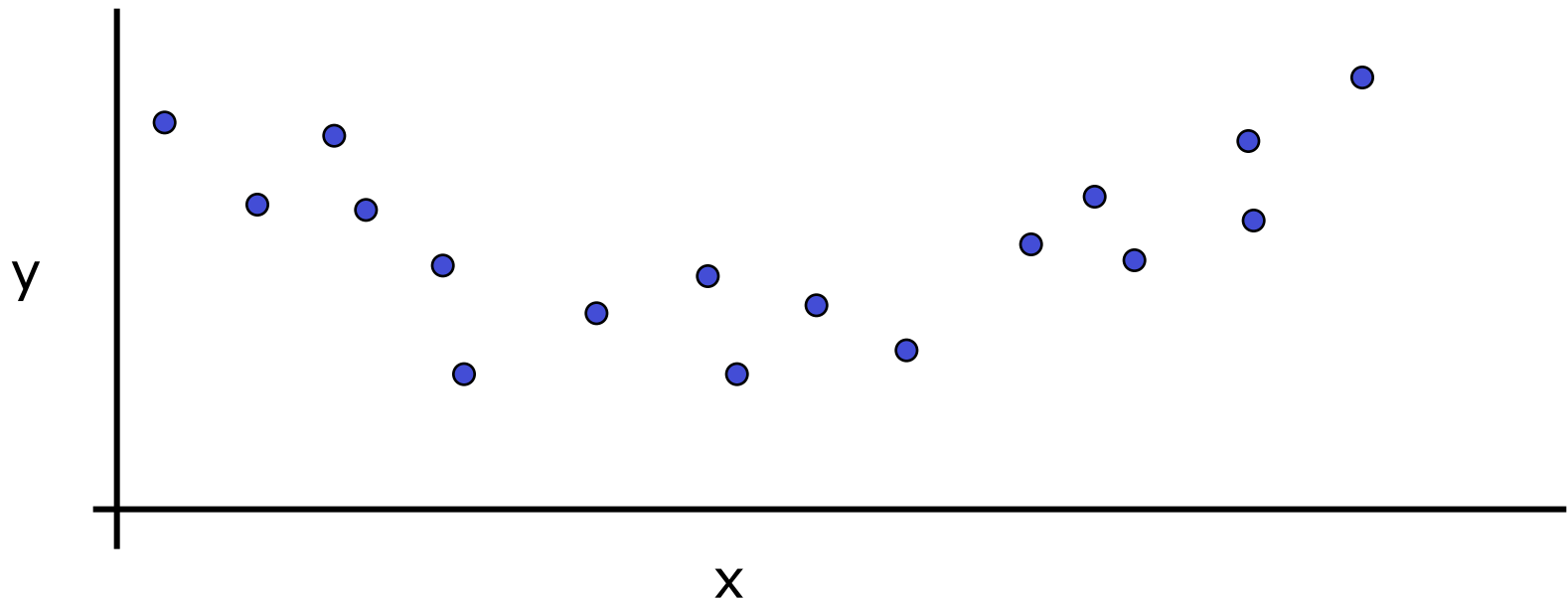


# Regression

- Output is a continuous numeric value
  - Locally-weighted averaging
  - Regression trees

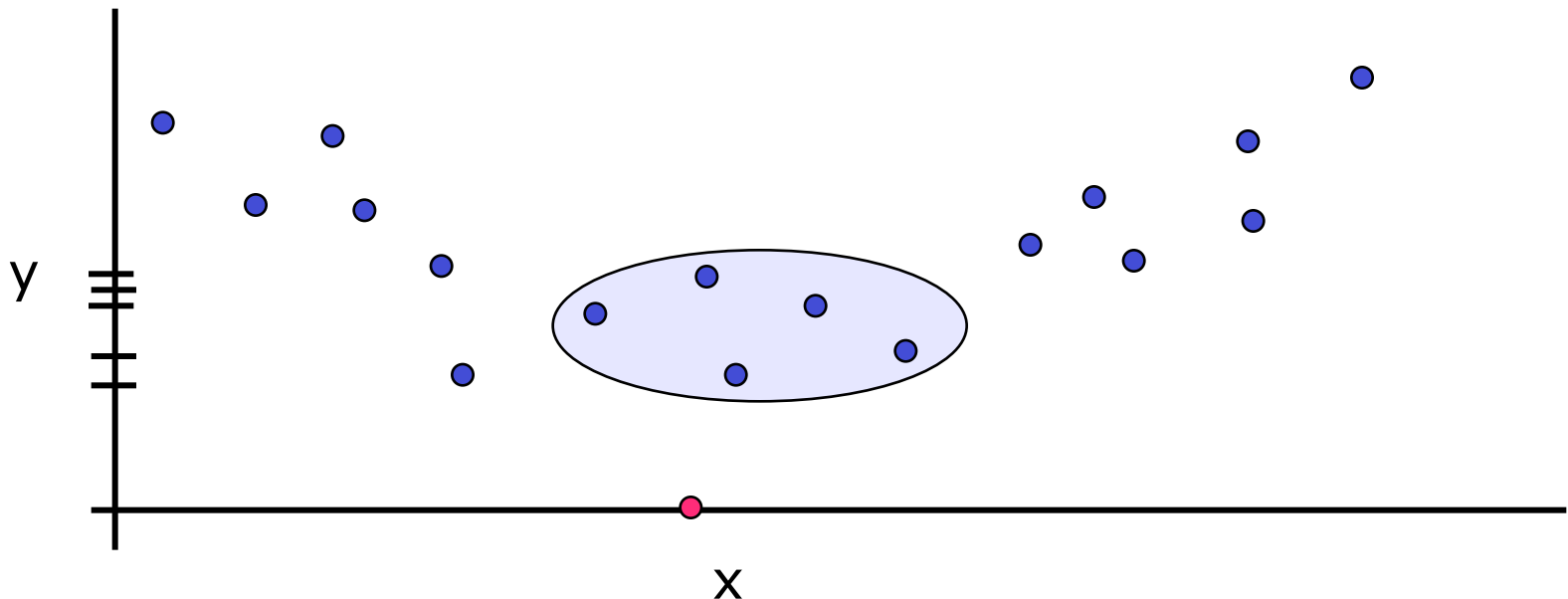
# Local Averaging

- Remember all your data



# Local Averaging

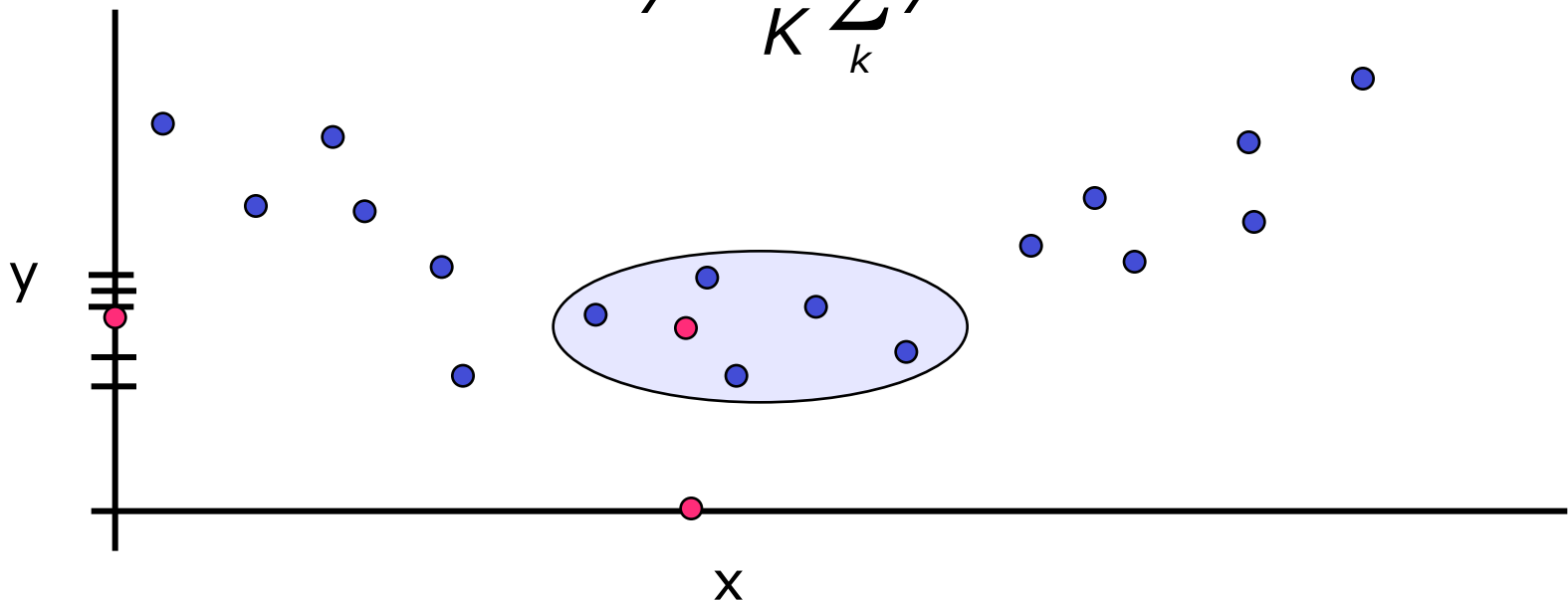
- Remember all your data
- When someone asks a question,
  - find the  $K$  nearest old data points



# Local Averaging

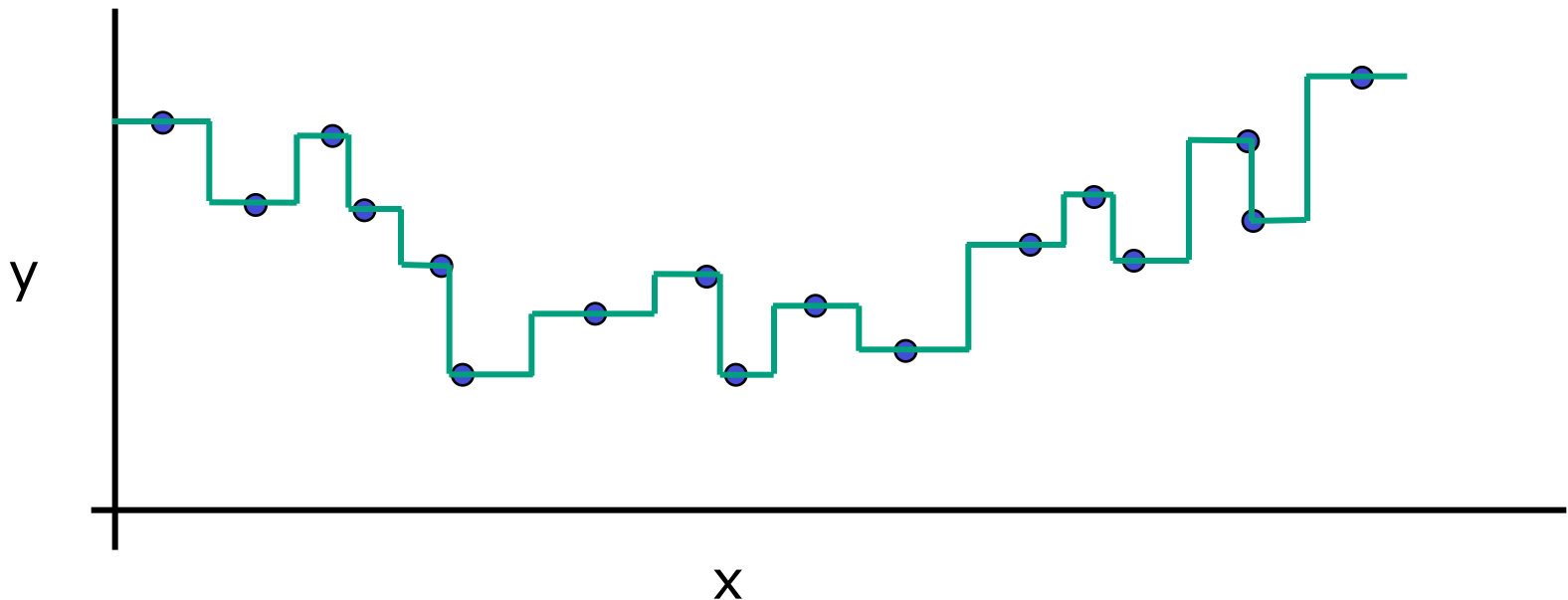
- Remember all your data
- When someone asks a question,
  - find the  $K$  nearest old data points
  - return the average of the answers associated with them

