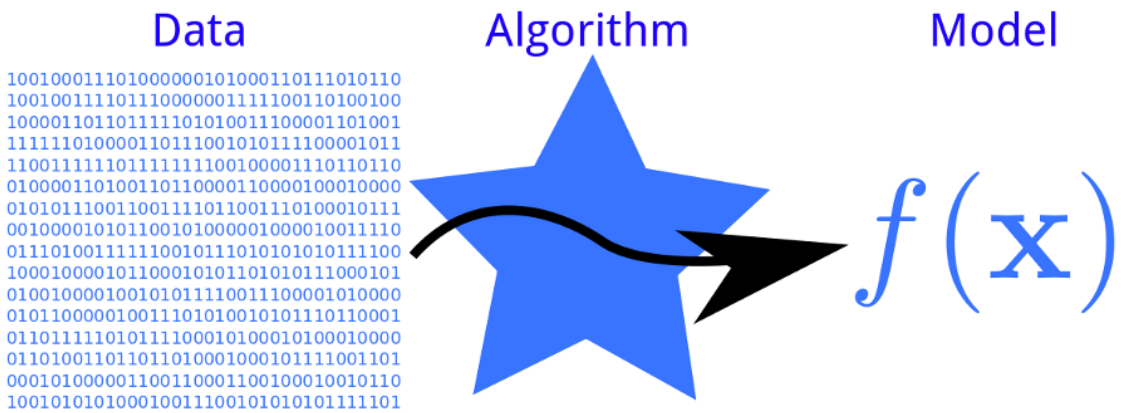
Ensemble Methods

Background

- We have studied various models of classification.
- What are we really aiming for?
  - Estimation of the real underlying function f: X -> Y
  - Aiming for good generalization results
  - Finding best set of parameters
- What we don't want to do
  - Overfit the training data
  - Don't care for generalization
  - Find parameters that are suited just for a particular dataset
  - Have too many/ too few parameters
- Another way to say this – we want to model the data generation process, and not just a model of the training data.



Task: Error Estimation

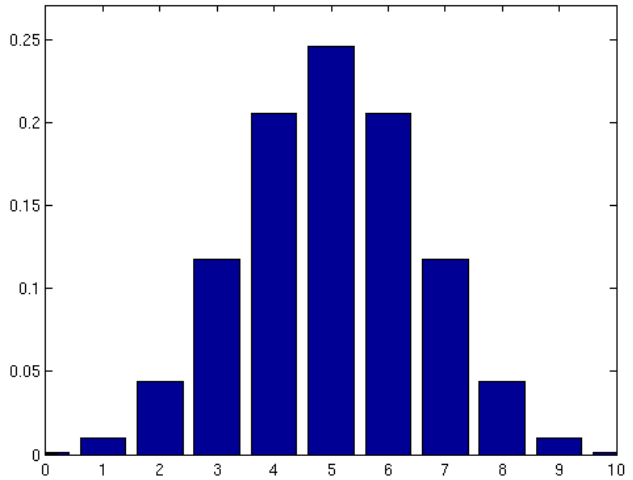
- Given a hypothesis h, I would like to know how well it performs.
- More technically, I would like to find the probability that the hypothesis will make a mistake on a random point.
- How do I measure this? Have we seen a similar problem before?

Task: Error Estimation

- Yes, we have seen a situation like this before.
- Given a coin, task was to estimate its true probability of heads (p).
- How did we proceed?
- Perform experiments, where each time you toss the cc count the number of heads, r.

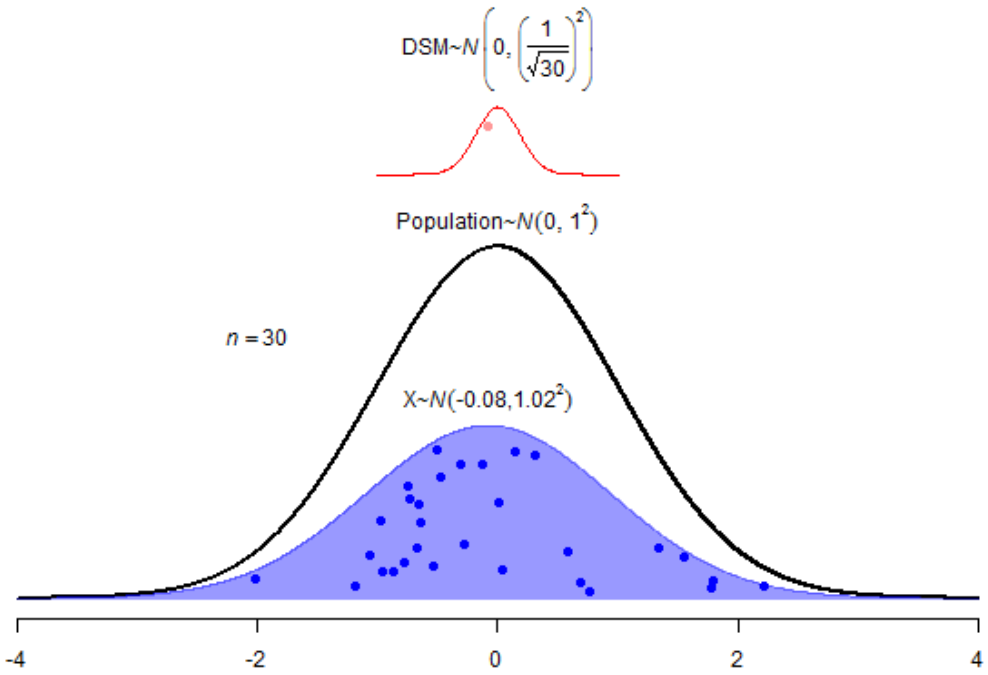
  The estimate from one such experiment = $\frac{r}{n}$

- Multiple such experiments would yield a Binomial Distribution



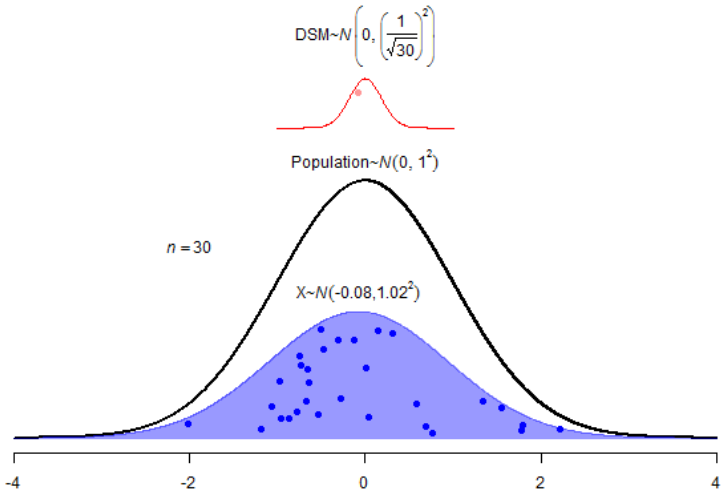
Background

- In ML, training data is obtained by sampling from a population distribution **D**.
- Example:
  - Sample students from entire UTD population
  - Create a sample of users that have clicked on a Google ad.
  - Sample of Facebook users.
- Given a hypothesis h, I would like to test how good it is i.e. its error on any randomly drawn point from **D**



True Error

- I am really looking for the true error, which is defined as the likelihood that the hypothesis will misclassify a data instance. (Equivalent to the true probability of a coin)



Definition: The **true error** (denoted ***error$_D$(h))*** of hypothesis ***h*** with respect to target function f and distribution D is the probability that h will misclassify an instance drawn at random from D

$$error_D(h) = \Pr_{x \in D}[f(x) \neq h(x)]$$

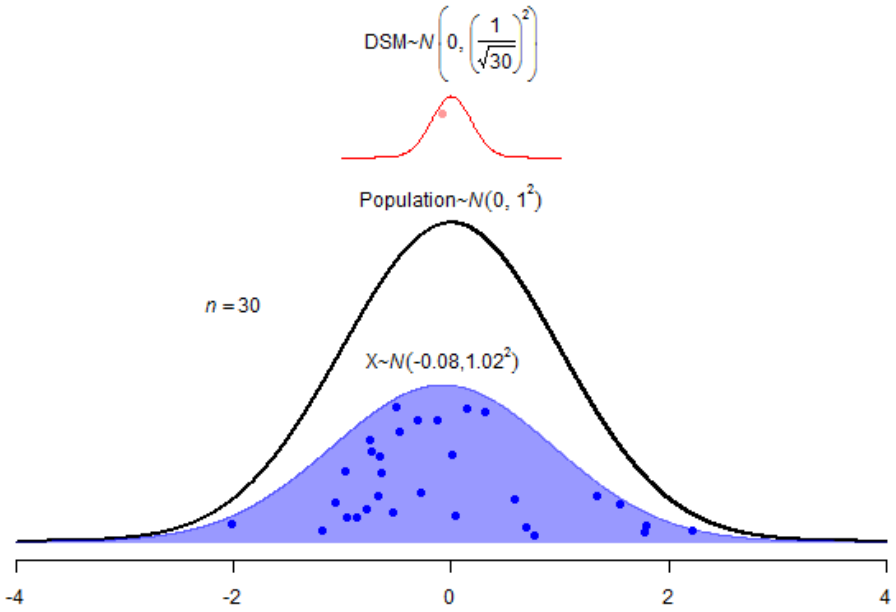
Background

- If we check the hypothesis on a sample, we get sample error. (Observable)

Definition: The **sample error** (denoted ***error$_s$(h))*** of hypothesis ***h*** with respect to target function f and data sample **S** is

$$error_S(h) = \frac{1}{n}\sum_{x \in S} \delta(f(x), h(x))$$

where ***n*** is the number of examples in **S,** and the quantity δ(f (x),h(x)) is 1 if f(x) ≠ h(x), and 0 otherwise.



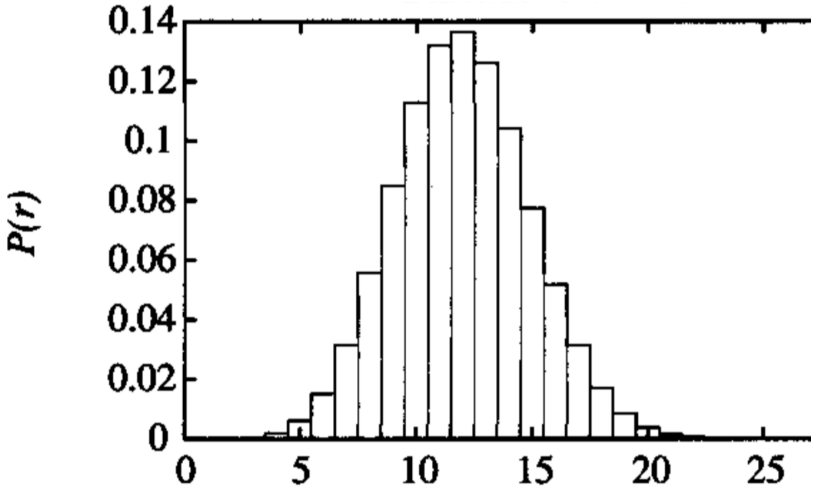
Sampling Theory

- When we measure the sample error of a hypothesis h, we are performing a random Binomial experiment.
- Outcome of the random experiment – sample error
  [Mitchell's book Chapter 5]
- We first collect a random sample **S** of ***n*** independently drawn instances from the distribution ***D,*** and then measure the sample error $\boldsymbol{error_s(h)}$
- If we were to repeat this experiment many times, each time drawing a different random sample ***Si*** of size n, we would expect to observe different values for the various $\boldsymbol{error_{si}(h)}$.

Remarks

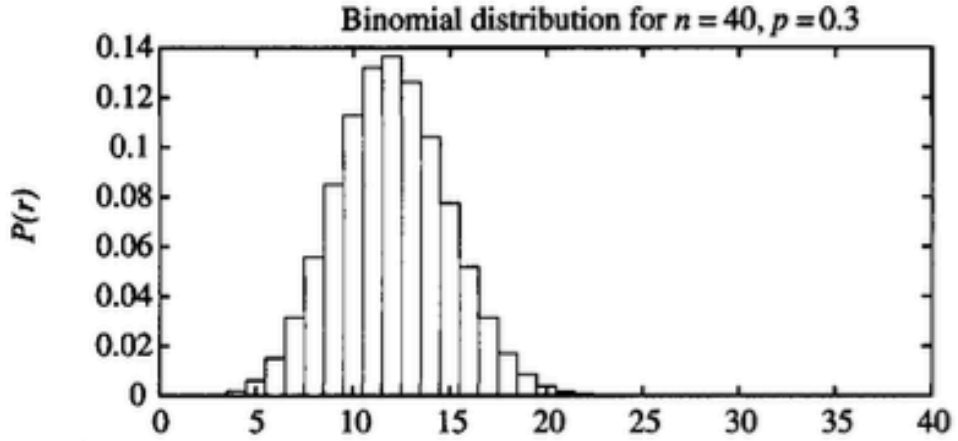
- Note that we don’t know the value of the real error of the hypothesis.
- We are simply testing the hypothesis on multiple samples to get a random output variable each time.
- Does this remind you of something?
- You have a coin whose probability of heads (p) is unknown.
- You toss it many times to estimate the value of p.
- $error_s(h)$ -> observable [Binomial Distribution]
  $error_D(h)$ -> equivalent to p (you want to find this)

Sampling Theory



- Imagine that we were to run k such random experiments, measuring the random variables $error_{s1}(h)$, $error_{s2}(h)$,...,$error_{sk}(h)$.
- Suppose we then plotted a histogram displaying the frequency with which we observed each possible error value.
- As we allowed k to grow, the histogram would approach the form of the distribution shown on next slide, called as the **Binomial Distribution**

Binomial Distribution



A *Binomial distribution* gives the probability of observing r heads in a sample of n independent coin tosses, when the probability of heads on a single coin toss is p. It is defined by the probability function

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

If the random variable X follows a Binomial distribution, then:

- The probability $\Pr(X = r)$ that X will take on the value r is given by $P(r)$
- The expected, or mean value of X, $E[X]$, is

$$E[X] = np$$

- The variance of X, $Var(X)$, is

$$Var(X) = np(1-p)$$

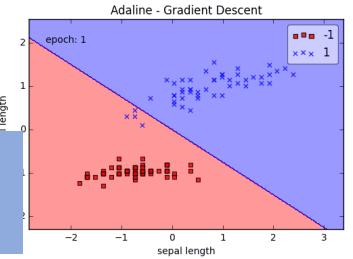
- The standard deviation of X, σ_X, is

$$\sigma_X = \sqrt{np(1-p)}$$

For sufficiently large values of n the Binomial distribution is closely approximated by a Normal distribution (see Table 5.4) with the same mean and variance. Most statisticians recommend using the Normal approximation only when $np(1-p) \geq 5$.

Point to note: Number of heads (r) has a Binomial distribution.

Comparison of two scenarios



Estimating P(H) of a coin

- Coin with unknown probability of heads **p**
- Binomial Experiments:

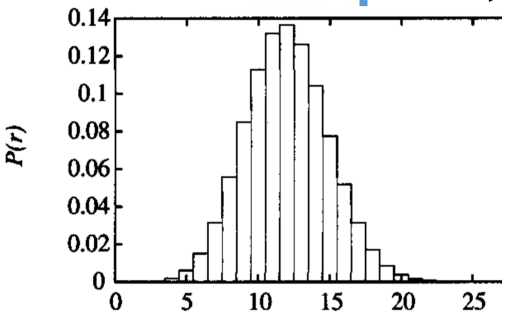
  E_1: Toss the coin n_1 times, observe number of heads r_1 => 1st estimate $y_1 = \frac{r_1}{n_1}$

  E_2: Toss the coin n_2 times, observe number of heads r_2 => 2nd estimate $y_2 = \frac{r_2}{n_2}$

  .....

  E_k: Toss the coin n_k times, observe number of heads r_k => kth estimate $y_k = \frac{r_k}{n_k}$

A histogram of p's would yield such a curve in each case



Estimating true error

- Classifier with unknown true error: **$error_D(h)$**
- Binomial Experiments:

  E_1: Sample n_1 items from the population, observe number of misclassifications r_1

  => 1st estimate $y_1 = \frac{r_1}{n_1}$

  E_2: Sample n_2 items from the population, observe number of misclassifications r_2

  => 2nd estimate $y_2 = \frac{r_2}{n_2}$

  .....

