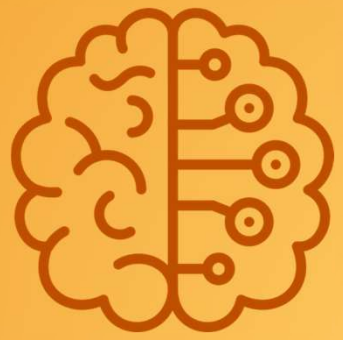
CS 4375 – Introduction to Machine Learning

Bias/Variance Tradeoff & Bagging

Erick Parolin

THE UNIVERSITY OF TEXAS AT DALLAS



[Based on the slides from Dr. Nicholas Ruozzi]

Previously…

- **PAC-learnability:** C is said to be *PAC-learnable* by L using H if and only if learner L will with probability at least $1-\delta$ output a hypothesis $h \in H$, such that:
  - $\mathbf{error_{\mathcal{D}}(h) \leq \varepsilon}$
  - **polynomial time in $1/\varepsilon$, $1/\delta$, n and $size(c)$**
- So, that's what we want: $\mathbf{P(error_{\mathcal{D}}(h) > \varepsilon) \leq \delta}$

Says: we are willing to tolerate a δ probability of having > ε true error.

- PAC-learnability is mostly concerned about the number of training examples required
  - Parameters $\boldsymbol{\varepsilon}$ and $\boldsymbol{\delta}$ will determine the number of training examples
  - The number of examples with respect to $\boldsymbol{\varepsilon}$ and δ must be polynomial
- **Theorem (Haussler, 1988):** For finite H and D ($m \geq 1$ independent random examples), the probability that the version space $VS_{H,D}$ is **not** ε-exhausted is less than or equal to $\mathbf{|H|e^{-\varepsilon m}}$

$$P\left(\{\exists h \in VS_{H,D} \mid error_{\mathcal{D}}(h) > \varepsilon\}\right) \leq |H|e^{-\varepsilon m} \quad \text{or simply} \quad \boldsymbol{P(error_{\mathcal{D}}(h) > \varepsilon) \leq |H|e^{-\varepsilon m}}$$

- **<u>From PAC</u>**: $P(error_{\mathcal{D}}(h) > \varepsilon) \leq |H|e^{-\varepsilon m} \leq \delta$

Previously…

- From PAC and Haussler:

$$P\left(\left\{\exists\, \mathbf{h} \in VS_{\mathbf{H},D} \mid error_{\mathcal{D}}(h) > \varepsilon\right\}\right) \leq |H|e^{-\varepsilon m} \leq \delta$$

- Which gives us the **sample complexity**:

$$m \geq (\ln 1/\delta + \ln |H|)/\varepsilon$$

  - This tells us how many training examples suffice to ensure (with probability (1 - **δ**)) that every hypothesis in H having zero training error will have a true error of at most **ε**.

- **Agnostic Learning** assumes no prior commitment about whether or not $c \in H$:

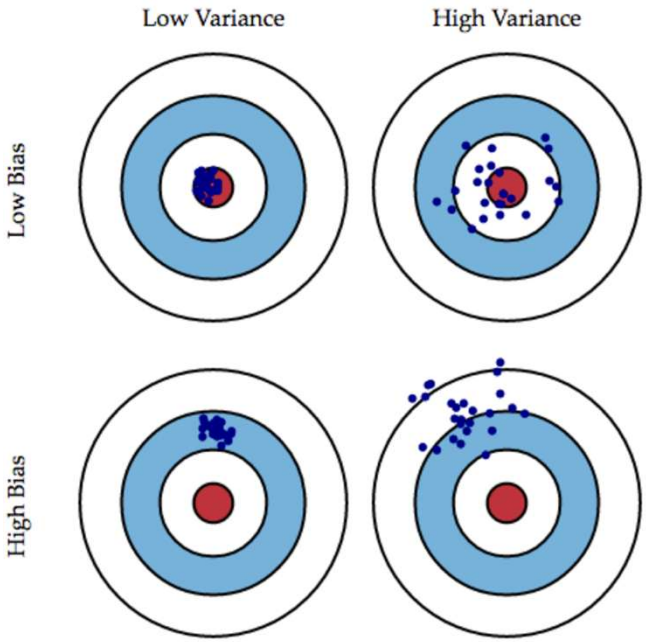
$$P\left(error_{\mathcal{D}}(h_{best}) > \varepsilon + error_{D}(h_{best})\right) \leq |H|e^{-2m\varepsilon^2}$$

- Sample Complexity: $\mathbf{m} \geq \frac{1}{2\varepsilon^2}\left(ln\frac{1}{\delta} + ln|H|\right)$

- For all h, with probability at least $1 - \delta$: $error_{\mathcal{D}}(h_{best}) \leq \underbrace{error_{D}(h_{best})}_{\textbf{Bias}} + \underbrace{\sqrt{\frac{ln\frac{1}{\delta} + ln|H|}{2m}}}_{\textbf{Variance}}$

Intuition

- **Bias**
  - Measures the accuracy or quality of the learner algorithm
  - High bias means a poor match
- **Variance**
  - Measures the precision or specificity of the match
  - High variance means a weak match
- We would like to minimize each of these
- Unfortunately, we can't do this independently, there is a tradeoff



