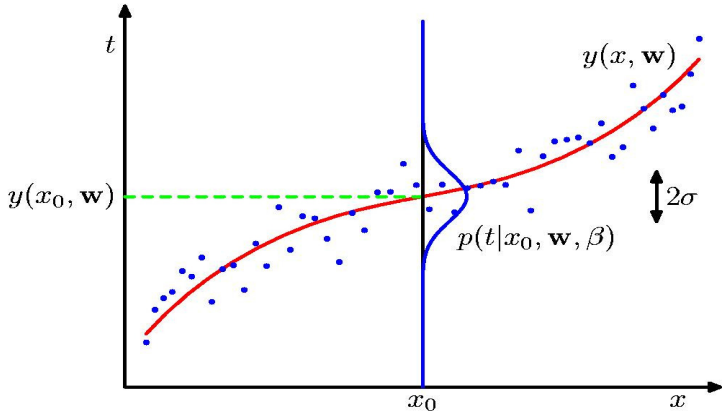
HTF: Ch3, 7
B, Ch3

Linear Regression, Evaluation, Bias-Variance Tradeoff

Thanks to: C Guestrin, T Dietterich, R Parr

Outline



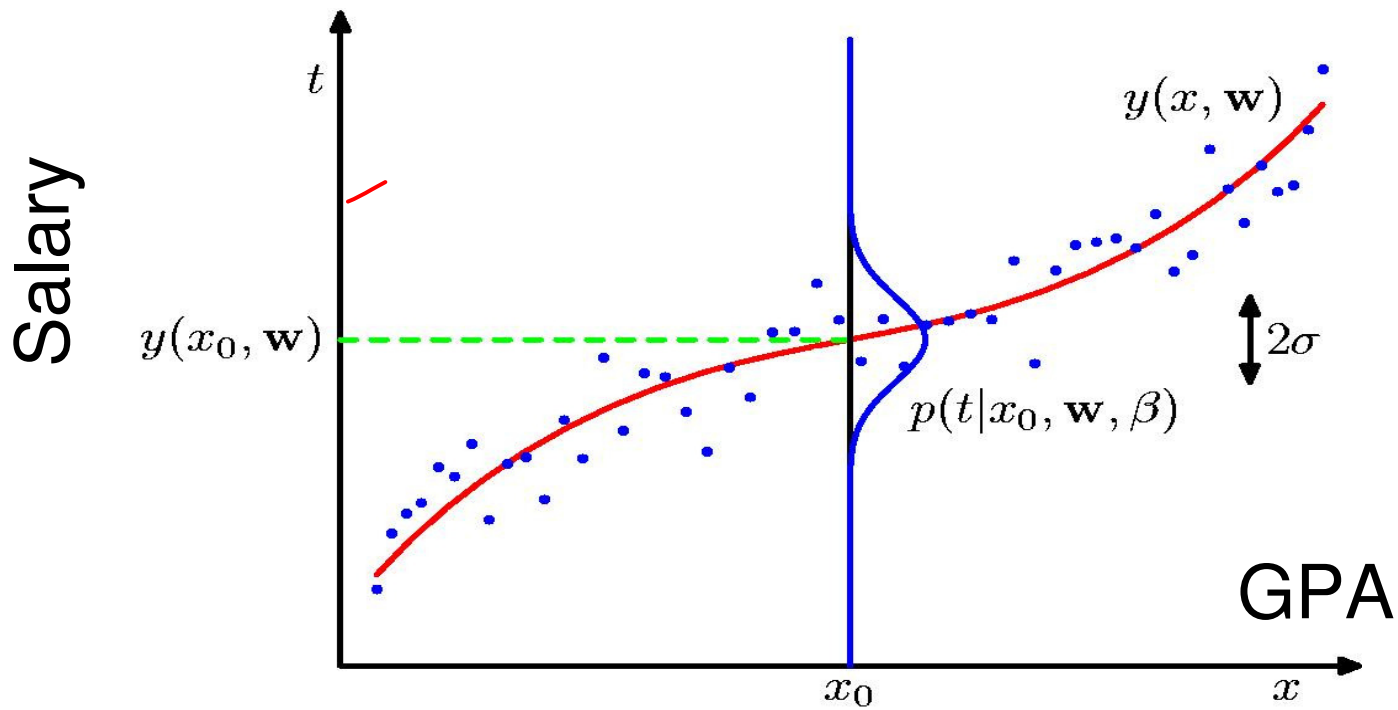
- Linear Regression
  - MLE = Least Squares!
  - Basis functions
- Evaluating Predictors
  - Training set error vs Test set error
  - Cross Validation
- Model Selection
  - Bias-Variance analysis
  - Regularization, Bayesian Model

Prediction Problems …

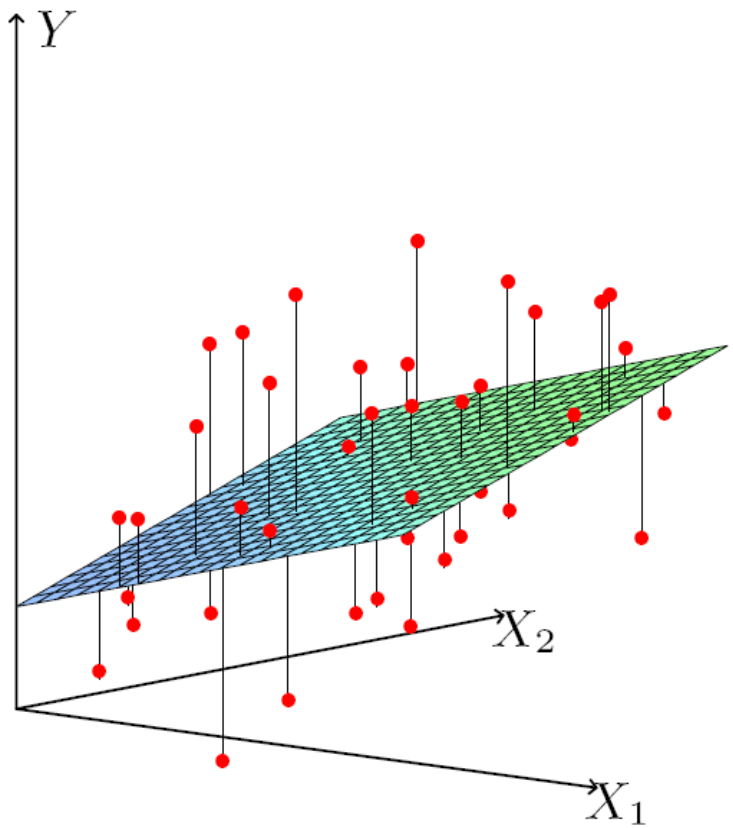
- Predict housing price from:
  - House size, lot size, rooms, neighborhood, …
- Predict weight from:
  - Gender, height, ethnicity, ...
- Predict life expectancy increase from:
  - Medication, disease state, ...
- Predict crop yield from:
  - Precipitation, fertilizer, temperature, ...

Prediction of Continuous Variables

- Predict a continuous variable based on set of continuous inputs:
  - Eg, predict salaries from GPA
  - **Regression**



Best LINEAR Fit



- *Finding LINEAR fit*
  - *Find* $(\beta_0, \beta_1, \ldots, \beta_k)$
    $y = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k$

- *Linear least squares fitting with* $X \in \Re^2$
- *Seek the linear function of X that minimizes the sum of squared residuals from Y*

The Linear Regression Task

- **Given set of labeled Instances:** { [$\mathbf{x}_j$, t_j] }

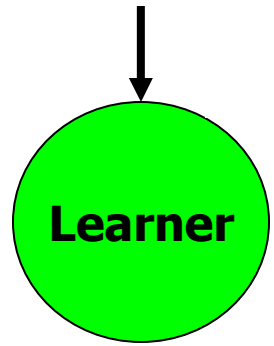
  GPA, Age, ShoeSize, … → Salary

  Eg: [(97, 14, 8); 150]
  [(93, 24, 12); 200]
  [(88, 20, 9); 45]

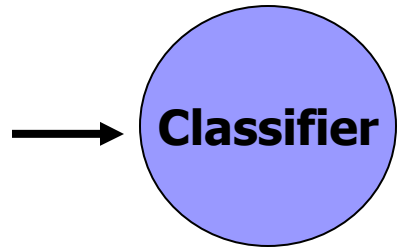
- **Learn:** Mapping from $\mathbf{x}$ to $t(\mathbf{x})$
  - Direct linear mapping: $t(\mathbf{x}) \approx \beta_0 + \sum_j \beta_j x_j$
  - Find coeffs $\beta = (\beta_0, \beta_1, \ldots, \beta_k)$
- **Model**: Observed value $t(x) = \beta_0 + \sum_j \beta_j x_j + \varepsilon$
  where $\varepsilon \sim N(0, \sigma^2)$

Training a Regressor

Width	Size	Eyes	…	Light	size
35	95	Y	…	Pale	22
22	110	N	…	Clear	18
:	:			:	:
10	87	N	…	Pale	33



Width	Size	Eyes	…	Light
32	90	N	…	Pale



size
19

Best Values of β ?

- **Model**: Observed value is $t(\mathbf{x}) = \sum_j \beta_j x_j + \varepsilon$ where $\varepsilon \sim N(0, \sigma^2)$

$$P(t, \mathbf{x} \mid \boldsymbol{\beta}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-\left[t - \sum_i \beta_i x_i\right]^2}{2\sigma^2}}$$

- Find MostLikely values of β … (MLE)

$$\ln P(D \mid \boldsymbol{\beta}, \sigma)$$

$$= N \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2\sigma^2}\left[\sum_j \left[t^j - \sum_i \beta_i x_i^j\right]^2\right]$$

Max Likely Estimate

$$\ln P(D \mid \boldsymbol{\beta}, \sigma) = N \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2\sigma^2}\left[\sum_j \left[t^j - \sum_i \beta_i x_i^j\right]^2\right]$$

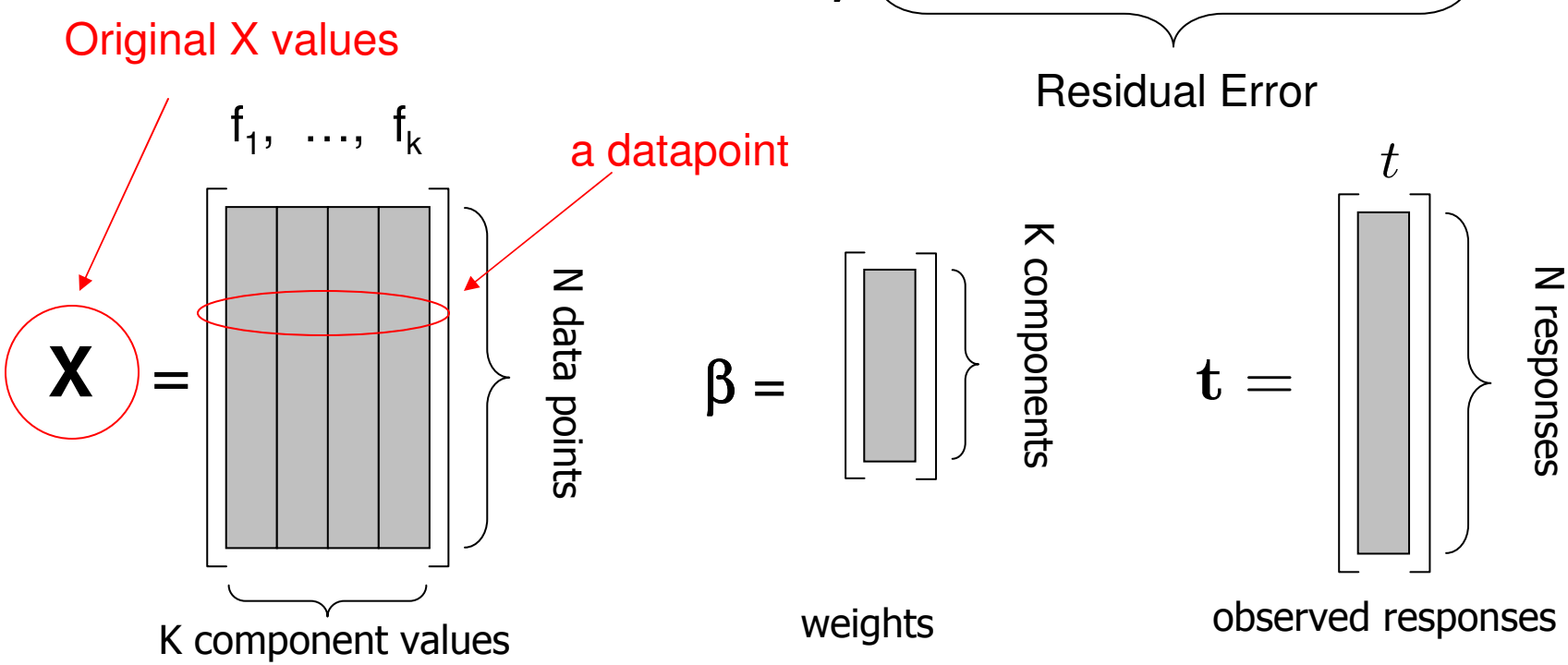
$$\arg\max_w \ln P(D \mid \boldsymbol{\beta}, \sigma) = \arg\min_w \left[\sum_j \left[t^j - \sum_i \beta_i x_i^j\right]^2\right]$$