  E_k: Sample n_k items from the population, observe number of misclassifications r_k

  => kth estimate $y_k = \frac{r_k}{n_k}$

Estimator and Bias

- We really want $error_D(h)$, and we are estimating it by $error_S(h)$
- Statisticians call $error_S(h)$ as the **estimator** of $error_D(h)$.
- By doing the experiment multiple times, we get an estimate of the parameter as: E[Y], where E[Y] is the **expected value** of Y i.e. average of Y over multiple values.

$$E[Y] = \sum_i p_i Y_i$$

Note if all experiments are equally likely, $E[Y] = \bar{Y}$

- For our case since the number of errors is Binomial, E[r] = np so E[y] = E[r/n] = 1/n E[r] = p.

How good is the estimate?

- Bias measures the difference between expected value of an estimator (E[Y]) and the "true" value (p).

$$\text{Bias} = E[Y] - p$$

A lower bias is preferred

- What is expected value of $error_s(h)$

$$E[error_s(h)] = E\left[\frac{r}{n}\right] = \frac{1}{n}E[r] = \frac{1}{n}(np) = p$$

We used the fact that the number of heads obtained (r) is a Binomial and E[r] = np

- So, bias of error = 0

How good is the estimate?

- Variance measures how much the estimate of error varies between samples. A low variance is generally preferred.

We used the fact that the number of heads obtained (r) is a Binomial and E[r] = np

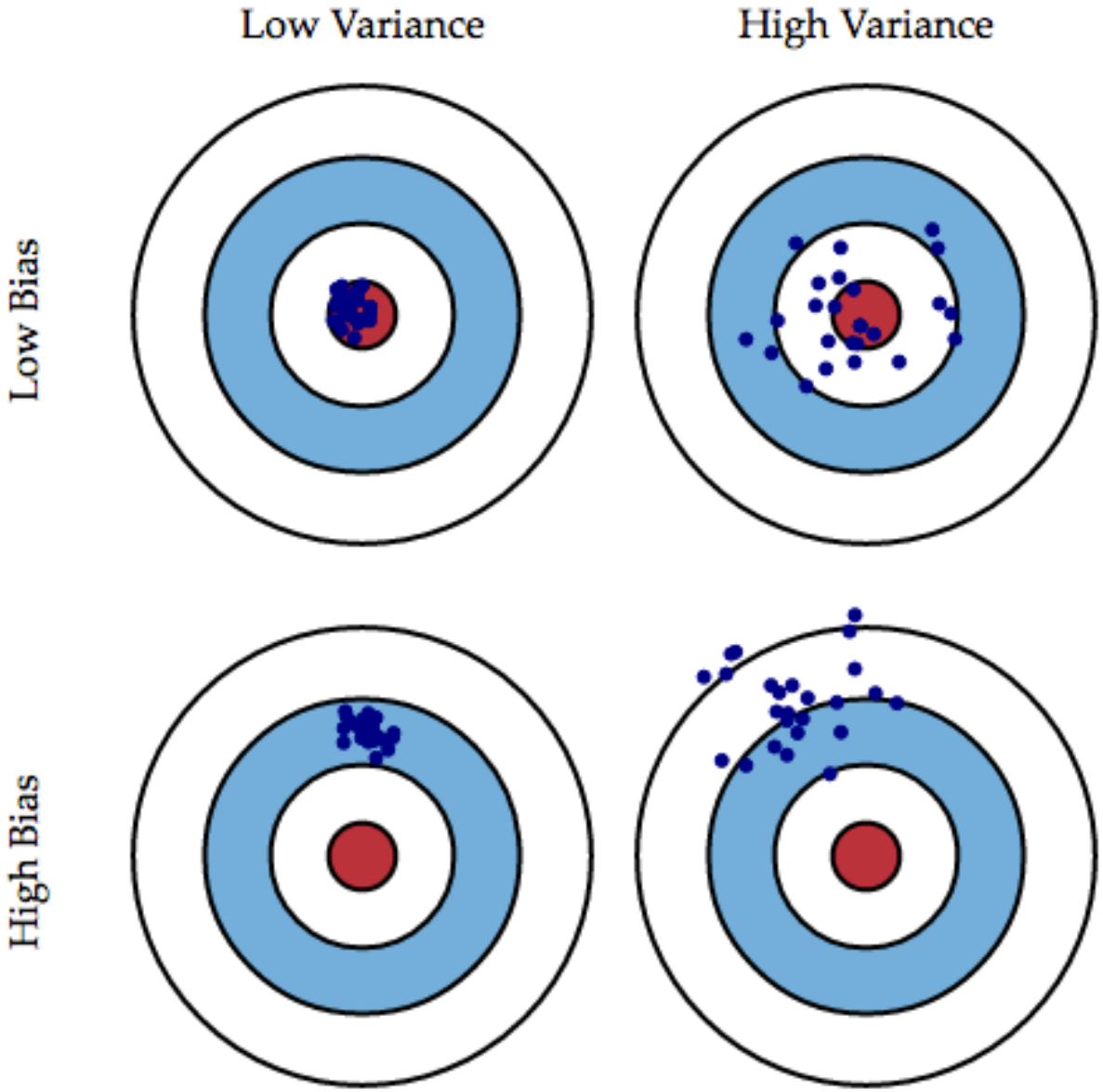
- What is the variance of $error_s(h)$

$$\text{var}[error_s(h)] = var\left[\frac{r}{n}\right] = \frac{1}{n} var[r] = \frac{1}{n}(np(1-p)) = p(1-p)$$

Intuition

- Bias:
  - measures accuracy of quality of an algorithm.
  - high bias means a poor match.
- Variance:
  - measures the precision or specificity of a match
  - Did the algorithm just get lucky on a few samples? Or is it really a good classifier?
  - a high variance means a weak match.
- Ideally, we would like to minimize both of them. Is that possible?

Bias-Variance



Mathematical Derivation

- Let's consider the case of regression
- True function is:

$$y = f(x) + \epsilon$$

where

y is the output,
f(x) is the "best" possible function
ϵ is a term that accounts for the noise in data

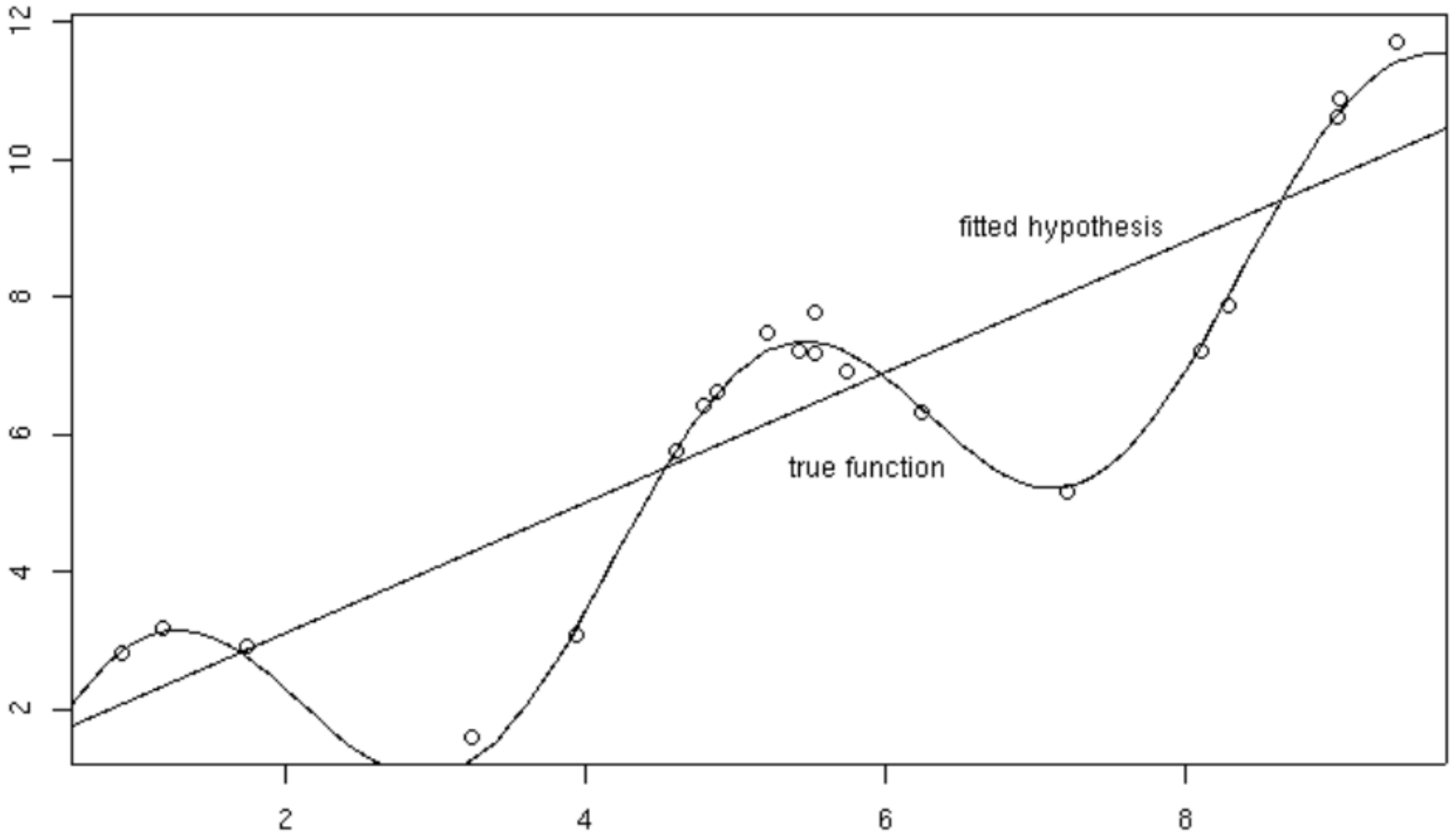
We are assuming that there is some noise in the data.

- Our hypothesis is $h(x) = w^T x + b$
- We seek to minimize squared error:

$$E = \sum_i [y_i - h(x_i)]^2$$

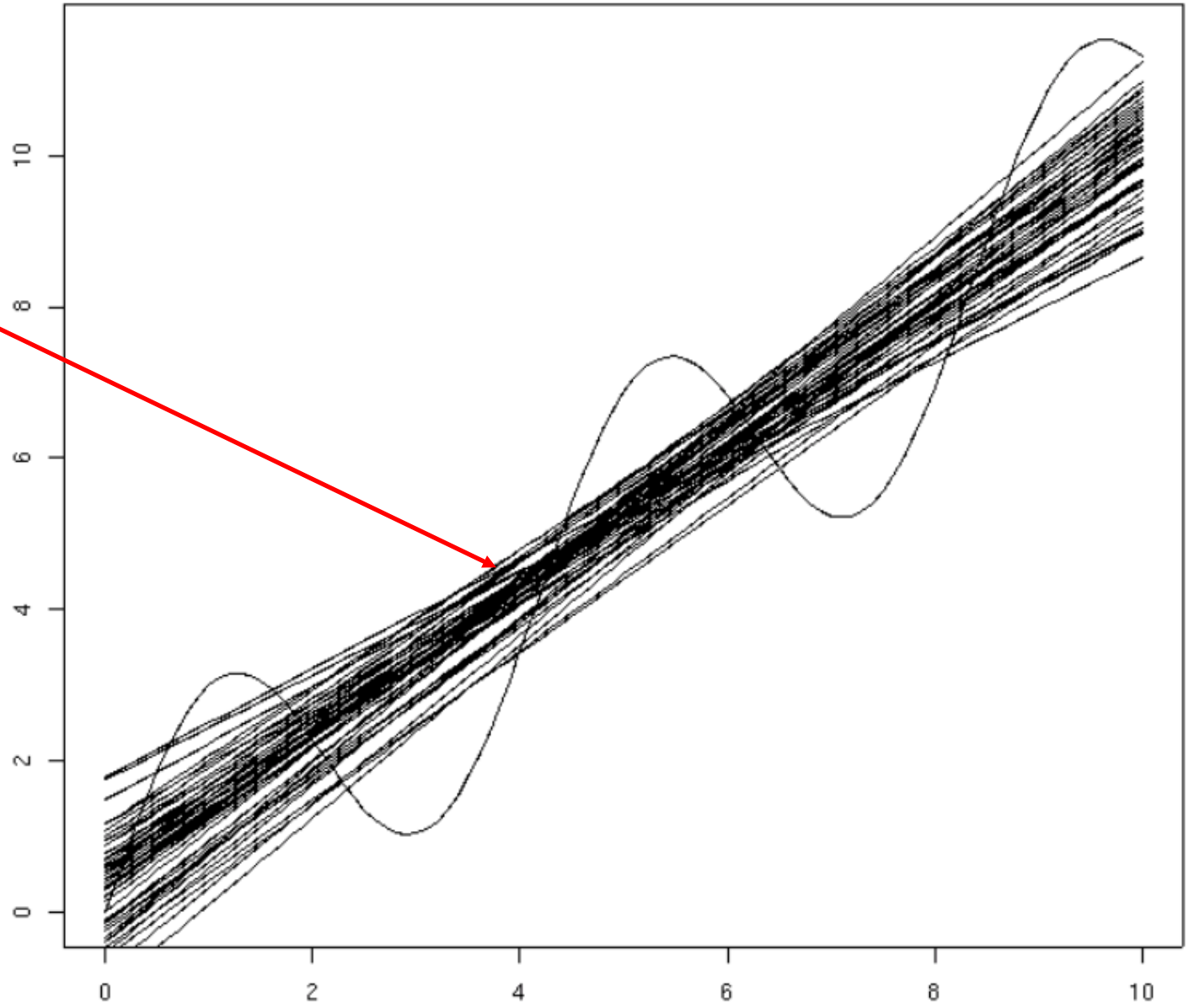
Example: 20 points

$y = x + 2 \sin(1.5x) + N(0,0.2)$ Noise term



50 fits (20 examples each)

50 samples of 20 instances each are sampled and hypothesis is tested on them.



Bias – Variance Analysis

- Now, given a new data point x*, with observed value $y^* = f(x^*) + \epsilon$
- Prediction of our hypothesis = h(x*)
- Let's examine expected value of squared error:

$$E[(h(x^*) - y^*)^2]$$

- Before going there, let's look at a lemma that will help us.

Bias – Variance Analysis

- Let Z be a random variable with distribution P(Z) and average value defined as:

$$\bar{Z} = E_P[Z]$$

- Lemma: $E[(Z - \bar{Z})^2] = E[Z^2] - \bar{Z}^2$

- Proof: $E[(Z - \bar{Z})^2] = E[Z^2 + \bar{Z}^2 - 2Z\bar{Z}]$

$$= E[Z^2] + \bar{Z}^2 - 2\bar{Z}E[Z]$$

$$= E[Z^2] - \bar{Z}^2$$

Using linearity property of E and the fact that
E[Z] = $\bar{Z}$; also the fact that
E[C] = C where C is a constant.