Bias-Variance Analysis in Regression

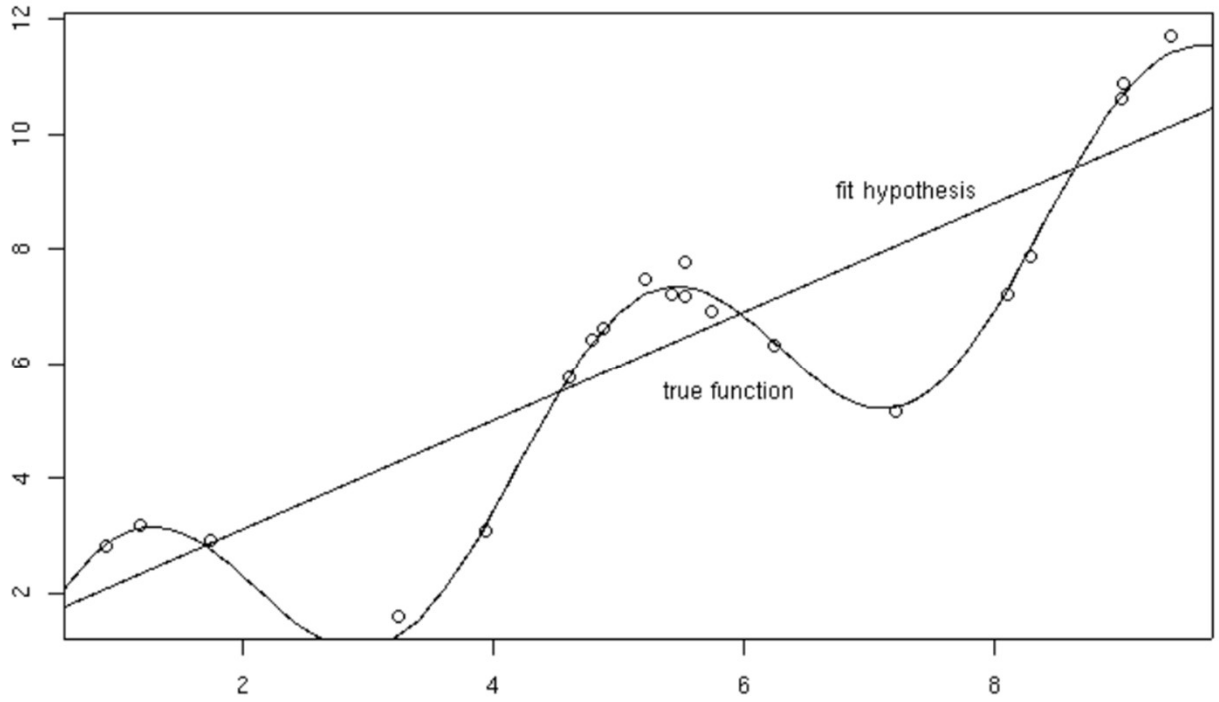
- **True function is** $y = f(x) + \epsilon$
  - Where noise, ϵ, is normally distributed with zero mean and standard deviation $\boldsymbol{\sigma}$

- Given a set of training examples, $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$, we fit a hypothesis $\boldsymbol{g(x) = w^T x + b}$ to the data to minimize the squared error

$$\sum_{\boldsymbol{i}} [\boldsymbol{y^{(i)} - g(x^{(i)})}]^{\mathbf{2}}$$

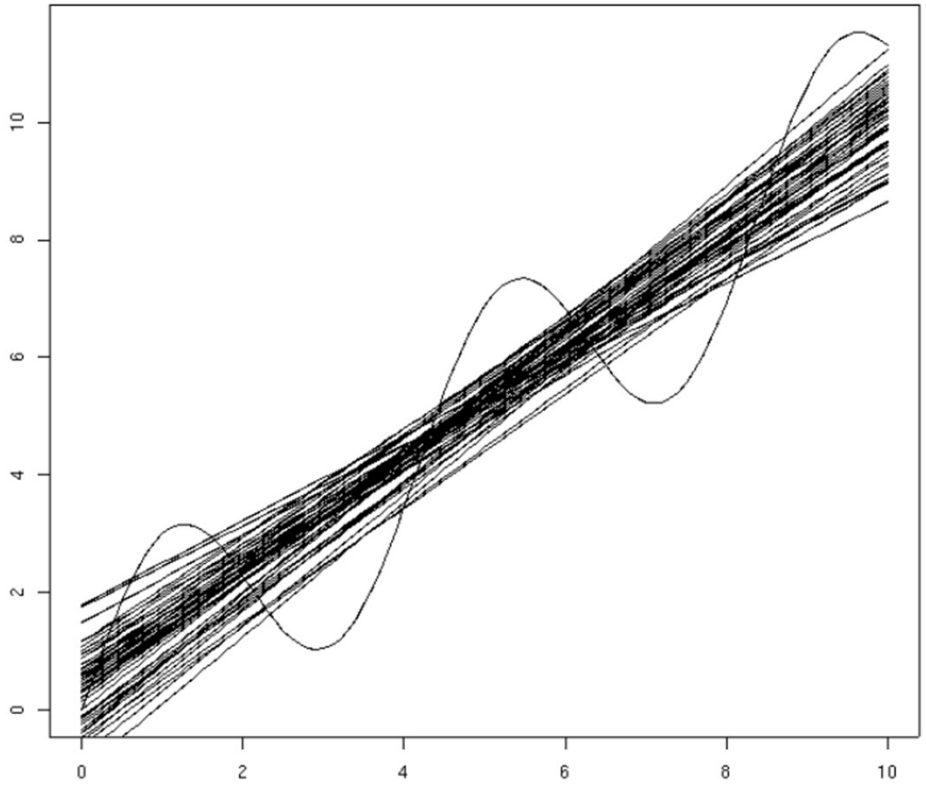
2-D Example

Sample 20 points from $f(x) = x + 2\sin(1.5x) + N(0,0.2)$



2-D Example

50 fits (20 examples each)



