

Applied Machine Learning

Linear Regression

Siamak Ravanbakhsh

COMP 551 (winter 2020)

Learning objectives

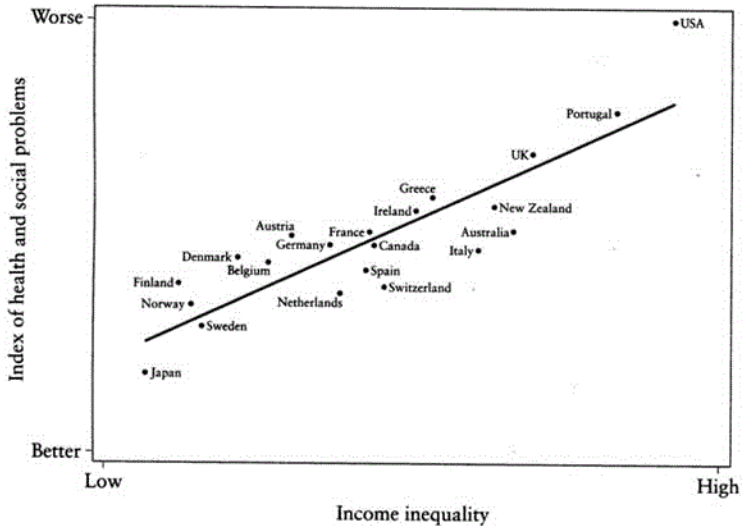
- linear model
- evaluation criteria
- how to find the best fit
- geometric interpretation

Motivation

History: method of least squares was invented by **Legendre** and **Gauss** (1800's)

Gauss at age 24 used it to predict the future location of Ceres (largest astroid in the astroid belt)

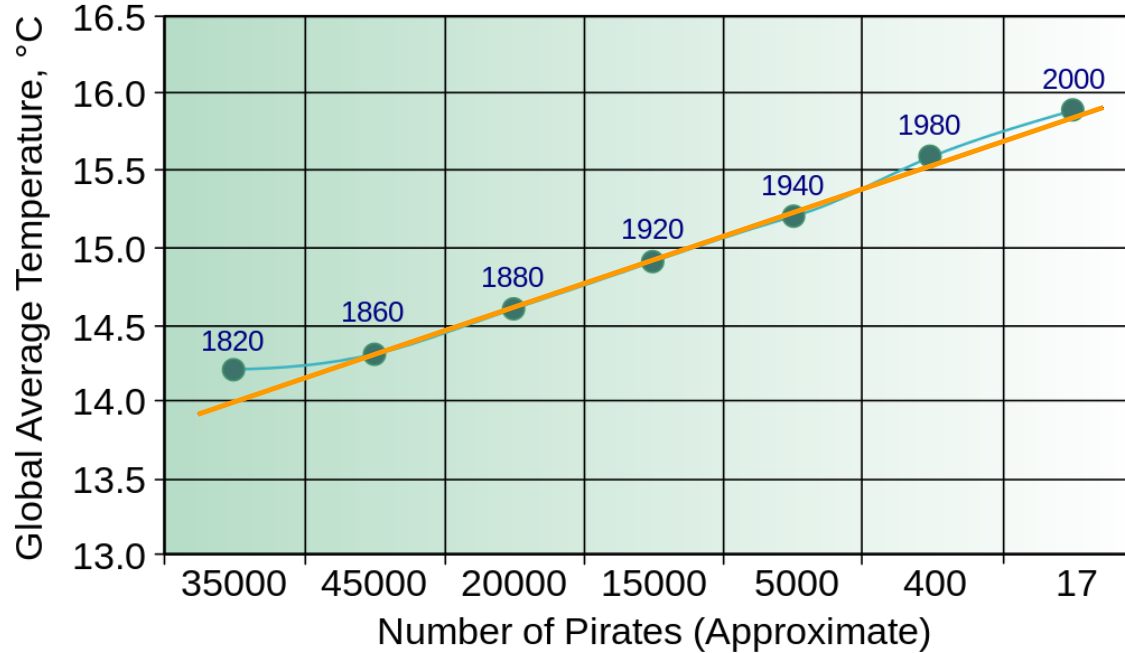
effect of income inequality on health and social problems



source: <http://chrisauld.com/2012/10/07/what-do-we-know-about-the-effect-of-income-inequality-on-health/>

Motivation (?)

