Relatório

Trabalho prático nº1

Entropia, Redundância e Informação Mútua

Teoria da Informação

Engenharia Informática

2º Ano

Diogo Honório, 2021232043

Gabriel Ferreira, 2021242097

Mateus Queiroz, 2021214102

UNIVERSIDADE D
COIMBRA

Introdução

No âmbito da disciplina de Teoria da Informação, realizamos diversos exercícios para adquirir sensibilidade para as questões fundamentais de teoria da informação, mais concretamente: informação, redundância, entropia e informação mútua.

Exercício 1

Neste exercício, recorremos a duas funções para contar as ocorrências de cada símbolo, *ocorrencias* e *ocorrenciasTXT*, que devolvem o *array* com o número de ocorrências para a imagem e som e para o texto, respetivamente. Decidimos pôr de lado o texto das restantes fontes de informação, de modo a facilitar a contagem na imagem e no som.

As duas funções recebem, como argumentos, a fonte de informação e o respetivo alfabeto, e é criado um *array* com o número de ocorrências para cada símbolo do alfabeto. No caso do texto, o alfabeto é percorrido e, usando a função *np.where*, encontra-se quantas vezes cada símbolo é encontrado na fonte de informação. Para o som e para a imagem, percorremos a fonte de informação, e adicionamos uma unidade para cada ocorrência.

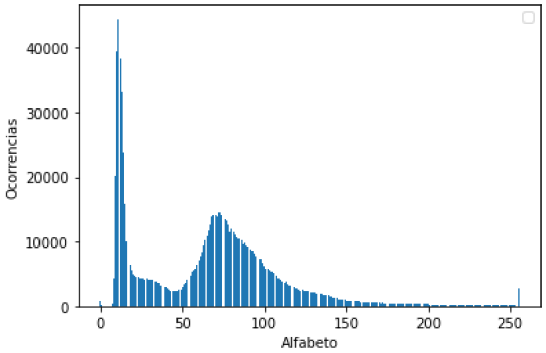
Para visualizarmos o resultado, utilizamos a função *histograma_ocorrencias*. Se estivermos perante um ficheiro de texto, o alfabeto é convertido para caracteres, usando a função *alfabetoToChar*. Por fim, é apresentado um gráfico de barras com o alfabeto no eixo *xx* e as ocorrências no eixo *yy*.

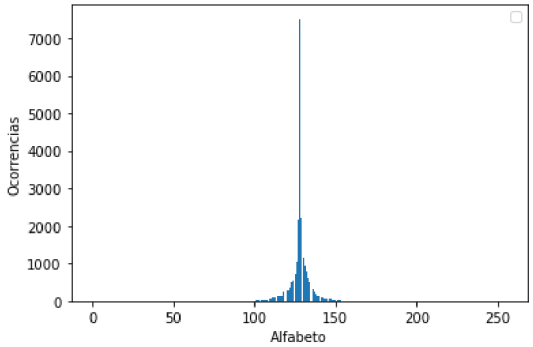
Exercício 2

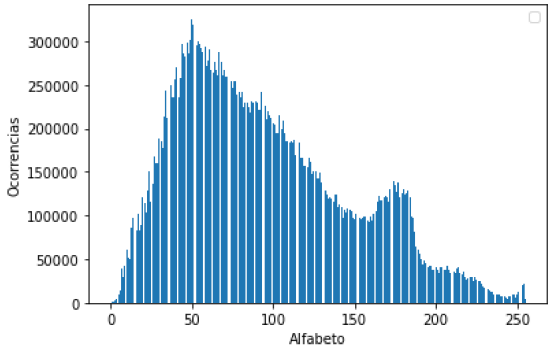
Para resolvermos este exercício, usamos a função *entropia* com o objetivo de calcular o limite mínimo teórico para o número médio de bits por símbolo. A função recebe um *array* contendo o número de ocorrências, e a partir daí, calcula-se a probabilidade de cada símbolo para depois obter o valor da entropia, de acordo com a seguinte fórmula: $H = \sum p_i * \log2 (1/p_i)$

