



UNIVERSIDADE DE
COIMBRA

Trabalho prático Nº 1

Entropia, redundância e informação mútua

Trabalho realizado por:

- Pedro Afonso Ferreira Lopes Martins n.º 2019216826
- João Carlos Borges Silva n.º 2019216753
- Rafael Teixeira da Silva n.º 2019219259

Índice

Introdução	3
Exercício 1	4
Exercício 2	5
Exercício 3	6
Exercício 4	14
Exercício 5	16
Exercício 6	19
Conclusão.....	24

Introdução:

Este trabalho consiste na utilização de Python e de diversas bibliotecas do mesmo (Ex: matplotlib, numpy, sounddevice) para realizar uma introdução à cadeira da Teoria da Informação e assim solidificar conhecimentos necessários para o nosso futuro.

Recorrendo às bibliotecas acima mencionadas conseguimos realizar o estudo e tratamento de dados de diversos ficheiros de texto, imagem e som, usando o conceito da entropia (limite mínimo teórico para o número médio de bits por símbolo), a qual pode ser obtida através da seguinte fórmula:

$$H(K) = - \sum_i^n P(K_i) \log_2 P(K_i)$$

Onde:

- K_i é um símbolo do alfabeto a analisar.
- $H(K)$ é a entropia que queremos calcular.
- n é o tamanho do nosso alfabeto.
- $P(K_i)$ é a probabilidade de num dado alfabeto, o símbolo aparecer.

Ao longo do trabalho também recorreremos à codificação de Huffman e ao agrupamento de símbolos 2 a 2 de forma a estudar e a comparar os ganhos ou perdas relativamente à entropia e incerteza.

Por fim, também trabalhámos com a informação mútua de forma a estudar e a comparar a variação da mesma entre vários ficheiros de som.

Exercício 1:

Neste exercício iremos proceder à criação de uma função com o intuito de, dada uma fonte de informação (iremos proceder aos testes no exercício 3), cujos símbolos pertencem a um alfabeto $A=\{a_1, \dots, a_n\}$, criar um histograma de ocorrências para cada um dos seus símbolos.

```
def histograma(fonte, alfabeto):  
    Ocorr=ocorrencias(fonte, alfabeto)  
    y=np.arange(len(alfabeto))  
  
    plt.bar(y, Ocorr, align='center', color='orange')  
    plt.xticks(y, alfabeto)  
    plt.xlabel('Alfabeto')  
    plt.ylabel('Ocorrencias')  
    plt.title('Sinal')  
    plt.show()
```

A função acima é responsável pela criação do histograma. Recebe uma fonte de símbolos e o respetivo alfabeto, recorrendo ao numpy para organizar o nosso alfabeto e ao matplotlib para criar o histograma. Também criámos uma função **ocorrencias**, apresentada abaixo.

```
def ocorrencias(fonte, alfabeto):  
    nrOcor = [0] * len(alfabeto)  
    f=fonte.tolist()  
  
    for i in range(len(alfabeto)):  
        nrOcor[i]=f.count(alfabeto[i])  
  
    return nrOcor
```

Esta função recebe a nossa fonte e o respetivo alfabeto e devolve uma lista com o número de vezes que um símbolo aparece na fonte.

Exercício 2:

Neste exercício vamos criar uma função cujo objetivo é o cálculo da entropia de uma fonte.

```
def entropia(fonte):  
    H=0  
    Alfabeto, Ocorr= np.unique(fonte, axis=0, return_counts=True)  
    for i in range(len(Ocorr)):  
        if(Ocorr[i]!=0):  
            probabilidade=Ocorr[i]/len(fonte)  
            H+=probabilidade*math.log2(probabilidade)  
  
    return round(-H,3)
```

A função **entropia** recebe como parâmetros a fonte de informação. Depois irá recorrer ao np.unique para obter a lista de ocorrências de cada símbolo diferente na fonte e o seu respetivo alfabeto (colocando o parâmetro return_counts = True) e por fim irá realizar um ciclo para calcular a probabilidade e, usando a fórmula citada na introdução, descobrir a entropia. O valor final de H é negativo logo, quando fazemos o retorno do resultado, usamos -H para passar a entropia para um valor positivo e arredondamos o valor final a 3 casas decimais.

Exercício 3:

Neste exercício iremos recorrer às duas funções que criámos acima para obter o histograma e a entropia para cada uma das fontes.

```
def LerFich(nome):
    extensao = nome.split('.')[1]
    alfabeto=[]
    P=[]

    if (extensao == "wav"):
        data = wavfile.read(nome)
        if (len(data[1].shape) == 2):
            P = data[1][:, 1]
        elif (len(data[1].shape) == 1):
            P = data[1]
        else:
            print("Ficheiro WAV inválido")
            return -1
        alfabeto = np.arange(256)

    elif (extensao == "bmp"):
        img = mpimg.imread(nome)
        if (len(img.shape) == 2):
            P = img.flatten()
        elif (len(img.shape) == 3):
            P = (img[:, :, 0].flatten())
        alfabeto = np.arange(256)

    elif (extensao == "txt"):
        P = [ch for ch in open(nome, 'r').read() if
            ((ord(ch) >= 65 and ord(ch) <= 90) or (ord(ch) >= 97 and ord(ch) <= 122))]
        alfabeto = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
                    'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
                    'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
        alfabeto=np.array(alfabeto)

    return np.array(P), alfabeto
```

Para isso recorreremos à função **LerFich** que recebe o nome do ficheiro a ler. Dentro dela, é feita uma verificação da sua extensão. Se for do tipo **.wav** sabemos que se trata de um ficheiro de áudio, se for **.bmp** é um ficheiro de imagem e se for **.txt** é um ficheiro de texto, sendo que cada um tem uma maneira diferente de ser aberto. Por fim a função devolve dois arrays, um P que corresponde à nossa fonte e um alfabeto que apresenta todos os símbolos possíveis para o tipo de fonte que temos.

