FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

TEORIA DA INFORMAÇÃO

Trabalho Prático nº1

Entropia, Redundância e Informação Mútua

2017251964– Madalena Maria Roque dos Santos Silva
2017125592– Maria Paula de Alencar Viegas
2017255810– Vitalina Holubenko

Introdução

Este trabalho tem como objetivo a aprendizagem de questões fundamentais de teoria de informação, em particular informação, redundância, entropia e informação mútua. Para isto, aplicámos os conceitos estudados nas aulas teóricas no programa MATLAB, que nos permite analisar gráficos e dados de diferentes fontes (como imagens, ficheiros de texto e áudio).

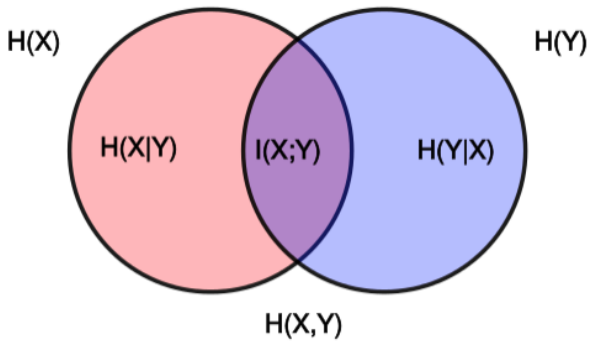
Para a realização deste trabalho, abordámos o conceito de **entropia** (limite mínimo teórico para o número médio de bits por símbolo), a medida de incerteza de uma variável aleatória, dada pela seguinte fórmula:

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

H = entropia
P(ai) = probabilidade de ocorrência
n = tamanho do alfabeto
ai = símbolo do alfabeto

Também abordamos o conceito de **informação mútua**, que é a medida de dependência mútua entre duas variáveis, a quantidade de informação que uma variável contém sobre outra. A informação mútua associada a um par de variáveis X e Y é dada por:

$$\begin{aligned} I(X,Y) &= I(Y,X) \\ &= H(Y) - H(Y \mid X) \\ &= H(X) - H(X \mid Y) \end{aligned}$$



I(X,Y) = informação mútua
H(X)= entropia de X
H(Y) = entropia de Y
H(X,Y) = entropia conjunta de X e Y

Exercício 1)

Neste exercício, criámos uma rotina *histograma_simbolos.m* que recebe como argumentos de entrada uma fonte e o nome do ficheiro.

```
prompt = 'Escreva o nome do ficheiro: ';
nome_ficheiro=input(prompt, 's');
fonte=ler_ficheiros(nome_ficheiro);

%exercicio 1

disp('Exercicio1: Histograma')
histograma_simbolos(fonte, nome_ficheiro);
disp('->Carregue enter para visualizar o proximo exercicio')
pause()
```

Esta função *histograma_simbolos(),* utiliza: a função *unique()* para obter um alfabeto sem repetições de símbolos da fonte fornecida e *bar()* para criar um gráfico (sendo o eixo do x o alfabeto e o do y as ocorrências).

```
function [contar] = histograma_simbolos(fonte, titulo)

%cria o histograma

close all;

alfabeto = unique(fonte);
contar = contar_ocorrencias(alfabeto, fonte);
bar(alfabeto,contar);


%output= contar;

title(titulo);
xlabel('Alfabeto');
ylabel('Numero de ocorrencias');


end
```

Para facilitar criámos a rotina *contar_ocorrencias.m* que conta as ocorrências do alfabeto na fonte (ambos argumentos da função) e devolve uma matriz com a contagem de cada símbolo (variável *contagem*).

```
function [contar] = contar_ocorrencias(alfabeto, fonte)
%Conta as ocorrências do alfabeto na fonte
%Devolve uma matriz com a contagem

    contar = zeros(length(alfabeto),1);

    for l=1:length(alfabeto)
        i = find(fonte==alfabeto(l));
        contar(l)=length(i);
    end
%output= contar;

end
```

Exercício 2)

Para este exercício, criámos uma rotina *entropia.m* para calcular o limite mínimo teórico para o número de bits por símbolo.

```
disp('Exercicio 2: Entropia')
prompt = 'Escreva o nome do ficheiro: ';
nome_ficheiro = input(prompt, 's');
fonte = ler_ficheiros(nome_ficheiro);
alfabeto = unique(fonte);
contagem = contar_ocorrencias(alfabeto, fonte);
H = entropia(contagem);
fprintf('Entropia: %.5f\n\n', H);
disp('->Carregue enter para visualizar o proximo exercicio')
pause()
```

Para isso recorremos à função *contar_ocorrências()* que vai ser utilizada na fórmula da entropia.

```
function [H] = entropia(contagem)

%limite mínimo teórico para o número médio de bits por símbolo

H = 0;
%Fórmula da Entropia:
%H = somatorio *  (probabilidade(ai)*log2(1/prob(ai)) )
%  = - somatorio * (probabilidade(ai)*log2(prob(ai))

    total=sum(contagem);
    probabilidade= contagem/total;
    probabilidade= probabilidade(probabilidade~=0); %elimina probabilidades = 0
    H = -sum(probabilidade.*log2(probabilidade));



end
```

Exercício 3)

Neste exercício, aplicamos as funções criadas nos exercícios anteriores a 3 tipos de ficheiro: imagem(.bmp), áudio(.wave) e texto(.txt).