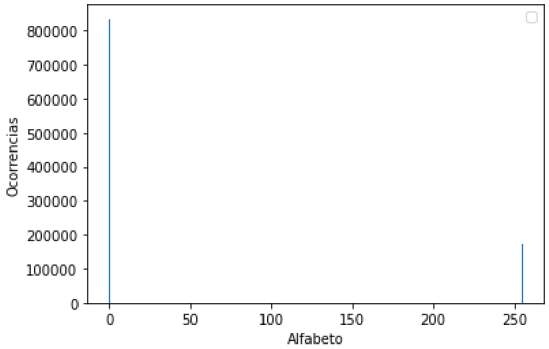
Exercício 3

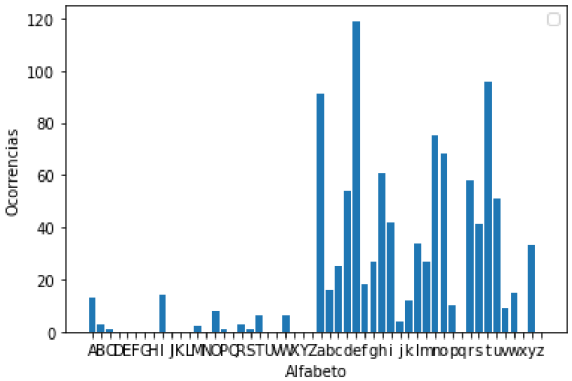
Nesta etapa, recorrendo aos exercícios anteriores, determinamos a distribuição estatística e o limite mínimo para o número médio de bits por símbolo das seguintes fontes:











O ficheiro com maior entropia é “landscape.bmp” que tem o gráfico mais uniforme das 3 e disperso, sendo necessário mais bits para guardar cada símbolo.

A imagem “MRI.bmp” tem um gráfico menos disperso do que a imagem anterior e, consequentemente, uma menor entropia.

Por sua vez, a imagem “MRIbin.bmp” tem apenas 2 símbolos: preto e branco, fazendo com que a sua entropia seja menor que 1, dado que a sua entropia máxima é 1 ($\log_2 2$), caso que aconteceria se o número de pixéis ‘0’ fosse igual ao número de pixéis ‘1’, o que não se verifica.

Apurou-se que o ficheiro de som apresenta picos de incidência que são muito significativos.

No ficheiro de texto constatamos que as letras minúsculas têm mais ocorrências que as letras maiúsculas.

Ficheiro	Entropia
landscape.bmp	7,61
MRI.bmp	6,86
MRIbin.bmp	0,66
soundMono.wav	4,07
lyrics.txt	4,41

Podemos então confirmar a teoria: gráficos mais dispersos e uniformes correspondem a maiores entropias, ao passo que gráficos menos uniformes correspondem a entropias mais baixas.

É possível comprimir fontes de forma não destrutiva, desde que a codificação seja unicamente decodificável. Sendo que a compressão máxima que se consegue alcançar corresponde à entropia arredondada por excesso às unidades.

Exercício 4

Neste exercício, para calcular o número médio de bits por símbolo utilizando código de *Huffman*, usamos as funções do ficheiro huffmancodec.py disponibilizado para obter os *arrays* com os símbolos e os respetivos comprimentos. Na prática, estamos a calcular a média ponderada, dado que consideramos o peso de cada símbolo, sendo o peso = número de ocorrências do símbolo / total. Assim, a média ponderada = ∑ peso * comprimento.

De seguida, calculou-se a variância ponderada recorrendo ao resultado obtido anteriormente.