$$y = \frac{1}{K} \sum_k y^k$$



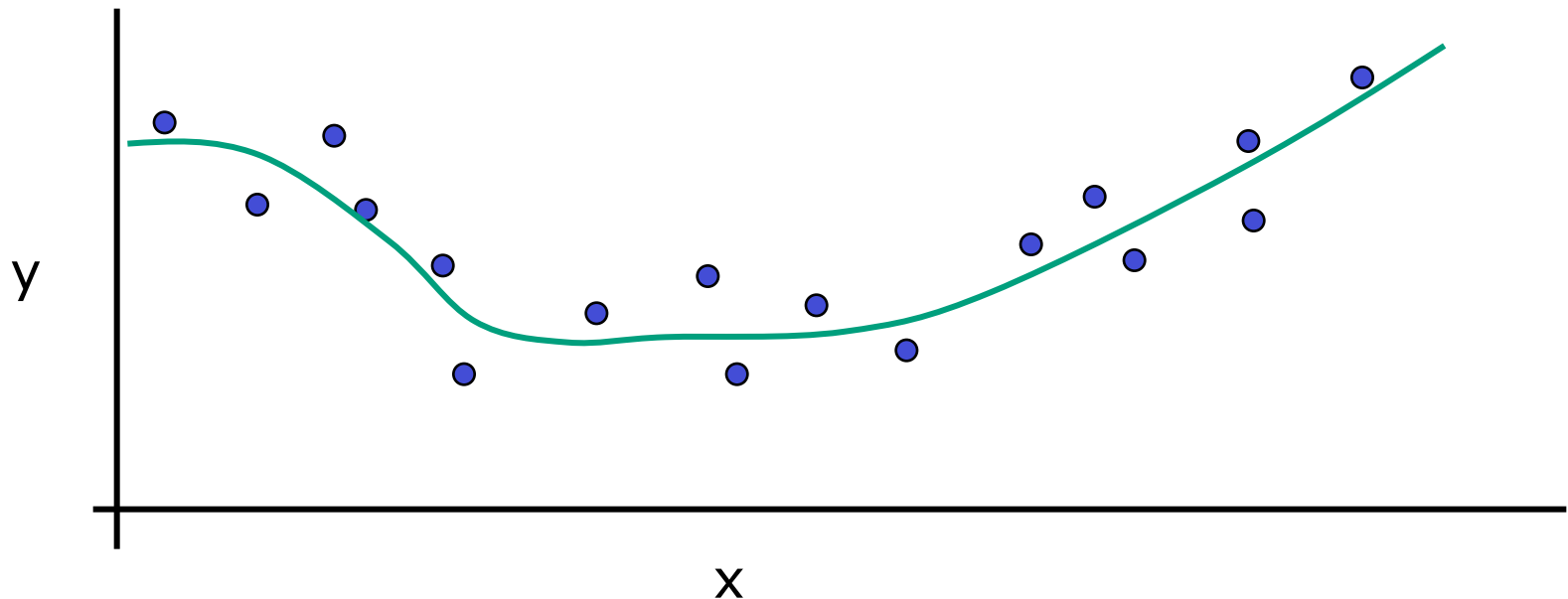
# $K = 1$

- Tracks data very closely
- Prone to overfitting



# Bigger K

- Smoothes out variations in data
- May introduce too much bias



# Locally Weighted Averaging

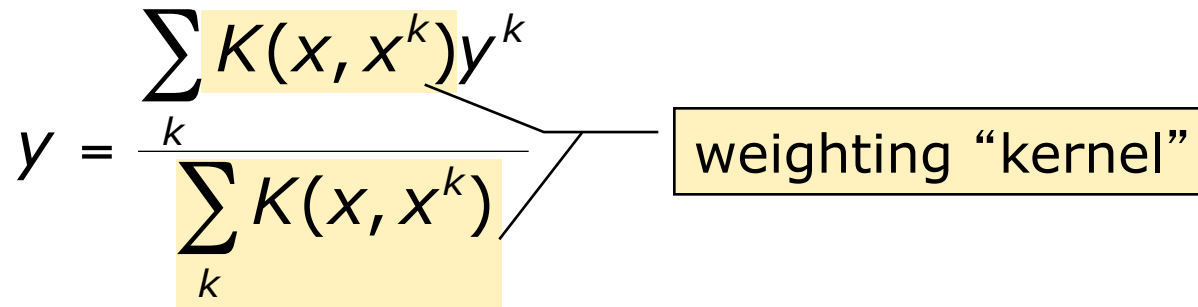
# Locally Weighted Averaging

- Find all points within distance  $\lambda$  from target point
- Average the outputs, weighted according to how far away they are from the target point



# Locally Weighted Averaging

- Find all points within distance  $\lambda$  from target point
- Average the outputs, weighted according to how far away they are from the target point
- Given a target  $x$ , with  $k$  ranging over neighbors,

$$y = \frac{\sum_k K(x, x^k) y^k}{\sum_k K(x, x^k)}$$


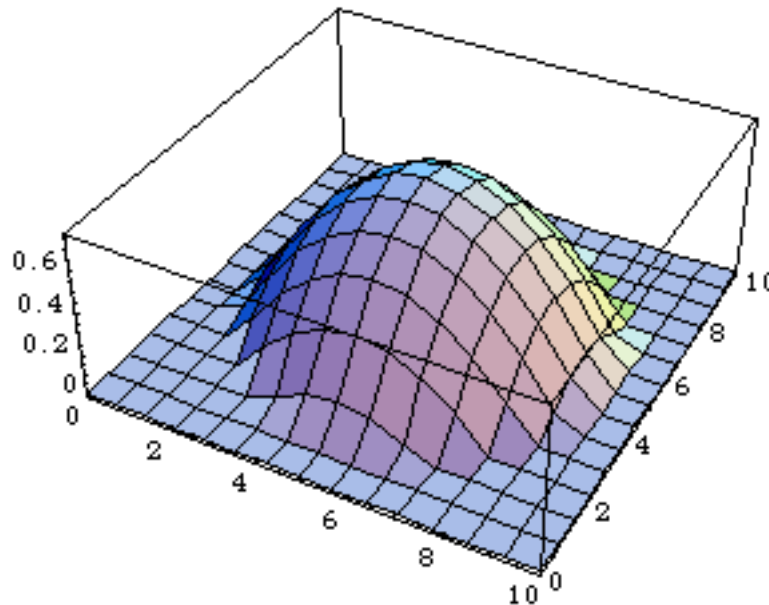
A diagram illustrating the weighting kernel. The equation  $y = \frac{\sum_k K(x, x^k) y^k}{\sum_k K(x, x^k)}$  is shown. The terms  $K(x, x^k)$  in both the numerator and denominator are highlighted with yellow boxes. A line connects these two boxes to a separate yellow box on the right containing the text "weighting 'kernel'".

# Epanechnikov Kernel

- D is Euclidean distance

$$K(x, x^k) = \max\left(\frac{3}{4}\left(1 - \frac{D(x, x^k)^2}{\lambda^2}\right), 0\right)$$

- $x = \langle 5, 5 \rangle$
- $\lambda = 4$



- Many other possible choices of kernel K

# Smooth

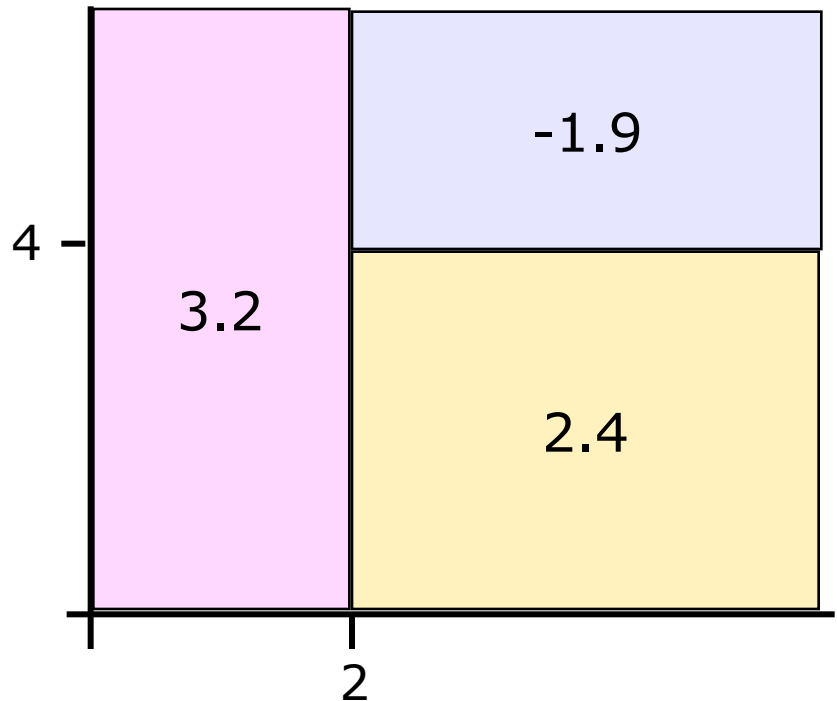
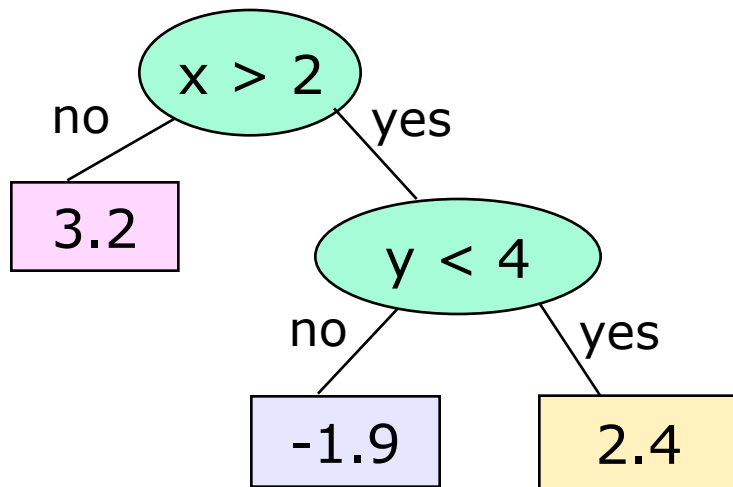
- How should we choose  $\lambda$ ?
  - If small, then we aren't averaging many points
    - Worse at averaging out noise
    - Better at modeling discontinuities
  - If big, we are averaging a lot of points
    - Good at averaging out noise
    - Smears out discontinuities
- Can use cross-validation to choose  $\lambda$
- May be better to let it vary according to local density of points

# Regression Trees

- Like decision trees, but with real-valued constant outputs at the leaves

# Regression Trees

- Like decision trees, but with real-valued constant outputs at the leaves



# Leaf Values

- Assign a leaf node the average of the  $y$  values of the data points that fall there.

# Leaf Values

- Assign a leaf node the average of the  $y$  values of the data points that fall there.
- We'd like to have groups of points in a leaf that have similar  $y$  values (because then the average is a good representative)

# Variance

- Measure of how much a set of numbers is spread out



# Variance

- Measure of how much a set of numbers is spread out
- Mean of  $m$  values,  $z_1$  through  $z_m$  :

$$\mu = \frac{1}{m} \sum_{k=1}^m z_k$$

# Variance

- Measure of how much a set of numbers is spread out
- Mean of  $m$  values,  $z_1$  through  $z_m$  :

$$\mu = \frac{1}{m} \sum_{k=1}^m z_k$$

- Variance: average squared difference between  $z$ 's and the mean:

$$\sigma^2 = \frac{1}{m-1} \sum_{k=1}^m (z_k - \mu)^2$$

# Let's Split

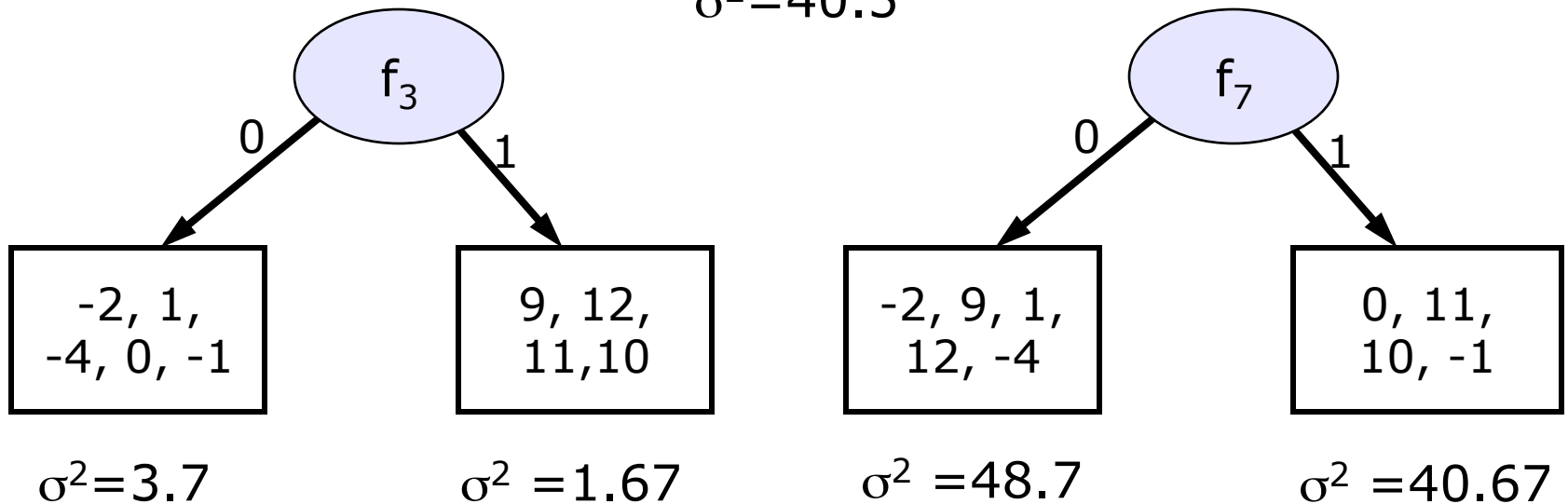
D: -2, 9, 1, 12, -4,  
0, 11, 10, -1

$$\sigma^2 = 40.5$$

# Let's Split

D: -2, 9, 1, 12, -4,  
0, 11, 10, -1

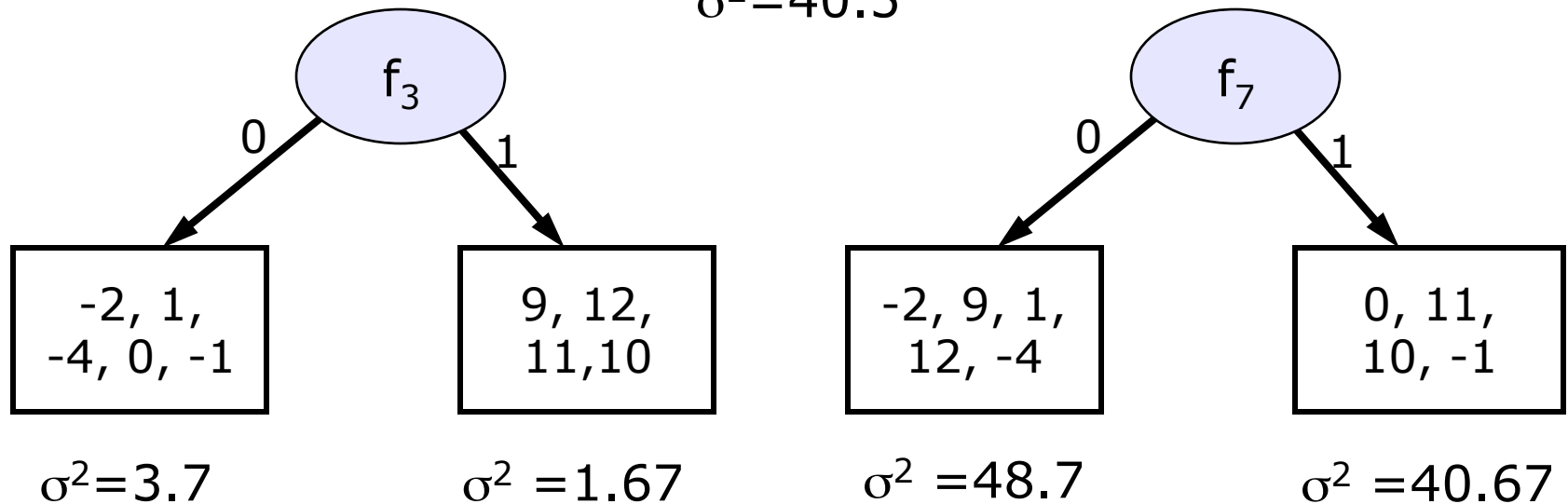
$$\sigma^2 = 40.5$$



# Let's Split

D: -2, 9, 1, 12, -4,  
0, 11, 10, -1

$$\sigma^2 = 40.5$$



$$AV(j) = p_j \sigma^2(D_j^+) + (1 - p_j) \sigma^2(D_j^-)$$

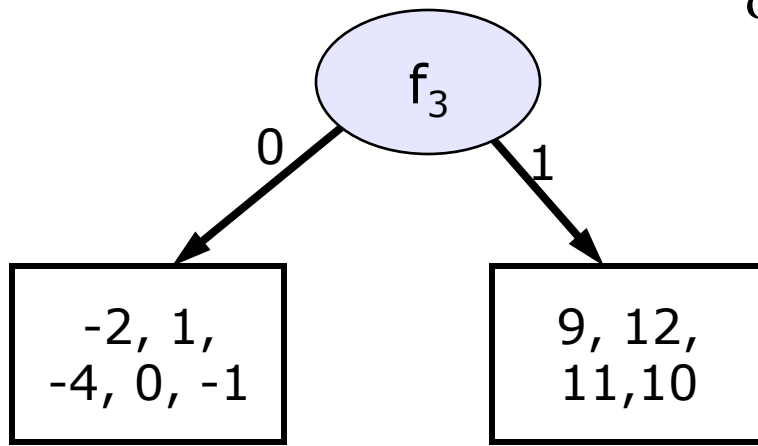
% of D with  $f_j = 1$

subset of D with  $f_j = 1$

# Let's Split

D: -2, 9, 1, 12, -4,  
0, 11, 10, -1

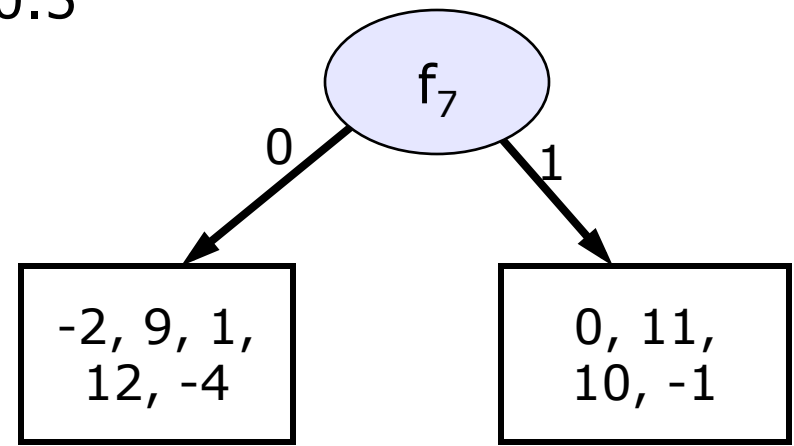
$$\sigma^2 = 40.5$$



$$\sigma^2 = 3.7$$

$$\sigma^2 = 1.67$$

$$\begin{aligned} AV &= (5/9) * 3.7 + (4/9) * 1.67 \\ &= 2.8 \end{aligned}$$



$$\sigma^2 = 48.7$$

$$\sigma^2 = 40.67$$

$$\begin{aligned} AV &= (5/9) * 48.7 + (4/9) * 40.67 \\ &= 45.13 \end{aligned}$$

# Stopping

- Stop when variance at a leaf is small enough

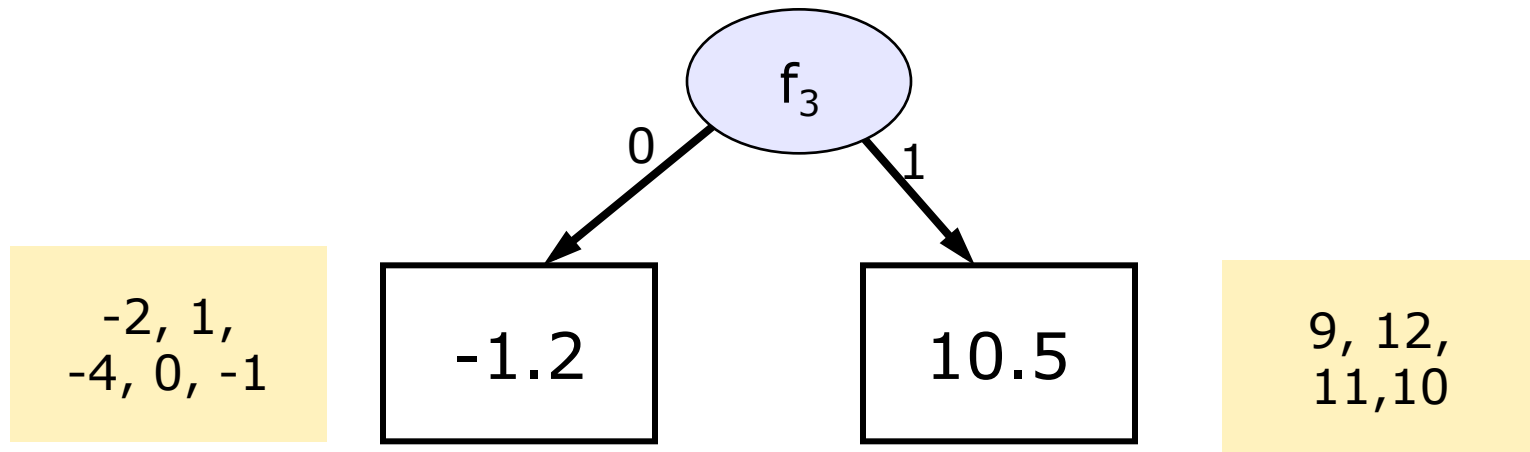
# Stopping

- Stop when variance at a leaf is small enough
- Or when you have fewer than min-leaf elements at a leaf



# Stopping

- Stop when variance at a leaf is small enough
- Or when you have fewer than min-leaf elements at a leaf
- Set  $y$  at a leaf to be the mean of the  $y$  values of the elements



# CS 2750 Machine Learning

## Lecture 6

### Linear regression

Milos Hauskrecht

[milos@cs.pitt.edu](mailto:milos@cs.pitt.edu)

5329 Sennott Square

# Outline

## Linear Regression

- Linear model
- Loss (error) function based on the least squares fit
- Parameter estimation.
- Gradient methods.
- On-line regression techniques.
- Linear additive models
- Statistical model of linear regression

# Supervised learning

**Data:**  $D = \{D_1, D_2, \dots, D_n\}$  a set of  $n$  examples

$$D_i = \langle \mathbf{x}_i, y_i \rangle$$

$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,d})$  is an input vector of size  $d$

$y_i$  is the desired output (given by a teacher)

**Objective:** learn the mapping  $f : X \rightarrow Y$

s.t.  $y_i \approx f(\mathbf{x}_i)$  for all  $i = 1, \dots, n$

- **Regression:**  $Y$  is **continuous**

Example: earnings, product orders  $\rightarrow$  company stock price

- **Classification:**  $Y$  is **discrete**

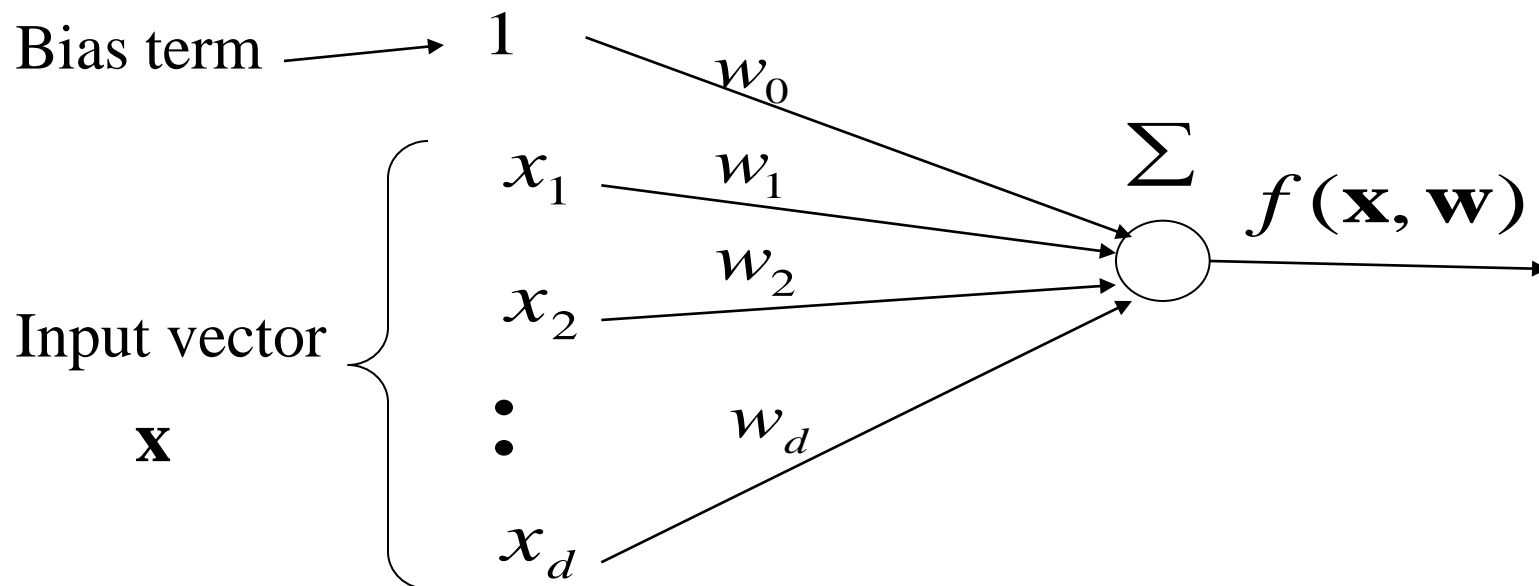
Example: handwritten digit in binary form  $\rightarrow$  digit label

# Linear regression

- **Function**  $f : X \rightarrow Y$  is a linear combination of input components

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \dots w_d x_d = w_0 + \sum_{j=1}^d w_j x_j$$