Global Average Temperature vs. Number of Pirates



Representing data

each instance: $\begin{cases} \mathbf{x}^{(n)} \in \mathbb{R}^D \\ \mathbf{y}^{(n)} \in \mathbb{R} \end{cases}$ one instance

vectors are assumed to be column vectors $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} = [x_1, x_2, \dots, x_D]^\top$ a feature

we assume N instances in the dataset $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$

each instance has D features indexed by d

for example, $x_d^{(n)} \in \mathbb{R}$ is the feature d of instance n

Representing data

design matrix: *concatenate all instances*

- *each row is a datapoint, each column is a feature*

$$X = \begin{bmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \vdots \\ \mathbf{x}^{(N)\top} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^{(1)}, & \mathbf{x}_2^{(1)}, & \cdots, & \mathbf{x}_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_1^{(N)}, & \mathbf{x}_2^{(N)}, & \cdots, & \mathbf{x}_D^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times D}$$

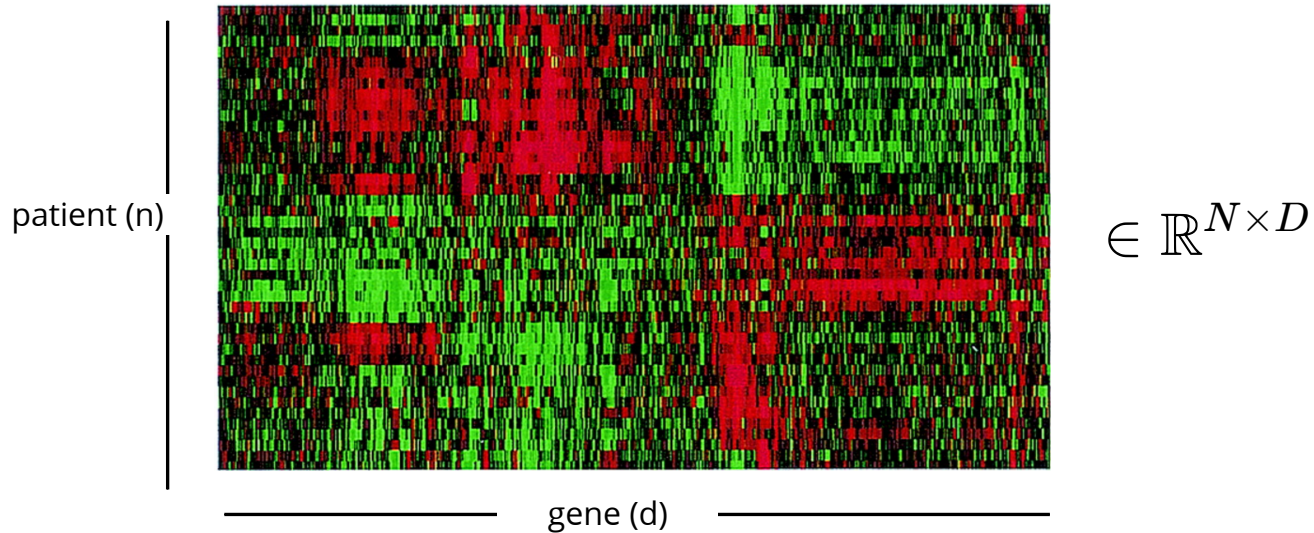
one instance

one feature

Representing data

Example:

Micro array data (X), contains gene expression levels
labels (y) can be {cancer/no cancer} label for each patient



Linear model

assuming a scalar output $f_w : \mathbb{R}^D \rightarrow \mathbb{R}$
will generalize to a vector later

$$f_w(x) = w_0 + w_1x_1 + \dots + w_Dx_D$$

model parameters or weights

bias or intercept

simplification

concatenate a 1 to $x \longrightarrow x = [1, x_1, \dots, x_D]^\top$

$$f_w(x) = w^\top x$$

...

```
yh_n = np.dot(w, x)
```


Loss function

objective: find parameters to **fit the data** $x^{(n)}, y^{(n)} \quad \forall n$

$$f_w(x^{(n)}) \approx y^{(n)}$$

minimize a measure of difference between $\hat{y}^{(n)} = f_w(x^{(n)})$ and $y^{(n)}$

square error **loss** (a.k.a. **L2** loss) $L(y, \hat{y}) \triangleq \frac{1}{2}(y - \hat{y})^2$

for a single instance (a function of labels)