Para isto, utilizamos uma rotina auxiliar *ler_ficheiros.m* que recebe uma string *nome_ficheiro* com o nome do ficheiro a ser analisado e devolve a fonte deste. A partir desta string, verifica-se o formato do ficheiro e lê-se a fonte de acordo com cada formato - *audioread*() para ler um audio, *imread*() para uma imagem e *fileread*()para um texto. Para o texto fizemos também um tratamento de dados que inclui todos os símbolos regulares maiúsculos e minúsculos excluindo outros caracteres.

```
function [fonte] = ler_ficheiros(nome_ficheiro)
%ler a fonte de acordo com o seu tipo(som, imagem ou texto)

%se o ficheiro for audio

if (endsWith(nome_ficheiro, '.wav')==1)
    fonte = audioread(nome_ficheiro);

%se o ficheiro for uma imagem

elseif (endsWith(nome_ficheiro, '.bmp')==1)
    fonte= imread(nome_ficheiro);

%se for um ficheiro de texto

else
    fonte= fileread(nome_ficheiro);
    %tratamento de dados do texto
    fonte(regexp(fonte,'[., \n]')) = [];
    fonte=double(fonte);

end

%output da fonte

end
```

Após a leitura apropriada da fonte, utiliza-se as rotinas dos exercícios 1 e 2.

Exemplo para o guitarSolo:

```
%exercicio 3

fprintf('\nExercicio 3\n\n');

disp('--- guitarSolo.wav ---');
figure(1);
nome_ficheiro = 'guitarSolo.wav';
fonte=ler_ficheiros(nome_ficheiro);
contagem_1 = histograma_simbolos(fonte, nome_ficheiro);
H_1= entropia(contagem_1);
fprintf('Entropia: %.3f\n\n', H_1);

pause()
```

Resultados:

De seguida apresentamos os histogramas das fontes pretendidas e a respetiva análise dos resultados.

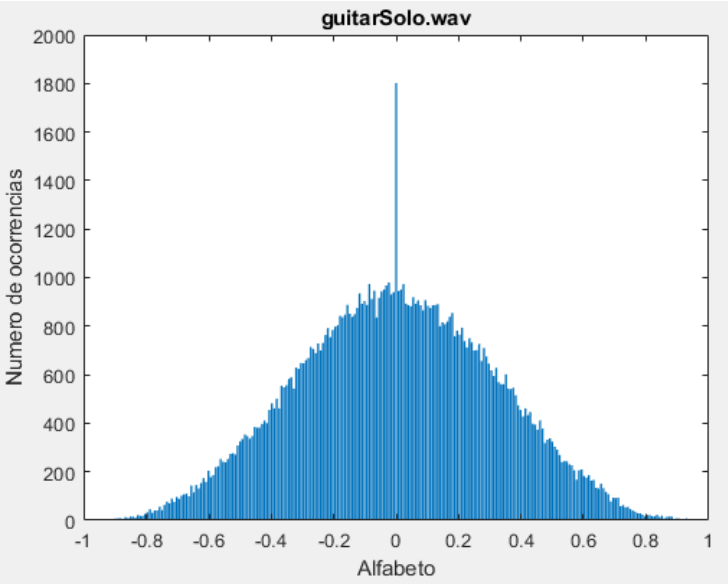


Fig. 1: Histograma do áudio "guitarSolo.wav"

GuitarSolo.wav

Entropia: 7.35802

Este áudio é de todas as fontes a que apresenta maior entropia, o que é também visualizado pela dispersão do histograma, no intervalo [-1, 1[.

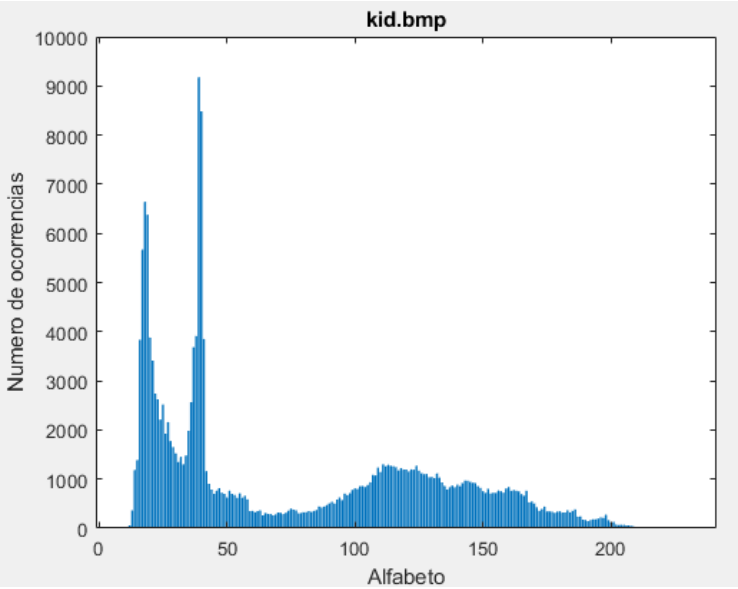


Fig. 2: Histograma da imagem "kid.bmp"

Kid.bmp

Entropia: 6.95414

Para esta fonte o alfabeto consiste nos 256 valores de pixeis.
Uma vez que a imagem apresenta vários tons de cinzento a entropia é maior e o histograma é mais disperso.