- **Least-squares Linear Regression is MLE for Gaussians !!!**

Regression in Matrix Notation

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \sum_{j=1}^{N} \left[t^j - \sum_{i=1}^{K} \beta_i x_i^j \right]^2$$

$$= \arg\min_{\boldsymbol{\beta}} \underbrace{(\mathbf{X}\boldsymbol{\beta} - \mathbf{t})^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{t})}_{\text{Residual Error}}$$

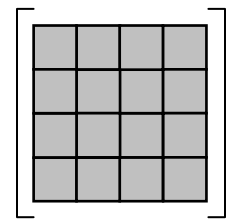
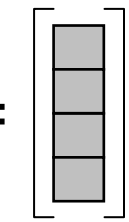


Regression solution
= simple matrix operations

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} (\mathbf{X}\boldsymbol{\beta} - \mathbf{t})^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{t})$$

Setting derivative to 0 yields:

$$Solution: \ \boldsymbol{\beta}^* = \underbrace{(\mathbf{X}^T\mathbf{X})^{-1}}_{\mathbf{A}^{-1}} \underbrace{\mathbf{X}^T\mathbf{t}}_{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{b}$$

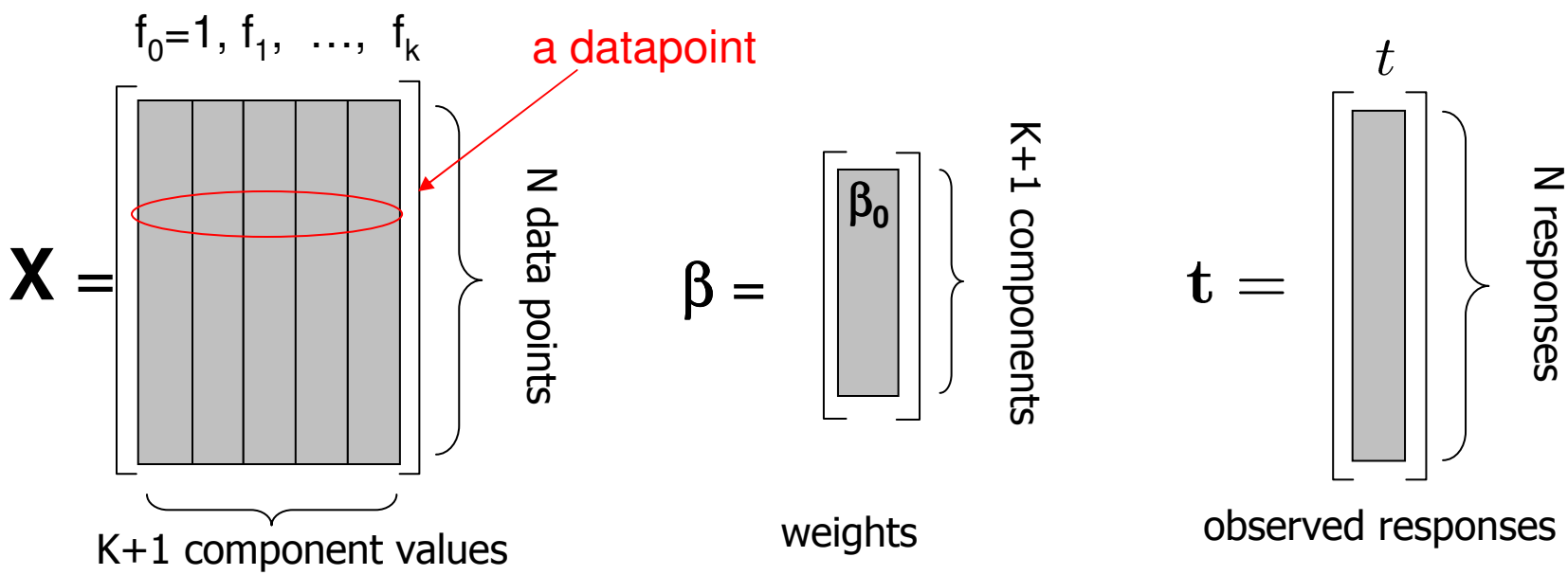
where $\mathbf{A} = \mathbf{X}^T\mathbf{X} =$  $\mathbf{b} = \mathbf{X}^T\mathbf{y} =$ 

K×K matrix
for k components

K×1 vector

Dealing with Offset

- *Actually want* $k+1$ *values* $(\beta_0, \beta_1, \ldots, \beta_k)$
  $y = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k$
- So view each k-tuple **x** as k+1 tuple [1, **x**]

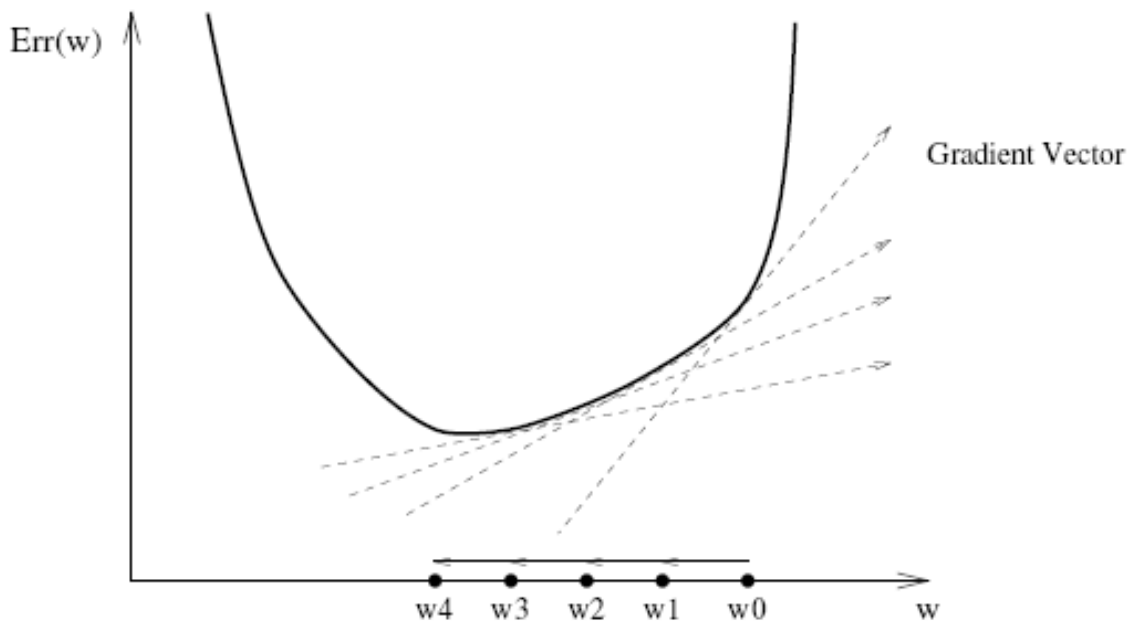


Another Approach: Gradient Descent

Skip 2009

- But matrix solution is $O(m^4 d)$
  - m = #training instances; d = #features
  - matrix inversion…
- Goal: Find w_i's that minimize squared error
  - $err(w) = (\sum_i [\, t^{(i)} - w\, x^{(i)}]^2 \,)/m$
- Why not use Gradient Descent!
  - aka Delta Rule, Adaline Rule, Widrow-Ho Rule, LMS Rule, Classical Conditioning

Local Search via Gradient Descent



Start w/ (random) weight vector $\mathbf{w}^0$.
Repeat until converged $\vee$ bored

Compute Gradient

$$\nabla \mathrm{err}(\mathbf{w}^t) = \left(\frac{\partial \mathrm{err}(\mathbf{w}^t)}{\partial w_0}, \frac{\partial \mathrm{err}(\mathbf{w}^t)}{\partial w_1}, \ldots, \frac{\partial \mathrm{err}(\mathbf{w}^t)}{\partial w_n}\right)$$

Let $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \nabla \mathrm{err}(\mathbf{w}^t)$

If CONVERGED: Return($\mathbf{w}^t$)

Gradient-Descent vs Matrix-Inversion

Pro: Gradient Descend advantages

- ≈Biologically plausible
- Each iteration costs only *O(mn)*
- If uses < m iterations,
  faster than Matrix Inversion!
- More easily parallelizable

Con: Gradient Descent disadvantages

- It's moronic...
  essentially a slow way to build X^TX matrix,
  then solve a set of linear equations
- If n is small, it's especially outrageous.
  If n is large then direct matrix inversion method can be problematic
  but not impossible if you want to be efficient.
- Need to choose a good learning rate -- how?
- Matrix inversion takes predictable time.
  You can't be sure when gradient descent will stop.

Computing the Gradient

$$\boxed{err(w) = (\textstyle\sum_i [\, t^{(i)} - w\, x^{(i)}]^2)\ /m}$$

$$\frac{\partial\, err(w)}{\partial\, w_j} = \frac{\partial}{\partial\, w_j}\left(\frac{1}{m}\sum_{i=1}^{m} err_i(w)\right) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\, w_j} err_i(w)$$

$$\begin{aligned}
\frac{\partial\, err_i(w)}{\partial\, w_j} &= \frac{\partial}{\partial\, w_j}\left[\left(\sum_{i=1}^{m} w_j x_{ij}\right) - t_i\right]^2 \\
&= 2\cdot\left[\left(\sum_{i=1}^{m} w_j x_{ij}\right) - t_i\right]\cdot\frac{\partial}{\partial\, w_j}\left[\left(\sum_{i=1}^{m} w_j x_{ij}\right) - t_i\right] \\
&= 2\cdot\left[\left(\sum_{i=1}^{m} w_j x_{ij}\right) - t_i\right]\cdot x_{ij}
\end{aligned}$$

Then descend a distance η along gradient $\left[\frac{\partial\, err(w)}{\partial\, w_0}, \frac{\partial\, err(w)}{\partial\, w_1}, \ldots, \frac{\partial\, err(w)}{\partial\, w_n}\right]$

Gradient Descent Algorithm

Gradient-Descent(S : training examples; $\eta \in \Re^+$)

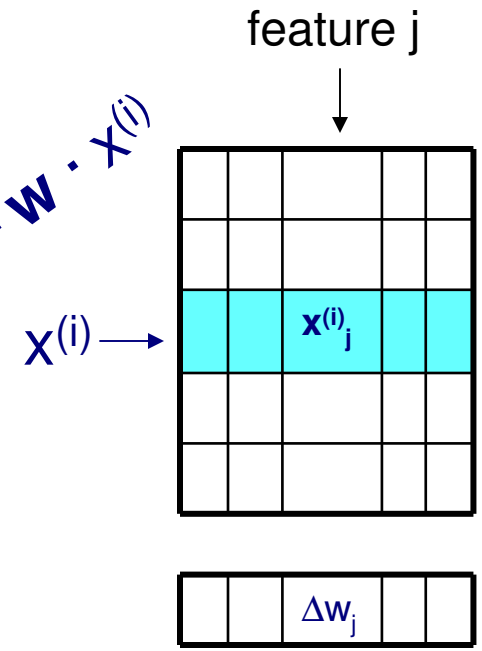
% $S = \{ [x^{(i)} , t^{(i)}] \},\ldots$

% x = vector of input values; t is target output value

% η is learning rate (eg, 0.05)

