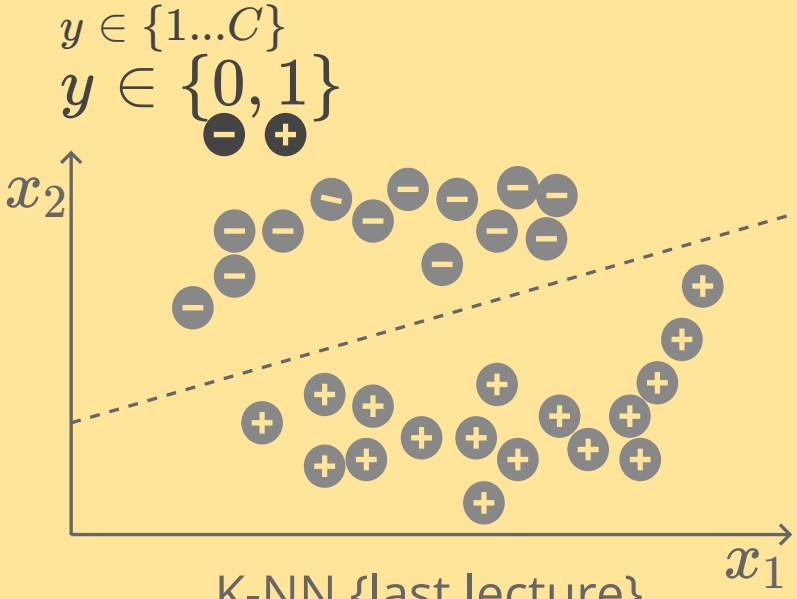
Applied Machine Learning

Linear Regression

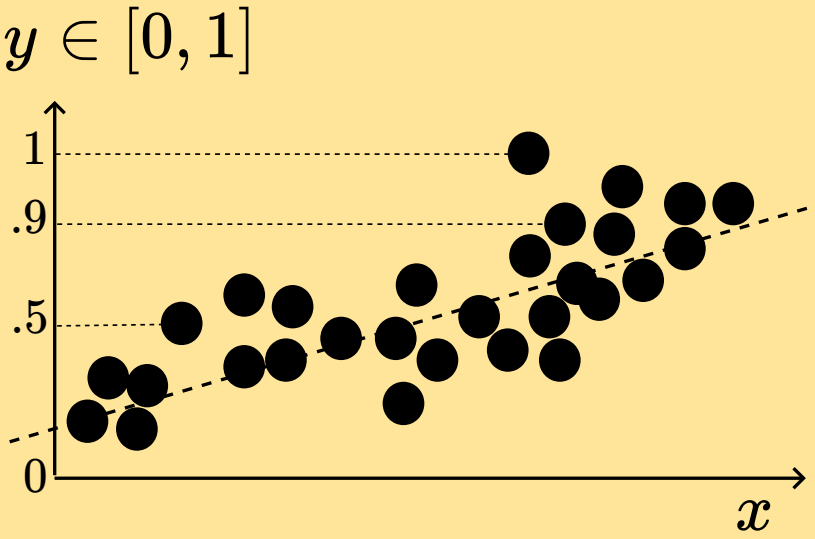
Reihaneh Rabbany

COMP 551 (winter 2020)

Linear Regression



K-NN {last lecture}
classification
method



this lecture
linear regression

Regression

$y \in \{1...C\}$

$y \in \{0, 1\}$

$y \in [0, 1]$

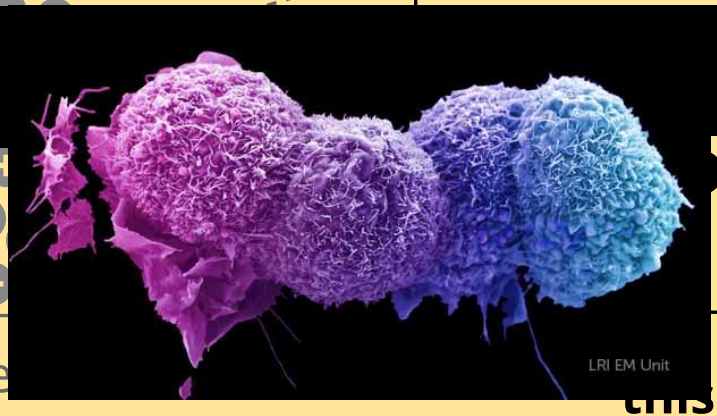
x_2

1

x

Is it cancer?

