

# Machine Learning

## Lecture 9: Bias-Variance Tradeoff and Ensembles

---

Prof. Dr. Aleksandar Bojchevski

15.05.24

Bias-variance tradeoff

Ensemble learning

No free lunch

## Decomposition of the expected test error

expected test error = variance + bias<sup>2</sup> + noise

## Supervised regression

$\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$  with  $(\mathbf{x}_i, y_i) \sim p(\mathbf{x}, y)$  and  $y_i \in \mathbb{R}$ .

For any input  $\mathbf{x}$  there might not exist a unique label  $y$ , the **expected label**  $\bar{y}(\mathbf{x})$  is:

$$\bar{y}(\mathbf{x}) = \mathbb{E}_{y|\mathbf{x}}[y]$$

The **expected test error** given a model  $h_{\mathcal{D}}$  learned on a dataset  $\mathcal{D}$  is:

$$\mathbb{E}_{(\mathbf{x}, y) \sim p} [(h_{\mathcal{D}}(\mathbf{x}) - y)^2]$$

The model  $h_{\mathcal{D}}$  is also a random variable since it depends on the dataset  $\mathcal{D}$  which is random. Denoting  $\mathcal{D} \sim p^N$  for  $N$  samples the **expected classifier**  $\bar{h}$  is:

$$\bar{h} = \mathbb{E}_{\mathcal{D} \sim p^N} [h_{\mathcal{D}}]$$

$\bar{h}$  is a weighted average of functions.

## Decomposition of the expected test error

The expected test error of an algorithm  $\mathcal{A}$  that takes a dataset  $\mathcal{D}$  and produces a model  $h_{\mathcal{D}}$ , taking into account that we sample the dataset  $\mathcal{D}$ , is:

$$\mathbb{E}_{\substack{(\mathbf{x}, y) \sim p \\ \mathcal{D} \sim p^N}} [(h_{\mathcal{D}}(\mathbf{x}) - y)^2]$$

First we introduce the expected classifier  $\bar{h}(\mathbf{x})$ :

$$\mathbb{E}_{\mathbf{x}, y, \mathcal{D}} [(h_{\mathcal{D}}(\mathbf{x}) - y)^2] = [(h_{\mathcal{D}}(\mathbf{x}) - \bar{h}(\mathbf{x}) + \bar{h}(\mathbf{x}) - y)^2]$$

We expand into:

$$\underbrace{\mathbb{E}_{\mathbf{x}, \mathcal{D}} [(h_{\mathcal{D}}(\mathbf{x}) - \bar{h}(\mathbf{x}))^2]}_{\text{variance}} + \underbrace{2\mathbb{E}_{\mathbf{x}, y, \mathcal{D}} [(h_{\mathcal{D}}(\mathbf{x}) - \bar{h}(\mathbf{x}))(\bar{h}(\mathbf{x}) - y)]}_{=0} + \underbrace{\mathbb{E}_{\mathbf{x}, y} [(\bar{h}(\mathbf{x}) - y)^2]}_{(*)}$$

## Decomposition of the expected test error

Next we consider the last term from the previous slide:

$$\begin{aligned} (*) &= \mathbb{E}_{\mathbf{x},y} [(\bar{h}(\mathbf{x}) - y)^2] = \mathbb{E}_{\mathbf{x},y} [(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}) + \bar{y}(\mathbf{x}) - y)^2] \\ &= \underbrace{\mathbb{E}_{\mathbf{x}} [(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}))^2]}_{\text{bias}^2} + \underbrace{2\mathbb{E}_{\mathbf{x},y} [(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}))(\bar{y}(\mathbf{x}) - y)]}_{=0} + \underbrace{\mathbb{E}_{\mathbf{x},y} [(\bar{y}(\mathbf{x}) - y)^2]}_{\text{noise}} \end{aligned}$$

Putting everything together:

$$\underbrace{\mathbb{E}_{\mathbf{x},y,\mathcal{D}} [(h_{\mathcal{D}}(\mathbf{x}) - y)^2]}_{\text{expected test error}} = \underbrace{\mathbb{E}_{\mathbf{x},\mathcal{D}} [(h_{\mathcal{D}}(\mathbf{x}) - \bar{h}(\mathbf{x}))^2]}_{\text{variance}} + \underbrace{\mathbb{E}_{\mathbf{x}} [(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}))^2]}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathbf{x},y} [(\bar{y}(\mathbf{x}) - y)^2]}_{\text{noise}}$$

