LinearRegression from scratch

October 25, 2023

1 LinearRegression from scratch

let us take,

- Input Matrix X of size (1000, 5)
- True Weight Array W_t of size (5,)
- True Bias b_t
- Output Array Y of size (1000,)
- Weight Array W of size (5,)
- Bias b

[1]:
```
from random import uniform

x = [[uniform(-1, 1) for _ in range(5)] for _ in range(1000)]
wt = [uniform(-1, 1) for _ in range(5)]
bt = uniform(-1, 1)
y = [sum([j*k+bt for j,k in zip(i, wt)]) for i in x]

w = [uniform(-1, 1) for _ in range(5)]
b = uniform(-1, 1)
```

forward function:

$$forward(x, w, b) = \sum_{n=1}^{i}(x_n w_n) + b$$

[2]:
```
def forward(x, w, b):
    s = 0
    for i in range(len(x)):
        s += x[i] * w[i]
    return s + b
```

`mod_w` is modify weight function

[3]:
```
def mod_w(w, h, index):
    w[index] = w[index] + h
    return w
```

loss function:

$loss(x, y, w, b, h, index) = (y - forward(x, mod_w(w, h, index), b))^2$

$loss_b(x, y, w, b, h) = (y - forward(x, w, b + h))^2$

[4]:
```
def loss(x, y, w, b, h, index):
    return (y - forward(x, mod_w(w, h, index), b)) ** 2

def loss_b(x, y, w, b, h):
    return (y - forward(x, w, b+h)) ** 2
```

grad function:

$$grad(x, y, w, b, h, index) = \frac{loss(x, y, w, b, h, index) - loss(x, y, w, b, -h, index)}{2 * h}$$

$$grad_b(x, y, w, b, h) = \frac{loss_b(x, y, w, b, h) - loss_b(x, y, w, b, -h)}{2 * h}$$

[5]:
```
def grad(x, y, w, b, h, index):
    return (loss(x, y, w, b, h, index) - loss(x, y, w, b, -h, index)) / (2 * h)

def grad_b(x, y, w, b, h):
    return (loss_b(x, y, w, b, h) - loss_b(x, y, w, b, -h)) / (2 * h)
```

overall_loss:

$$overall\_loss(x, y, w, b) = \frac{1}{N} \sum^{i} \left(y_i - \sum^{j} (x_{ij} * w_j) - b \right)^2$$

[6]:
```
def overall_loss(x, y, w, b):
    loss = 0
    for index, x_row in enumerate(x):
        s = 0
        for i in range(len(w)):
            s += w[i] * x_row[i]
        s += b
        loss += (y[index] - s) ** 2
    loss /= len(x)
    return loss
```

$W \leftarrow W - lr \cdot \triangle W$

$b \leftarrow b - lr \cdot \triangle b$

[7]:
```
h = 0.001
lr = 0.001
epochs = 10
print(f'Initial Loss: {overall_loss(x, y, w, b)}')
for _ in range(epochs):
    for i in range(len(x)):
        dw = []
```

```
        for w_i in range(len(w)):
            dw.append(grad(x[i], y[i], w, b, h, w_i))
        b -= lr * grad_b(x[i], y[i], w, b, h)
        for w_i in range(len(w)):
            w[w_i] -= lr * dw[w_i]
print(f'Final Loss: {overall_loss(x, y, w, b)}')

print(w, b)
```

```
Initial Loss: 0.8563467004061935
Final Loss: 0.00045021505358297287
[0.7572138982576813, 0.3777155637953271, -0.8685347316357224,
-0.15010974717466563, -0.1774377866127173] -0.35374644562915714
```

