intro

March 4, 2025

[1]:
```python
import pandas as pd
import numpy as np
```

0.1 Kardec data read

[2]:
```python
def matriz(num_colunas, array1):
    """
    Gera uma matriz sequencial a partir de um array, com o número de colunas␣
 ↪especificado.

    Args:
        array (list ou np.ndarray): Array de entrada.
        num_colunas (int): Número de colunas desejado na matriz.

    Returns:
        np.ndarray: Matriz sequencial.
    """
    if num_colunas > len(array1):
        raise ValueError("O número de colunas não pode ser maior que o tamanho␣
 ↪do array.")

    # Número de linhas na matriz
    num_linhas = len(array1) - num_colunas + 1

    # Criando a matriz sequencial
    matriz = np.array([array1[i:i + num_colunas] for i in range(num_linhas)])
    return matriz
```

[3]:
```python
array1 = np.arange(600)
array1
```

[3]:
```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
```

```
 91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259,
260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272,
273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311,
312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337,
338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363,
364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376,
377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402,
403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415,
416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,
429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,
442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467,
468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,
481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,
494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506,
507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532,
533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545,
546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558,
559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571,
572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584,
585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597,
598, 599])
```

[4]:
```
matriz1 = matriz(60, array1)
matriz1
```

[4]:
```
array([[  0,   1,   2, …,  57,  58,  59],
       [  1,   2,   3, …,  58,  59,  60],
       [  2,   3,   4, …,  59,  60,  61],
```

```
       …,
       [538, 539, 540, …, 595, 596, 597],
       [539, 540, 541, …, 596, 597, 598],
       [540, 541, 542, …, 597, 598, 599]])
```

[5]:
```python
data1 = pd.read_csv('/home/darkcover/Documentos/Out/dados/Saidas/FUNCOES/DOUBLE␣
 ↪- 17_09_s1.csv')
data1.columns
```

[5]:
```
Index(['n', 'Entrada', 'Odd', 'P60', 'P120', 'P180', 'P240', 'P300', 'P360',
       'P420', 'P480', 'P540', 'P600', 'P660', 'P720', 'P780', 'P840', 'P900',
       'P960', 'P1020', 'P1080', 'P1140', 'P1200', 'P1260', 'P1320', 'P1380',
       'P1440', 'P1500', 'P1560', 'P1620', 'P1680', 'P1740', 'P1800', 'P1860',
       'P1920', 'P1980', 'P1200.1', 'Media Movel', 'Unnamed: 38',
       'Unnamed: 39', 'Unnamed: 40', 'Unnamed: 41', 'Unnamed: 42',
       'Unnamed: 43', 'Unnamed: 44', 'Unnamed: 45', 'Acertos 60',
       'Unnamed: 47', 'Unnamed: 48', 'Unnamed: 49', 'Unnamed: 50',
       'Unnamed: 51', 'Unnamed: 52', 'Unnamed: 53', 'Unnamed: 54',
       'Unnamed: 55', 'Unnamed: 56', 'Unnamed: 57', 'Unnamed: 58',
       'Unnamed: 59', 'Unnamed: 60', 'Unnamed: 61', 'Unnamed: 62',
       'Unnamed: 63', 'Unnamed: 64', 'Unnamed: 65', 'Unnamed: 66',
       'Unnamed: 67', 'Unnamed: 68', 'Acertos Geral', 'Média Global',
       'Unnamed: 71', 'Unnamed: 72', 'Unnamed: 73', 'Unnamed: 74',
       'Unnamed: 75', 'Unnamed: 76', 'Unnamed: 77', 'Unnamed: 78',
       'Unnamed: 79', 'Unnamed: 80', 'Unnamed: 81', 'Unnamed: 82',
       'Unnamed: 83', 'acertos_intervalos', 'Unnamed: 85'],
      dtype='object')
```

[6]:
```python
array2 = data1['Entrada']
array2
```

[6]:
```
0        11,6
1        2,02
2        2,02
3        1,54
4         2,2
        …
1667        1
1668     4,34
1669    19,98
1670     2,52
1671     10,2
Name: Entrada, Length: 1672, dtype: object
```

[7]:
```python
array3 = []
array4 = []
for i in range(600):
```

```python
    odd = array2[i].replace(',','.')
    if float(odd) >= 6:
        odd = 6
    array3.append(float(odd))
    if float(odd) >= 3:
        corte1 = 1
    else:
        corte1 = 0
    array4.append(corte1)

array3
```

