

ideia

April 6, 2025

## 1 Sequências aleatórias

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

import matplotlib.pyplot as plt
```

### 1.1 Função de Matriz Principal

Tomaremos um array de tamanho  $n$ , no qual  $n$  seja divisível por 60.

```
[3]: 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
```

#### 1.1.1 Teste 1

- Cria-se um array com entrada  $n = 1200$
- Transforma-se esse array em uma matriz com tamanho de colunas fixo  $\text{num\_colunas} = 60$

- Espera-se retorno de uma matriz com tamanho 1141x60

```
[4]: array_teste = np.arange(1, 1201)
      print("Tamanho array de teste:", len(array_teste))
      matriz_teste = matriz(60, array_teste)
      print(f'Shape matriz de teste:{matriz_teste.shape}')
```

Tamanho array de teste: 1200

Shape matriz de teste:(1141, 60)

## 1.2 Carregando dados para testes

Aqui carrega-se uma data especifica de algum de dia de coleta e realize-se o ajuste necessários.

```
[5]: data = pd.read_csv('/home/ozielramos/Documents/Out/dados/Saidas/FUNCOES/DOUBLE_
      ↪- 17_09_s1.csv')
```

```
[6]: data.head()
```

```
[6]:
```

	22,11	0,5	0,5.1	1	BET	Entrada	Odd	P60	P120	P180	...	P(1)	P(0)	\
0	0	22,11	-0,25	1	0	1,83	0	NaN	NaN	NaN	...	NaN	NaN	
1	1	22,11	-1	0	0	1,07	0	NaN	NaN	NaN	...	NaN	NaN	
2	2	22,11	1,75	2	0	24,83	1	NaN	NaN	NaN	...	NaN	NaN	
3	3	22,11	1,75	2	0	25,25	1	NaN	NaN	NaN	...	NaN	NaN	
4	4	22,11	1,75	2	0	8,55	1	NaN	NaN	NaN	...	NaN	NaN	

	LOG(P(1);2)	LOG(P(2);2)	Unnamed: 125	Unnamed: 126	Unnamed: 127	\
0	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	

	Unnamed: 128	Unnamed: 129	Unnamed: 130
0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	NaN	NaN	NaN

[5 rows x 131 columns]

```
[7]: array_data = []
      for i in range(1200):
          trip = data['Entrada'][i].replace(',', '.')
          if float(trip) >= 4:
              odd = 1
          else:
              odd = 0
```

```

    array_data.append(odd)
print("Tamanho array de teste:", len(array_data))
matriz_data = matriz(60, array_data)
print(f'Shape matriz de teste:{matriz_data.shape}')

```

Tamanho array de teste: 1200

Shape matriz de teste:(1141, 60)

### 1.3 Função para calcular o grau de entropia de um array

Para calcular o grau de entropia de um array e verificar o quão aleatório ele é, podemos usar o conceito de entropia de Shannon, que mede a incerteza ou imprevisibilidade dos dados. Quanto maior a entropia, mais aleatório é o array.

```

[8]: import numpy as np
from collections import Counter
import math

def calcular_entropia(array):
    # Conta a frequência de cada elemento no array
    contador = Counter(array)
    total_elementos = len(array)

    entropia = 0.0

    for count in contador.values():
        # Calcula a probabilidade de cada elemento
        probabilidade = count / total_elementos
        # Adiciona à entropia: -p * log2(p)
        entropia -= probabilidade * math.log2(probabilidade)

    return entropia

def verificar_aleatoriedade(array, limiar=0.7):
    """
    Verifica se um array é aleatório com base na entropia.

    Parâmetros:
    - array: Lista ou array numpy a ser analisado.
    - limiar: Valor entre 0 e 1 que define o quão alta a entropia deve ser para
    ser considerada aleatória.
    Quanto mais próximo de 1, mais aleatório o array precisa ser.