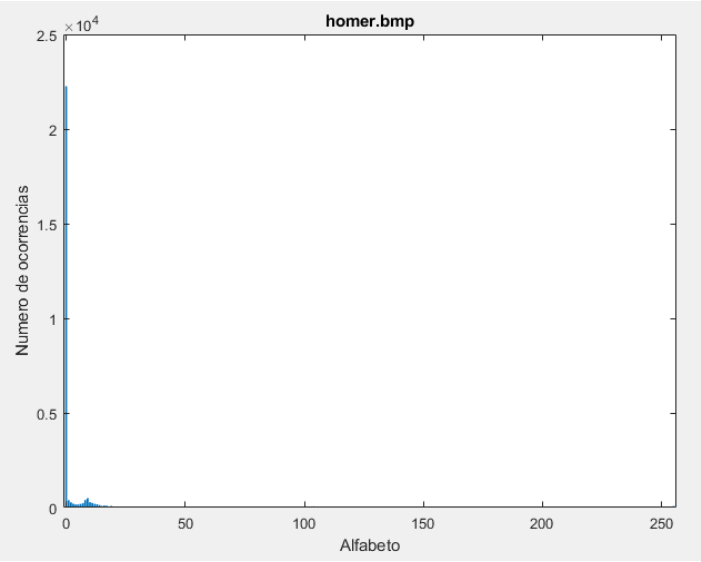


Fig 3: Histograma da imagem “homer.bmp”

Homer.bmp

Entropia: 3.46587

Para esta fonte (.bmp), o alfabeto consiste nos 256 valores de pixeis. Temos um grande número de ocorrência nos valores próximos de zero porque a imagem fornecida apresenta muitos pixeis escuros e pouca variação de tons. Isso também explica a redução do valor da entropia.

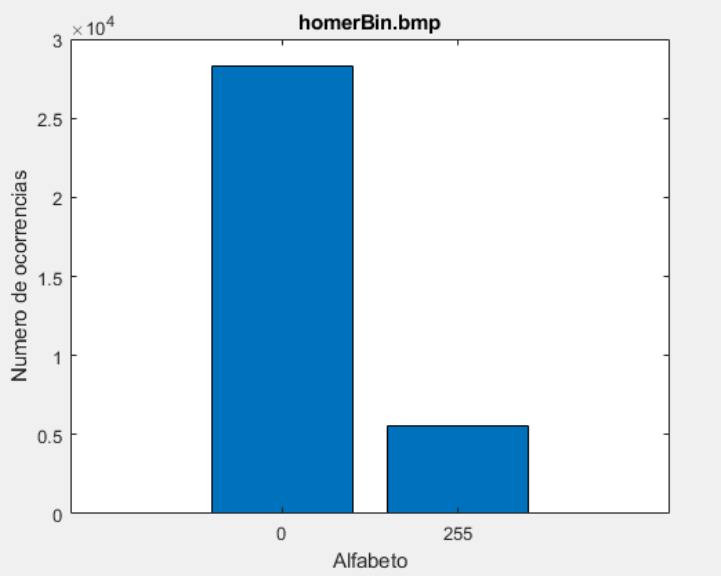


Fig. 4: Histograma da imagem “homerBin.bmp”

HomerBin.bmp

Entropia: 0.64478

O histograma aparece desta forma porque temos uma fonte imagem em binário, ou seja, composta apenas com pixeis com valor 0 (pretos) ou valor 255 (brancos). Esta também é a fonte com menor entropia uma vez que temos apenas dois tons.

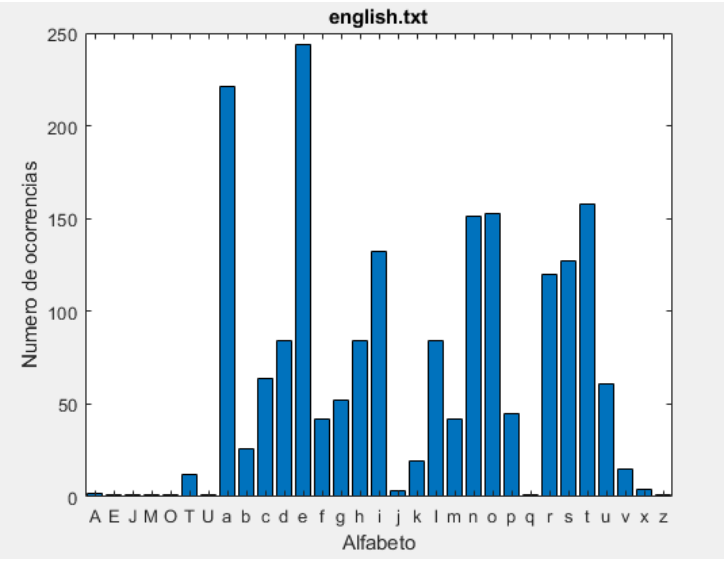


Fig. 5: Histograma do texto “english.txt”

English.txt

Entropia: 4.228

O histograma é apresentado desta forma pois temos um texto como fonte, cujo o alfabeto consiste nos símbolos regulares do alfabeto latino. Podemos verificar que existe alguma dispersão.

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

A compressão de dados consiste em reduzir o espaço ocupado por dados num dispositivo.

Para que essa compressão seja não destrutiva é necessário eliminar a redundância de modo a que os dados originais sejam idênticos aos dados após a compressão, ou seja, que haja uma reconstrução exata da fonte original.

Neste caso, desde que a imagem, texto ou áudio pretendidos tenham redundâncias que possam ser eliminadas sem deixar de ser igual à fonte original, podemos comprimir de forma não destrutiva.

Como neste caso usamos 8 bits para guardar a informação de cada amostra, mas a entropia em todas as amostras é menor que 8 bits, podemos comprimir a amostra de forma não destrutiva.

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

Os ficheiros com menor taxa de compressão diferem pouco em relação ao valor da entropia máxima e o valor da entropia calculado.