versus

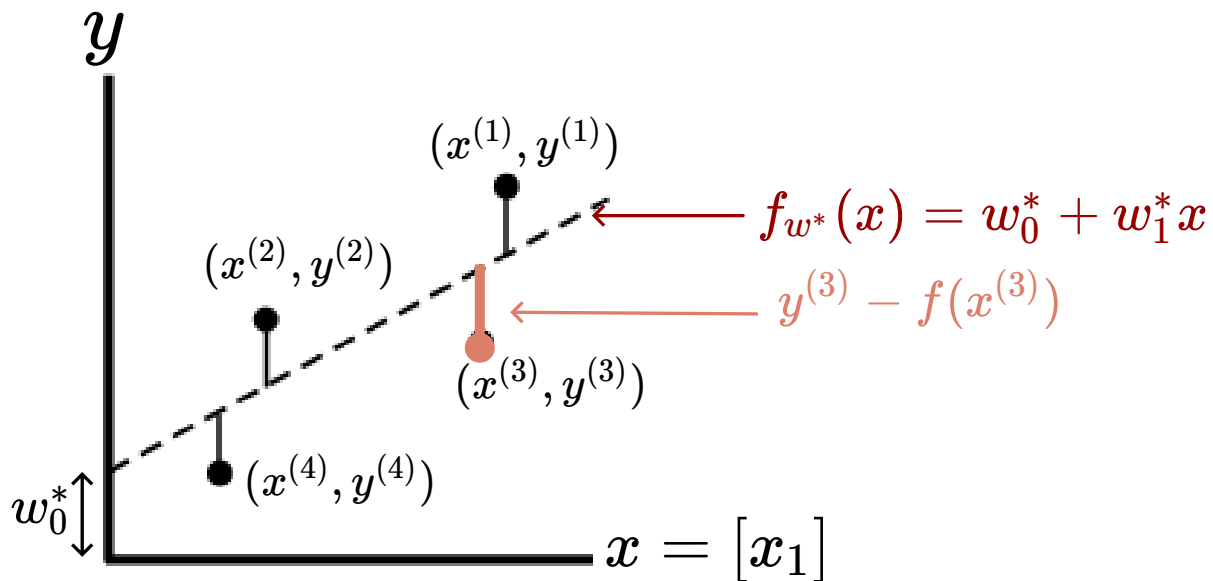
for future convenience

for the whole dataset

sum of squared errors **cost function**

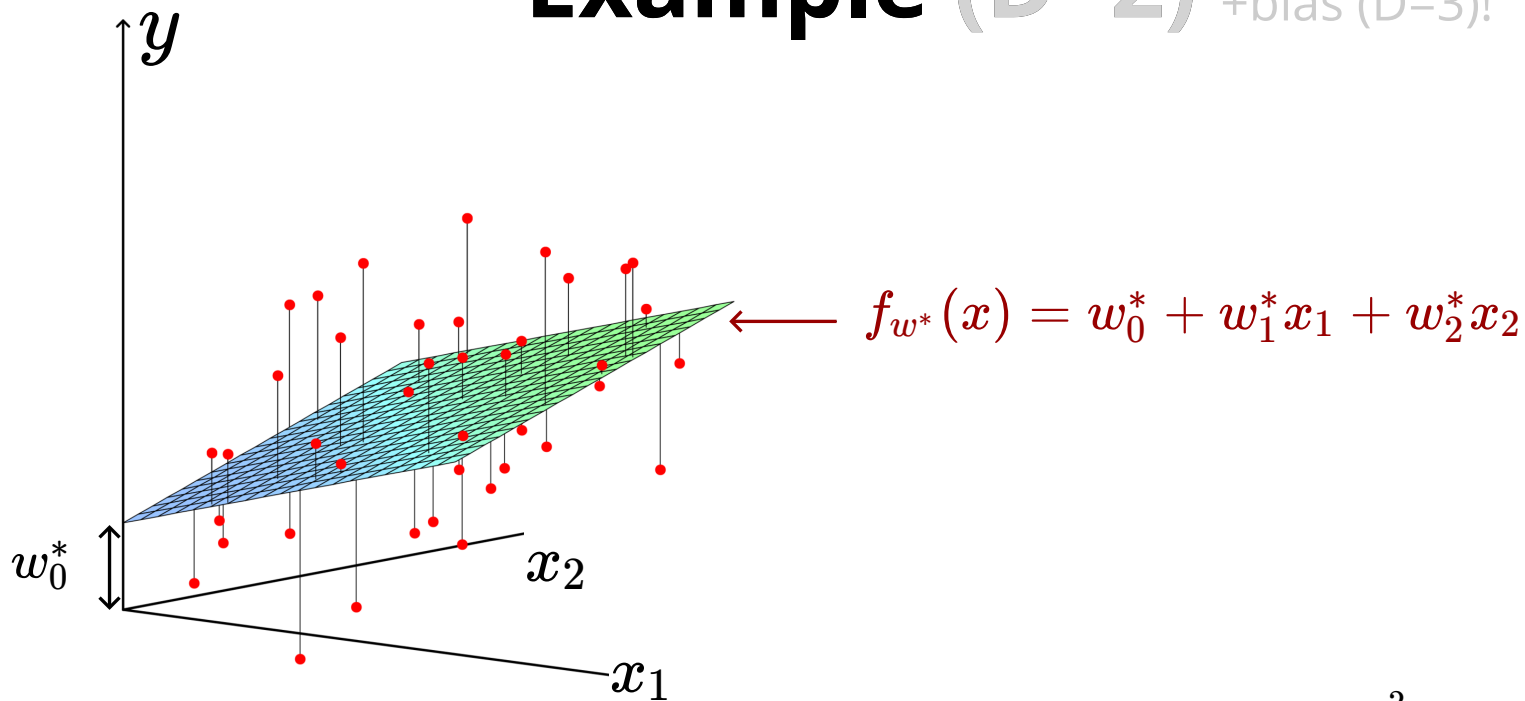
$$J(w) = \frac{1}{2} \sum_{n=1}^N \left(y^{(n)} - w^\top x^{(n)} \right)^2$$