2 Rust code

```
use rand::Rng;

fn main() {
    let mut rng = rand::thread_rng();

    let mut x = [[0.0;5];1000];
    for i in 0..1000 {
        for j in 0..5 {
            x[i][j] = rng.gen_range(-1.0..1.0);
        }
    }

    let mut wt = [0.0;5];
    for i in 0..5 {
        wt[i] = rng.gen_range(-1.0..1.0);
    }

    let bt = rng.gen_range(-1.0..1.0);

    let mut y = [0.0;1000];
    for i in 0..1000 {
        let mut s = 0.0;
        for j in 0..5 {
            s += x[i][j] * wt[j];
        }
        s += bt;
        y[i] = s;
    }

    let mut w = [0.0;5];
    for i in 0..5 {
```

```
    w[i] = rng.gen_range(-1.0..1.0);
}

let mut b = rng.gen_range(-1.0..1.0);

fn forward(x:[f64;5], w:[f64;5], b:f64) -> f64 {
    let mut s = 0.0;
    for i in 0..x.len() {
        s += x[i] * w[i];
    }
    return s + b
}

fn mod_w(mut w:[f64;5], h:f64, index:usize) -> [f64;5] {
    w[index] = w[index] + h;
    return w
}

fn loss(x:[f64;5], y:f64, w:[f64;5], b:f64, h:f64, index:usize) -> f64 {
    return (y - forward(x, mod_w(w, h, index), b)).powi(2)
}

fn loss_b(x:[f64;5], y:f64, w:[f64;5], b:f64, h:f64) -> f64 {
    return (y - forward(x, w, b+h)).powi(2);
}

fn grad(x:[f64;5], y:f64, w:[f64;5], b:f64, h:f64, index:usize) -> f64 {
    return (loss(x, y, w, b, h, index) - loss(x, y, w, b, -h, index)) / (2.0 * h)
}

fn grad_b(x:[f64;5], y:f64, w:[f64;5], b:f64, h:f64) -> f64 {
    return (loss_b(x, y, w, b, h) - loss_b(x, y, w, b, -h)) / (2.0 * h)
}

fn overall_loss(x:[[f64;5];1000], y:[f64;1000], w:[f64;5], b:f64) -> f64 {
    let mut loss = 0.0;
    for (index, x_row) in x.iter().enumerate() {
        let mut s = 0.0;
        for i in 0..w.len() {
            s += w[i] * x_row[i];
        }
        s += b;
        loss += (y[index] - s).powi(2);
    }
    loss /= x.len() as f64;
    return loss
}
```

```rust
    let h = 0.001;
    let lr = 0.001;
    let epochs = 10;
    println!("Initial Loss: {}", overall_loss(x, y, w, b));
    for _ in 0..epochs {
        for i in 0..x.len() {
            let mut dw = vec![];
            for w_i in 0..w.len() {
                dw.push(grad(x[i], y[i], w, b, h, w_i));
            }
            b -= lr * grad_b(x[i], y[i], w, b, h);
            for w_i in 0..w.len() {
                w[w_i] -= lr * dw[w_i];
            }
        }
    }
    println!("Final Loss: {}", overall_loss(x, y, w, b));

    println!("{:?}, {}", w, b);
}
```

output:

```
Initial Loss: 2.068049256824565
Final Loss: 0.000003865092504845875
[-0.2929867467493502, -0.8034504562551548, -0.06222288698532642, -0.4897335869051992, -0.27658
```

3 test.py

[8]:
```python
from time import time
import os

print('Python...')
start = time()
os.system('python3 main.py')
python = time() - start
print()

print('Rust...')
start = time()
os.system('./target/release/rust')
rust = time() - start
print()

if rust < python:
    print('Rust wins')
    print(f'Rust is {python/rust:.3} times faster than Python')
else:
```

```
    print('Python wins')
    print(f'Python is {rust/python:.3} times faster than Rust')
```

```
Python…
Initial Loss: 6.5533515924195775
Final Loss: 0.00013338218358322388
[-0.011026382454198862, 0.06956567281878773, -0.27789901167633374,
-0.16600908165267766, 0.04886392486687178] -2.8845026007399257

Rust…
Initial Loss: 1.388254800292784
Final Loss: 0.000030826400600496903
[0.2971948968042957, -0.7231705842722433, -0.4373733804989181,
0.46360040410574654, -0.7510819303205699], -0.03502819958995704

Rust wins
Rust is 63.2 times faster than Python
```

Use Standard libraries like Numpy, Pandas, Scikit-learn, to increase python's performance