$w_0, w_1, \dots, w_k$  - **parameters (weights)**



# Linear regression

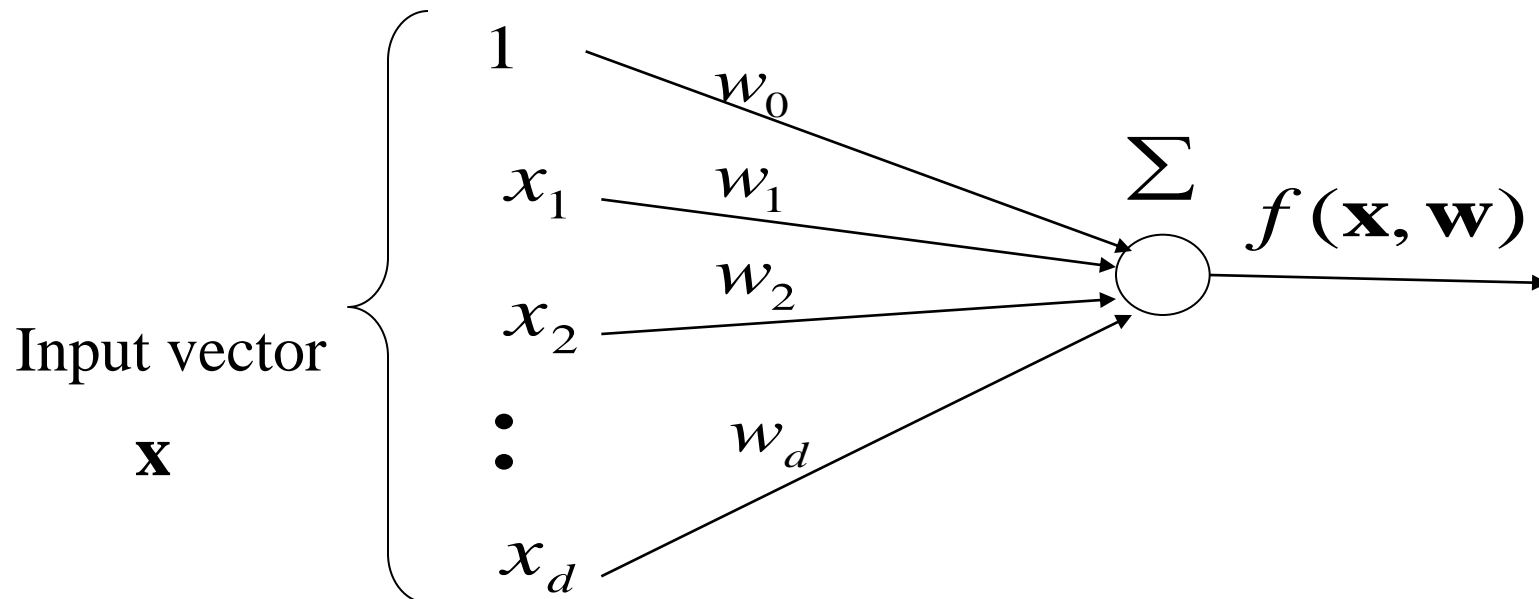
- **Shorter (vector) definition of the model**

- Include bias constant in the input vector

$$\mathbf{x} = (1, x_1, x_2, \dots, x_d)$$

$$f(\mathbf{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \dots w_d x_d = \mathbf{w}^T \mathbf{x}$$

$w_0, w_1, \dots, w_k$  - **parameters (weights)**



# Linear regression. Error.

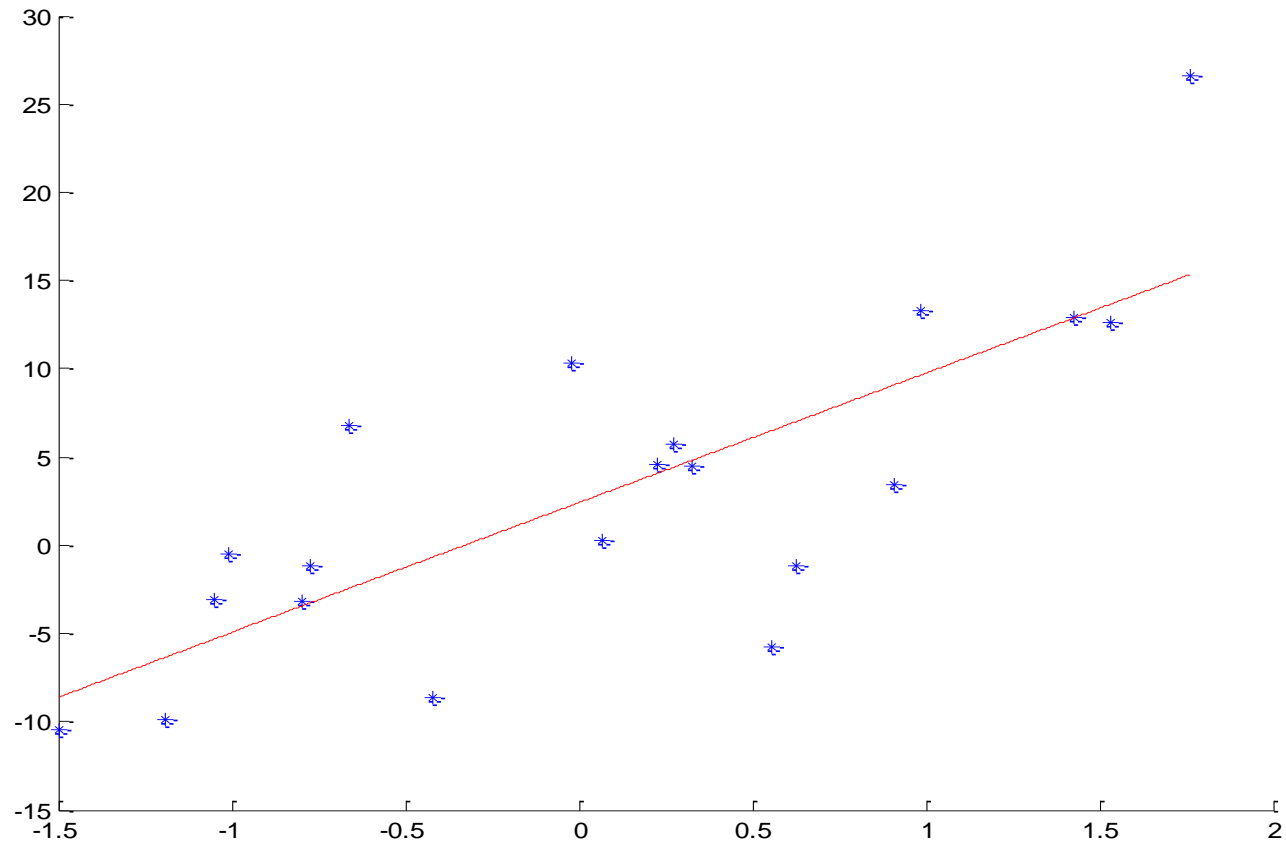
- **Data:**  $D_i = \langle \mathbf{x}_i, y_i \rangle$
- **Function:**  $\mathbf{x}_i \rightarrow f(\mathbf{x}_i)$
- We would like to have  $y_i \approx f(\mathbf{x}_i)$  for all  $i = 1, \dots, n$
- **Error function**
  - measures how much our predictions deviate from the desired answers

**Mean-squared error**  $J_n = \frac{1}{n} \sum_{i=1, \dots, n} (y_i - f(\mathbf{x}_i))^2$

- **Learning:**
  - We want to find the weights minimizing the error !**

# Linear regression. Example

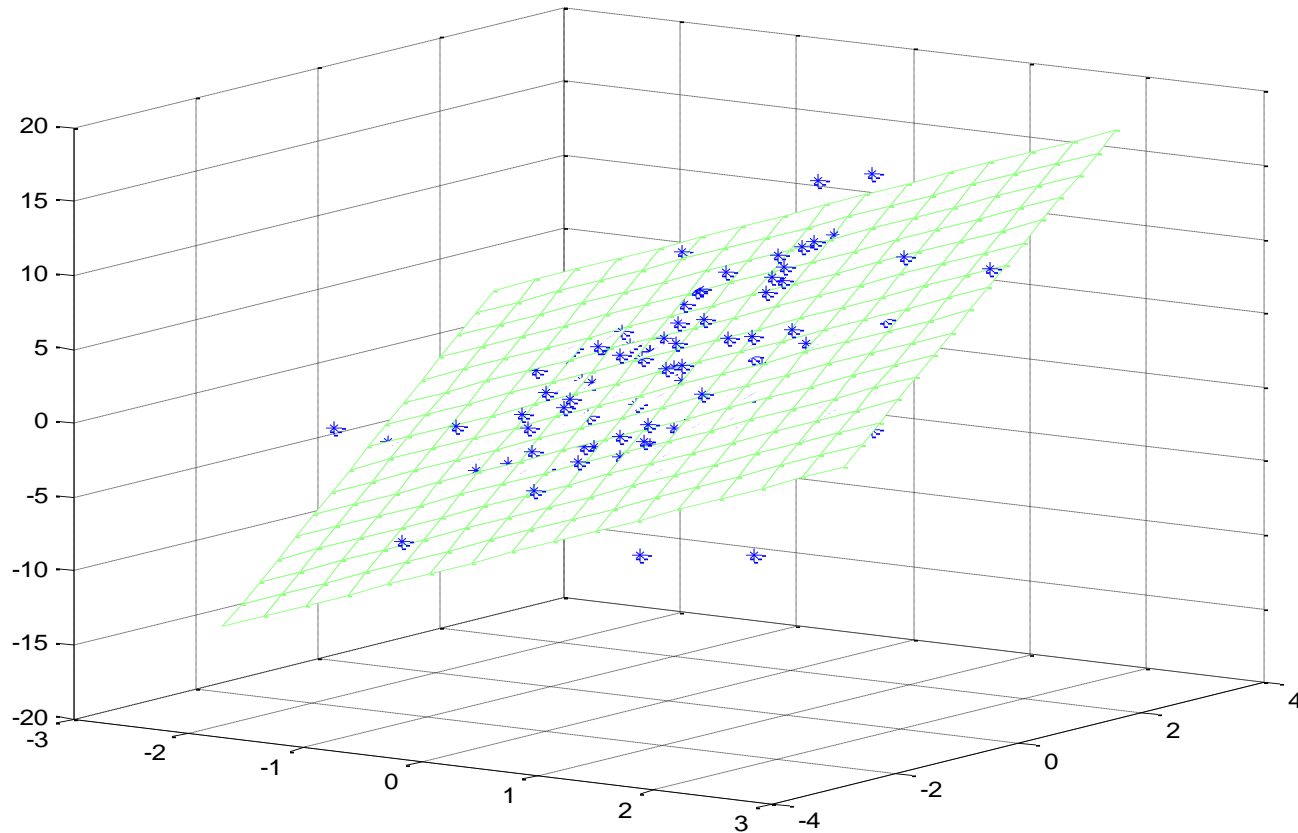
- 1 dimensional input  $\mathbf{x} = (x_1)$





# Linear regression. Example.

- 2 dimensional input  $\mathbf{x} = (x_1, x_2)$



# Linear regression. Optimization.

- We want the **weights minimizing the error**

$$J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2 = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^n (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \dots - w_d x_{i,d}) x_{i,j} = 0$$

- **Vector of derivatives:**

$$\text{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = \bar{\mathbf{0}}$$

# Linear regression. Optimization.

- $\text{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \bar{\mathbf{0}}$  defines a set of equations in  $\mathbf{w}$

$$\frac{\partial}{\partial w_0} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^n (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \dots - w_d x_{i,d}) = 0$$

$$\frac{\partial}{\partial w_1} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^n (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \dots - w_d x_{i,d}) x_{i,1} = 0$$

...

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^n (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \dots - w_d x_{i,d}) x_{i,j} = 0$$

...

$$\frac{\partial}{\partial w_d} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^n (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \dots - w_d x_{i,d}) x_{i,d} = 0$$

# Solving linear regression

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^n (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \dots - w_d x_{i,d}) x_{i,j} = 0$$

By rearranging the terms we get a **system of linear equations** with  $d+1$  unknowns

$$\mathbf{Aw} = \mathbf{b}$$

$$w_0 \sum_{i=1}^n x_{i,0} 1 + w_1 \sum_{i=1}^n x_{i,1} 1 + \dots + w_j \sum_{i=1}^n x_{i,j} 1 + \dots + w_d \sum_{i=1}^n x_{i,d} 1 = \sum_{i=1}^n y_i 1$$

$$w_0 \sum_{i=1}^n x_{i,0} x_{i,1} + w_1 \sum_{i=1}^n x_{i,1} x_{i,1} + \dots + w_j \sum_{i=1}^n x_{i,j} x_{i,1} + \dots + w_d \sum_{i=1}^n x_{i,d} x_{i,1} = \sum_{i=1}^n y_i x_{i,1}$$

...

$$w_0 \sum_{i=1}^n x_{i,0} x_{i,j} + w_1 \sum_{i=1}^n x_{i,1} x_{i,j} + \dots + w_j \sum_{i=1}^n x_{i,j} x_{i,j} + \dots + w_d \sum_{i=1}^n x_{i,d} x_{i,j} = \sum_{i=1}^n y_i x_{i,j}$$

...