## Decomposition of the expected test error

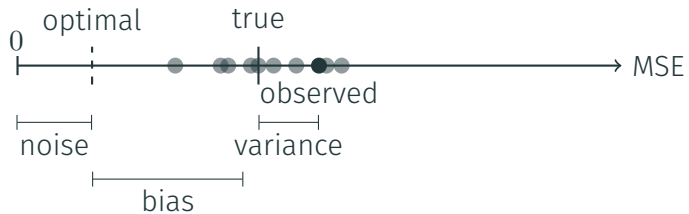
$$\underbrace{\mathbb{E}_{\mathbf{x}, y, \mathcal{D}} [(h_{\mathcal{D}}(\mathbf{x}) - y)^2]}_{\text{expected test error}} = \underbrace{\mathbb{E}_{\mathbf{x}, \mathcal{D}} [(h_{\mathcal{D}}(\mathbf{x}) - \bar{h}(\mathbf{x}))^2]}_{\text{variance}} + \underbrace{\mathbb{E}_{\mathbf{x}} [(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}))^2]}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathbf{x}, y} [(\bar{y}(\mathbf{x}) - y)^2]}_{\text{noise}}$$

**Variance:** How much your classifier changes if you train on a different training set.  
How much the classifier (over)fits to a particular training set.  
How far off is the best possible model (for our data) from the average classifier?

**Bias:** Inherent error of your classifier even with infinite training data.  
Due to bias towards a particular kind of models (e.g. linear classifiers).

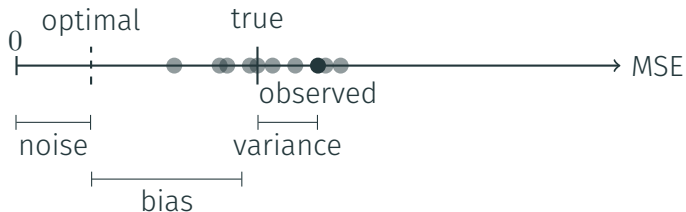
**Noise:** irreducible data-intrinsic error.


