

---

# Regression

## Overview of Statistical Machine Learning

Simon Guenter

based in part on material by:

N.Schraudolph and

Andrew W. Moore

<http://www.cs.cmu.edu/~awm/tutorials>

# Overview

---

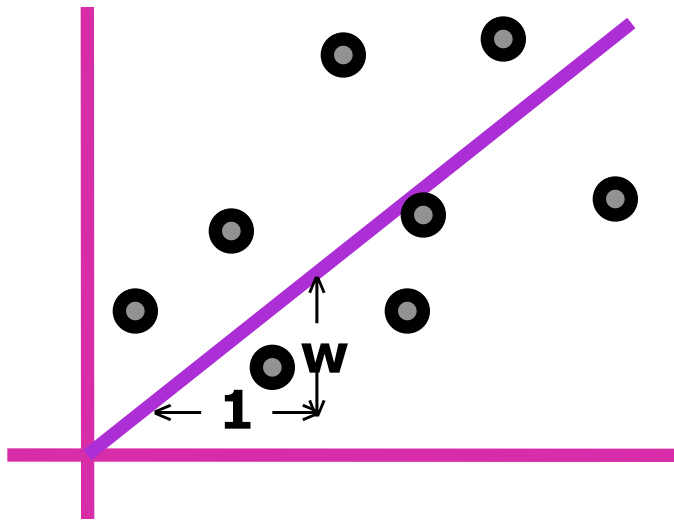
## Last lectures:

- parametric vs. non-parametric models  
semi-parametric models, mixture models
- density estimation methods

## Today:

- least-squares regression: SVD, gradient descent
- linear vs. non-linear models, basis functions

# Linear Regression



**DATA:**

inputs	outputs
$x_1 = 1$	$y_1 = 1$
$x_2 = 3$	$y_2 = 2.2$
$x_3 = 2$	$y_3 = 2$
$x_4 = 1.5$	$y_4 = 1.9$
$x_5 = 4$	$y_5 = 3.1$

Linear regression assumes that the expected value of the output given an input,  $E[y|x]$ , is linear.

Simplest case:  $\hat{y} = f(x) = wx$  for some unknown parameter  $w$ .

Given some data points  $(x_i, y_i)$ , we can estimate  $w$ .

# 1-D Linear Regression

---

Assume that the data is formed by  $y_i = wx_i + \varepsilon_i$ ,  
with  $\varepsilon_i \gg N(0, \sigma^2)$  (i.i.d. Gaussian noise).

$P(y_i|w, x_i)$  is then normally distributed with mean  $wx$   
and variance  $\sigma^2$ :  $P(y|w, x) \gg N(wx, \sigma^2)$ .

The likelihood is 
$$\prod_{i=1}^n P(y_i|w, x_i) = \prod_{i=1}^n \exp\left(-\frac{1}{2}\left(\frac{y_i - wx_i}{\sigma}\right)^2\right)$$

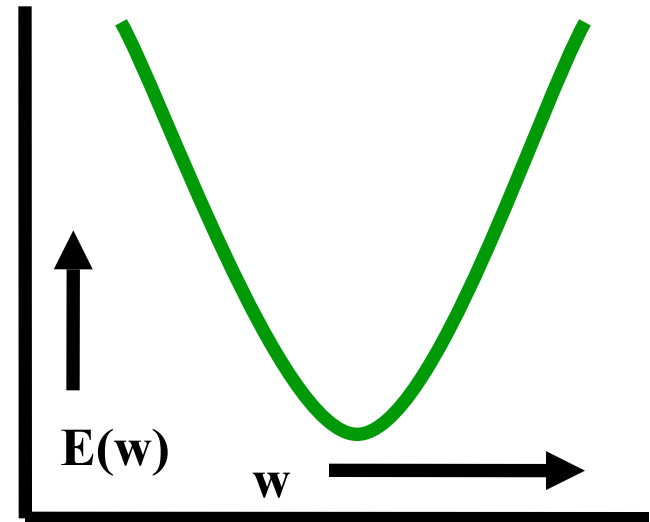
As usual, we minimize the negative log-likelihood

$$E(w) = \sum_{i=1}^n \frac{1}{2} \left( \frac{y_i - wx_i}{\sigma} \right)^2 \propto \sum_{i=1}^n (y_i - wx_i)^2$$