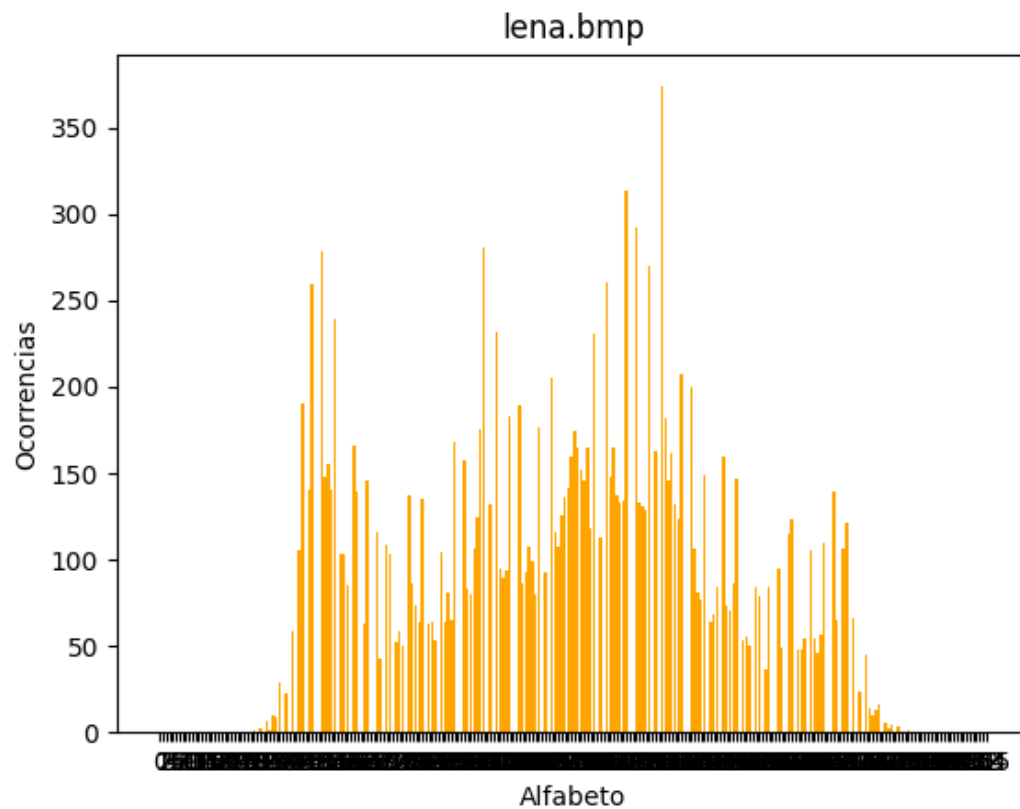
Por fim vamos executar a nossa função no main e analisar os resultados:

```
if __name__ == "__main__":  
    Fontes=["lena.bmp", "ct1.bmp", "binaria.bmp", "saxriff.wav", "texto.txt"]  
    fonte, alfabeto = LerFich(Fontes[0])  
  
    #3), 4) e 5)  
  
    for i in range(5):  
        nome = Fontes[i]  
        fonte, alfabeto = LerFich(nome)  
        Bs, var = huffman(fonte)  
  
        print(nome + ":")  
        print("H(x)= ", entropia(fonte),)  
        histograma(fonte, alfabeto)
```

A variável nome corresponde ao nome do ficheiro. De seguida obtemos a fonte e o alfabeto recorrendo à função **LerFich** usamos as funções **histograma** e **entropia** para obter os resultados pretendidos.

De seguida iremos obter o histograma e a entropia associados a cada um dos ficheiros, seguidos de uma análise e comparação relativos aos resultados obtidos.