[7]:
```
[6.0,
 2.02,
 2.02,
 1.54,
 2.2,
 1.51,
 6.0,
 3.89,
 1.02,
 1.48,
 1.25,
 1.07,
 1.93,
 4.33,
 3.59,
 3.26,
 1.83,
 1.6,
 2.07,
 6.0,
 1.21,
 5.46,
 3.81,
 1.3,
 1.95,
 1.04,
 1.28,
 1.03,
 1.05,
 3.39,
 1.05,
 2.59,
 2.53,
 6.0,
 1.34,
```

```
1.56,
1.0,
1.22,
1.05,
6.0,
2.54,
2.44,
2.34,
6.0,
6.0,
1.44,
1.63,
2.57,
1.73,
1.22,
1.34,
1.33,
1.0,
2.87,
2.28,
6.0,
1.0,
1.73,
1.05,
1.52,
5.63,
4.44,
3.91,
6.0,
2.44,
1.43,
1.83,
1.31,
1.61,
1.73,
1.4,
2.12,
1.49,
5.97,
1.12,
1.55,
1.4,
2.62,
2.87,
1.77,
1.02,
1.28,
```

```
6.0,
2.18,
1.99,
4.71,
1.01,
1.15,
1.41,
6.0,
1.92,
1.2,
1.0,
2.35,
6.0,
1.24,
4.81,
1.0,
1.27,
1.49,
1.94,
1.98,
5.93,
1.76,
1.9,
1.71,
6.0,
6.0,
6.0,
2.03,
1.14,
1.29,
6.0,
1.0,
2.21,
3.09,
1.49,
3.34,
1.04,
2.26,
1.21,
1.26,
1.01,
1.21,
3.08,
2.7,
2.3,
2.51,
1.17,
```

```
1.02,
4.1,
1.17,
1.61,
6.0,
2.74,
1.67,
5.5,
1.41,
1.13,
1.0,
1.49,
2.3,
1.31,
1.18,
6.0,
6.0,
3.16,
4.08,
6.0,
6.0,
6.0,
2.16,
5.1,
6.0,
1.65,
1.0,
6.0,
2.12,
1.1,
6.0,
1.13,
1.64,
1.01,
2.11,
1.21,
1.71,
3.63,
6.0,
6.0,
3.42,
6.0,
5.45,
2.62,
3.2,
6.0,
3.1,
```

```
3.65,
2.88,
1.74,
6.0,
6.0,
2.47,
6.0,
1.52,
1.5,
2.36,
1.0,
6.0,
1.87,
2.11,
5.53,
1.46,
1.14,
6.0,
1.0,
2.3,
1.41,
3.8,
1.03,
3.41,
6.0,
1.0,
1.39,
2.36,
2.18,
1.3,
1.1,
1.87,
3.76,
1.52,
6.0,
2.08,
4.89,
5.59,
1.18,
6.0,
2.66,
1.65,
1.02,
1.59,
6.0,
1.28,
1.42,
```

```
1.08,
2.57,
1.02,
5.11,
1.96,
1.26,
1.17,
1.34,
1.05,
1.18,
2.99,
1.08,
2.19,
1.08,
1.14,
6.0,
2.29,
3.45,
1.07,
6.0,
1.19,
1.05,
1.67,
1.1,
6.0,
1.02,
1.53,
2.38,
1.38,
1.28,
5.21,
6.0,
1.17,
1.06,
3.69,
1.04,
1.54,
1.45,
1.47,
3.41,
6.0,
1.08,
2.15,
1.38,
6.0,
1.02,
4.11,
```

```
2.2,
4.17,
2.7,
1.04,
2.14,
1.37,
1.88,
2.85,
6.0,
1.09,
2.08,
4.32,
1.08,
5.52,
2.37,
2.47,
1.66,
1.3,
6.0,
2.6,
6.0,
1.65,
1.0,
1.88,
6.0,
1.51,
3.01,
2.11,
4.35,
1.14,
1.8,
5.07,
6.0,
2.43,
6.0,
1.97,
6.0,
6.0,
3.93,
1.57,
1.31,
5.01,
3.79,
1.34,
5.43,
1.48,
1.26,
```

```
1.02,
1.63,
3.74,
1.16,
6.0,
1.29,
2.93,
6.0,
1.01,
6.0,
1.02,
2.73,
2.0,
1.11,
1.49,
4.1,
1.0,
5.61,
1.19,
1.03,
1.22,
1.27,
2.54,
1.36,
1.16,
1.14,
2.85,
1.38,
1.36,
6.0,
1.28,
1.25,
2.47,
3.92,
2.87,
1.09,
1.02,
1.59,
1.72,
1.28,
1.19,
3.48,
1.79,
1.0,
1.08,
3.69,
1.32,
```

```
6.0,
1.42,
1.0,
4.26,
1.37,
4.83,
1.65,
1.36,
6.0,
6.0,
6.0,
2.95,
2.53,
6.0,
1.42,
1.65,
4.93,
1.3,
1.76,
1.21,
4.58,
1.84,
3.43,
6.0,
6.0,
1.61,
1.16,
4.51,
6.0,
2.75,
6.0,
1.65,
1.48,
1.53,
1.09,
1.12,
1.89,
3.44,
5.54,
1.37,
6.0,
3.48,
6.0,
1.07,
1.53,
1.14,
2.26,
```

```
1.01,
1.08,
1.01,
1.87,
1.04,
1.87,
3.55,
6.0,
1.05,
2.15,
1.54,
1.06,
1.35,
1.14,
2.66,
2.07,
1.37,
3.49,
1.09,
5.71,
1.85,
6.0,
1.5,
1.58,
2.32,
1.82,
3.7,
1.01,
6.0,
2.69,
3.86,
1.37,
1.74,
2.34,
1.75,
1.95,
1.25,
6.0,
6.0,
4.43,
2.03,
4.2,
6.0,
1.0,
1.08,
1.65,
1.09,
```

```
3.88,
3.35,
1.34,
1.64,
2.08,
2.59,
1.1,
1.08,
3.16,
6.0,
1.39,
6.0,
2.74,
3.18,
1.22,
1.12,
1.0,
1.5,
6.0,
1.73,
1.29,
6.0,
1.59,
1.55,
1.21,
1.7,
1.66,
4.99,
1.02,
1.73,
1.19,
2.93,
1.35,
1.07,
2.48,
1.12,
1.28,
1.96,
4.83,
1.2,
1.06,
1.09,
1.04,
2.14,
6.0,
2.36,
1.22,
```

```
6.0,
1.19,
1.17,
3.17,
2.85,
1.66,
2.03,
3.91,
4.78,
2.15,
1.23,
1.39,
2.1,
6.0,
2.43,
2.0,
1.12,
1.95,
1.0,
6.0,
1.0,
1.25,
1.58,
1.75,
1.81,
6.0,
3.14,
5.62,
1.37,
6.0,
1.0,
1.51,
1.43,
2.12,
1.33,
1.6,
1.06,
1.05,
1.01,
1.08,
1.67,
1.96,
5.43,
1.21,
1.11,
6.0,
2.13,
```

1.04,
2.98,
2.78,
1.87,
1.66,
1.09,
1.83,
1.7,
6.0,
1.35,
1.47,
1.01,
1.09,
1.21,
1.87,
1.51,
1.59,
2.17,
4.36,
3.61,
1.3,
1.06,
1.17,
1.11,
4.19,
1.78,
1.71,
1.12,
2.36,
1.71,
6.0,
1.2,
4.29,
1.23,
1.67,
1.02,
2.33,
1.0,
6.0,
3.95,
1.77,
1.33,
6.0,
2.49,
2.24,
6.0,
3.22,

```
1.49]
```

[8]:
```
matriz2 = matriz(60, array3)
matriz2
```

[8]:
```
array([[6.  , 2.02, 2.02, …, 1.73, 1.05, 1.52],
       [2.02, 2.02, 1.54, …, 1.05, 1.52, 5.63],
       [2.02, 1.54, 2.2 , …, 1.52, 5.63, 4.44],
       …,
       [2.12, 1.33, 1.6 , …, 2.49, 2.24, 6.  ],
       [1.33, 1.6 , 1.06, …, 2.24, 6.  , 3.22],
       [1.6 , 1.06, 1.05, …, 6.  , 3.22, 1.49]])
```

[9]:
```
matriz3 = matriz(60,array4)
matriz3
```

[9]:
```
array([[1, 0, 0, …, 0, 0, 0],
       [0, 0, 0, …, 0, 0, 1],
       [0, 0, 0, …, 0, 1, 1],
       …,
       [0, 0, 0, …, 0, 0, 1],
       [0, 0, 0, …, 0, 1, 1],
       [0, 0, 0, …, 1, 1, 0]])
```

[10]:
```
array5 = []
for i in range(len(array4) - 1):
    if i >= 59:
        order = sum(array4[i - 59: i])
        array5.append(order)
array5
```

