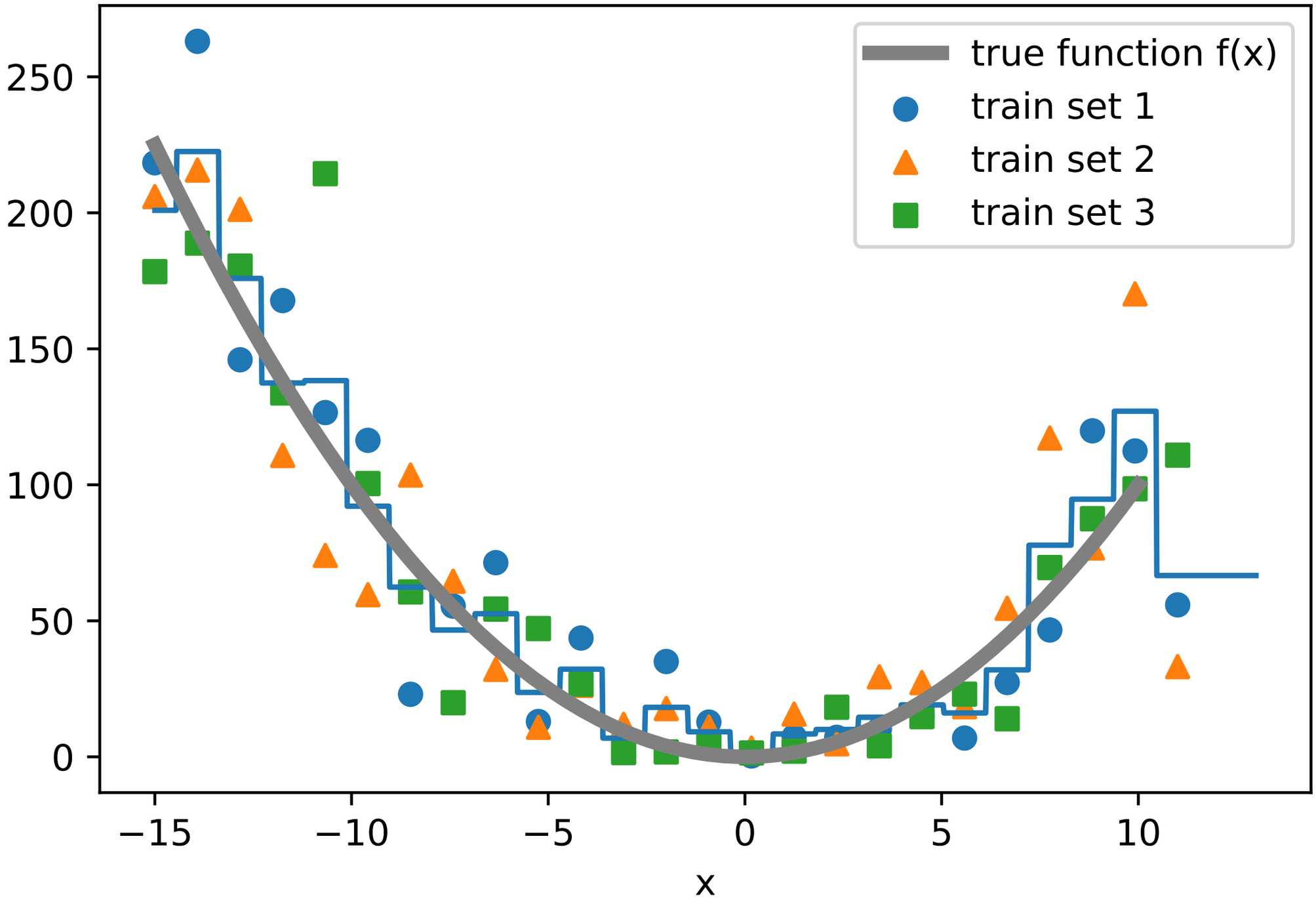
We now fit three unpruned decision trees, one tree for a different training set (train set 1, 2, and 3), each drawn from the same data generating distribution