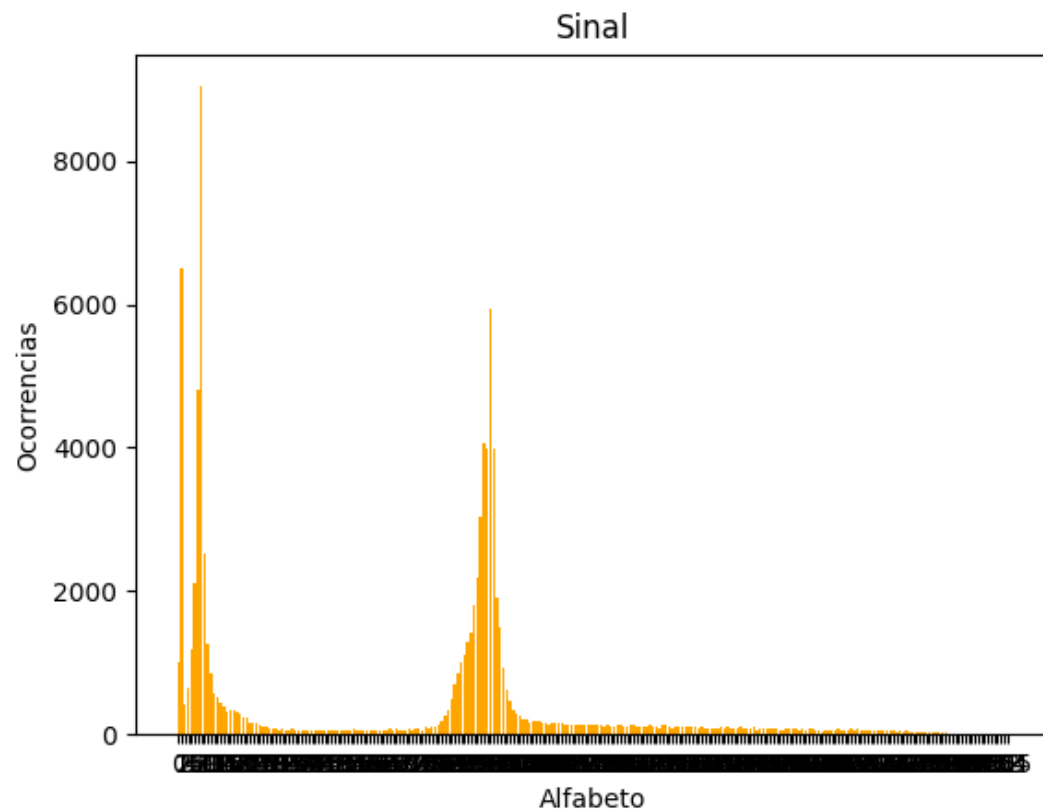
- Lena.bmp:



A entropia da fonte de informação fornecida é 6.9153 bits/símbolo

O nosso alfabeto está compreendido entre 0 e 255 uma vez que se trata de uma imagem. Uma vez que as ocorrências estão bastante dispersas, o valor da entropia é elevado.

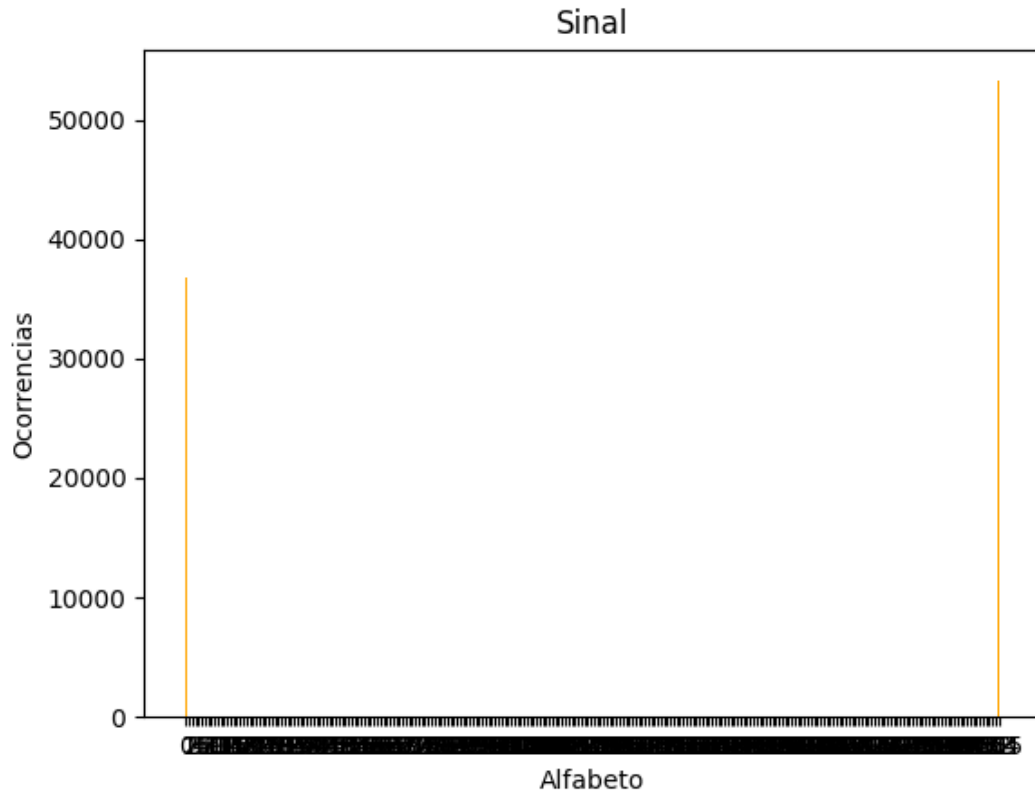
- CT1.bmp:



A entropia da fonte de informação fornecida é 5.972 bits/símbolo

Como também se trata de uma imagem o alfabeto está compreendido entre 0 e 255. Contudo, face à imagem anterior, a nossa entropia é inferior, o que denota uma diminuição de incerteza. É possível observar dois picos de ocorrências em pontos distintos.

