WEEK - 2

• Multi-Variate Linear Regression

(n) = number of features

$x_j^{(i)}$ = value of feature j in the (i^{th}) training example

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in R^{n+1}$$

∴ Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n$$

$\uparrow$

$x_0 = 1$

↳ for convenience of notation

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in R^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in R^{n+1}$$

vector

∴ $$\boxed{h_\theta(x) = \theta^T X}$$

• Gradient descent for Multiple Variables

Hypothesis:

$$h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \dots$$

parameters: θ ← $n+1$ dimensional vector

cost function:

$$J(\theta_0, \theta_1 \dots \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right)^2$$

m = Training ex.
n = features

Gradient descent: $(n \geq 1)$

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots \theta_n)$$

until convergence

}

⇓ o/p

$$\theta_j := \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right) x_j^i$$

example : (for $n=2$)

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum^{m} (h_\theta(x^i) - y^i)\, x_0^i$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum^{m} (h_\theta(x^i) - y^i)\, x_1^i$$

$$\theta_2 := \theta_2 - \alpha \cdot \frac{1}{m} \cdot \sum^{m} (h_\theta(x^i) - y^i) \cdot x_2^i$$

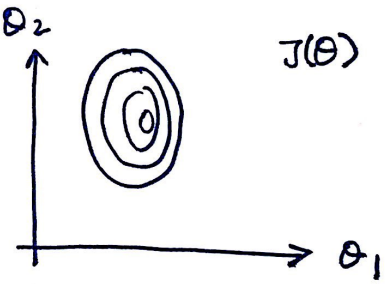
---

* **Feature Scaling :** To run gradient descent much faster }

Idea : Make sure features are on a similar scale

eg. x_1 = size (0–2000 ft^2)

x_2 = no. of bedrooms (0–1–5)



$$x_1 = \frac{\text{size}}{2000} \qquad 0 \le x_1 \le 1$$

$$x_2 = \frac{\text{no. of bedrooms}}{5} \qquad 0 \le x_2 \le 1$$

→ We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate down to the optimum when the variables are uneven ***

General Idea : Feature Scaling

Get every feature into approximately

$[-1 \leq x_i \leq 1]$ range

↙

This is subjective, but the idea is they should be close enough

Mean Normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$)

Eg. $x_1 = \dfrac{\text{size} - 1000}{2000}$ (μ_i), , $x_2 = \dfrac{\#\text{bedrooms} - 2}{5}$

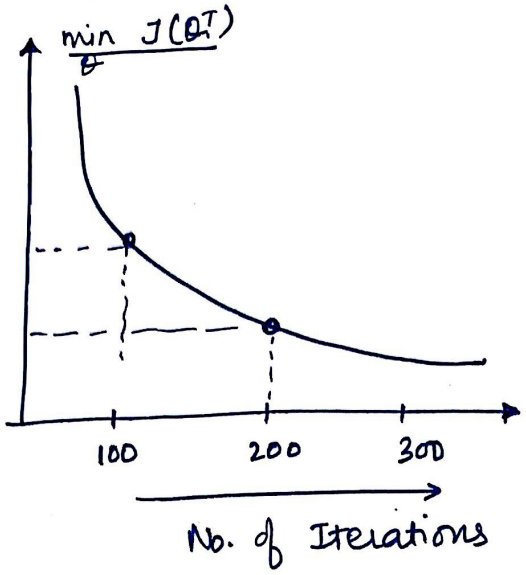
or

$x_1 \leftarrow \dfrac{x_1 - \mu_i}{s_i}$

μ_i → Avg. value of x in training set

s_i → range (max − min) of feature set/training ** or standard deviation

- Gradient descent : **Learning Rate** α



* $J(\theta^T)$ should dec. after every iteration

← * Look at such plots to check how α is working

* Example autodecet :

Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration

* for sufficiently small α, $J(\theta)$ should decrease on every iteration

* But if α is too small, gradient descent can be slow to converge

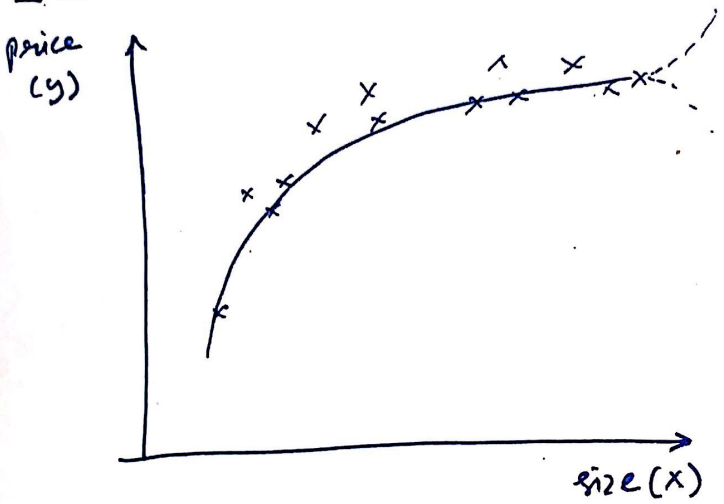
- Features and Polynomial regression:

eg.

$h_\theta(x) = \theta_0 + \underbrace{\theta_1 \times \text{frontage}} + \underbrace{\theta_2 \times \text{depth}}$

$x = \underline{\text{Area}} = \text{frontage} \times \text{depth}$

$h_\theta(x) = \theta_0 + \theta_1 x$ ← land area



$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$

$\Rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

$\therefore h_\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2 + \theta_3 x_3^3$

$= \theta_0 + \theta_1 (\text{size}) + \theta_2 (\text{size})^2 + \theta_3 (\text{size})^3$

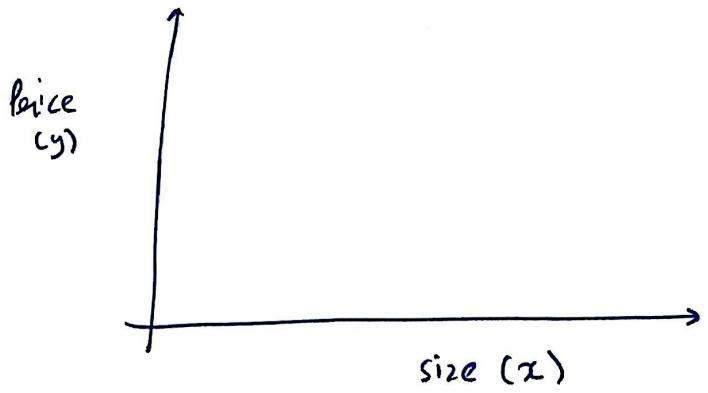
feature scaling becomes very important

$\text{size} \rightarrow 1\text{-}1000$

$\text{size}^2 \rightarrow 1 - 10^6$

$\text{size}^3 \rightarrow 1 - 10^9$

choices of feature :



$$h_\theta(x) = \theta_0 + \theta_1 \underbrace{(\text{size})}_{x_1} + \theta_2 \underbrace{\sqrt{\text{size}}}_{x_2}$$

eg: if size range → 1 – 1000

$x_1 \leftarrow$ size $= \frac{\text{size}}{1000}$

$x_2 \leftarrow \sqrt{\text{size}} = \frac{\sqrt{\text{size}}}{32}$

} feature scaling without mean normalization

- **Computing Parameters Analytically :-**

(Normal Equation): solve for θ analytically

Intuition: if 1D ($\theta \in \mathbb{R}$)

$$J(\theta) = a\theta^2 + b\theta + c$$

$$\frac{\partial J(\theta)}{\partial \theta} = 0 \quad \} \text{ solve for } \theta$$

$\theta \in \mathbb{R}^{n+1} \rightarrow$ n-dimensional

$$J(\theta^T) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

---

eg:

$m = 4$

x_1 size (ft^2)	No. of bedrooms x_2	Floors x_3	Age of home x_4	y Price ($)'000
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

Design Matrix

X_0

$m \times (n+1)$

$m \times 1$

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

→ pinv (x'*x)*x'*y

↳ octave.

