Subject: Linear Regression with multiple features / /

Terminology:

j=column

1) x_j = j^{th} feature
2) n = number of features
3) $\vec{x}^{i}$ = i^{th} training examples → list / Vector

Row

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

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

$$\vec{w} = [w_1 \; w_2 \; w_3 \cdots w_n]$$

b is a number

$$\vec{x} = [x_1 \; x_2 \; x_3 \cdots x_n]$$

parametres of the model

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

Vectorization:

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

Group One: Routines Allocating Memory and Filling Arrays with Values

These routines allocate memory and fill arrays with default or random values. They accept the shape of the array as an input argument.

```python
a = np.zeros(4)
print(f"np.zeros(4) :   a = {a}, a shape = {a.shape}, a data type = {a.dtype}")
```

- `np.zeros(4)`:
  - Creates a 1D array of shape `(4,)` filled with zeros.
  - Default data type is `float64`.

```python
a = np.zeros((4,))
print(f"np.zeros(4,) :  a = {a}, a shape = {a.shape}, a data type = {a.dtype}")
```

- `np.zeros((4,))`:
  - Functionally the same as `np.zeros(4)`. Creates a 1D array of shape `(4,)` filled with zeros.
  - Explicitly specifying the shape as a tuple.

```python
a = np.random.random_sample(4)
print(f"np.random.random_sample(4): a = {a}, a shape = {a.shape}, a data type = {a.dtype}"
```

- `np.random.random_sample(4)`:
  - Creates a 1D array of shape `(4,)` filled with random values sampled from a uniform distribution over `[0, 1)`.
  - Default data type is `float64`.

Group Two: Routines Allocating Memory and Filling Arrays with Values but Not Accepting Shape as Input Argument

These routines also allocate memory and fill arrays with values, but they do not explicitly take the shape as an input argument. Instead, they use other parameters to determine the size and values.

```python
a = np.arange(4.)
print(f"np.arange(4.):     a = {a}, a shape = {a.shape}, a data type = {a.dtype}")
```

- `np.arange(4.)`:
  - Generates a 1D array with values `[0, 1, 2, 3]`.
  - The input argument is the stop value.
  - Shape is `(4,)` and data type is `float64` (due to the floating-point input `4.`).

```python
a = np.random.rand(4)
print(f"np.random.rand(4): a = {a}, a shape = {a.shape}, a data type = {a.dtype}")
```

- `np.random.rand(4)`:
  - Generates a 1D array of shape `(4,)` filled with random values sampled from a uniform distribution over `[0, 1)`.
  - The input argument specifies the shape implicitly.

Group Three: Routines Allocating Memory and Filling with User-Specified Values

These routines allow the user to directly specify the values that fill the array.

```python
a = np.array([5, 4, 3, 2])
print(f"np.array([5,4,3,2]):  a = {a},     a shape = {a.shape}, a data type = {a.dtype}")
```

- `np.array([5, 4, 3, 2])`:
  - Creates a 1D array with values `[5, 4, 3, 2]`.
  - Shape is `(4,)` and data type is inferred from the values, which is `int64`.

```python
a = np.array([5., 4, 3, 2])
print(f"np.array([5.,4,3,2]): a = {a}, a shape = {a.shape}, a data type = {a.dtype}")
```

- `np.array([5., 4, 3, 2])`:
  - Creates a 1D array with values `[5., 4, 3, 2]`.
  - Shape is `(4,)` and data type is inferred from the values, which is `float64` (due to the floating-point `5.`).