[10]:
```
[15,
 14,
 15,
 16,
 17,
 18,
 18,
 17,
 16,
 16,
 16,
 16,
 16,
 16,
 15,
 15,
```

```
14,
14,
14,
14,
13,
13,
12,
11,
12,
12,
12,
13,
13,
13,
12,
13,
13,
13,
12,
12,
13,
13,
14,
14,
13,
13,
13,
13,
13,
12,
12,
12,
13,
14,
15,
15,
15,
15,
16,
16,
15,
16,
16,
17,
17,
16,
15,
```

```
14,
13,
13,
14,
14,
14,
14,
14,
14,
15,
15,
14,
15,
15,
15,
16,
16,
16,
16,
16,
15,
15,
15,
15,
16,
17,
18,
18,
19,
20,
20,
21,
21,
21,
20,
21,
21,
21,
22,
22,
21,
21,
21,
21,
20,
20,
20,
```

```
21,
22,
23,
23,
23,
24,
24,
25,
25,
25,
25,
26,
27,
27,
28,
27,
27,
27,
27,
28,
28,
27,
28,
28,
27,
28,
28,
27,
27,
28,
28,
29,
30,
30,
30,
29,
28,
27,
26,
25,
25,
24,
25,
24,
24,
25,
25,
```

```
25,
25,
25,
24,
24,
25,
25,
25,
25,
25,
24,
24,
23,
22,
21,
20,
20,
19,
18,
17,
16,
16,
16,
16,
15,
16,
15,
16,
16,
16,
16,
15,
16,
16,
15,
15,
15,
14,
15,
16,
16,
15,
16,
15,
14,
14,
14,
```

```
15,
16,
16,
16,
16,
16,
16,
16,
16,
16,
15,
15,
14,
14,
14,
14,
15,
14,
14,
15,
15,
16,
16,
15,
15,
15,
16,
16,
17,
17,
17,
17,
18,
18,
19,
18,
19,
18,
18,
18,
19,
19,
20,
20,
20,
21,
22,
```

```
22,
22,
23,
23,
22,
23,
23,
22,
22,
22,
23,
23,
23,
22,
22,
23,
23,
23,
23,
22,
22,
21,
21,
22,
22,
23,
23,
23,
22,
22,
22,
21,
21,
20,
20,
20,
20,
21,
20,
20,
19,
20,
20,
20,
19,
19,
18,
```

```
18,
17,
18,
18,
17,
16,
17,
16,
17,
16,
15,
15,
15,
16,
15,
14,
15,
15,
16,
16,
16,
17,
16,
16,
16,
16,
16,
15,
16,
15,
16,
17,
18,
18,
18,
18,
19,
18,
19,
19,
19,
19,
19,
19,
19,
20,
21,
```

```
21,
22,
22,
23,
23,
23,
22,
22,
22,
22,
22,
22,
22,
22,
22,
23,
23,
23,
22,
22,
21,
21,
21,
20,
20,
20,
20,
21,
20,
20,
19,
19,
19,
18,
19,
19,
19,
19,
20,
20,
19,
19,
18,
17,
16,
17,
18,
```

```
18,
17,
18,
18,
18,
18,
18,
18,
19,
20,
19,
18,
18,
17,
16,
15,
16,
17,
17,
18,
18,
19,
19,
19,
19,
19,
19,
18,
18,
19,
19,
19,
19,
19,
19,
20,
20,
19,
19,
18,
18,
17,
17,
17,
17,
17,
17,
```

```
17,
16,
16,
15,
15,
16,
16,
16,
17,
17,
16,
16,
15,
15,
14,
14,
15,
15,
15,
15,
14,
14,
14,
14,
14,
14,
14,
15,
14,
13,
13,
12,
12,
12,
13,
14,
14,
15,
14,
14,
14,
13,
13,
13,
13,
13,
13,
```

```
12,
12,
12,
13,
13,
13,
14,
14,
14,
14,
14,
13,
13,
13,
13,
13,
14,
13,
13,
13,
12,
12,
12,
11,
11,
11,
12,
12,
11,
11,
11,
11,
12,
11,
11,
11,
11,
11,
12,
11,
12,
12,
12,
12,
12,
11,
11,
```

```
  11,
  11,
  10,
  11,
  11,
  11,
  12]
```

[11]:
```
matriz4 = matriz(60, array5)
matriz4
```

[11]:
```
array([[15, 14, 15, …, 16, 16, 17],
       [14, 15, 16, …, 16, 17, 17],
       [15, 16, 17, …, 17, 17, 16],
       …,
       [14, 14, 13, …, 10, 11, 11],
       [14, 13, 13, …, 11, 11, 11],
       [13, 13, 13, …, 11, 11, 12]])
```

0.2 Indicador de futura função:

import numpy as np

def calculate_means(array4): “ “ ” Calcula a média dos elementos de array4 em janelas deslizantes de 59 elementos.

```
Args:
    array4 (list): Lista de inteiros (0 ou 1).

Returns:
    list: Lista com a média dos elementos em janelas de 59 elementos.
"""
array6 = []
array7 = []
for i in range(len(array4) - 1):
    array6.append(array4[i])
    if i >= 59:
        order = float(np.mean(array6))
        array7.append(order)
return array7
```

1 Exemplo de uso:

2 array4 = [0, 1, 0, 1, ...] # Supondo que array4 tenha elementos suficientes

3 array7 = calculate_means(array4)

4 Teste unitário básico

def test_calculate_means(): array4 = [1] * 60 # Lista com 60 elementos, todos iguais a 1 expected_output = [1.0] # A média dos primeiros 59 elementos é 1.0 array7 = calculate_means(array4) assert array7 == expected_output, "Teste falhou!"

5 Executar o teste

test_calculate_means()

[12]:
```python
array6 = []
array7 = []
for i in range(len(array4) - 1):
    array6.append(array4[i])
    if i >= 59:
        order = float(np.mean(array6))
        array7.append(order)

array7
```

[12]:
```
[0.25,
 0.26229508196721313,
 0.27419354838709675,
 0.2857142857142857,
 0.296875,
 0.2923076923076923,
 0.2878787878787879,
 0.2835820895522388,
 0.27941176470588236,
 0.2753623188405797,
 0.2714285714285714,
 0.2676056338028169,
 0.2638888888888889,
 0.2602739726027397,
 0.2702702702702703,
 0.26666666666666666,
 0.2631578947368421,
 0.2597402597402597,
 0.2564102564102564,
 0.25316455696202533,
```

```
0.25,
0.24691358024691357,
0.24390243902439024,
0.25301204819277107,
0.25,
0.24705882352941178,
0.2558139534883721,
0.25287356321839083,
0.25,
0.24719101123595505,
0.25555555555555554,
0.25274725274725274,
0.25,
0.24731182795698925,
0.24468085106382978,
0.25263157894736843,
0.25,
0.25773195876288657,
0.25510204081632654,
0.25252525252525254,
0.25,
0.24752475247524752,
0.24509803921568626,
0.2524271844660194,
0.25,
0.24761904761904763,
0.24528301886792453,
0.2523364485981308,
0.25925925925925924,
0.26605504587155965,
0.2636363636363636,
0.26126126126126126,
0.25892857142857145,
0.26548672566371684,
0.2631578947368421,
0.2608695652173913,
0.2672413793103448,
0.26495726495726496,
0.2711864406779661,
0.2689075630252101,
0.26666666666666666,
0.2644628099173554,
0.26229508196721313,
0.2601626016260163,
0.25806451612903225,
0.264,
0.2619047619047619,
```