Bias-Variance Analysis

- Given a new data point x' with observed value $y' = f(x') + \epsilon$, want to understand the **expected prediction error**.
- Suppose that training samples are drawn independently from a distribution $p(S)$, want to compute the **expected error** of the estimator

$$\boldsymbol{E}[(\boldsymbol{y'} - \boldsymbol{g_S}(\boldsymbol{x'}))^2]$$

Probability Reminder

- Variance of a random variable, Z

$$
\begin{aligned}
Var(Z) &= E[(Z - E[Z])^2] \\
&= E[Z^2 - 2ZE[Z] + E[Z]^2\ ] \\
&= E[Z^2] - E[Z]^2
\end{aligned}
$$

Bias-Variance-Noise Decomposition

$$E\left[\left(y' - g_S(x')\right)^2\right] = E[g_S(x')^2 - 2g_S(x')y' + y'^2]$$

$$= E[g_S(x')^2] - \boxed{2E[g_S(x')]E[y']} + E[y'^2]$$

Notes:

- $E[A + B] = E[A] + E[B]$
- $E[AB] = E[A] \cdot E[B]$ only if the two random variables are statistically independent
- The samples S and the noise ϵ are independent, so $\boldsymbol{g_S(x')}$ **and** $\boldsymbol{y'}$ **are independent**

Bias-Variance-Noise Decomposition

$$E\left[\left(y' - g_S(x')\right)^2\right] = E[g_S(x')^2 - 2g_S(x')y' + y'^2]$$

$$= E[g_S(x')^2] - 2E[g_S(x')]E[y'] + E[y'^2]$$

$$= Var\left(g_S(x')\right) + E[g_s(x')]^2 - 2E[g_S(x')]f(x')$$

$$+ Var(y') + f(x')^2$$

Notes:

- Remember: $y' = f(x') + \epsilon$ where ϵ is normally distributed with zero mean → $E[y'] = f(x')$
- From definition of Variance: $Var[Z] = E[Z^2] - E[Z]^2$
- $Var\left(g_S(x')\right) = E[g_S(x')^2] - E[g_S(x')]^2 \Rightarrow \boldsymbol{E\left[g_S(x')^2\right] = Var\left(g_S(x')\right) + E[g_S(x')]^2}$
- Similarly: $Var(y') = E\left[y'^2\right] - E[y']^2 \Rightarrow \boldsymbol{E\left[y'^2\right] = Var(y') + f(x')^2}$

Bias-Variance-Noise Decomposition

$$E\left[\left(y' - g_S(x')\right)^2\right] = E[g_S(x')^2 - 2g_S(x')y' + y'^2]$$

$$= E[g_S(x')^2] - 2E[g_S(x')]E[y'] + E[y'^2]$$

$$= Var\left(g_S(x')\right) + E[g_s(x')]^2 - 2E[g_S(x')]f(x')$$

$$+ Var(y') + f(x')^2$$

$$= Var\left(g_S(x')\right) + \left(E[g_s(x')] - f(x')\right)^2 + Var(\epsilon)$$

Notes:

- $\boldsymbol{E[g_S(x')]^2 - 2E[g_S(x')]f(x') + f(x')^2 = \left(E[g_S(x')] - f(x')\right)^2}$
- $\boldsymbol{Var(y') = Var(f(x') + \epsilon) = \mathrm{Var}(\epsilon) = \sigma^2}$