Subject: /   /

Indexes:

- Negative values means counting from end

Slicing:

```
#vector slicing operations
a = np.arange(10)
print(f"a         = {a}")

#access 5 consecutive elements (start:stop:step)
c = a[2:7:1];     print("a[2:7:1] = ", c)

# access 3 elements separated by two
c = a[2:7:2];     print("a[2:7:2] = ", c)

# access all elements index 3 and above
c = a[3:];        print("a[3:]    = ", c)

# access all elements below index 3
c = a[:3];        print("a[:3]    = ", c)

# access all elements
c = a[:];         print("a[:]     = ", c)
```

For 2D array;

a[row, col]

into 2D
from 1D   colums

np.arange(6).reshape(-1, 2)

auto
choose
rows

Gradient descent for muliple linear regression:

cost function $J(w_1, \dots, w_n, b)$

$J(\vec{w}, b)$

$$w = w - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$

Subject:

$\frac{\partial}{\partial w_j} J(\vec{w}, b)$;

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

$\frac{\partial}{\partial b} J(\vec{w}, b)$;

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

Alternate method;

Normal equation:
- Only for LR

Feature Scaling:

price = $w_1 x_1 + w_2 x_2 + b$

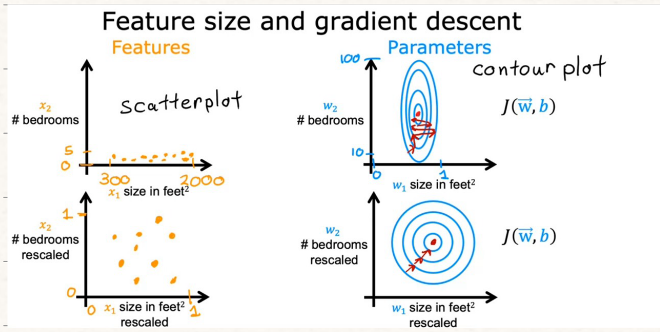
x_1: size (ft^2) — 300 — 2000 — large — model would pick small w_1

x_2: # bedrooms — 0 — 5 — small — model will pick large w_2

	size of x_j	size of the parametre
size	⟷	⟷
#bedrooms	⟷	⟷

it would take a small w to cause a massive shift in cost cuz x_1 is large. it would take a large w to cause a a massive cuz x_1 is small.

Subject:



How to scale features?

$\frac{300}{2000} \le x_1 \le \frac{2000}{2000}$ $\qquad$ $\frac{0}{5} \le x_2 \le \frac{5}{5}$

$0.15 \le x_1 \le 1$ $\qquad$ $0 \le x_2 \le 1$

Do normalization aswell / Centre at 0.

$x_1 = \frac{x_1 - \mu_1}{2000 - 300}$ avg $\qquad$ $x_2 = \frac{x_2 - \mu_2}{5 - 0}$

$-0.18 \le x_1 \le 0.82$ $\qquad$ $-0.46 \le x_2 \le 0.54$

Z-score normalization:

$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$ $\qquad$ $x_2 = \frac{x_2 - \mu}{\sigma_2}$

$-0.67 \le x_2 \le 3.1$ $\qquad$ $-1.6 \le x_2 \le 1.9$

Aim for $-1 \le x \le 1$ for each feature

Subject: / /

How to check if gradient descent converges?

J should always decreases

Curve should become a straight line

Automatic convergence test:

let ϵ be 10^{-3}

if $J(\vec{w},b)$ decreases by $\leq \epsilon$ in one direction, declare convergence

How to choose α (learning rate)?

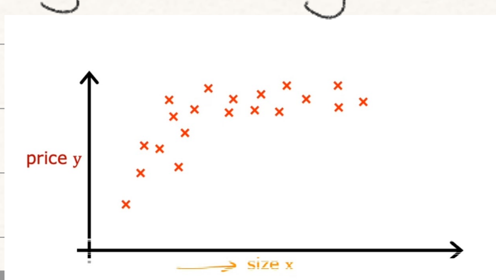
move by factors of 3.

0.1 0.001 0.0001 ...

Feature Engineering: → How to choose features

Using intuition to design new features by transforming or combining original features.

Polynomial Regression:



Subject: / /

$f_{\vec{w},b}(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$ $\qquad x_2 = x_1^2 \qquad x_3 = x_1^3$

$0-10^3 \qquad 0-10^6 \qquad 0-10^9$

• feature scaling begins very imp with this regression.

np.c_[x, x^2, x^3]

Join x, x^2, x^3 cols