Example ($D = 1$) +bias ($D=2$)!



Linear Least Squares $\min_w \sum_n \left(y^{(n)} - w^T x^{(n)} \right)^2$

Example (D=2) +bias (D=3)!



Linear Least Squares $w^* = \arg \min_w \sum_n \left(y^{(n)} - w^T x^{(n)} \right)^2$

Matrix form

instead of $\hat{y}^{(n)} = w^\top x^{(n)}$
 $\in \mathbb{R}$ $1 \times D$ $D \times 1$

use **design matrix** to write $\hat{y} = Xw$
 $N \times 1$ $N \times D$ $D \times 1$

Linear least squares

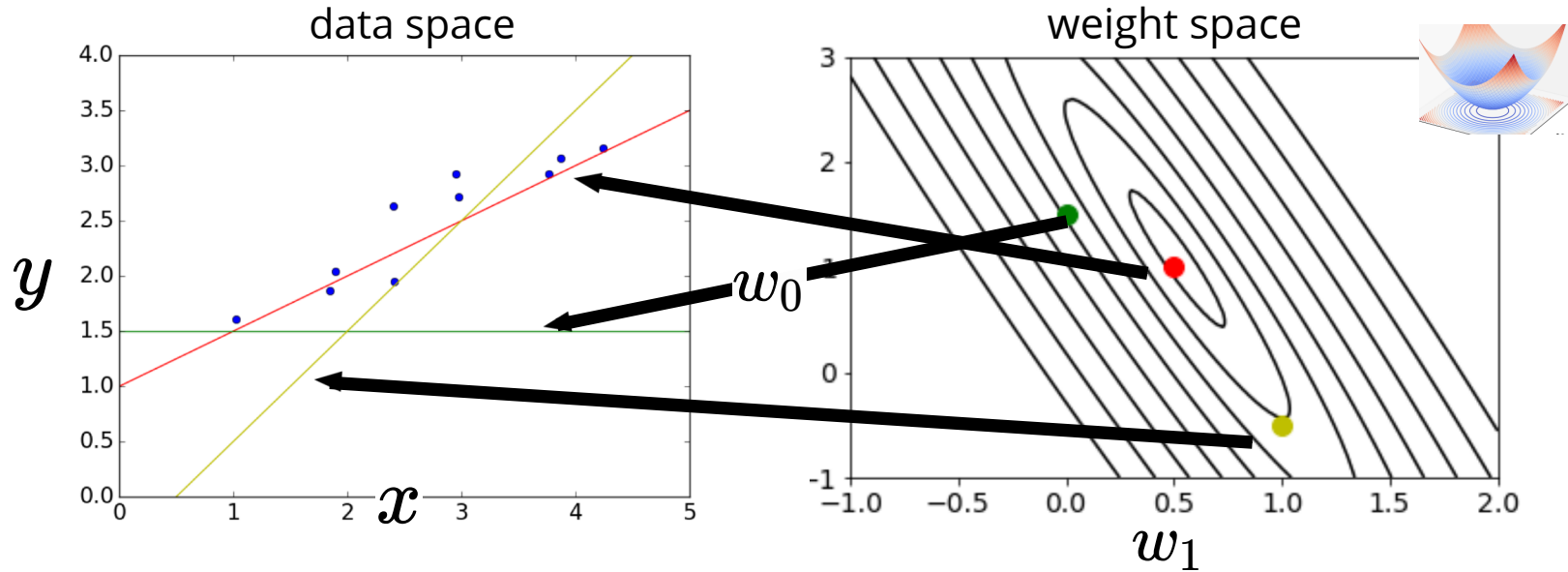
$$\arg \min_w \frac{1}{2} \|y - Xw\|^2 = \frac{1}{2} (y - Xw)^\top (y - Xw)$$