Bias-Variance-Noise Decomposition

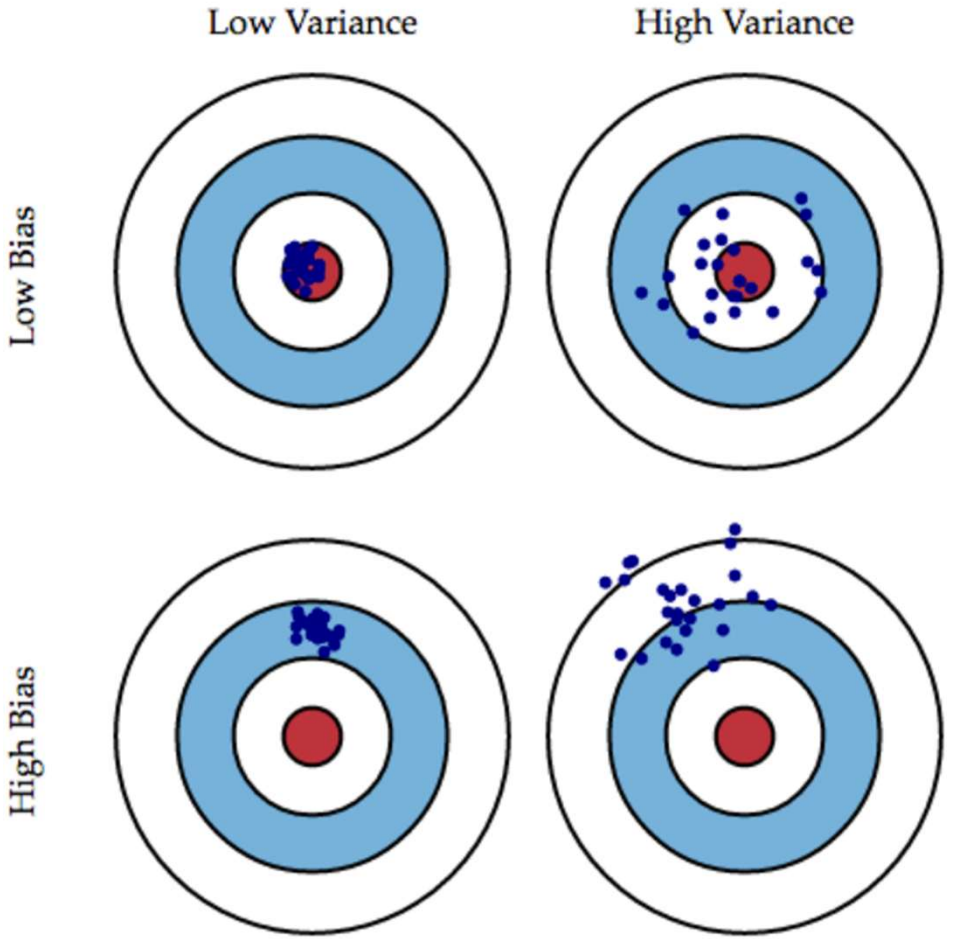
$$
\begin{aligned}
E\left[\left(y' - g_S(x')\right)^2\right] &= E[g_S(x')^2 - 2g_S(x')y' + y'^2] \\
&= E[g_S(x')^2] - 2E[g_S(x')]E[y'] + E\left[y'^2\right] \\
&= Var\left(g_S(x')\right) + E[g_s(x')]^2 - 2E[g_S(x')]f(x') \\
&\quad + Var(y') + f(x')^2 \\
&= Var\left(g_S(x')\right) + \left(E[g_s(x')] - f(x')\right)^2 + Var(\epsilon) \\
&= \underbrace{Var\left(g_S(x')\right)}_{\text{Variance}} + \underbrace{\left(E[g_s(x')] - f(x')\right)^2}_{\text{Bias}} + \underbrace{\sigma^2}_{\text{Noise}}
\end{aligned}
$$

Bias, Variance, and Noise

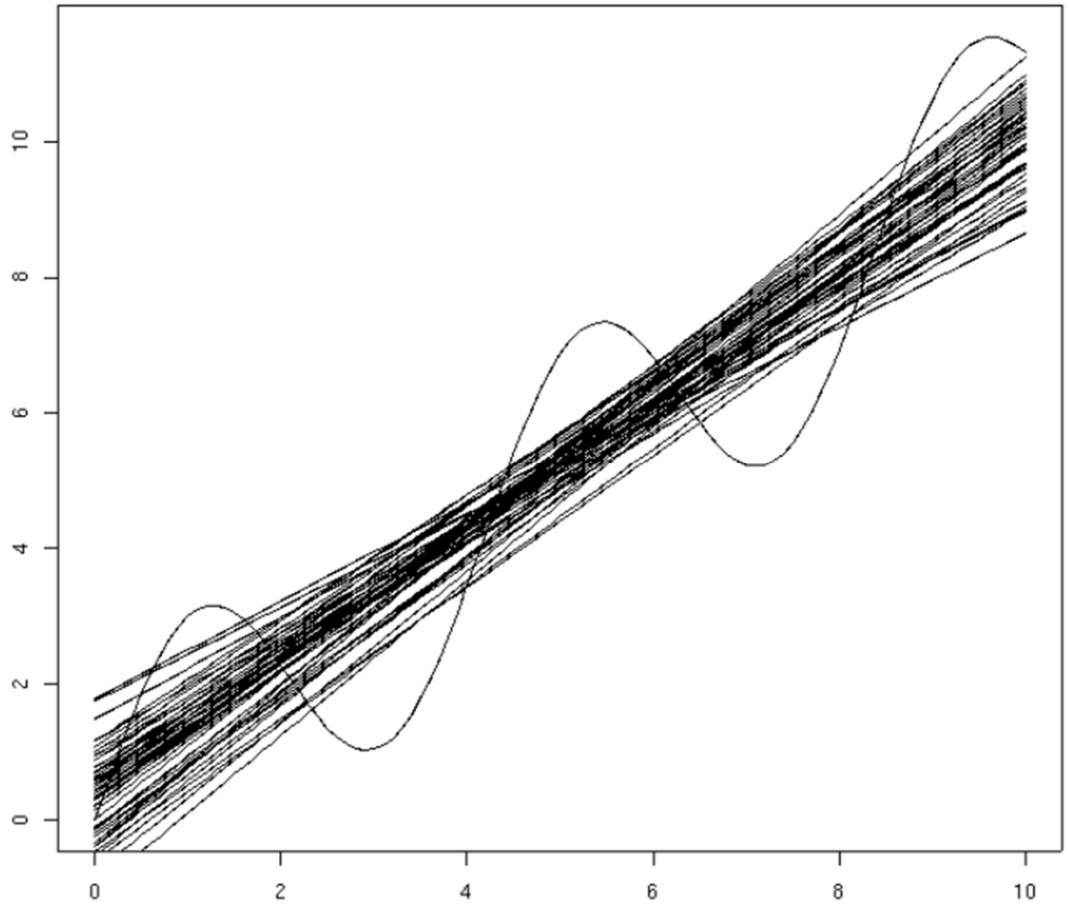
- Variance: $\boldsymbol{E}\left[\left(\boldsymbol{g}_{\boldsymbol{S}(\boldsymbol{x}')} - \boldsymbol{E}[\boldsymbol{g}_{\boldsymbol{S}(\boldsymbol{x}')}]\right)^{\boldsymbol{2}}\right]$
  - Describes how much $\boldsymbol{g}_{\boldsymbol{S}}(\boldsymbol{x}')$ varies from one training set $\boldsymbol{S}$ to another
- Bias: $\boldsymbol{E}[\boldsymbol{g}_{\boldsymbol{S}}(\boldsymbol{x}')] - \boldsymbol{f}(\boldsymbol{x}')$
  - Describes the average error of $g_S(x')$
- Noise: $\boldsymbol{E}\left[\left(\boldsymbol{y}' - \boldsymbol{f}(\boldsymbol{x}')\right)^{\boldsymbol{2}}\right] = \boldsymbol{E}[\boldsymbol{\epsilon}^{\boldsymbol{2}}] = \boldsymbol{\sigma}^{\boldsymbol{2}}$
  - Describes how much y' varies from $f(x')$