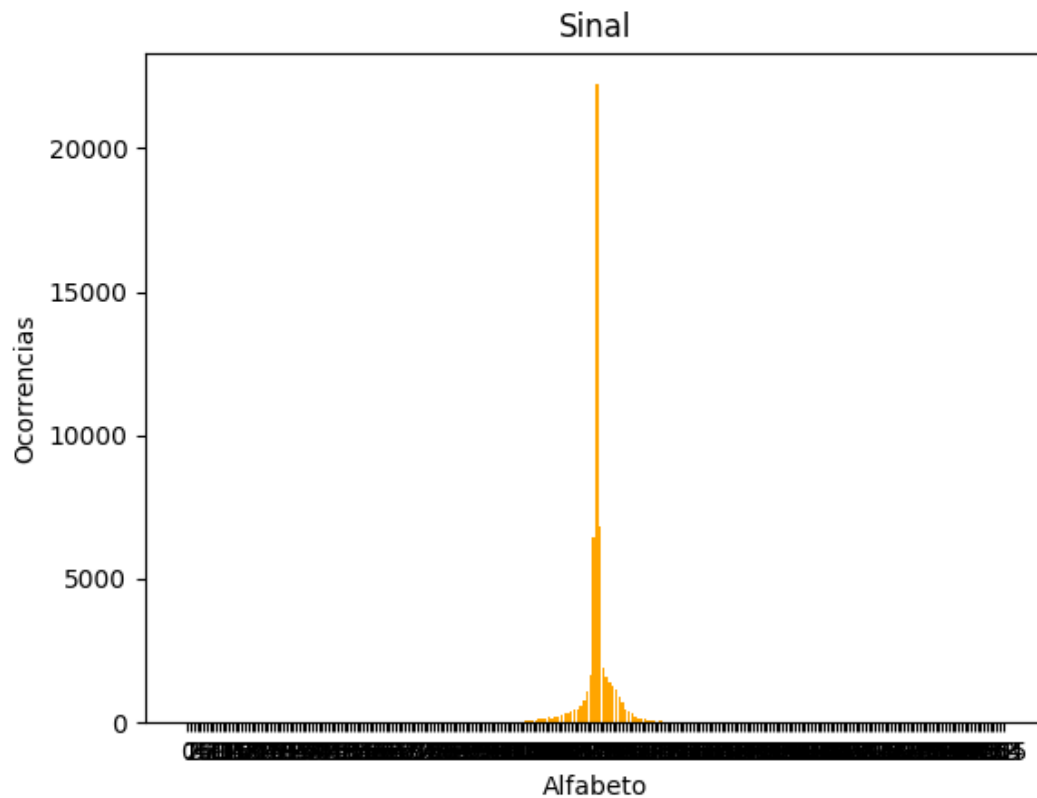
- Binaria.bmp:



A entropia da fonte de informação fornecida é 0.976 bits/símbolo

À semelhança dos dois ficheiros anteriores, este também é uma imagem e, como tal, tem um alfabeto compreendido entre 0 e 255. Contudo, como a imagem é binária, apenas são registadas ocorrências de dois símbolos (branco e preto), o que leva a um decréscimo bastante elevado da entropia (incerteza baixa).

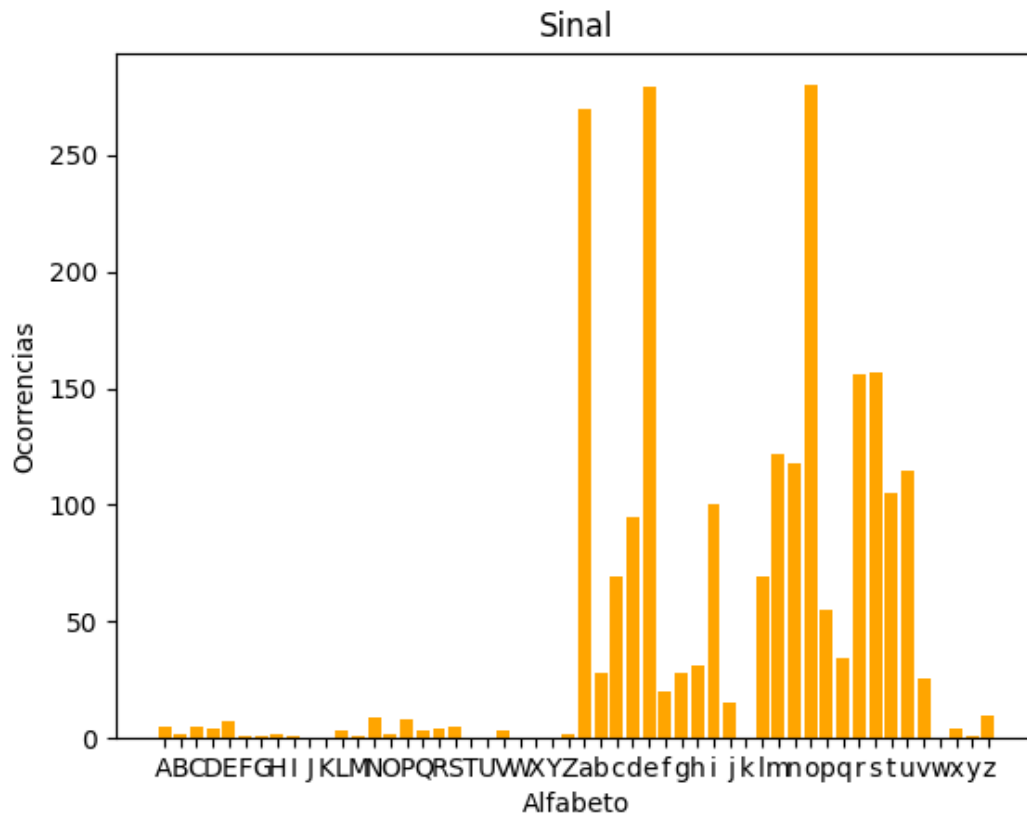
- Saxriff.wav:



A entropia da fonte de informação fornecida é 3.54 bits/símbolo

Este é o primeiro ficheiro de áudio que analisamos o seu alfabeto vai ser semelhante (0 a 255). Relativamente à entropia, a maioria das ocorrências está concentrada num ponto do alfabeto, o que origina uma entropia baixa.