Para obter tais valores, utilizamos as fórmulas abaixo:

$$H_{\max} = \log_2(k)$$

O valor máximo de entropia de uma fonte discreta com alfabeto de k elementos

$$TaxaCompressão(X) = \frac{EntropiaMáxima - Entropia}{Entropia\ Máxima} \times 100$$

FICHEIRO	ENTROPIA (bits/ símbolo)	TAXA DE COMPRESSÃO(%)
kid.bmp	6.9541	13
homer.bmp	3.4659	56.677
homerBin.bmp	0.6448	91,94
guitarSolo.wav	7.3580	8.03
english.txt	4.1943	47.39

Exercício 4)

Nesta etapa, utilizamos as rotinas de codificação de **Huffman** fornecidas e uma rotina *media.m* criada para obter o número médio de bits por símbolo para cada uma das fontes de informação.

```
function media = media(contagem)

    hl = hufflen(contagem);
    p = contagem/sum(contagem);
    media = sum(hl.*p);

end
```

A rotina hufflen.m recebe uma matriz com a contagem de cada símbolo da fonte e une dois a dois os símbolos de menor probabilidade (e depois o símbolo auxiliar com a soma dos dois é readicionado no conjunto de símbolos) até todos terem passado pelo processo. Devolve-se uma matriz com o conjunto comprimido.

Já a rotina *media()* calcula a média de bits por símbolo mediante o somatório de todas probabilidades de cada símbolo multiplicada pela respetiva compressão.

FICHEIRO	ENTROPIA (bits/ símbolo)	HUFFMAN (bits/ símbolo)	VARIÂNCIA
kid.bmp	6.9541	6.9832	2.0984
homer.bmp	3.4659	3.5483	13.1968
homerBin.bmp	0.6448	1	0
guitarSolo.wav	7.3580	7.3791	0.7563
english.txt	4.1943	4.2518	1.1988

Observamos através da tabela que os códigos de Huffman têm valores próximos aos valores da entropia anteriormente calculada, sendo os valores de Huffman maiores. Depreendemos assim que os códigos de Huffman contêm algumas limitações ao ultrapassam o limite mínimo teórico para o número médio de bits por símbolo.

🕮 Será possível reduzir-se a variância?

Para obter um código de Huffman de variância mínima, é preciso colocar os símbolos combinados na lista usando a ordem mais elevada possível, pois a variância é mínima num código de Huffman quando, na formação da árvore de Huffman, damos preferência aos símbolos que têm menor comprimento (por resultarem, ou não, da junção de vários símbolos).

- 🕮 **Se sim, como pode ser feito em que circunstância será útil?**

É possível diminuir a variância sem alterar o comprimento médio do código. Cria-se árvores de Huffman com menor profundidade, ou seja, o resultado da associação de dois símbolos deve ser colocado o mais acima possível na árvore. Assim, com o comprimento dos símbolos mais uniforme, os bits produzem-se de uma maneira mais regular e diminui-se assim a possibilidade de erros na descodificação, tornando o código mais eficiente.

Exercício 5)

No exercício 5 é pedido para fazer o mesmo que no exercício 3, mas com o alfabeto agrupado. Ou seja, fazendo com que cada símbolo seja uma sequência de dois símbolos contíguos.

Para este efeito criámos a função *agrupamento()*, que recebe a fonte e vai começar por verificar se o tamanho da função é par. Se não for, retira o último elemento da fonte (esta verificação é feita uma vez que vamos agrupar os símbolos 2 a 2).

De seguida separamos em dois arrays os elementos ímpares e pares e por fim criamos uma nova fonte com a junção dos elementos dos dois arrays fazendo um shift left aos elementos ímpares e anexando os pares, agrupando assim o alfabeto de 2 em 2.

```
function [nova_fonte] = agrupamento(fonte)
%agrupar os simbolos do alfabeto em dois simbolos continuos

%a nova fonte vai ter metade do tamanho da fonte original visto que os simbolos sao agrupados 2 a 2

fonte = double(fonte);
if (mod(length(fonte), 2) == 1)
    fonte(end)=[];
end

array_impar = fonte(1:2:end-1);
array_par = fonte(2:2:end);

nova_fonte = double(array_impar*(2^8) + array_par);

%output da nova_fonte

end
```

Seguidamente apresentamos os histogramas das fontes pretendidas e a respetiva análise dos resultados.

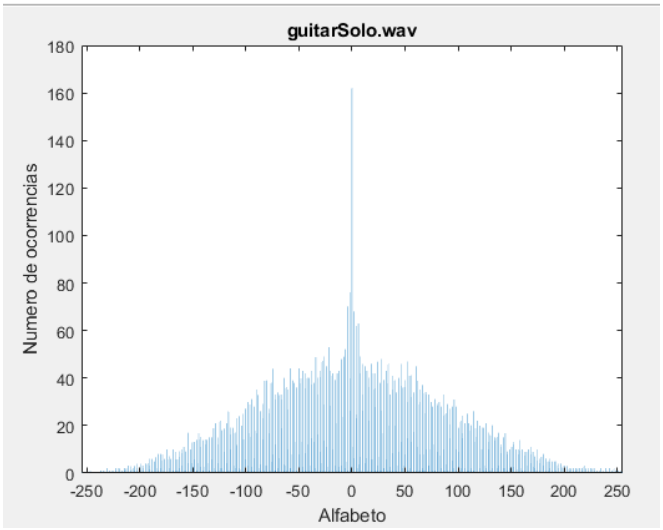
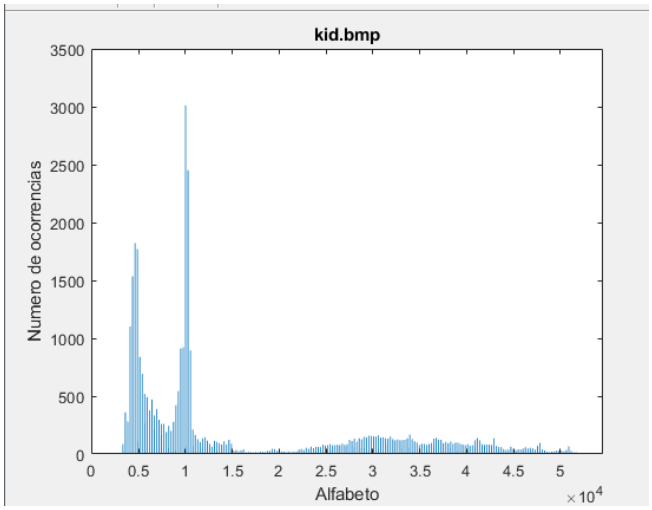