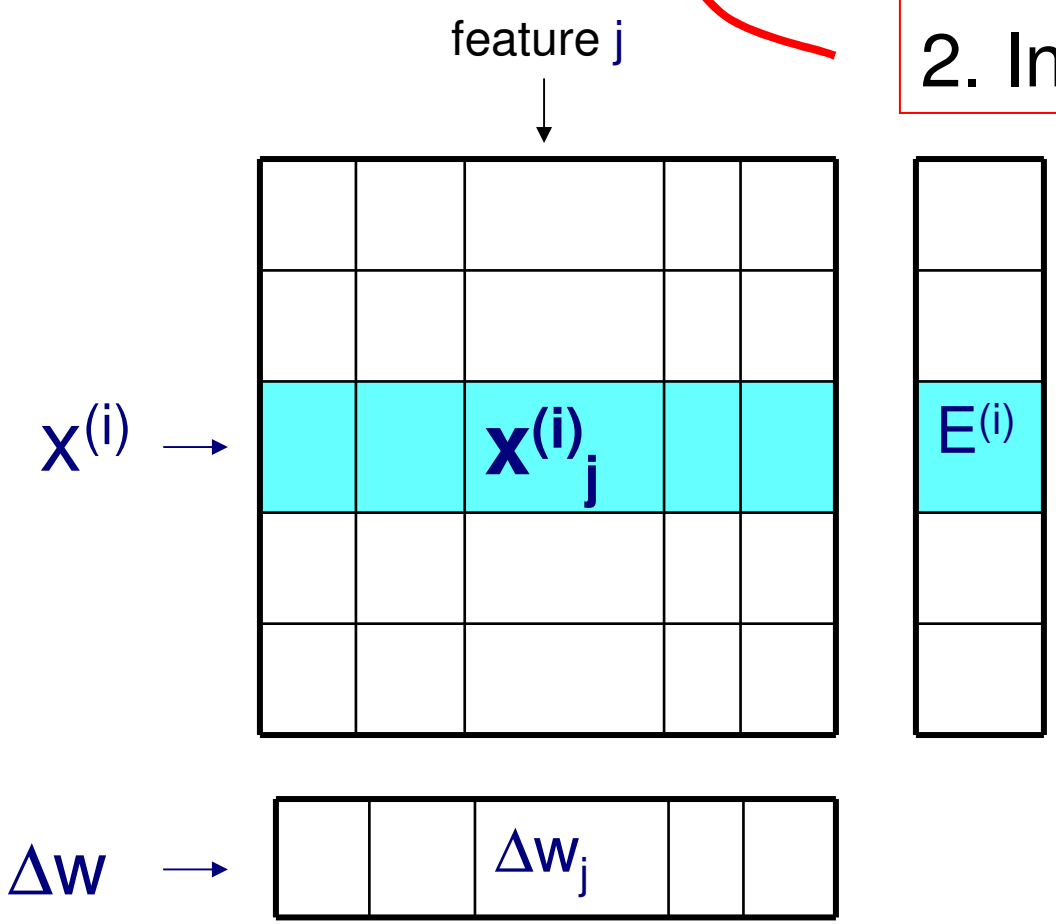
- Initialize each w_j to small random value
  - Typically $\in$ [–0.05, +0.05]
- Until termination condition is met, do
  - Initialize each $\Delta w_j \leftarrow 0$
  - For each $[x^{(i)} , t^{(i)}] \in S$, do
    - Set $E^{(i)} \leftarrow t^{(i)} - \mathbf{w} \cdot x^{(i)}$
    - For each j, do
      - $\Delta w_j \leftarrow \Delta w_j + r^{(i)} x^{(i)}_j$
  - For each j do
    - $w_j \leftarrow w_j + \eta \, \Delta w_j$
- Return w

$E^{(i)} \leftarrow t^{(i)} - \mathbf{w} \cdot x^{(i)}$



0. New $\mathbf{w}$
1. For each row i, compute
   a. $\Delta\mathbf{w} = 0$
   b. $E^{(i)} = t^{(i)} - \mathbf{w} \cdot x^{(i)}$
   c. $\Delta\mathbf{w} \mathrel{+}= E^{(i)}\, \mathbf{x}^{(i)}$

      $[\ \ldots\ \Delta w_j \mathrel{+}= E^{(i)}\, x^{(i)}_j \ldots\ ]$
2. Increment $w \mathrel{+}= \eta\, \Delta\mathbf{w}$



Batch Gradient Descent:

```
Do until satisfied
1.  Compute gradient ∇err_S[w]
2.  w ← w − η∇err_S[w]
```

On-Line Gradient Descent:

```
Do until satisfied
  For each training example ⟨x, y⟩ in S
  1.  Compute gradient ∇err_(x,y)[w]
  2.  w ← w − η∇err_(x,y)[w]
```

$$\begin{aligned} \mathrm{err}_{(x,y)}[\mathrm{w}] &\equiv (y - \mathrm{w}\cdot x)^2 \\ \mathrm{err}_S[\mathrm{w}] &\equiv \sum_{(\mathrm{x},y)\in S} \mathrm{err}_{(\mathrm{x},y)}[\mathrm{w}] = \sum_{(\mathrm{x},y)\in S} (y - \mathrm{w}\cdot \mathrm{x})^2 \end{aligned}$$

On-Line Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η small enough

Comments on On-Line Mode

- Stochastic gradient descent may avoid local minima as “noisier”
  - aka “Stochastic Gradient Descent“, “Robbins-Munro algorithm"
- Some versions store all training examples; make repeated passes through them until convergence

Learning Rates and Convergence

- Learning rate $\eta \equiv$ "step size"
- Convergence whenever…
  - $\lim_{t\to\infty}\eta_t = 0$
  - $\sum \eta_t = \infty$
  - $\sum \eta_t^2 < \infty$
- $\exists$ sophisticated alg's

  (Newton's method; Line Search; …)

  that choose step size automatically, converge faster.
- $\exists$ only one "basin" for linear threshold units

  $\Rightarrow$ local minimum is global minimum!
- Good starting point $\Rightarrow$ algorithm converges faster

Results wrt Gradient Descent

Gradient descent (Delta training rule)

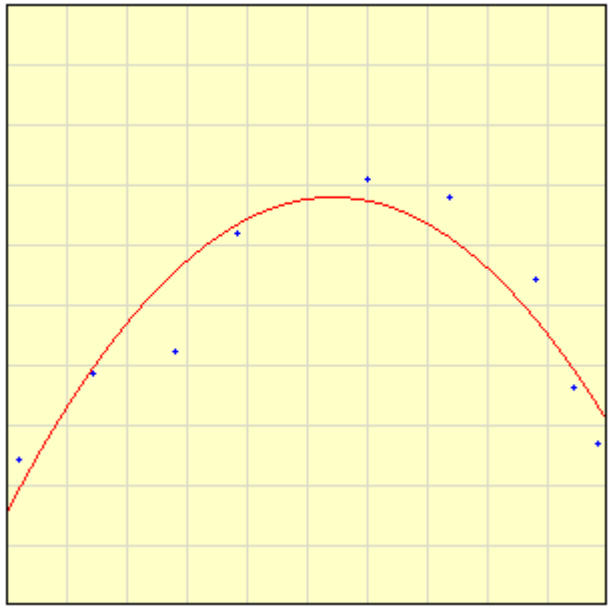
- guaranteed to converge to hypothesis with minimum squared error (eventually!)
  if
- Sufficiently small learning rate η
- . . . even when training data
  - contains noise
  - not separable!

What about other features?

- Data:
  - Not linear!!
  - Perhaps

$$f(x) = \alpha_2 x^2 + \alpha_1 x + \alpha_0$$

- How to fit ???

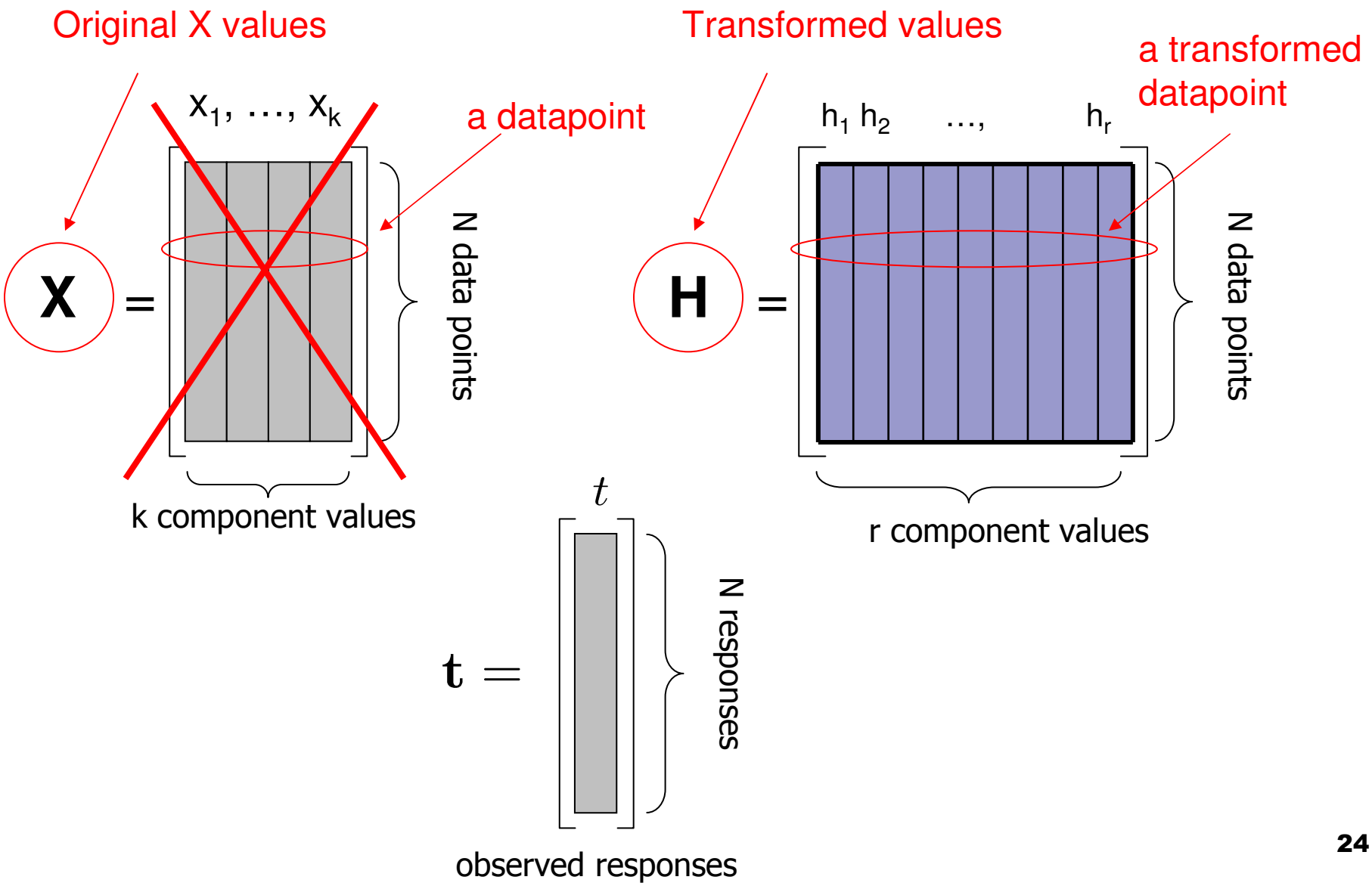


t	~~x~~	x^2	x	1
18	~~5~~	25	5	1
11	~~-2~~	4	-2	1
38	~~7~~	49	7	1
6	~~3~~	9	3	1

$\alpha_2 = 1$
$\alpha_1 = -2$
$\alpha_0 = 3$

$\alpha_2 = -0.179$
$\alpha_1 = 1.938$
$\alpha_0 = 1.543$

General Approach



General Linear Regression Task

- **Given set of labeled Instances:** $\{ [\mathbf{x}_j, t_j] \}$
- **Learn:** Mapping from x to $t(\mathbf{x})$
  - Can use **BASIS** functions: $H = \{ h_1(\mathbf{x}), \ldots h_r(\mathbf{x}) \}$
    - Eg: x_i^2, x_i^3, $(x_1\ x_3)$, $x_i \sin(x_i)$, …
  - (Basis) linear mapping: $t(\mathbf{x}) \approx \sum_j \beta_j h_j(\mathbf{x})$
  - Find coeffs $\beta = (\beta_1, \ldots, \beta_r)$
- **Model**: Observed value $t^*(x) = \sum_j \beta_j h_j(\mathbf{x}) + \varepsilon$ where $\varepsilon \sim N(0, \sigma^2)$