- Texto.txt:



A entropia da fonte de informação fornecida é 4.197 bits/símbolo

Este é também o primeiro ficheiro de texto que analisamos e, como tal, o alfabeto é constituído pelas letras maiúsculas e minúsculas de A a Z, com acentos e caracteres especiais não incluídos. Verifica-se uma maior concentração das ocorrências nas letras minúsculas e uma entropia mais baixa que os dois primeiros ficheiros.

Será possível comprimir cada uma das fontes de forma não destrutiva? Se Sim, qual a compressão máxima que se consegue alcançar? Justifique.

Sim, uma vez que para cada uma das fontes, a sua entropia é inferior ao número de bits que é usado para as representar (1 byte ou 8 bits). Como tal, podemos calcular a sua compressão máxima através da seguinte fórmula:

$$CompressãoMáxima(X) = \frac{Hmax(X) - H(X)}{Hmax(X)} \times 100$$

Onde:

- $Hmax(X)$ corresponde à entropia máxima da fonte.
- $H(X)$ corresponde à entropia calculada

A tabela seguinte apresenta a compressão máxima, em percentagem, para cada uma das nossas fontes:

Fonte	Entropia	Compressão
Lena.bmp	6,915	13,562%
CT1.bmp	5,972	25,35%
Binaria.bmp	0,976	87,8%
Saxriff.wav	3,540	55,75%
Texto.txt	4,197	47,538%

Exercício 4:

Neste exercício iremos recorrer à codificação de Huffman para obter a entropia de cada uma das fontes. Como tal utilizaremos o ficheiro Python fornecido para proceder ao cálculo da entropia.

```
def huffman(fonte):  
    codec = HuffmanCodec.from_data(fonte)  
    symbols, lenghts = codec.get_code_len()  
    Ocorr=ocorrenciasU(np.unique(fonte))  
    BS=Ef=Ef_2=0  
  
    for i in range(len(lenghts)):  
        for j in fonte:  
            if(j==symbols[i]):  
                BS+=lenghts[i]  
  
        Ef_2+=lenghts[i]**2 * Ocorr[i]/len(fonte)  
        Ef+=lenghts[i] * Ocorr[i]/len(fonte)  
  
    V = round(Ef_2 - Ef ** 2, 4)  
  
    return round(BS/len(fonte),3), V
```

Vamos executar a função **get_code_len** fornecida para obter dois arrays, um com os símbolos diferentes e outro com o número de bits para codificar cada um dos símbolos. De seguida iremos proceder ao cálculo da média, adicionando a uma variável o número de bits cada vez que, ao percorrer a fonte, encontramos um símbolo igual ao símbolo do alfabeto a analisar, fazendo uma iteração pelo alfabeto completo. Por fim iremos dividir este valor pelo comprimento da fonte, de forma a obter assim o número médio de bits. Esta função também é responsável por calcular a variância, obedecendo à seguinte fórmula:

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

Na tabela seguinte procede-se à comparação da entropia calculada no exercício 3 com a entropia usando codificação de Huffman, também apresentando a variância relativa a cada ficheiro:

	Entropia normal	Entropia de Huffman	Variância
Lena.bmp	6,915	6,943	0,639
CT1.bmp	5,972	6,008	5,202
Binaria.bmp	0,976	1,000	0
Saxriff.wav	3,540	3,595	7,739
Texto.txt	4,197	4,217	1,883

Como é possível observar na tabela, a entropia calculada através da codificação de Huffman é ligeiramente superior à entropia obtida sem o recurso a esta codificação.

Será possível reduzir-se a variância? Se sim, como pode ser feito em que circunstância será útil?

Sim, uma vez que, de forma a obter uma variação mínima usando a codificação de Huffman, podemos dar preferência aos símbolos de menor comprimento, aquando da formação da árvore de Huffman. De forma a podermos efetuar uma redução da variância de forma útil, criamos árvores de Huffman onde o resultado da associação de símbolos são priorizados no topo da nossa árvore. Como tal, iremos ter uma distribuição mais uniforme e regular dos bits e nota-se um aumento da otimização do código.

Exercício 5:

Neste exercício iremos proceder ao cálculo da entropia e à apresentação dos histogramas, usando o agrupamento de símbolos dois a dois.

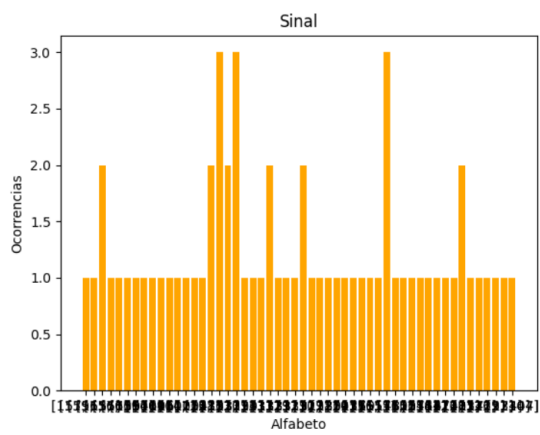
```
def entropiaMelhorada(fonte):  
    P_novo, alfabeto2 = Agrupamento2(fonte)  
    H = entropia(P_novo)  
  
    return round(H / 2, 3)
```

Na função **entropiaMelhorada** recebemos a fonte de informação, que já vem do exercício 3, e, com ajuda da função **Agrupamento2**, convertemos a nossa fonte num array bidimensional, cujos símbolos já se encontram agrupados 2 a 2. De seguida chamamos a função **entropia** para calcular a entropia da nossa fonte já agrupada 2 a 2 que depois é dividida por 2.