0.25984251968503935,
0.2578125,
0.2558139534883721,
0.25384615384615383,
0.2595419847328244,
0.25757575757575757,
0.2556390977443609,
0.26119402985074625,
0.25925925925925924,
0.25735294117647056,
0.26277372262773724,
0.2608695652173913,
0.2589928057553957,
0.2571428571428571,
0.2553191489361702,
0.2535211267605634,
0.2517482517482518,
0.25,
0.25517241379310346,
0.2602739726027397,
0.2653061224489796,
0.2702702702702703,
0.2751677852348993,
0.28,
0.2847682119205298,
0.28289473684210525,
0.2875816993464052,
0.2922077922077922,
0.2903225806451613,
0.28846153846153844,
0.2929936305732484,
0.2911392405063291,
0.2893081761006289,
0.29375,
0.2919254658385093,
0.29012345679012347,
0.2883435582822086,
0.2865853658536585,
0.28484848484848485,
0.28313253012048195,
0.2874251497005988,
0.2916666666666667,
0.2958579881656805,
0.3,
0.30409356725146197,
0.3081395348837209,
0.3063583815028902,

```
0.3103448275862069,
0.3142857142857143,
0.3181818181818182,
0.3220338983050847,
0.3202247191011236,
0.31843575418994413,
0.3222222222222224,
0.3259668508287293,
0.3241758241758242,
0.32786885245901637,
0.32608695652173914,
0.32432432432432434,
0.3225806451612903,
0.32085561497326204,
0.324468085106383,
0.32275132275132273,
0.32105263157894737,
0.32460732984293195,
0.3229166666666667,
0.32124352331606215,
0.3247422680412371,
0.3230769230769231,
0.32142857142857145,
0.3197969543147208,
0.32323232323232326,
0.32160804020100503,
0.325,
0.3283582089552239,
0.32673267326732675,
0.3251231527093596,
0.3235294117647059,
0.32195121951219513,
0.32038834951456313,
0.3188405797101449,
0.3173076923076923,
0.32057416267942584,
0.319047619047619,
0.3222748815165877,
0.32075471698113206,
0.323943661971831,
0.32710280373831774,
0.32558139534883723,
0.3287037037037037,
0.3271889400921659,
0.3256880733944954,
0.3242009132420091,
0.32272727272727275,
```

```
0.3257918552036199,
0.32432432432432434,
0.32286995515695066,
0.32142857142857145,
0.32,
0.3185840707964602,
0.32158590308370044,
0.3201754385964912,
0.31877729257641924,
0.3173913043478261,
0.31601731601731603,
0.3146551724137931,
0.3133047210300429,
0.31196581196581197,
0.31063829787234043,
0.3093220338983051,
0.3080168776371308,
0.3067226890756303,
0.30962343096234307,
0.30833333333333335,
0.3112033195020747,
0.30991735537190085,
0.31275720164609055,
0.3114754098360656,
0.31020408163265306,
0.3089430894308943,
0.3076923076923077,
0.31048387096774194,
0.3092369477911647,
0.308,
0.30677290836653387,
0.3055555555555556,
0.30434782608695654,
0.30708661417322836,
0.30980392156862746,
0.30859375,
0.30739299610894943,
0.31007751937984496,
0.3088803088803089,
0.3076923076923077,
0.3065134099616858,
0.3053435114503817,
0.30798479087452474,
0.3106060606060606,
0.30943396226415093,
0.3082706766917293,
0.30711610486891383,
```

0.30970149253731344,
0.30855018587360594,
0.3111111111111111,
0.30996309963099633,
0.3125,
0.31135531135531136,
0.3102189781021898,
0.3090909090909091,
0.3079710144927536,
0.30685920577617326,
0.3057553956834532,
0.30824372759856633,
0.30714285714285716,
0.30604982206405695,
0.30851063829787234,
0.30742049469964666,
0.30985915492957744,
0.3087719298245614,
0.3076923076923077,
0.30662020905923343,
0.3055555555555556,
0.3079584775086505,
0.30689655172413793,
0.30927835051546393,
0.3082191780821918,
0.30716723549488056,
0.30612244897959184,
0.30847457627118646,
0.30743243243243246,
0.30976430976430974,
0.3087248322147651,
0.3110367892976589,
0.31,
0.3089700996677741,
0.31125827814569534,
0.31353135313531355,
0.3125,
0.31475409836065577,
0.3137254901960784,
0.31596091205211724,
0.3181818181818182,
0.32038834951456313,
0.3193548387096774,
0.3183279742765273,
0.32051282051282054,
0.3226837060702875,
0.321656050955414,

0.3238095238095238,
0.3227848101265823,
0.3217665615141956,
0.32075471698113206,
0.31974921630094044,
0.321875,
0.32087227414330216,
0.32298136645962733,
0.3219814241486068,
0.32098765432098764,
0.3230769230769231,
0.3220858895705521,
0.3241590214067278,
0.3231707317073171,
0.3221884498480243,
0.3212121212121212,
0.3202416918429003,
0.3192771084337349,
0.3213213213213213,
0.3203592814371258,
0.32238805970149254,
0.32142857142857145,
0.32047477744807124,
0.31952662721893493,
0.3185840707964602,
0.3176470588235294,
0.31671554252199413,
0.3157894736842105,
0.31486880466472306,
0.313953488372093,
0.3130434782608696,
0.31213872832369943,
0.31412103746397696,
0.3132183908045977,
0.3123209169054441,
0.31142857142857144,
0.31339031339031337,
0.3125,
0.311614730878187,
0.3107344632768362,
0.30985915492957744,
0.3089887640449438,
0.3081232492997199,
0.30726256983240224,
0.30919220055710306,
0.30833333333333335,
0.3074792243767313,

0.30662983425414364,
0.3085399449035813,
0.3076923076923077,
0.3095890410958904,
0.3087431693989071,
0.3079019073569482,
0.30978260869565216,
0.3089430894308943,
0.3108108108108108,
0.30997304582210244,
0.30913978494623656,
0.3109919571045576,
0.31283422459893045,
0.31466666666666665,
0.3138297872340425,
0.3129973474801061,
0.3148148148148148,
0.31398416886543534,
0.3131578947368421,
0.31496062992125984,
0.31413612565445026,
0.3133159268929504,
0.3125,
0.3142857142857143,
0.3134715025906736,
0.3152454780361757,
0.3170103092783505,
0.31876606683804626,
0.31794871794871793,
0.3171355498721228,
0.31887755102040816,
0.32061068702290074,
0.3197969543147208,
0.32151898734177214,
0.3207070707070707,
0.3198992443324937,
0.31909547738693467,
0.3182957393483709,
0.3175,
0.3167082294264339,
0.31840796019900497,
0.3200992555831266,
0.3193069306930693,
0.32098765432098764,
0.3226600985221675,
0.32432432432432434,
0.3235294117647059,