# Solving linear regression

- The optimal set of weights satisfies:

$$\nabla_{\mathbf{w}} (J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = \bar{\mathbf{0}}$$

Leads to a **system of linear equations (SLE)** with  $d+1$  unknowns of the form

$$\mathbf{A}\mathbf{w} = \mathbf{b}$$

$$w_0 \sum_{i=1}^n x_{i,0} x_{i,j} + w_1 \sum_{i=1}^n x_{i,1} x_{i,j} + \dots + w_j \sum_{i=1}^n x_{i,j} x_{i,j} + \dots + w_d \sum_{i=1}^n x_{i,d} x_{i,j} = \sum_{i=1}^n y_i x_{i,j}$$

**Solution to SLE: ?**

# Solving linear regression

- The optimal set of weights satisfies:

$$\nabla_{\mathbf{w}} (J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = \bar{\mathbf{0}}$$

Leads to a **system of linear equations (SLE)** with  $d+1$  unknowns of the form

$$\mathbf{A}\mathbf{w} = \mathbf{b}$$

$$w_0 \sum_{i=1}^n x_{i,0} x_{i,j} + w_1 \sum_{i=1}^n x_{i,1} x_{i,j} + \dots + w_j \sum_{i=1}^n x_{i,j} x_{i,j} + \dots + w_d \sum_{i=1}^n x_{i,d} x_{i,j} = \sum_{i=1}^n y_i x_{i,j}$$

**Solution to SLE:**

$$\mathbf{w} = \mathbf{A}^{-1} \mathbf{b}$$

- matrix inversion

# Gradient descent solution

**Goal:** the weight optimization in the linear regression model

$$J_n = Error(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

An alternative to SLE solution:

- **Gradient descent**

**Idea:**

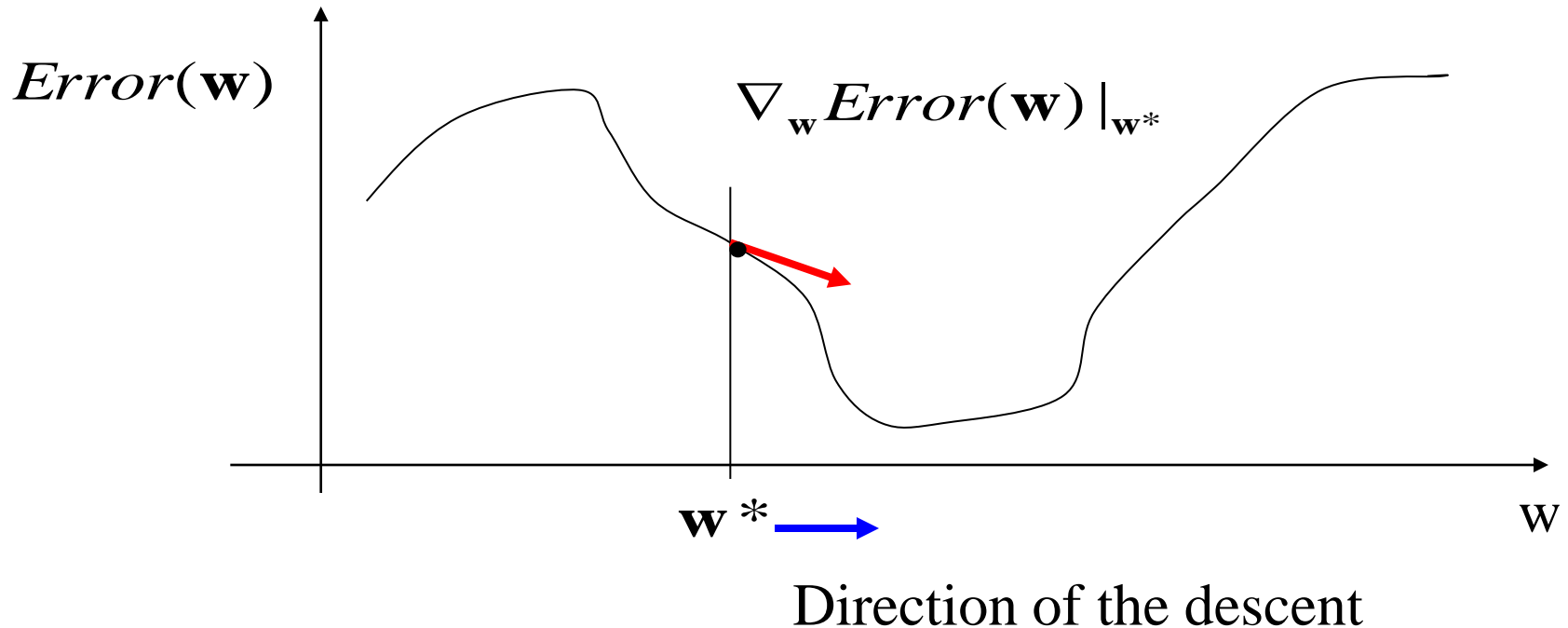
- Adjust weights in the direction that improves the Error
- The gradient tells us what is the right direction

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

$\alpha > 0$     -    a **learning rate** (scales the gradient changes)

# Gradient descent method

- Descend using the gradient information

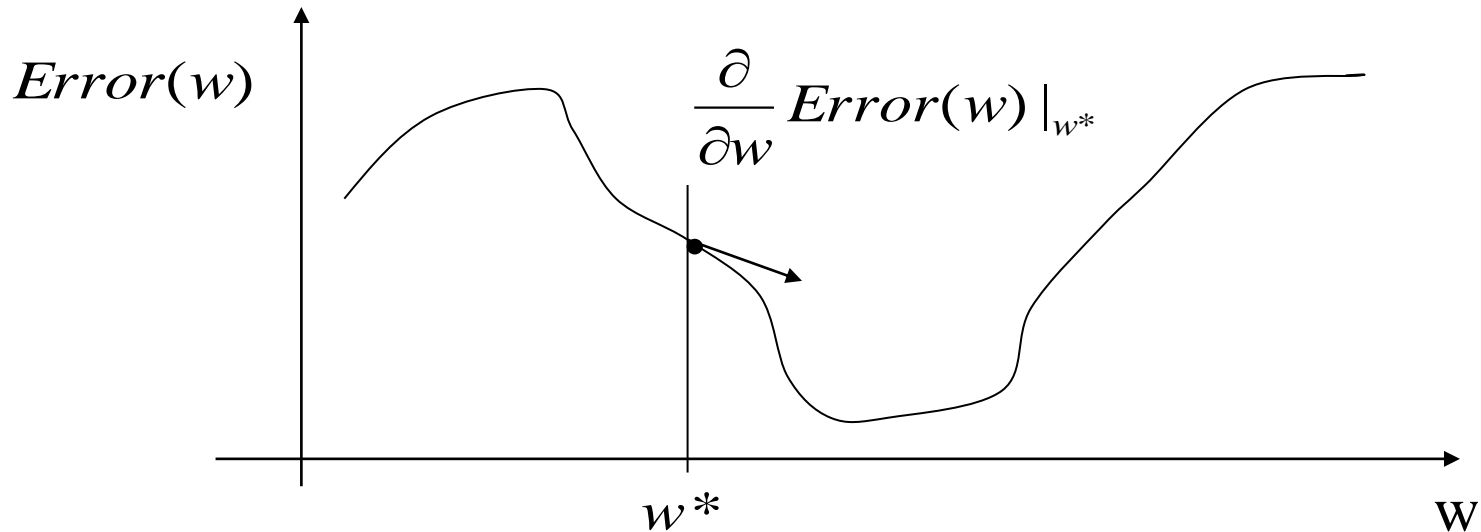


- Change the value of  $\mathbf{w}$  according to the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$



# Gradient descent method



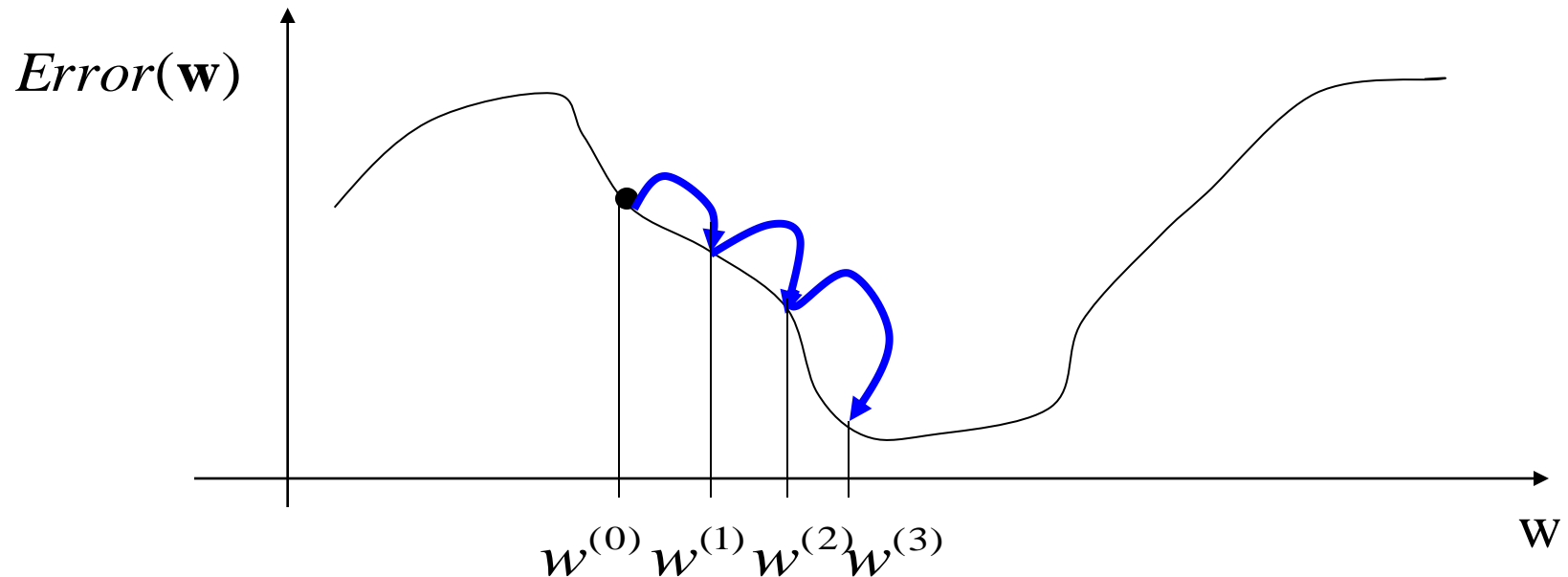
- New value of the parameter

$$w_j \leftarrow w_j^* - \alpha \frac{\partial}{\partial w_j} Error(w) |_{w^*} \quad \text{For all } j$$

$\alpha > 0$  - a learning rate (scales the gradient changes)

# Gradient descent method

- Iteratively approaches the optimum of the Error function



# Online gradient algorithm

- The error function is defined for the whole dataset  $D$

$$J_n = Error(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- **error for a sample**  $D_i = \langle \mathbf{x}_i, y_i \rangle$

$$J_{\text{online}} = Error_i(\mathbf{w}) = \frac{1}{2} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- **Online gradient method: changes weights after every sample**

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} Error_i(\mathbf{w})$$

- **vector form:**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

$\alpha > 0$  - Learning rate that depends on the number of updates

# Online gradient method

Linear model

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

On-line error  $J_{online} = Error_i(\mathbf{w}) = \frac{1}{2} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$

**On-line algorithm:** generates a sequence of online updates

**(i)-th update step with :**  $D_i = \langle \mathbf{x}_i, y_i \rangle$

**j-th weight:**

$$w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i) \frac{\partial Error_i(\mathbf{w})}{\partial w_j} \Big|_{\mathbf{w}^{(i-1)}}$$

$$w_j^{(i)} \leftarrow w_j^{(i-1)} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)}))x_{i,j}$$