yes, no



How fast is it growing? 1.5

K-NN {last le

classification method

this lecture

linear regression

Regression: Example

Age-estimating
input: face
output: age

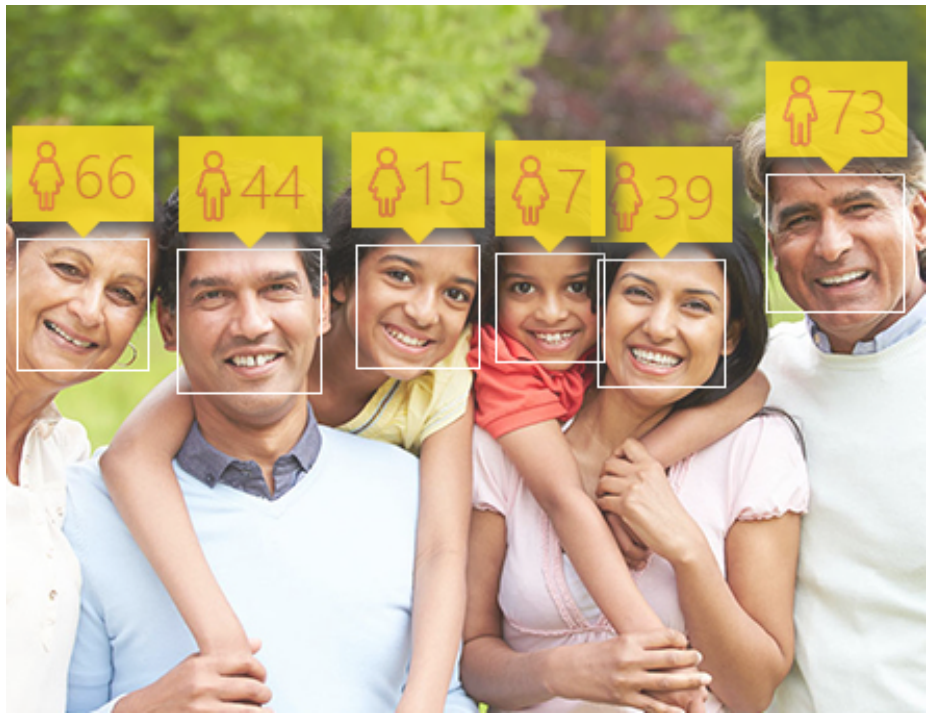


image form Microsoft age estimator here

Regression: Example

Protein folding
input: sequences
output: 3D structure

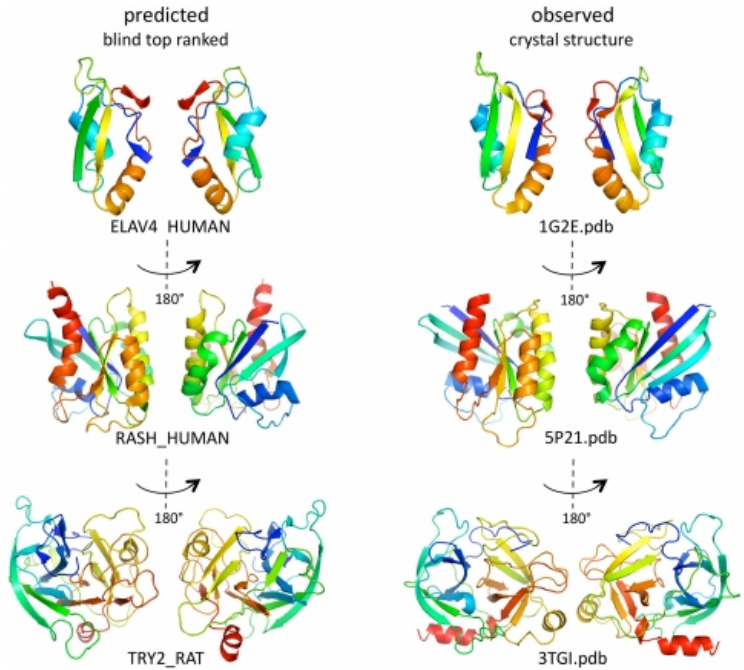


Image form Marks et al. link

Regression: Example

Colourization

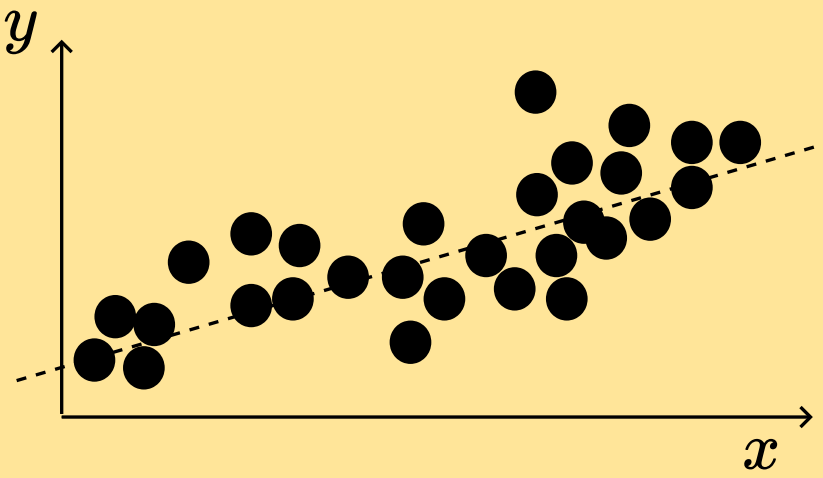
input: gray scale image, output: colour image

Image form Zhang et al. link

Learning objectives

linear least square regression

- Mathematical formulation
- Definition of loss function
- Optimization method
- Geometric interpretation
- ...



Motivation

History: method of least squares was invented by **Legendre** and **Gauss** (1800's)
Gauss used it to predict the future location of Ceres (largest astroid in the astroid belt)

ocean navigation

image from wiki history of navigation

Gauss
used it

Legendre
published it

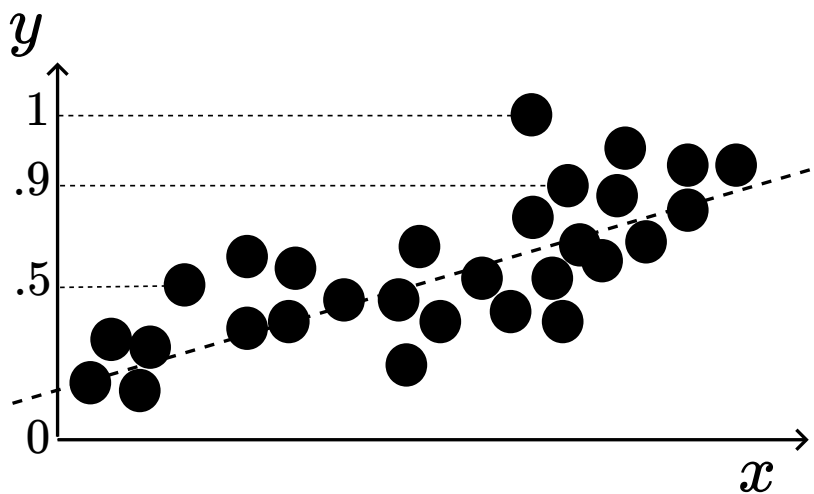
Pearson
named it regression

read more on the history if interested

Terminology

$x = [x_1, \ldots, x_D]^T$ input, features, independent var., covariates

y target, dependent var., response var.

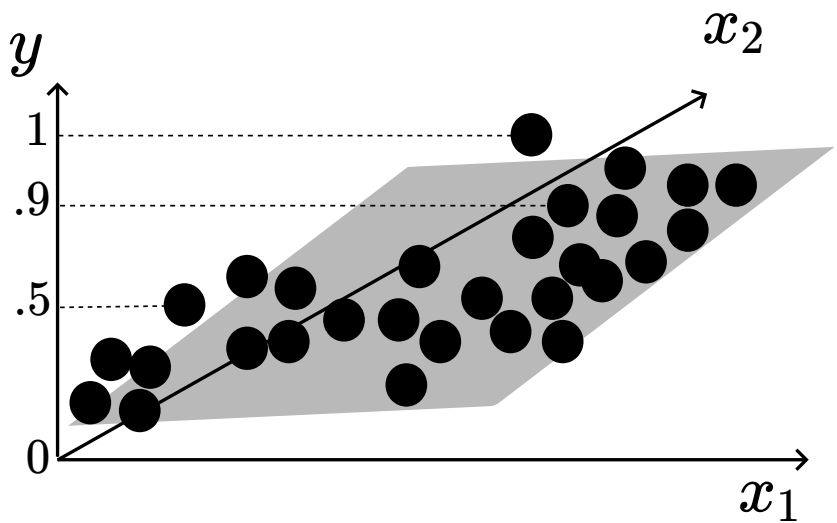


How many dimentions are in the input of this example?

Terminology

$x = [x_1, \ldots, x_D]^T$ input, features, independent var., covariates

y target, dependent var., response var.



