Linear Regression w/ Multiple Features: $x_1, x_2, x_3, \ldots x_j$

Multiple features (variables)

Size in feet² x_1	Number of bedrooms x_2	Number of floors x_3	Age of home in years x_4	Price ($) in $1000's
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$i=2$

$j=1\ldots4$

$n=4$

x_j = j^{th} feature

n = number of features

$\vec{x}^{(i)}$ = features of ith training example

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

$\vec{x}^{(2)} = [1416\ 3\ 2\ 40]$

$x_3^{(2)} = 2$

→ What if the price of our house depended on all these? Not just one variable.

$\vec{x}^{(2)}$ = x is a vector (list of numbers)

↓

row vector.

$x_j^{(i)}$ → row, → column

Theorem

Previously = $f_{w,b}(x) = wx + b$

Multiple features = $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + \ldots w_nx_n + b$

$f_{\vec{w},b}(\vec{x}) = 0.1 \times x_1 + 0.2 \times x_2 + \ldots 80$

x_1 ↓ size; x_2 ↓ #bedrooms; 80 ↳ base price

$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

↳ dot product

General Formula for Multiple Linear Regression

Note: Not multivariate regression!!!!

Vectorization:

→ Makes code shorter.

→ Makes program more efficient (run faster).

→ Numpy uses paralel hardware, that's why it's more efficient than a regular for loop.

Example

```
w = np.array([1,2,3,4])
b = 4
x = np.array([5,6,7,8])
```

Without vectorization

$$f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} w_j x_j\right) + b$$

$\sum_{j=1}^{n}$ → $j=1\ldots n$ (1, 2, 3)

range(0,n) → $j=0\ldots n-1$

```
f = 0        range(n)
for j in range(0,n):
    f = f + w[j] * x[j]
f = f + b
```

Vectorization

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

```
f = np.dot(w,x) + b
```

Without vectorization

```
for j in range(0,16):
    f = f + w[j] * x[j]
```

t_0: f + w[0] * x[0]

t_1: f + w[1] * x[1]

...

t_{15}: f + w[15] * x[15]

Vectorization

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

t_0: w[0] w[1] ... w[15]

in parallel: * * ... *

t_1: x[0] x[1] ... x[15]

w[0]*x[0] + w[1]*x[1] + ... + w[15]*x[15]

Made with Goodnotes

Gradient descent

One feature

repeat {

$$w = w - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)x^{(i)}}_{\frac{\partial}{\partial w}J(w,b)}$$

$$b = b - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$$

simultaneously update w, b

}

n features ($n \geq 2$)

repeat {

$j=1$

$$w_1 = w_1 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_1^{(i)}}_{\frac{\partial}{\partial w_1}J(\vec{w},b)}$$

$\vdots$

$j=n$

$$w_n = w_n - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_n^{(i)}$$

$$b = b - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

simultaneously update w_j (for $j = 1, \cdots, n$) and b

}

Normal Equation:

- Only for linear regression.
  Solve for w,b without iterations

- Doesn't work for other algorithms.
  Very slow when too many parameters

Made with Goodnotes

Feature Scaling:

Example

size (300 − 2000)

bedrooms (0 − 5)

$$price = w_1 x_1 + w_2 x_2 + b$$

Note: Let's say $x_1 = 2000$ $x_2 = 5$ price = 500k

How to choose $w = ?$

Ex 1

$w_1 = 50$ $w_2 = 0.1$ $b = 50$

$$price = 50 \times 2000 + 0.1 \times 5 + 50$$
$$= 100{,}050.5k = 100\text{ mil}$$

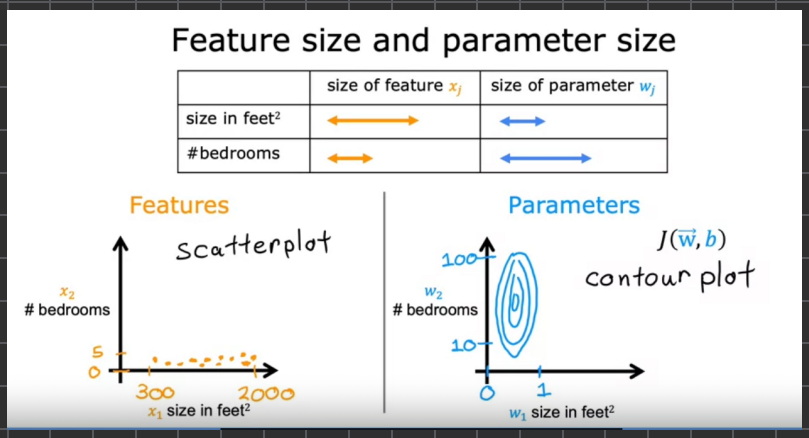
way too high!