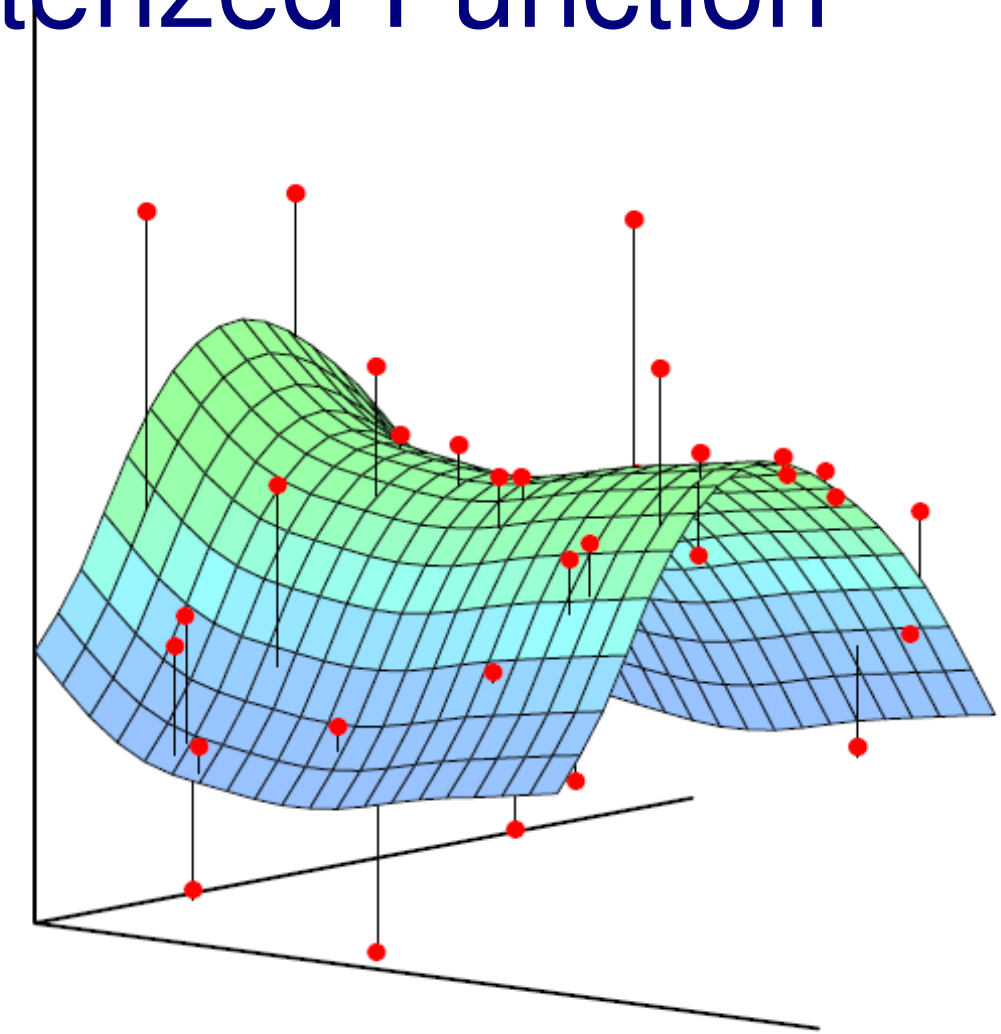
Model is LINEAR in these bases… even if bases are NOT linear

Features/Basis Functions

- Polynomials
  - $1, x, x^2, x^3, x^4, ..$
- Indicators
- Gaussian densities
- Step functions or sigmoids
- Sinusoids (Fourier basis)
- Wavelets
- Anything you can imagine…

Fitting Parameterized Function

- *Least squares fitting of a function of two inputs*
- *Find parameters of* $f_\theta(x)$ *that minimize the sum-of-squared vertical errors*



What if $t^{(i)}$ is a vector?

- Nothing changes!
- Scalar prediction:

$$Solution: \ \boldsymbol{\beta}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$$

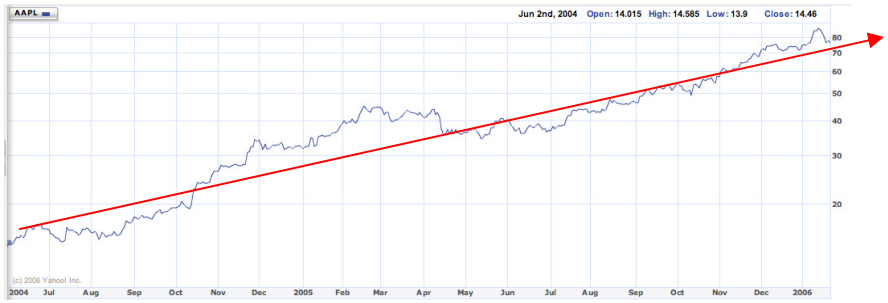
- Vector prediction:

$$Solution: \ \boldsymbol{\beta}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{T}$$

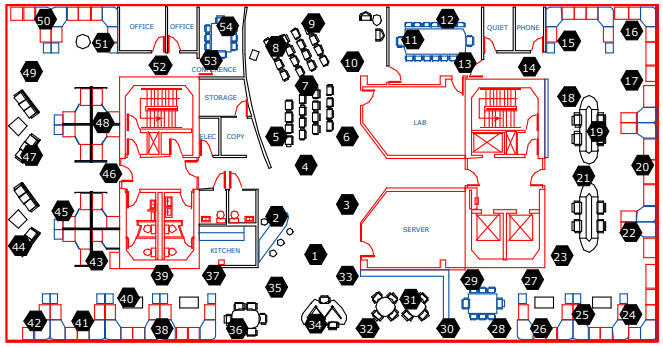
Target MATRIX

Applications Corner 1

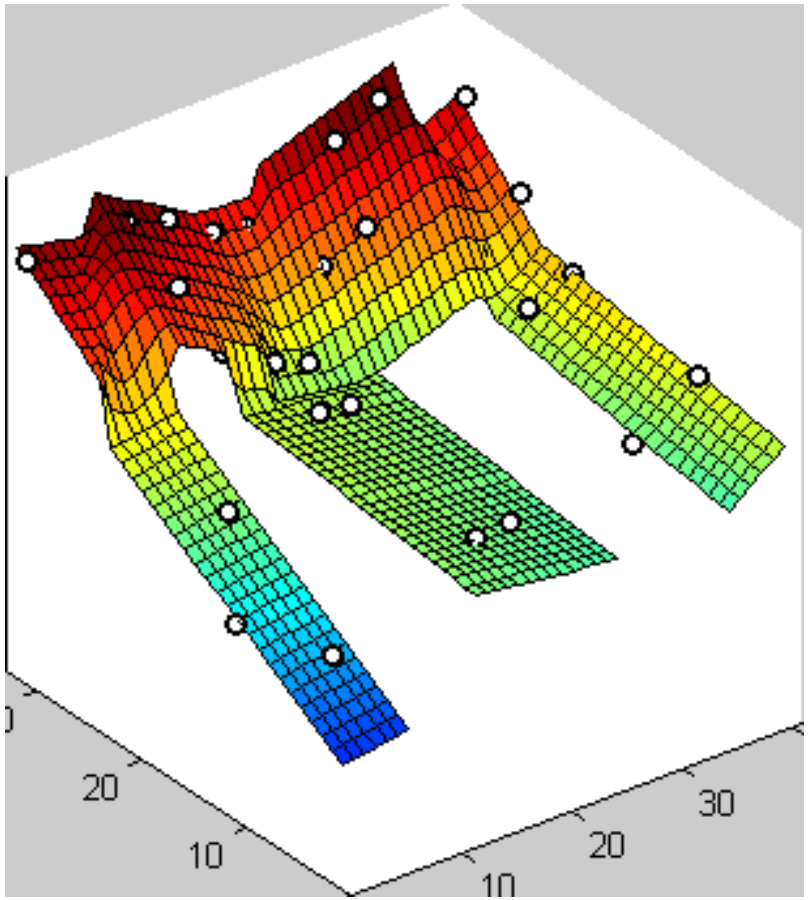
- Predict stock value over time from
  - past values
  - other relevant vars
    - e.g., weather, demands, etc.



Applications Corner 2



- Measure temperatures at some locations
- Predict temperatures throughout the environment

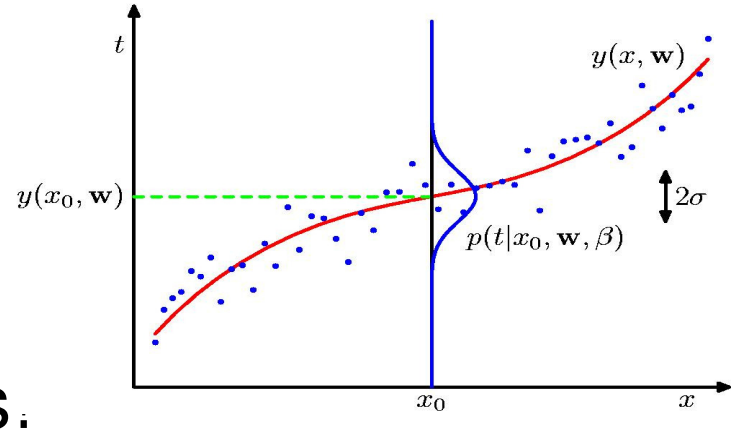


[Guestrin et al. ’04]

Applications Corner 3

- Predict when a sensor will fail
  - based several variables
    - age, chemical exposure, number of hours used,…

Outline



- Linear Regression
  - MLE = Least Squares.
  - Basis functions
- Evaluating Predictors
  - Training set error vs Test set error
  - Cross Validation
- Model Selection
  - Bias-Variance analysis
  - Regularization, Bayesian Model

Training Set Error

- Choose a loss function
  - eg, squared error (L_2) for regression
- Given a labeled dataset S, learn optimal predictor $\mathbf{w}_S$

S = "Training data"

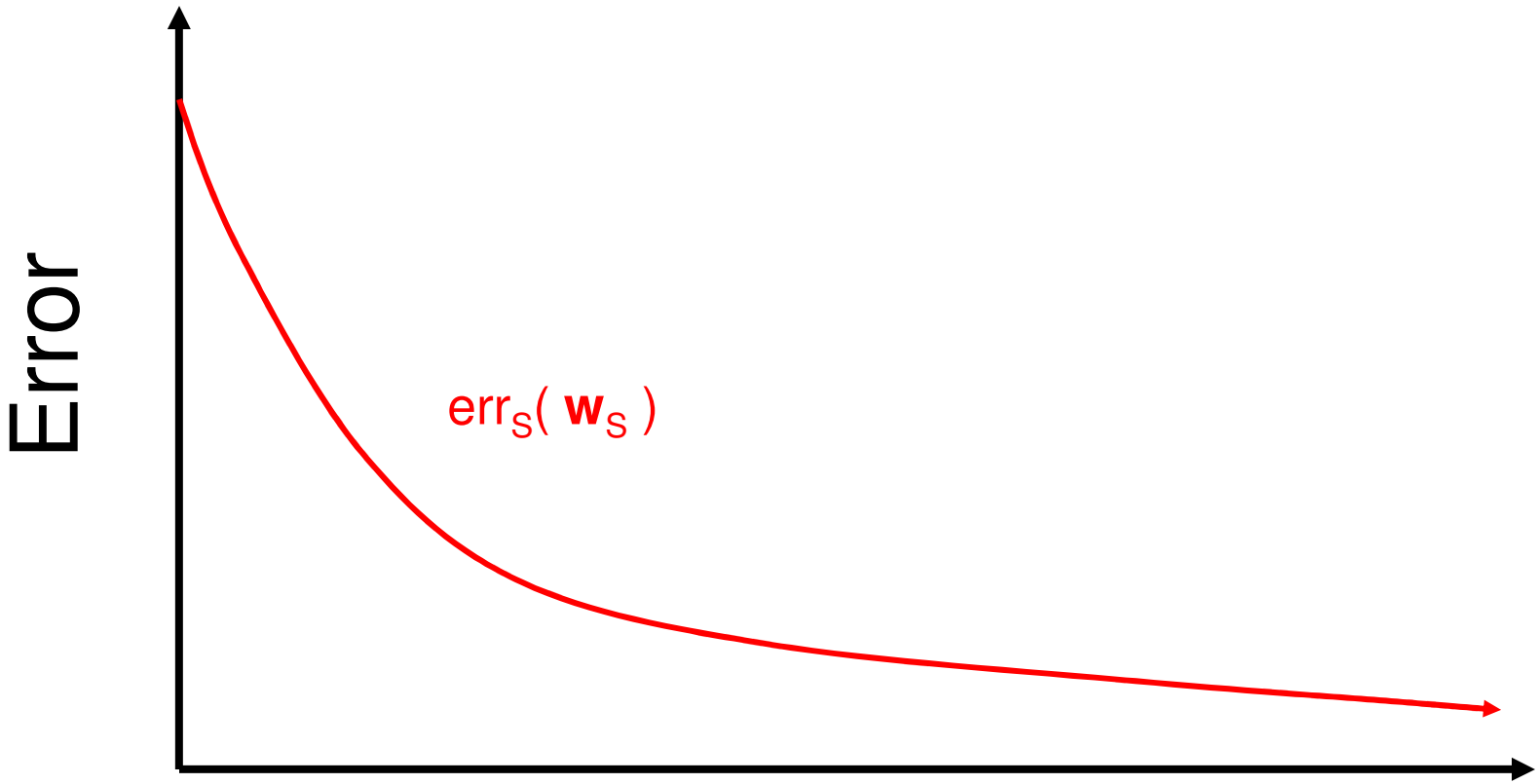
$$\mathbf{w}_S = \mathbf{w}^*(S) = \arg\min_{\mathbf{w}} \sum_{(\mathbf{x},t)\in S} \left(t - \sum_i w_i h_i(\mathbf{x}) \right)^2$$