$$x^{(i)} = \begin{bmatrix} x_0^i \\ x_1^i \\ x_2^i \\ x_3^i \\ x \\ \vdots \\ x_n^i \end{bmatrix} \quad x \in R^{n+1}$$

$$X = \begin{bmatrix} \text{---} (x^{(1)})^T \text{---} \\ \text{---} (x^2)^T \text{---} \\ \\ \\ \text{---} x^{n\,T} \text{---} \end{bmatrix}$$

↓ Design Matrix

eg. $x^{(i)} = \begin{bmatrix} 1 \\ x_1^i \end{bmatrix}$ $\quad X = \begin{bmatrix} 1 & x_1^1 \\ 1 & x_2^1 \\ \vdots & \\ 1 & x_m^{m(i)} \end{bmatrix}$ $\quad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$

* Feature scaling is not required in the normal-equation method.

choice between Gradient descent & Normal Equation

'm' training examples

'n' features

Gradient	Normal eqn
→ Need to choose α. (−)	→ No need to choose α
→ Needs many iterations (−)	→ Don't need to iterate.
→ Work well even when n is large (+)	→ need to compute $(X^TX)^{-1}$
	→ slow if n is very large
$O(kn^2)$	$O(n^3)$

Non- Equation Noninvertibility

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

- what if $X^T X$ is non-invertible (singular / generate)

{ pinv → psuedo inverse
  inv → inverse }

Prob. Reasons

- Redundant features (linear dependence)

eg. x_1 = size in ft^2

x_2 = size in m^2

- Too many features ($m \leq n$)

  - delete some features, or use regularization.

MATLAB/OCTAVE

TUTORIAL

→ MATLAB / OCTAVE TUTORIAL :

①

→ disp(a)

→ disp (sprintf ('2 decimals : %0.2f', a))

→ disp (sprintf ('6 decimals : %0.6f', a))

} string

②

A = [1 2 ; 3 4 ; 5 6]

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3\times 2}$$ matrix

A = [1 2 3] row vector

V = [1 ; 2 ; 3] or V = 1 : 0.1 : 2

step, start, stop

③

C = 2 * ones (2,3)

$$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

Z = zeros (1,3)

$$\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

rand (3,3) ← range (0,1) → randn (1,3)

$$\begin{bmatrix} 0.08 & - & - \\ - & - & - \\ - & - & - \end{bmatrix}$$

[-1.44 1.55 _]

Identity matrix:

`I = eye(3)` → $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- ✓ `help eye`
- ✓ `hist` → histogram function

Moving data around

* `size(A)` → `[3 2]` (3 = rows, 2 = columns)
* `size(A,1)` → 3
* `size(A,2)` → 2
* `length(A)` → (3) highest dimension

`pwd` → gives the current direct.

`cd 'C:\Users\Naveen.mandolia\Desktop'`

↓

change directory

* load features X.dat
* load ('featuresX.dat') } filename
* load ('PriceY.dat')

* who → shows me variables in memory currently
* whos → gives detail view

* clear ——— } clears variable

* v = PriceY(1:10)
* save hello.txt var -ascii % save as ASCII format

* A(3,2) : element of Matrix

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

A(2,:) ⇒ [3 4]

A(:,2) ⇒ $\begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$

A([1 3],:) ⇒ $\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}$

can also be used for assignments.

* A = [A, [100; 101; 102]] → append another column vector to right.

$$\begin{bmatrix} 1 & 2 & 100 \\ 3 & 4 & 101 \\ 5 & 6 & 102 \end{bmatrix}$$

A = [1 2 ; 3 4 ; 5 6]

B = [7 8 ; 9 10 ; 11 12]

C = [A B] → Treats as column

[A, B]

$$\begin{bmatrix} 1 & 2 & 7 & 8 \\ 3 & 4 & 9 & 10 \\ 5 & 6 & 11 & 12 \end{bmatrix}$$

C = [A ; B] → Treats as row

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$$

Computing on data:

A = [1 2 ; 3 4 ; 5 6]

B = [11 12 ; 13 14 ; 15 16]

C = [11 ; 2 2]

operations:

① A*B [3x2].[3x2]

② A.*B → element wise multiplication

(3) A.^2 ↳ element wise square.