0.32273838630806845,
0.32195121951219513,
0.32116788321167883,
0.32038834951456313,
0.3196125907990315,
0.3188405797101449,
0.3180722891566265,
0.3173076923076923,
0.31654676258992803,
0.3181818181818182,
0.3198090692124105,
0.319047619047619,
0.3182897862232779,
0.3175355450236967,
0.31678486997635935,
0.3160377358490566,
0.31529411764705884,
0.3145539906103286,
0.31381733021077285,
0.3130841121495327,
0.3146853146853147,
0.313953488372093,
0.31554524361948955,
0.3148148148148148,
0.3163972286374134,
0.315668202764977,
0.31494252873563217,
0.31422018348623854,
0.3135011441647597,
0.3150684931506849,
0.3143507972665148,
0.3159090909090909,
0.31519274376417233,
0.3167420814479638,
0.3160270880361174,
0.3153153153153153,
0.3146067415730337,
0.31390134529147984,
0.3131991051454139,
0.3125,
0.31403118040089084,
0.31555555555555553,
0.3170731707317073,
0.3163716814159292,
0.31788079470198677,
0.31938325991189426,
0.31868131868131866,

0.31798245614035087,
0.3172866520787746,
0.3165938864628821,
0.31808278867102396,
0.31956521739130433,
0.3188720173535792,
0.3181818181818182,
0.3174946004319654,
0.3168103448275862,
0.3161290322580645,
0.315450643776824,
0.3169164882226981,
0.31837606837606836,
0.31769722814498935,
0.3191489361702128,
0.3184713375796178,
0.3199152542372881,
0.3192389006342495,
0.31856540084388185,
0.3178947368421053,
0.3172268907563025,
0.31865828092243187,
0.3179916317991632,
0.3173277661795407,
0.31875,
0.3180873180873181,
0.31742738589211617,
0.3167701863354037,
0.31611570247933884,
0.3154639175257732,
0.3168724279835391,
0.3162217659137577,
0.3155737704918033,
0.3149284253578732,
0.3142857142857143,
0.313645621181262,
0.3130081300813008,
0.31237322515212984,
0.3117408906882591,
0.3111111111111111,
0.31048387096774194,
0.3118712273641851,
0.3112449799196787,
0.3106212424849699,
0.31,
0.3093812375249501,
0.30876494023904383,

```
0.3101391650099404,
0.30952380952380953,
0.3089108910891089,
0.3102766798418972,
0.3096646942800789,
0.3090551181102362,
0.3104125736738703,
0.30980392156862746,
0.30919765166340507,
0.30859375,
0.30994152046783624,
0.311284046692607,
0.3106796116504854,
0.31007751937984496,
0.30947775628626695,
0.3088803088803089,
0.31021194605009633,
0.3096153846153846,
0.30902111324376197,
0.30842911877394635,
0.3078393881453155,
0.30725190839694655,
0.30857142857142855,
0.30798479087452474,
0.30740037950664134,
0.3068181818181818,
0.30623818525519847,
0.30566037735849055,
0.3069679849340866,
0.3082706766917293,
0.30956848030018763,
0.3089887640449438,
0.3102803738317757,
0.30970149253731344,
0.3091247672253259,
0.30855018587360594,
0.3079777365491651,
0.3074074074074074,
0.3068391866913124,
0.3062730627306273,
0.30570902394106814,
0.30514705882352944,
0.30458715596330277,
0.304029304029304,
0.30347349177330896,
0.30474452554744524,
0.30418943533697634,
```

0.30363636363636365,
0.30490018148820325,
0.30434782608695654,
0.3037974683544304,
0.30324909747292417,
0.3027027027027027,
0.302158273381295,
0.3016157989228007,
0.3010752688172043,
0.3005366726296959,
0.3,
0.30124777183600715,
0.30071174377224197,
0.30017761989342806,
0.299645390070922,
0.2991150442477876,
0.29858657243816256,
0.2980599647266314,
0.2975352112676056,
0.29701230228471004,
0.2964912280701754,
0.29772329246935203,
0.29895104895104896,
0.29842931937172773,
0.2979094076655052,
0.29739130434782607,
0.296875,
0.29809358752166376,
0.2975778546712803,
0.2970639032815199,
0.296551724137931,
0.29604130808950085,
0.29553264604810997,
0.2967409948542024,
0.2962328767123288,
0.29743589743589743,
0.29692832764505117,
0.29642248722316866,
0.29591836734693877,
0.29541595925297115,
0.29491525423728815,
0.2961082910321489,
0.2972972972972973,
0.29679595278246207,
0.2962962962962963,
0.29747899159663865,
0.29697986577181207,

```
 0.2964824120603015,
 0.2976588628762542,
 0.2988313856427379]
```

[13]:
```
matriz5 = matriz(60, array7)
matriz5
```

[13]:
```
array([[0.25      , 0.26229508, 0.27419355, …, 0.26495726, 0.27118644,
        0.26890756],
       [0.26229508, 0.27419355, 0.28571429, …, 0.27118644, 0.26890756,
        0.26666667],
       [0.27419355, 0.28571429, 0.296875  , …, 0.26890756, 0.26666667,
        0.26446281],
       …,
       [0.30855019, 0.30797774, 0.30740741, …, 0.29747899, 0.29697987,
        0.29648241],
       [0.30797774, 0.30740741, 0.30683919, …, 0.29697987, 0.29648241,
        0.29765886],
       [0.30740741, 0.30683919, 0.30627306, …, 0.29648241, 0.29765886,
        0.29883139]])
```

[14]:
```
matriz2.shape, matriz3.shape, matriz4.shape, matriz5.shape
```

[14]:
```
((541, 60), (541, 60), (481, 60), (481, 60))
```

[15]:
```
order = matriz2.shape[0] - matriz4.shape[0]
order
```

[15]:
```
60
```

[18]:
```
matrizfloat = matriz2[60:,:]
matrizint = matriz3[60:,:]
```

[19]:
```
matrizfloat.shape, matrizint.shape, matriz4.shape, matriz5.shape
```

[19]:
```
((481, 60), (481, 60), (481, 60), (481, 60))
```

[20]:
```
len(array2), len(array3), len(array4), len(array5), len(array7)
```

[20]:
```
(1672, 600, 600, 540, 540)
```

[21]:
```
x1 = matrizfloat[:,:(matrizfloat.shape[1] - 1)]
x2 = matriz4[:,:(matriz4.shape[1] - 1)]
x3 = matriz5[:,:(matriz5.shape[1] - 1)]

y = matrizint[:,-1]
```

[22]:
```
x1.shape, x2.shape, x3.shape, y.shape
```

[22]: ((481, 59), (481, 59), (481, 59), (481,))

[35]:
```
# Empilhar as matrizes para ter um eixo extra (60, 8, 3)
X_stack = np.stack([x1, x2, x3], axis=2)  # Formato (60, 8, 3)

# Reorganizar para intercalar coluna por coluna
X_intercalado = X_stack.reshape(59, -1)  # Agora está no formato (60, 24)

print(X_intercalado.shape)  # Saída: (60, 24)
```

```
(59, 1443)
```