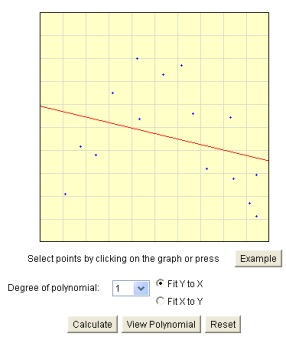
  - compute empirical error (of any $\mathbf{w}$)

$$\mathrm{err}_S(\mathbf{w}) = \frac{1}{|S|} \sum_{(\mathbf{x},t)\in S} \left(t - \sum_i w_i h_i(\mathbf{x}) \right)^2$$

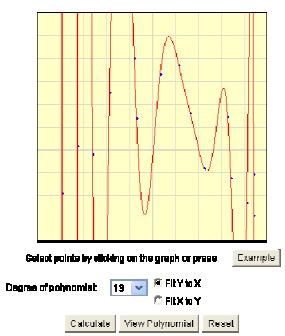
- **Training set error:** $\mathrm{err}_S(\mathbf{w}_S)$

Training Set Error as a function of Model Complexity

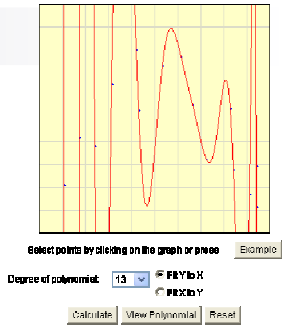




"Model Complexity"
… eg, #basis functions;
degree of poly, …



True Prediction Error

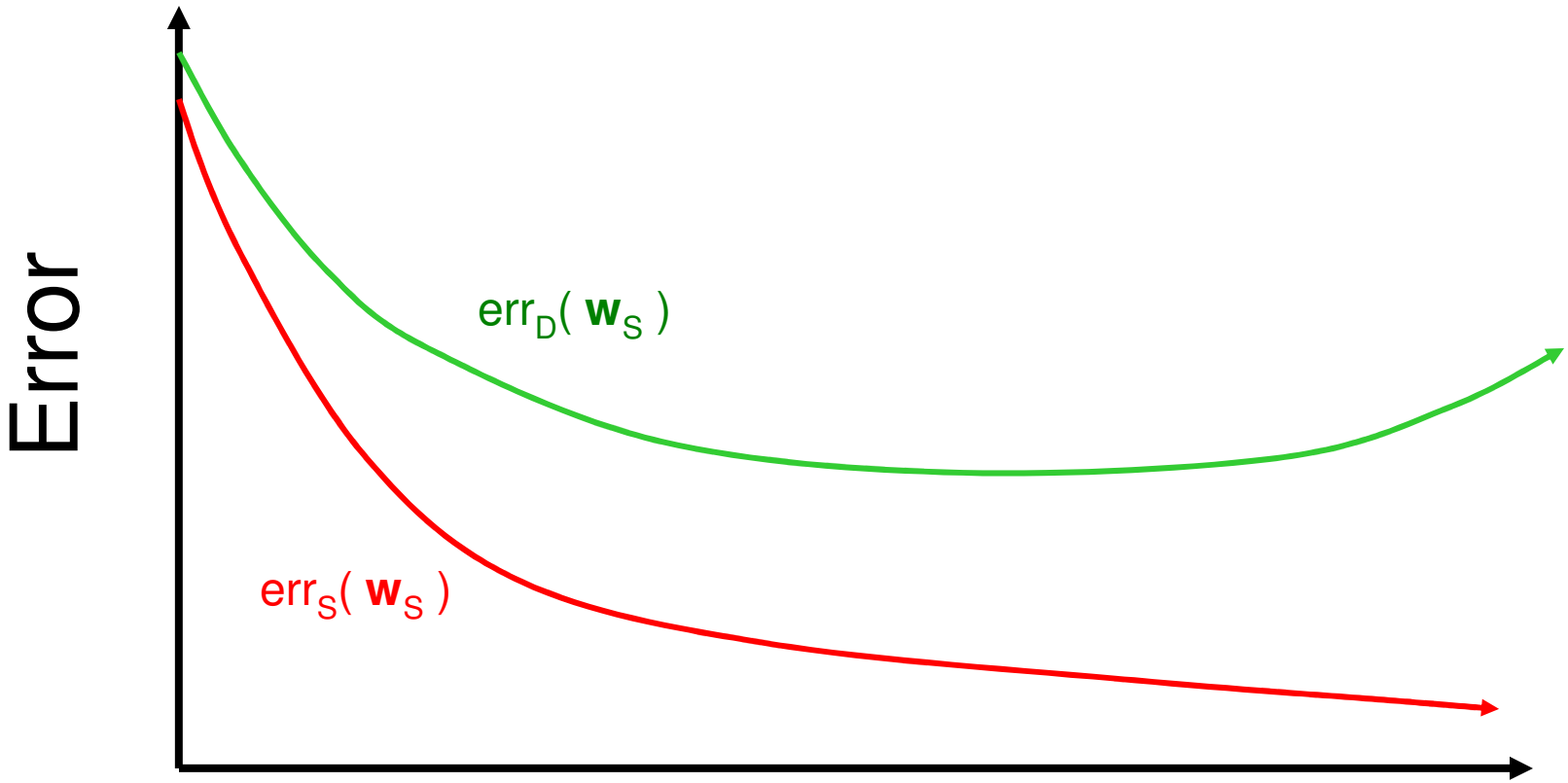


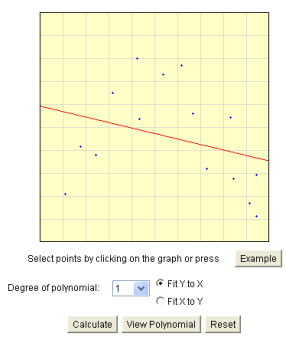
- WARNING: Training set error can be poor measure of "quality" of solution
- **Want:** *error over all possible input points*, **not just training data**:
  - **Prediction error:**

$$
\begin{aligned}
err_D(\mathbf{w}) &= E_{x,t}\left[\left(t(\mathbf{x}) - \sum_i w_i h_i(\mathbf{x})\right)^2\right] \\
&= \int_{x,t} \left(t - \sum_i w_i h_i(\mathbf{x})\right)^2 D(\mathbf{x}, \mathrm{t})\; dx\, dt
\end{aligned}
$$

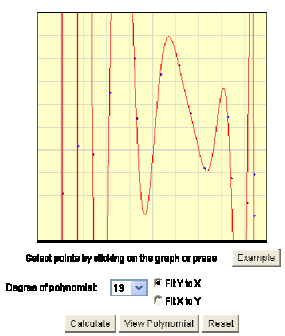
Requires D(**x**,t) – unknown!

Prediction Error as a function of Model Complexity

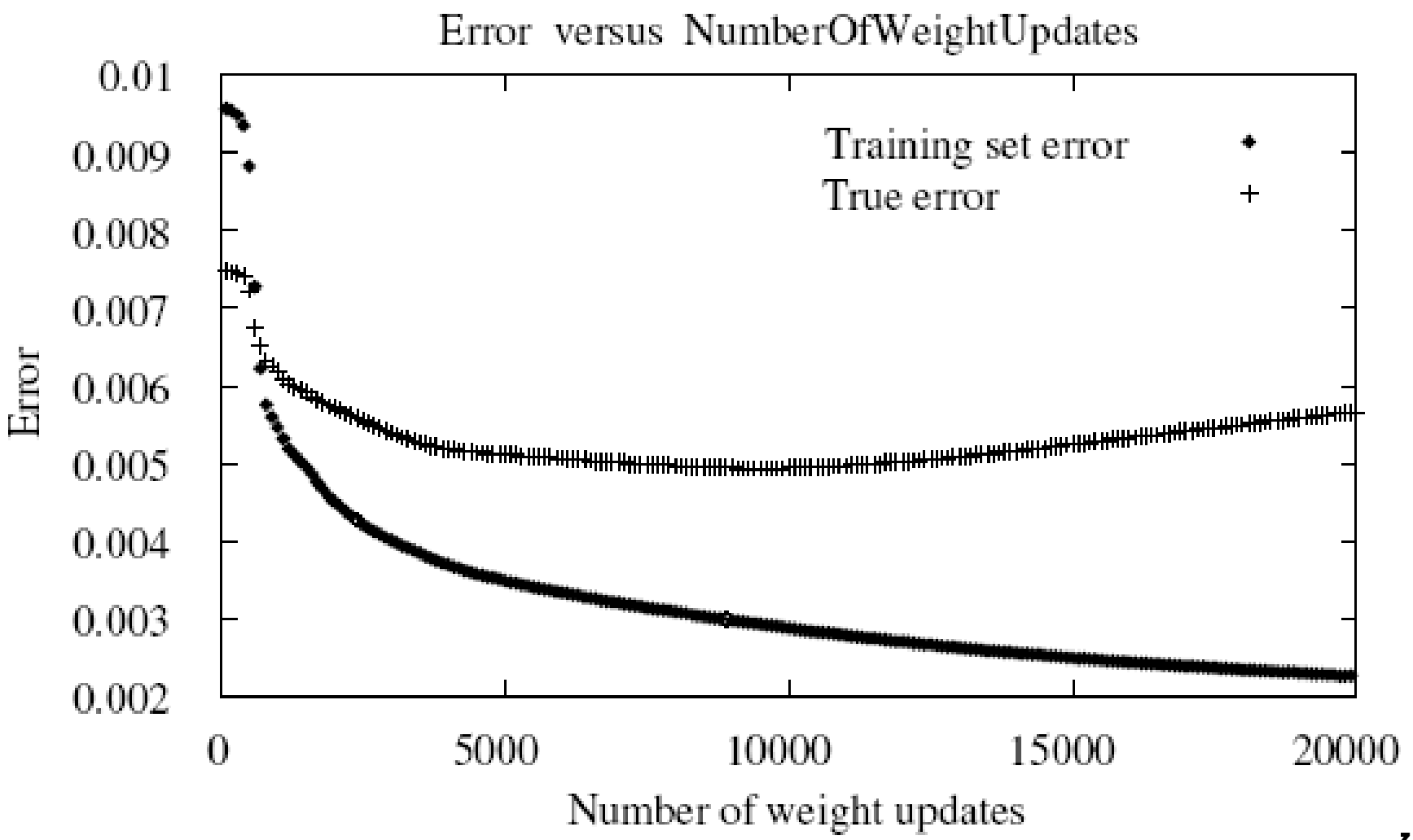




"Model Complexity"
… eg, #basis functions;
degree of poly, …



$\underline{err}_S(\mathbf{w}_S)$ vs $err_D(\mathbf{w}_S)$

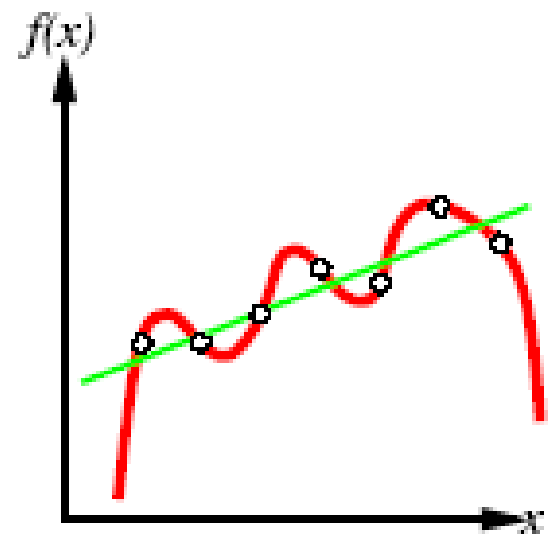


But …

- $\underline{err}_S(\mathbf{w}_S) \neq err_D(\mathbf{w}_S)$

- $\underline{err}_S(\mathbf{w}_S) \equiv$
  Eval $\mathbf{w}_S$ on training set S
  - only approximation to $err_D(\mathbf{w}_S)$
  - ... can be TOO optimistic!

