

MAC0460 - Relatório do EP3

Nome: Andrew Ijano Lopes

NUSP: 10297797

1 Implementação

O programa `ep3.py` é composto de duas funções: `logistic_fit`, que usa o algoritmo de regressão logística para encontrar o vetor \mathbf{w} de pesos e o `logistic_predict`, que calcula as previsões à partir do vetor de pesos.

1.1 Função de *Fitting*

Primeiro, guardamos valores úteis e adicionamos a coluna de 1's em X .

```
N, d = X.shape
Xe = np.hstack((np.ones((N, 1)), X))
```

Em seguida, tratamos o caso em que \mathbf{w} é `None`, criando o vetor de pesos aleatório, e criamos a lista `w_list` que guardará o histórico dos pesos.

```
if w is None:
    w = np.random.rand(d + 1)
w_list = [w]
```

Além disso, tratamos os casos em que igualamos `batch_size` a N e calculamos o número de *batches*.

```
if batch_size is None or batch_size > N:
    batch_size = N
batch_num = math.ceil(N/batch_size)
```

Com isso, realizamos `num_iterations` iterações e, para cada iteração, executamos os próximos cálculos para cada *batch*.

```
for _ in range(num_iterations):
    batch_ini = 0
    for __ in range(batch_num):
        batch_end = batch_ini + batch_size
```

A expressão abaixo implementa o cálculo do gradiente de E_{in} para o conjunto de pontos do *batch* atual. Ou seja,

$$\nabla E_{in} = -\frac{1}{batch_size} \sum_{n=batch_ini}^{batch_end} \frac{y_n x_n}{1 + e^{y_n w^T(t) x_n}}.$$

```

gradient = (-1/batch_size) * sum(
    y[i]*Xe[i] / (1 + math.exp((y[i]*w.T).dot(Xe[i])))
    for i in range(batch_ini, batch_end)
)

```

Em seguida, o vetor de pesos é atualizado, implementando a expressão $w(t+1) = w(t) - \eta \nabla E_{in}$, onde t é a iteração atual. E ainda, guardamos o novo vetor de pesos na lista de histórico e atualizamos o índice de início do próximo *batch*.

```

w = w - learning_rate * gradient
w_list += [w]

batch_ini = batch_end

```

Ao final das iterações, retornamos w e o histórico de pesos, de acordo com a flag dada.

```

if return_history:
    return w, w_list
return w

```

1.2 Função de *Prediction*

Como na função anterior, primeiro guardamos valores úteis e adicionamos a coluna de 1's em X .

```

N, d = X.shape
Xe = np.hstack((np.ones((N, 1)), X))

```

Em seguida, definimos a função *sigmoid*, dada por $h(s) = \frac{e^s}{1+e^s}$.

```

def h(s): return math.exp(s) / (1 + math.exp(s))

```

Por fim, retornamos a lista da função $h(s)$ aplicada para s , com $s = w^T x$ e x um elemento de Xe .

```

return [h(w.T.dot(x)) for x in Xe]

```

2 Fontes consultadas

Para a realização desse exercício, foram consultadas apenas as *Lectures* 09 e 10 do Mostafa, as documentações do *numpy* e os script dado pelo monitor no Paca.

3 Testes realizados e resultados obtidos

Para testar as funções implementadas, foi criado um script `logistic_regression_test.py`. Com ele, são gerados duas amostras de pontos, aleatórios que alimentam o algoritmo de *Fitting*, e, em seguida, são usados no função de *Prediction*.

