

## Relatório TI TP1

### Entropia, Redundância e Informação Mútua



Trabalho realizado por:

- Leonardo Pina, 2019234318
- Luís Neto, 2020215474
- João Moura, 2020235800

## Conteúdo

Introdução .....	3
Exercício 1 .....	3
Exercício 2 .....	3
Exercício 3 .....	3
Análise da entropia/histograma das imagens.....	6
Análise da entropia/histograma da fonte áudio .....	6
Análise da entropia/histograma da fonte de texto.....	6
-Será possível comprimir cada uma das fontes de forma não destrutiva?.....	6
Exercício 4 .....	7
Análise da variância obtida para as diferentes fontes .....	7
-Será possível reduzir-se a variância? .....	8
Exercício 5 .....	8
Exercício 6.a).....	9
Exercício 6.b) .....	9
Análise dos resultados.....	10
Exercício 6.c).....	10
Análise dos resultados.....	11
Apêndices .....	12
Funções .....	12
Função oco .....	12
Função histograma.....	13
Função entropia .....	13
Função reader .....	14
Função bitmediosimb.....	15
Função entropia_conj .....	16
Função shazam .....	17
Conclusão .....	19

## Introdução

O presente relatório tem como objetivo a análise dos resultados obtidos nos exercícios da ficha TP1, assim como a respetiva análise e explicação das funções de código que auxiliaram às suas resoluções. Desta forma, o trabalho realizado introduziu-nos a conceitos como a entropia, a informação mútua, entre outros, sendo aplicados através do uso da linguagem *Python*. No decorrer deste processo, aprendemos mais sobre a cadeira, a relação entre os conceitos teóricos e os elementos informáticos e ainda sobre a importância e eficiência do trabalho de grupo.

**NOTA:** Toda a informação relativa a informações utilizadas estará discriminada no apêndice

## Exercício 1

Para este exercício foi escrita a função  $oco(P,a,flag)$  e  $histograma(P,a)$  (consultar apêndice para mais detalhes).

## Exercício 2

Para este exercício foi escrita a função  $entropia(oc,P)$  (consultar apêndice para mais detalhes)

## Exercício 3 (apresentação e análise de resultados obtidos):

- *Kid.bmp*:

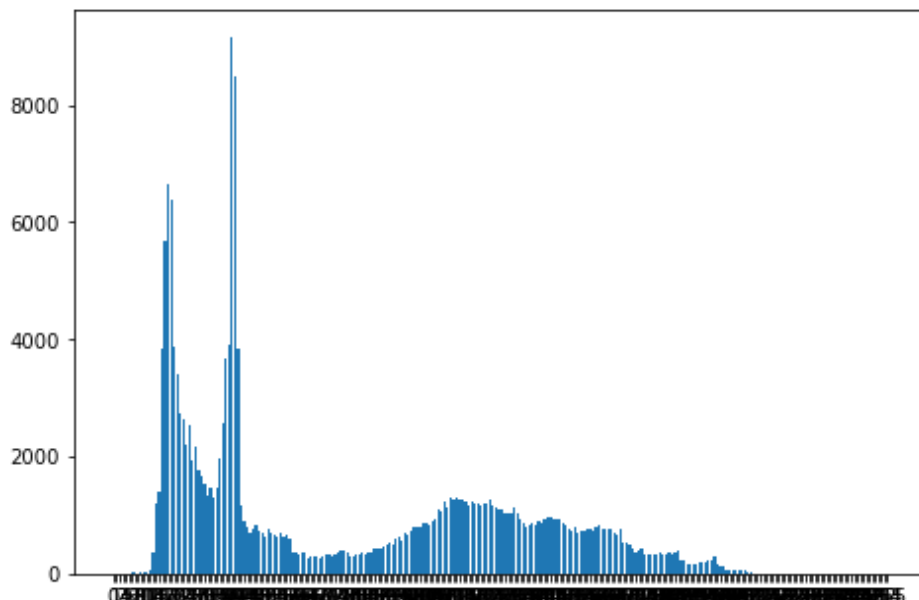


Fig. 1 – histograma de ocorrências de *kid.bmp*

A figura 1 representa a distribuição estatística (histograma) obtido a partir da fonte *kid.bmp*, ou seja, estuda as ocorrências de cada símbolo existente na fonte

fornecida. Assim, foi calculada a entropia (número médio de bits por símbolo) desta fonte, tendo sido obtido um resultado de aproximadamente 6,954 bits/símbolo.