- Corollary: $E[Z^2] = E[(Z - \bar{Z})^2] + \bar{Z}^2$

Bias Variance Analysis

- Let's go back to what we were trying to solve:

$$E[(h(x^*) - y^*)^2] = E\left[h(x^*)^2 - 2h(x^*)y^* + y^{*2}\right]$$

$$= E[h(x^*)^2] - 2E[h(x^*)]E[y^*] + E[y^{*2}]$$

$$= \left\{E\left[\left(h(x^*) - \overline{h(x^*)}\right)^2\right] + \overline{h(x^*)}^2\right\} - 2\overline{h(x^*)}\,f(x^*) +$$

$$\{E\left[\left(y^* - f(x^*)\right)^2\right] + f(x^*)^2\}$$

$$= E\left[\left(h(x^*) - \overline{h(x^*)}\right)^2\right] + \left(\overline{h(x^*)} - f(x^*)\right)^2 + E\left[\left(y^* - f(x^*)\right)^2\right]$$

$$= var\left(h(x^*)\right) + Bias\left(h(x^*)\right)^2 + Noise^2$$

We have used the fact that
$E[y^*] = E[\mathrm{f}(\mathrm{x}^*) + \epsilon]$
$= f(x^*)$
as $E[\epsilon] = 0$ and f is a function output [not random variable]

This is the noise term as
$y^* - f(x^*) = \epsilon$

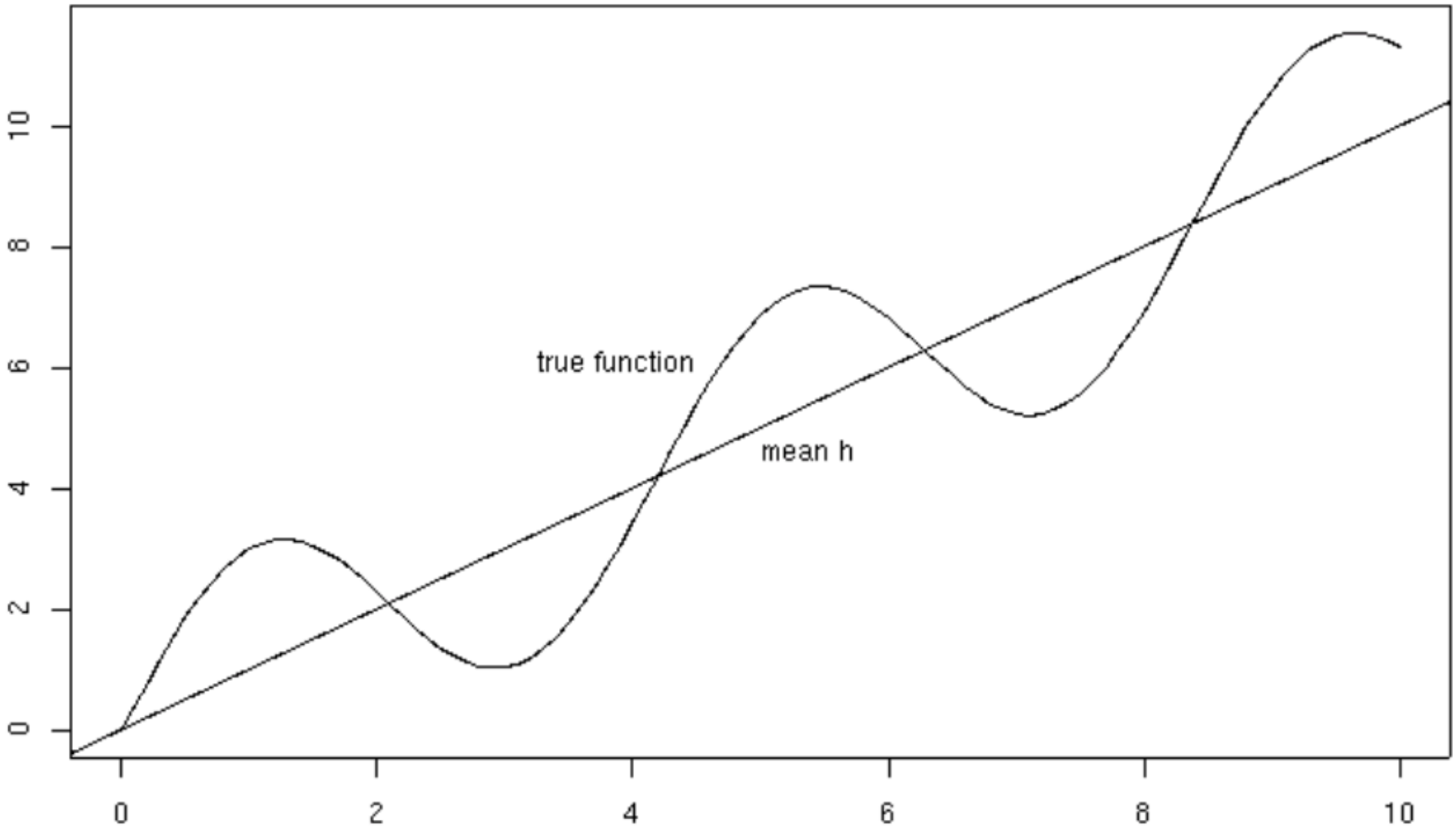
Bias Variance Tradeoff

- Summarizing,

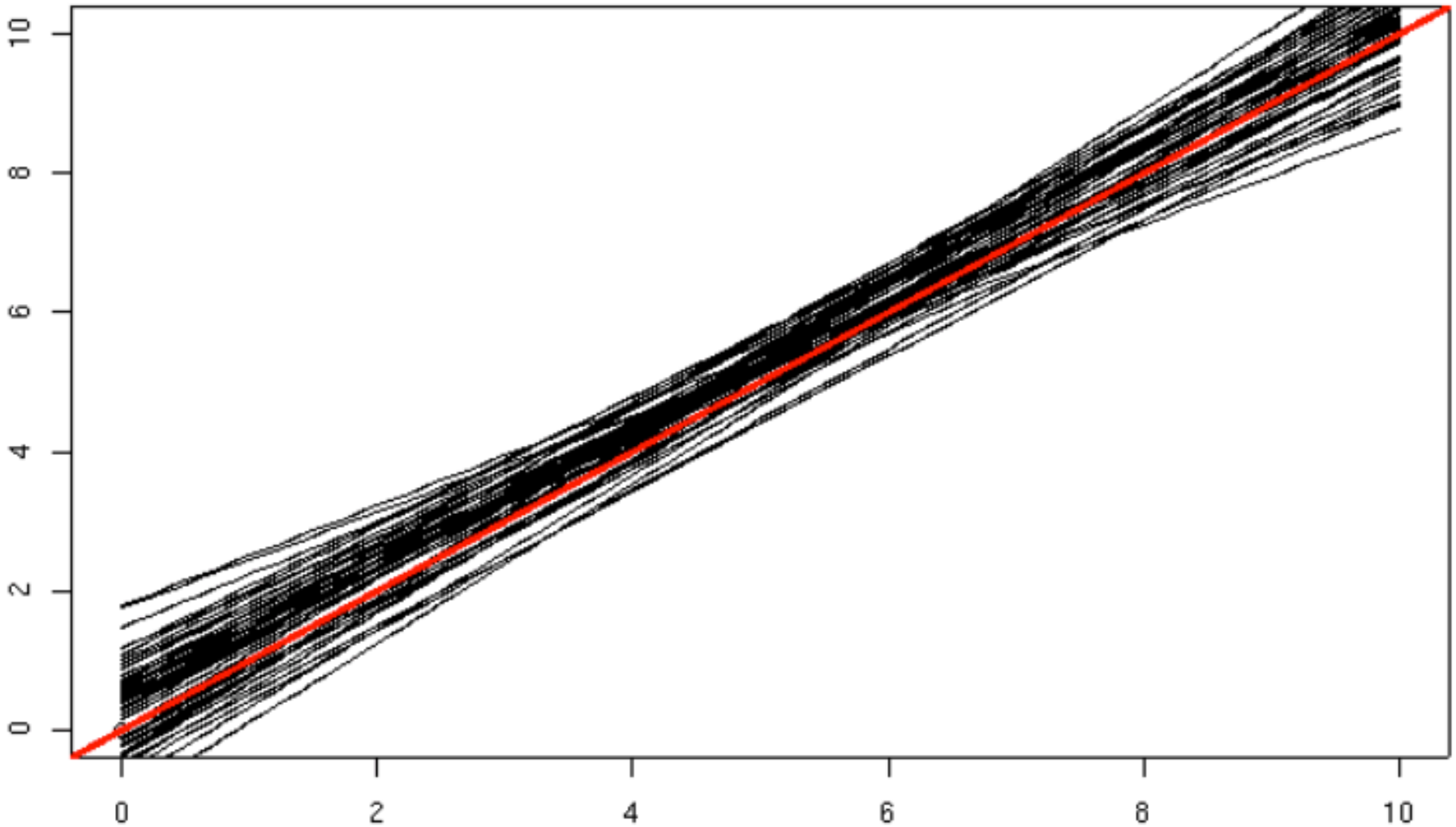
$$E[(h(x^*) - y^*)^2] = var\big(h(x^*)\big) + Bias\big(h(x^*)\big)^2 + Noise^2$$

- Variance = How much h(x*) varies from its average value over multiple samples.
- Bias = Describes average error of h i.e. how far is average error from actual expected value
- Noise = Describes how much the best function differs from actual value

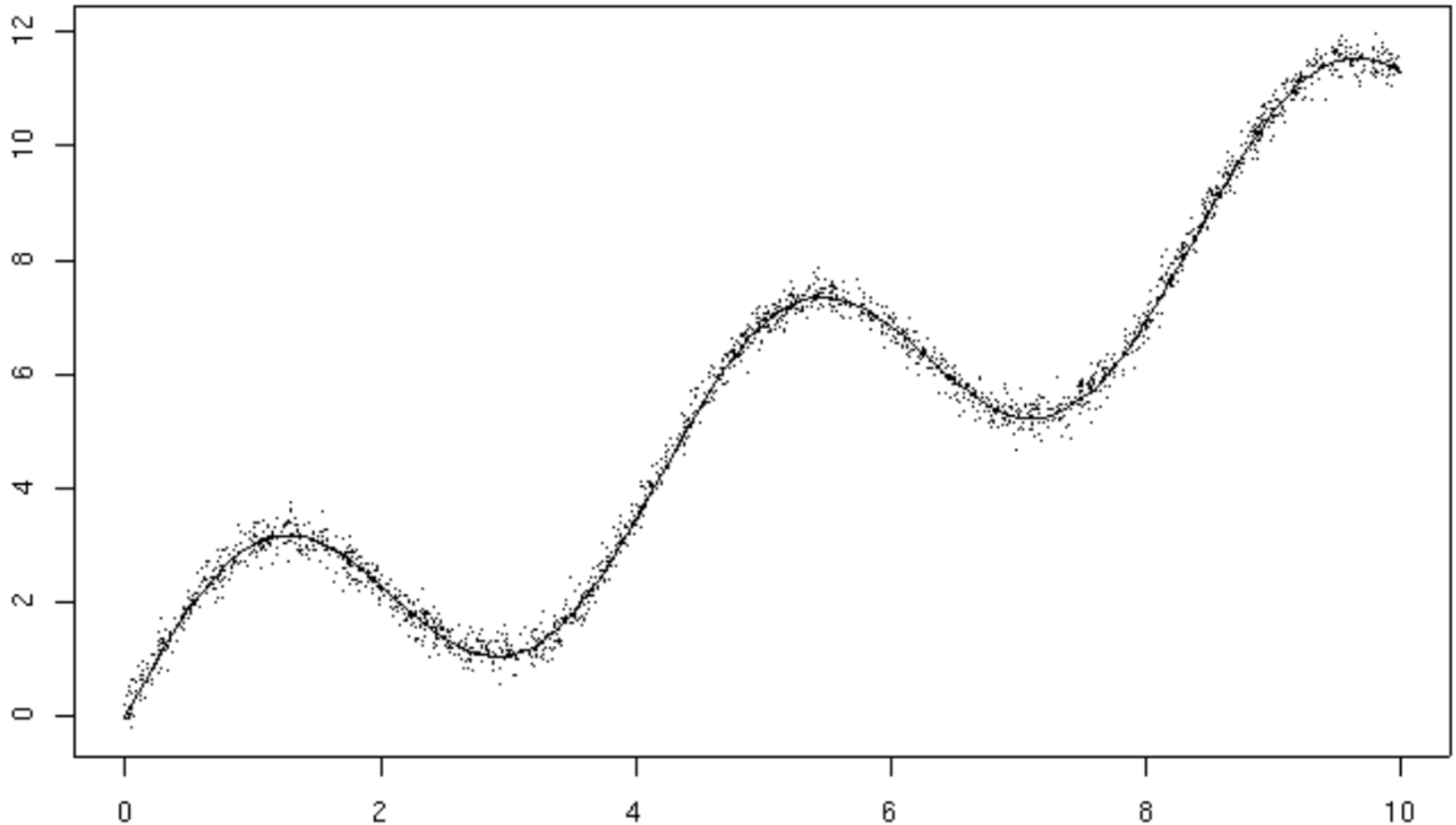
Bias



Variance

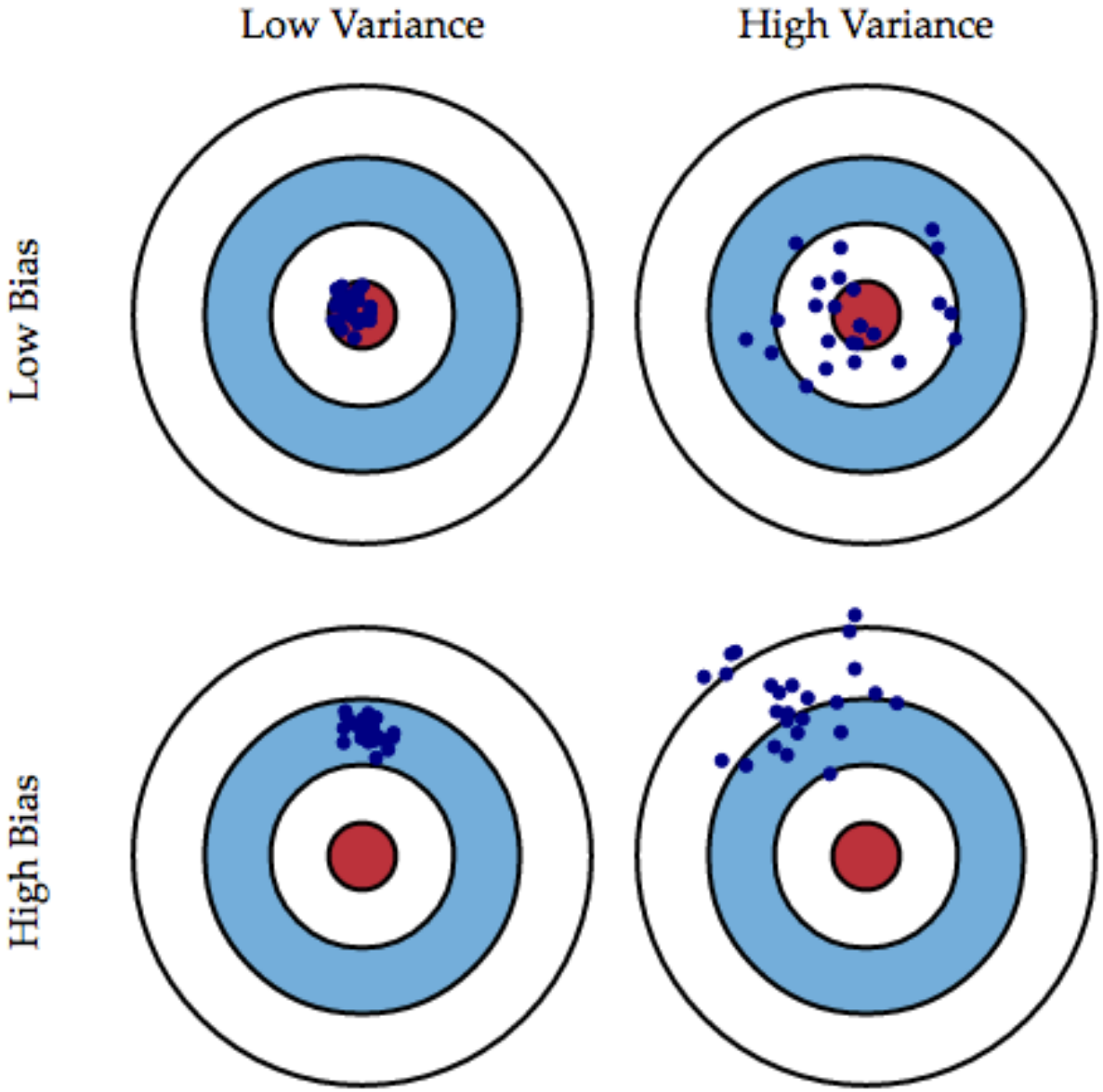


Noise



Bias-Variance

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$



Bias Variance Tradeoff

- For machine learning models, there is always a tradeoff between bias and variance.
- Generally, we can't have low bias and low variance for real world data.
- The next slides explain this concept.

Bias – Variance Tradeoff

- Low Bias – model gets close to actual value
  - linear regression applied to linear data
  - 2nd degree polynomial applied to quadratic data
- High Bias – model is a poor approximator
  - constant function applied to highly non-linear data
  - linear regression applied to non-linear data
- Low Variance – for different samples, output of model doesn't change much
  - constant function always
  - simple model independent of data
- High Variance – for different samples, output of the model changes a lot
  - complex model e.g. neural net
  - high degree polynomial function

Bias Variance Tradeoff

- Can I get both low bias and low variance?
- To get low bias
  -> you need to get very close to the real function.
  -> need a complex model that has a number of parameters.
  -> parameters depend on data
  -> so for each sample, you get a different output from model.
- If you get different output from the model for different samples, you have a high variance.
- So, I get a low bias but ended up with high variance.