- "Cheating"
  Like being evaluated on test
  after seeing SAME test. . .



$\underline{err}_S(\mathbf{w}) = 0$
$\underline{err}_S(\mathbf{w}) > 0$

Computing Prediction Error

- Computing prediction

$$err_D(\mathbf{w}) = \int_{x,t} \left(t - \sum_i w_i h_i(\mathbf{x})\right)^2 D(\mathbf{x}, \mathrm{t})\, dx\, dt$$

  - Depends on $D(\mathbf{x}, t)$ for every $\mathbf{x}$ – typically not known
  - Hard integral

- New sample: a set of i.i.d. points

$S' = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_M, t_M)\}$ from $D(\mathbf{x}, t)$

$$\mathrm{err}_D(\mathbf{w}_S) \approx \mathrm{err}_{S'}(\mathbf{w}_S) = \frac{1}{|S'|} \sum_{(\mathbf{x},t)\in S'} \left(t - \sum_i w_i h_i(\mathbf{x})\right)^2$$

Training Error ≠ Prediction Error

- Sampling approximation of prediction error:

$$err_{S'}(\mathbf{w}_S) \approx err_D(\mathbf{w}_S)$$

- Training error :

$$err_S(\mathbf{w}_S) \not\approx err_D(\mathbf{w}_S)$$

- Very similar equations!!!
  - Why is *training error* a bad measure of *prediction error*?

Training Error ≠ Prediction Error

Because you cheated!!!

- r:

Training error is good estimate for a single **w,**
But you optimized **w** with respect to the training error,
and found **w** that is good for *this set of instances*

- **Training error is a (optimistically) biased estimate of prediction error**

- Very similar equations!!!
  - Why is *training error* a bad measure of *prediction error*?

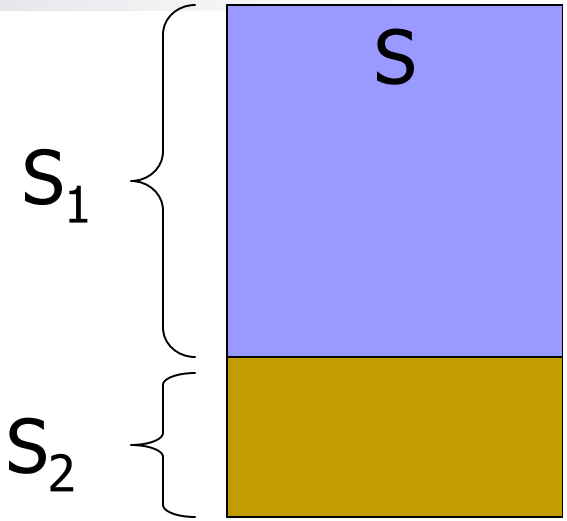
Test Set Error

$$\mathbf{w}_S = \mathbf{w}^*(S) = \arg\min_{\mathbf{w}} \sum_{(\mathbf{x},t)\in S} \left(t - \sum_i w_i h_i(\mathbf{x}) \right)^2$$

- **Randomly** split dataset into two parts:
  - Training data – $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_{Ntrain}\}$
  - Test data – $S' = \{\mathbf{x}_{Ntrain+1}, \ldots, \mathbf{x}_{Ntrain+Ntest}\}$
- Use *training data* to optimize $\mathbf{w} = \mathbf{w}_S$
- **Test set error:**
  Given $\mathbf{w}_S$, estimate error using:

$$\mathrm{err}_{S'}(\mathbf{w}_S) = \frac{1}{|S'|} \sum_{(\mathbf{x},t)\in S'} \left(t - \sum_i w_i h_i(\mathbf{x}) \right)^2$$

Estimating Error: Hold-Out Set



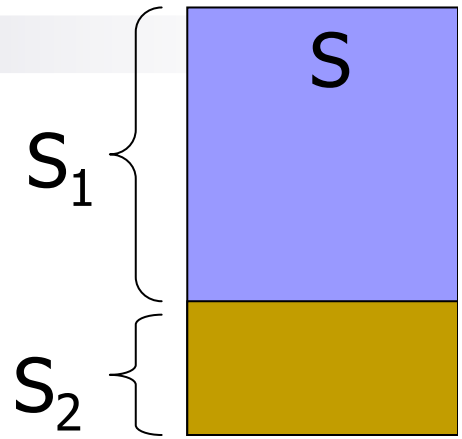
- Run learner $L(.)$ on S
  – produce regressor $w_S = L(S)$
  What is true error $err_D(w_S)$?
- Want to return $[w_S, err_D(w_S)]$
  … or at least $[w_S, e]$ where $e \approx err_D(w_S)$

- Divide S into disjoint S_1, S_2
  - *Train* on S_1: computing $w_{S_1} := L(S_1)$
  - *Test* on S_2: $\underline{err}_{S_2}(w_{S_1})$

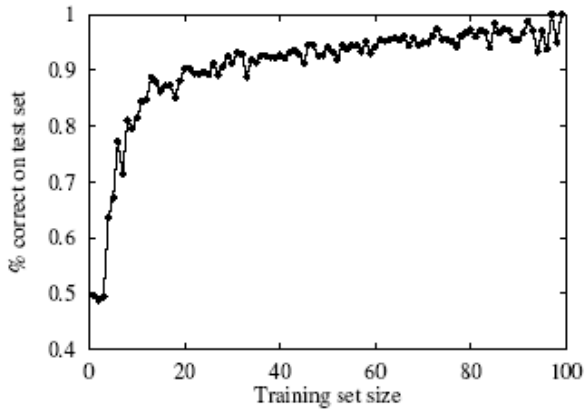
  Return $[w_S, \underline{err}_{S_2}(w_{S_1})]$

- Why is $\underline{err}_{S_2}(w_{S_1}) \approx err_D(w_S)$?
  - As $S_1 \approx S$, $w_{S_1} = L(S_1) \approx L(S) = w_S$
  - $\underline{err}_{S_2}(w_{S_1})$ is estimate of $err_D(w_{S1}) \approx err_D(w_S)$

Challenge wrt Hold-Out Set



- How to divide S into disjoint S_1, S_2
- As $|S_1| < |S|$,
  $L(S_1)$ not as good as $L(S)$
  Learning curve: $L(S)$ improves as $|S|$ increases)
  $\Rightarrow$ want S_1 to be large

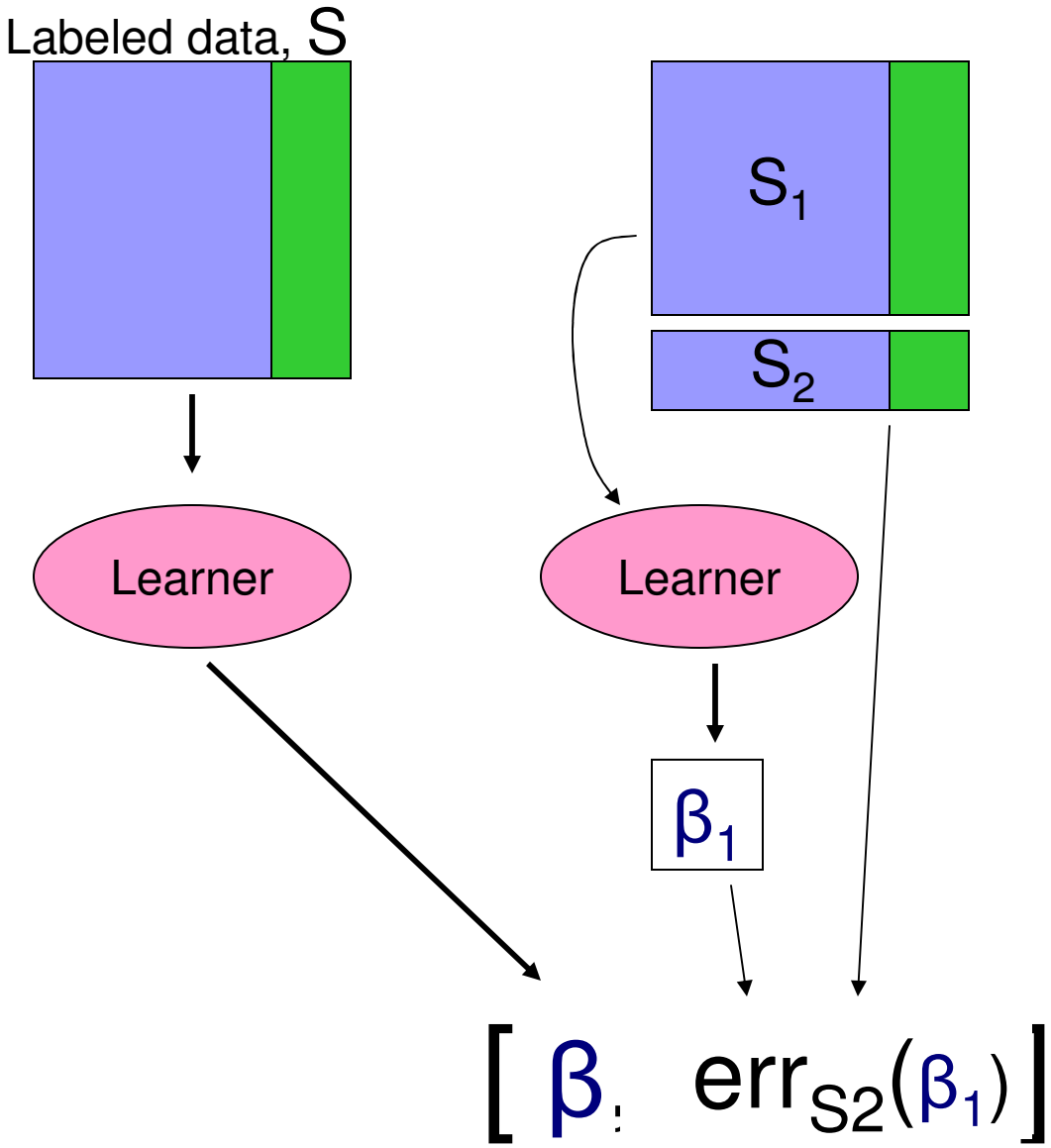


- $\underline{err}_{S_2}(w_{S_1})$ is estimate of $err_D(w_S)$
  Estimate improves as S_2 gets larger
  $\Rightarrow$ want S_2 to be as large as possible