Representing data

each instance: $\begin{array}{l} x^{(n)} \in \mathbb{R}^D \\ y^{(n)} \in \mathbb{R} \end{array}$ a (column) vector $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}$ a features

$\mathbb{R}$ denotes set of real numbers

design matrix: *concatenate all instances*

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

$$X = \begin{bmatrix} {x^{(1)}}^T \\ {x^{(2)}}^T \\ \vdots \\ {x^{(N)}}^T \end{bmatrix} = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix}$$

one instance

Representing data

design matrix: *concatenate all instances*

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

instances

$$X = \begin{bmatrix} x^{(1)T} \\ x^{(2)T} \\ \vdots \\ x^{(N)T} \end{bmatrix} = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times D} \qquad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

features

$$\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$$

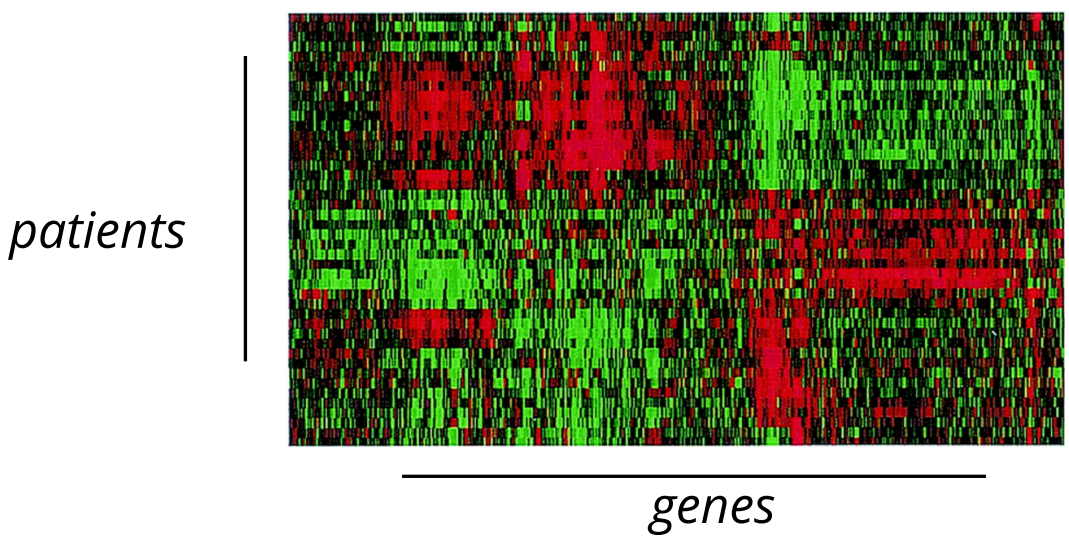
we assume N instances in the dataset

each instance has D features indexed by d

Example: representing data

design matrix: *each row is a datapoint, each column is a feature*

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



Linear model $D = 1$

assuming a scalar output $f_w : \mathbb{R} \to \mathbb{R}$

can generalize to a vector

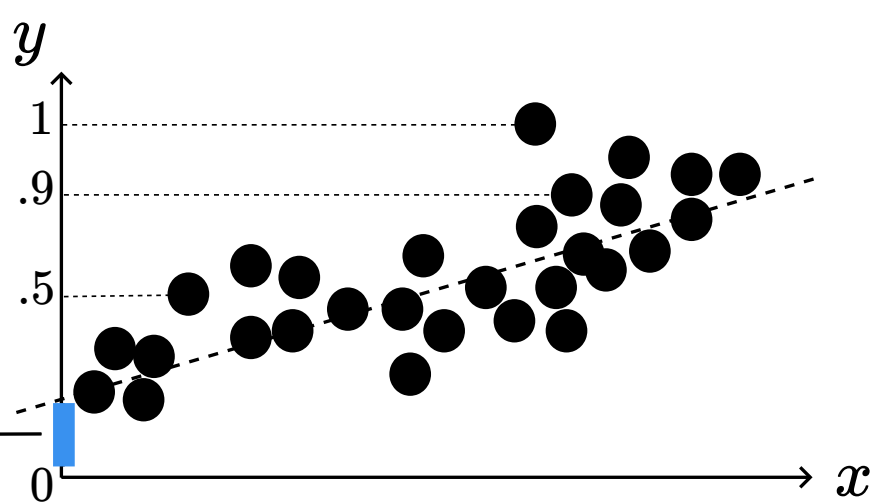
$$f_w(x) = w_0 + w_1 x_1$$

model parameters or weights

$[w_0, w_1]$

forms a line in 1 dimension

higher dimensions?



$f_w(0)$: bias or intercept

Linear model

assuming a scalar output $f_w : \mathbb{R}^D \rightarrow \mathbb{R}$

can generalize to a vector

$$f_w(x) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

w: model parameters or weights

w_0: bias or intercept

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

simplification

concatenate a 1 to x $\rightarrow$ $x = [1, x_1, \ldots, x_D]^T$

$w = [w_0, w_1, \ldots, w_D]^T$

$$f_w(x) = w^T x$$

How should we pick the weights?