Ficheiro	*Huffman*	Variância
landscape.bmp	7,63	0,75
MRI.bmp	6,89	2,19
MRIbin.bmp	1	0
soundMono.wav	4,11	4,35
lyrics.txt	4,44	1,08

A imagem “landscape.bmp” apresenta resultados muito similares no que diz respeito à codificação de *Huffman* e à entropia.

A imagem “MRI.bmp” também obteve resultados parecidos.

Quanto à imagem “MRIbin.bmp”, é possível inferir que para imagens binárias este algoritmo não tem nenhuma vantagem, pois cada símbolo ocupa 1 bit, assim como na codificação de *Huffman*.

No ficheiro de áudio também é possível verificar que a entropia e a codificação de *Huffman* apresentam valores próximos.

O ficheiro de texto, à semelhança do áudio e das duas primeiras imagens, também apresenta resultados muito idênticos quanto à entropia e à codificação de *Huffman*.

Quanto à variância, é possível constatar que todos os ficheiros, à exceção da imagem binária, não apresentam variâncias muito elevadas, embora os ficheiros “MRI.bmp” e “soundMono.wav” valores um pouco mais altos em relação aos restantes ficheiros, e que analisando o gráfico, é possível observar que existem pixéis com muitas ocorrências.

O ficheiro “MRIbin.bmp” tem variância 0, uma vez que o comprimento dos pixéis são iguais.

Na maioria dos casos é possível reduzir-se a variância criando árvores de *Huffman* com menor profundidade. Esta mudança, será útil no caso de utilização de buffers, exemplificando. Estes buffers são bastante usados na reprodução de vídeo e, quanto menor é a variância dos comprimentos de cada símbolo, mais fácil é prever o tamanho do buffer, evitando que este seja totalmente preenchido, caso os símbolos tenham comprimentos muito grandes.

Exercício 5

Neste exercício, tínhamos como objetivo calcular a entropia utilizando agrupamentos de dois símbolos consecutivos. Então, decidimos criar um alfabeto com o quadrado do tamanho do original e, por fim, calcular o número de ocorrências que cada um destes símbolos do novo alfabeto ocorre.

Primeiramente o alfabeto com os símbolos agrupados dois a dois foi obtido segundo a seguinte fórmula: 1º símbolo * dimensão do alfabeto + 2º símbolo.

Ficheiro	Entropia	Entropia do Agrupamento
landscape.bmp	7,61	6.204
MRI.bmp	6,86	5.193
MRIbin.bmp	0,66	0.402
soundMono.wav	4,07	3.310
lyrics.txt	4,41	

Logo, podemos concluir que a entropia calculada com os símbolos agrupados é sempre menor que a entropia calculada no exercício 3. Confirmando, deste modo, que $H(X,Y) <= H(X) + H(Y)$.

A exceção de que $H(X,Y) = H(X) + H(Y)$, apenas é verdade no caso de H(X) e H(Y) serem independentes, no entanto, em maior parte das fontes de informação isto não acontece.

De notar que para o ficheiro lyrics.txt, nós obtivemos um valor de 0 para a entropia o que sabemos que teoricamente é impossível ser esse o valor, então decidimos não colocar nada e deixar a nota que certamente por um erro de código não conseguimos calcular a entropia do agrupamento para o ficheiro lyrics.txt.