Bias, Variance, and Noise

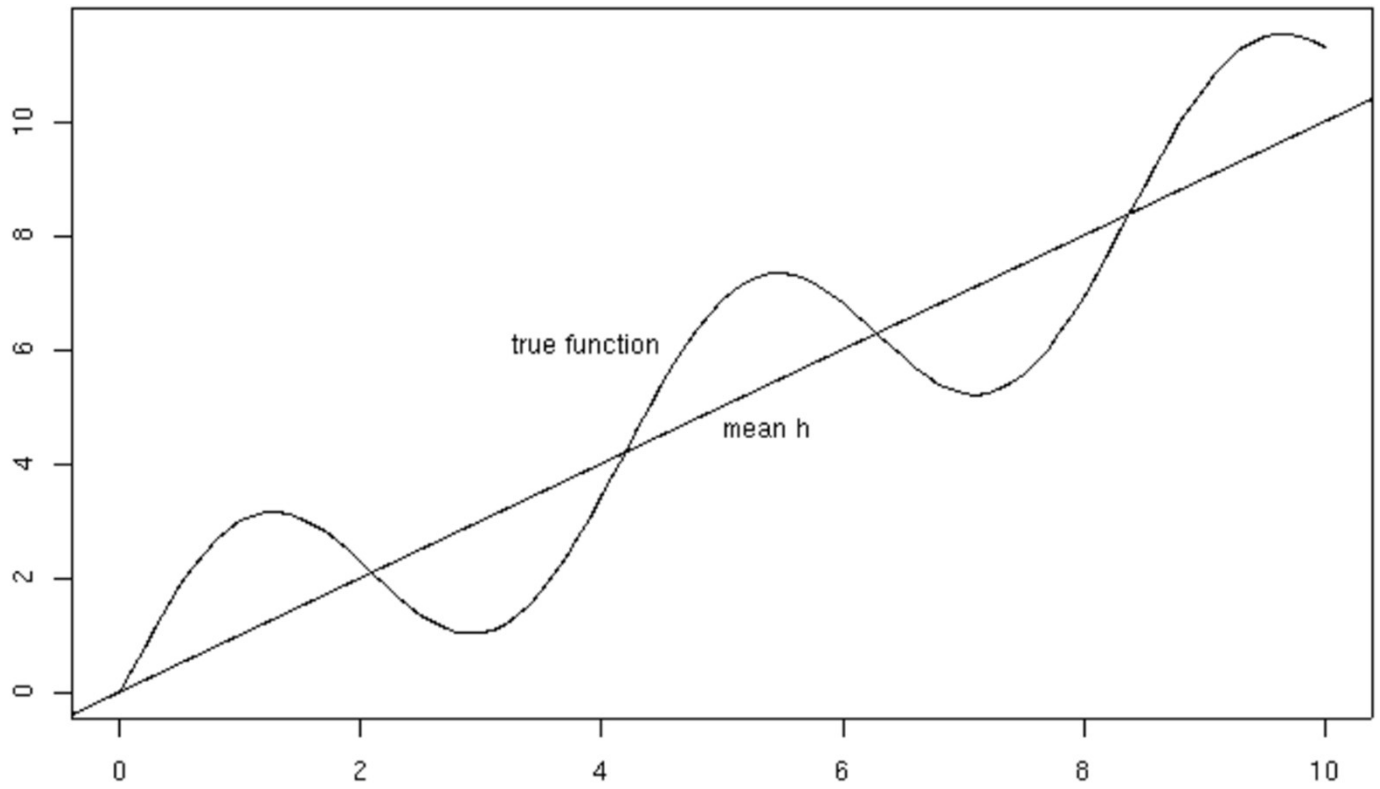
$$\textbf{Error} = Bias^2 + Variance + Noise$$