Fig. 6: Histograma do áudio "guitarSolo.wav" com agrupamento

GuitarSolo.wav

Entropia agrupada:
11.562/2 = 5.781 bits/símbolo

Sem agrupamento o valor era 7.35802, que é maior que a entropia agrupada.

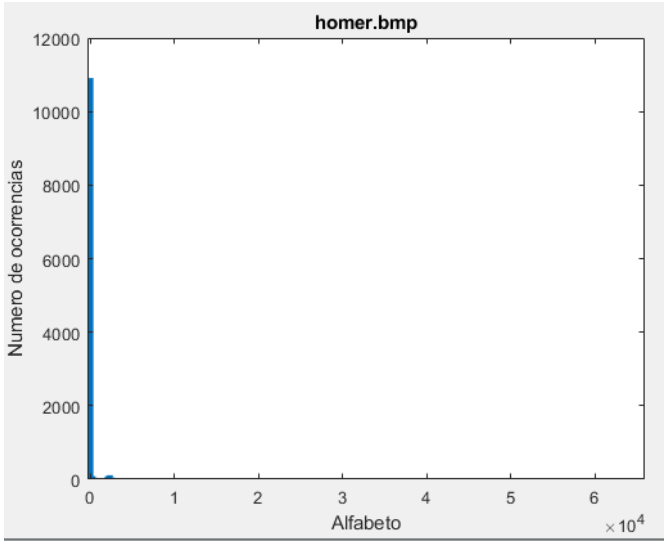


Kid.bmp

Entropia agrupada: 9.840/2 = 4.920 bits/símbolo

Sem agrupamento o valor era 6.95414, que é maior que a entropia agrupada.

Fig. 7: Histograma da imagem "kid.bmp" com agrupamento

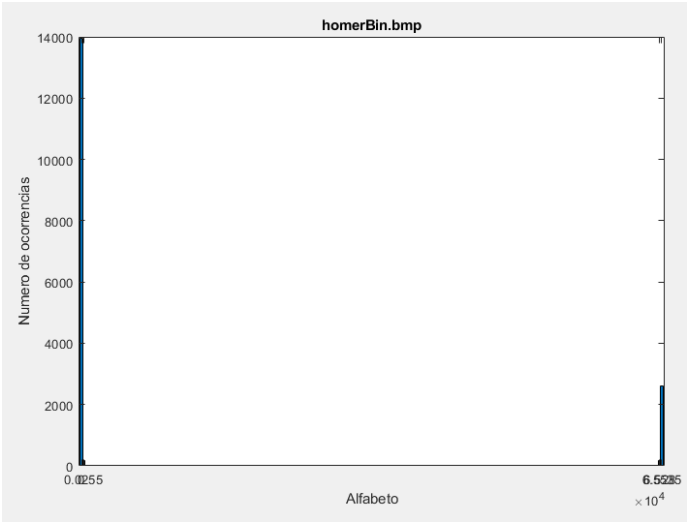


Homer.bmp

Entropia agrupada: 4.840/2 = 2.420 bits/símbolo

Sem agrupamento o valor era: 3.46587, que é maior que a entropia agrupada.

Fig. 8: Histograma da imagem "homer.bmp" com agrupamento

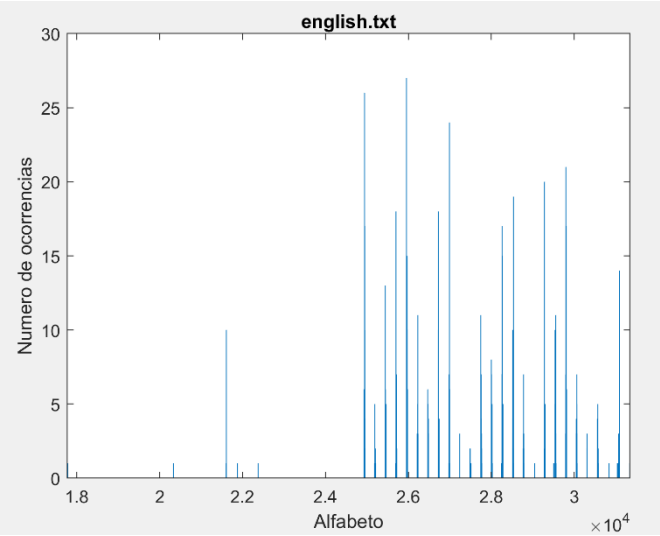


HomerBin.bmp

Entropia: 0.781/2 = 0.391 bits/símbolo

Sem agrupamento o valor era 0.64478, que é maior que a entropia agrupada.