# Bias-variance tradeoff





# Bias-variance tradeoff



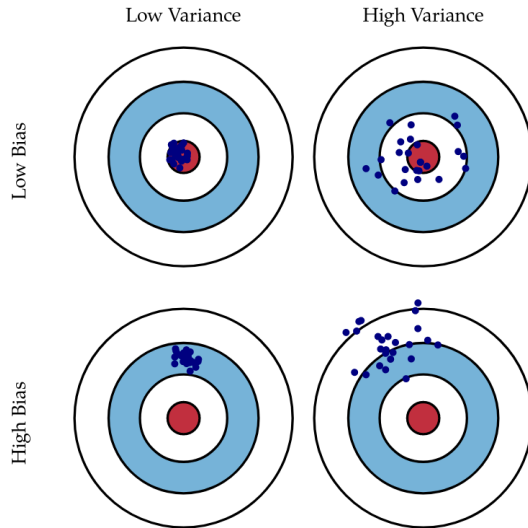
Low bias, low variance:  MSE

Low bias, high variance:  MSE

High bias, low variance:  MSE

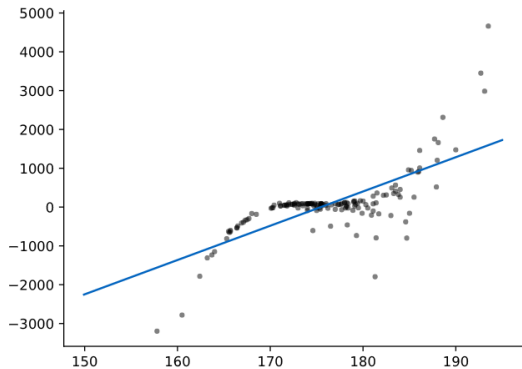
High bias, high variance:  MSE

# Bias-variance tradeoff

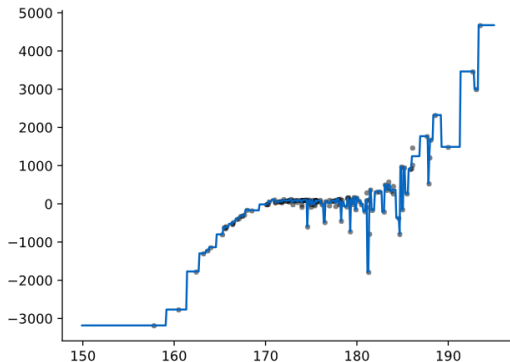


# Bias-variance tradeoff

High bias, Low Variance: Underfitting



Low bias, High variance: Overfitting



## Bias-variance trade-off for polynomial regression

Increasing the degree of the polynomial  $M$

Increasing the regularization strength  $\lambda$

## Bias-variance trade-off for polynomial regression

Increasing the degree of the polynomial  $M$

- decreases bias – the function is more flexible

Increasing the regularization strength  $\lambda$

# Bias-variance trade-off for polynomial regression

Increasing the degree of the polynomial  $M$

- decreases bias – the function is more flexible
- increases variance – we can more easily fit the noise

Increasing the regularization strength  $\lambda$

# Bias-variance trade-off for polynomial regression

Increasing the degree of the polynomial  $M$

- decreases bias – the function is more flexible
- increases variance – we can more easily fit the noise

Increasing the regularization strength  $\lambda$

- increases bias

# Bias-variance trade-off for polynomial regression

Increasing the degree of the polynomial  $M$

- decreases bias – the function is more flexible
- increases variance – we can more easily fit the noise

Increasing the regularization strength  $\lambda$

- increases bias
- decreases variance



# Making the trade-off

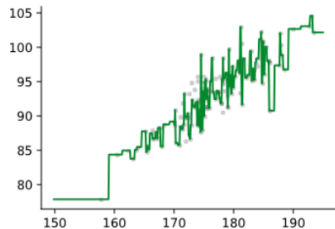
Reducing bias: increase model capacity, add features.

Reducing variance: reduce model capacity (e.g. tree depth), add regularization.

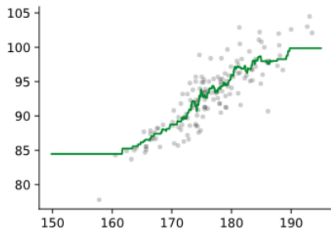
Ensembles usually decrease variance but have a negligible increase on bias.

Example on  $k$ -NN regression: increasing  $k$  increases bias and decreases variance.

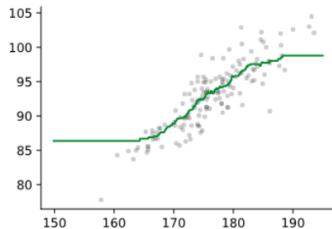
$k = 1$



$k = 10$



$k = 25$



Bias-variance tradeoff

Ensemble learning

No free lunch

# Ensemble learning

Improve the performance by aggregating predictions of many (diverse) classifiers.

The main benefit: Similar bias but lower variance compared to the base models.

For regression a simple way is to **average** multiple models:

$$f(y) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(\mathbf{x})$$

where  $f_m$  is the  $m$ 'th base model.

For classifiers, we can take e.g. the **majority vote** of the outputs.

**Stacking:** Train a meta-classifier with the base classifiers' predictions as features.

**Stacking:** Train a meta-classifier with the base classifiers' predictions as features.

**Bagging** (bootstrap **agg**regating):

- Create new datasets by sampling the training set (with replacement)
- Train separate classifiers on each dataset
- Combine the predictions, e.g. average or majority vote

**Stacking:** Train a meta-classifier with the base classifiers' predictions as features.

**Bagging** (bootstrap **aggregating**):

- Create new datasets by sampling the training set (with replacement)
- Train separate classifiers on each dataset
- Combine the predictions, e.g. average or majority vote

**Boosting:**

- Incrementally train (weak) classifiers that correct previous mistakes
- Focus (give higher weight) on hard (misclassified) examples

**Stacking:** Train a meta-classifier with the base classifiers' predictions as features.

**Bagging** (bootstrap **aggregating**):

- Create new datasets by sampling the training set (with replacement)
- Train separate classifiers on each dataset
- Combine the predictions, e.g. average or majority vote

**Boosting:**

- Incrementally train (weak) classifiers that correct previous mistakes
- Focus (give higher weight) on hard (misclassified) examples

Bucket of models, Bayesian model averaging, Dropout, ....