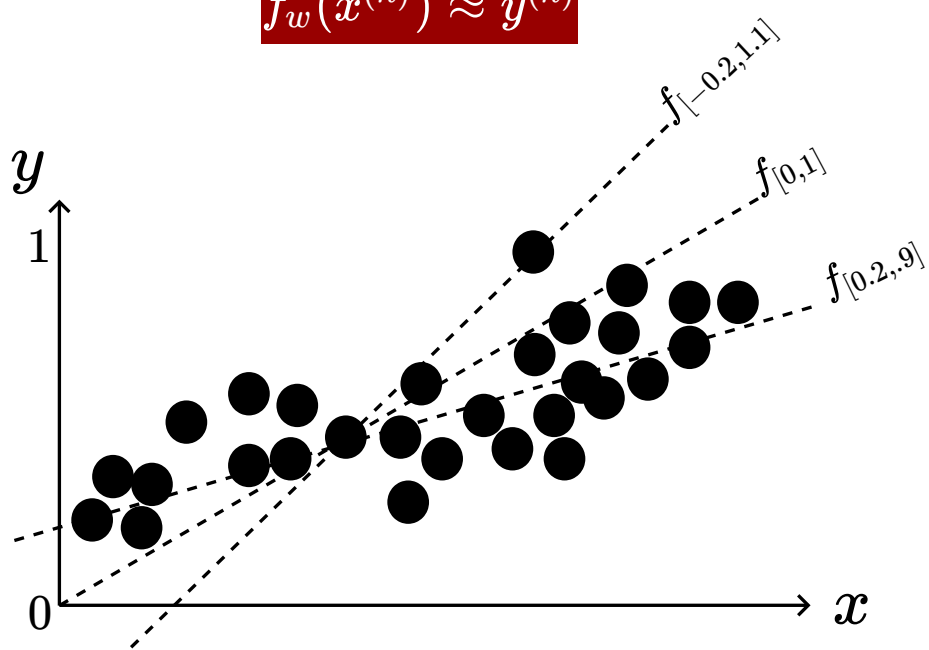
Loss function

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

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

$$f_w(x) = w^T x$$

$$w = [w_0, w_1]$$

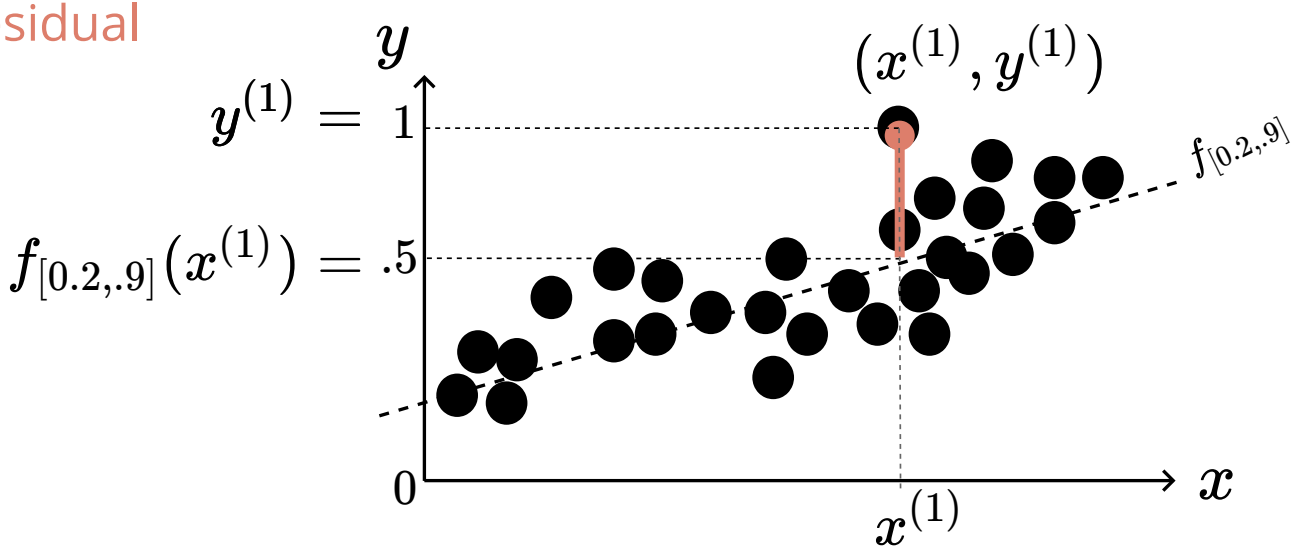


Loss function

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

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

compute residual



Loss function

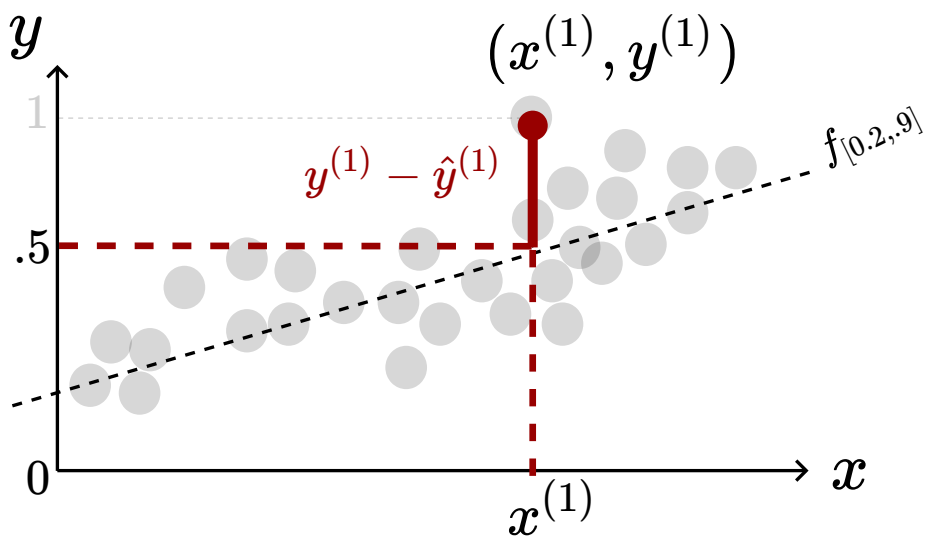
objective: find parameters to fit the data $x^{(n)}, y^{(n)} \quad \forall n$

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

why residual?

true: $y^{(1)} = 1$

predicted: $\hat{y}^{(1)} = f(x^{(1)}) = .5$

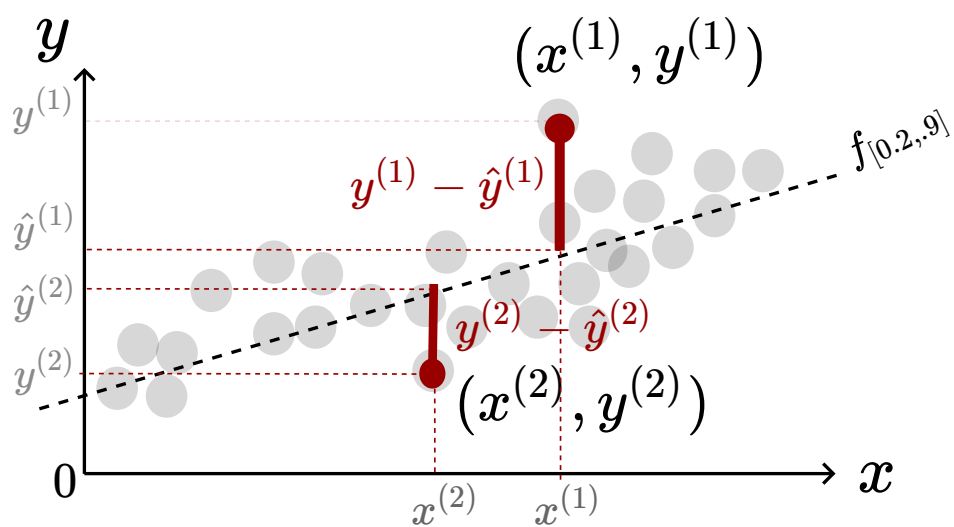


Loss function

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

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

how to sum all residuals?



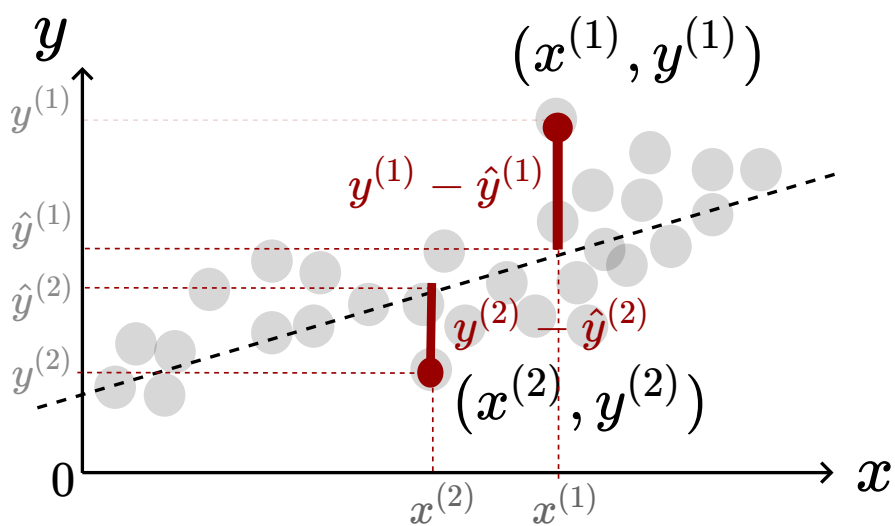
Loss function

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

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

square error **loss**
(a.k.a. **L2** loss)