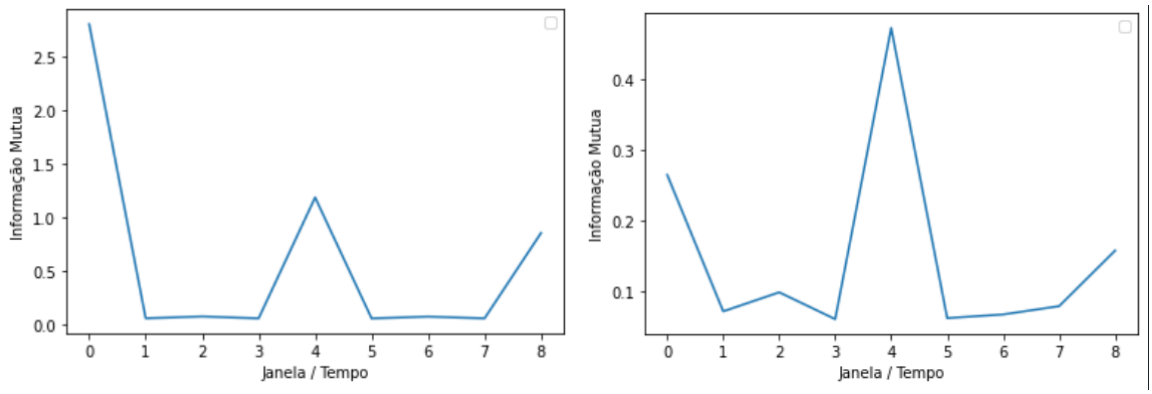
Assim, segundo os resultados apresentados, a entropia diminuiu quando calculada com agrupamentos, tal como expectável.

Exercício 6

Para a alínea a) e b), queremos calcular a informação mútua que é dada pela seguinte fórmula: I(X,Y) = H(X) + H(Y) - H(X,Y). Através da função “informacaoMutua”, calculamos a entropia de X, entropia de Y e a probabilidade conjunta, sendo que esta foi utilizada para calcular a entropia conjunta.

Na alínea a), calcula-se a informação mútua entre dois *arrays* com os valores já predefinidos do enunciado. Estes *arrays* simulam um *query* e um *target*, sendo também predefinido o alfabeto e o passo.

Na alínea b), calcula-se a informação mútua entre a *query* e cada zona do *target*.



Tanto o target01 como o target02 são representações do mesmo som, a única diferença é o ruído, ao analisar os gráficos podemos reparar que o gráfico que representa o target01(gráfico da esquerda) tem uma informação mútua maior, ou seja, tem um ruído menor, enquanto que o target02 já tem um ruído maior entre os [0,3]s e [5,8]s, já entre o os [3,5]s vemos que há um pico na informação mútua, esse momento é onde a presença do ruído já não é tão forte. O ruído do target02 passa a ter uma intensidade semelhante ao ruído do target01 nesse intervalo de tempo([3,5]s) e assim desta forma a informação mútua toma valores iguais nos gráficos.

Na alínea c), começamos por implementar uma função *simulador*, esta função tem como objetivo funcionar de igual modo à função *percorrerOtarget*. No entanto, recebe o endereço do ficheiro que irá funcionar como target.

Os resultados obtidos são depois devolvidos por ordem crescente num *array* e impressos por ordem decrescente.

Ficheiro	Informação mútua entre Ficheiro e *saxriff.wav*
Song01.wav	0.1363
Song02.wav	0.1890
Song03.wav	0.1675
Song04.wav	0.1905
Song05.wav	
Song06.wav	0.5704
Song07.wav	1.0039

Podemos observar que os ficheiros “Song06.wav” e “Song07.wav” possuem valores de informação mútua elevados. Este resultado é previsível, visto que ambos os áudios são bastante parecidos com a *query* (*saxriff.wav*).

Porém, os ficheiros “Song01.wav”, “Song02.wav”, “Song03.wav” e “Song04.wav”, apresentavam valores de informação mútua baixos, isto deve-se ao facto de estes ficheiros não possuírem semelhanças com o ficheiro *saxriff.wav*.

De notar também que para o ficheiro "Song05.wav" obtivemos uma informação mútua de valor negativo o que sabemos que pela teórica seria impossível então decidimos também deixar de fora e deixamos em forma de nota que devido a algum problema que nós não conseguimos encontrar/resolver o resultado estava sempre a dar negativo e não podia ser.

Conclusão

Com a realização deste trabalho, conseguimos compreender melhor os conceitos de teoria de informação, mais concretamente: informação, redundância, entropia e informação mútua.