- *Homer.bmp*:



Fig. 2 - histograma de ocorrências de *homer.bmp*

A figura 2 representa a distribuição estatística (histograma) obtido a partir da fonte *homer.bmp*, ou seja, estuda as ocorrências de cada símbolo existente na fonte fornecida. Assim, foi calculada a entropia (número médio de bits por símbolo) desta fonte, tendo sido obtido um resultado de aproximadamente 3,466 bits/símbolo.

- *HomerBin.bmp*:

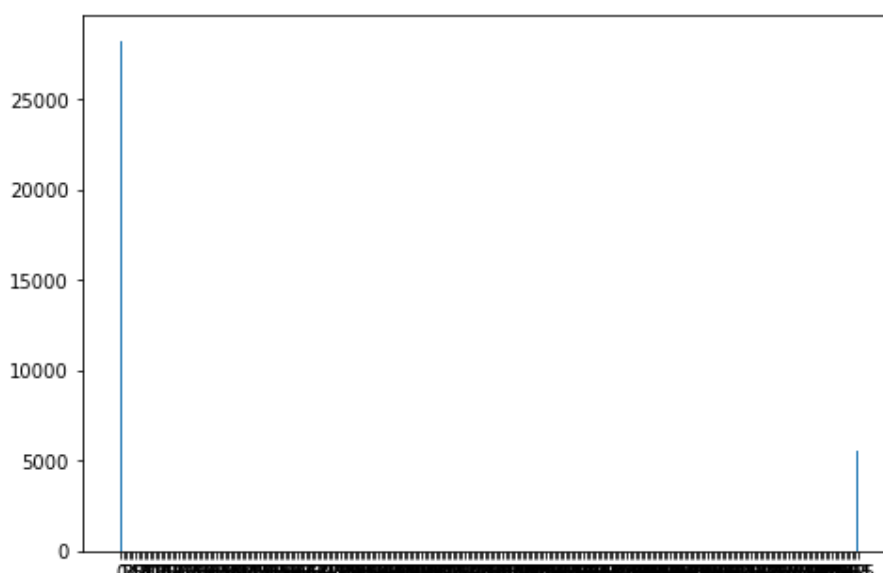


Fig. 3 - histograma de ocorrências de *homerBin.bmp*

A figura 3 representa a distribuição estatística (histograma) obtido a partir da fonte *homerBin.bmp*, ou seja, estuda as ocorrências de cada símbolo existente na

fonte fornecida. Assim, foi calculada a entropia (número médio de bits por símbolo) desta fonte, tendo sido obtido um resultado de aproximadamente 0,645 bits/símbolo.

- *GuitarSolo.wav*:

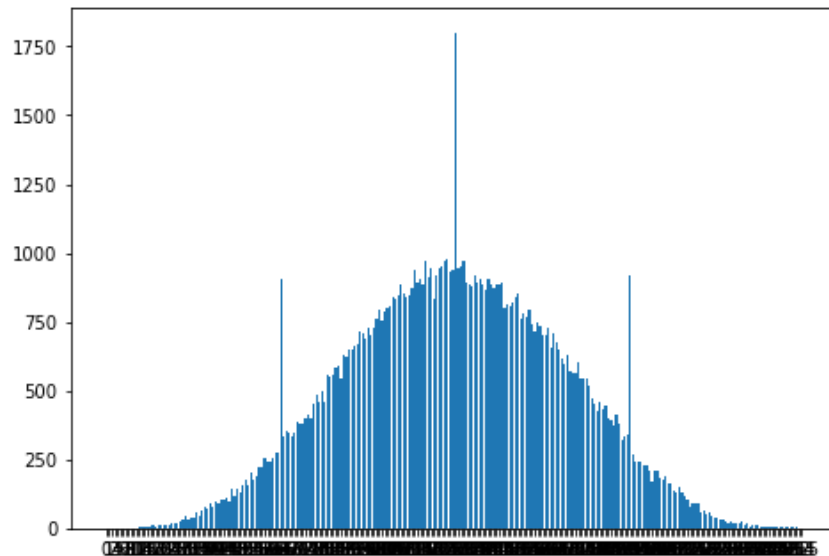


Fig. 4 - histograma de ocorrências de *GuitarSolo.wav*

A figura 4 representa a distribuição estatística (histograma) obtido a partir da fonte *GuitarSolo.wav*, ou seja, estuda as ocorrências de cada símbolo existente na fonte fornecida. Assim, foi calculada a entropia (número médio de bits por símbolo) desta fonte, tendo sido obtido um resultado de aproximadamente 7,329 bits/símbolo.

- *English.txt*:

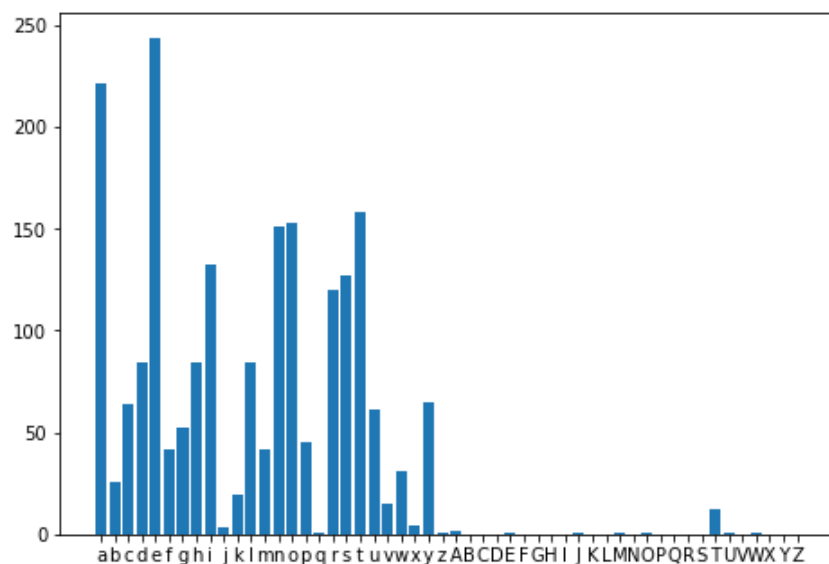


Fig. 5 - histograma de ocorrências de *English.txt*

A figura 5 representa a distribuição estatística (histograma) obtido a partir da fonte *English.txt*, ou seja, estuda as ocorrências de cada símbolo existente na fonte

fornecida. Assim, também foi calculada a entropia (número médio de bits por símbolo) desta fonte, tendo sido obtido um resultado de aproximadamente 4.228 bits/símbolo.

### Análise da entropia/histograma das imagens:

Tanto os histogramas como os valores da entropia se apresentam dentro do teoricamente previsto. No caso das imagens, embora todas sejam monocromáticas, apresentam variações na tonalidade, ou seja, uma dispersão diferente no espectro de RGB. Isto traduz-se numa maior entropia nas fontes com maior dispersão (maior variação no RGB) e consequentemente menor nas restantes. Por exemplo, a fonte *kid.bmp* apresenta a maior entropia entre todas as imagens analisadas, uma vez que é a que apresenta maior variação nas tonalidades de cor, que se traduz numa maior variação no espectro de RGB. A fonte *homer.bmp* apresenta menor dispersão, sendo que a frequência é concentrada num menor intervalo de valores. A fonte *homerBin.bmp* é concentrada apenas em dois valores, dado que é uma imagem binária, isto traduz-se numa dispersão reduzida e consequentemente na menor entropia entre as imagens analisadas.

### Análise da entropia/histograma da fonte áudio:

O valor da entropia apresenta-se dentro do teoricamente previsto. Analisando o histograma, vemos uma dispersão considerável dos valores o que justifica o valor relativamente elevado da entropia.

### Análise da entropia/histograma da fonte de texto:

De novo, valores teoricamente previstos. Apesar do histograma apresentar alguma dispersão, esta é apenas mais visível dentro do campo das letras minúsculas, mas que se encontra de acordo com o previsto, dado que um texto tem normalmente mais letras minúsculas que maiúsculas.