# 1-D Linear Least-Squares Regression

The maximum likelihood  $w$  is the one that minimizes the sum-of-squares of residuals  $y - \hat{y}$ , *i.e.* the quadratic function

$$E(w) = \sum_i (y_i - wx_i)^2$$
$$= \sum_i y_i^2 - (2 \sum_i x_i y_i)w + (\sum_i x_i^2)w^2$$



Setting the derivative to zero gives:

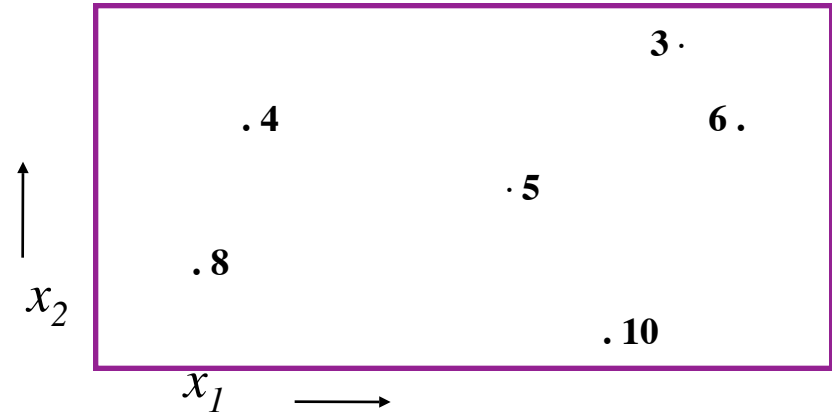
$$(\partial/\partial w) E(w^*) = 2w^* \sum x_i^2 - 2 \sum x_i y_i$$

$$w^* = \frac{\sum x_i y_i}{\sum x_i^2}$$

# Multivariate Regression

What if the inputs are vectors?

**2-d input  
example:**



For  $R$  data points in  $m$  dimensions, data set has form

$$\mathbf{X} = \begin{bmatrix} \dots \mathbf{x}_1 \dots \\ \dots \mathbf{x}_2 \dots \\ \mathbf{M} \\ \dots \mathbf{x}_R \dots \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \mathbf{M} \\ x_{R1} & x_{R2} & \dots & x_{Rm} \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \mathbf{M} \\ y_R \end{bmatrix}$$

# Multivariate Regression

---

The linear regression model assumes a parameter vector  $\mathbf{w}$

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} = w_1 x_1 + w_2 x_2 + \dots + w_m x_m$$

or in matrix notation:  $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$ . The sum-squared loss is now

$E(\mathbf{w}) = (\hat{\mathbf{y}} - \mathbf{y})^T (\hat{\mathbf{y}} - \mathbf{y})$ ; setting its derivative to zero gives

$$(\partial/\partial \mathbf{w}) E(\mathbf{w}^*) = 2\mathbf{X}^T(\hat{\mathbf{y}} - \mathbf{y}) = 0 \quad \rightarrow \quad \mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

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

$\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$  is called the **pseudo-inverse** of  $\mathbf{X}$ ; it is best obtained by **singular value decomposition (SVD)**.