# Stacking

Compared to use an unweighted average or majority vote, an alternative is **stacking**, which learns how to combine the base models.

For example, we can learn a linear combination with ensemble weights  $w_m$

$$f(\mathbf{x}) = \sum_{m \in \mathcal{M}} w_m f_m(\mathbf{x})$$

**Note:** Here  $w_m$  need to be trained on a [separate dataset](#), otherwise they would put all their mass on the best performing base model.

You can also use any other model to combine individual models.



Different version of your dataset by sampling instances **with replacement**.

Probability that a single item is not selected from a set of size  $N$  in any of the  $N$  draws is  $(1 - \frac{1}{N})^N \rightarrow e^{-1} \approx 0.37$  as  $N \rightarrow \infty$ .

Each base model only sees, on average, 63% ( $1 - 0.37$ ) unique instances.

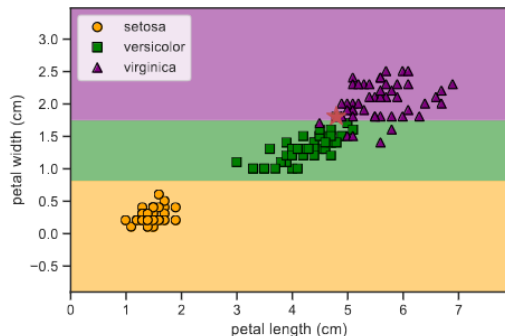
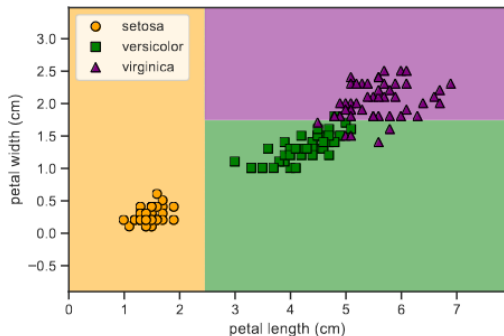
The unused instances ( $\approx 37\%$ ) are called out-of-bag instances (oob).

We can use them as a validation set to estimate test performance.

Prevents that a single instance has a large influence.

# Influence of a single instance on decision tree (iris dataset)

Omitting a single data point (shown by red star) changes the decision boundary.



## Bagging: Bootstrap AGgregating

Generate  $B$  different datasets  $\mathcal{D}_1, \dots, \mathcal{D}_B$  sampled from  $\mathcal{D}$  with bootstrap.

Fit a tree  $f_b$  on each  $\mathcal{D}_b$  and predict the average  $f_{\text{bag}} = \frac{1}{B} \sum_{b=1}^B f_b$ .

The variance of the average of  $k$  i.i.d. samples with variance  $\sigma$  and correlation  $\rho$  is  $\rho\sigma^2 + \frac{(1-\rho)}{k}\sigma^2$ . If the samples are correlated there benefit is reduced.

Since bootstrap samples have a large overlap, bagged trees are highly correlated.

This reduces the benefit of bagging since we need **diverse** classifiers.

## Bagging + trees + random features = Random forests

Idea: Use a **random subset** of features to learn each **bagged** tree.

Widely used in practice due to good “out-of-the-box” performance.

Can be efficiently trained in parallel.