[29]:
```
X_intercalado, y
```

[29]:
```
(array([[ 5.63      , 15.        ,  0.25      , …,  0.25641026,
          1.77      , 14.        ],
        [ 0.25316456,  1.02      , 13.        , …, 14.        ,
          0.25510204,  1.49      ],
        [14.        ,  0.25252525,  1.94      , …,  1.04      ,
         16.        ,  0.27118644],
        …,
        [ 1.6       , 13.        ,  0.30740741, …,  0.30107527,
          1.7       , 13.        ],
        [ 0.30053667,  6.        , 13.        , …, 12.        ,
          0.29757785,  1.12      ],
        [11.        ,  0.2970639 ,  2.36      , …,  3.22      ,
         11.        ,  0.29765886]]),
 array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
        0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
        0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
        0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
        0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
        0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
        0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
        0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
        1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
        1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
        1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
```

```
        0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0]))
```

[57]:
```
import pandas as pd
import numpy as np

# Garantir que são arrays NumPy
x1, x2, x3 = np.array(x1), np.array(x2), np.array(x3)

# Criar DataFrames separando cada coluna
df_x1 = pd.DataFrame(x1, columns=[f'X1_{i}' for i in range(x1.shape[1])])
df_x2 = pd.DataFrame(x2, columns=[f'X2_{i}' for i in range(x2.shape[1])])
df_x3 = pd.DataFrame(x3, columns=[f'X3_{i}' for i in range(x3.shape[1])])

# Juntar todas as colunas
X_df = pd.concat([df_x1, df_x2, df_x3], axis=1)

# Transformar para valores NumPy
X = X_df.values
```

[58]:
```
print(X.shape, y.shape)  # X deve ter o mesmo número de linhas de y
```

```
(481, 177) (481,)
```

[59]:
```
X
```

[59]:
```
array([[5.63      , 4.44      , 3.91      , …, 0.26724138, 0.26495726,
        0.27118644],
       [4.44      , 3.91      , 6.        , …, 0.26495726, 0.27118644,
        0.26890756],
       [3.91      , 6.        , 2.44      , …, 0.27118644, 0.26890756,
        0.26666667],
       …,
       [2.12      , 1.33      , 1.6       , …, 0.2962963 , 0.29747899,
        0.29697987],
       [1.33      , 1.6       , 1.06      , …, 0.29747899, 0.29697987,
        0.29648241],
       [1.6       , 1.06      , 1.05      , …, 0.29697987, 0.29648241,
        0.29765886]])
```

[34]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import accuracy_score, f1_score

# Dividir os dados em treino e teste
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Criar e treinar o modelo XGBoost
model = xgb.XGBClassifier(
    objective='multi:softmax',  # Classificação multiclasse
    num_class=2,  # Número de categorias na saída
    eval_metric='mlogloss',
    learning_rate=0.01,
    n_estimators=100,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)

model.fit(X_train, y_train)

# Fazer previsões
y_pred = model.predict(X_test)

# Avaliar o modelo
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')

print(f'Acurácia do modelo: {accuracy:.4f}')
print(f'F1-Score do modelo: {f1:.4f}')
```

```
Acurácia do modelo: 0.6701
F1-Score do modelo: 0.5631
```

[37]:

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.metrics import Precision, Recall
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow as tf
import tensorflow_addons as tfa
#activation = tf.keras.activations.elu
from tensorflow.keras.optimizers import Nadam

import skfuzzy as fuzz

# Libs
import time
import warnings
```

```
2025-03-04 17:30:25.719648: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
```

```
2025-03-04 17:30:25.836661: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-03-04 17:30:25.942496: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1741123826.052265   14195 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1741123826.077160   14195 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
2025-03-04 17:30:26.317913: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
/home/darkcover/Documentos/Out/venv/lib/python3.12/site-
packages/tensorflow_addons/utils/tfa_eol_msg.py:23: UserWarning:

TensorFlow Addons (TFA) has ended development and introduction of new features.
TFA has entered a minimal maintenance and release mode until a planned end of
life in May 2024.
Please modify downstream libraries to take dependencies from other repositories
in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).

For more information see: https://github.com/tensorflow/addons/issues/2807

  warnings.warn(
/home/darkcover/Documentos/Out/venv/lib/python3.12/site-
packages/tensorflow_addons/utils/ensure_tf_install.py:53: UserWarning:
Tensorflow Addons supports using Python ops for all Tensorflow versions above or
equal to 2.13.0 and strictly below 2.16.0 (nightly versions are not supported).
 The versions of TensorFlow you are currently using is 2.18.0 and is not
supported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration,
either change the TensorFlow version or the TensorFlow Addons's version.
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  warnings.warn(
```

[63]:
```
def reden(array1, array3, m, n):
    """
```

```
    Função para treinar uma rede neural usando as entradas e saídas fornecidas.

    Args:
        array1 (numpy.array): Saídas (rótulos) binárias (0 ou 1).
        array3 (numpy.array): Entradas preditoras.
        m (int): Número de amostras.
        n (int): Número de características por amostra.

    Returns:
        keras.Model: Modelo treinado.
    """
    # Dividindo os dados em treino e teste
    X = np.array(array3)
    y = np.array(array1)
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

    # Ajustando dimensões para entrada no modelo
    x_train = np.expand_dims(x_train, -1).astype("float32")
    x_test = np.expand_dims(x_test, -1).astype("float32")
    input_shape = (n , 1)  # Formato esperado de entrada

    # Convertendo saídas para categóricas
    num_classes = 2
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    # Definição do modelo
    model = keras.Sequential([
        keras.Input(shape=input_shape),
        layers.Flatten(),
        layers.Dense(128, activation="relu", use_bias=True),
        layers.Dropout(0.5),
        layers.Dense(64, activation="relu", use_bias=True),
        layers.Dense(32, activation=tf.keras.activations.swish, use_bias=True),
        layers.Dense(num_classes, activation="softmax"),
    ])

    model.compile(
        loss=tfa.losses.SigmoidFocalCrossEntropy(alpha=0.7, gamma=2.0),
        #loss=tfa.losses.SigmoidFocalCrossEntropy(alpha=0.25, gamma=2.0),␣
↪#testa loss = tfa.losses.SigmoidFocalCrossEntropy(alpha=0.25, gamma=2.0)
        optimizer=tf.keras.optimizers.AdamW(learning_rate=0.001,␣
↪weight_decay=1e-4),
        metrics=['accuracy', Precision(name="precision"), Recall(name="recall")]
    )
```

```python
    # Treinamento
    batch_size = 2**10
    epochs = 50
    model.fit(
        x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_split=0.2,
    )

    # Avaliação
    score = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test loss: {score[0]:.4f}")
    print(f"Test accuracy: {score[1]:.4f}")
    print(f"Precision: {score[2]:.4f}")
    print(f"Recall: {score[3]:.4f}")

    return model
```