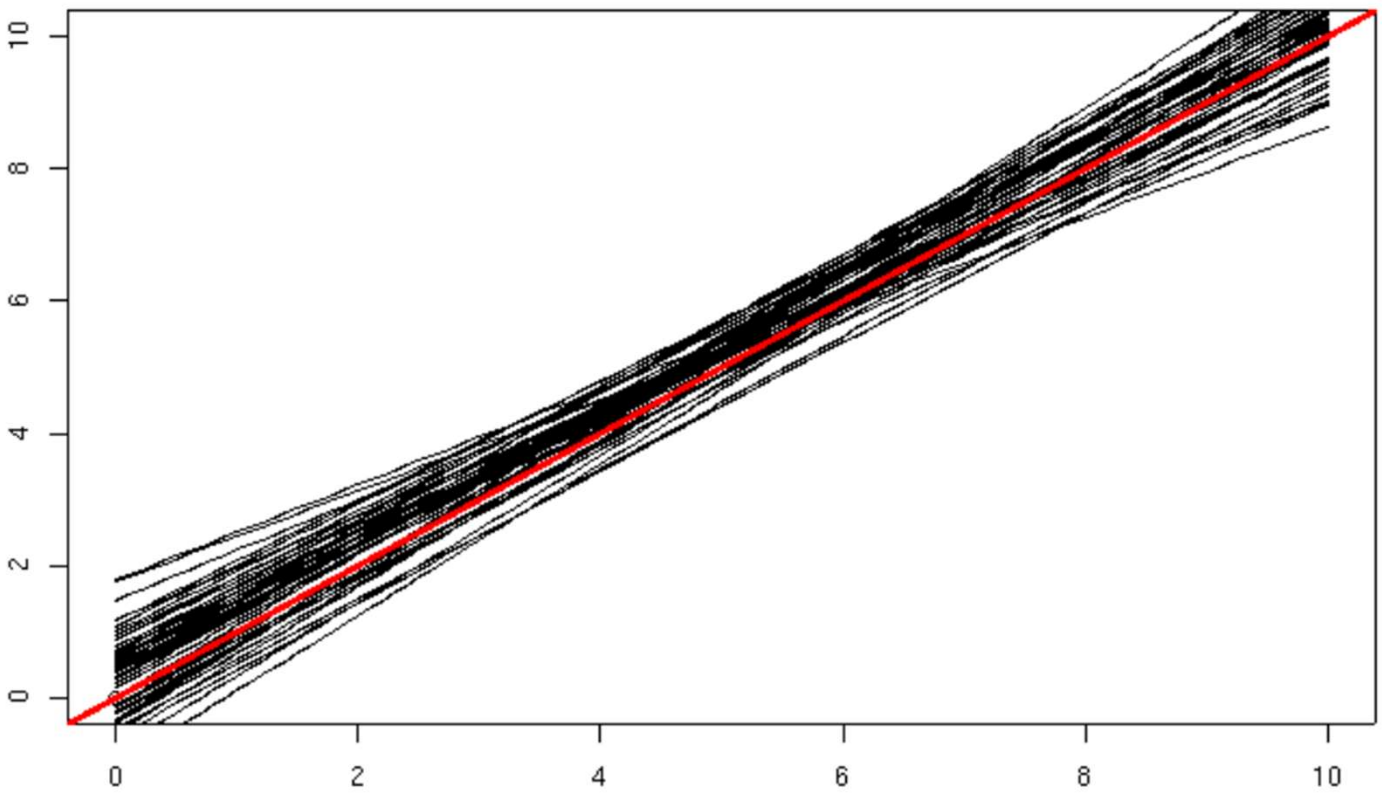
50 fits (20 examples each)