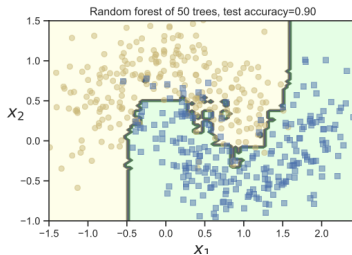
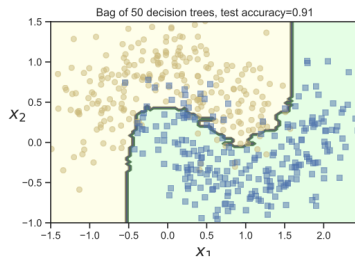
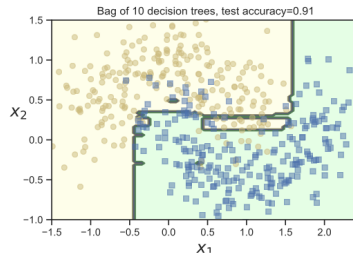
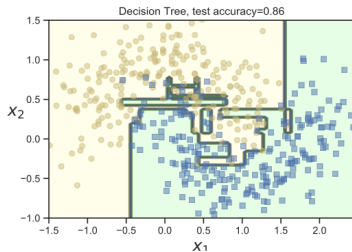
But less interpretable than simple decision trees.

Rule-of-thumb for the number of random features:  $m = \log_2(D)$  for regression or  $m = \sqrt{D}$  for classification where  $D$  is the number of features.

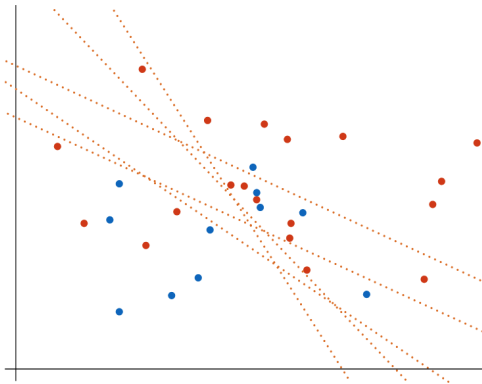
Example, assume there is one very strong predictor:

- All trees will have this predictor, probably even at the top, thus they are similar
- The random forest will have the predictor only in  $m/D$  of the splits

# Ensemble of trees and random forests

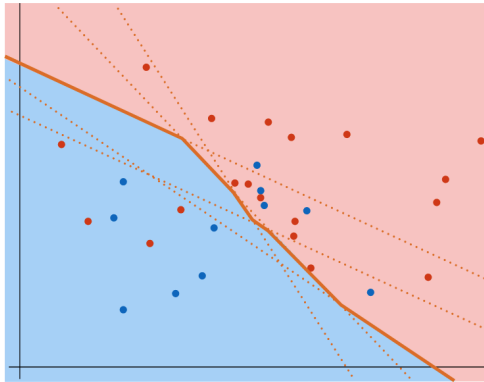


## Bagging linear classifiers



Multiple linear classifiers trained on different bootstrap samples of our data are slightly different (indicated by dotted lines).

## Bagging linear classifiers



Piecewise linear decision boundary: Every time two decision boundaries in the ensemble cross, the majority changes.

## Limitation of bagging

Works well for trees, but bagging does not always improve performance.

Relies on the base models being **unstable** estimators: Omitting some of the data **significantly** changes the resulting model fit.

It does not work well for other models, such as

- Nearest neighbor classifiers
- Neural networks: Although they can be unstable w.r.t. their training set, they will underperform if they can only see 63% of the data.



## Boosting turns many weak learners into a strong learner

How can we turn many **weak** classifiers  $f_m(\cdot)$  into a strong classifier?

Weak means that we cannot obtain 0 training loss, e.g. one-level decision tree.

# Boosting turns many weak learners into a strong learner

How can we turn many **weak** classifiers  $f_m(\cdot)$  into a strong classifier?

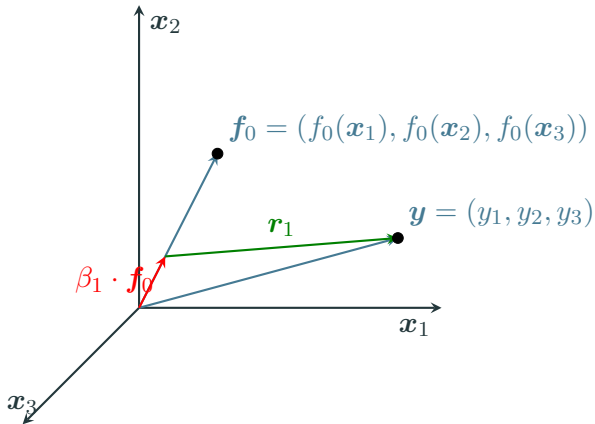
Weak means that we cannot obtain 0 training loss, e.g. one-level decision tree.