[51]:
```python
y
```

[51]:
```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0])
```

[60]:
```python
X.shape
```

[60]:
```
(481, 177)
```

[43]:
```
np.unique(y)
```

[43]:
```
array([0, 1])
```

[64]:
```
model = reden(y, X, X.shape[0], X.shape[1])
```

```
Epoch 1/50
1/1              4s 4s/step -
accuracy: 0.5075 - loss: 2.7479 - precision: 0.5075 - recall: 0.5075 -
val_accuracy: 0.7059 - val_loss: 3.2853 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 2/50
1/1              0s 88ms/step -
accuracy: 0.6343 - loss: 2.8283 - precision: 0.6343 - recall: 0.6343 -
val_accuracy: 0.7059 - val_loss: 2.7800 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 3/50
1/1              0s 117ms/step -
accuracy: 0.6679 - loss: 2.3336 - precision: 0.6679 - recall: 0.6679 -
val_accuracy: 0.7059 - val_loss: 1.3745 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 4/50
1/1              0s 250ms/step -
accuracy: 0.5784 - loss: 1.9375 - precision: 0.5784 - recall: 0.5784 -
val_accuracy: 0.7353 - val_loss: 0.3892 - val_precision: 0.7353 - val_recall:
0.7353
Epoch 5/50
1/1              0s 83ms/step -
accuracy: 0.5448 - loss: 1.9259 - precision: 0.5448 - recall: 0.5448 -
val_accuracy: 0.5882 - val_loss: 0.3475 - val_precision: 0.5882 - val_recall:
0.5882
Epoch 6/50
1/1              0s 91ms/step -
accuracy: 0.4590 - loss: 2.2392 - precision: 0.4590 - recall: 0.4590 -
val_accuracy: 0.7353 - val_loss: 0.3811 - val_precision: 0.7353 - val_recall:
0.7353
Epoch 7/50
1/1              0s 92ms/step -
accuracy: 0.5075 - loss: 1.7750 - precision: 0.5075 - recall: 0.5075 -
val_accuracy: 0.7059 - val_loss: 0.6182 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 8/50
1/1              0s 91ms/step -
accuracy: 0.5448 - loss: 1.5854 - precision: 0.5448 - recall: 0.5448 -
val_accuracy: 0.7059 - val_loss: 0.8214 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 9/50
1/1              0s 104ms/step -
```

```
accuracy: 0.5858 - loss: 1.2033 - precision: 0.5858 - recall: 0.5858 -
val_accuracy: 0.7059 - val_loss: 0.8370 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 10/50
1/1            0s 121ms/step -
accuracy: 0.5709 - loss: 1.5434 - precision: 0.5709 - recall: 0.5709 -
val_accuracy: 0.7059 - val_loss: 0.6720 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 11/50
1/1            0s 100ms/step -
accuracy: 0.6194 - loss: 0.9267 - precision: 0.6194 - recall: 0.6194 -
val_accuracy: 0.7059 - val_loss: 0.4404 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 12/50
1/1            0s 110ms/step -
accuracy: 0.6157 - loss: 0.9634 - precision: 0.6157 - recall: 0.6157 -
val_accuracy: 0.6912 - val_loss: 0.2622 - val_precision: 0.6912 - val_recall:
0.6912
Epoch 13/50
1/1            0s 92ms/step -
accuracy: 0.5522 - loss: 0.8705 - precision: 0.5522 - recall: 0.5522 -
val_accuracy: 0.6471 - val_loss: 0.1913 - val_precision: 0.6471 - val_recall:
0.6471
Epoch 14/50
1/1            0s 101ms/step -
accuracy: 0.5821 - loss: 0.7536 - precision: 0.5821 - recall: 0.5821 -
val_accuracy: 0.6618 - val_loss: 0.1744 - val_precision: 0.6618 - val_recall:
0.6618
Epoch 15/50
1/1            0s 105ms/step -
accuracy: 0.5560 - loss: 0.7239 - precision: 0.5560 - recall: 0.5560 -
val_accuracy: 0.6176 - val_loss: 0.1778 - val_precision: 0.6176 - val_recall:
0.6176
Epoch 16/50
1/1            0s 97ms/step -
accuracy: 0.5112 - loss: 0.7424 - precision: 0.5112 - recall: 0.5112 -
val_accuracy: 0.6324 - val_loss: 0.1808 - val_precision: 0.6324 - val_recall:
0.6324
Epoch 17/50
1/1            0s 103ms/step -
accuracy: 0.5709 - loss: 0.5661 - precision: 0.5709 - recall: 0.5709 -
val_accuracy: 0.5882 - val_loss: 0.1812 - val_precision: 0.5882 - val_recall:
0.5882
Epoch 18/50
1/1            0s 97ms/step -
accuracy: 0.4925 - loss: 0.6641 - precision: 0.4925 - recall: 0.4925 -
val_accuracy: 0.6324 - val_loss: 0.1757 - val_precision: 0.6324 - val_recall:
0.6324
```

```
Epoch 19/50
1/1            0s 89ms/step -
accuracy: 0.5410 - loss: 0.5586 - precision: 0.5410 - recall: 0.5410 -
val_accuracy: 0.6912 - val_loss: 0.1702 - val_precision: 0.6912 - val_recall:
0.6912
Epoch 20/50
1/1            0s 92ms/step -
accuracy: 0.5672 - loss: 0.5508 - precision: 0.5672 - recall: 0.5672 -
val_accuracy: 0.6618 - val_loss: 0.1670 - val_precision: 0.6618 - val_recall:
0.6618
Epoch 21/50
1/1            0s 144ms/step -
accuracy: 0.5336 - loss: 0.4977 - precision: 0.5336 - recall: 0.5336 -
val_accuracy: 0.6324 - val_loss: 0.1642 - val_precision: 0.6324 - val_recall:
0.6324
Epoch 22/50
1/1            0s 90ms/step -
accuracy: 0.5821 - loss: 0.4447 - precision: 0.5821 - recall: 0.5821 -
val_accuracy: 0.6471 - val_loss: 0.1630 - val_precision: 0.6471 - val_recall:
0.6471
Epoch 23/50
1/1            0s 89ms/step -
accuracy: 0.5522 - loss: 0.4848 - precision: 0.5522 - recall: 0.5522 -
val_accuracy: 0.5735 - val_loss: 0.1663 - val_precision: 0.5735 - val_recall:
0.5735
Epoch 24/50
1/1            0s 92ms/step -
accuracy: 0.5672 - loss: 0.3844 - precision: 0.5672 - recall: 0.5672 -
val_accuracy: 0.6029 - val_loss: 0.1692 - val_precision: 0.6029 - val_recall:
0.6029
Epoch 25/50
1/1            0s 89ms/step -
accuracy: 0.5784 - loss: 0.3223 - precision: 0.5784 - recall: 0.5784 -
