Gradiente Descendente

Fernanda Rafaela, RA: 0051352121018

November 2024

1 Introdução

O gradiente descendente é um algoritmo que busca o menor valor de uma função de custo, ajustando os parâmetros do modelo na direção em que o erro diminui mais rápido. Ele atualiza os parâmetros em passos definidos pela taxa de aprendizado e pelo gradiente. O processo continua até o erro ser minimizado ou até que um critério de parada seja atingido. Nesse trabalho vamos usar esse algoritmo para regressão polinomial e para regressão linear.

2 Regressão polinomial

A regressão polinomial modela a relação entre uma variável dependente e uma ou mais variáveis independentes por meio de um polinômio.

Primeiro vamos criar o dataset:

```
X = [i / 10 for i in range(1, 5)]
y = [i**18 + i + 1 for i in X]
```

Gráfico do nosso dataset:

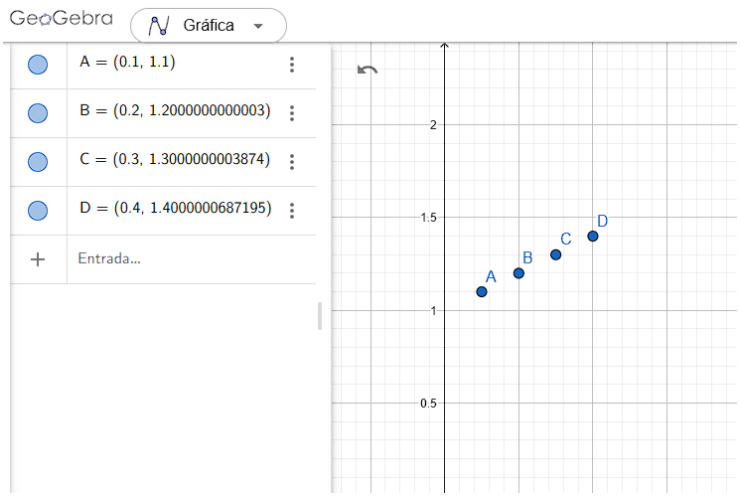

Figure 1: Gráfico polinomial grau 18

Algoritmo:

```python
def grad_18(input, output, w1, w2, b):
    dL_dw1 = 0
    dL_dw2 = 0
    dL_db = 0
    for i, out in zip(input, output):
        pred = w1 * i**18 + w2 * i + b
        dL_dw1 += -2 * (out - pred) * i**18
        dL_dw2 += -2 * (out - pred) * i
        dL_db += -2 * (out - pred)
    return dL_dw1, dL_dw2, dL_db

def descent(init, lr, epochs, input, output, gradient, tol=10**-6):
    count = 0
    for _ in range(epochs):
        count += 1
        grad = gradient(input, output, *init)
        grad = [init[i] - lr * grad[i] for i in range(len(grad))]
        err = sum((output[i] - (init[0] * input[i]**18 + init[1] * input[i] + init[2]))**2 for i in range(len(input)))
        if err < tol:
            break
        init = grad
    return grad, err, count
```

Figure 2: Algoritmo python

Ao rodar, esses são os resultados:

```
Parametros finais: [1.999999828745158, 1.0043267923377086, 0.9989054157785566]
Erro final: 9.730717691697086e-07
Numero de iteracoes: 268
```

Aplicando os parâmetros finais no gráfico inicial:

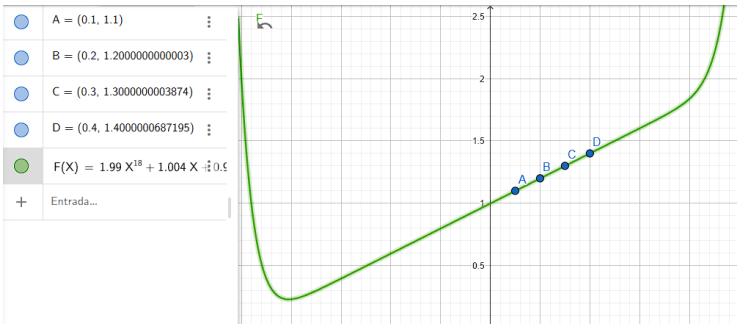

Figure 3: Regressão polinomial grau 18

3 Regressão Logística

A regressão logística estima a probabilidade de ocorrência de um evento, como chuva ou não chuva, com base no dataset.