squared L2 norm of the **residual** vector



```
yh = np.dot(X, w)
cost = np.sum((yh - y)**2)/2.
# or
cost = np.mean((yh - y)**2)/2.
```

Minimizing the cost



the objective is a smooth function of w
find minimum by setting partial derivatives to zero

image: Grosse, Farahmand, Carrasquilla

Simple case: $D = 1$

model $f_w(x) = wx$
both scalar

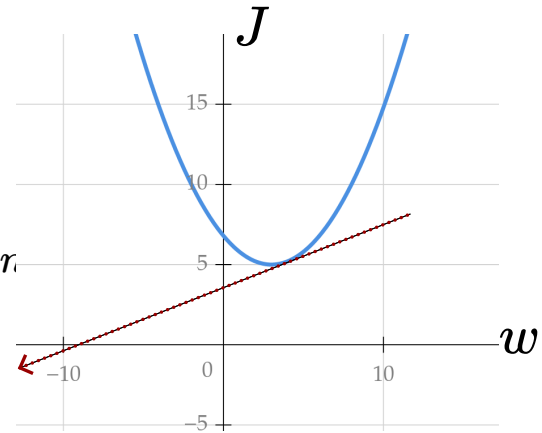
cost function $J(w) = \frac{1}{2} \sum_n (y^{(n)} - wx^{(n)})^2$

derivative $\frac{dJ}{dw} = \sum_n x^{(n)} (wx^{(n)} - y^{(n)})$

setting the derivative to zero $w = \frac{\sum_n x^{(n)} y^{(n)}}{\sum_n x^{(n)2}}$

global minimum because cost is smooth and *convex*

more on convexity layer



Gradient

for a multivariate function $J(w_0, w_1)$

partial derivatives instead of derivative

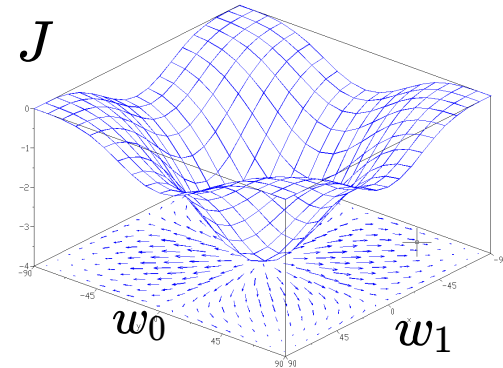
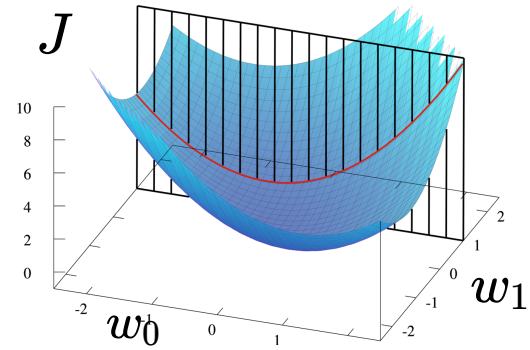
= derivative when other vars. are fixed

$$\frac{\partial}{\partial w_1} J(w_0, w_1) \triangleq \lim_{\epsilon \rightarrow 0} \frac{J(w_0, w_1 + \epsilon) - J(w_0, w_1)}{\epsilon}$$

critical point: all partial derivatives are zero

gradient: vector of all partial derivatives

$$\nabla J(w) = \left[\frac{\partial}{\partial w_1} J(w), \dots, \frac{\partial}{\partial w_D} J(w) \right]^\top$$