Bias and Variance Example

High Variance



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


true function f(x)
train set 1
train set 2
train set 3
250
200
150
100
50
0
−15
−10
−5
0
5
10
x

How to use bagging

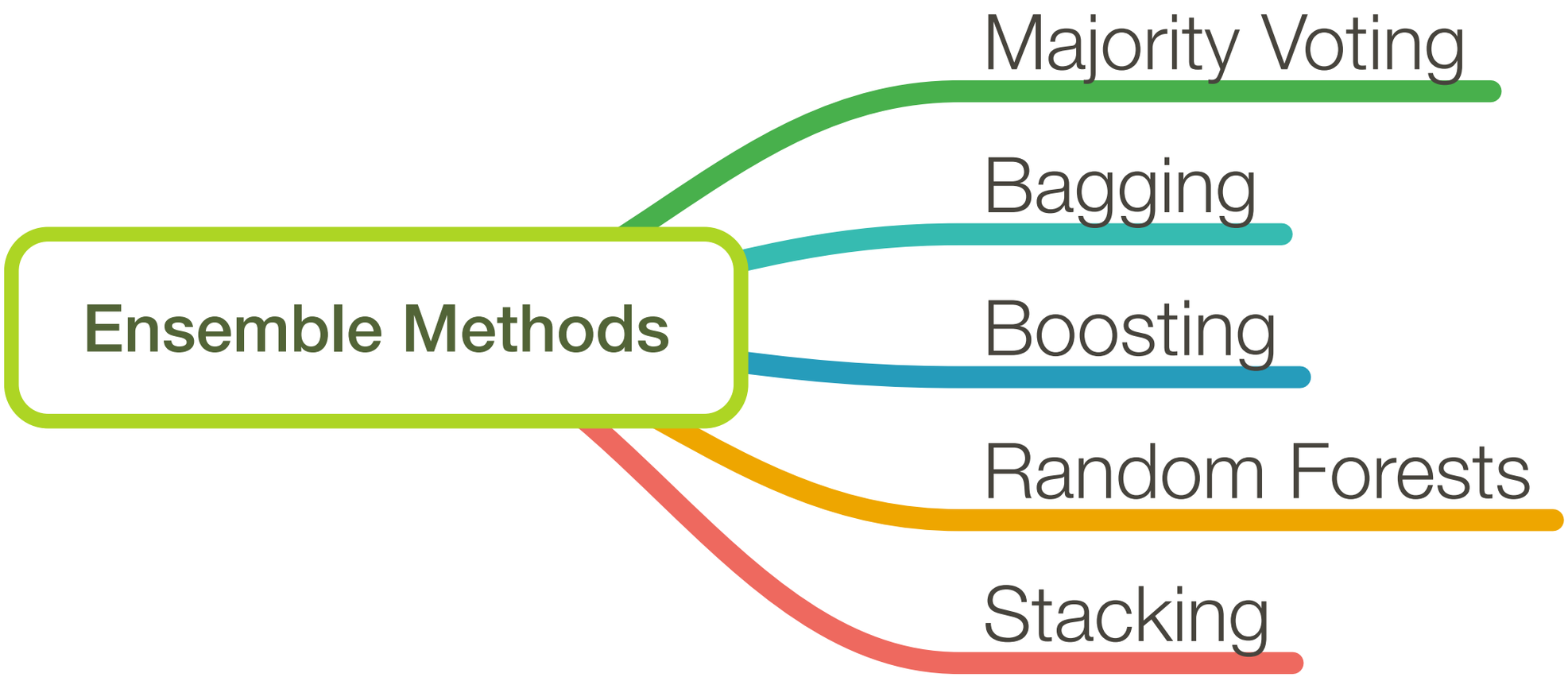
```
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.ensemble import BaggingClassifier


data = datasets.load_breast_cancer()
X, y = data.data, data.target

X_temp, X_test, y_temp, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=123, stratify=y)

X_train, X_valid, y_train, y_valid = \
    train_test_split(X_temp, y_temp, test_size=0.2, random_state=123, stratify=y_temp)

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