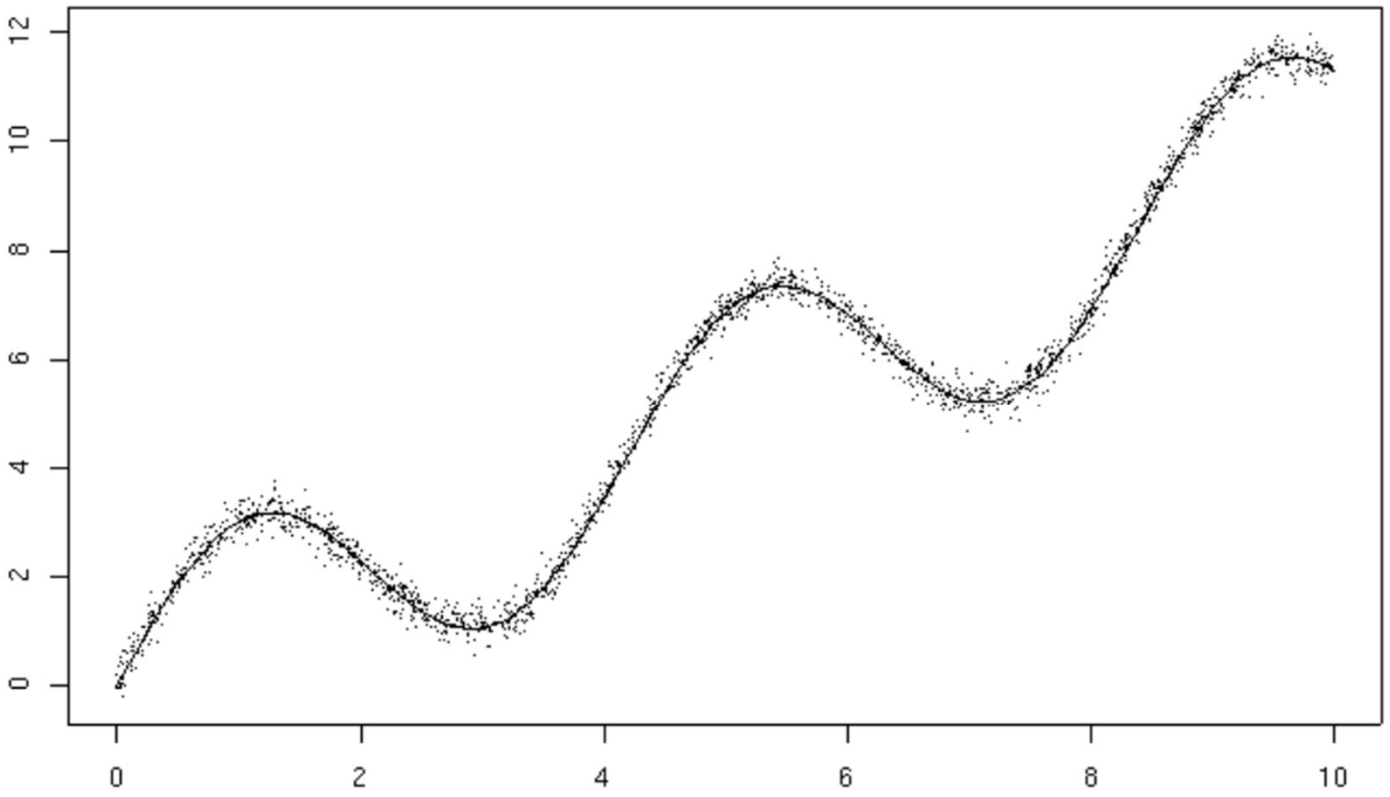
Bias



Variance



Noise



Bias

- **Low bias**
  - Linear regression applied to linear data
  - 2nd degree polynomial applied to quadratic data
- **High bias**
  - Constant function
  - Linear regression applied to highly non-linear data

Variance

- **Low variance**
  - Constant function
  - Simple model independent of training data
- **High variance**
  - Complex model, e.g., neural nets
  - High degree polynomial

Bias/Variance Tradeoff

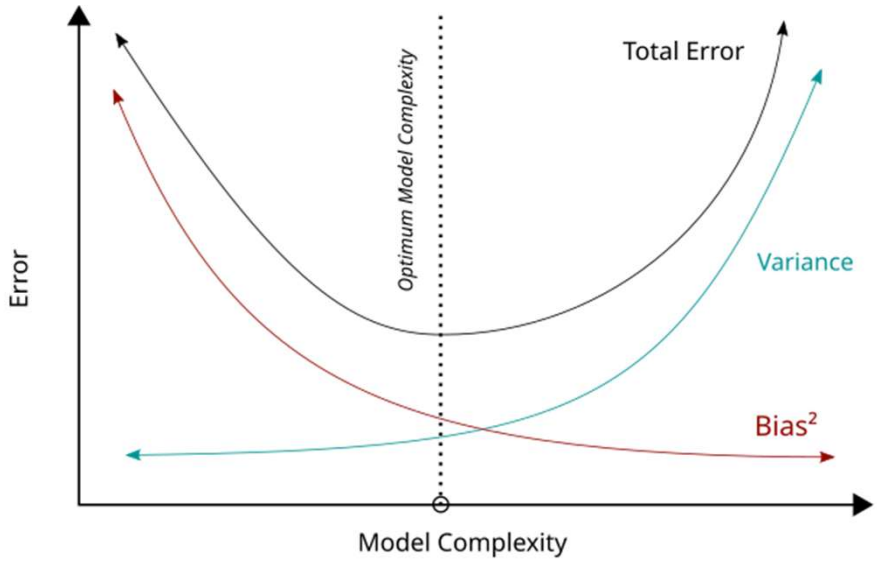
- **To get low bias**
  - You need to **approximate** as much as possible to the **real function**.
  - Usually requires a **complex model** that has larger number of parameters.
    - Higher complexity
  - Parameters are learned based on **training data**
  - So, for **each sample**, you are likely to get a **different model**.
  - Different models for different samples ⇒ **high variance**
  - **So, you have a low bias with high variance model.**

Bias/Variance Tradeoff

- **To get low variance**
  - You need a model more "**stable**", whose output doesn't change much on different samples.
  - A very simple model (e.g., a constant) **doesn't change its outputs for different samples.**
    - Very low complexity
  - But the output will be very **different from the real function** that you are trying to approximate ⇒ **high bias**
  - **So, you have a low variance with high bias model.**

Bias/Variance Tradeoff

- **(bias² + variance) is what counts for prediction**
- **Often:**
  - Low bias ⇒ high variance
  - Low variance ⇒ high bias
- **Tradeoff**
  - **bias2 vs. variance**
- **Is this a firm rule?**



Reduce Variance Without Increasing Bias

- **Averaging reduces variance:**

$$\boldsymbol{Var}\left(\frac{\boldsymbol{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i}}\boldsymbol{z_i}\right)=\frac{\boldsymbol{1}}{\boldsymbol{N}}\boldsymbol{Var}(\boldsymbol{Z})$$

where $Z = z_1, \ldots, z_N$ is i.i.d. random variables

- Problem:
  - Only one training set
  - Where do multiple models come from?