Given model  $f_0$  adjust the vector of predictions  $\mathbf{f}_0 = (f_0(\mathbf{x}_1), f_0(\mathbf{x}_2), f_0(\mathbf{x}_3))$  to reach the label vector  $\mathbf{y} = (y_1, y_2, y_3)$ .

Take a small step towards  $\mathbf{f}_0$ .

Train  $f_{m+1}$  to predict the residual of the previous weak learner  $r_m = \beta_m \mathbf{f}_m - \mathbf{y}$ .

Final model:  $f(\mathbf{x}) = \sum_{m=1}^M \beta_m f_m(\mathbf{x})$ .

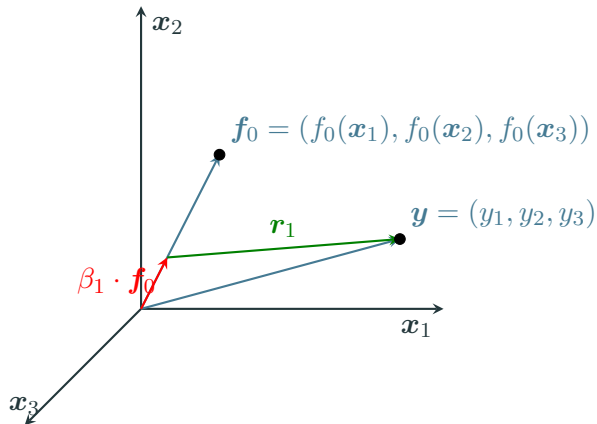


## When does boosting work?

If  $\mathbf{f}$  is not orthogonal to  $\mathbf{y}$  in each step we will make some progress and eventually reach the target vector  $\mathbf{y}$ .

- If they point in the same direction continue, otherwise flip the sign.

For classification, each  $f_m(\cdot)$  only needs to be slightly better than random.



# Gradient boosting

1. Initialize  $f_0(\mathbf{x}) = \arg \min_h \sum_{i=1}^N l(h(\mathbf{x}_i), y_i)$ , e.g.  $h$  is a weak tree.
2. For  $m = 1$  until  $M$  do
  - 2a Compute the gradient residuals  $r_{im} = \left[ \frac{\partial l(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \right]_{f(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i)}$
  - 2b Learn residuals with weak learner  $f_m = \arg \min_h \sum_{i=1}^N (r_{im} - h(\mathbf{x}))^2$
  - 2c Update the base model with  $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu f_m(\mathbf{x})$
3. Output the final model  $f(\mathbf{x}) = f_M(\mathbf{x})$ . which combines all “weak learners” trained during the process.

For regression we have  $r_{im} = \left[ \frac{\partial l(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \right]_{f(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i)} = f_{m-1}(\mathbf{x}_i) - y_i$ .

---

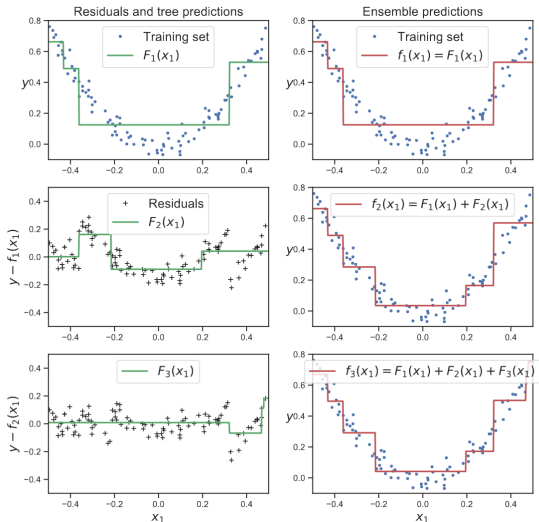
XGBoost is an efficient and widely used implementation of gradient boosting.

# Example

First fit a regression tree (depth 2)  
on the training data (top).