Bias Variance Tradeoff

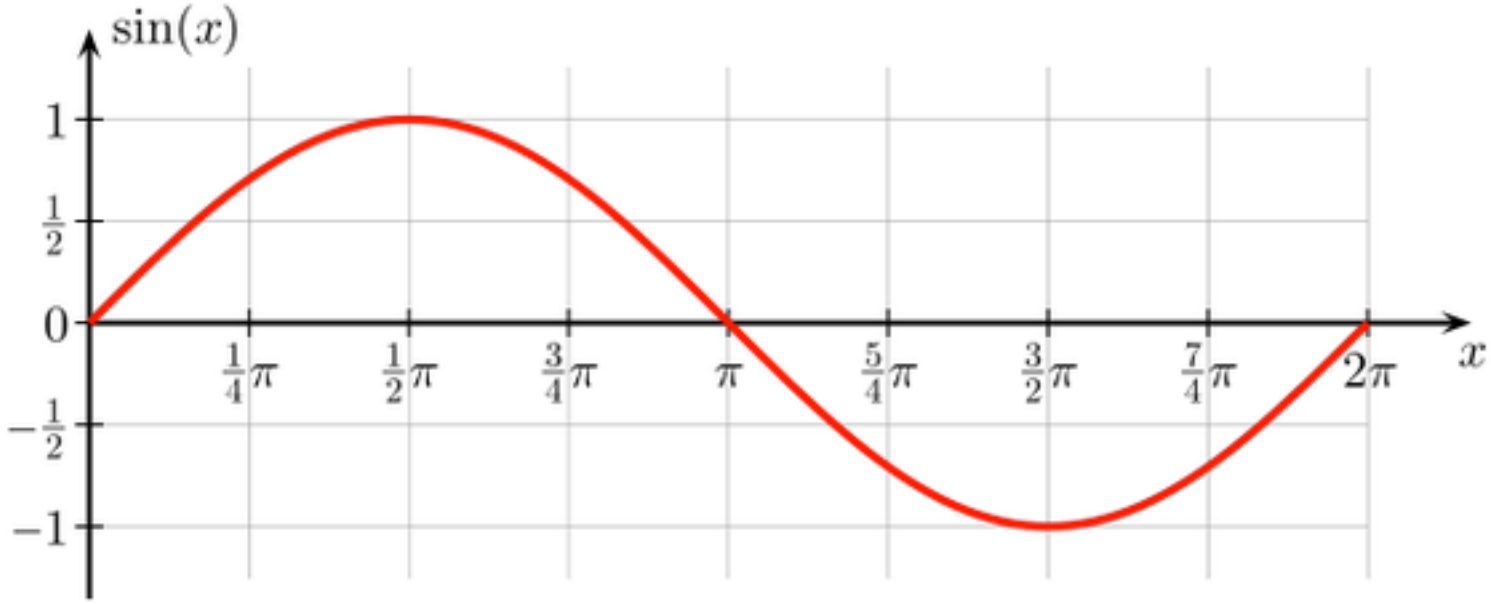
- Can I get both low bias and low variance?
- To get low variance
  -> you need a model whose output doesn't change across multiple samples
  -> Suppose you choose a constant function. It doesn't change its output for different data samples.
- But, the output will be very different from the real function that you are trying to approximate => High Bias
- So, I get a low variance but ended up with high bias.

Why can't we reduce both error and variance

- If we make the mode more complex, bias goes down but variance goes up.
- If we create a very simple model, variance is low but bias goes up.

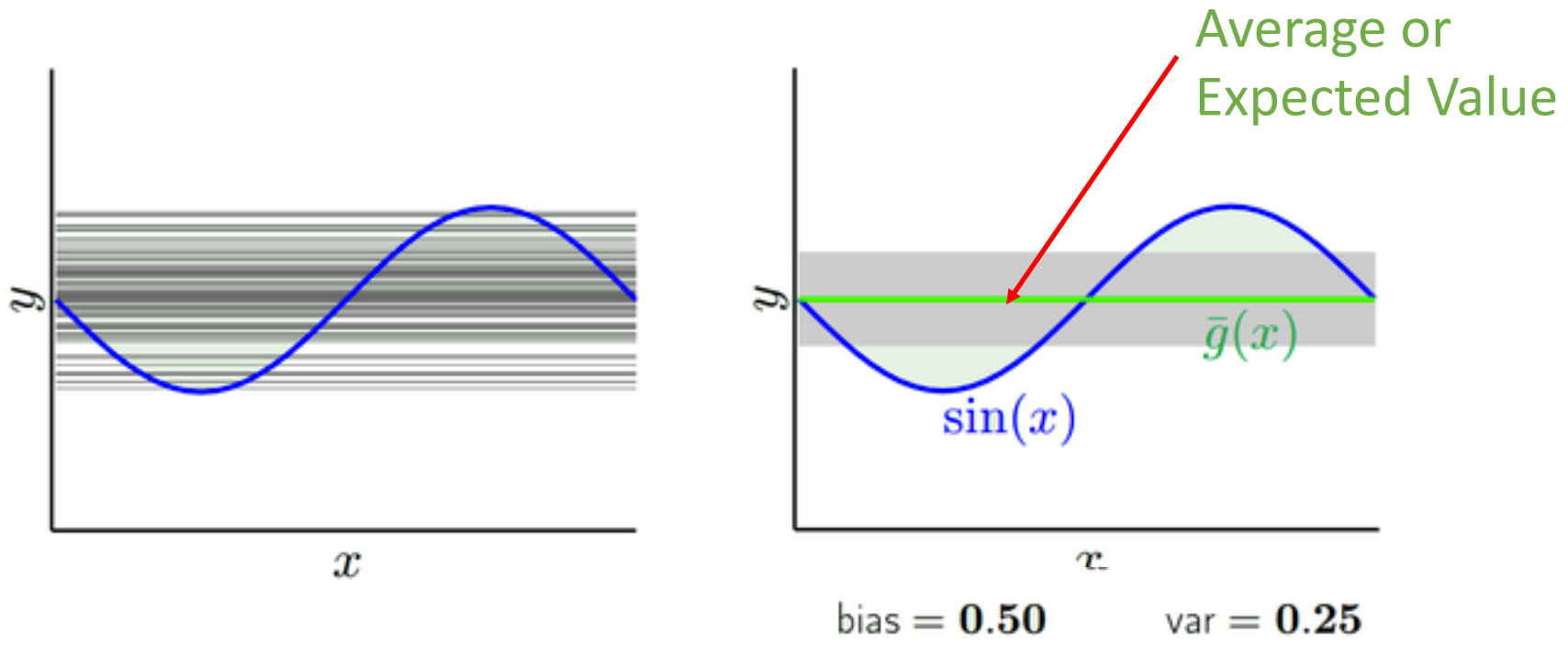
Bias / Variance

- Suppose, we are trying to learn the sine function:



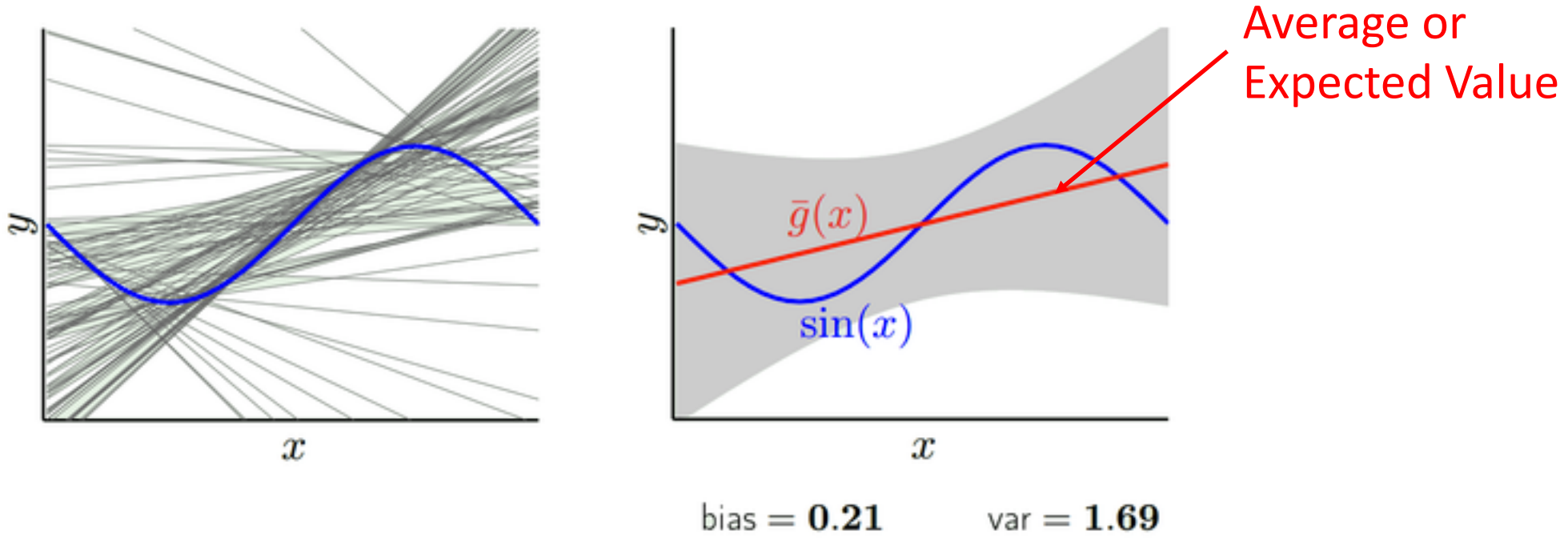
Bias/Variance

- We have a small sample dataset e.g. only 2 training points.
- Case I:
  Model: $h_0 = b$
  where b is a parameter that we optimize

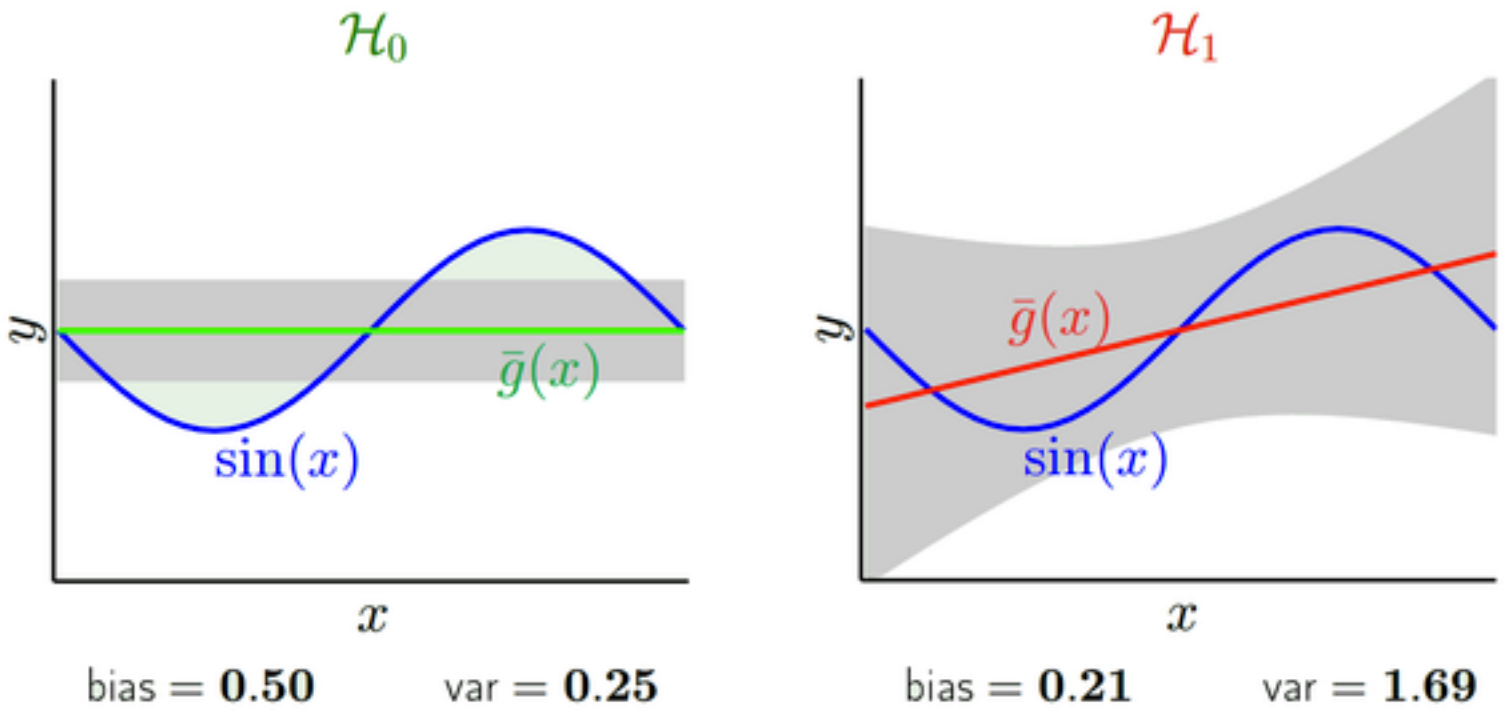


Bias/Variance

- Case II :
  Model: $h_1(x) = a\,x + b$
  where b is a parameter that we optimize



Comparison

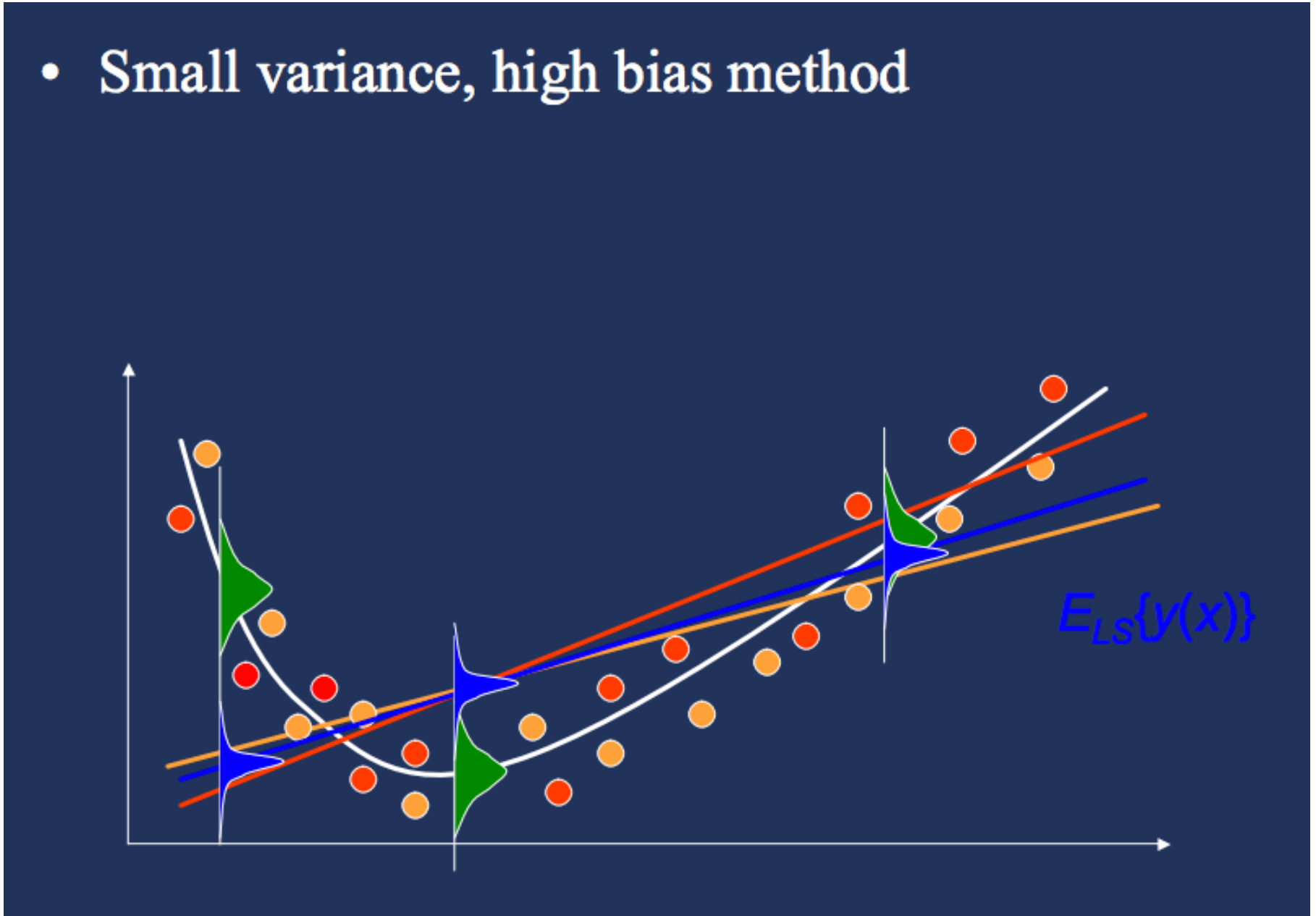


Observations:

- By making the model more complex, we have reduced bias but increase variance.
- A simple model has low variance, but high bias.
- Bias Variance Tradeoff

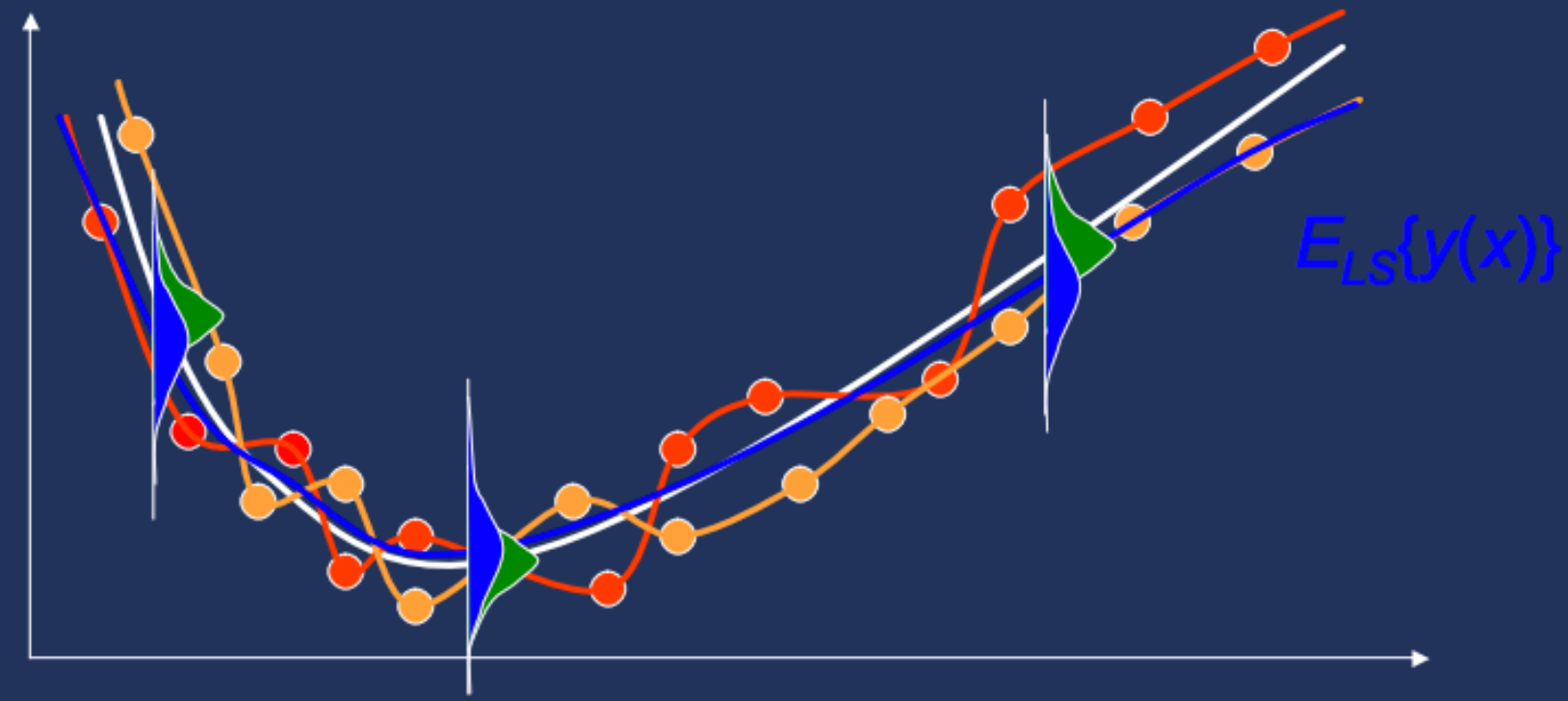
Bias-Variance Example 2

- Small variance, high bias method



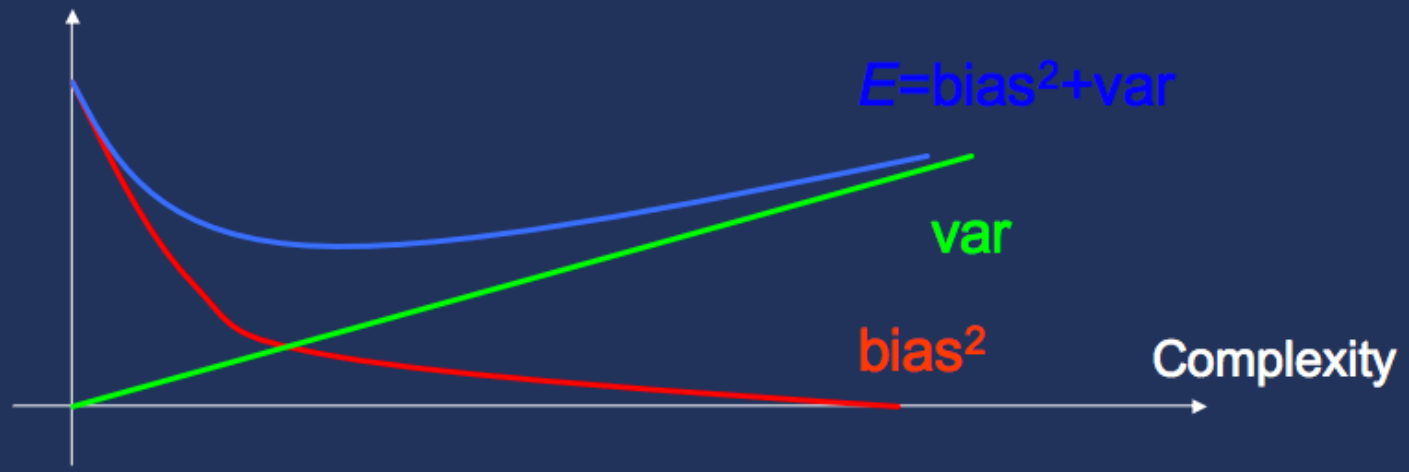
Bias-Variance Example 2

- Small bias, high variance method



Model Complexity

Complexity of the model



Usually, the bias is a decreasing function of the complexity, while variance is an increasing function of the complexity.

How does all of this help me in ML?

- You begin to ask – do I really need to understand all of this?
- Keep in mind that my wish list is:
  - low bias
  - low variance
  - reasonable model complexity
  - avoid overfitting

How does all of this help me in ML?

- You begin to ask – do I really need to understand all of this?
- Keep in mind that my wish list is:
  - low bias
  - low variance
  - reasonable model complexity
  - avoid overfitting
- Can I get all of the above?

Classifiers based on idea of bias and variance

Background - Variance

- Property of Variance:

$$var(aX + b) = a^2 var(X)$$

- Let's compute variance of the mean of a random variable:

$$var(\bar{X}) = var(\frac{1}{N}\sum_i X_i) = \frac{1}{N^2}(\sum_i var(X_i)) = \frac{1}{N}\frac{\sum_i var(X_i)}{N}$$

- Suppose each X_i represents the output of a model on a different dataset.
  If we take the variance of the average of these outputs, it will be lower than the average of the variances by a factor of 1/N

Background - Variance

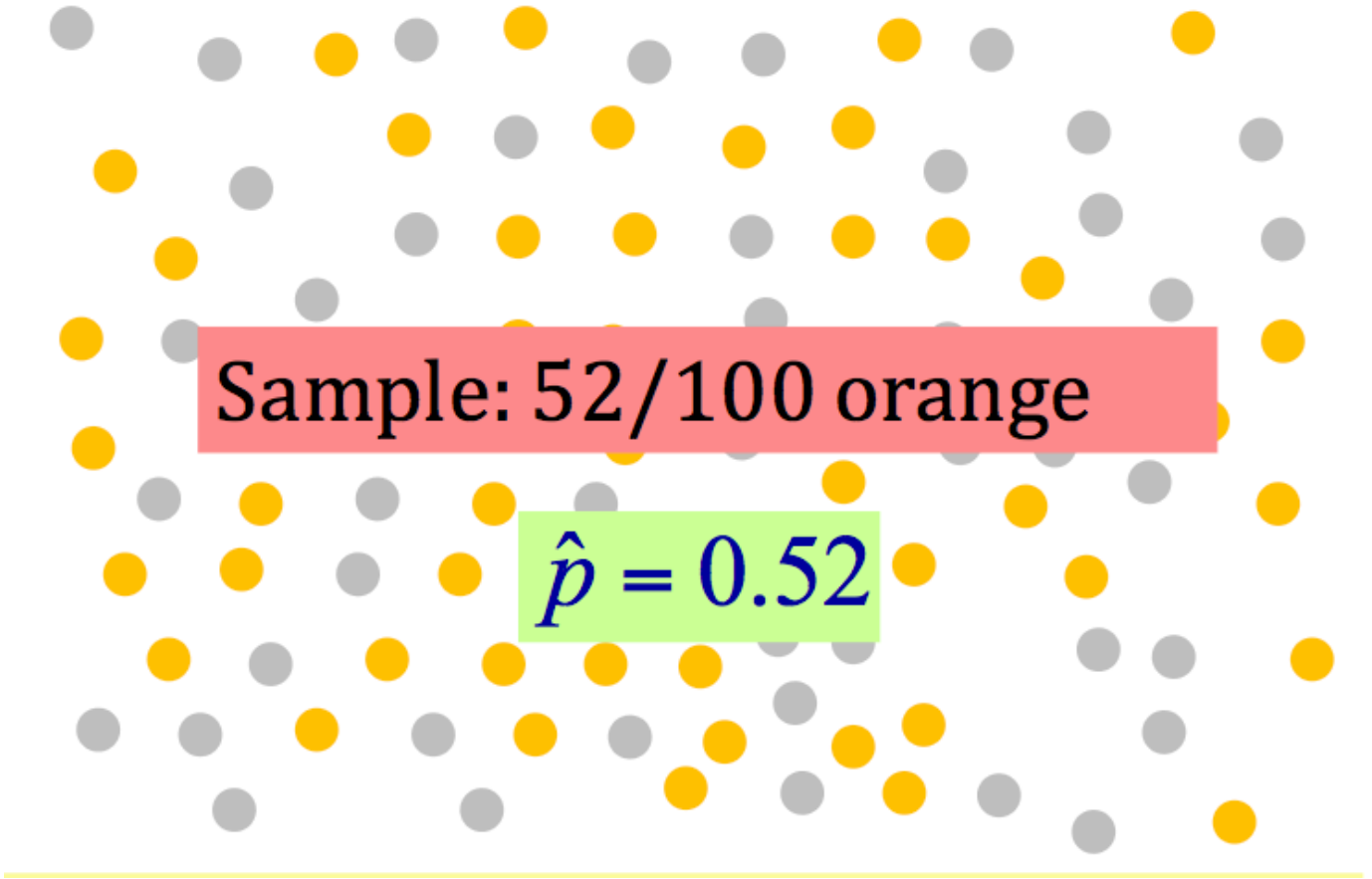
- How do we get different datasets?
- In reality, you have only 1 dataset available and don't have the power to sample over and over from a population.
- How do we simulate random sampling using only the data available?

BOOTSTRAP

- Bootstrap – create new samples of the same size from original dataset by sampling with replacement.

Bootstrap - Example

- I have one data sample.
- I can't sample again from the population.
- Can I simulate new samples from this sample?
- Sample with replacement from the dataset available.

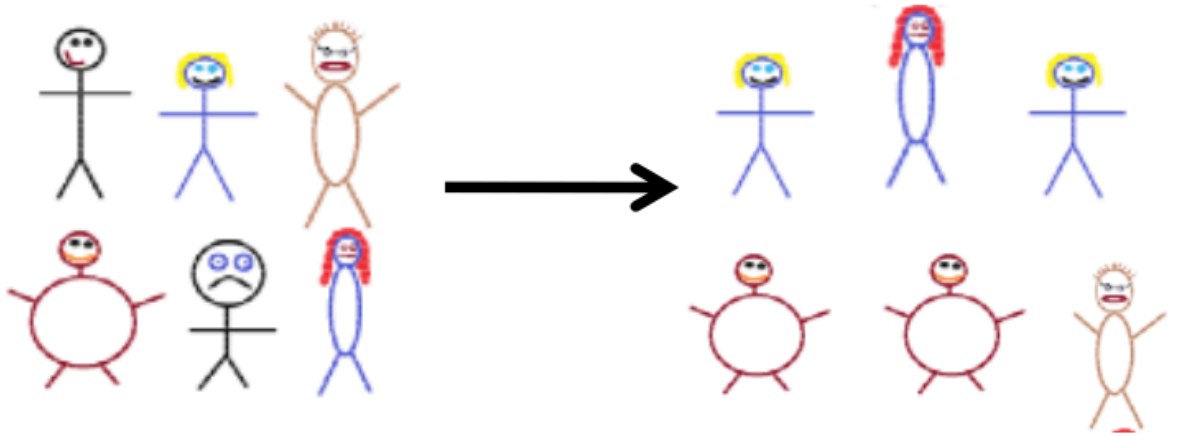


Bootstrap - Example

- Suppose I have a sample of 6 people.
- That's all the data that you have to work with.
- I'd like another sample of same size.

Bootstrap - Example

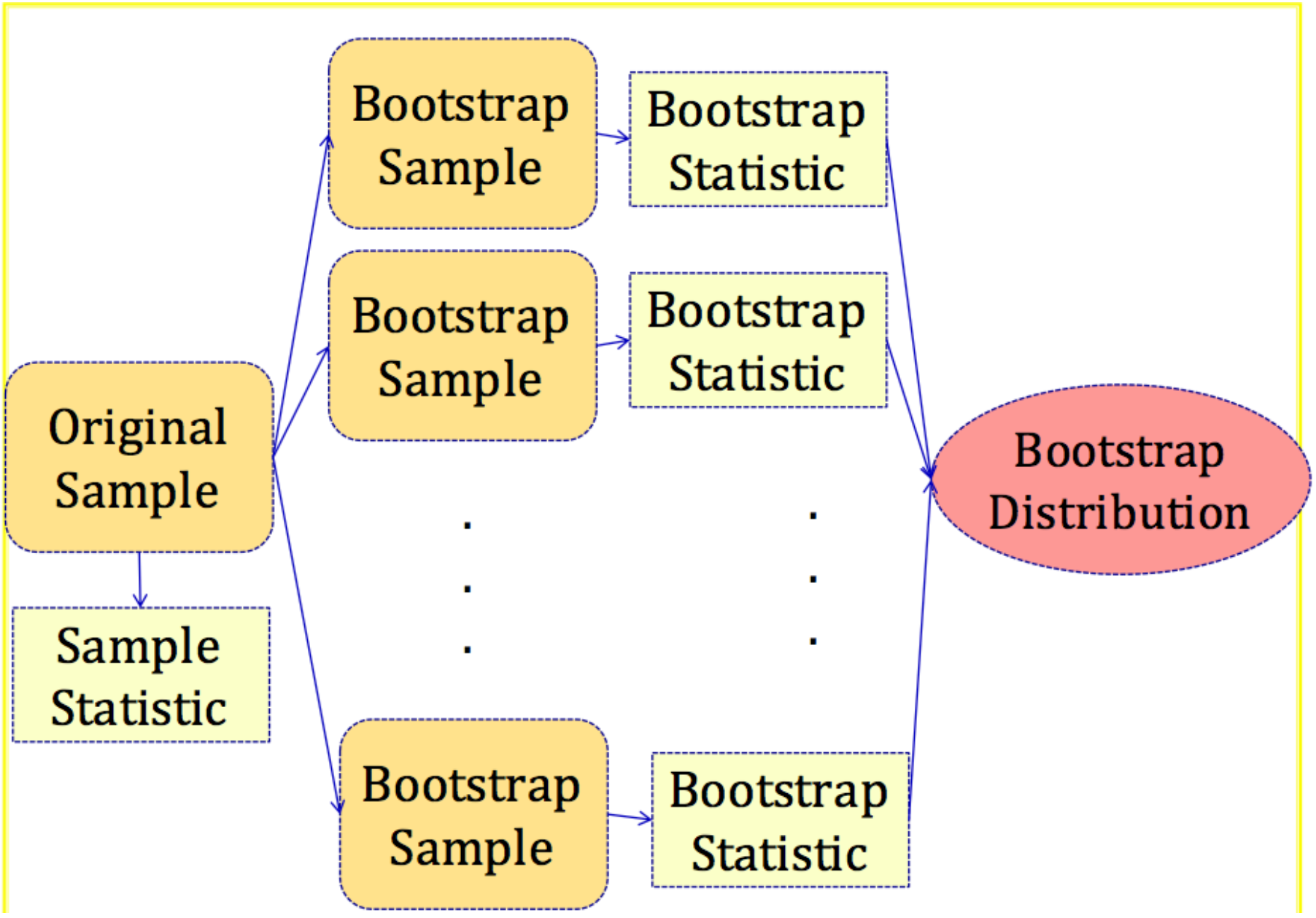
- Suppose I have a sample of 6 people.
- That's all the data that you have to work with.
- I'd like another sample.
- Idea of sampling with replacement: For each slot, I have all n choices.