(4) 1./A ↳ element wise invert

(5) abs(V)
↳ absolute value element wise

(6) V+ ones(length(V), 1) or (V+1)
↳ increaments each element by 1

(7) A'
↳ gives transpose

(8) exp(v) –
~~abs(v)~~
log(v) –

(9) a = [1 15 2 0.5]

max(a) = 15

[val, ind] = max(a)

val = 15
ind = 2

(10) a < 3 ⟶ does element wise operation check.

(11) find(a<3) : [r, c] = find(A>=7)

↳ returns the index, of values which are less than 3

r = 1 2 3 c = 1 2 3

(12) A = magic(3)

↳ creates a magic matrix of 3×3

↳ sum = same

(13) sum(a)

↳ sum of all the elements.

(14) prod(a) → prod. of all elements.

floor(a) →

ceil(a) →

(15) rand(3)

↓

3×3

[- - -]
[- - -]
[- - -]

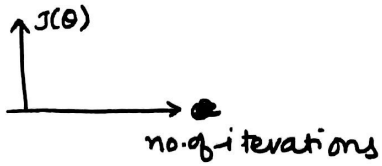
(16) max(A, [], 1/2)

1 → column

2 → row

→ Each matrix operations are column first.

Plotting data :




```
t = [0 : 0.01 : 0.98]

y1 = sin(2*pi*4*t);
y2 = cos(2*pi*4*t);

plot(t, y1)
```

`plot(x, y)` **

`hold on;` → holds on the previous graph.

`plot(t, y2, 'r');` → plots in red color

```
xlabel('time');
ylabel('value');
legend('sin', 'cos');
title('my plot');
```

~~cd print~~ `print -dpng 'myplot.png'`

↓ saves in directory.

```
close;
```

```
figure(1); plot(t, y1);
figure(2); plot(t, y2);
subplot(1,2,1); % Divides plot a 1X2 grid, access first element
```

`clf;` → clears a figure/plot

◆ : control statements : for, while, if

① v= zeros(10,1)

FOR

```
for  i= 1:10,
     v(i)= 2^i ;
end;
```

② while

```
i=1
while i<=5,
   v(i) = 100
    i = 1+1;
end;
```

③

```
if  i==6,
    disp(i);
elseif  i==8,
    disp(i);
else
    disp(i);
end;
```

◆ Functions:

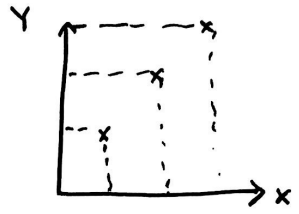
→
```
function y = squareThisNumber(x)
y = x^2
```

→
```
function [y1, y2] = sqandcube(x1, x2)
  y1 = x1^2
  y2 = x2^3
```
* returns multiple values.

→ design matrix

$X = [1 \;\; 1 \;;\; 1 \;\; 2 \;;\; 1 \;\; 3]$

$y = [1 ; 2 ; 3]$



Goal: Define a fn. to compute the cost fn. J(θ)

theta = [0; 1]

Soln: fn.

→ function J = costFunctionJ(X, y, theta)

```
m = size(X,1);
predictions = X*theta;
squareerror = (predictions-y).^2;
J = 1/2m * (sum(squareerrors));
```

Vectorization

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \Rightarrow \theta^T x$$

$$\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \quad x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$$

Unvectorized implementation

```
pred = 0
for j = 1:n+1
  predict = pred +
            theta * x

end;
```

Vectorized implementation

```
prediction = theta' * x
```

Vectorized implementation of GRADIENT DESCENT

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j}\left[\frac{1}{2m}\cdot \sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2\right]$$

$$\theta := \theta - \alpha\, \delta$$

θ : $(n+1)$ dimensional vector ($\mathbb{R}^{n+1}$)

α : $\mathbb{R}$

δ : $\mathbb{R}^{n+1}$

$$\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \quad \delta = \begin{bmatrix} \delta_0 \\ \vdots \\ \delta_n \end{bmatrix}$$