Fig. 9: Histograma da imagem "homerBin.bmp" com agrupamento



English.txt

Entropia: 7.304/2 = 3.655 bits/símbolo

Sem agrupamento o valor era 4.10066, que é maior que a entropia agrupada.

Fig. 10: Histograma do texto "english.txt" com agrupamento

Podemos assim, através da análise dos valores obtidos, verificar que ao agrupar o alfabeto há uma otimização do código uma vez que a entropia agrupada é menor que a entropia anteriormente verificada (há uma redução do número médio de bits necessários por símbolo).

FICHEIRO	ENTROPIA (bits/simbolo)	ENTROPIA AGRUPADA (bits/simbolo)
guitarSolo.wav	7.358	5.781
kid.bmp	6.954	4.920
homer.bmp	3.466	2.420
homerBin.bmp	0.645	0.391
english.txt	4.101	3.655

Exercício 6a)

Para esta primeira parte do exercício, a partir de um *query, um target, um alfabeto e um passo* uma função deve devolver um vetor de valores de informação mútua em cada janela.

Para isso, fizemos uma rotina *informacaoMutua.m* que realiza a função na qual a cada iteração do ciclo avançamos a janela (*window*) que estamos a comparar com o target, criando assim um vetor com a evolução da Informação Mútua entre o sinal que estamos a pesquisar e o sinal onde estamos a pesquisar.

```
function [inf] = informacaoMutua(query, target, alfabeto, step)
    qLen = length(query);
    tLen = length(target);
    %guarda os tamanhos de query e de target
    pLim = (tLen-qLen+1)/step;

    inf = zeros(int8(pLim),1);
    %inf e um vetor de tamanho pLim convertidos para 8-bit [-128, 127]
    inf = double(inf);
    %inf e reconvertido para double
    k = 1;
    %inicia variavel

    for i = 1:step:(tLen-qLen+1)
        %enquanto i for menor  que a diferenca de tamanhos de step em step
        window = target(i:i+qLen-1);
        %janela deslizante
        inf(k) = calcInf(query, window, alfabeto);
        %calcula-se a informacao m?tua
        k = k+1;
        %incrementa variavel
    end
end
```

Recorremos também a outra função chamada *calcInf()* que recebe como parâmetros o querry, a janela e o alfabeto e calcula a informação mútua através da fórmula matemática.

```
function [val] = calcInf(query,window,alfabeto)
    c1 = contar_ocorrencias(alfabeto, query);
    h1 = entropia(c1);
    c2 = contar_ocorrencias(alfabeto, window);
    h2 = entropia(c2);
    %guarda a ocorrencia de cada simbolo e entropia do query e da window
    p = zeros(length(alfabeto));
    %cria uma matriz de zeros com dimencoes do tamanho alfabeto

    for i = 1:length(query)
        %para a fonte query inteira
        posX = find(alfabeto == query(i));
        %guarda o indice linear de todas as ocorrencias do s?mbolo querry(i) no alfabeto
        posY = find(alfabeto == window(i));
        %guarda o indice linear de todas as ocorrencias do s?mbolo window(i) no alfabeto
        p(posX,posY) = p(posX,posY) + 1;
        %Aumenta +1 de tamanho na janela
    end

    p = p(:);
    hConj = entropia(p);
    %calcula entropia conjunta
    val = h1 + h2 - hConj;
    %formula Informa??o M?tua

    %devolve val
end
```

Exercício 6b)

Nesta alínea, usámos a rotina desenvolvida na alínea anterior para procurar a informação mútua entre o ficheiro de querry "guitarSolo.wav" e os ficheiros de target "target01 - repeat.wav" e "target02 – repeatNoise.wav" e determinar a variação de informação mútua entre eles.

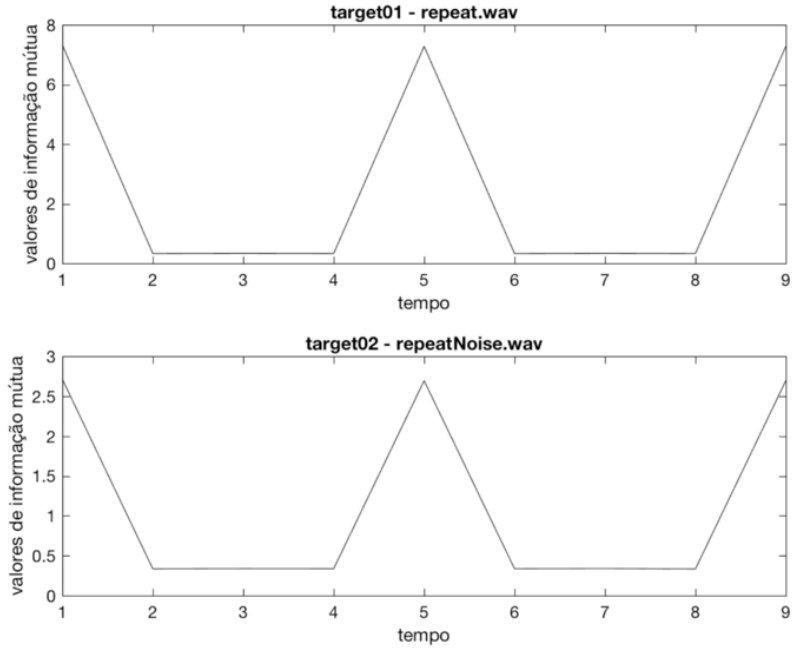
Resultados:

"guitarSolo.wav" e "target01 - repeat.wav"

[7.3207, 0.3289, 0.3354, 0.3297, 7.2789, 0.3289, 0.3355, 0.3293, 7.2806]

"guitarSolo.wav" e "target02 - repeatNoise.wav"

[2.7113, 0.3354, 0.3379, 0.3366, 2.6934, 0.3363, 0.3388, 0.3340, 2.6946]



Podemos assim concluir que há mais informação mútua entre o targert01 – repeat.wav e o guitarSolo.wav. No ficheiro target02 a informação mútua é menor e isto pode dever-se à insersão de ruído no audio, o que o faz afastar mais mais do original encontrando menos informação comum entre os fichieros de audio.