    Retorna:
    - True se a entropia for alta (acima do limiar), False caso contrário.
    - O valor da entropia normalizada (entre 0 e 1).
    """
    if len(array) == 0:

```

```

    return False, 0.0

entropia = calcular_entropia(array)

# Calcula a entropia máxima possível (log2 do número de elementos únicos)
elementos_unicos = len(set(array))
if elementos_unicos <= 1:
    return False, 0.0 # Sem variação, entropia zero

entropia_maxima = math.log2(elementos_unicos)
entropia_normalizada = entropia / entropia_maxima

# Verifica se a entropia está acima do limiar
return entropia_normalizada >= limiar, entropia_normalizada

```

### 1.3.1 Teste 2

Teste para verificar a entropia de um array

```

[9]: # Exemplo 1: Array com baixa entropia (não aleatório)
array1 = [1, 1, 1, 1, 1, 1, 1, 1]
resultado1, entropia_norm1 = verificar_aleatoriedade(array1)
print(f"Array1 é aleatório? {resultado1} (Entropia normalizada: {entropia_norm1:
↪.4f})")

# Exemplo 2: Array com alta entropia (aleatório)
array2 = np.random.randint(0, 256, 1000) # 1000 números aleatórios entre 0 e ↪
↪255
resultado2, entropia_norm2 = verificar_aleatoriedade(array2)
print(f"Array2 é aleatório? {resultado2} (Entropia normalizada: {entropia_norm2:
↪.4f})")

```

Array1 é aleatório? False (Entropia normalizada: 0.0000)

Array2 é aleatório? True (Entropia normalizada: 0.9814)

### 1.3.2 Explicação

#### 1. Cálculo da Entropia:

- A entropia de Shannon é calculada como  $-\sum p(x) \log_2 p(x)$ , onde  $p(x)$  é a probabilidade de um elemento  $x$  no array.
- Quanto mais uniforme a distribuição dos elementos, maior a entropia.

#### 2. Normalização:

- A entropia é normalizada pela entropia máxima possível (que ocorre quando todos os elementos são igualmente prováveis).
- Se  $entropia\_normalizada \approx 1$ , o array é considerado aleatório.

#### 3. Limiar (Threshold):

- Um limiar (padrão 0.8) é usado para decidir se o array é suficientemente aleatório. Você pode ajustar esse valor conforme necessário.

Esta função pode ser útil para verificar se um array contém padrões ou se é efetivamente aleatório. Quanto maior a entropia normalizada, mais imprevisível e aleatório é o array.

## 1.4 Junção das duas bibliotecas com a data teste

```
[10]: resultado_data, entropia_norm_data = verificar_aleatoriedade(array_data)
print(f"Array_data é aleatório? {resultado_data} (Entropia normalizada:
↪ {entropia_norm_data:.4f})")
```

Array\_data é aleatório? True (Entropia normalizada: 0.7937)

```
[11]: grafico_data = []
for i in range(matriz_data.shape[0]):
    resultado_data, entropia_norm_data = verificar_aleatoriedade(matriz_data[i])
    print(f"Array_data[{i}] é aleatório? {resultado_data} (Entropia normalizada:
↪ {entropia_norm_data:.4f})")
    grafico_data.append(entropia_norm_data)
print("Tamanho grafico_data:", len(grafico_data))
```