O script permite a geração de amostras com médias e covariâncias aleatórias e plotagem em 2D e 3D, além da variação de várias flags de acordo com o modo de uso abaixo:

```
usage: logistic_regression_test.py [-h] [-N N] [--batch_size B_SIZE]
  [--learning_rate L_RATE] [--iterations ITER]
  [--3d] [--random]
optional arguments:
-h, --help            show this help message and exit
-N N                  the number of samples used
--batch_size B_SIZE  the batch size used
--learning_rate L_RATE
the fitting learning rate used
--iterations ITER    the number of iterations used in fitting
--3d                 test with a 3D set of points
--random              use random means and covariances
```

O caso padrão desse script é a geração de amostras 2D. O teste abaixo obtêm o seguinte resultado. As imagens abaixo mostram as amostras geradas à esquerda e o resultado das probabilidades usando o *Prediction* à direita.

```
python3 logistic_regression_test.py
```

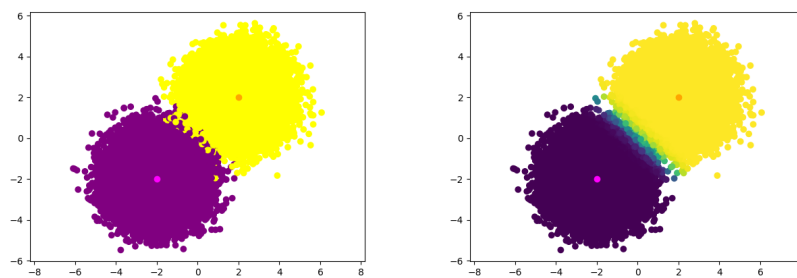


Figure 1: Primeira amostra de pontos 2D

Alterando um pouco as flags usadas obtemos o seguinte resultado.

```
python3 logistic_regression_test.py --random -N=50000
--batch_size=50 --iterations=200
```

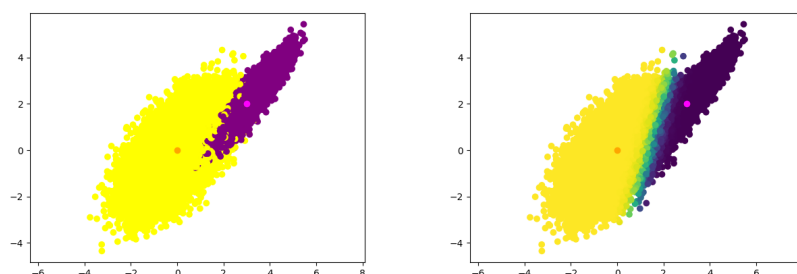


Figure 2: Segunda amostra de pontos 2D

Para testar o algoritmo em um espaço de dimensões maior que 2, é possível gerar e plotar amostras 3D. Como nos dois testes abaixo.

```
python3 logistic_regression_test.py --3d
```

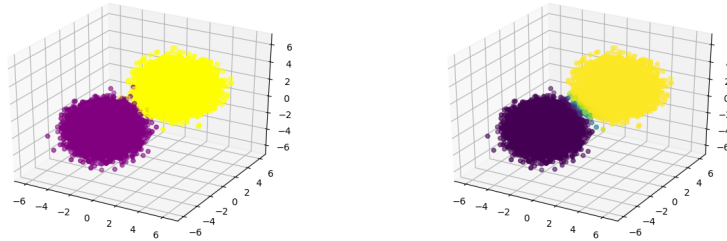


Figure 3: Primeira amostra de pontos 3D

```
python3 logistic_regression_test.py --3d --random
```

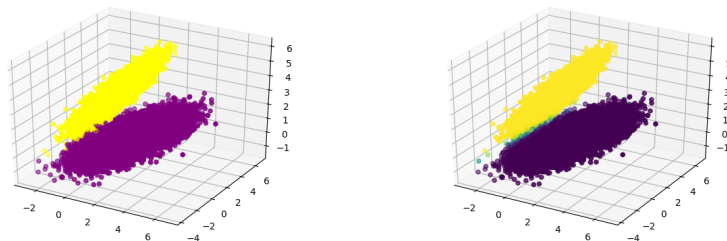


Figure 4: Segunda amostra de pontos 3D

Assim, verificamos que as duas funções implementadas se comportam corretamente nos casos testados.