val_accuracy: 0.5882 - val_loss: 0.1699 - val_precision: 0.5882 - val_recall:
0.5882
Epoch 26/50
1/1            0s 219ms/step -
accuracy: 0.5858 - loss: 0.2961 - precision: 0.5858 - recall: 0.5858 -
val_accuracy: 0.5735 - val_loss: 0.1696 - val_precision: 0.5735 - val_recall:
0.5735
Epoch 27/50
1/1            0s 207ms/step -
accuracy: 0.5970 - loss: 0.3057 - precision: 0.5970 - recall: 0.5970 -
val_accuracy: 0.5882 - val_loss: 0.1684 - val_precision: 0.5882 - val_recall:
0.5882
Epoch 28/50
1/1            0s 108ms/step -
accuracy: 0.5299 - loss: 0.3247 - precision: 0.5299 - recall: 0.5299 -
```

```
val_accuracy: 0.6176 - val_loss: 0.1675 - val_precision: 0.6176 - val_recall:
0.6176
Epoch 29/50
1/1            0s 98ms/step -
accuracy: 0.5112 - loss: 0.3181 - precision: 0.5112 - recall: 0.5112 -
val_accuracy: 0.6765 - val_loss: 0.1658 - val_precision: 0.6765 - val_recall:
0.6765
Epoch 30/50
1/1            0s 86ms/step -
accuracy: 0.5112 - loss: 0.3256 - precision: 0.5112 - recall: 0.5112 -
val_accuracy: 0.6618 - val_loss: 0.1658 - val_precision: 0.6618 - val_recall:
0.6618
Epoch 31/50
1/1            0s 90ms/step -
accuracy: 0.5821 - loss: 0.2877 - precision: 0.5821 - recall: 0.5821 -
val_accuracy: 0.6765 - val_loss: 0.1662 - val_precision: 0.6765 - val_recall:
0.6765
Epoch 32/50
1/1            0s 87ms/step -
accuracy: 0.6082 - loss: 0.2704 - precision: 0.6082 - recall: 0.6082 -
val_accuracy: 0.7059 - val_loss: 0.1657 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 33/50
1/1            0s 90ms/step -
accuracy: 0.5821 - loss: 0.2879 - precision: 0.5821 - recall: 0.5821 -
val_accuracy: 0.7059 - val_loss: 0.1641 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 34/50
1/1            0s 97ms/step -
accuracy: 0.5821 - loss: 0.2549 - precision: 0.5821 - recall: 0.5821 -
val_accuracy: 0.7059 - val_loss: 0.1622 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 35/50
1/1            0s 88ms/step -
accuracy: 0.5634 - loss: 0.2293 - precision: 0.5634 - recall: 0.5634 -
val_accuracy: 0.7059 - val_loss: 0.1606 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 36/50
1/1            0s 92ms/step -
accuracy: 0.6157 - loss: 0.2348 - precision: 0.6157 - recall: 0.6157 -
val_accuracy: 0.7059 - val_loss: 0.1595 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 37/50
1/1            0s 90ms/step -
accuracy: 0.6007 - loss: 0.2304 - precision: 0.6007 - recall: 0.6007 -
val_accuracy: 0.7059 - val_loss: 0.1590 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 38/50
```

```
1/1            0s 103ms/step -
accuracy: 0.6045 - loss: 0.2132 - precision: 0.6045 - recall: 0.6045 -
val_accuracy: 0.7059 - val_loss: 0.1592 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 39/50
1/1            0s 88ms/step -
accuracy: 0.5709 - loss: 0.2319 - precision: 0.5709 - recall: 0.5709 -
val_accuracy: 0.6765 - val_loss: 0.1592 - val_precision: 0.6765 - val_recall:
0.6765
Epoch 40/50
1/1            0s 91ms/step -
accuracy: 0.5634 - loss: 0.2277 - precision: 0.5634 - recall: 0.5634 -
val_accuracy: 0.6912 - val_loss: 0.1596 - val_precision: 0.6912 - val_recall:
0.6912
Epoch 41/50
1/1            0s 89ms/step -
accuracy: 0.5522 - loss: 0.2155 - precision: 0.5522 - recall: 0.5522 -
val_accuracy: 0.7059 - val_loss: 0.1597 - val_precision: 0.7059 - val_recall:
0.7059
Epoch 42/50
1/1            0s 116ms/step -
accuracy: 0.6269 - loss: 0.1995 - precision: 0.6269 - recall: 0.6269 -
val_accuracy: 0.6912 - val_loss: 0.1594 - val_precision: 0.6912 - val_recall:
0.6912
Epoch 43/50
1/1            0s 90ms/step -
accuracy: 0.5448 - loss: 0.2233 - precision: 0.5448 - recall: 0.5448 -
val_accuracy: 0.6912 - val_loss: 0.1587 - val_precision: 0.6912 - val_recall:
0.6912
Epoch 44/50
1/1            0s 87ms/step -
accuracy: 0.5896 - loss: 0.1945 - precision: 0.5896 - recall: 0.5896 -
val_accuracy: 0.6618 - val_loss: 0.1582 - val_precision: 0.6618 - val_recall:
0.6618
Epoch 45/50
1/1            0s 88ms/step -
accuracy: 0.5560 - loss: 0.2082 - precision: 0.5560 - recall: 0.5560 -
val_accuracy: 0.6618 - val_loss: 0.1574 - val_precision: 0.6618 - val_recall:
0.6618
Epoch 46/50
1/1            0s 182ms/step -
accuracy: 0.5560 - loss: 0.2015 - precision: 0.5560 - recall: 0.5560 -
val_accuracy: 0.6618 - val_loss: 0.1567 - val_precision: 0.6618 - val_recall:
0.6618
Epoch 47/50
1/1            0s 284ms/step -
accuracy: 0.5858 - loss: 0.2015 - precision: 0.5858 - recall: 0.5858 -
val_accuracy: 0.6618 - val_loss: 0.1558 - val_precision: 0.6618 - val_recall:
```

```
0.6618
Epoch 48/50
1/1              0s 94ms/step -
accuracy: 0.6157 - loss: 0.1901 - precision: 0.6157 - recall: 0.6157 -
val_accuracy: 0.6765 - val_loss: 0.1549 - val_precision: 0.6765 - val_recall:
0.6765
Epoch 49/50
1/1              0s 91ms/step -
accuracy: 0.5933 - loss: 0.1797 - precision: 0.5933 - recall: 0.5933 -
val_accuracy: 0.6765 - val_loss: 0.1542 - val_precision: 0.6765 - val_recall:
0.6765
Epoch 50/50
1/1              0s 87ms/step -
accuracy: 0.5784 - loss: 0.2014 - precision: 0.5784 - recall: 0.5784 -
val_accuracy: 0.6912 - val_loss: 0.1536 - val_precision: 0.6912 - val_recall:
0.6912
Test loss: 0.1560
Test accuracy: 0.7172
Precision: 0.7172
Recall: 0.7172
```

[62]:
```
X.shape, y.shape
```

[62]:
```
((481, 177), (481,))
```