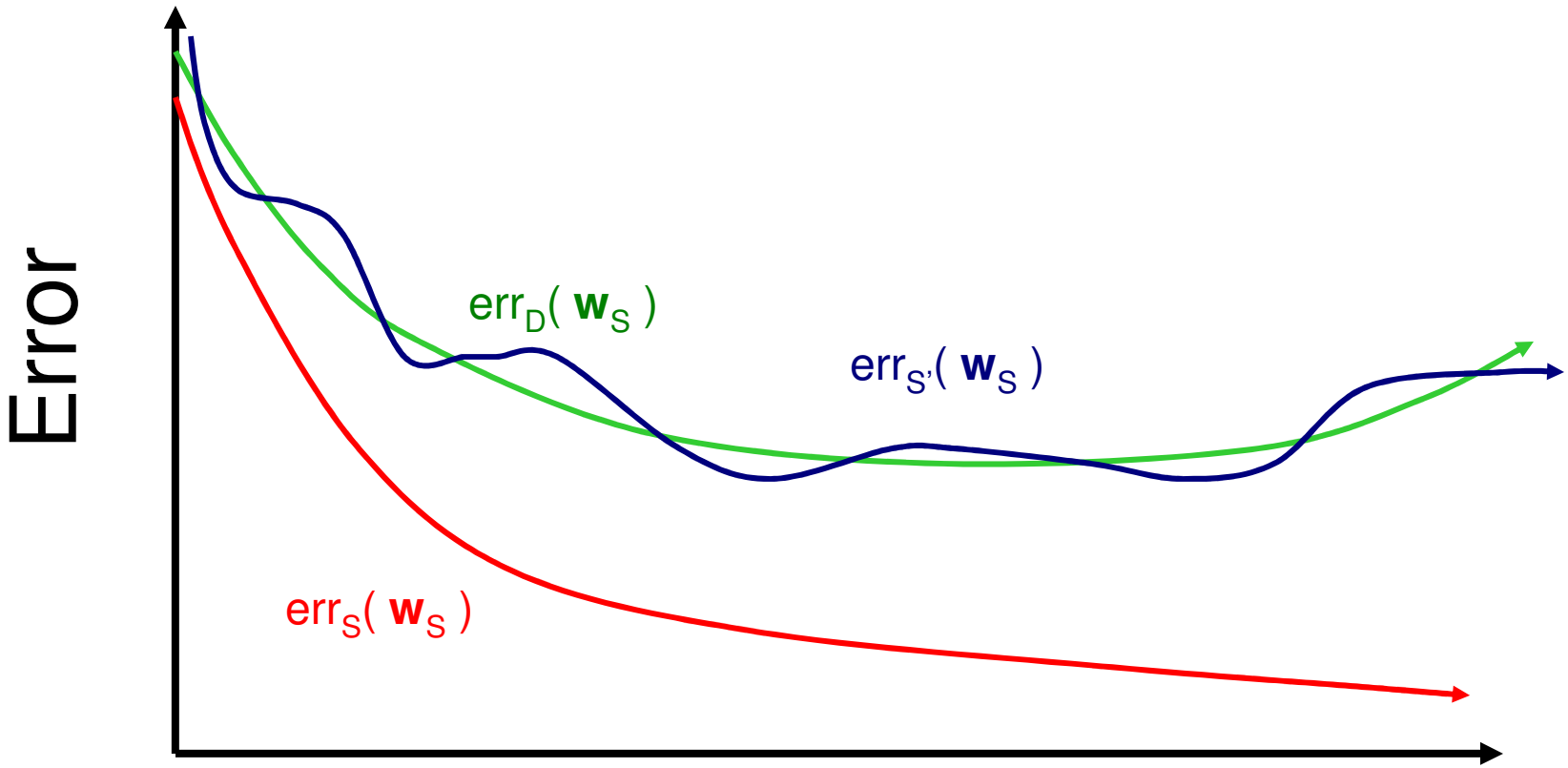
$$|err_S(h) - err_D(h)| \approx \frac{\alpha}{\sqrt{|S|}}$$

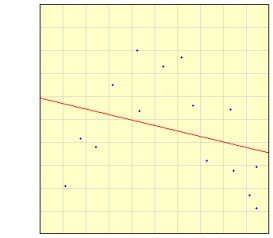
- As $S = S_1 \cup S_2$, must trade off
  quality of classifier $w_{S_1} = L(S_1)$
  with
  accuracy of estimate $\underline{err}_{S_2}(w_{S_1})$

Return: [Predictor + Est Quality]

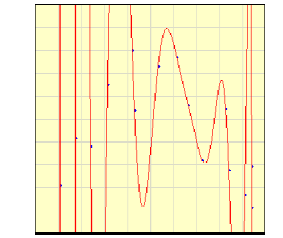


Test Set Error as a function of Model Complexity

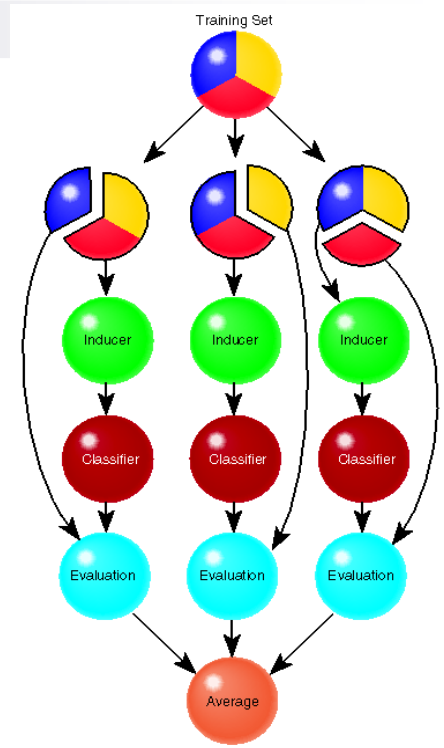




"Model Complexity"
… eg, #basis functions;
degree of poly, …



Estimating Error: Cross Validation



- "Cross-Validation"

CV(data S, alg L, int k)
  Divide S into k disjoint sets $\{ S_1, S_2, \ldots, S_k \}$
  For i = 1..k do
    Run L on $S_{-1} = S - S_i$
    obtain $h_i := L(S_{-i})$
    Evaluate h_i on S_i
      $err_{S_i}(h_i) = 1/|S_i| \sum_{\langle x,t \rangle \in S_i} [h_i(x) - t]^2$
  Return Average $1/k \sum_i err_{S_i}(h_i)$

$\Rightarrow$ Less Pessimistic

as train on $(k - 1)/k\ |S|$ of the data

Comments on Cross-Validation

- Every point used as
  Test 1 time, Training k – 1 times
- Computational cost for *k*-fold Cross-validation … linear in *k*
- Should use "balanced CV"
  If class c_i appears in m_i instances,
  insist each S_k include $\approx \frac{1}{k}\frac{m_i}{|S|}$ such instances
- Use **CV(S, L, k)** as ESTIMATE of true error of L(S)
  Return L(S) and CV(S, L, k)
- Leave-One-Out-Cross-Validation k = m !
  - eg, for Nearest-Neighbor
- Notice different folds are correlated
  as training sets overlap: (k-2)/k unless k=2
- 5 × 2-CV
  - Run 2-fold CV, 5 times. . .

Can use CV to estimate parameters in general!

To Form k *Balanced* Folds

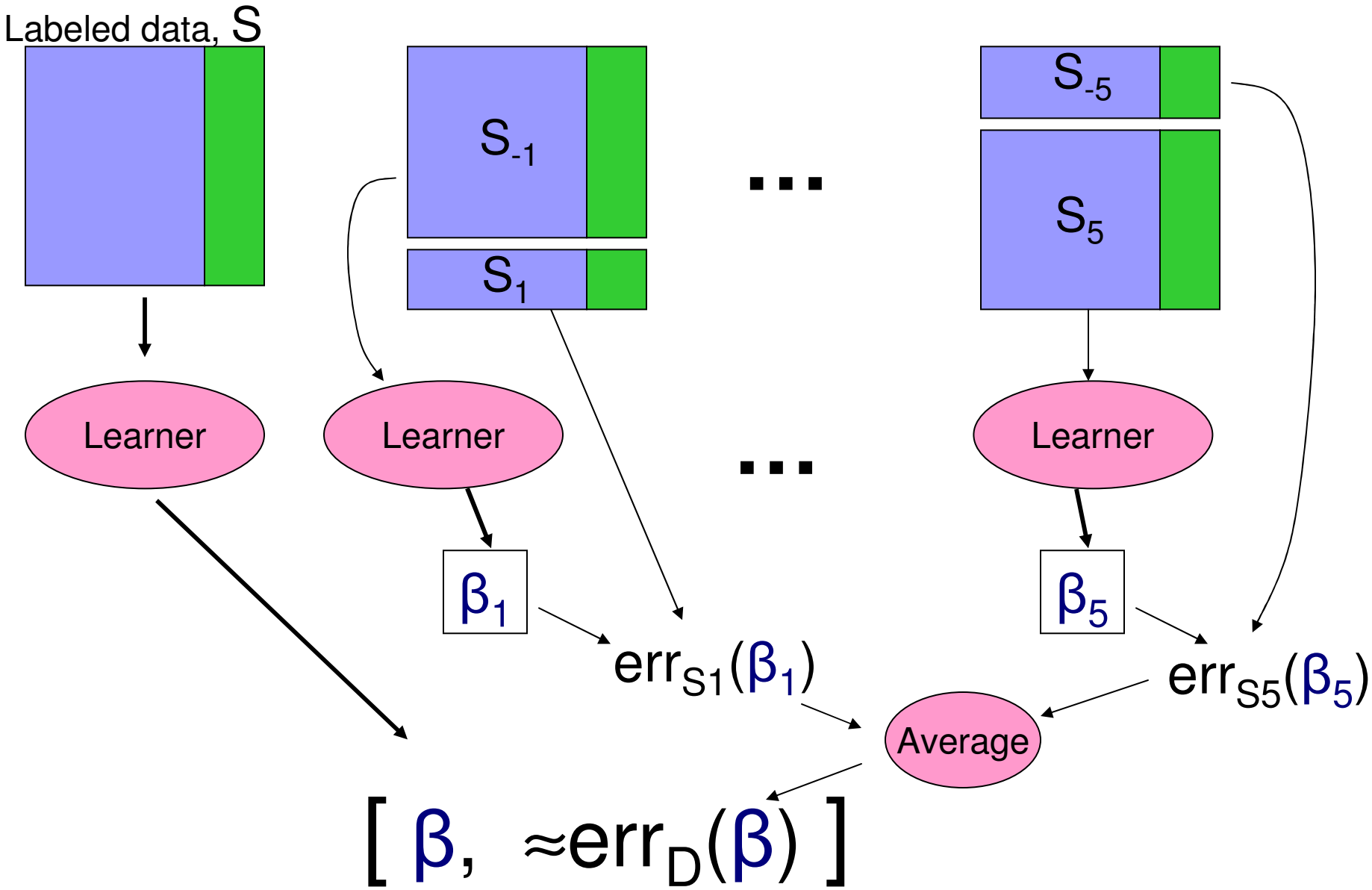
1. Partition the data S based on the class:
   - subset S_+ has all the positive instances,
   - subset S_- has all the negative instances.

2. Randomly partition each subset into k folds:

$$S_+ = \cup \{ S_{+1}, \ldots, S_{+k} \}$$

3. $S_j = S_{+j} \cup S_{-j}$ for j=1..k

Return: [Predictor + Est Quality]



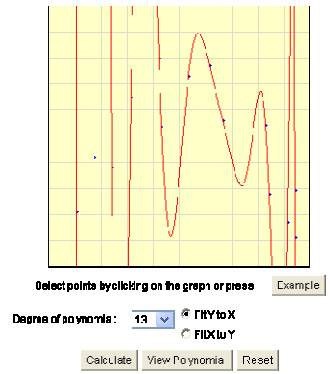
How many points needed for training/testing?