Original Sample

Bootstrap Sample

Bootstrap Distribution



Bootstrap

Your original sample has data values

18, 19, 19, 20, 21

Is the following a possible bootstrap sample?

18, 19, 20, 21, 22

Bootstrap

Your original sample has data values

18, 19, 19, 20, 21

Is the following a possible bootstrap sample?

18, 19, 20, 21, 22

a) Yes

b) No

22 is not a value from the original sample

Bootstrap

Your original sample has data values

18, 19, 19, 20, 21

Is the following a possible bootstrap sample?

18, 19, 20, 21

Bootstrap

Your original sample has data values

18, 19, 19, 20, 21

Is the following a possible bootstrap sample?

18, 19, 20, 21

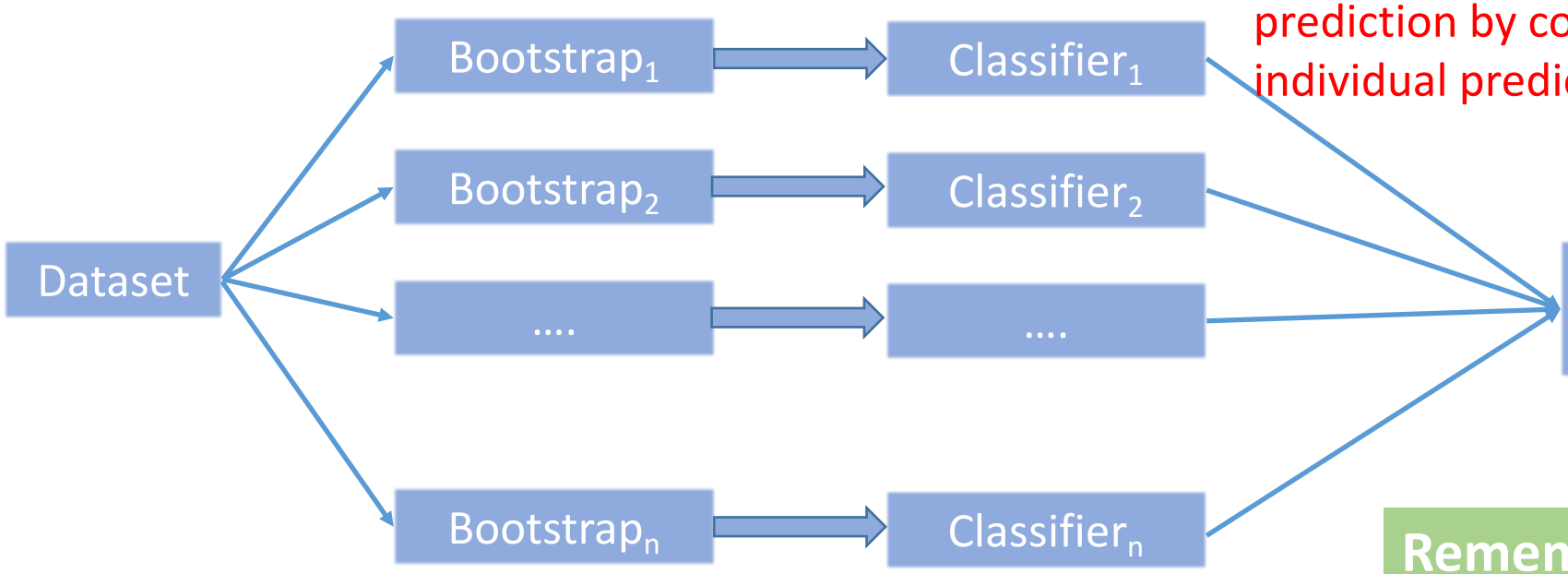
a) Yes

b) No

Bootstrap samples must be the same size as the original sample

Bagging – Bootstrap Aggregation

- A set of classifiers based on the idea of bootstrap + aggregating the output of individual classifiers.



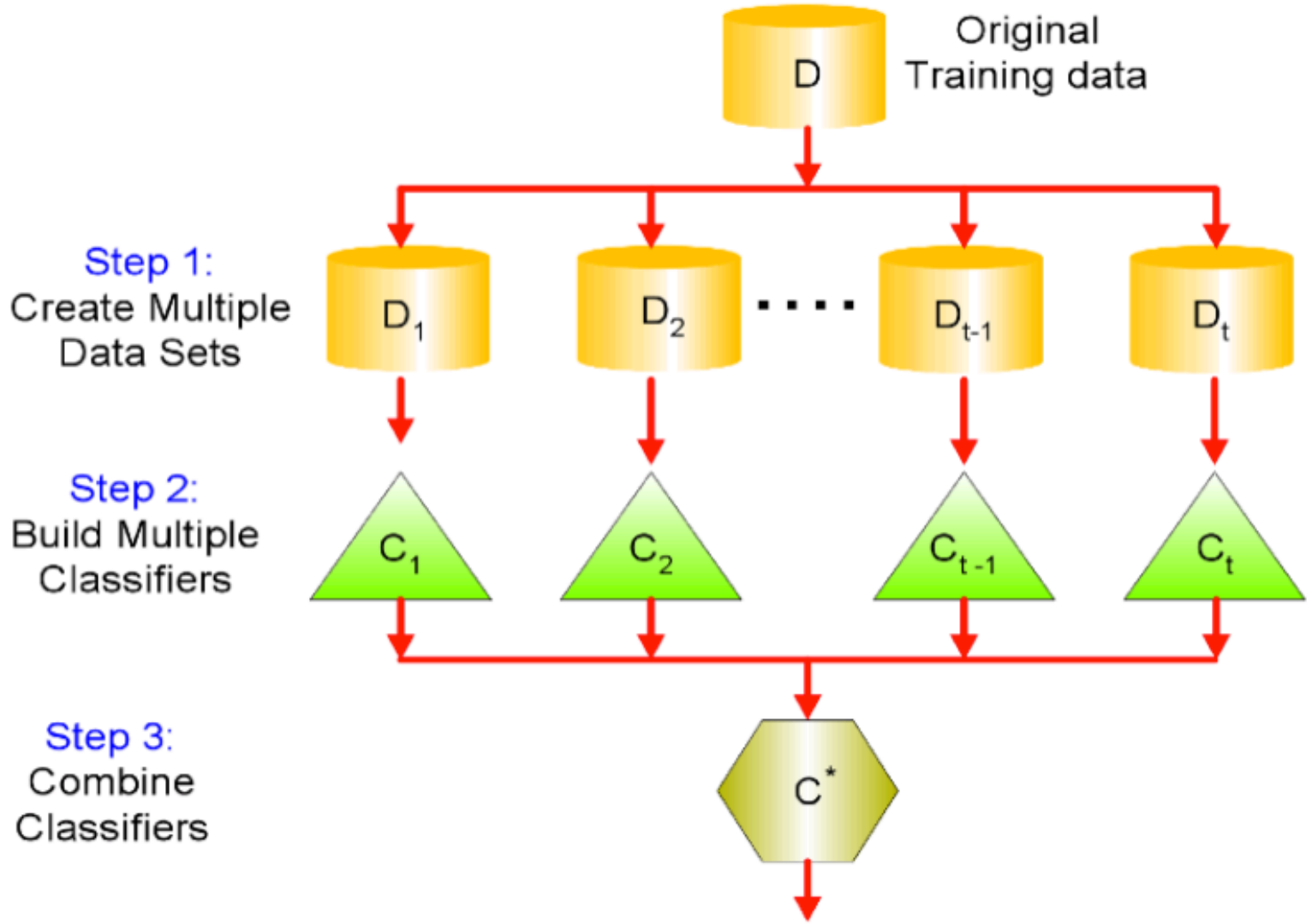
Remember: Averaging reduces variance

Bootstrap Algorithm

- Input: Dataset D containing N training examples.
- Steps:

1. Create k bootstrap samples D_1, D_2, …, D_k
2. Train distinct classifiers C_1, C_2, …, C_k on each of these datasets
3. Create an aggregate prediction by taking the majority / average of these individual classifiers.

Bootstrap Algorithm



Analysis

- In a any bootstrap sample of size N, a given data point has probability of not being selected as: $\left(1-\frac{1}{N}\right)^N$
- So, probability of being selected as (p): $1-\left(1-\frac{1}{N}\right)^N$
- Think of it like a binomial distribution with p as being probability of heads.
- Expected number of points that are selected in any given bootstrap sample = Np

  $= \mathrm{N}[1-\left(1-\frac{1}{N}\right)^N]$

Analysis

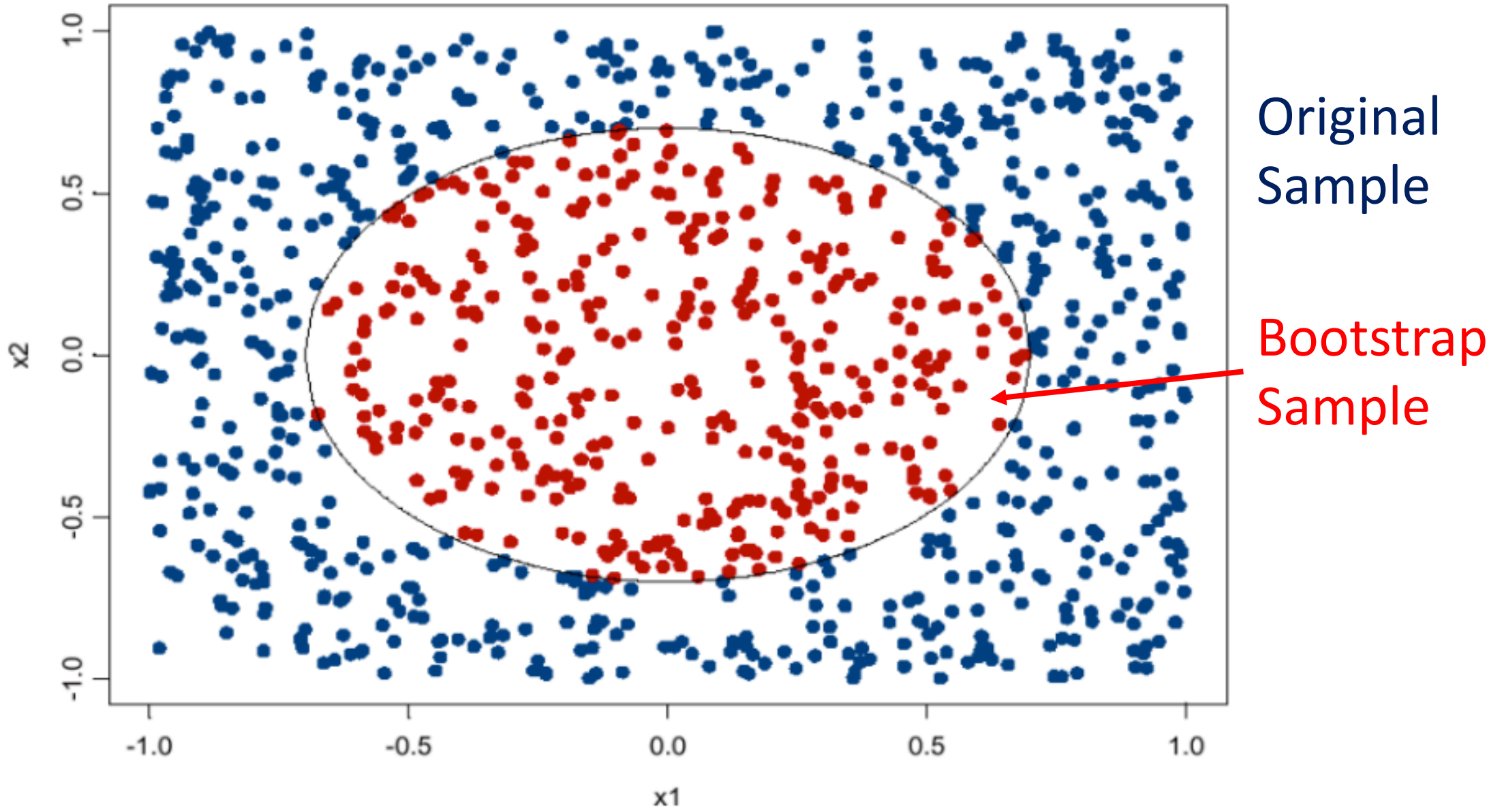
- Remember, that e^x can be written as:

$$\exp(x) = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$$

- So, for large enough values of N, the expected number of points in a bootstrap sample becomes:

$$\begin{aligned} \mathrm{E} &= \mathrm{N}[1 - \left(1 - \frac{1}{N}\right)^N] \\ &= N[1 - \exp(-1)) \\ &= 0.632\ N \end{aligned}$$

Analysis


1.0
0.5
0.0
-0.5
-1.0
x2
-1.0
-0.5
0.0
0.5
1.0
x1
Original Sample
Bootstrap Sample

Analysis

- So, what does Bagging model do for me?
  Creates an aggregated model with less variance.
- What is the reduction?

  Ideally, $\mathrm{var}(\mathrm{Bagging}(\mathrm{C}(\mathrm{x}, \mathrm{D})) = \frac{\mathrm{var}(\mathrm{C}(\mathrm{x}, \mathrm{D}))}{\mathrm{N}}$

- But, in reality the models are not independent (since their datasets are correlated), so reduction is less than (1/N) times.
- Does Bagging also lead to an increase in accuracy?
  Yes, it has been empirically shown to

Bagging Results

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 *D* causes a large change in the output hypothesis φ.
  - If small changes in *D* causes large changes φ in then there will be an improvement in performance.
- Bagging helps unstable procedures, but could hurt the performance of stable procedures.
- Neural nets and decision trees are unstable.
- k-nn and naïve Bayes classifiers are stable.

Random Forests

- In Bagging classifiers, we created new datasets by varying the instances from the original dataset.
- What if we vary the attributes also?
- This is the idea behind Random Forests (RF). The base classifiers are Decision Trees.
- Idea: Create Decision Tree (DT) models from the bootstrap samples, but in each DT limit the splitting criteria to a random subsample of the attributes.

Random Forest Algorithm

Input:

Dataset D containing N instances and M attributes

Training Phase:

For b = 1 to B

Create a bootstrap sample of size N from original sample
Grow tree T_b using the bootstrap sample as follows:
Create a random sample of m attributes (m < M)
Use these attributes to construct a decision tree as usual

Test Phase:

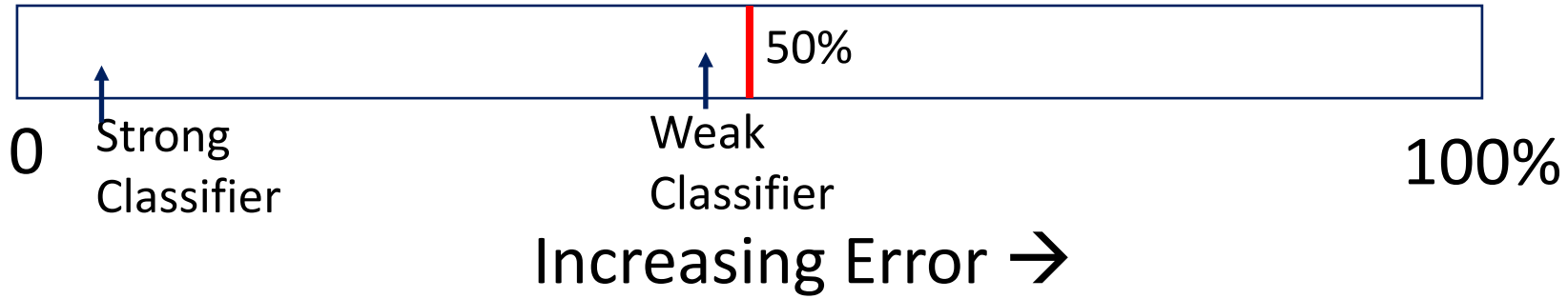
For a new data point X, take the aggregated prediction of models
For classification: aggregate = majority
For regression: aggregate = average

Best of both worlds

- Can we get lower variance and lower bias at the same time?
- We studied the bias-variance tradeoff.
- But, is there a way around it?
- Can we average models ***and*** reduce bias?
- Boosting

Boosting

- Boosting – combining a set of "weak" classifiers to produce a strong classifier.
- What is a "weak classifier"?
- In machine learning terminology, even a random predictor can predict with 50% error.
- If there exists a classifier that generates slightly better results than a random predictor, it's known as a weak classifier.
- Error of a weak classifier = (50 - γ) %, where γ is a small positive number.