```
Train/Valid/Test sizes: 318 80 171
```

```
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("Validation Accuracy: %0.2f" % bag.score(X_valid, y_valid))
print("Test Accuracy: %0.2f" % bag.score(X_test, y_test))
```

```
OOB Accuracy: 0.95
Validation Accuracy: 0.96
Test Accuracy: 0.96
```

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

7.1 Ensemble Methods -- Intro and Overview

7.2 Majority Voting

7.3 Bagging

7.4 Boosting

7.5 Gradient Boosting

7.6 Random Forests

7.7 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

1. weights are updated and
2. classifiers are combined

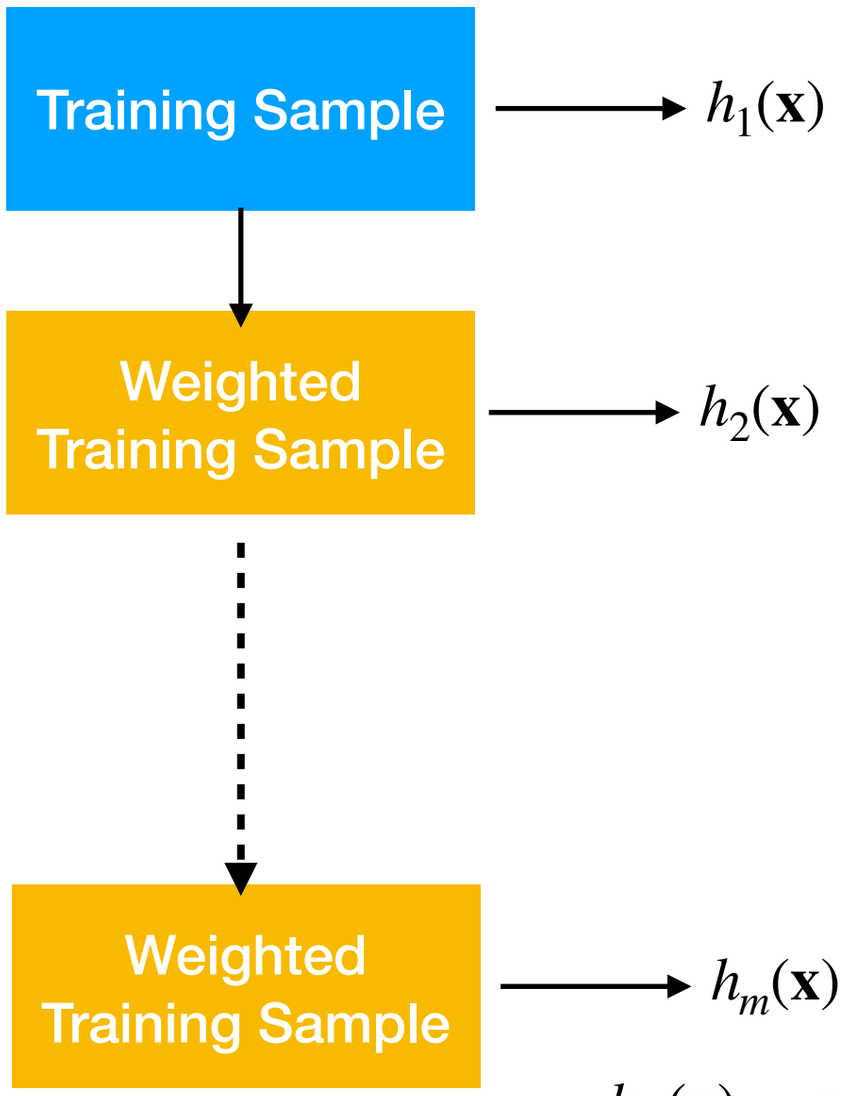
Gradient Boosting

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.

A Simple Boosting Approach



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

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

A Simple Boosting Approach

- Initialize a weight vector with uniform weights

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

- (Weighted) majority voting on trained classifiers

* a learner slightly better than random guessing

AdaBoost

Algorithm 1 AdaBoost