in Matlab: `pinv(X)`

# Singular value decomposition

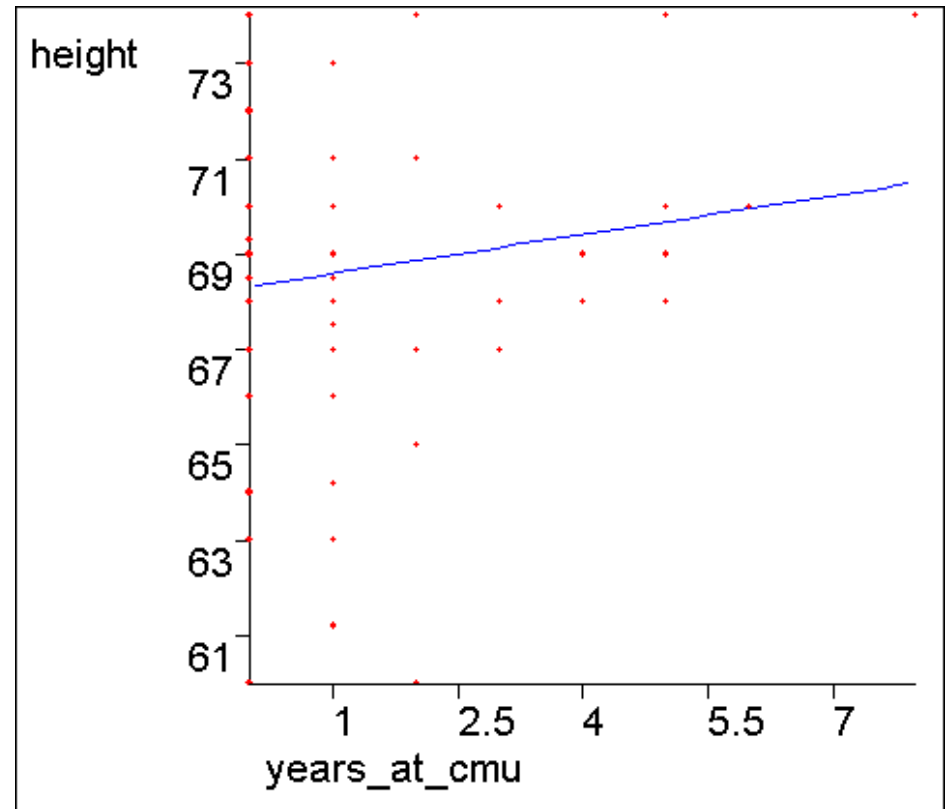
---

- ❑ Related to eigenvalue decomposition
- ❑ Given input matrix  $X$  with input patterns as rows
- ❑ Result of SVD is  $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$
- ❑  $\mathbf{U}$ ,  $\mathbf{V}$  are orthogonal matrices
- ❑  $\mathbf{D}$  is diagonal matrix with  $\forall i < j \quad \mathbf{D}[i, i] \geq \mathbf{D}[j, j]$
- ❑  $d_i = \mathbf{D}[i, i]$  are the singular values
- ❑ Easy to calculate inverse of diagonal matrix
- ❑ Inverse of orthogonal matrix is its transpose

# What about a constant offset?

We may have linear data that does not go through the origin.

To deal with this, there is a simple, universally adopted hack.



# The Bias Input

The trick: create a “fake” **bias input**  $x_0$  that is always equal to 1:

$x_1$	$x_2$	$y$
2	4	16
3	4	17
5	5	20

Before:

$$y = w_1x_1 + w_2x_2$$

...bound to be a lousy fit!

$x_0$	$x_1$	$x_2$	$y$
1	2	4	16
1	3	4	17
1	5	5	20

After:

$$y = w_0x_0 + w_1x_1 + w_2x_2$$

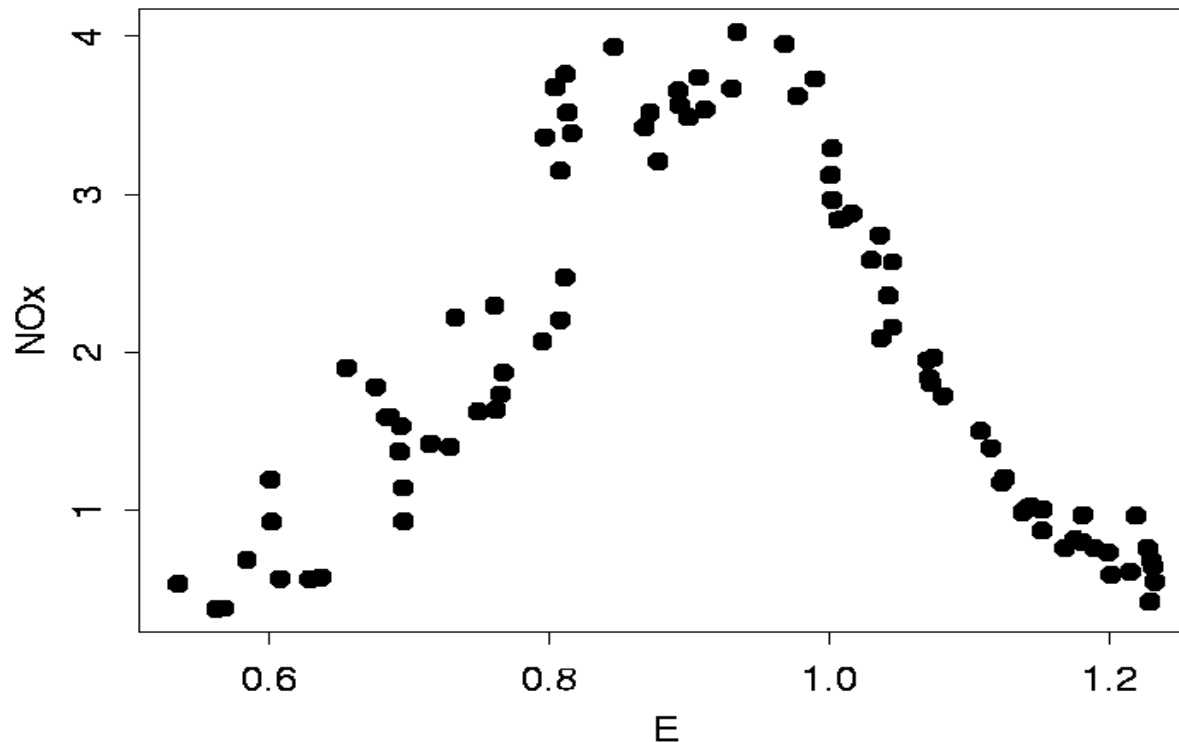
$$= w_0 + w_1x_1 + w_2x_2$$

...has a fine constant term  $w_0$ !

# What about non-linear data?

---

Relative concentration of NO and NO<sub>2</sub> in exhaust fumes as a function of the richness of the ethanol/air mixture burned in a car engine:



**No straight line will ever fit this data well!**

# Generalized Least-Squares Regression

---

The trick: add extra inputs whose values are some **non-linear** function(s) of the original inputs. Least-squares regression now allows us to find the linear combination of these non-linear **basis functions** that best fits our data.

Example:

quadratic basis

$x_0$	$x_1$	$x_2$	$x_1^2$	$x_2^2$	$x_1x_2$	$y$
1	2	4	4	16	8	16
1	3	4	9	16	12	17
1	5	5	25	25	25	20

The algorithm (compute pseudo-inverse of  $X$  via SVD) remains the same, except that now  $X$  may be substantially bigger.

# Basis Functions

---

Commonly used basis functions include

- ⌘ polynomial basis:  $x_k = x^k$  (not good for extrapolation!)
- ⌘ Fourier basis:  $x_k = [\sin(kx), \cos(kx)]$  (for periodic signals)
- ⌘ Wavelets, in 2-D: Gabor functions, Gaussians, ...

Many of these bases are **orthogonal** (*i.e.*,  $E(x_i x_j) = 0 \forall i \neq j$ ) and/or **universal**, *i.e.* any „reasonable“ function can be approximated to any degree given a sufficiently large basis.

Important: the choice of basis determines the character of the regression model; this form of prior is called **inductive bias**.