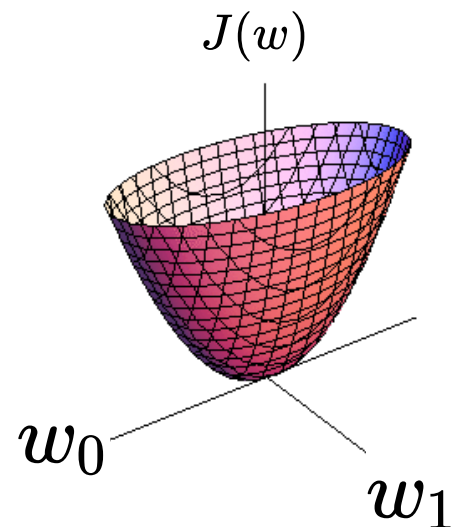
Finding w (any D)

setting $\frac{\partial}{\partial w_i} J(w) = 0$

$$\frac{\partial}{\partial w_i} \sum_n \frac{1}{2} (y^{(n)} - f_w(x^{(n)}))^2 = 0$$

using **chain rule**: $\frac{\partial J}{\partial w_i} = \frac{dJ}{df_w} \frac{\partial f_w}{\partial w_i}$

we get $\sum_n (w^\top x^{(n)} - y^{(n)}) x_d^{(n)} = 0 \quad \forall d \in \{1, \dots, D\}$



cost is a smooth and convex function of w

Normal equation

system of D linear equations

$$\sum_n (y^{(n)} - w^\top x^{(n)}) x_d^{(n)} = 0 \quad \forall d$$

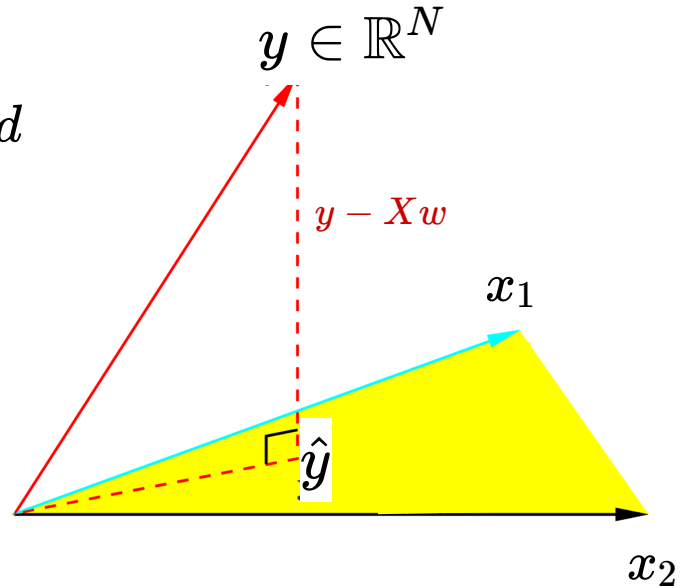
matrix form (using the design matrix)

each row enforces one of D equations

$D \times N$

$N \times 1$

$$\overline{X}^\top (\overline{y} - \overline{X}w) = \vec{0}$$



Normal equation: because for optimal w , the residual vector is normal to column space of the design matrix

Direct solution

we can get a closed form solution!

$$X^T(y - Xw) = \vec{0}$$

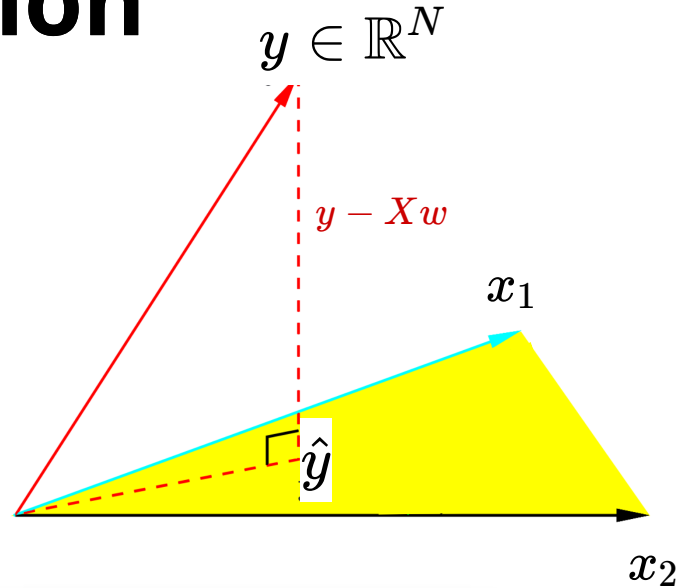
$$X^T X w = X^T y$$

pseudo-inverse of X

$$w^* = \underbrace{(X^T X)^{-1}}_{D \times D} \underbrace{X^T}_{D \times N} \underbrace{y}_{N \times 1}$$

$$\hat{y} = Xw = X(X^T X)^{-1} X^T y$$

projection matrix into column space of X



```
w = np.linalg.lstsq(X,y)[0]
```

Time complexity

$$w^* = \overbrace{(X^T X)^{-1}}^{D \times D} \overbrace{X^T y}^{D \times N \quad N \times 1}$$

$\mathcal{O}(ND)$ D elements, each using N ops.