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

Line 11: Assuming constant alpha, lower the weights for correct prediction; increase the weight for incorrect prediction

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

Classifier weight

Training example weight

Decision Tree Stumps

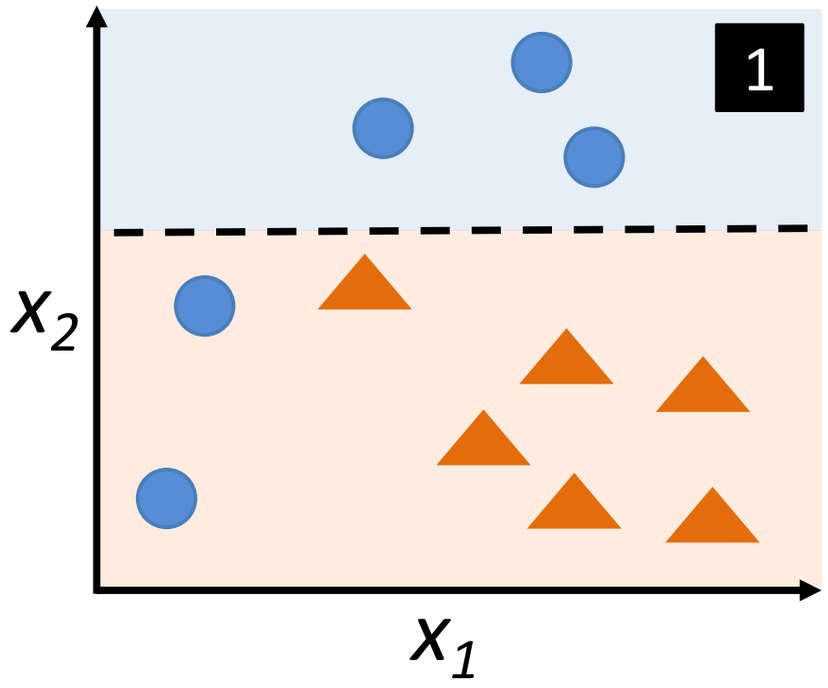
Weak classifier, here: decision tree stump for binary classification problem with labels -1, 1

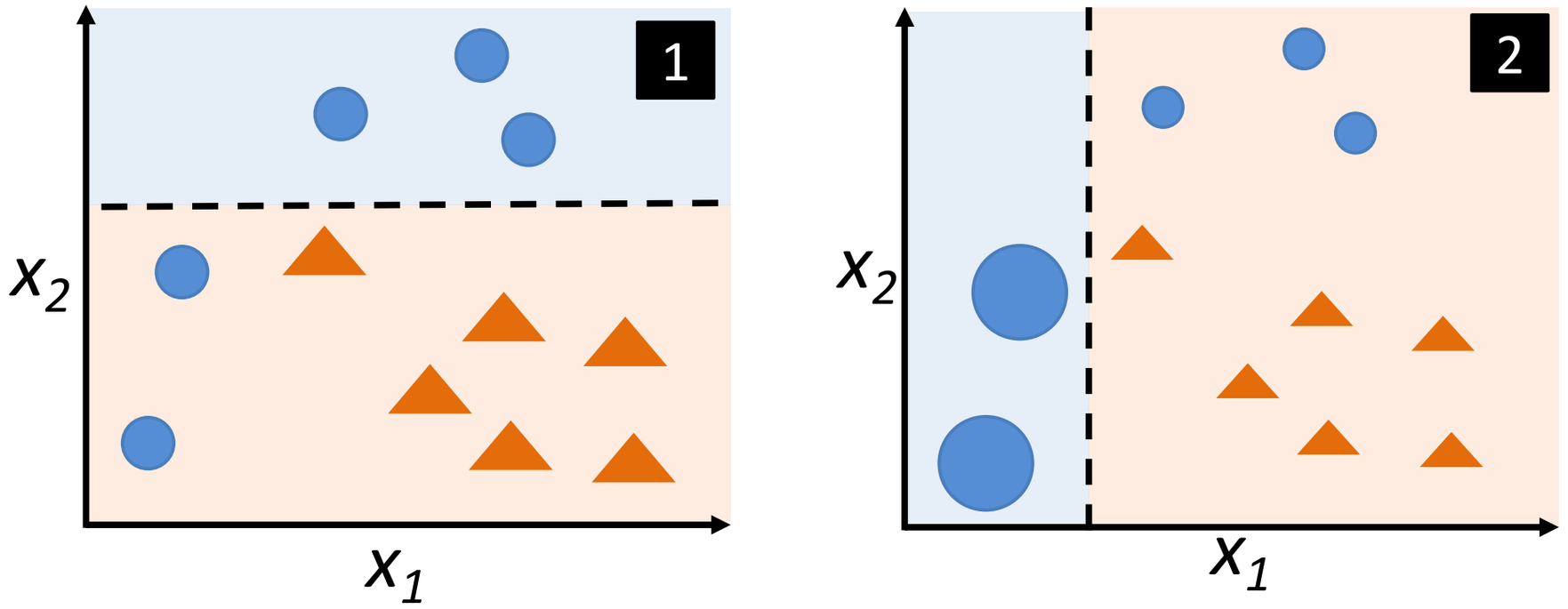
$$h(\mathbf{x}) = s\Big(\mathbf{1}(x_k \geq t)\Big)$$