**Fixed learning rate:**  $\alpha(i) = C$       **Annealed learning rate:**  $\alpha(i) \approx \frac{1}{i}$

- Use a small constant
- Gradually rescales changes

# Online regression algorithm

**Online-linear-regression** ( $D$ , *number of iterations*)

**Initialize** weights  $\mathbf{w} = (w_0, w_1, w_2 \dots w_d)$

**for**  $i=1:1: \text{number of iterations}$

**do**      **select** a data point  $D_i = (\mathbf{x}_i, y_i)$  from  $D$

**set** learning rate  $\alpha(i)$

**update** weight vector

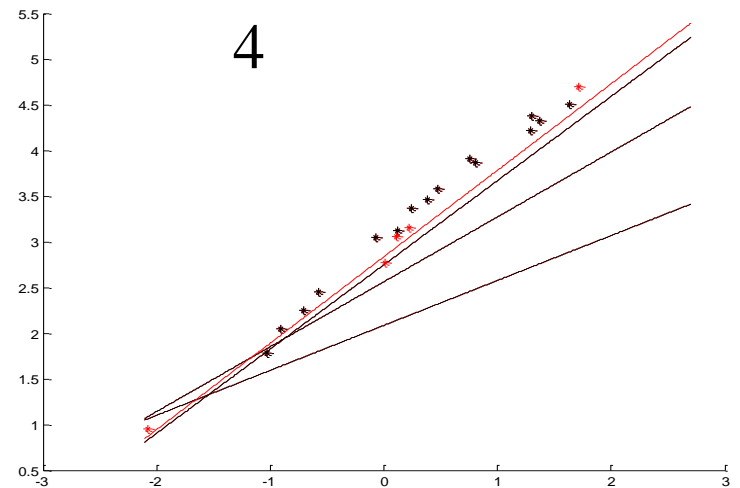
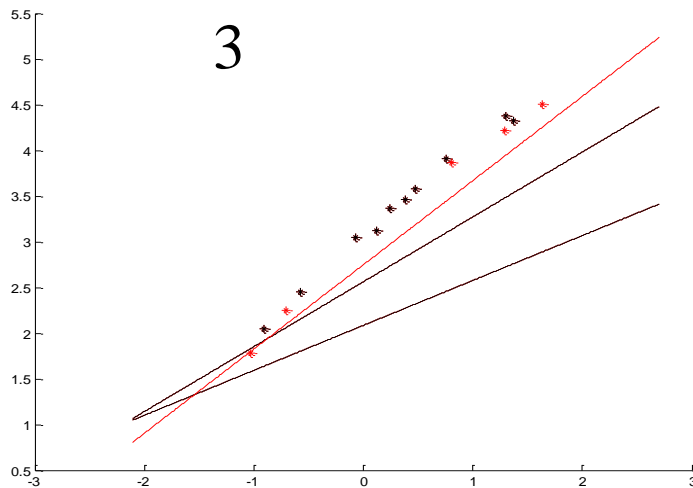
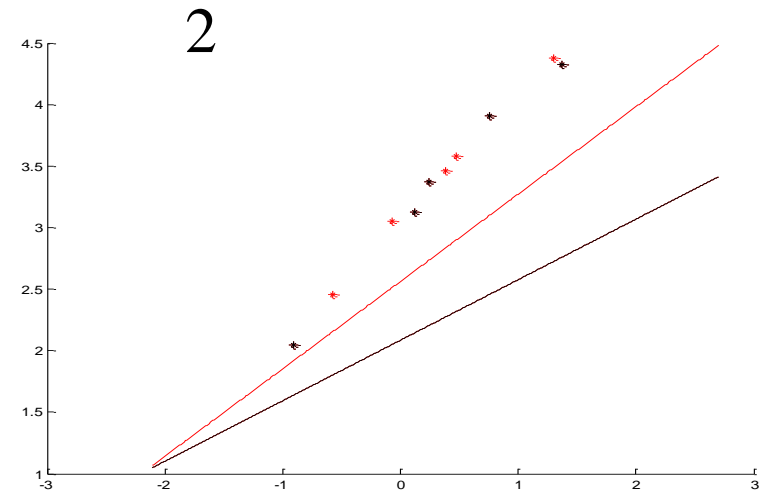
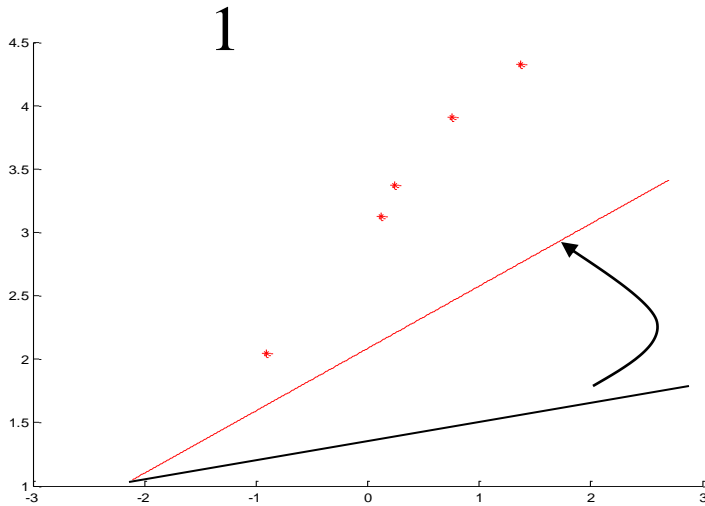
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}))\mathbf{x}_i$$

**end for**

**return** weights  $\mathbf{w}$

- **Advantages:** very easy to implement, continuous data streams

# On-line learning. Example



# Practical concerns: Input normalization

- **Input normalization**

- makes the data vary roughly on the same scale.
- Can make a huge difference in **on-line learning**

**Assume on-line update (delta) rule for two weights  $j, k$ :**

$$\begin{aligned} w_j &\leftarrow w_j + \alpha(i)(y_i - f(\mathbf{x}_i)) x_{i,j} \\ &= \\ w_k &\leftarrow w_k + \alpha(i)(y_i - f(\mathbf{x}_i)) x_{i,k} \end{aligned}$$

Change depends on the magnitude of the input

For inputs with a large magnitude the change in the weight is huge: changes to the inputs with high magnitude disproportional as if the input was more important

# Input normalization

- **Input normalization:**

- Solution to the problem of different scales
- Makes all inputs vary in the same range around 0

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{i,j} \qquad \sigma_j^2 = \frac{1}{n-1} \sum_{i=1}^n (x_{i,j} - \bar{x}_j)^2$$

**New input:**  $\tilde{x}_{i,j} = \frac{(x_{i,j} - \bar{x}_j)}{\sigma_j}$

More complex normalization approach can be applied when we want to process data with correlations

**Similarly we can renormalize outputs  $y$**



# Regularized linear regression

- If the number of parameters is large relative to the number of data points used to train the model, we face the threat of overfit (generalization error of the model goes up)
  - The prediction accuracy can be often improved by setting some coefficients to zero
    - Increases the bias, reduces the variance of estimates
  - **Solutions:**
    - **Subset selection**
    - **Ridge regression**
    - **Lasso regression**
    - **Principal component regression**
  - Next: **ridge regression**
-

# Ridge regression

- Error function for the standard least squares estimates:

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- **We seek:**  $\mathbf{w}^* = \arg \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$

- **Ridge regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|^2$$

- Where  $\|\mathbf{w}\|^2 = \sum_{i=0}^d w_i^2$  and  $\lambda \geq 0$

- What does the new error function do?
-

# Ridge regression

- **Standard regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- **Ridge regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_{L2}^2$$

- $\|\mathbf{w}\|_{L2}^2 = \sum_{i=0}^d w_i^2$  penalizes non-zero weights with the cost proportional to  $\lambda$  (a **shrinkage coefficient**)
  - If an input attribute  $x_j$  has a small effect on improving the error function it is “shut down” by the penalty term
  - Inclusion of a shrinkage penalty is often referred to as **regularization**
-

# Regularized linear regression

How to solve the least squares problem if the error function is enriched by the regularization term  $\lambda \|\mathbf{w}\|^2$  ?

**Answer:** The solution to the optimal set of weights  $\mathbf{w}$  is obtained again by solving a set of linear equation.

**Standard linear regression:**

$$\nabla_{\mathbf{w}} (J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = \bar{\mathbf{0}}$$

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

where  $\mathbf{X}$  is an  $n \times d$  matrix with rows corresponding to examples and columns to inputs

**Regularized linear regression:**

$$\mathbf{w}^* = (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

---

# Lasso regression

- **Standard regression:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- **Lasso regression/regularization:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_{L1}$$

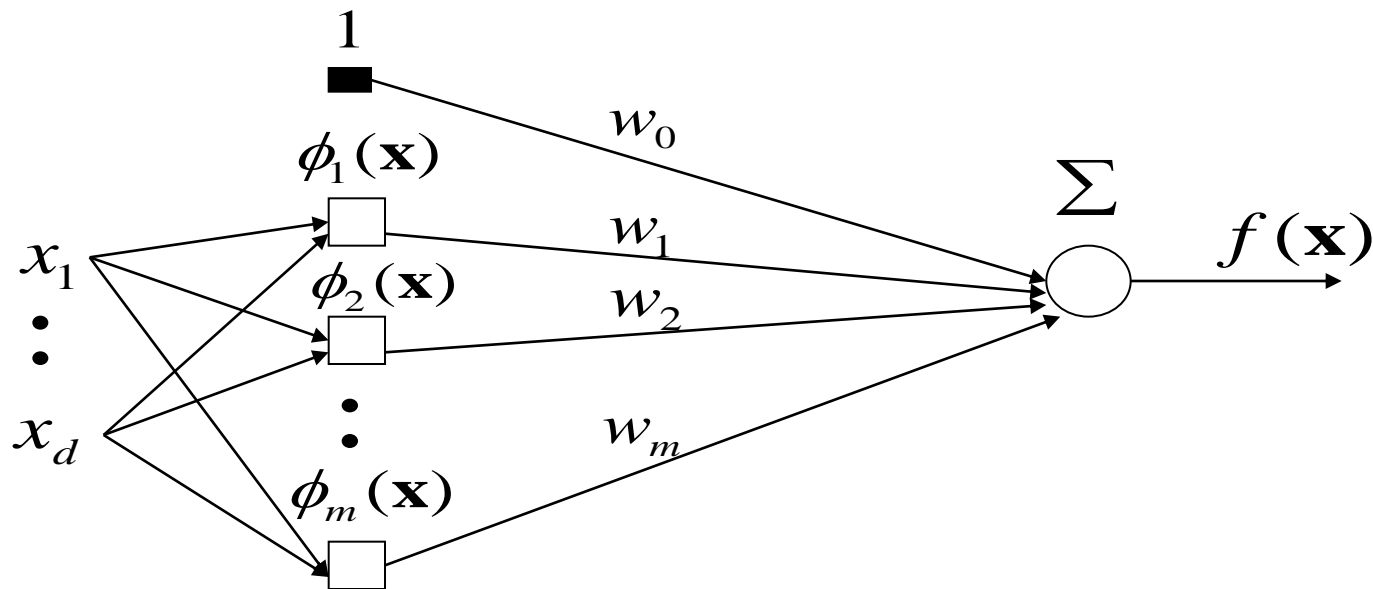
- $\|\mathbf{w}\|_{L1} = \sum_{i=0}^d |w_i|$  penalizes non-zero weights with the cost proportional to  $\lambda$ .
  - L1 is more aggressive pushing the weights to 0 compared to L2.
-

# Extensions of simple linear model

Replace inputs to linear units with **feature (basis) functions** to model **nonlinearities**

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^m w_j \phi_j(\mathbf{x})$$

$\phi_j(\mathbf{x})$  - an arbitrary function of  $\mathbf{x}$



**The same techniques as before to learn the weights**

# Additive linear models

- **Models linear in the parameters we want to fit**

$$f(\mathbf{x}) = w_0 + \sum_{k=1}^m w_k \phi_k(\mathbf{x})$$

$w_0, w_1 \dots w_m$  - parameters

$\phi_1(\mathbf{x}), \phi_2(\mathbf{x}) \dots \phi_m(\mathbf{x})$  - **feature or basis functions**

- **Basis functions examples:**

- a higher order polynomial, one-dimensional input  $\mathbf{x} = (x_1)$

$$\phi_1(x) = x \quad \phi_2(x) = x^2 \quad \phi_3(x) = x^3$$

- Multidimensional quadratic  $\mathbf{x} = (x_1, x_2)$

$$\phi_1(\mathbf{x}) = x_1 \quad \phi_2(\mathbf{x}) = x_1^2 \quad \phi_3(\mathbf{x}) = x_2 \quad \phi_4(\mathbf{x}) = x_2^2 \quad \phi_5(\mathbf{x}) = x_1 x_2$$

- Other types of basis functions

$$\phi_1(x) = \sin x \quad \phi_2(x) = \cos x$$

# Fitting additive linear models

- **Error function**  $J_n(\mathbf{w}) = 1/n \sum_{i=1, \dots, n} (y - f(\mathbf{x}_i))^2$

Assume:  $\boldsymbol{\phi}(\mathbf{x}_i) = (1, \phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \dots, \phi_m(\mathbf{x}_i))$

$$\nabla_{\mathbf{w}} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1, \dots, n} (y_i - f(\mathbf{x}_i)) \boldsymbol{\phi}(\mathbf{x}_i) = \bar{\mathbf{0}}$$

- Leads to a **system of  $m$  linear equations**

$$w_0 \sum_{i=1}^n 1 \phi_j(\mathbf{x}_i) + \dots + w_j \sum_{i=1}^n \phi_j(\mathbf{x}_i) \phi_j(\mathbf{x}_i) + \dots + w_m \sum_{i=1}^n \phi_m(\mathbf{x}_i) \phi_j(\mathbf{x}_i) = \sum_{i=1}^n y_i \phi_j(\mathbf{x}_i)$$