Primeiro vamos criar o dataset:

```
X = [i / 10 for i in range(1, 11)]
y = [1 if i >= 0.5 else 0 for i in X]
```

Gráfico do nosso dataset:

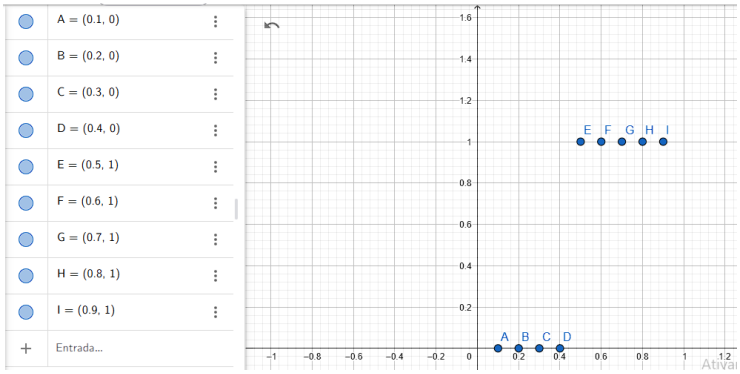

Figure 4: Gráfico regressão logística

Algoritmo:

liner.py > descent

```python
import math

def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def grad_logistic(input, output, w1, b):
    dL_dw1 = 0
    dL_db = 0
    for i, out in zip(input, output):
        pred = sigmoid(w1 * i + b)
        dL_dw1 += - (out - pred) * i
        dL_db += - (out - pred)
    return dL_dw1, dL_db

def descent(init, lr, epochs, input, output, gradient, tol=10**-6):
    count = 0
    for _ in range(epochs):
        count += 1
        grads = gradient(input, output, *init)
        init = [init[i] - lr * grads[i] for i in range(len(grads))]
        err = -sum(out * math.log(sigmoid(init[0] * i + init[1])) + (1 - out) * math.log(1 - sigmoid(init[0] * i + init[1]))
                   for i, out in zip(input, output))
        if err < tol:
            break
    return init, err, count
```

Figure 5: Algoritmo python

Calcular acuracia e f1-score:

```python
def evaluate(y_true, y_pred):
    y_pred_classes = [1 if pred >= 0.5 else 0 for pred in y_pred]

    correct = sum(1 for true, pred in zip(y_true, y_pred_classes) if true == pred)
    accuracy = correct / len(y_true)

    tp = sum(1 for true, pred in zip(y_true, y_pred_classes) if true == pred == 1)
    fp = sum(1 for true, pred in zip(y_true, y_pred_classes) if true == 0 and pred == 1)
    fn = sum(1 for true, pred in zip(y_true, y_pred_classes) if true == 1 and pred == 0)

    precision = tp / (tp + fp) if tp + fp > 0 else 0
    recall = tp / (tp + fn) if tp + fn > 0 else 0

    f1 = 2 * (precision * recall) / (precision + recall) if precision + recall > 0 else 0

    return accuracy, f1
```

Figure 6: Algoritmo acuracia e f1-score

Ao rodar, esses são os resultados:

```
Parametros finais: [65.0735159543723, -29.26358529339361]
Erro final: 0.07593793118873807
Numero de iteracoes: 10000
Acuracia: 1.0
F1-Score: 1.0
```

Aplicando os parâmetros finais no gráfico inicial:

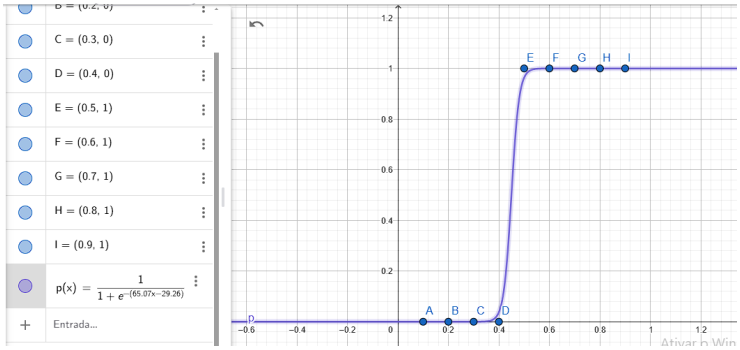

Figure 7: Regressão Logística

Repo: https://github.com/xfeehrxx/gradienteDesc