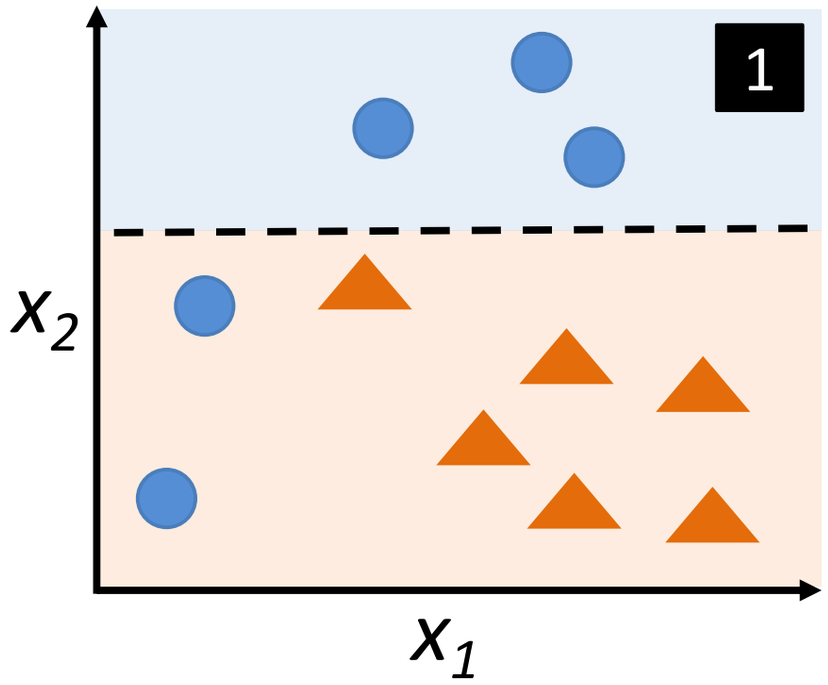
where

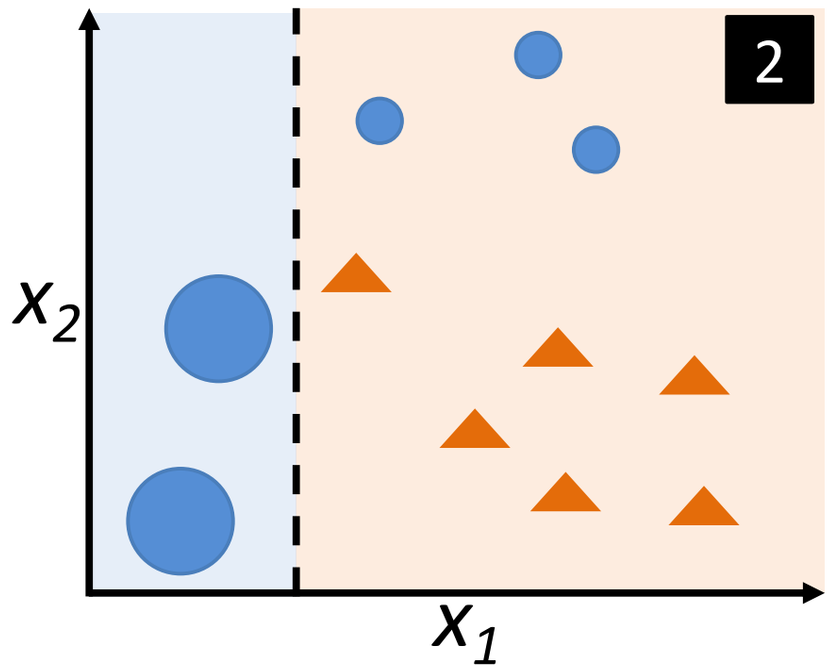
$s(x) \in \{-1, 1\}$

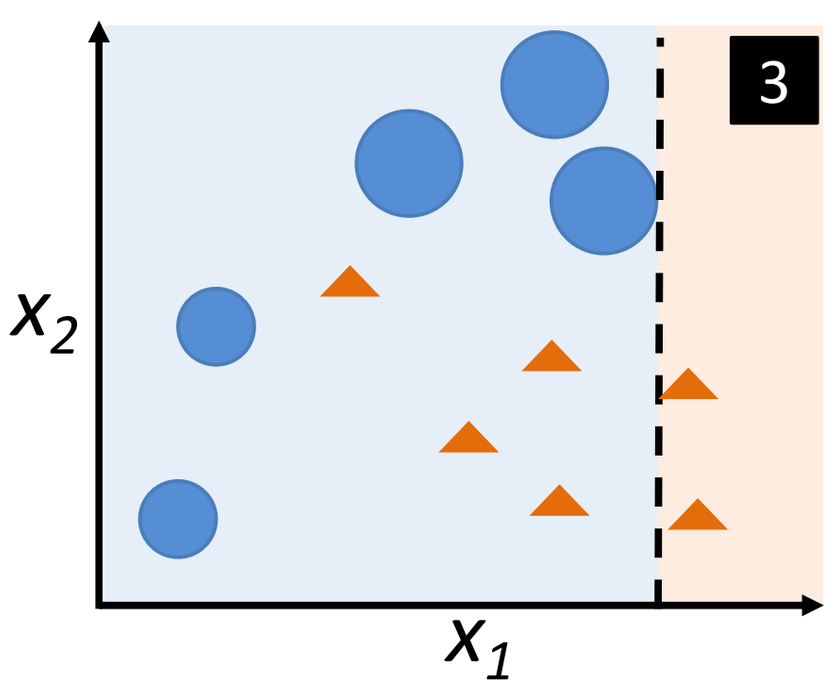
$k \in \{1, ..., K\}$ (K is the number of features)

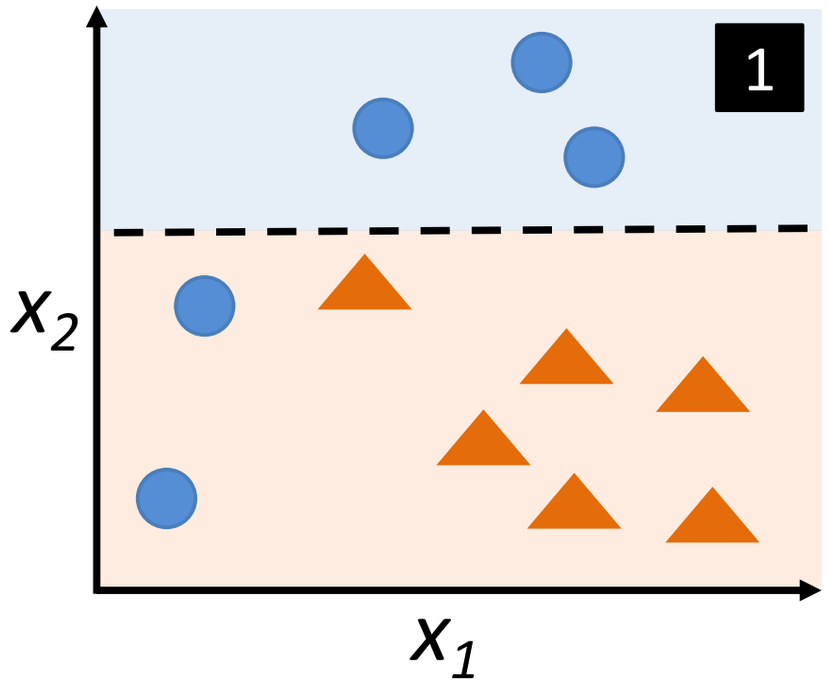

1
x_2
x_1

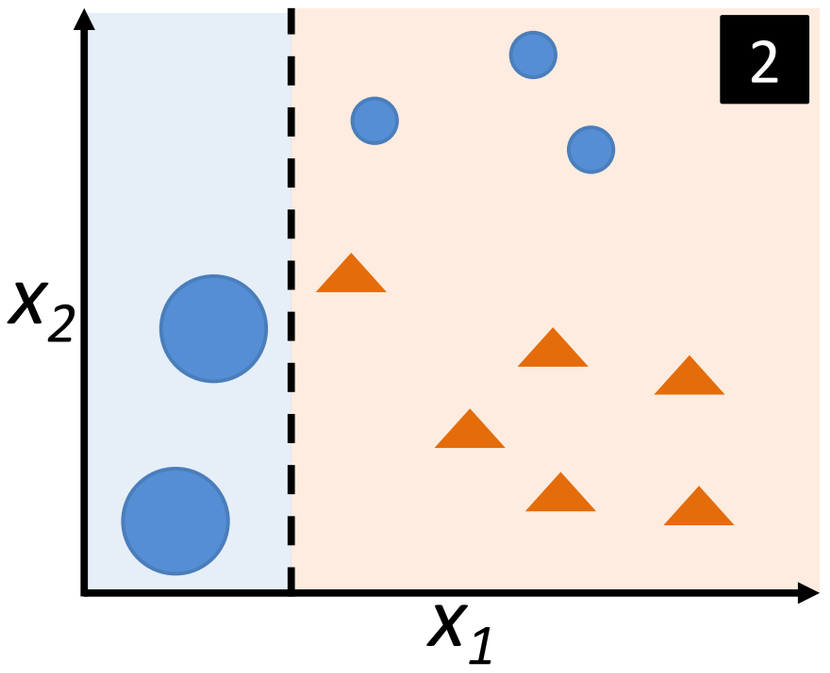

1
x_2
x_1
2
x_2
x_1

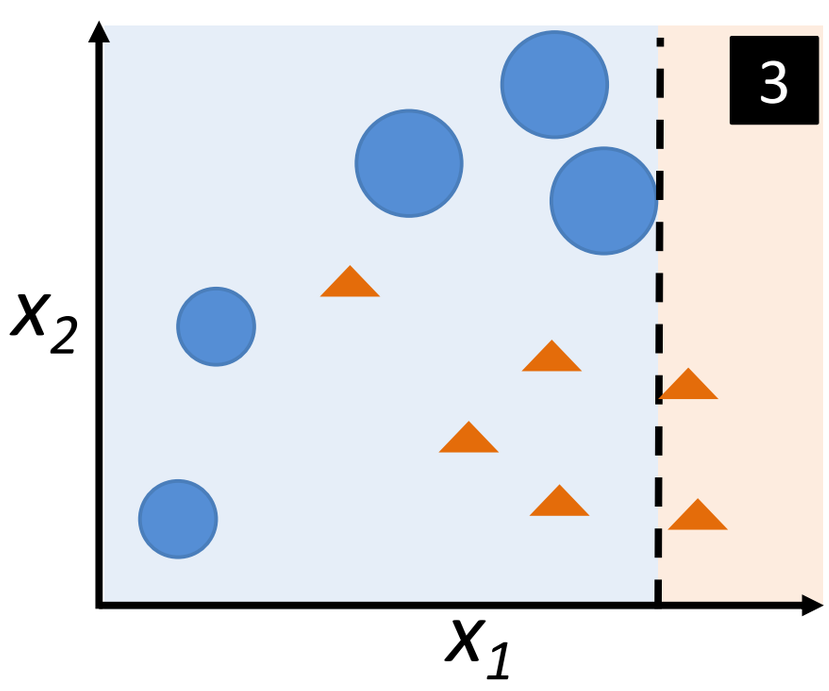

1
x_2
x_1

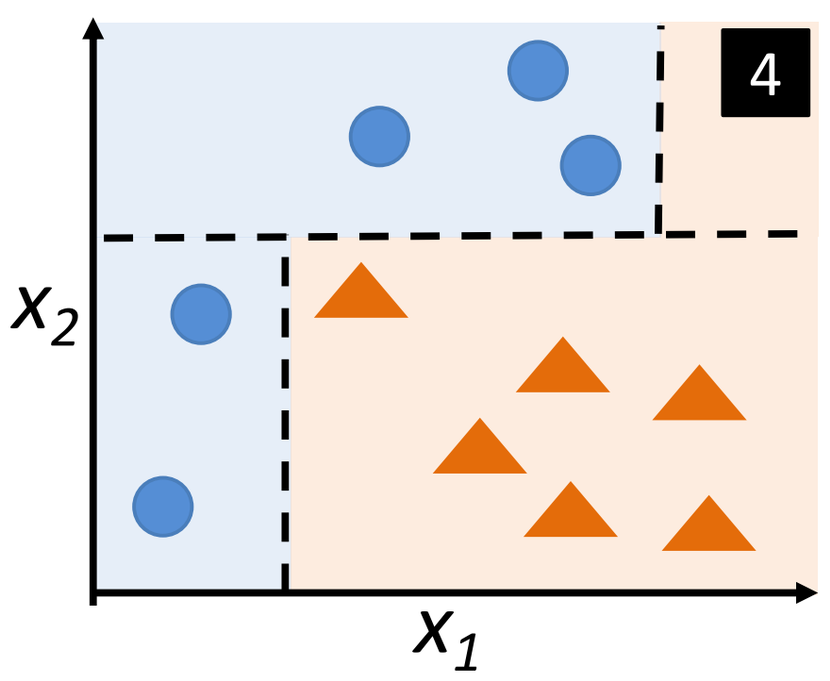

2
x_2
x_1


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

AdaBoost for Multi-class Settings

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*

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

   (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 for $\alpha^{(m)}$ to be positive, we only need $(1 - err^{(m)}) > 1/K$, or the accuracy of each weak classifier to be better than random guessing rather than $1/2$. To appreciate its effect, we

Adaboost example

```
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

data = datasets.load_breast_cancer()
X, y = data.data, data.target

X_temp, X_test, y_temp, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=123, stratify=y)

X_train, X_valid, y_train, y_valid = \
    train_test_split(X_temp, y_temp, test_size=0.2, random_state=123, stratify=y_temp)

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

```
Train/Valid/Test sizes: 318 80 171
```

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

```
Valid Accuracy: 0.97
Test Accuracy: 0.98
```

```
boost.estimator_weights_
```

```
array([2.50552594, 1.92278773, 1.55587214, 1.25187013, 0.83252642,
       1.08142915, 0.47669717, 0.52052104, 0.95125412, 0.80934036,
       0.55109052, 1.05012036, 0.55221499, 0.63914069, 0.59736276,
```