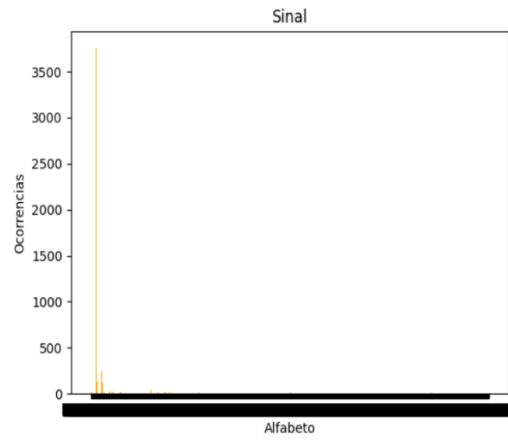
```
def Agrupamento2(fonte):  
    if (len(fonte) % 2 != 0):  
        fonte = fonte[:len(fonte) - 1]  
        P_novo = np.reshape(fonte, ((int)((len(fonte) + 1) / 2), 2))  
    else:  
        P_novo = np.reshape(fonte, ((int)((len(fonte)) / 2), 2))  
  
    alf = []  
  
    for i in P_novo.tolist():  
        if i not in alf:  
            alf.append((i))  
  
    return P_novo, alf
```

Acima temos a função já referida no texto anterior que, ao receber a nossa fonte, devolve dois arrays bidimensionais com a nova fonte e alfabeto organizados com símbolos 2 a 2. Para tal verificamos se o array da fonte é par ou ímpar e, recorrendo ao método **reshape** do numpy iremos transformar este array unidimensional em array bidimensional.

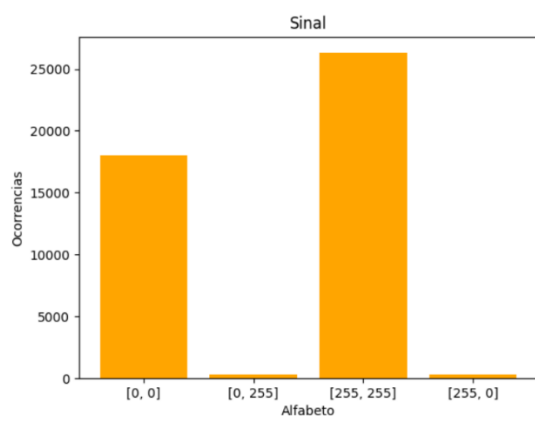
Lena.bmp:



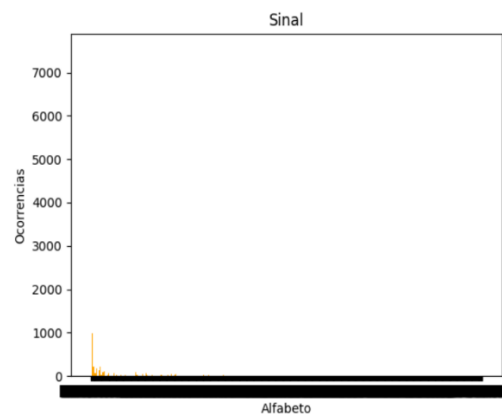
CT1.bmp:



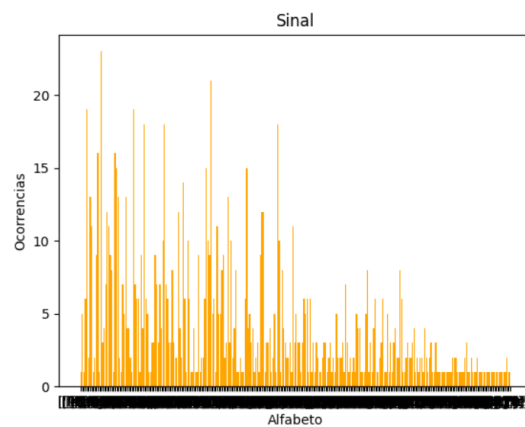
Binaria.bmp:



Saxriff.wav:



Texto.txt:



	Entropia normal	Entropia 2 a 2
Lena.bmp	6,915	5,596
CT1.bmp	5,972	4,481
Binaria.bmp	0,976	0,542
Saxriff.wav	3,540	2,897
Texto.txt	4,197	3,754

Como é possível observar pela tabela acima, a entropia calculada neste exercício é significativamente inferior àquela que é obtida normalmente (exercício 3), o que nos leva a concluir que há uma redução do número médio de bits necessários para codificar os símbolos e, como tal, uma otimização do código.