$\mathcal{O}(D^3)$ matrix inversion

$\mathcal{O}(D^2 N)$ $D \times D$ elements, each requiring N multiplications

total complexity for $N > D$ is $\mathcal{O}(ND^2 + D^3)$

in practice we don't directly use matrix inversion (unstable)

Multiple targets

instead of $\mathbf{y} \in \mathbb{R}^N$ we have $\mathbf{Y} \in \mathbb{R}^{N \times D'}$
a different weight vectors for each target

each column of \mathbf{Y} is associated with a column of \mathbf{W}

$$\hat{\mathbf{Y}} = \mathbf{X}\mathbf{W}$$

$N \times D'$ $N \times D$ $D \times D'$

$$\mathbf{W}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

$D \times D$ $D \times N$ $N \times D'$



```
w = np.linalg.lstsq(X,Y)[0]
```

Nonlinear basis functions

so far we learned a linear function $f_w = \sum_d w_d x_d$

nothing changes if we have nonlinear bases $f_w = \sum_d w_d \phi_d(x)$

solution simply becomes $w^* = (\Phi^\top \Phi)^{-1} \Phi^\top y$

replacing X with Φ

a (nonlinear) feature

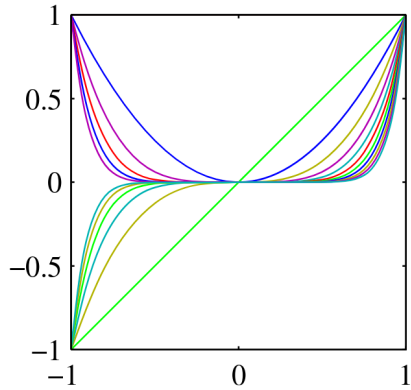
$$\Phi = \begin{bmatrix} \phi_1(x^{(1)}), & \phi_2(x^{(1)}), & \cdots, & \phi_D(x^{(1)}) \\ \phi_1(x^{(2)}), & \phi_2(x^{(2)}), & \cdots, & \phi_D(x^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x^{(N)}), & \phi_2(x^{(N)}), & \cdots, & \phi_D(x^{(N)}) \end{bmatrix}$$

one instance

Nonlinear basis functions

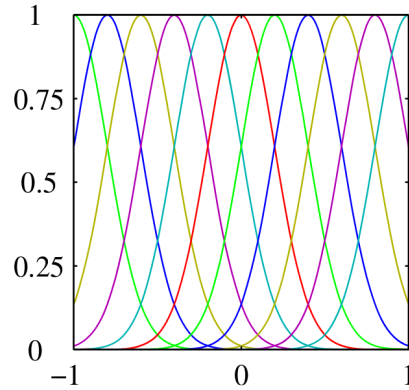
examples

original input is scalar $x \in \mathbb{R}$



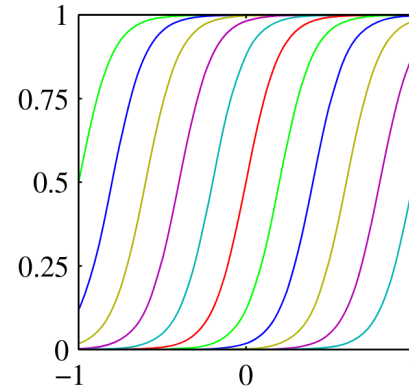
polynomial bases

$$\phi_k(x) = x^k$$



Gaussian bases

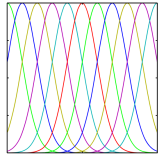
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