```
Array_data[0] é aleatório? False (Entropia normalizada: 0.6098)
Array_data[1] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[2] é aleatório? False (Entropia normalizada: 0.6873)
Array_data[3] é aleatório? False (Entropia normalizada: 0.6873)
Array_data[4] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[5] é aleatório? False (Entropia normalizada: 0.6098)
Array_data[6] é aleatório? False (Entropia normalizada: 0.6098)
Array_data[7] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[8] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[9] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[10] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[11] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[12] é aleatório? False (Entropia normalizada: 0.6873)
Array_data[13] é aleatório? True (Entropia normalizada: 0.7219)
Array_data[14] é aleatório? True (Entropia normalizada: 0.7540)
Array_data[15] é aleatório? True (Entropia normalizada: 0.7540)
Array_data[16] é aleatório? True (Entropia normalizada: 0.7838)
Array_data[17] é aleatório? True (Entropia normalizada: 0.7540)
Array_data[18] é aleatório? True (Entropia normalizada: 0.7838)
Array_data[19] é aleatório? True (Entropia normalizada: 0.7838)
Array_data[20] é aleatório? True (Entropia normalizada: 0.7838)
Array_data[21] é aleatório? True (Entropia normalizada: 0.7540)
Array_data[22] é aleatório? True (Entropia normalizada: 0.7838)
Array_data[23] é aleatório? True (Entropia normalizada: 0.7838)
Array_data[24] é aleatório? True (Entropia normalizada: 0.7838)
Array_data[25] é aleatório? True (Entropia normalizada: 0.8113)
Array_data[26] é aleatório? True (Entropia normalizada: 0.8113)
Array_data[27] é aleatório? True (Entropia normalizada: 0.8113)
Array_data[28] é aleatório? True (Entropia normalizada: 0.8113)
```

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

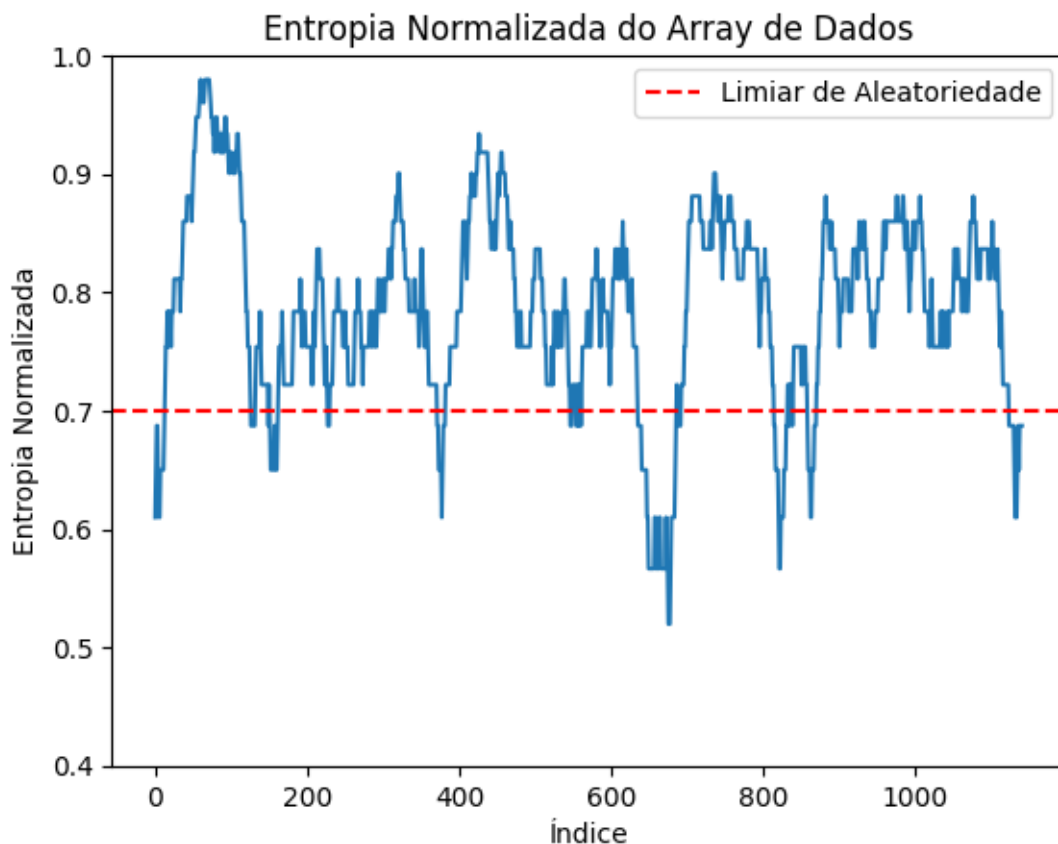
[illegible]

[illegible]

[illegible]