- Can be solved exactly like the linear case



# Example. Regression with polynomials.

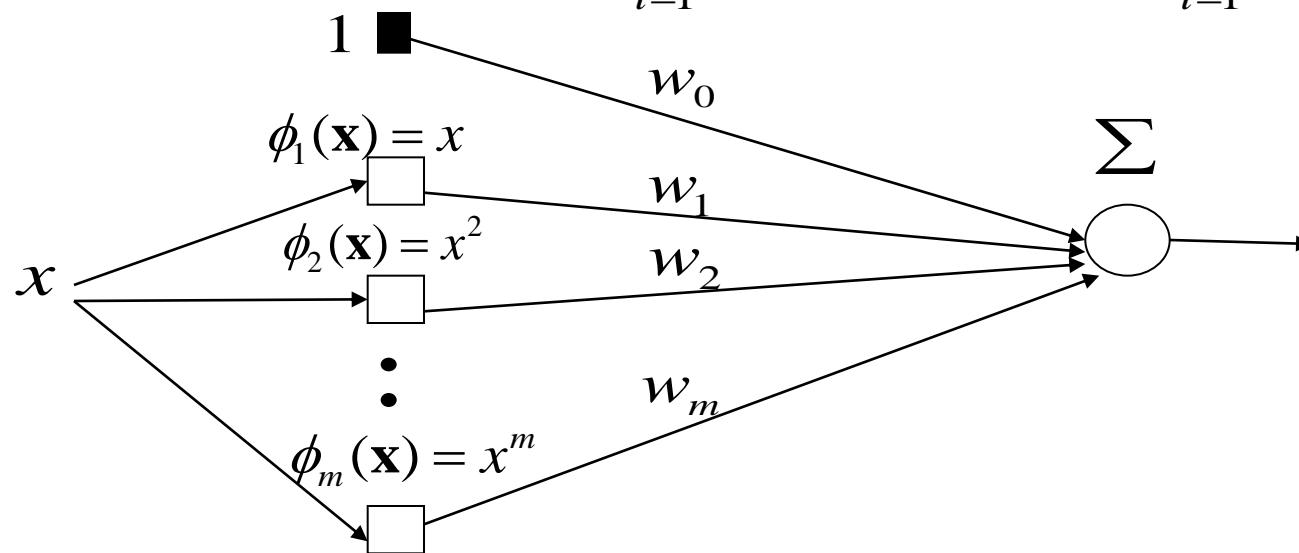
Regression with polynomials of degree  $m$

- **Data points:** pairs of  $\langle x, y \rangle$
- **Feature functions:**  $m$  feature functions

$$\phi_i(x) = x^i \quad i = 1, 2, \dots, m$$

- **Function to learn:**

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^m w_i \phi_i(x) = w_0 + \sum_{i=1}^m w_i x^i$$



# Learning with feature functions

**Function to learn:**

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^k w_i \phi_i(x)$$

**On line gradient update** for the  $\langle x, y \rangle$  pair

$$w_0 = w_0 + \alpha(y - f(\mathbf{x}, \mathbf{w}))$$

•  
•

$$w_j = w_j + \alpha(y - f(\mathbf{x}, \mathbf{w}))\phi_j(\mathbf{x})$$

Gradient updates are of the same form as in the linear and logistic regression models

# Example. Regression with polynomials.

**Example:** Regression with polynomials of degree  $m$

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^m w_i \phi_i(x) = w_0 + \sum_{i=1}^m w_i x^i$$

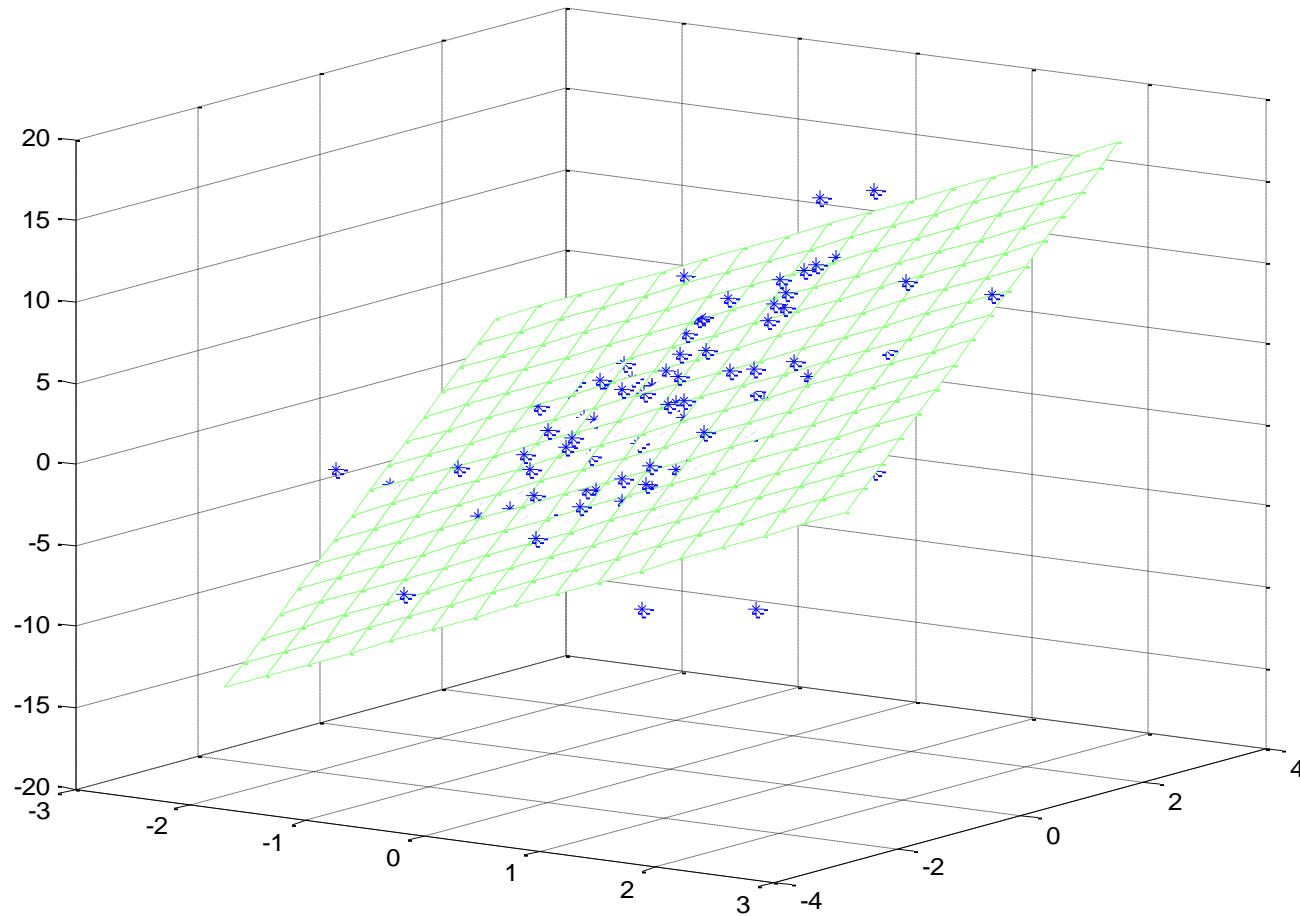
- **On line update** for  $\langle x, y \rangle$  pair

$$w_0 = w_0 + \alpha(y - f(\mathbf{x}, \mathbf{w}))$$

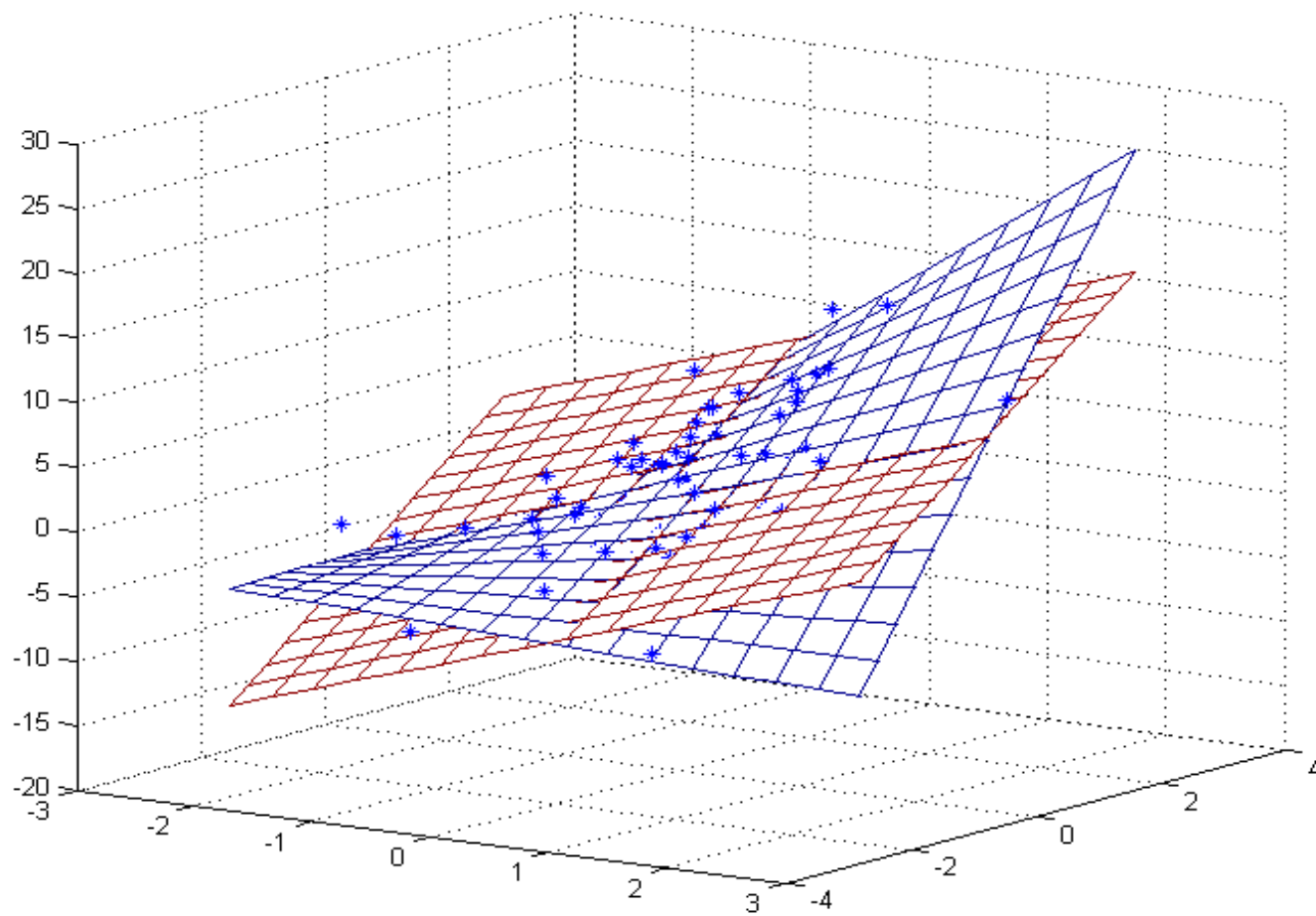
•  
•

$$w_j = w_j + \alpha(y - f(\mathbf{x}, \mathbf{w}))x^j$$

# Multidimensional additive model example



# Multidimensional additive model example



# Overview of Kernel Methods

Prof. Bennett

Math Model of Learning and  
Discovery 2/27/05

Based on Chapter 2 of  
Shawe-Taylor and Cristianini



# Linear Regression

Given training data:


$$S = \left( (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_i, y_i), \dots, (\mathbf{x}_n, y_n) \right)$$

points  $\mathbf{x}_i \in \mathbb{R}^n$  and labels  $y_i \in \mathbb{R}$

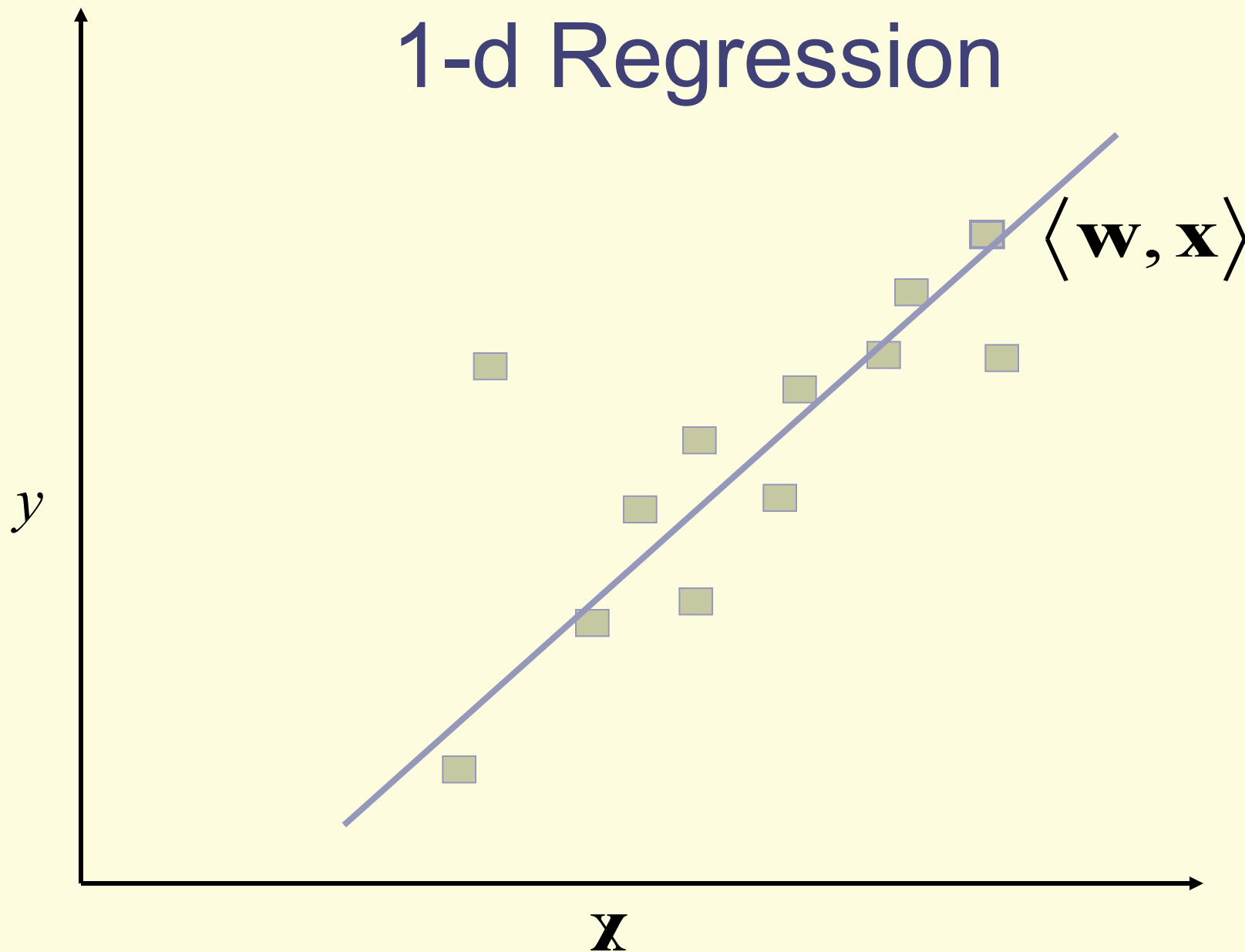
• Construct linear function:

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}' \mathbf{x} = \sum_{i=1}^n w_i x_i$$

• Creates pattern function:

$$f(\mathbf{x}, y) = y - g(\mathbf{x}) = y - \langle \mathbf{w}, \mathbf{x} \rangle \approx 0$$


# 1-d Regression








# Predict Drug Absorption

- Human intestinal cell line CACO-2
- Predict permeability  $y \in R$
- 718 descriptors generated
  - Electronic TAE
  - Shape/Property (PEST)  $\mathbf{x}_i \in R^{718}$
  - Traditional
- 27 molecules with tested permeability

1 = 28






# Least Squares Approximation

• Want  $g(x) \approx y$

• Define error  $f(\mathbf{x}, y) = y - g(\mathbf{x}) = \xi$

• Minimize loss

$$\begin{aligned} L(g, s) &= L(w, S) = \sum_{i=1}^l (y_i - g(\mathbf{x}_i)) \\ &= \sum_{i=1}^l \xi_i^2 = \sum_{i=1}^l l((\mathbf{x}_i, y_i), g) \end{aligned}$$




# Optimal Solution

• Want:  $\mathbf{y} \approx \mathbf{X}\mathbf{w}$

• Mathematical Model:

$$\min_{\mathbf{w}} L(\mathbf{w}, S) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})'(\mathbf{y} - \mathbf{X}\mathbf{w})$$

• Optimality Condition:

$$\frac{\partial L(\mathbf{w}, S)}{\partial \mathbf{w}} = -2\mathbf{X}'\mathbf{y} + 2\mathbf{X}'\mathbf{X}\mathbf{w} = 0$$

• Solution satisfies:  $\mathbf{X}'\mathbf{X}\mathbf{w} = \mathbf{X}'\mathbf{y}$

Solving  $n \times n$  equation is  $O(n^3)$





# Solution

- Assume  $(\mathbf{X}'\mathbf{X})^{-1}$  exists, then

$$\mathbf{X}'\mathbf{X}\mathbf{w} = \mathbf{X}'\mathbf{y} \Rightarrow \mathbf{w} = (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{y}$$

- Alternative Representation:

$$\begin{aligned}\mathbf{w} &= (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{y} = \mathbf{X}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{y} \\ &= \mathbf{X}'\left(\mathbf{X}(\mathbf{X}'\mathbf{X})^{-2} \mathbf{X}'\mathbf{y}\right) = \mathbf{X}'\boldsymbol{\alpha}\end{aligned}$$

$$\text{where } \boldsymbol{\alpha} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-2} \mathbf{X}'\mathbf{y}, \mathbf{w} = \sum_{i=1}^l \alpha_i \mathbf{x}_i$$

Is this a good assumption?



# Ridge Regression

- Inverse typically does not exist.
- Use least norm solution for fixed  $\lambda > 0$ .
- **Regularized problem**

$$\min_{\mathbf{w}} L_{\lambda}(\mathbf{w}, S) = \lambda \|\mathbf{w}\|^2 + \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

- Optimality Condition:

$$\frac{\partial L_{\lambda}(\mathbf{w}, S)}{\partial \mathbf{w}} = 2\mathbf{w} - 2\mathbf{X}'\mathbf{y} + 2\mathbf{X}'\mathbf{X}\mathbf{w} = 0$$

$$(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n)\mathbf{w} = \mathbf{X}'\mathbf{y}$$

Requires  $O(n^3)$  operations

# Ridge Regression (cont)

- Inverse always exists for any  $\lambda > 0$ .

$$\mathbf{w} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1} \mathbf{X}'\mathbf{y}$$

- Alternative representation:

$$(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}'\mathbf{y} \Rightarrow \mathbf{w} = \lambda^{-1} (\mathbf{X}'\mathbf{y} - \mathbf{X}'\mathbf{X}\mathbf{w})$$

$$\Rightarrow \mathbf{w} = \lambda^{-1} \mathbf{X}'(\mathbf{y} - \mathbf{X}\mathbf{w}) = \mathbf{X}'\boldsymbol{\alpha}$$

$$\boldsymbol{\alpha} = \lambda^{-1} (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\Rightarrow \lambda\boldsymbol{\alpha} = (\mathbf{y} - \mathbf{X}\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{X}'\boldsymbol{\alpha})$$

$$\Rightarrow \mathbf{X}\mathbf{X}'\boldsymbol{\alpha} + \lambda\boldsymbol{\alpha} = \mathbf{y}$$

Solving  $\nabla$  equation is  $O(l^3)$

$$\Rightarrow \boldsymbol{\alpha} = (\mathbf{G} + \lambda\mathbf{I}_l)^{-1} \mathbf{y} \text{ where } \mathbf{G} = \mathbf{X}\mathbf{X}'$$

# Dual Ridge Regression

- To predict new point:

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \left\langle \sum_{i=1}^l \alpha_i \mathbf{x}_i, \mathbf{x} \right\rangle = \mathbf{y}' (\mathbf{G} + \lambda \mathbf{I})^{-1} \mathbf{z}$$

where  $\mathbf{z} = \langle \mathbf{x}_i, \mathbf{x} \rangle$

- Note need only compute  $\mathbf{G}$ , the Gram Matrix  $\mathbf{G} = \mathbf{X}\mathbf{X}'$   $G_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

Ridge Regression requires only  
inner products between data points

# Efficiency

- To compute

$\mathbf{w}$  in primal ridge regression is  $O(n^3)$

$\alpha$  in primal ridge regression is  $O(p)$

- To predict new point  $\mathbf{x}$

primal

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^n w_i (\mathbf{x})_i \quad O(n)$$

dual


$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \left\langle \sum_{i=1}^l \alpha_i \mathbf{x}_i, \mathbf{x} \right\rangle = \sum_{i=1}^l \alpha_i \left( \sum_{j=1}^n (\mathbf{x}_i)_j (\mathbf{x})_j \right) \quad O(nl)$$

Dual is better if  $n \gg l$





# Notes on Ridge Regression

- “Regularization” is key to address stability and regularization.
  - Regularization lets method work when  $n \gg p$ .
  - Dual more efficient when  $n \gg p$ .
  - Dual only requires inner products of data.
- 



# Linear Regression in Feature Space

Key Idea:

Map data to higher dimensional space (feature space) and perform linear regression in embedded space.

Embedding Map:

$$\phi : \mathbf{x} \in R^n \rightarrow F \subseteq R^N \quad N \gg n$$



# Nonlinear Regression in Feature Space

In primal representation:

$$\mathbf{x} = [a, b]$$

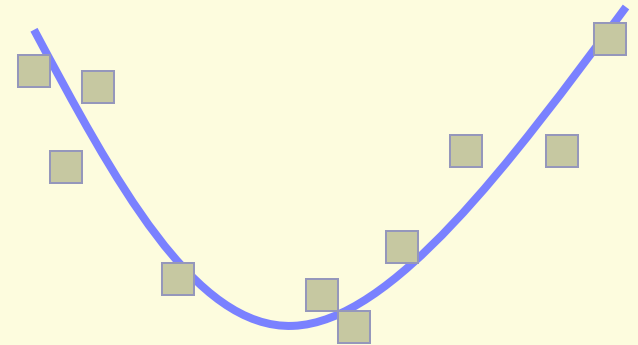
$$\langle \mathbf{x}, \mathbf{w} \rangle = w_1 a + w_2 b$$

↓

$$\theta(\mathbf{x}) = [a^2, b^2, \sqrt{2}ab]$$

$$g(\mathbf{x}) = \langle \theta(\mathbf{x}), \mathbf{w} \rangle_F$$

$$= w_1 a^2 + w_2 b^2 + w_3 \sqrt{2}ab$$



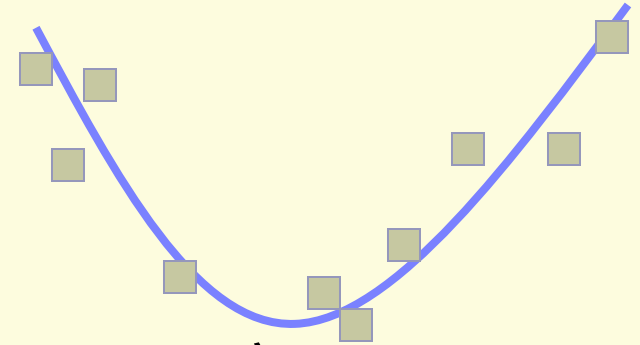
# Nonlinear Regression in Feature Space

In dual representation:

$$g(\mathbf{x}) = \langle \phi(\mathbf{x}), \mathbf{w} \rangle_F$$

$$= \sum_{i=1}^l \alpha_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle$$

$$= \sum_{i=1}^l \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$






# Kernel Function

- A kernel is a function  $K$  such that

$$K \langle \mathbf{x}, \mathbf{u} \rangle = \langle \phi(\mathbf{x}), \phi(\mathbf{u}) \rangle_F$$

where  $\phi$  is a mapping from input space to feature space  $F$ .

- There are many possible kernels.  
Simplest is linear kernel.


$$K \langle \mathbf{x}, \mathbf{u} \rangle = \langle \mathbf{x}, \mathbf{u} \rangle$$




# Derivation of Kernel

$$\begin{aligned} & \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle \\ &= \left\langle (u_1^2, u_2^2, \sqrt{2}u_1u_2), (v_1^2, v_2^2, \sqrt{2}v_1v_2) \right\rangle \\ &= u_1^2v_1^2 + u_2^2v_2^2 + 2u_1u_2v_1v_2 \\ &= (u_1v_1 + u_2v_2)^2 \\ &= \langle \mathbf{u}, \mathbf{v} \rangle^2 \end{aligned}$$

Thus:  $K(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, \mathbf{v} \rangle^2$



Age Group	Don't know	Not a fan	Dislike	Like	Love
18-29	10%	5%	15%	40%	30%
30-39	10%	5%	15%	40%	30%
40-49	10%	5%	15%	40%	30%
50-59	10%	5%	15%	40%	30%
60+	10%	5%	15%	40%	30%

- 

$$g(\phi(\mathbf{x})) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \left\langle \sum_{i=1}^l \alpha_i \phi(\mathbf{x}_i), \phi(\mathbf{x}) \right\rangle = \mathbf{y}' (\mathbf{G} + \lambda \mathbf{I})^{-1} \mathbf{z}$$

where  $\mathbf{z} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle$

- 

$$\mathbf{G} = \phi(\mathbf{X})\phi(\mathbf{X})' \qquad G_{ij} = \left\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \right\rangle = K(\mathbf{x}_i, \mathbf{x}_j)$$

## Use kernel to compute inner product






# Popular Kernels based on vectors

By Hilbert-Schmidt Kernels (Courant and Hilbert 1953)

$$\langle \theta(u), \theta(v) \rangle \equiv K(u, v)$$

for certain  $\theta$  and  $K$ , e.g.


$\theta(u)$	$K(u, v)$
Degree $d$ polynomial	$(\langle u, v \rangle + 1)^d$
Radial Basis Function Machine	$\exp\left(-\frac{\ u - v\ ^2}{\sigma}\right)$
Two Layer Neural Network	$\text{sigmoid}(\eta \langle u, v \rangle + c)$







# Kernels Intuition

- Kernels encode the notion of similarity to be used for a specific applications.
    - Document use cosine of “bags of text”.
    - Gene sequences can used edit distance.
  - Similarity defines distance:
$$\| \mathbf{u} - \mathbf{v} \|^2 = (\mathbf{u} - \mathbf{v})'(\mathbf{u} - \mathbf{v}) = \langle \mathbf{u}, \mathbf{u} \rangle - 2\langle \mathbf{u}, \mathbf{v} \rangle + \langle \mathbf{v}, \mathbf{v} \rangle$$
  - Trick is to get right encoding for domain.
- 



# Important Points

- Kernel method =  
linear method + embedding in feature space.
  - Kernel functions used to do embedding efficiently.
  - Feature space is higher dimensional space so must regularize.
  - Choose kernel appropriate to domain.
- 