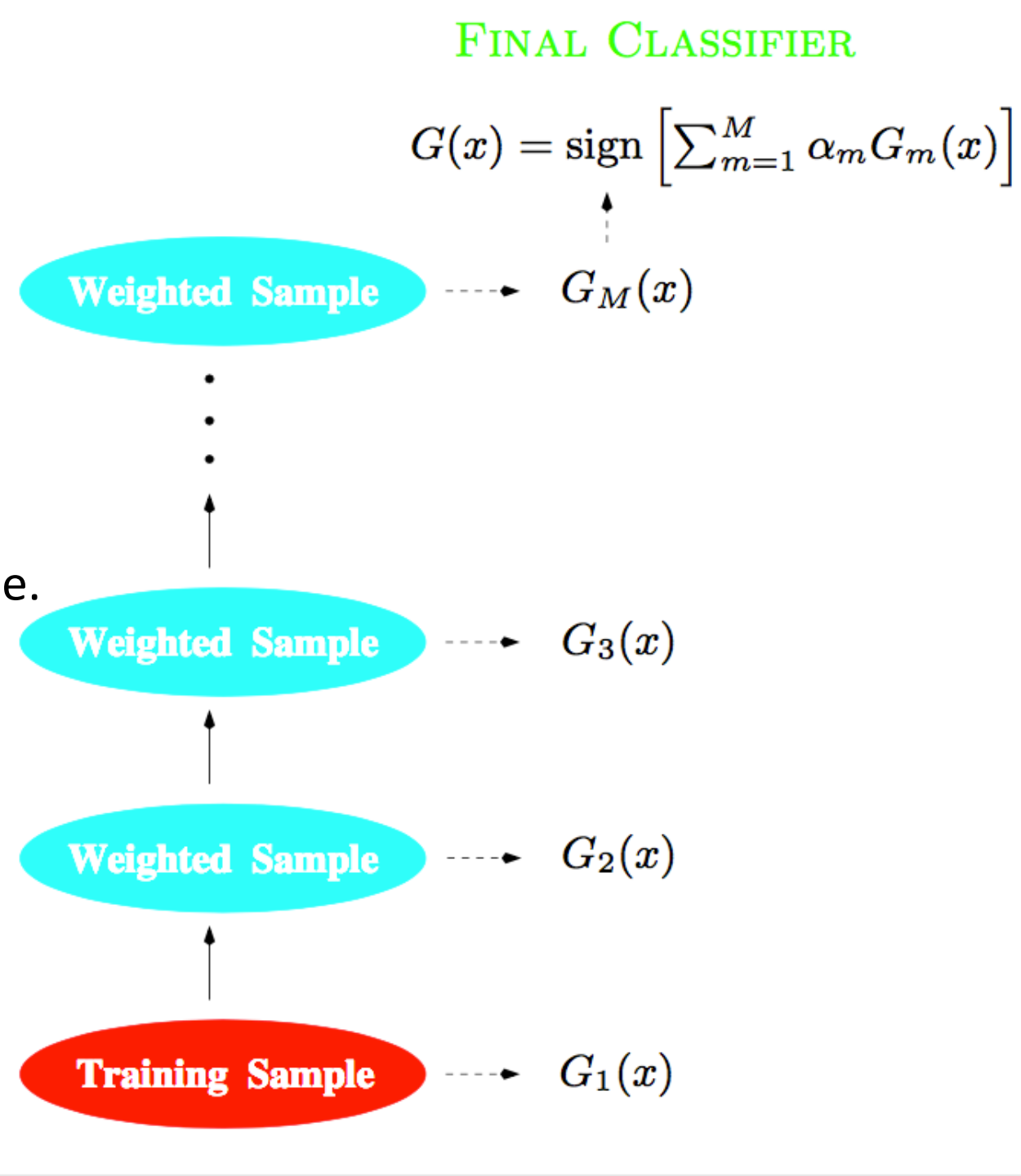
Boosting - Idea

- Start with equal weights given to all training instances.
- Run 1st rule of thumb (hypothesis G_1) => Keep track of the instances where G_1 makes an error. Increase the weight of those instances.
- Repeat this process for 2nd, 3rd, …, kth hypothesis.
- Create a final hypothesis by taking a weighted average of these k hypotheses.

$$G_{final} = \sum_{i=1}^{k} \alpha_i G_i$$

Boosting

Key points to note:
- Each classifier gets a differently weighted sample.
- The classifiers work in a serial way i.e. one after the other.
- The classifiers are not independent as the output of one is input for the other



Boosting

- You want to build a spam filter based on N training examples $(\boldsymbol{X}^i, y^i)$ where y^i is {Non-Spam (0), Spam(1)}, and i = 1 to N.
- Approach:

  Start with equal weight for each example: $w_i = \frac{1}{N}$

  The initial weight distribution $D_0 = \{w_0 = \frac{1}{N}, \ldots, w_N = \frac{1}{N}\}$

  for i = 1 to T:

  - Apply a rule of thumb (hypothesis G_i) to the data with weight distribution D_i
  - Keep track of examples where the hypothesis makes an error and increase their weights to create a new distribution D_{i+1}

  end for

Adaboost

- One of the most popular implementations of Boosting was proposed by Freund & Schapire in 1996.
- Simple to implement and combines many weak classifier to produce a single strong hypothesis with excellent results.
- Algorithm presented on next slide.

Adaboost

- Given: Set of examples $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ where $x_i \in X\ (Domain)$, $y_i \in Y = \{-1, +1\}$
- Algorithm:
  Initialize $D_1(i) = 1/m$
  For t = 1 to T:
  - Train weak learner h_t using distribution D_i
  - Get weak hypothesis $h_t: X \to \{-1, +1\}$ with error:

$$\epsilon_t = \frac{\sum_{i=1}^{m} w_i\, I[h_t(x_i) \neq y_i]}{\sum_{i=1}^{m} w_i}$$

I is the identity function:
$I(1) = 1$
$I(0) = 0$

  - Choose $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
  - Update weights as:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

  where Z_t is a normalization factor so that D_{t+1} sums to 1.

Adaboost

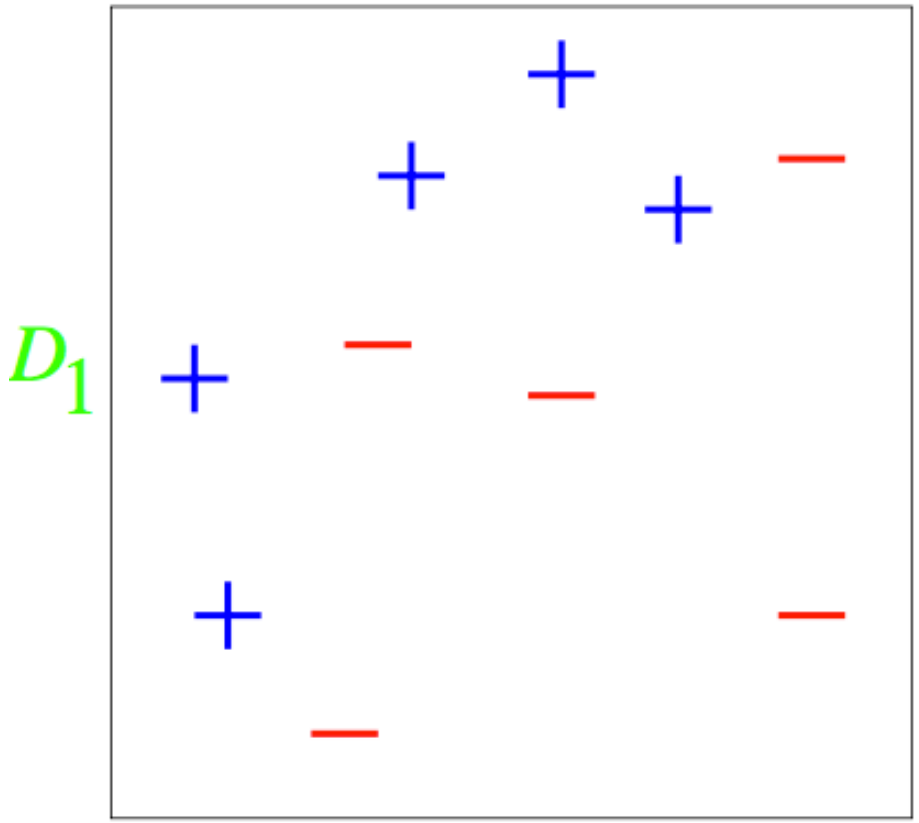
- The final hypothesis is obtained as:

$$H_{final}(x) = sign(\sum_{t} \alpha_t h_t(x))$$

Toy Example:

- Let's say we have the training data as shown on the right.
- There are 10 data points – 5 of each class.
- Initial weight distribution:

$D_1 = \{w_i = \frac{1}{10}\}$ for each i = 1 to 10.



Toy Example – Round 1

- Our first weak hypothesis is shown on right.
- 3 points are misclassified.

$$\epsilon_1 = \frac{3 \times 0.1}{1} = 0.30$$

$$\alpha_1 = \frac{1}{2} \ln\left(\frac{1 - 0.30}{0.30}\right) = 0.424$$

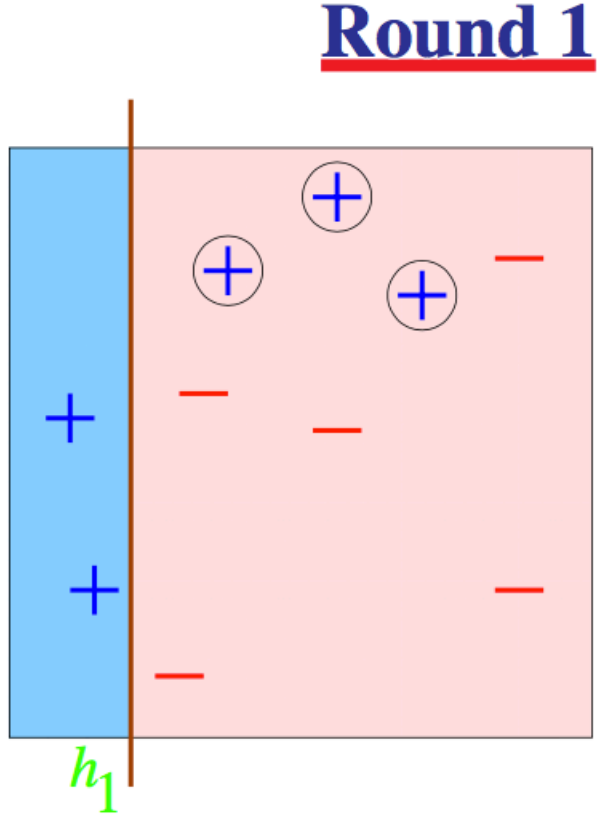
The weight of misclassified points:

$$D_2 = 0.1 \times \exp(0.424) = 0.153$$

The weight of correctly classified points:

$$D_2 = 0.1 \times \exp(-0.424) = 0.065$$

These are not normalized. After normalizing – dividing each value by total of weight of 0.914 -



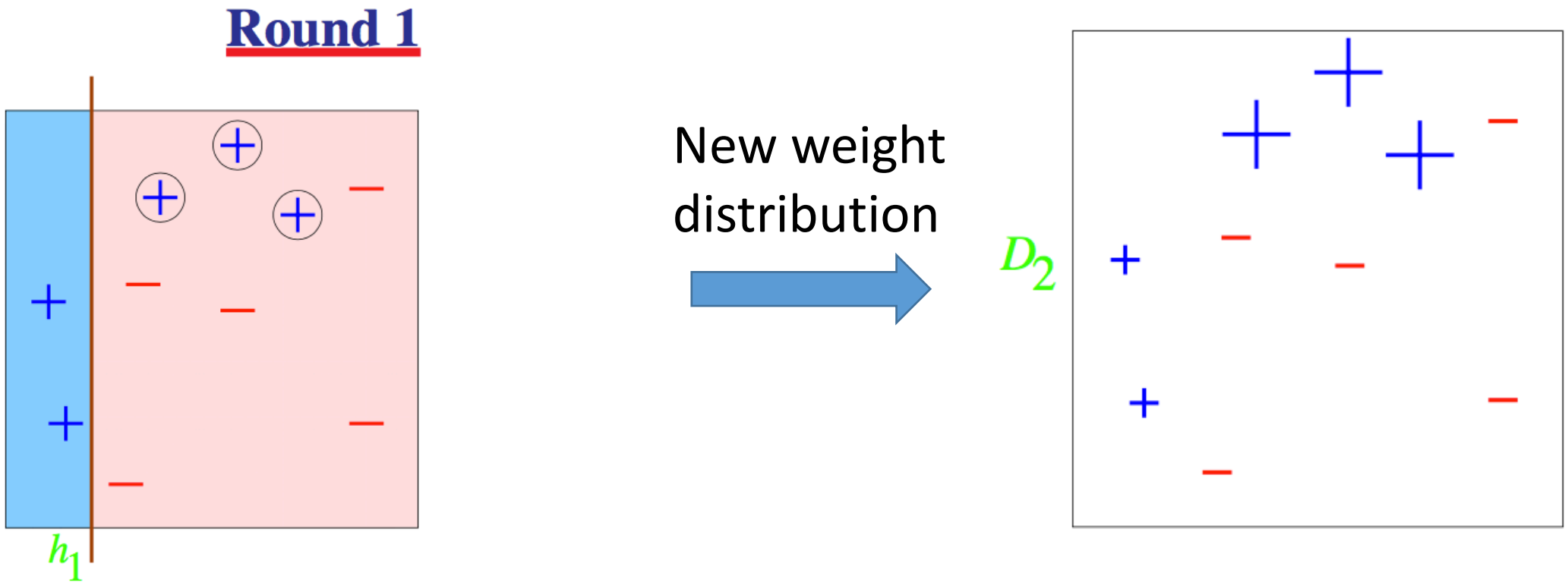
Weight of misclassified:

0.167 each

Weight of correctly classified:

0.071 each

Toy Example:



- After round 1, we get a new distribution D2.
  Note that the 3 misclassified points get a higher weight

Toy Example – Round 2

- Our second weak hypothesis is shown on right.
- 3 points are misclassified.

$$\epsilon_2 = \frac{3 \times 0.071}{1} = 0.213$$

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1 - 0.213}{0.213}\right) = 0.653$$

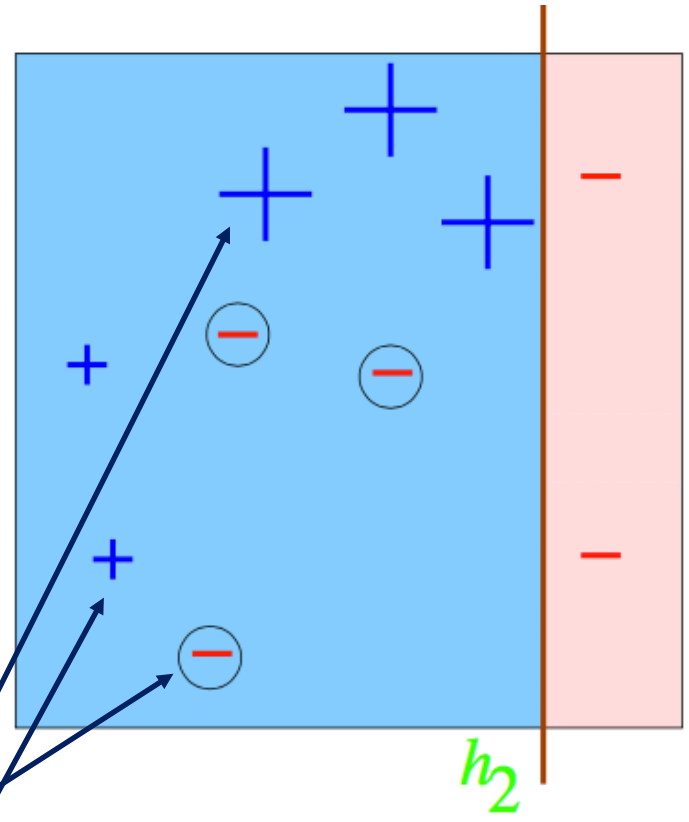
The weight of misclassified points:

$$D_2 = 0.071 \times \exp(0.653) = 0.136$$

The weight of correctly classified points:

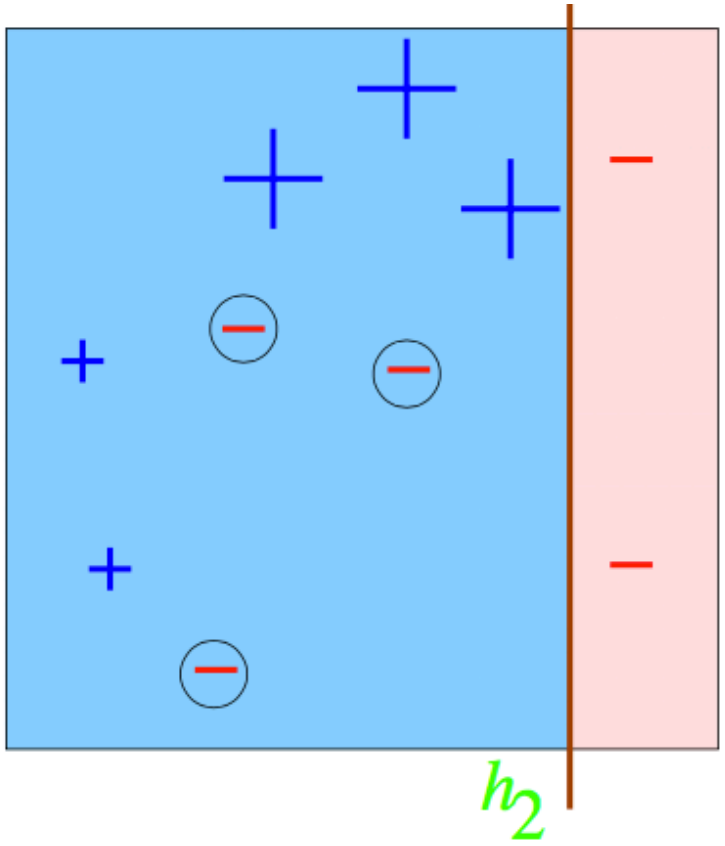
$$D_2 = 0.167 \times \exp(-0.653) = 0.087$$

$$D_2 = 0.071 \times \exp(-0.653) = 0.037$$

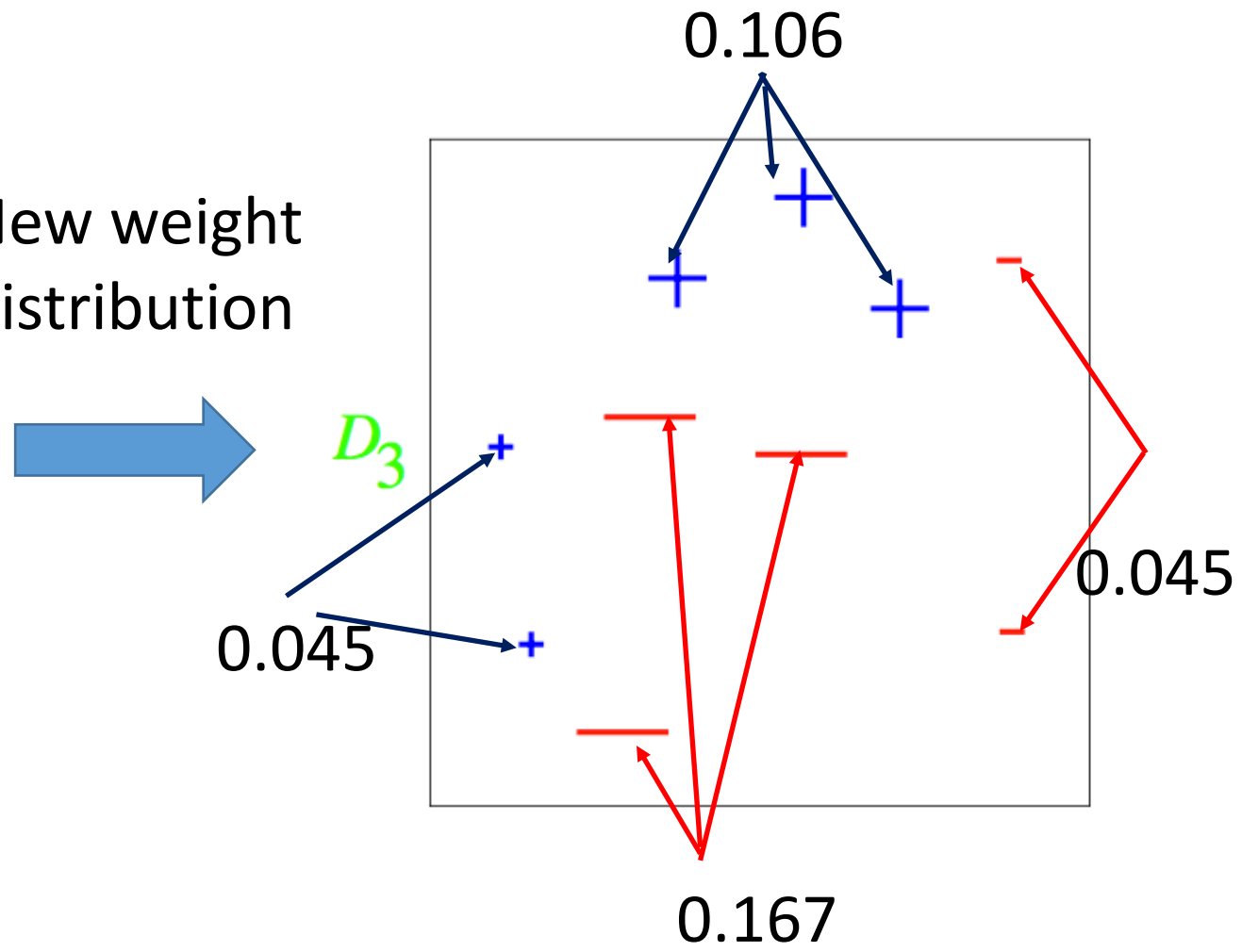


Total weight = 3 * 0.136 +
3 * 0.087 + 4 * 0.037 = 0.817

Toy Example



New weight distribution

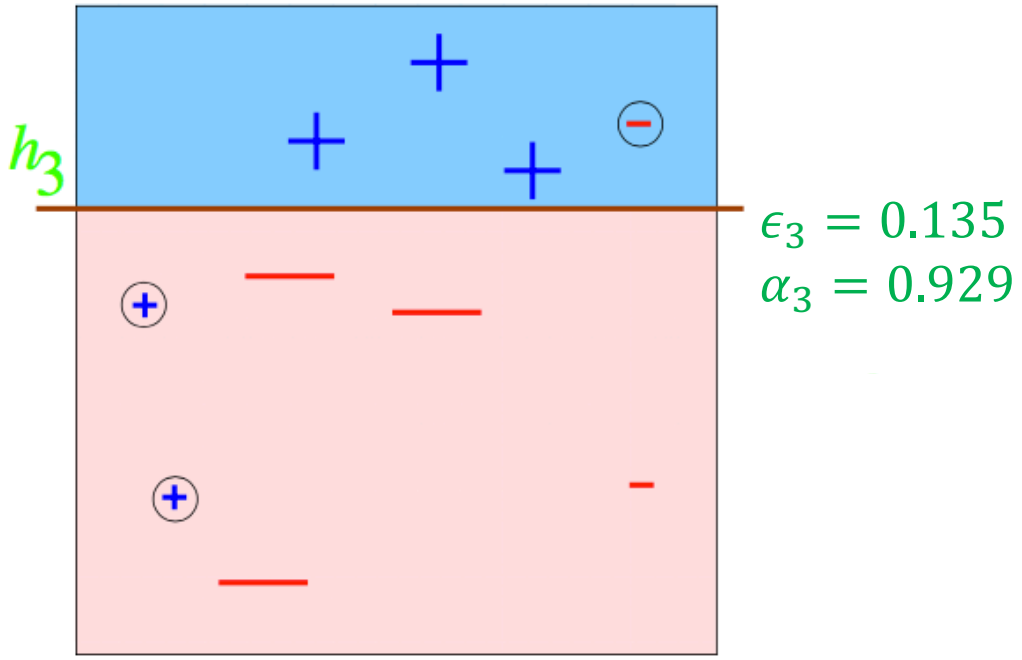


Toy Example – Round 3

- Our third weak hypothesis is shown on right.
- 3 points are misclassified.

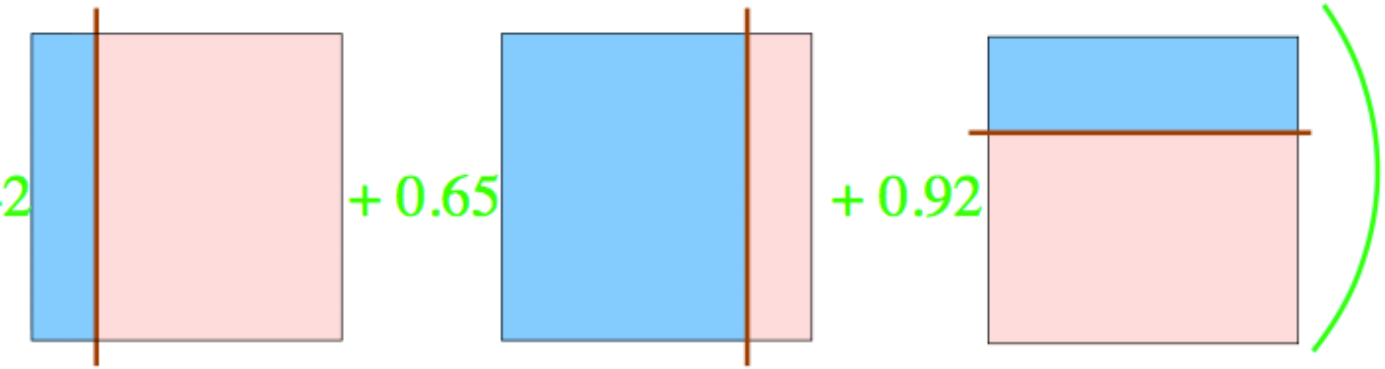
$$\epsilon_3 = \frac{3 \times 0.045}{1} = 0.135$$

$$\alpha_3 = \frac{1}{2}\ln\left(\frac{1 - 0.135}{0.135}\right) = 0.929$$

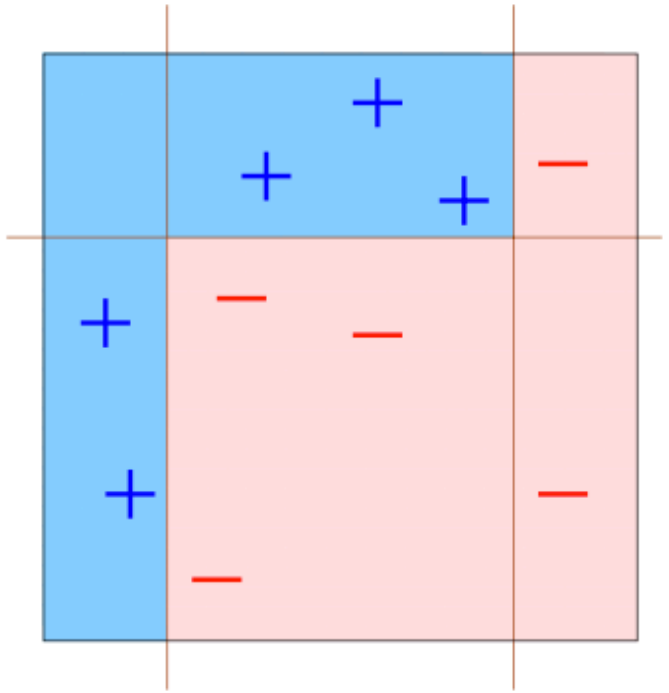


Final Hypothesis

$$H_{\text{final}} = \text{sign}\left(0.42 \; + 0.65 \; + 0.92 \; \right)$$

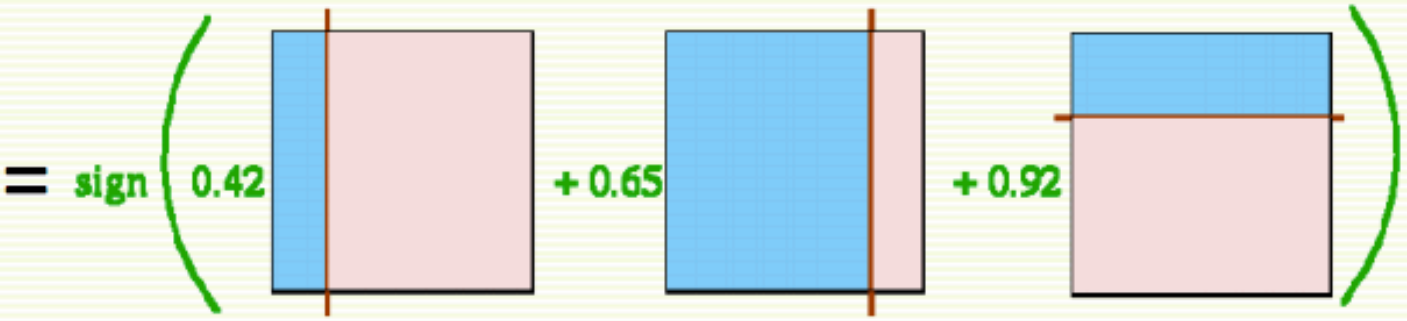


$=$

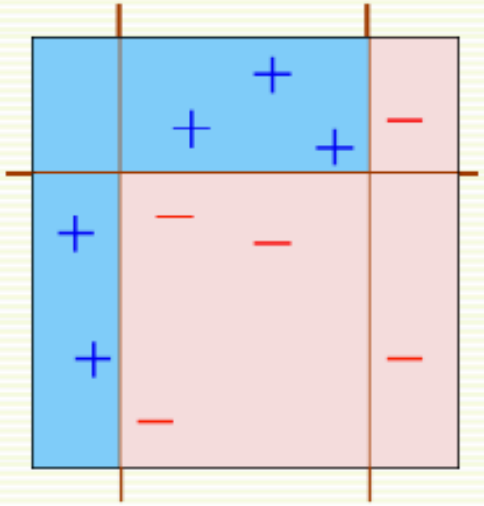


How to use this hypothesis

$$f_{final}(x) = \text{sign}\left(0.42 \quad + 0.65 \quad + 0.92 \quad\right)$$



=



$$f_{final}(x) = \text{sign}\left(0.42\,\text{sign}\left(3 - x_1\right) + 0.65\,\text{sign}\left(7 - x_1\right) + 0.92\,\text{sign}\left(x_2 - 4\right)\right)$$

- note non-linear decision boundary

AdaBoost Analysis

- It is shown in the paper that the training error drops exponentially fast.

$$Error_{train} \leq \exp(-2\sum_{t}\gamma_t^2)$$

- Remember the definition of $\boldsymbol{\gamma}$: $\gamma_t = \epsilon_t - \frac{1}{2}$, where ϵ_t is the error at round t of the Boosting algorithm.
- Example: : Let errors for the first four rounds be, 0.3, 0.14, 0.06, 0.03, 0.01 respectively. Then

$$Error_{train} \leq \exp[-2(0.2^2 + 0.36^2 + 0.44^2 + 0.47^2 + 0.49^2)]$$

$$\approx 0.19$$

Another Toy Example

MIT Admissions Training Data

ID	Name	**Admit/Deny**	# of High School Detentions	SAT
1	Andrew	Deny	3	2050
2	Burt	Admit	1	2200
3	Charlie	Admit	2	2090
4	Derek	Deny	4	2230
5	Erica	Admit	5	2330
6	Faye	Deny	6	2220
7	Greg	Admit	6	2390
8	Helga	Admit	7	2320
9	Ivana	Deny	8	2330
10	Jan	Deny	8	2090

Another Toy Example

- Propose two simple hypothesis:
  h1: If # of High School Detentions < 3, then ADMIT
  h2: If SAT > 2200, then ADMIT
- Can you combine these two weak hypothesis to produce a strong hypothesis?

Summary

- Ensemble methods combine individual classifiers to achieve better results.
- Bagging doesn’t work so well with stable models. Boosting might still help.
- Boosting might hurt performance on noisy datasets. Bagging doesn’t have this problem.
- On average, boosting helps more than bagging, but it is also more common for boosting to hurt performance.
- Bagging is easier to parallelize.