$$L(y, \hat{y}) \triangleq (y - \hat{y})^2$$



Loss function

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

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

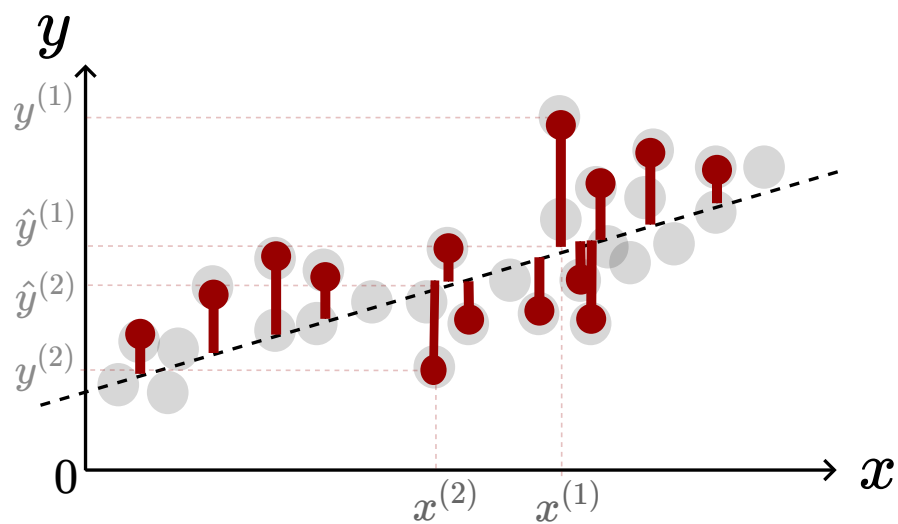
linear least squares

cost function

$$J(w) = \frac{1}{2} \sum_n \left(y^{(n)} - w^T x^{(n)} \right)^2$$

mean squared error

$$\frac{1}{N} J(w)$$



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

versus

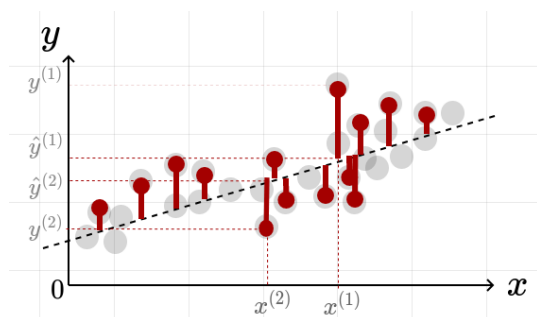
for the whole dataset

for future convenience

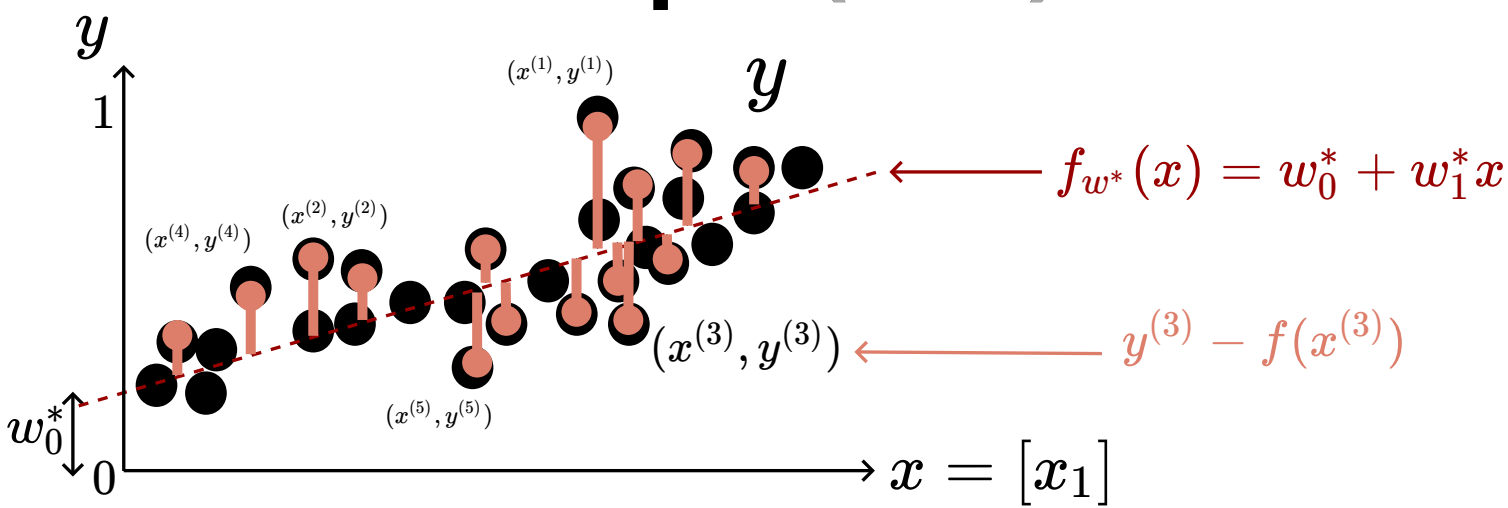
linear least squares **cost function**

$$J(w) = \frac{1}{2} \sum_n \left(y^{(n)} - w^T x^{(n)} \right)^2$$

$$w^* = \arg\min_w J(w)$$



Example (D=1)

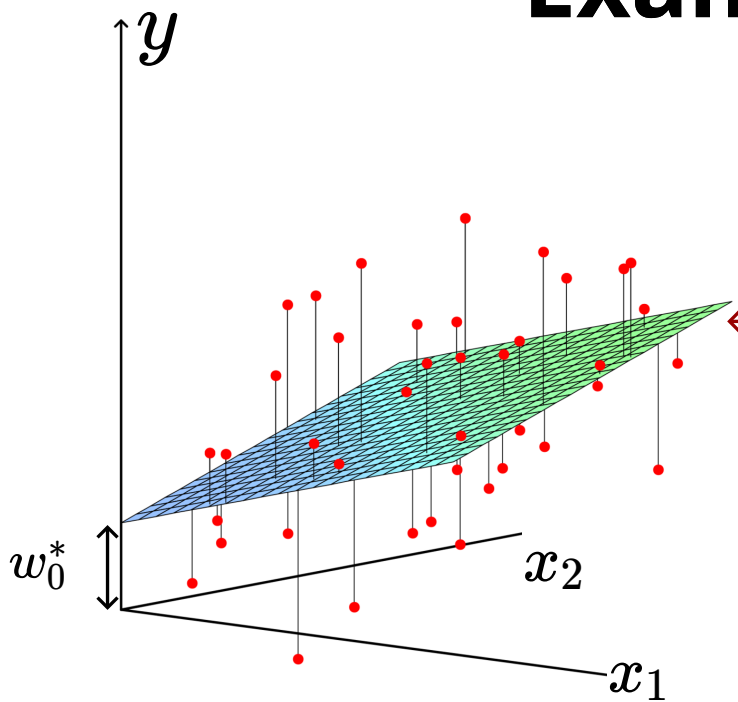


Linear Least Squares

cost: $$J(w) = \sum_n \frac{1}{2}\left(y^{(n)} - w^T x^{(n)}\right)^2$$

solution: $$w^* = \arg\min_w \sum_n \frac{1}{2}\left(y^{(n)} - w^T x^{(n)}\right)^2$$

Example (D=2)