**-Será possível comprimir cada uma das fontes de forma não destrutiva?**

**-Se Sim, qual a compressão máxima que se consegue alcançar?**

Para comprimir as fontes de forma a não ocorrerem perdas de dados, ou seja, de uma forma não destrutiva, a entropia obtida não deve corresponder ao seu valor máximo. De acordo com os materiais teóricos, sabemos que a entropia é um valor positivo e no máximo igual ao logaritmo de base 2 do tamanho do alfabeto. No caso de fontes imagem e áudio, o alfabeto tem tamanho 256, logo a entropia máxima é igual a 8, no caso de fontes de texto, o tamanho do alfabeto é 52, logo a entropia máxima é aproximadamente 5,70. Visto que em nenhum dos casos analisados a entropia é igual ou superior ao maior valor possível, então é possível comprimir as fontes de forma não destrutiva. Dos materiais teóricos sabemos ainda que a taxa de compressão máxima é dada por:

$$\text{Taxa de compressão máxima} = \frac{\text{Entropia máxima} - \text{Entropia}}{\text{Entropia máxima}} \times 100$$

Fig. 6 - Fórmula da taxa de compressão máxima

Os valores de compressão máxima são:

**kid.bmp->13.08%**

**homer.bmp->56.68%**

**homerBin.bmp->91.94%**

**guitarSolo.wav->8.39%**

**english.txt->25.82%**

#### Exercício 4 (apresentação e análise de resultados obtidos):

Neste exercício, através do auxílio de funções de Huffman fornecidas, calculou-se o número médio de bits por símbolo de cada fonte.

**Número médio de bits por símbolo:**

- kid.bmp=6.983
- homer.bmp=3.548
- homerBin.bmp=1.0
- guitarSolo.wav=7.350
- english.txt=4.252

Assim, os valores do número médio de bits por símbolo (entropia) obtidos neste exercício, revelam-se de facto superiores aos valores calculados na alínea anterior.

$$V(X) = E(X^2) - E(X)^2$$

Fig. 6- Fórmula da variância

**Cálculo da variância:**

- kid.bmp=2,099
- homer.bmp=13,197
- homerBin.bmp=0,0
- guitarSolo.wav=0,727
- english.txt=1,191

Análise da variância obtida para as diferentes fontes:

A variância calculada é uma análise ao tamanho do código de codificação criado pela função de *Huffman*.

Desta forma, a variância será tanto maior quanto mais dispersos forem os comprimentos dos códigos de codificação, ou seja, caso estes tenham tamanho variado, e o inverso quando o comprimento é semelhante.

Assim, começando pelas imagens, seria expectável que a imagem *homerBin.bmp* tivesse uma variância igual 0, por ser possível codificar a fonte com códigos de um único tamanho (comprimento igual a 1). No que toca às imagens, seria expectável que a imagem *homer.bmp* apresentasse a variância mais alta devido aos diversos valores encontrados nos seus dados, sendo o que efetivamente se verificou. Por fim, a imagem *kid.bmp* apresenta uma variância baixa dado que os códigos gerados, não apresentam tamanhos muito diferentes.

Ao mesmo tempo, o *guitarSolo.wav* apresenta uma variância consideravelmente baixa.

No ficheiro *english.txt*, os dados encontram-se mais concentrados nos valores das letras minúsculas, variando mais quando ocorrem letras maiúsculas. Assim sendo, é previsível que este tenha uma variância relativamente baixa.

-Será possível reduzir-se a variância?

**-Se Sim, como pode ser feito e em que circunstância será útil?**

Não, para se obter a variância mínima, é necessário ordenar pela ordem mais elevada possível os símbolos da lista. Contudo, as funções que realizamos para este exercício foram criadas de modo a obter a variância mínima, sendo que no nosso caso em específico, não é possível obter valores menores de variância.

## Exercício 5 (apresentação e análise de resultados obtidos):

Neste exercício, calculámos a entropia conjunta de cada uma das fontes, ou seja, repetiu-se o mesmo processo do exercício 3, mas agora pretendemos calcular a entropia aplicando agrupamentos de símbolos, isto é, admitindo que cada símbolo será composto por uma sequência de dois contíguos.