# Non-linear regression

---

There are cases where linear regression doesn't work well

- ❑ Example: We know that  $y = a \cdot e^{bx}$
- ❑ To use all bases ( $e^x, e^{2x}, e^{3x} \dots$ ) is impractical
- ❑  $b$  may not even be an integer
- ❑ Solution: minimize  $E(a, b) = \sum_i (y_i - a \cdot e^{bx_i})^2$
- ❑ Often  $\frac{\partial E}{\partial w} = 0$  has no close form solution (which exists in the linear case)  $\Rightarrow$  use optimization methods, e.g. gradient descent

# Simple Gradient Descent

---

Given  $f(\mathbf{w}) : \mathbb{R}^m \times \mathbb{R}$ , the **gradient**

$$\nabla f(\mathbf{w}) = \begin{pmatrix} \frac{\partial}{\partial w_1} f(\mathbf{w}) \\ \vdots \\ \frac{\partial}{\partial w_m} f(\mathbf{w}) \end{pmatrix} \quad \text{points in the direction} \\ \text{of **steepest ascent** of } f.$$

Simple gradient descent rule:  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w})$

for the  $j^{\text{th}}$  weight:  $w_j \leftarrow w_j - \eta \frac{\partial}{\partial w_j} f(\mathbf{w})$

where **learning rate**  $\eta$  is a small positive number, *e.g.*  $\eta = 0.05$ .

Start values for parameters must be selected

# Gradient Descent Example

---

$$y = a \cdot e^{bx}$$

$$E(a, b) = \sum_i (y_i - a \cdot e^{bx_i})^2$$

$$\frac{\partial E}{\partial a} = \sum_i -2(y_i - a \cdot e^{bx_i}) \cdot e^{bx_i}$$

$$\frac{\partial E}{\partial b} = \sum_i -2(y_i - a \cdot e^{bx_i}) \cdot a \cdot x_i \cdot e^{bx_i}$$

Use estimations of real value for initial a and b

Use some small value for the learning rate

Control value of E, if it increases -> learning rate too high

# Gradient Descent for LS Regression

$$E(\mathbf{w}) = \sum_{k=1}^R (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

Gradient descent tells us we should update  $\mathbf{w}$  thusly if we wish to minimize  $E$ :

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

So what's  $\frac{\partial E}{\partial w_j}$ ?

$$\begin{aligned} \frac{\partial E}{\partial w_j} &= \sum_{k=1}^R \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)^2 \\ &= \sum_{k=1}^R 2(y_k - \mathbf{w}^T \mathbf{x}_k) \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k) \\ &= -2 \sum_{k=1}^R \ddot{a}_k \frac{\partial}{\partial w_j} \mathbf{w}^T \mathbf{x}_k \end{aligned}$$

where  
 $\ddot{a}_k = y_k - \mathbf{w}^T \mathbf{x}_k$

$$\begin{aligned} &= -2 \sum_{k=1}^R \ddot{a}_k \frac{\partial}{\partial w_j} \sum_{i=1}^m w_i x_{ki} \\ &= -2 \sum_{k=1}^R \ddot{a}_k x_{kj} \end{aligned}$$

# Gradient Descent for LS Regression

$$E(\mathbf{w}) = \sum_{k=1}^R (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

Gradient descent tells us we should update  $\mathbf{w}$  thusly if we wish to minimize  $E$ :

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

where

$$\frac{\partial E}{\partial w_j} = -2 \sum_{k=1}^R \ddot{a}_k x_{kj}$$

We frequently neglect the 2 (meaning we halve the learning rate)

$$w_j \leftarrow w_j + 2\eta \sum_{k=1}^R \ddot{a}_k x_{kj}$$

where  $\ddot{a}_k = y_k - \mathbf{w}^T \mathbf{x}_k$

# Why Gradient Descent for LS Regression

---

- ❑ Given  $R$  patterns and  $m$  parameters
- ❑ Time complexity of exact solution  
 $O(mR^2 + m^2R + R^3)$
- ❑ Time complexity of gradient descent  $O(mRI)$   
with  $I$  number of iterations