Logistic regression 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

sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

In [ ]:
```
from random import uniform
from math import exp

sigmoid = lambda x: 1 / (1 +  exp(-x))

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 = [sigmoid(sum([j*k for j,k in zip(i, wt)]) + bt) for i in x]

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

forward function:

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

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

`mod_w` is modify weight function

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

loss function:

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

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

In [ ]:
```
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:

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

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

In [ ]:
```
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:

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

In [ ]:
```
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] - sigmoid(s)) ** 2
    loss /= len(x)
    return loss
```

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

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

In [ ]:
```
h = 0.001
lr = 0.001
epochs = 100
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: 0.0315531837739302
Final Loss: 0.0007861730447066337
[-0.5791577855052624, -0.013714686445052426, 0.3944937597901887, 0.3068389984979608
7, 0.8505579825715035] 0.5777139741019259
```

Rust code

```rust
use rand::Rng;
use std::f64::consts::E;

fn main() {
    fn sigmoid(x:f64) -> f64 {
        1.0 / (1.0 + E.powf(-x))
    }

    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] = sigmoid(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 sigmoid(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] - sigmoid(s)).powi(2);
        }
        loss /= x.len() as f64;
        return loss
    }

    let h = 0.001;
    let lr = 0.01;
    let epochs = 100;
    let mut low_loss = overall_loss(x, y, w, b);
    let mut opt_w = w.clone();
    let mut opt_b = b.clone();
    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];
            }
        }
        let l = overall_loss(x, y, w, b);
        if l < low_loss {
            low_loss = l;
            opt_w = w.clone();
            opt_b = b.clone();
        }
    }
    println!("Final Loss: {}", overall_loss(x, y, w, b));
    // println!("w: {:?}, b: {}", w, b);
    println!("Lowest Loss: {}", low_loss);
    // println!("opt w: {:?}\nopt b: {}", opt_w, opt_b);
}
```

output:

```
Initial Loss: 0.061047323023535924
Final Loss: 0.00000000000000012481917588280155
Lowest Loss: 0.00000000000000008043114705082252
```

test.py

In [ ]:
```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: 0.08861057513058064
Final Loss: 2.6220992829503962e-08
Lowest Loss: 2.6220992829503962e-08

Rust...
Initial Loss: 0.0660014402299525
Final Loss: 0.00000000000000084887449529843575
Lowest Loss: 0.00000000000000050596395636515 29

Rust wins
Rust is 43.5 times faster than Python
```

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