6. Multiple features (variables)

Previously, we only had one feature. Our model looked something like this.

$f_{w,b}(x) = wx + b$, where x is only one feature.

What if we have multiple features as seen in image.
We call each features, x_1, x_2, x_3 and x_4.

Let x_j = j^{th} feature.

n = number of features.

$\vec{x}^{(i)}$ = features of i^{th} training example. (vector).

$\vec{x}^{(2)} = [1416 \quad 3 \quad 2 \quad 40]$ a row vector. This is our 2nd training example.

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

$x_3^{(2)} = 2$. in our above example.

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

Now, $f_{w,b}(x) = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$.
for n features.

for our new example, $f_{w,b} = 0.1x_1 + 4x_2 + 10x_3 + (-2)x_4 + 80$.

Here $x_1 \rightarrow$ size, $x_2 \rightarrow$ bedrooms, $x_3 \rightarrow$ floors, $x_4 \rightarrow$ years.

Now, for n features we will have.

$f_{w,b}(x) = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$.

Let our $\vec{w} = [w_1 \quad w_2 \quad w_3 \dots w_n]$ vector } their are parameters of our model.
b is a number.

$\vec{x} = [x_1 \quad x_2 \quad x_3 \dots x_n]$ $\rightarrow$ input vector.

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b.$$

$\vec{w} \cdot \vec{x}$ is the dot product.

We have our model in a compact form.

This model is called multiple Linear Regression model.
(not multivariate regression)

In our previous example, what will be value of
$x_1^{(4)}$ → 1st feature and 4th row.
→ 852

I. Vectorization.

Parameters and features.

$\vec{w} = [w_1 \quad w_2 \quad w_3]$ b is a number.

$\vec{x} = [x_1 \quad x_2 \quad x_3]$ $n = 3$.

In linear algebra: count starts from 1.

In python, indexing starts from 0.

- Without vectorization.

$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b.$$

In code, see python code.

If value of n is 100, or 10000.

for
$$f_{\vec{w},b}(\vec{x}) = \left(\sum_{j=1}^{n} w_j x_j\right) + b.$$
} we can implement this using for loop.

- Vectorization.

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b.$$
} we can implement the same using numpy.

when n is large, our code runs very fast.

numpy uses parallel hardware makes it much more faster.
very useful when n is very large.

numpy ⇒ f = np.dot(w, x) + b.

- Without vectorization.

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

to f + w(0) * x(0)

:

tis f + w(15) * x(15)

Vectorization.

np.dot(w,x)

to. it multiplies the two vectors in parallel.

th. Add all the 16 values at the same time using specialized hardware.

So, vectorization is really fast.
efficient on large datasets.

- Gradient Descent.

Let's say $\vec{w} = (w_1, w_2 \ldots w_{16})$ · b parameters.
$\vec{d} = (d_1, d_2 \ldots d_{16})$.

w = np.array([0.5, 1.3, ..., 3.4])
d = np.array([0.3, 0.2, ..., 0.4])

Compute; $w_j = w_j - 0.1 d_j$ for $j = 1 \ldots 16$.

Without vectorization

$w_1 = w_1 - 0.1 d_1$
:
$w_{16} = w_{16} - 0.1 d_{16}$.

for j in range(0,16):
  w[j] = w[j] - 0.1 * d[j]

with vectorization.

$\vec{w} = \vec{w} - 0.1 \vec{d}$.

code:
w = w - 0.1 * d.

} efficient arrays.

II. Gradient Descent for multiple linear regression.

Previous notation	Vector notation.
Parameters. $w_1, w_2, \dots w_n$ b	$\vec{w} = [w_1 \dots w_n]$ {length n b still a number.
Model $f_{\vec{w},b}(\vec{x}) = w_1x_1 + \dots + w_nx_n + b$	$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$ ↑ dot product.
Cost function $J(w_1, \dots, w_n, b)$	$J(\vec{w}, b)$
Gradient descent. repeat { $w_j = w_j - \alpha \frac{d}{dw_j} J(w_1, \dots, w_n, b)$	repeat { $w_j = w_j - \alpha \frac{d}{dw_j} J(\vec{w}, b)$
$b = b - \alpha \frac{d}{db} J(w_1, w_2, \dots w_n, b)$.	$b = b - \alpha \frac{d}{db} J(\vec{w}, b)$
	}.

- Gradient Descent.

  for one feature. { $w = w - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

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

  simultaneously update w, b.

  }.

for n features ($n \geq 2$).

$j=1$ { $w_1 = w_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$

↳ this is $\frac{d}{dw_1} J(\vec{w}, b)$.

$\vdots$

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

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

we simultaneously update
w_j (for $j = 1, \ldots, n$) and b.

III. An alternative to gradient descent.

Normal Equation: Works only for linear regression.
Solve for w, b without iterations.

Disadvantages
- Doesn't generalize to other learning algorithms.
- Slow when number of features is large (> 1000).

-· What we need to know:
Normal equation method may be used in machine learning libraries that implement linear regression.
Gradient descent is the recommended method for finding parameters w, b.

IV. Quiz.

1. Given data, what is $x_4^{(3)}$?
4^{th} feature, 3^{rd} example.
So, 3^{rd} row, 4^{th} column.

2. Which of the following are potential benefits of vectorization?
a) code run fast b) code shorter.
c) code run more easily on parallel compute hardware
d) All ·· → Answer d). all.

3) To make gradient descent converge about twice as fast, a technique that almost works is to double the learning rate alpha α?
→ False.