Linear Least Squares solution

$$\longleftarrow \quad f_{w^*}(x) = w_0^* + w_1^* x_1 + w_2^* x_2$$

$$w^* = \arg\min_w \sum_n \frac{1}{2} \left(y^{(n)} - w^T x^{(n)} \right)^2$$

Minimizing the cost

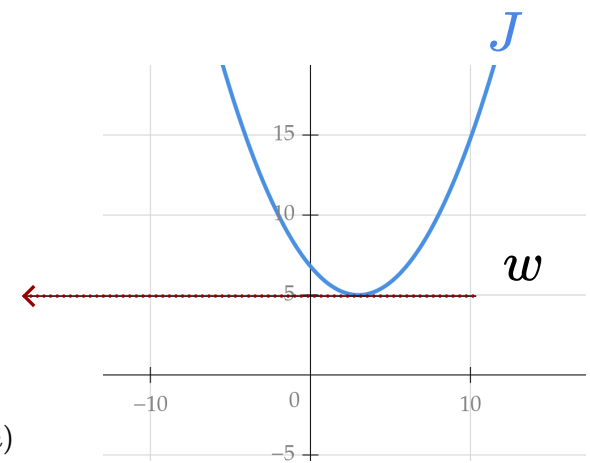
Simple case: D = 1

model $f_w(x) = wx$

both scalar

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

derivative $\frac{\mathrm{d}J}{\mathrm{d}w} = \sum_n x^{(n)}(wx^{(n)} - y^{(n)})$

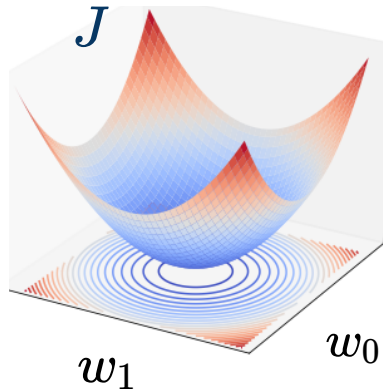


setting the derivative to zero ➡ $w^* = \frac{\sum_n x^{(n)} y^{(n)}}{\sum_n x^{(n)} x^{(n)}}$

global minimum because cost is smooth and *convex*

Minimizing the cost

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



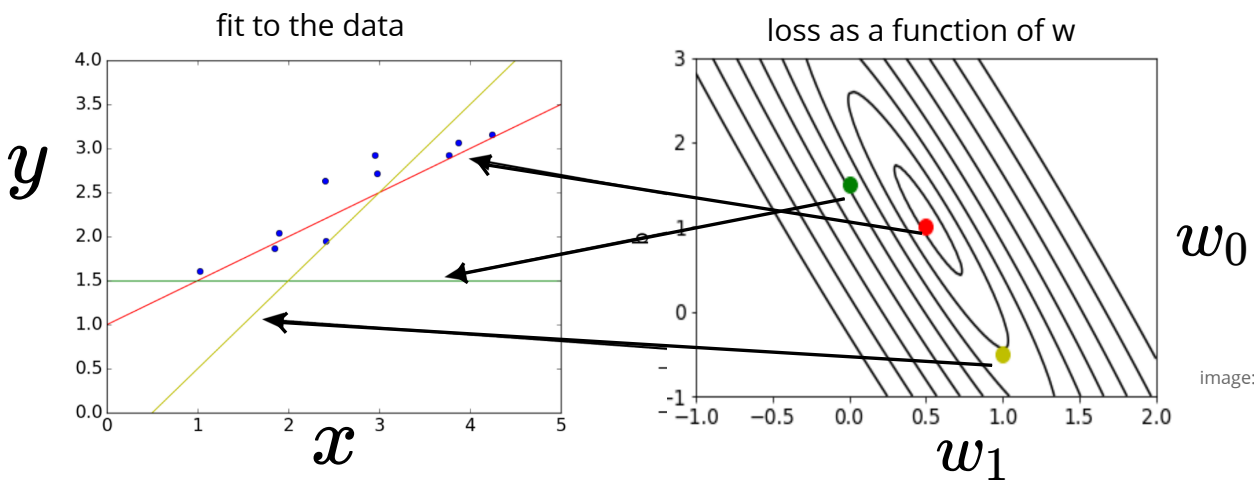


image: Grosse, Farahmand, Carrasquilla

The objective is a smooth function of w

Find minimum by setting partial derivatives to zero

image: Grosse, Farahmand, Carrasquilla

Gradient

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

partial derivatives instead of derivative

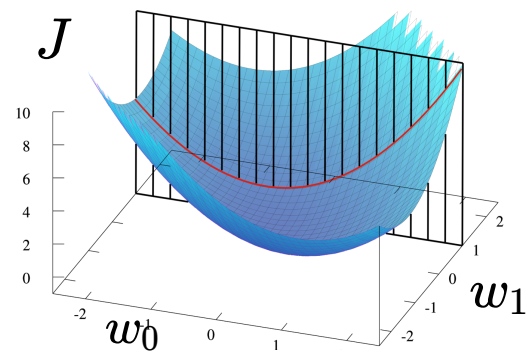
= derivative when other vars. are fixed

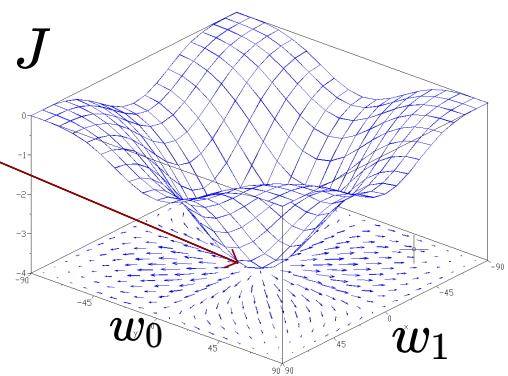
$$\frac{\partial}{\partial w_1} J(w_0, w_1) \triangleq \lim_{\epsilon \to 0} \frac{J(w_0, w_1+\epsilon) - J(w_0, w_1)}{\epsilon}$$

critical point: all partial derivatives are zero

gradient: vector of all partial derivatives

$$\nabla J(w) = [\frac{\partial}{\partial w_1} J(w), \cdots \frac{\partial}{\partial w_D} J(w)]^T$$





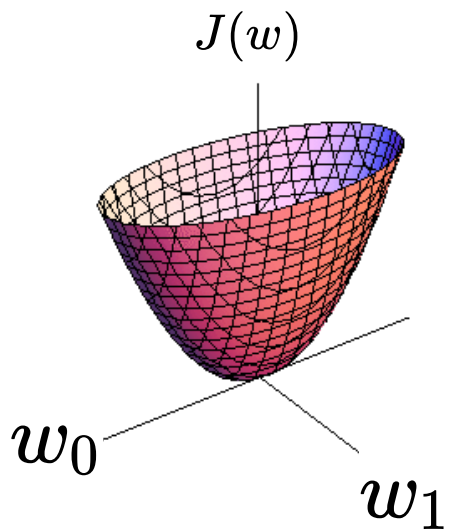
Finding w (any D)

$$f_w(x) = w^T x$$

setting $\frac{\partial}{\partial w_i} J = 0$

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

using **chain rule**: $\frac{\partial J}{\partial w_i} = \frac{\mathrm{d}J}{\mathrm{d}f_w} \frac{\partial f_w}{\partial w_i}$



cost is a smooth and convex function of w

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

D equations with D unknowns

Matrix form

D is in fact dimensions of the input +1 due to the simplification and adding the bias/intercept term

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

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

$$\hat{y}^{(1)} = w_0 + x_1^{(1)} w_1 + x_2^{(1)} w_2 + \cdots + x_D^{(1)} w_D$$

$$\hat{Y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(N)} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix} \begin{bmatrix} w^{(0)} \\ w^{(1)} \\ w^{(2)} \\ \vdots \\ w^{(D)} \end{bmatrix}$$