**Valores da entropia conjunta:**

- **kid.bmp=4,909**
- **homer.bmp=2,413**
- **homerBin.bmp=0,398**
- **guitarSolo.wav=5,754**
- **english.txt=3,652**

Assim, note-se desde já que seria expectável que nos deparássemos com variações nos valores da entropia simples e agrupada, variações estas que foram mais tarde comprovadas com dados concretos, após o cálculo da nova entropia.

Concomitantemente, o cálculo desta nova entropia é feito da mesma maneira para as fontes de informação com e sem agrupamento de símbolos, tendo apenas como única



diferença a divisão pelo número de elementos emparelhados (neste caso 2). Seria então previsível que esta fosse menor que a entropia calculada anteriormente (no exercício 3), como podemos verificar ao realizar a comparação entre os valores agora obtidos e os valores obtidos anteriormente no exercício 3. Isto deve-se a já não estarmos à procura de ocorrências isoladas, mas sim de padrões, sendo que estes são muito mais frequentes, isto é, têm uma dispersão menor.

Concluimos assim que, quando os símbolos se encontram agrupados, o valor da entropia diminui devido à redução da dispersão no espetro da fonte.

### Exercício 6.a)

Nesta alínea, temos como objetivo a identificação da informação mútua, sendo neste caso um *target* e uma *query* que nos foram fornecidos como valores de teste pelo enunciado do trabalho. Para podermos proceder à identificação da informação mútua, é necessário a utilização do conceito de janela flutuante, na qual a *query* vai sendo comparada a intervalos do mesmo tamanho dentro do *target*, com um passo que neste caso será de 1, de modo a procurar pelos valores de informação mútua. Verificou-se que os valores obtidos em relação à informação mútua (*array* de valores) pelo programa, foram semelhantes aos valores indicados no enunciado.

### Exercício 6.b) (apresentação e análise de resultados obtidos):

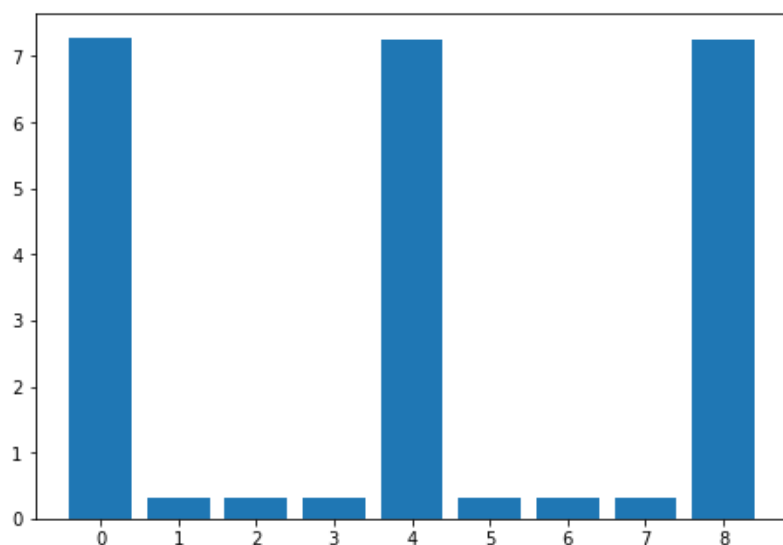


Fig. 7 – histograma da informação mútua de *guitarSolo.wav* com *target01 - repeat.wav*

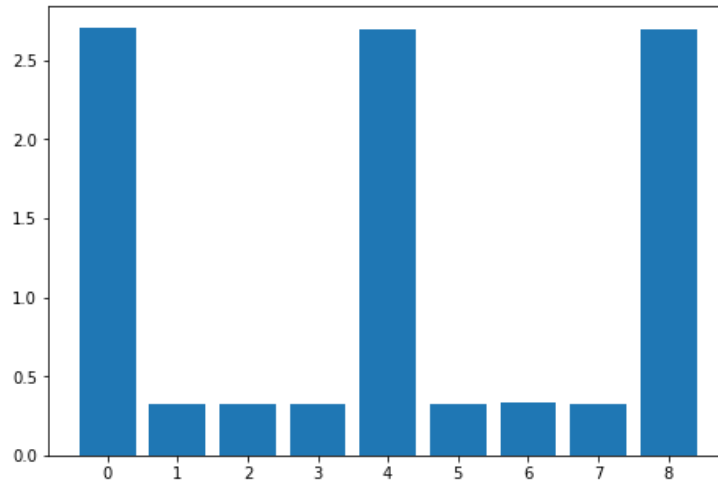


Fig 8 – histograma da informação mútua de *guitarSolo.wav* com *target02 - repeat.wav*