Exercício 6 a)

Neste exercício iremos proceder ao cálculo da informação mútua entre dois sinais de som. Para tal temos o sinal a pesquisar (query) e o sinal a ser pesquisado (target).

```
def InforMutua(query, target, passo):
    if(len(query)>len(target)):
        print('ERRO: impossível executar esta função visto que a "len" da query é maior que a "len" do target')
        return -1

    passo=(int)(passo)
    infMutua=[0.0]*math.ceil((len(target)-len(query)+1)/passo)
    target=np.array(target)
    query=np.array(query)

    entropia_query=entropia(np.array(query))

    for i in range(len(infMutua)):
        lista=target[passo*i:passo*i+len(query)]
        query_lista=[[query[i],lista[i]] for i in range(len(lista))]

        infMutua[i]=round((entropia_query - entropia(np.array(query_lista)) + entropia(np.array(lista))),4)

    return infMutua
```

Para proceder a esse cálculo criamos a função **InforMutua** que recebe a query, o target e o passo (salto que iremos dar quando analisamos o target). Para tal iremos analisar o nosso target, passo a passo, e realizando um ciclo para obter a informação mútua relativa à nossa query e a cada porção do target, usando a função seguinte:

$$I(X,Y) = H(X) - H(X,Y) + H(Y)$$

Onde:

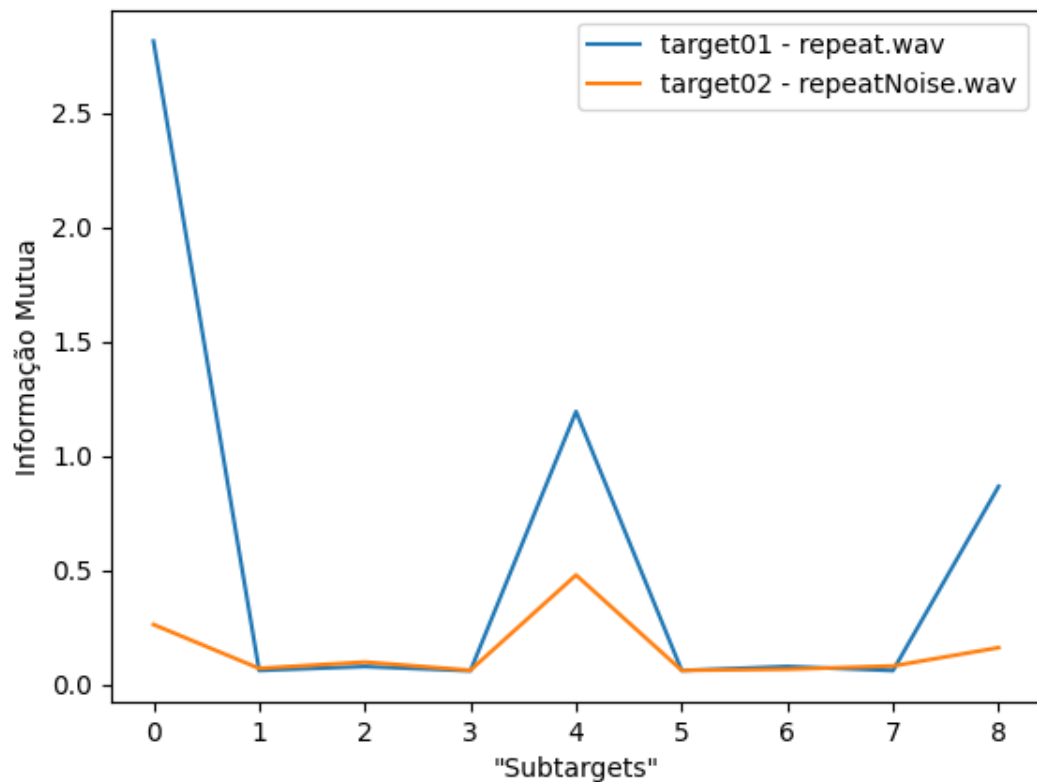
- X corresponde à query
- Y corresponde ao target
- $H(X)$ corresponde à entropia da query
- $H(Y)$ corresponde à entropia do target
- $H(X,Y)$ corresponde à entropia conjunta de X e Y.

Exercício 6 b)

Neste exercício iremos usar a função **InforMutua** criada anteriormente para calcular a informação mútua de um ficheiro de som (saxriff.wav) com dois ficheiros de som target, um normal (target02 – repeat) e outro com ruído adicionado (target02 -repeatNoise) e estudar a sua variação. Como indicado no enunciado, o passo definido para este exercício corresponde a $\frac{1}{4}$ do comprimento arredondado da fonte da nossa query.

```
def Targets(query, target, target2, passo):  
    plt.plot(InforMutua(query, target, passo), label="target01 - repeat.wav")  
    plt.plot(InforMutua(query, target2, passo), label="target02 - repeatNoise.wav")  
  
    plt.xlabel('Subtargets')  
    plt.ylabel('Informação Mútua')  
  
    plt.legend()  
    plt.show()
```

A função **Targets** irá receber a nossa query (fonte do som a analisar) e os nossos dois targets (fontes dos sons a ser analisado) e o passo já estabelecido acima. De seguida irá criar o nosso plot que irá usar a função **InforMutua** para calcular a informação mútua de cada target com a query e apresentá-la num gráfico. Os parâmetros das fontes são obtidos com recurso à função **LerFich** criada no exercício 3.



Como é possível observar, a informação mútua é superior quando o nosso target não tem adicionado o ruído, ruído esse que irá causar uma diminuição da informação mútua obtida entre os dois ficheiros de som. Também é notório um aumento significativo da informação mútua no “subtarget” 0, 4 e 8.