Then fit a regression tree (depth 2)  
on the residuals (mid).

Re-calculate residuals and fit  
another regression tree (bottom).



# Adaptive boosting (AdaBoost)

AdaBoost is a powerful ensemble learning method used primarily for binary classification tasks.

The procedure of loss boosting:

1. Initialize the weights of training instances uniformly
2. For each iteration:
  - 2a Train a “weak learner” on the weighted training data
  - 2b Calculate the error rate of the “weak learner”
  - 2c Adjust the weights of the training instances to give more weight to the misclassified instances
3. Combine the “weak learners” into a strong learner by weighted averaging or voting according to performance.

## Boosting vs. Bagging

Boosting reduces the **bias** of the “weak learners”, by fitting models (e.g. shallow trees or decision stumps) that depend on each other.

On the contrary, bagging (e.g. random forests) reduce the **variance** by fitting independent trees.

Bias-variance tradeoff

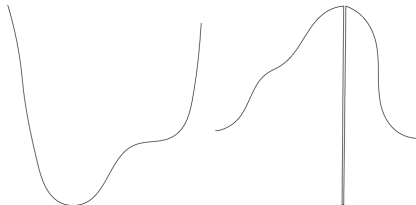
Ensemble learning

No free lunch



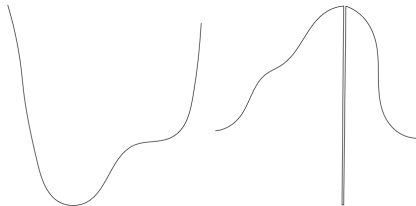
## No free lunch theorem(s)

**Theorem (Wolpert & MacReady, 1997):** All optimization algorithms are equivalent when their performance is averaged across all possible problems.



## No free lunch theorem(s)

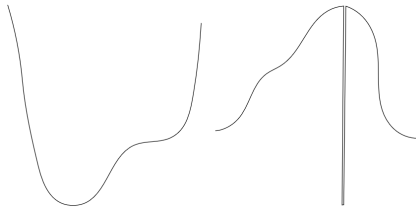
**Theorem (Wolpert & MacReady, 1997):** All optimization algorithms are equivalent when their performance is averaged across all possible problems.



The problem of induction: learning from experience works in practice, but there are exceptions, and we might not be able to tell from data when they will happen.

## No free lunch theorem(s)

**Theorem (Wolpert & MacReady, 1997):** All optimization algorithms are equivalent when their performance is averaged across all possible problems.



The problem of induction: learning from experience works in practice, but there are exceptions, and we might not be able to tell from data when they will happen.

“Solution”: the universal distribution generating all “natural” data is not uniform.

## Inductive bias and Occam's razor

Inductive bias = assumptions that are implicitly or explicitly baked into the model.

For example, a linear method has an inductive bias for linear relations.

Rotation-invariant neural network produces same output for any rotated input.

Graph neural networks often assume homophily.

---

<sup>1</sup>Determining which model is “simpler” is not trivial. Naively counting parameters might not work.

# Inductive bias and Occam's razor

Inductive bias = assumptions that are implicitly or explicitly baked into the model.

For example, a linear method has an inductive bias for linear relations.

Rotation-invariant neural network produces same output for any rotated input.

Graph neural networks often assume homophily.

**Occam's razor:** The simplest explanation is often the best.

We should bias our algorithms towards simple models.

If models A and B perform similarly, prefer the simpler model.<sup>1</sup>

---

<sup>1</sup>Determining which model is “simpler” is not trivial. Naively counting parameters might not work.

## Summary

Ensemble learning combines multiple models to improve performance.

Trees + bagging + random features = Random forests (competitive on many tasks).

Boosting: Get “strong learners” by incrementally training “weak learners”.

The expected test error can be decomposed into bias, variance, and noise.

Boosting reduces the bias and bagging/random forest reduces the variance.

“Entities are not to be multiplied beyond necessity.” – Occam’s razor

## Main reading

- “Probabilistic Machine Learning: An Introduction” by Murphy  
[ch. 4.7.6, 5.2.3, 18]

---

Some figures are from Murphy’s “Probabilistic Machine Learning: An Introduction”.