Ouvindo os ficheiros *guitarSolo.wav* e *target01 - repeat.wav*, *target02 - repeat.wav*, é possível concluir que em ambas as fontes *target* são *loops* do *guitarSolo.wav*, sendo que o *target02* é uma “versão” mais ruidosa deste.

Assim, num processo análogo ao do exercício 6.a), a única grande diferença é o uso de faixas de áudio ao invés de *arrays* numéricos, sendo que como é necessária a leitura destas é mais fácil observar-se o conceito de informação mútua.

Desta forma, conseguimos inferir que quanto mais parecido o áudio emitido por cada faixa, mais próximo se encontra o valor da informação mútua do valor da entropia (do *query* escolhido) e menos ruído e outras distorções existem.

### Análise dos resultados:

Como referido antes, no que toca à similaridade dos ficheiros de áudio, é bastante natural que a primeira janela de análise apresente uma informação mútua elevada, mas ainda assim um pouco abaixo da entropia calculada para o ficheiro de áudio que se encontra na *query*.

As janelas seguintes não apresentam uma informação mútua tão elevada pois estas têm um ponto inicial em outra parte do áudio que não o início, ou seja, as janelas estão desfasadas uma da outra. Quanto ao segundo áudio, este trata-se de uma cópia do áudio *target01 - repeat* mas com ruído em algumas das repetições. Assim, ao contrário do primeiro áudio, a primeira janela de análise apresenta uma informação mútua baixa devido ao ruído do som.

Nas restantes janelas verificamos que a informação mútua é relativamente próxima de zero, pelo mesmo motivo apresentado para o primeiro áudio (janelas em comparação estarem desfasadas).

### Exercício 6.c) (apresentação e análise de resultados obtidos):

Neste exercício, à semelhança das alíneas 6.a) e 6.b), pretendemos comparar desta vez utilizando vários ficheiros de áudio como target (*Song\*.wav*). Desta forma, repetiremos mais uma vez o processo realizado nas alíneas anteriores.

Após a recolha dos valores da informação mútua (*array* de valores), apenas guardamos num novo *array* criado os valores mais elevados, em conjunto com o índice do target em análise. De seguida, utilizou-se a função do *python* *.sort(reverse = True)* de forma a organizar o *array* por ordem decrescente.

#### Valores da informação:

- **Informação mútua máxima de song01:**  
0.25157649663930925
- **Informação mútua máxima de song02:**  
0.36729409269530855
- **Informação mútua máxima de song03:**  
0.29673143378946065
- **Informação mútua máxima de song04:**  
0.39806329767864845
- **Informação mútua máxima de song05:**  
3.9598751768031555
- **Informação mútua máxima de song06:**  
7.3095203231987504
- **Informação mútua máxima de song07:**  
6.296821392318373

#### Análise dos resultados:

Após obtermos os valores e ouvirmos os ficheiros de áudio, conseguimos averiguar que os ficheiros *Song07.wav*, *Song05.wav* e maioritariamente *Song06.wav* apresentam maior similaridade com a nossa *query* (*guitarSolo.wav*).

Como os restantes valores de informação mútua são extremamente reduzidos, podemos concluir que os targets *Song01.wav* a *Song04.wav* não apresentam grandes semelhanças com a *query*.