Ex 2.

$w_1 = 0.1$ $w_2 = 50$ $b = 50$

$$price = 0.1 \times 2000 + 50 \times 5 + 50$$
$$= 500k$$

more reasonable



Small change in x_2 results in large change in cost.

Change in x_1 doesn't drastically affect cost.

How do we scale features? so GD can run faster.

OP1 Divide by Maximum.

$$300 \le x_1 \le 2000 \quad /2000$$

2000

$$0.15 \le x_1, \text{scaled} \le 1$$

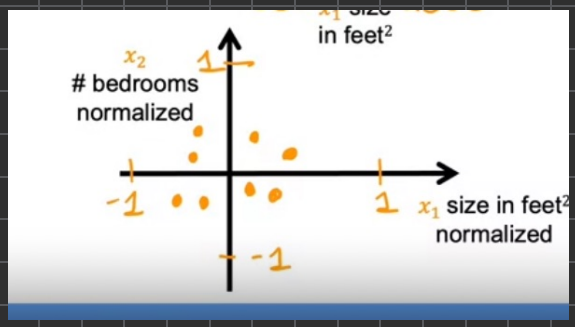
$$0 \le x_2 \le 5$$

$$0 \le x_2, \text{scaled} \le 1$$

OP2 Mean Normalization:

Assume μ_1 (mu) = 600

$$x_1 = \frac{x_1 - 600}{2000 - 300} \longrightarrow -0.18 \le x_1 \le 0.82$$



OP3

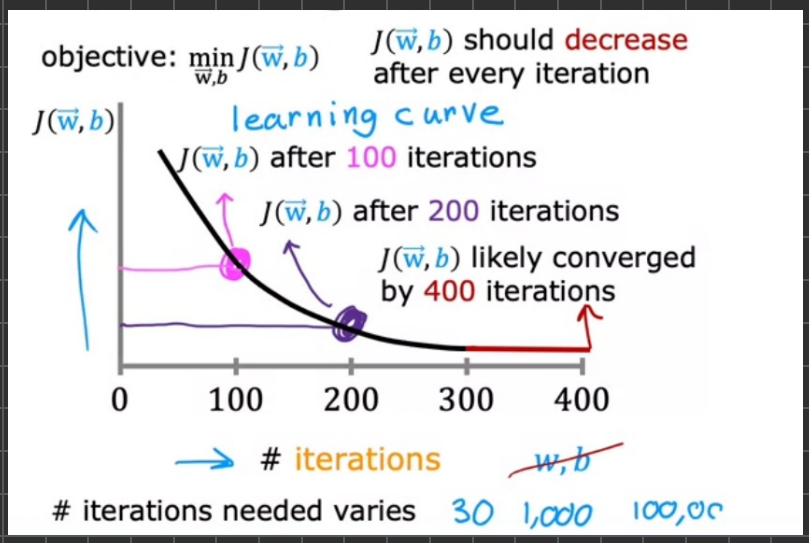
Z score Normalization.

Note: Generally, try to keep range close to $-1 \le x_j \le 1$

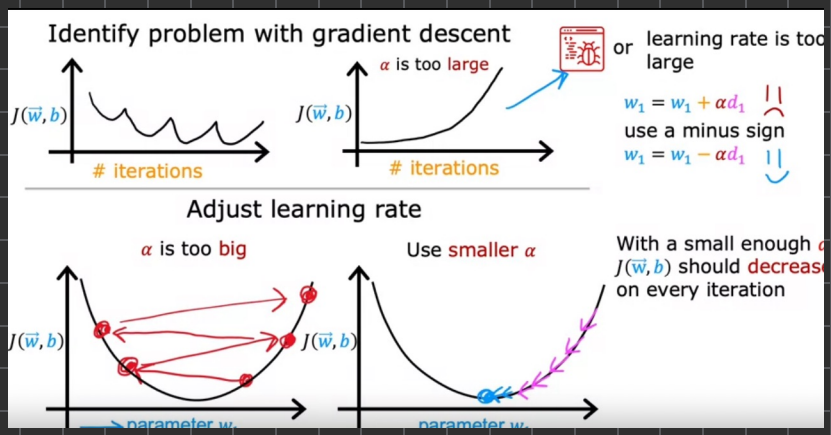
Can be close to these values. If not, scale the features.

Made with Goodnotes

Does GD converge?



Choosing Learning Rate:



Trial and error: 0.001, 0.01, 0.1, 1, 10

Made with Goodnotes

1. Evolution of Code Smells. (purely research)

2. Software Development Team Patterns ([redacted])

Good bad?

Git log

3. Github App that provides refactoring based on difference

↓

Can it be better?

suggestion via LLM with PR

Made with Goodnotes