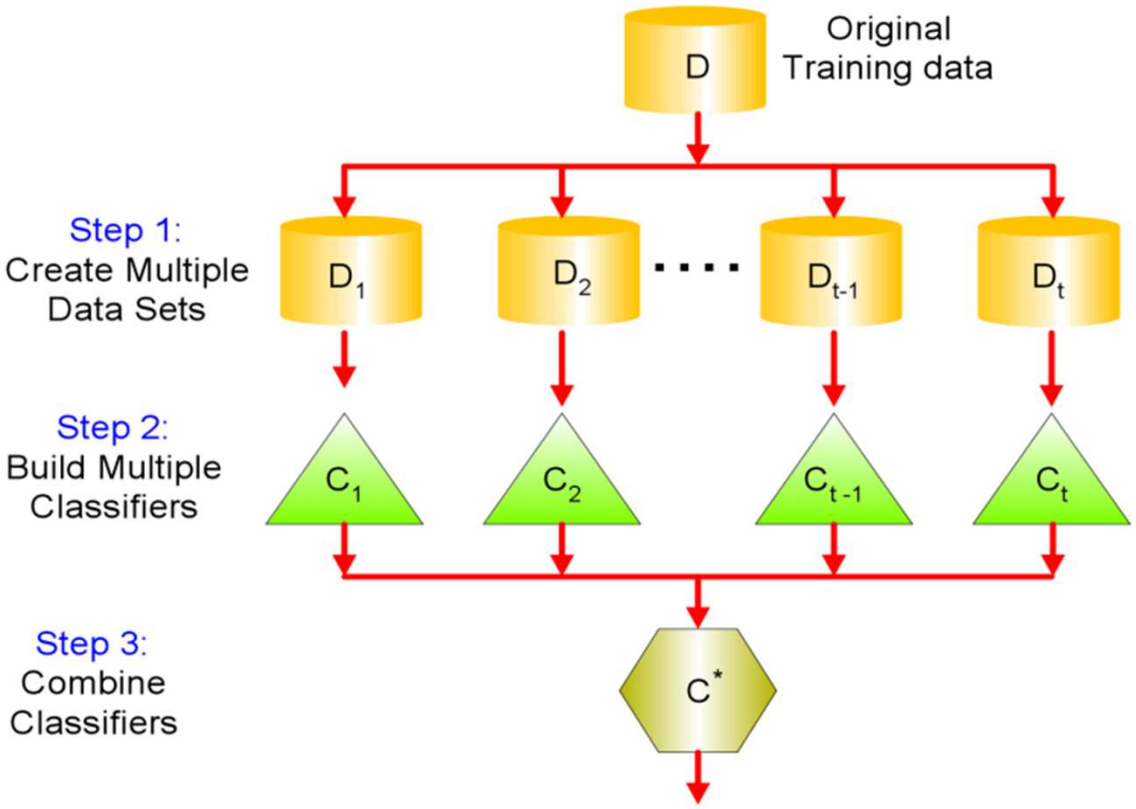
Bagging: Bootstrap Aggregation

- Take repeated bootstrap samples from training set D (Breiman, 1994)

- **Bootstrap sampling:** Given set **D** containing **N** training examples, create D' by drawing N examples at random with replacement from D

- **Bagging**:
  - Create $\boldsymbol{k}$ bootstrap samples $\boldsymbol{D_1}, \ldots, \boldsymbol{D_k}$
  - Train distinct classifier on each $\boldsymbol{D_i}$
  - Classify new instance by majority vote / average

Bagging: Bootstrap Aggregation



Bootstrap Sampling

- In each bootstrap sample, each data point has probability $\left(1-\frac{1}{\mathrm{N}}\right)^{\mathrm{N}}$ of **not** being selected
- So, probability of being selected is $1-\left(1-\frac{1}{\mathrm{N}}\right)^{\mathrm{N}}$
- Expected number of distinct data points in each sample is then

$$N \cdot \left(1-\left(1-\frac{1}{\mathrm{N}}\right)^{\mathrm{N}}\right) \approx N \cdot \left(1-exp(-1)\right) = 0.632N$$

Remember: e^x can be written as $\exp(\mathrm{x}) = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n$

Analysis

- **Bagging:** Output an aggregated model with less variance.
- **Ideally:**

$$\boldsymbol{Var}\left(\boldsymbol{Bagging}\left(\boldsymbol{L}(\boldsymbol{x},\boldsymbol{D})\right)\right) = \frac{\boldsymbol{Var}\left(\boldsymbol{L}(\boldsymbol{x},\boldsymbol{D})\right)}{\boldsymbol{N}}$$

- **In practice:**
  - Models are correlated, so reduction is smaller than 1/N

Bagging Empirical Results

Experiments design:

- Many datasets were split into Train and Test.
- Two learners/classifiers were tested:
  - Decision Tree
  - Bagging (using Decision Tree)
- Observed error (misclassification rate):
  - $\bar{e}_S$ is the error for Decision Tree learner.
  - $\bar{e}_B$ is the error for Bagging.
- Experiments were repeated 100 times, and the reported $\bar{e}_S$ and $\bar{e}_B$ are the average over the 100 iterations.

Data Set	$\bar{e}_S$	$\bar{e}_B$	Decrease
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

Breiman “Bagging Predictors” Berkeley Statistics Department TR#421, 1994

When Will Bagging Improve Accuracy?

- Depends on the stability of the base-level classifiers
- A learner is **unstable** if a small change to the training set causes a large change in the output hypothesis
  - If small changes in *D* cause large changes in the output, then there will likely be an improvement in performance with bagging
- Bagging can help unstable procedures, but could hurt the performance of stable procedures
  - Decision trees are unstable
  - K-Nearest Neighbor and Naïve Bayes are stable classifiers

Random Forests

- Remember from couple of slides ago: "*In practice, models are correlated, so reduction is smaller than 1/N*"
- So, why not introduce some more randomness?
- Build large collection of de-correlated trees and average them.
- How to make the trees de-correlated?

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Random Forests

- Ensemble method specifically designed for decision tree classifiers
- Introduce two sources of randomness: “bagging” and “random input vectors”
- Bagging method: each tree is grown using a bootstrap sample of training data
- **Random vector method:** best split at each node is chosen from a random sample of *m* attributes instead of all attributes

Random Forest = Bagging + Random Vector Method

Readings

- **Pattern Recognition and Machine Learning by Christopher M. Bishop – Chapter 14**
- **Bagging Predictors by Leo Breiman (1996)**