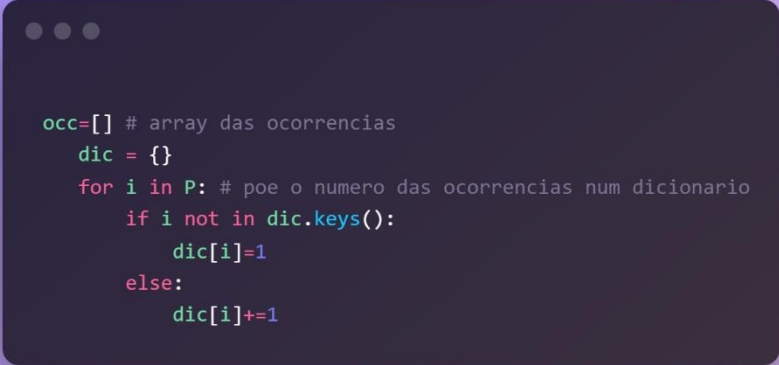
## Apêndices

Nesta secção encontram-se apêndices relacionados com a componente técnica do código

## Funções

### Função oco(P,a,flag):

Esta função é extremamente relevante para o funcionamento das diversas tarefas que o programa deve cumprir, sendo por sua vez reutilizada várias vezes ao longo do mesmo. Esta função consoante uma dada fonte de informação (texto, imagem, áudio, etc..) vai analisar a matriz/array que lhe está associada e contabilizar as ocorrências de cada elemento do alfabeto colocando a informação organizada num dicionário.



```
occ=[] # array das ocorrencias
dic = {}
for i in P: # poe o numero das ocorrencias num dicionario
    if i not in dic.keys():
        dic[i]=1
    else:
        dic[i]+=1
```

Fig. 9- Organização das ocorrências em dicionário

Esta função serve-se também de uma variável inteira “flag” para determinar se devolve um array (posteriormente convertido em array do *numPy*) ou o dicionário, dado que qualquer uma das estruturas de dados será necessária no desempenho de diferentes tarefas.

```

for i in a: # constroi o array das ocorrencias pretendido
    if i not in dic.keys():
        occ += [0]
    else:
        occ+=dic[i]

if flag == 1: # devolve dicionario sendo as chaves os elementos e os valores as
respetivas ocorrencias
    return dic

occ=np.array(occ)

return occ # devolve array das respetivas ocorrencias

```

Fig. 10-Devolução de diferentes tipos de estruturas

### Função histograma (**P**, **a**):

Esta função é responsável por gerar os gráficos relativos à ocorrência de cada elemento do alfabeto, servindo-se não só da função `oco(P,a,flag)` como de algumas funções da biblioteca *matplotlib* que geram as diversas componentes gráficas.

```

def histograma(P, a): # funcao do histograma / P - fonte a ler, a - alfabeto
    occ=oco(P,a,0) # devolve array das respetivas ocorrencias
    fig = plt.figure()
    ax = fig.add_axes([0,0,1,1])
    a = np.array(a)

    nelem = a.shape[0]
    x = np.arange(0, nelem, 1,int)
    plt.xticks(x, a)
    ax.bar(x, occ)
    plt.show()

```

Fig.11-Função histograma (P, a)

### Função entropia(**oc,P**):

Esta função é responsável pelo cálculo da entropia (limite mínimo teórico para o número médio de bits por símbolo) para uma dada fonte de informação. Recebe um *array* de

ocorrências e a respetiva fonte procedendo ao cálculo da probabilidade de cada um dos elementos da fonte.

```
n_elem = len(P)

if n_elem <= 1:
    return 0

counts = oc
probs = counts[np.nonzero(counts)] / n_elem # índices de todos os numeros != 0
n_prob = len(probs)
```

Fig. 12- Cálculo de probabilidades

De seguida aplica a fórmula da entropia:

$$H(A) = \sum_{i=1}^n P(a_i) \log_2 \frac{1}{P(a_i)} = - \sum_{i=1}^n P(a_i) \log_2 P(a_i)$$

Fig. 13-Fórmula da entropia

```
soma = 0
for i in probs:
    soma -= i * np.log2(i)

return soma # devolve valor da entropia
```

Fig. 14-Respetiva aplicação em código da fórmula da entropia

Função reader():

Esta função é responsável pela abertura de ficheiros de texto e a passagem dos seus caracteres para um *array*.



```
def reader(): # guarda num array apenas as letras do texto
    english = open('C:/Users/leona/OneDrive/Ambiente de Trabalho/II (2 ano)/TP1
data/english.txt', 'r')
    english_txt = english.read()
    eng = []

    for letra in english_txt:
        if letra.isalpha() == True:
            eng += [letra]
    english_array = np.asarray(eng)
    english.close()
    return english_array # devolve o array
```

