

Week 02

- Linear regression with multiple variables
 - Multivariate Linear Regression
 - Computing Parameters Analytically
 - Octav/ Matlab usage
-

I. Linear regression with multiple variables

1. Multivariate Linear Regression

a. Multiple features (variables)

- Linear regression with multiple variables = "multivariate linear regression".

Size (x_1)	Nb of bedrooms (x_2)	Nb of floors (x_3)	Age (x_4)	Price (y)
2100	2	1	45	460

...

n = number of features

$x^{(i)}$ = input features of i^{th} training example

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

m = number of training examples

$h_{\theta}(x) = \sum_{k=1}^n \theta_k x_k = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n$ and $x_0^i = 1$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}; \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}; h_{\theta}(x) = \theta^T x = [\theta_0 \dots \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

hence, the multivariate regression.

b. Gradient Descent for Multiple Variables

- Hypothesis: $h_{\theta}(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n$
- Parameters: $\theta_0, \theta_1, \dots, \theta_n = \theta$
- Cost function: $J(\theta_0, \theta_1, \dots, \theta_n) = J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h - \theta(x^{(i)}) - y^{(i)})^2$
- Gradient descent:

Repeat:

$$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$$

where $j = 0..n$

Previously number of features is $n = 1$; so there's only θ_0 and θ_1 ;

- Now, $n \geq 1$; so θ_j takes values $j = 0..n$;

c. Gradient Descent

i. Feature Scaling

- Speed up gradient descent, how? → Put input values in the same range
 - θ will descend:
 - quickly on small ranges
 - slowly on large ranges.
 - can oscillate and change range.
- The way to prevent this is to modify the ranges. Example: $-1 \leq x_i \leq 1$
- To get all input variables into roughly one of these ranges?
 - Two techniques to help with this are feature scaling and mean normalization
 - Feature scaling = dividing the input values by the range ($max - min$) of the input variable so get values close to 1.
 - Mean normalization = subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero.

Both of these techniques are implemented as $\rightarrow x_i = (x_i - \mu_i) / s_i$

μ_i = the average of all the values for feature (i)

s_i = the range of values (max - min) = the standard deviation

Example: if x_i represents housing prices with a range of 100 to 2000 and a mean value of 1000, then,

$$x_i = \frac{price - 1000}{1900}$$

ii. Learning rate

- Plot cost function $J(\theta)$
 - Declare convergence if $J(\theta)$ goes from $+\infty \rightarrow 0$ and decreases by less than $E \leq 10^{-3}$ in one iteration.
 - Declare divergence if $J(\theta)$ goes from $+\infty \rightarrow 0 \rightarrow +\infty$ or get to oscillate.
- If $J(\theta)$ ever increases, then you probably need to decrease α .
- If α is too small: slow convergence.

- If α is too large: may not decrease on every iteration and thus may not converge.

d. Features and Polynomial Regression

- We can improve our features and the form of our hypothesis function in a couple different ways.
- We can combine multiple features into one: multiply them $x_3 = x_1 * x_2$
→ Our hypothesis function need not be linear

change the behavior or curve of our hypothesis function : go polynomial
create additional features based on x_1 like $x_2 = x_1^2$, etc.
this way then feature scaling becomes very important.

2. Computational parameters analytically

a. Normal equation

- Gradient descent gives one way of minimizing J
- Another way to minimize J is to use Normal Equation:
 - minimize J by explicitly taking its derivatives with respect to the θ_j 's, and setting them to zero.
→ find the optimum θ without iteration.
- Normal equation formula is $\theta = (X^T X)^{-1} X^T y$
- Example: $m = 2$

x_0	Size (x_1)	Nb of bedrooms (x_2)	Nb of floors (x_3)	Age (x_4)	Price (y)
1	2100	3	2	45	460
1	1416	2	1	35	232

So matrices are as follow $X = \begin{bmatrix} 1 & 2100 & 3 & 2 & 45 \\ 1 & 1416 & 2 & 1 & 35 \end{bmatrix}; y = \begin{bmatrix} 460 \\ 232 \end{bmatrix}$

/!\ /!\ /!\ there is no need to do feature scaling with the normal equation.

Comparison Gradient descent vs Normal equation

Gradient Descent	Normal Equation
Need to choose α	No need to choose α
Needs many iterations	No need to iterate

Gradient Descent	Normal Equation
$O(kn^2)$	$O(n^3)$
Works well when n is large	Slow if n is very large

n is number of features

b. Normal Equation Noninvertibility

If $\mathbf{X}^T \mathbf{X}$ is noninvertible, the common causes might be having :

- Redundant features, closes or related.
- Too many features so use regularization techniques.

II. Octave commands

~ #for not

1 && 1

xor(1, 1)

pi

disp(pi)

sprintf("ahhhsq %0.2f", pi)

A = [1 2; 2 3]

ones(x,y)

zeros(x,y)

eye(x)

size(m)

length(v)

rand(x,y)

randn(x,y)

hist(m)

pwd

load file_name

load('filename')

who

whos

clear x

clear

z = x(1:10)

save f.txt z

save f.txt z -ascii

m(x,:)

m([x,y],:)

m = [x; y; z]

m = [m, [a, b, c]]

m = [m n]

m = [m; n]

m*n

m.*n # element by element

m.^n

m./n

log(v)

exp(v)

abs(v)

-v

v' # transpose

max(v)

[val, i] = max(v)

v < 3

find(a < 3)

magic(3)

sum(v)

prod(v)

floor(v)

ceil(v)

pinv(m)

plot(x, y, 'color')

hold on

xlabel("text")

title("text")

print -dpng file.png

close

figure(1); plot(x, y)

subplot(1, 2, 1)

axis([0.5 1 -1 1])

clf;

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

for i=indices, disp(i); end;

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

while true, v(i) = 900; i = i+1; if i == 6, break; end; end;

if true, break; elseif i==2, break; else break;; end;

function y = sq(x) y = x^2;

addpath(path)

cd path

function [x, y] op(a) x=a^3; y=a^2