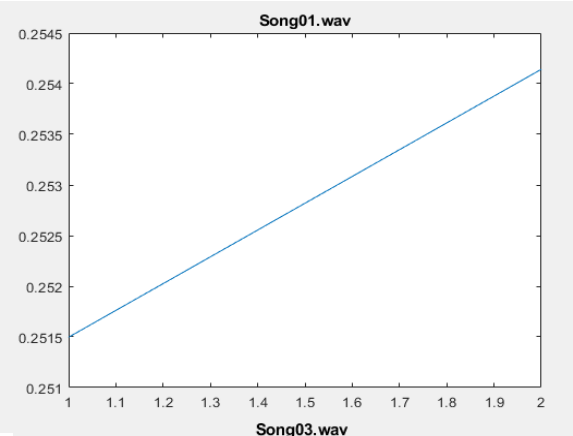
Exercício 6c)

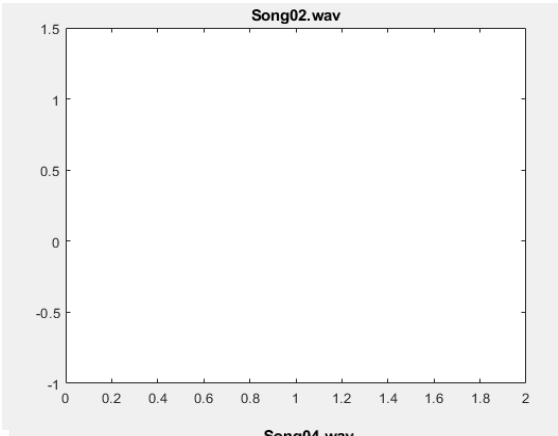
Neste exercício usamos a função informação mútua para comparar a query "guitarSolo.wav" desta vez com os 9 ficheiros "Song0nº.wav" como targets .Obtemos o máximo de informação mútua de cada um.

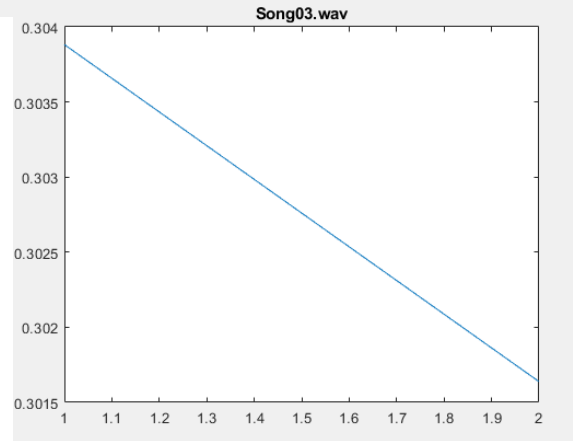
Assim, temos que, o valor máximo para cada ficheiro áudio foi:

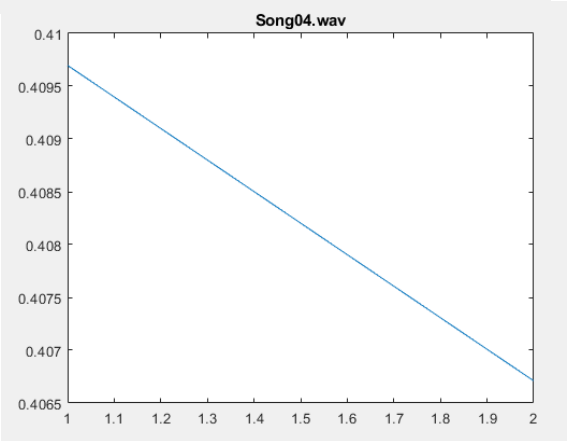
	Ficheiro WAVE	*Informação Mútua*
maior	*Song06.wav*	7.3384
	Song07.wav	6.3104
	Song08.wav	5.0731
	Song09.wav	4.5108
	Song05.wav	3.9617
	Song04.wav	0.4097
	Song02.wav	0.3777
	Song03.wav	0.3039
menor	*Song01.wav*	0.2541

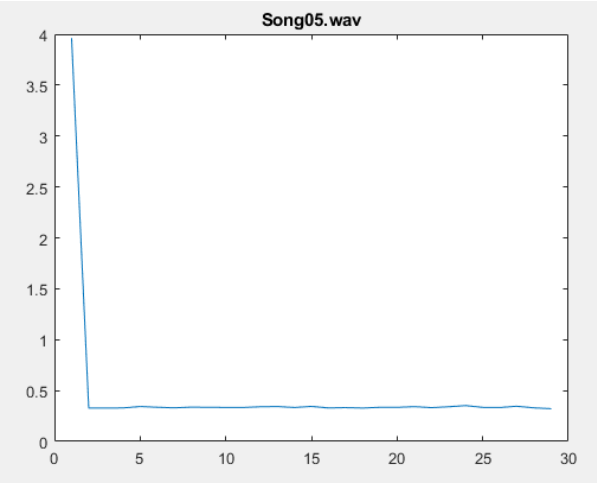
Com esta informação podemos avaliar que os ficheiros mais parecidos ao ficheiro guitarSolo.wav são os Song06.wav e Song07.wav