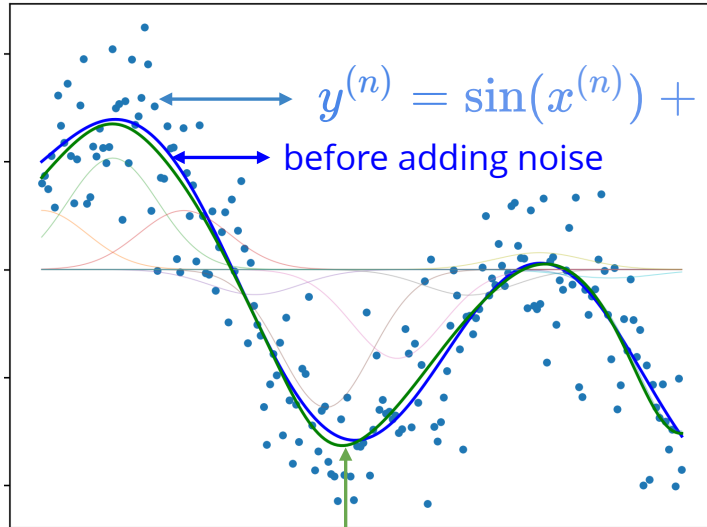
Sigmoid bases

$$\phi_k(x) = \frac{1}{1+e^{-\frac{x-\mu_k}{s}}}$$

Example: Gaussian bases



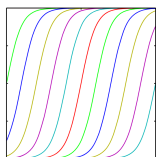
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



our fit to data using 10 Gaussian bases

noise

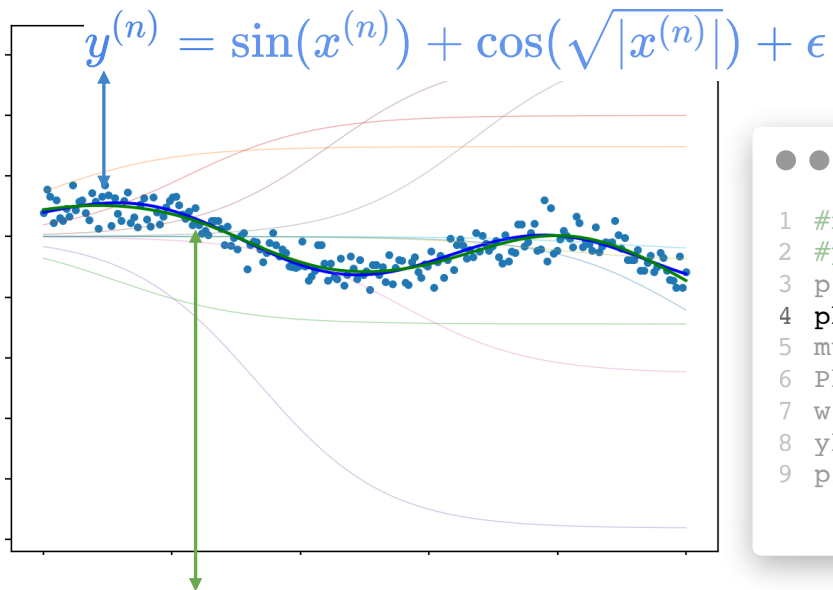
```
1 #x: N
2 #y: N
3 plt.plot(x, y, 'b.')
4 phi = lambda x,mu: np.exp(-(x-mu)**2)
5 mu = np.linspace(0,10,10) #10 Gaussian bases
6 Phi = phi(x[:,None], mu[None,:]) #N x 10
7 w = np.linalg.lstsq(Phi, y)[0]
8 yh = np.dot(Phi,w)
9 plt.plot(x, yh, 'g-')
```



Sigmoid bases

$$\phi_k(x) = \frac{1}{1 + e^{-\frac{x - \mu_k}{s}}}$$

Example: Sigmoid bases



our fit to data using 10 sigmoid bases



```

1 #x: N
2 #y: N
3 plt.plot(x, y, 'b.')
4 phi = lambda x, mu: 1/(1 + np.exp(-(x - mu)))
5 mu = np.linspace(0, 10, 10) #10 sigmoid bases
6 Phi = phi(x[:, None], mu[None, :]) #N x 10
7 w = np.linalg.lstsq(Phi, y)[0]
8 yh = np.dot(Phi, w)
9 plt.plot(x, yh, 'g-')

```


Problematic settings

In $W^* = (X^T X)^{-1} X^T Y$

what if we have a **large dataset**? $N > 100,000,000$

- use stochastic gradient descent

what if $X^T X$ is **not invertible**?

columns of X (features) are not linearly independent
(either redundant features or $D > N$)

- W^* is **not unique, make it unique** by
 - removing redundant features
 - regularization (later!)
- **or find one** of the solutions
 - decomposition-based (not discussed) methods still work
 - use gradient descent (later!)



```
w = np.linalg.lstsq(X,Y)[0]
```

Summary

linear regression:

- models targets as a **linear function of features**
- fit the model by **minimizing sum of squared errors**
- has a **direct solution** with $\mathcal{O}(ND^2 + D^3)$ complexity
- gradient descent: future

we can build more expressive models:

- using any number of **non-linear features**
- ensure features are linearly independent