```
Array_data[1133] é aleatório? False (Entropia normalizada: 0.6098)
Array_data[1134] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[1135] é aleatório? False (Entropia normalizada: 0.6873)
Array_data[1136] é aleatório? False (Entropia normalizada: 0.6500)
Array_data[1137] é aleatório? False (Entropia normalizada: 0.6873)
Array_data[1138] é aleatório? False (Entropia normalizada: 0.6873)
Array_data[1139] é aleatório? False (Entropia normalizada: 0.6873)
Array_data[1140] é aleatório? False (Entropia normalizada: 0.6873)
Tamanho grafico_data: 1141
```

```
[12]: plt.plot(grafico_data)
plt.title('Entropia Normalizada do Array de Dados')
plt.xlabel('Índice')
plt.ylabel('Entropia Normalizada')
plt.ylim(0.4, 1)
plt.axhline(y=0.7, color='r', linestyle='--', label='Limiar de Aleatoriedade')
plt.legend()
plt.show()
```



## 1.5 Criação de Sequências aleatórias para treinamento

Elaboraremos uma função que gera um banco de dados com sequências criadas artificialmente que tenham sequências aleatórias, com grau acima de 0.5

```
[13]: import random
from math import log2

def calcular_entropia(seq):
    n = len(seq)
    p0 = seq.count(0) / n
    p1 = seq.count(1) / n

    entropia = 0
    if p0 > 0:
        entropia -= p0 * log2(p0)
    if p1 > 0:
        entropia -= p1 * log2(p1)
    return entropia

def gerador_sequencias(n, limite_entropia=0.5):
    while True:
        seq = [random.randint(0, 1) for _ in range(n)]
        if calcular_entropia(seq) > limite_entropia:
            yield seq

# Exemplo de uso: gerar infinitamente, mas parar manualmente
gerador = gerador_sequencias(60, limite_entropia=0.5)

# Exibir as 10 primeiras
for i in range(10):
    seq = next(gerador)
    print(f"{i+1}: {seq} (entropia = {calcular_entropia(seq):.3f})")
```

```
1: [0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 1, 0, 0, 1, 0, 0, 0] (entropia = 0.948)
2: [1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1,
0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
1, 1, 1, 1, 1, 0, 1, 0] (entropia = 0.980)
3: [0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0,
0, 0, 1, 0, 0, 1, 0, 0] (entropia = 0.971)
4: [1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0,
1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
1, 1, 0, 0, 1, 0, 1, 1] (entropia = 0.934)
5: [1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0] (entropia = 0.971)
```

```

6: [1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0] (entropia = 0.987)
7: [1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1,
0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1] (entropia = 0.960)
8: [1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1,
0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1] (entropia = 0.987)
9: [0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1,
1, 0, 1, 0, 0, 0, 0, 1] (entropia = 1.000)
10: [0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0,
0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0,
1, 1, 0, 1, 0, 1, 1, 0] (entropia = 0.993)

```

## 1.6 Teste em produção

Testar e combinar a produção do código

```

[ ]: data_matriz_possivel = []
for i in range(len(array_data)):
    if i >= 360 and (i % 60) == 0:
        gerador = gerador_sequencias(60, limite_entropia=0.5)
        for j in range(10*6):
            seq = next(gerador)
            entropia = calcular_entropia(seq)
            if entropia >= 0.7:
                array_artificial = np.concatenate((array_data[:i], seq))
                media_artificial = np.mean(array_artificial)
                if media_artificial >= 0.15 and media_artificial <= 0.35:
                    matriz_teste1 = matriz(60, array_data[:i])
                    data_matriz_possivel.append(matriz_teste1)
        break
print(f'Shape matriz artificial:{data_matriz_possivel.shape}')

```

## 2 Conclusão

A geração de sequências artificiais é inviável até o momento por motivos computacionais, isto é, precisamos de um aporte maior em relação a isto.