Matrix form

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

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

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

Linear least squares

$$J(w) = \frac{1}{2}||y - Xw||^2 = \frac{1}{2}(y - Xw)^T(y - Xw)$$

squared L2 norm of the **residual** vector

Normal equation

system of D linear equations

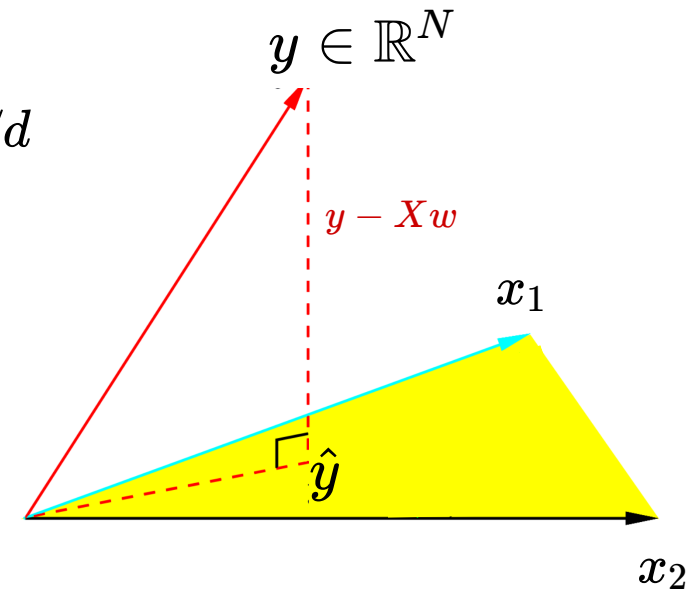
$$\sum_n (y^{(n)} - w^T x^{(n)}) x_d^{(n)} = 0 \quad \forall d$$

matrix form (using the design matrix)

each row enforces one of D equations

$D \times N$ $N \times 1$

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



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!

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

$$X^T Xw = X^T y$$

pseudo-inverse of X

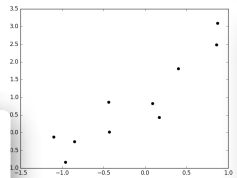
$$w^* = (X^T X)^{-1} X^T y$$

$D \times D$ $D \times N$ $N \times 1$

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

projection matrix into column space of **X**

Example



$$w^* = (X^T X)^{-1} X^T y$$

```
X.T.dot(X)
>> array([[ 10.    ,  -1.39  ],
          [ -1.39  ,   4.9261]])
np.linalg.inv(X.T.dot(X))
>> array([[ 0.10408228,  0.02936895],
          [ 0.02936895,  0.2112874 ]])

w = np.linalg.inv(X.T.dot(X)).dot(X.T.dot(Y))
array([[ 1.05881341],
       [ 1.61016842]])
```

```
X=array([[ 0.86],
         [ 0.09],
         [-0.85],
         [ 0.87],
         [-0.44],
         [-0.43],
         [-1.1 ],
         [ 0.4 ],
         [-0.96],
         [ 0.17]])
```

```
Y=array([[ 2.49],
         [ 0.83],
         [-0.25],
         [ 3.1 ],
         [ 0.87],
         [ 0.02],
         [-0.12],
         [ 1.81],
         [-0.83],
         [ 0.43]])
```

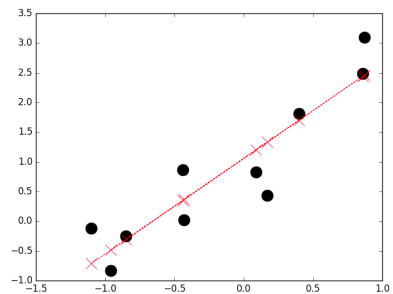
```
X = np.hstack((np.ones((X.shape[0],1)),X))

array([[ 1.  ,  0.86],
       [ 1.  ,  0.09],
       [ 1.  , -0.85],
       [ 1.  ,  0.87],
       [ 1.  , -0.44],
       [ 1.  , -0.43],
       [ 1.  , -1.1 ],
       [ 1.  ,  0.4 ],
       [ 1.  , -0.96],
       [ 1.  ,  0.17]])
```

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

```
yh = X.dot(w)
>>  array([[ 2.44355825],
           [ 1.20372857],
           [-0.30982974],
           [ 2.45965993],
           [ 0.35033931],
           [ 0.36644099],
           [-0.71237185],
           [ 1.70288078],
           [-0.48694827],
           [ 1.33254204]])
```

$$y = 1.06 + 1.61x$$



Todo: try this at home.

Time complexity

$$w^* = (X^T X)^{-1} X^T y$$

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

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

$N \times 1$

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

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

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

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

Multiple targets

instead of $y \in \mathbb{R}^N$ we have $Y \in \mathbb{R}^{N \times D'}$

a different weight vectors for each target

each column of Y is associated with a column of W

$$\underset{N \times D}{\hat{Y}} = \underset{N \times D}{X} \underset{D \times D'}{W}$$

$$\underset{D \times D}{W^*} = \underset{D \times D}{(X^T X)^{-1}} \underset{D \times N}{X^T} \underset{N \times D'}{Y}$$

```
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 $w^* = (\Phi^T \Phi)^{-1} \Phi^T y$

replacing X with Φ

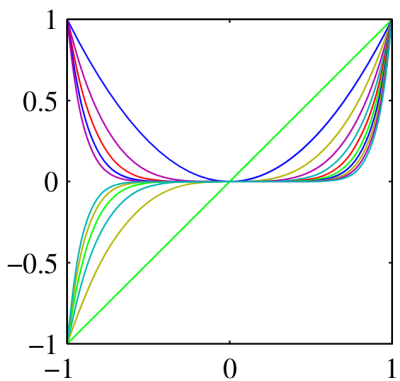
a (nonlinear) feature

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

one instance

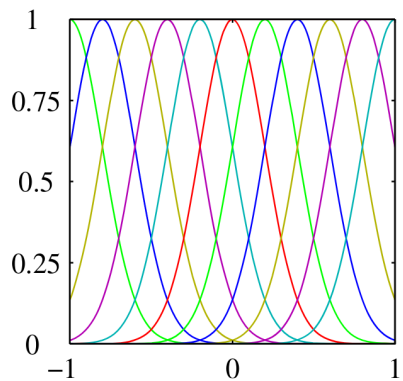
Nonlinear basis functions

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



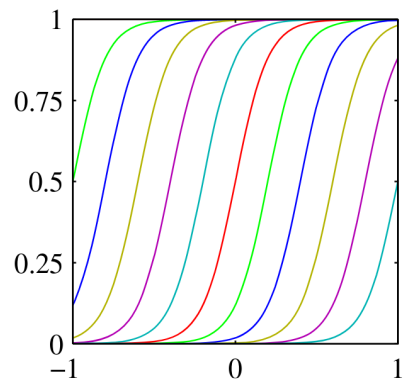
polynomial bases

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



Gaussian bases

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



Sigmoid bases

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

Nonlinear basis: example