Fig. 15- Função reader()

### Função bitmediosimb(**oc, s, l**):

Esta função recebe um dicionário de ocorrências, um *array* com cada símbolo presente numa dada fonte (“s”) e um *array* com o comprimento de cada elemento (“l”). Esta função vai então calcular o número de bits médio por cada símbolo da fonte e a sua variância, aplicando a seguinte fórmula:

$$V(X) = E(X^2) - E(X)^2$$

Fig 16- Fórmula da entropia

```

media_p = 0
media_p_var = 0

e=0
for i in s: #range(len(l))
    media_p += (l[e] * oc[i]) #oc e dicionario
    media_p_var += ((l[e]**2) * oc[i])
    e+=1

soma_oc=0
for i in oc.values():
    soma_oc += i

media_p = media_p / soma_oc

media_p_var = media_p_var / soma_oc

```

Fig. 17- Cálculo de  $E(X)$  e  $E(X^2)$

```

var = media_p_var - (media_p**2)
print("var:")
print(var)

```

Fig. 18- Cálculo da variância

Função entropia\_conj(**P**, **a=None**):



Esta função, tal como a `entropia(oc,P)`, calcula a entropia de uma dada fonte sendo a única diferença que esta vai ser calculada para conjuntos 2 a 2 de símbolos da fonte original.



```
fonte1 = []
for i in range(0, len(P) - 1, 2):
    fonte1.append(str(P[i]) + "/" + str(P[i + 1]))
```

Fig. 19- Conversão da fonte para uma com agrupamentos de símbolos 2 a 2

### Função `shazam(query, target, a, passo)`:

Esta função é responsável pela determinação da informação mútua entre uma dada *query* em relação a um *target*. Para concretizar este objetivo vai inicialmente calcular as ocorrências e entropia da *query* e gerar um novo *array* que contenha os elementos da *query* e o *target* alternados, dentro do comprimento da janela deslizante para posteriormente se calcular a entropia conjunta.



```
while(k <= len(target) - len(query)):
    i = 0
    for i in range(len(query)):
        lista_c += [query[i]] + [target[k+i]]
```

Fig. 20- Criação da janela deslizante conjunta

Posteriormente calcula-se as ocorrências e entropia da janela deslizante do *target* e procede-se ao cálculo da informação mútua, a partir desta fórmula:

$$I(X_1; X_2) = H(X_1) + H(X_2) - H(X_1, X_2)$$

Fig. 21-Fórmula Informação Mútua

```
target_oc = oco(target[k:len(query)+k], a, 0)
target_entropia = entropia (np.array(target_oc), np.array(target[k:len(query)+k]))

inf_mutua = query_entropia + target_entropia - entropia_conj(lista_c, a)
lista_inf_mutua+=[inf_mutua]
k+=passo

lista_c=[]
return lista_inf_mutua
```

Fig. 22- Aplicação da fórmula em código

### Função grafico(P, a):

Função utilizada para gerar um gráfico de barras com o valor de cada elemento de um alfabeto numa determinada fonte. Utiliza funções da biblioteca *matplotlib*.

```
def grafico(P, a): # funcao do histograma / P - fonte a ler, a - alfabeto
    fig = plt.figure()
    ax = fig.add_axes([0,0,1,1])
    a = np.array(a)
    nelem = a.shape[0]
    x = np.arange(0, nelem, 1,int)
    plt.xticks(x, a)
    ax.bar(x, P)
    plt.show()
```

Fig.23- Função grafico(P,a)

## Conclusão

Concluimos que, ao longo da realização deste trabalho, aprendemos mais acerca da relação entre conceitos como a entropia, entre outros, e a informática, bem como as suas aplicações num contexto de programação, neste caso, utilizando a linguagem *python*. Ao mesmo tempo, verificou-se que os resultados obtidos ao longo do trabalho foram considerados fiáveis, levando a que os objetivos propostos nas aulas tenham sido então devidamente alcançados.