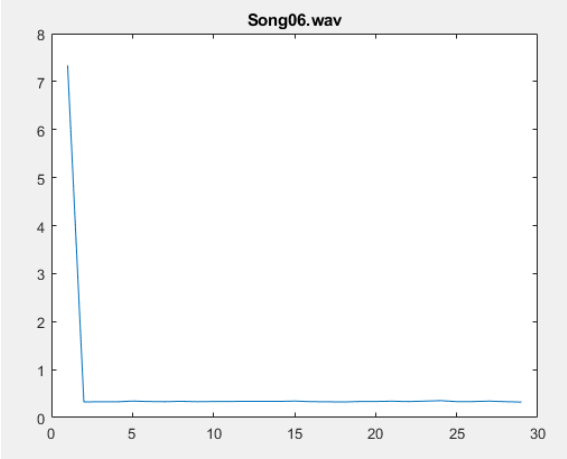


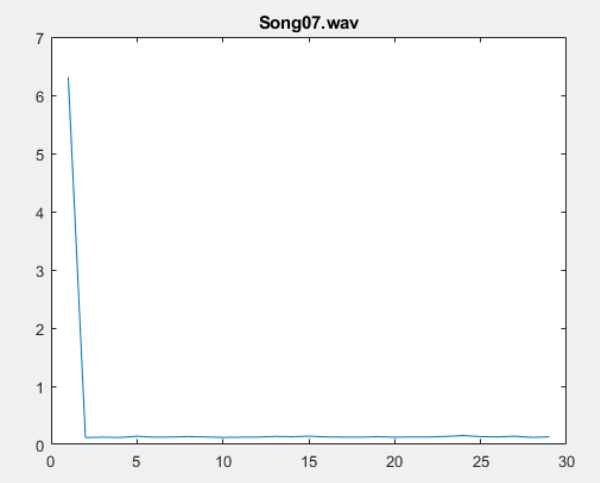


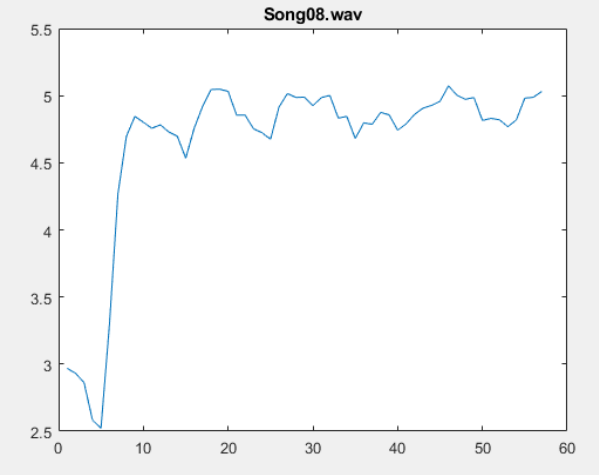


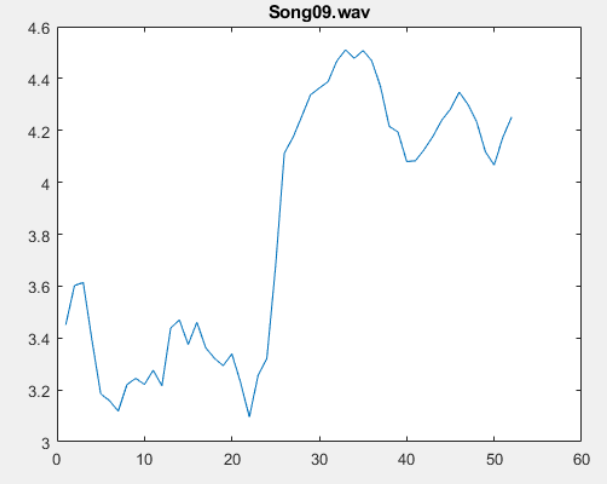



Song05.wav
4
3.5
3
2.5
2
1.5
1
0.5
0
0
5
10
15
20
25
30


Song06.wav
8
7
6
5
4
3
2
1
0
0
5
10
15
20
25
30


Song07.wav
7
6
5
4
3
2
1
0
0
5
10
15
20
25
30


Song08.wav
5.5
5
4.5
4
3.5
3
2.5
0
10
20
30
40
50
60


Song09.wav
4.6
4.4
4.2
4
3.8
3.6
3.4
3.2
3
0
10
20
30
40
50
60

Conclusões:

Com este trabalho conseguimos aprofundar os conceitos esperados bem como os conhecimentos de MATLAB, ao aplicar os conceitos teóricos em perguntas de carácter prático.

Com a análise dos histogramas e comparação de resultados podemos concluir que a entropia está relacionada com a dispersão do histograma e com a variedade de símbolos de uma fonte.

Concluímos também que os códigos de Huffman apesar de eficientes apresentam algumas falhas uma vez que não atingem o limite mínimo teórico para o número de bits por símbolo, sendo que a variância pode reduzir-se se houver uma associação de símbolos colocada o mais acima da árvore de Huffman possível.

Para além disso podemos também concluir que a entropia agrupada é menor que a entropia havendo uma otimização do código quando agrupamos.

Finalizando, tendo dois ficheiros de áudio diferentes podemos compará-los e calcular a sua informação mútua, ou seja, ver onde os áudios são semelhantes como no programa Shazam.