```
X=array([[ 1.  ,  0.86],
         [ 1.  ,  0.09],
         [ 1.  , -0.85],
         [ 1.  ,  0.87],
         [ 1.  , -0.44],
         [ 1.  , -0.43],
         [ 1.  , -1.1 ],
         [ 1.  ,  0.4 ],
         [ 1.  , -0.96],
         [ 1.  ,  0.17]])


X = np.hstack((X, (X[:,1]**2)[:,None]))


array([[ 1.  ,  0.86  ,  0.7396],
       [ 1.  ,  0.09  ,  0.0081],
       [ 1.  , -0.85  ,  0.7225],
       [ 1.  ,  0.87  ,  0.7569],
       [ 1.  , -0.44  ,  0.1936],
       [ 1.  , -0.43  ,  0.1849],
       [ 1.  , -1.1   ,  1.21  ],
       [ 1.  ,  0.4   ,  0.16  ],
       [ 1.  , -0.96  ,  0.9216],
       [ 1.  ,  0.17  ,  0.0289]])
```

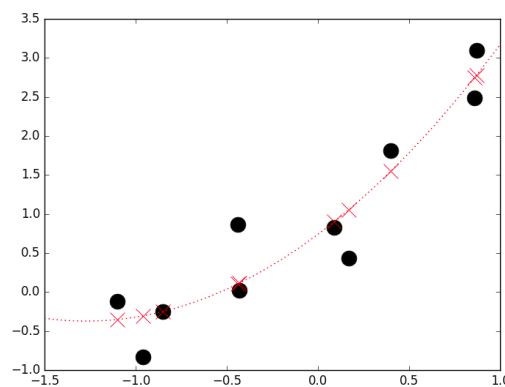
```
Y=array([[ 2.49],
         [ 0.83],
         [-0.25],
         [ 3.1 ],
         [ 0.87],
         [ 0.02],
         [-0.12],
         [ 1.81],
         [-0.83],
         [ 0.43]])
```

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

array([[ 0.73960888],
       [ 1.74736998],
       [ 0.68670054]])
```

```
yh = X.dot(w)
>>  array([[ 2.75023077],
           [ 0.90243445],
           [-0.24951447],
           [ 2.77958439],
           [ 0.10371131],
           [ 0.11521071],
           [-0.35159045],
           [ 1.54842895],
           [-0.30500309],
           [ 1.05650742]])
```

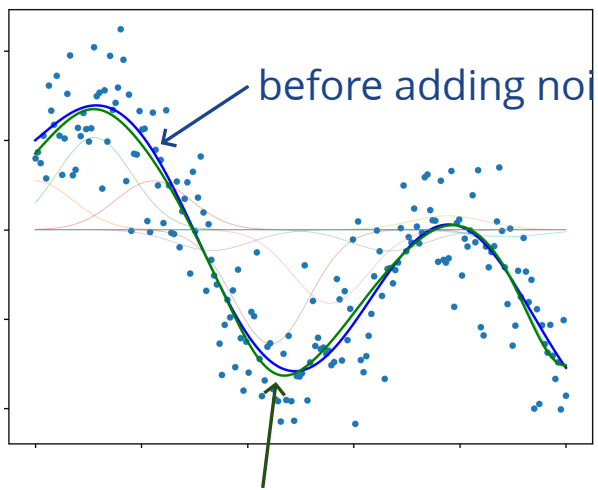
$y = 0.74 + 1.75x + 0.68x^2$



Todo: try this at home and add $x^3, x^4, \ldots$

Gaussian bases example

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



$$y^{(n)} = sin(x^{(n)}) + cos(\sqrt{|x^{(n)}|}) + \epsilon$$

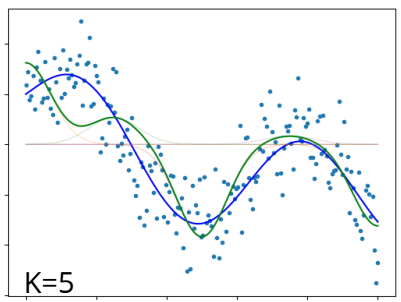
```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,10) #10 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

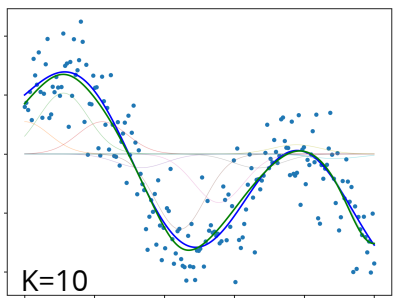
Todo: try this at home

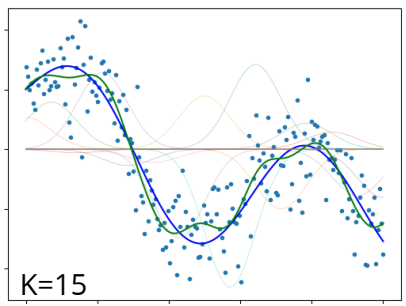
Gaussian bases example

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

$$y^{(n)} = sin(x^{(n)}) + cos(\sqrt{|x^{(n)}|}) + \epsilon$$





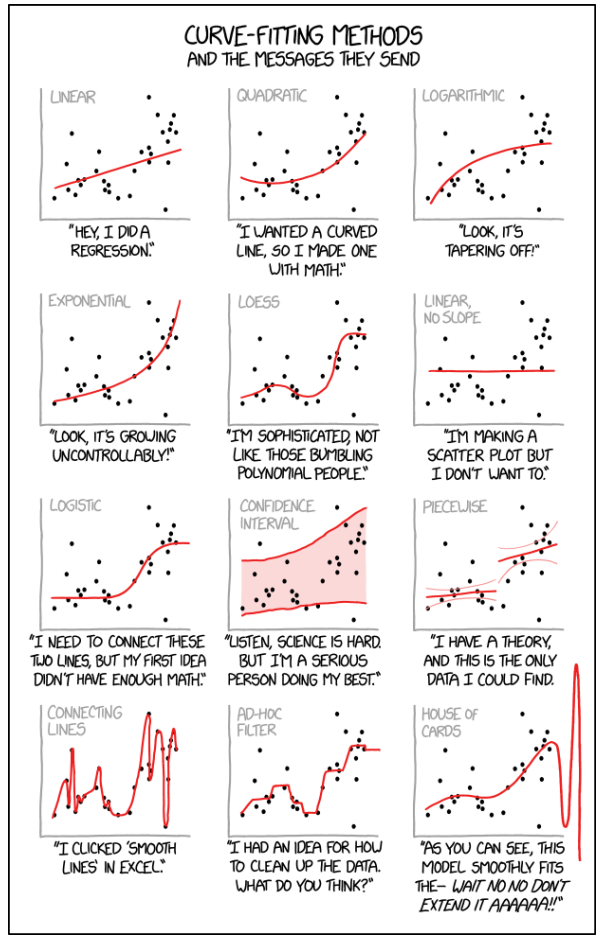


Todo: try this at home, change number of bases, see how the fit changes.

```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,K) #K Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

For fun!

https://www.explainxkcd.com/wiki/index.php/2048:_Curve-Fitting



Problematic settings

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

what if we have a **large dataset**?

- use stochastic gradient descent

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

columns of X 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 $\mathcal{O}(D^3 + ND^2)$ complexity
- gradient descent: future

we can build complex models:

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