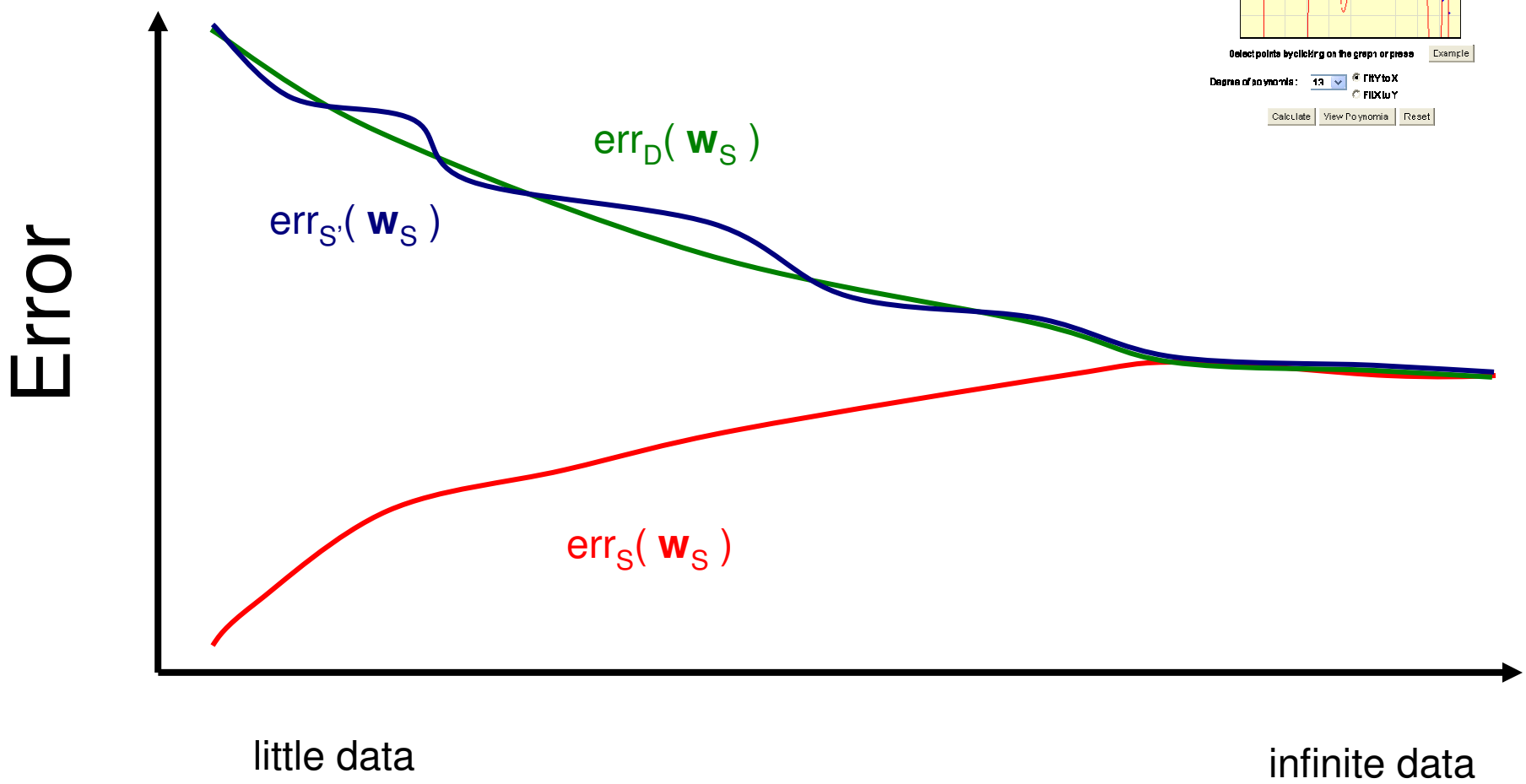
- Very hard question to answer!
  - Too few training points, learned **w** is bad
  - Too few test points, you never know if you reached a good solution
- Bounds, such as Hoeffding's inequality can help:

$$P(|\ \widehat{\theta} - \theta^* \ | \geq \epsilon) \ \ \leq \ \ 2e^{-2N\epsilon^2}$$

- More on this later this semester, but still hard to answer
- Typically:
  - if you have a reasonable amount of data, pick test set "large enough" for a "reasonable" estimate of error, and use the rest for learning
  - if you have little data, then you need to pull out the big guns…
    - e.g., bootstrapping

Error as function of #Training Examples, for a fixed Model Complexity





Error Estimators

$$err_D(\mathbf{w}) = \int_{x,t} \left(t - \sum_i w_i h_i(\mathbf{x})\right)^2 D(\mathbf{x}, \mathrm{t})\, dx\, dt$$

Gold Standard!
Unbiased

Uses TRAIN data
… **optimistic**

$$\mathrm{err}_S(\mathbf{w}_S) = \frac{1}{|S|} \sum_{(\mathbf{x},t)\in S} \left(t - \sum_i w_i h_i(\mathbf{x})\right)^2$$

Approx truth…
Unbiased
… if you are careful

$$\mathrm{err}_{S'}(\mathbf{w}_S) = \frac{1}{|S'|} \sum_{(\mathbf{x},t)\in S'} \left(t - \sum_i w_i h_i(\mathbf{x})\right)^2$$

Error Estimators

Be careful!!!

Test set only unbiased if you *never never never never* do *any any any* learning/adjustment/… on the test data

Eg,
if you use the test set to select the degree of the polynomial…
no longer unbiased!!!
(We will address this problem later in the semester)

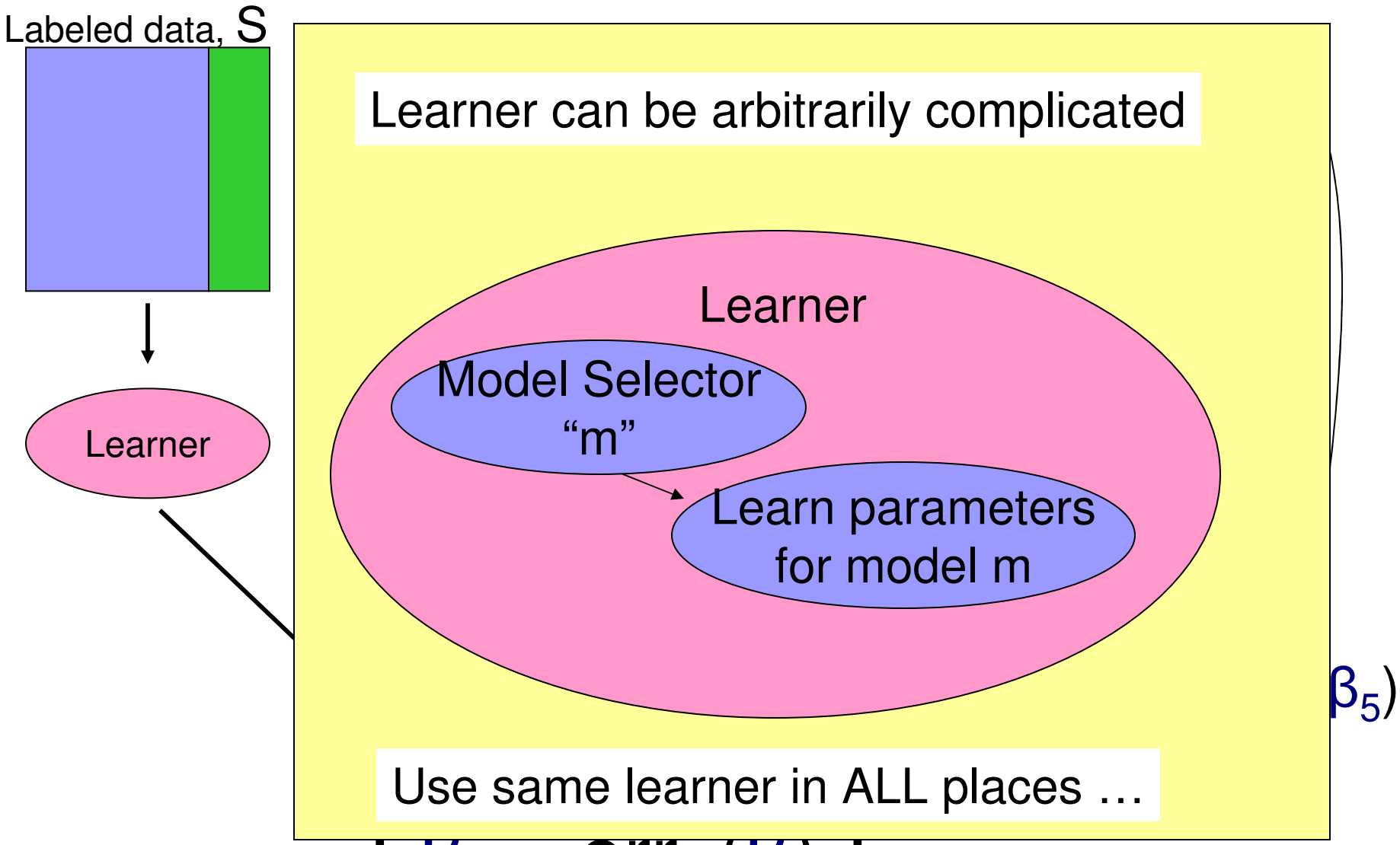
… if you are careful

$$\mathrm{err}_{S'}(\mathbf{w}_S) = \frac{1}{|S'|} \sum_{(\mathbf{x},t)\in S'} \left(t - \sum_i w_i h_i(\mathbf{x}) \right)^2$$

Finding Best Parameters

- Want to learn what "parameters" work best?
  - Best model (RBF vs linear? degree of polynomial)? Feature selection? Trade-off parameter?…
  - $\text{argmin}_v \{ \ err_D(\ L(S, v)) \ \}$
- #1?: Try each value on entire dataset. Report which has smallest TRAINING SET error?
  - $\text{argmin}_v \{ \ err_S(\ L(S, v)) \ \}$
- #2?: For each value, run 5-fold C-V (wrt entire dataset)
  - $v^* = \text{argmax}_v \{ E[err_{Si} (L(S_{-I}, v)])$
- Run cross-validation on "best-value" algorithm

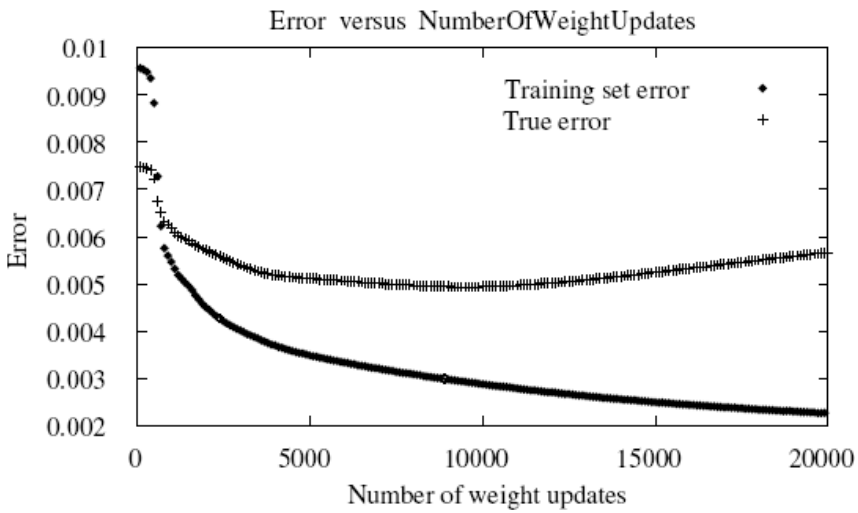
Return: [Predictor + Est Quality]



β_5)

[β, $\approx err_D(\beta)$]

Fit-to-Data ≠ Generalization



- h_k = hyp after k updates
  $\widehat{err}_S(h_{20000}) < \widehat{err}_S(h_{10000})$ but
  $err_{\mathcal{D},f}(h_{20000}) > err_{\mathcal{D},f}(h_{10000})$!

- "Overfitting"
  Best "fit-to-data" can find "meaningless regularity" in data
  (coincidences in the noise)
  $\Rightarrow$ bad generalization behavior
- Gen'l: Hypothesis $h \in H$ overfits training data if
  $\exists$ alternative hypothesis $h' \in H$ s.t.

$$\widehat{err}_S(h) < \widehat{err}_S(h')$$

but

$$err_{\mathcal{D},f}(h) > err_{\mathcal{D},f}(h')$$

Summary of Estimating Error

- SetUp: Learner L,
  - using labeled training data S
  - produces predictor $h_S = L(S)$
- Want $err_D(h_S)$

  h's Generalization Error over distribution D
  - to evaluate predictor h_S
  - to decide among possible predictors
  - to evaluate learner
- But depends on $D(\mathbf{x},t)$: not known!

Estimating $err_D(h_S)$

1. Training Set Error
   - Use h_S's empirical error on S

     $\underline{err}_S(h_S)$

   $\Rightarrow$ Very Optimistic
2. Hold Out Error
   - Divide $S = S_1 \cup S_2$;
     Return $\underline{err}_{S_2}(h_{S_1})$

   $\Rightarrow$ Slightly Pessimistic
3. Cross Validation
   - $1/k \sum_i err_{S_i}(L(S_{-i})$

   $\Rightarrow$ Slightly Less Pessimistic

For evaluating GENERAL PREDICTORS
- classifiers, regressors
- … best values for parameters