Exercício 6 c)

Neste exercício iremos também utilizar a função **InforMutua** criada na alínea a) para comparar a informação mútua de uma query (saxriff.wav) e vários ficheiros de som (targets) e estabelecer uma ordem decrescente de informação mútua entre si.

```
def Shazam(query, passo):
    Targets=["Song01.wav", "Song02.wav", "Song03.wav", "Song04.wav", "Song05.wav", "Song06.wav", "Song07.wav"]

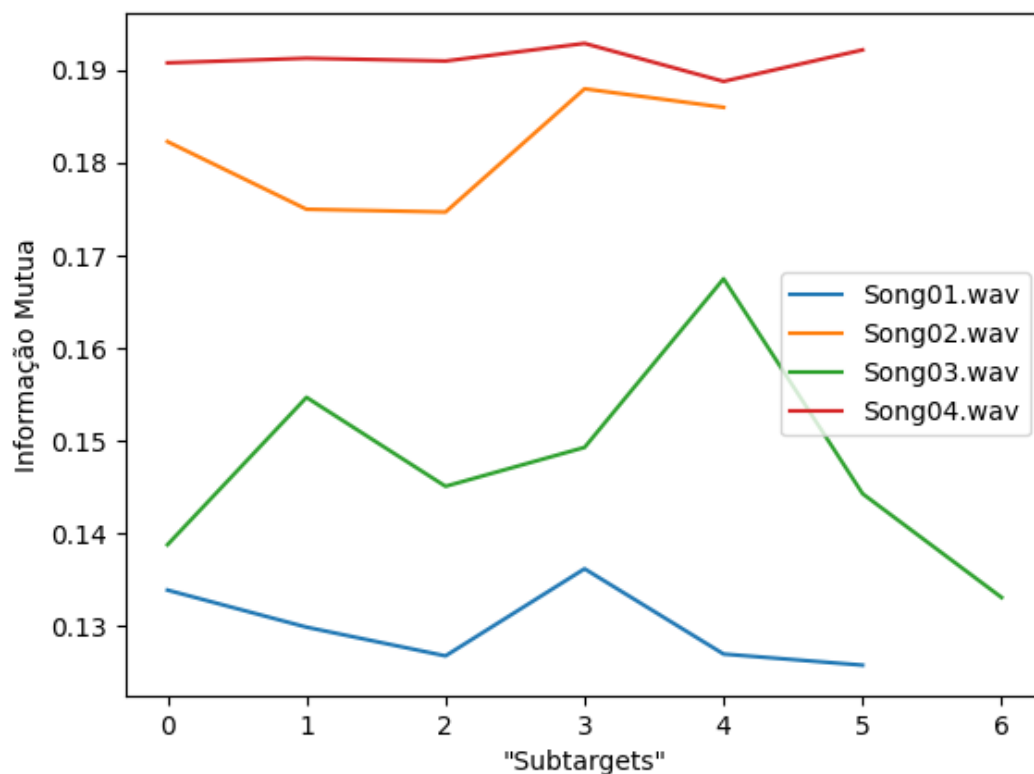
    for i in range(7):
        fonteT2, alfabetoT2 = LerFich(Targets[i])
        inf = InforMutua(query, fonteT2, passo)

        print("A informação mútua máxima do ficheiro '" + Targets[i] + "' é", max(inf))
        if(i<4):
            plt.plot(inf, label=Targets[i])
        else:
            print('Não é possível representar o gráfico pois o tamanho da "query" é igual ao do "target", logo só haverá um ponto onde é calculada a informação mútua')

    plt.xlabel("Subtargets")
    plt.ylabel('Informação Mútua')

    plt.legend()
    plt.show()
```

Para isso criámos a função **Shazam** que recebe como parâmetros a nossa query e o passo (também $\frac{1}{4}$ do comprimento arredondado da fonte da nossa query). Já tendo os targets num array irá realizar um ciclo de forma a poder calcular a informação mútua para cada um deles e adicionar essa informação ao nosso plot. Caso o tamanho da query seja igual ao tamanho do target em questão, não será possível analisar a evolução da informação mútua e, como tal, irá apresentar uma mensagem de aviso no ecrã do utilizador.



No gráfico acima representado temos a representação gráfica da informação mútua ao longo de cada “subtarget” para cada um dos targets. Apesar de a análise ser feita a 7 ficheiros de som, apenas 4 deles estão presentes uma vez que 3 deles têm um comprimento igual ao da nossa query, o que impossibilita o estudo.

Informação mútua máxima ordenada
Song07.wav : 3,540
Song06.wav : 3,540
Song05.wav : 0,533
Song04.wav : 0,193
Song02.wav : 0,188
Song03.wav : 0,168
Song01.wav : 0,136

Conclusão:

Este trabalho serviu como uma base de conhecimentos muito importante para nós. Ficámos a conhecer muito melhor diversos conceitos como a entropia e a informação mútua, assim como formas de os colocar em prática. Com recurso ao Python, linguagem de programação onde já possuímos algumas bases, e às diversas bibliotecas como o numpy, que nos ajudou a tratar os arrays de fontes, e o matplotlib, responsável pela criação e exposição de gráficos e histogramas, conseguimos mostrar o nosso conhecimento e tratar a informação que nos era fornecida.

Concluimos também que, das várias entropias calculadas, o melhor método é o agrupamento de símbolos 2 a 2, o que resulta numa melhor otimização do código e como funcionam aplicações como o Shazam, responsáveis por analisar e comparar ficheiros de som, e como recorrem à informação mútua.