



UNIVERSIDADE D
COIMBRA

Relatório do Trabalho Prático Nº 1

Teoria da Informação

Feito por:

Filipe Rodrigues

Nº 2021228054

Joás Silva

Nº 2021226149

André Louro

Nº 2021232388

Relatório

Exercício 1:

Para criar o histograma da ocorrência de símbolos, foi utilizada a função **histograma**, que possui como parâmetros o alfabeto, a fonte (informação) e o title que é o nome do ficheiro a ser analisado (vai ser utilizado depois no exercício 3).

A função cria um dicionário do histograma (com todas as chaves do alfabeto) e inicializa todos a zero e com a função *Counter*, importada, obtêm-se os valores das chaves que são o número de ocorrências do símbolo na fonte.

Exercício 2:

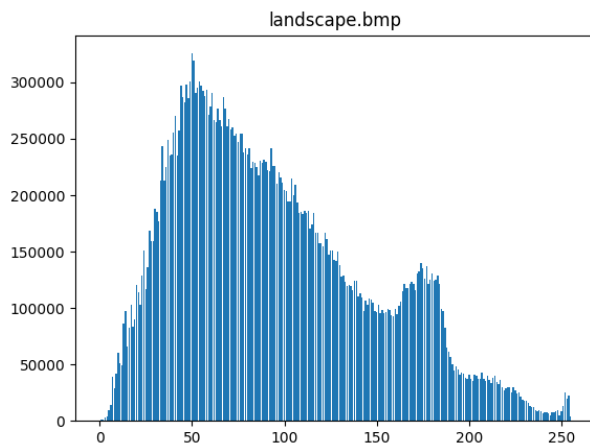
Para determinar o limite mínimo teórico para o número médio de bits por símbolo, foi utilizada a função **entropia**, esta função possui como parâmetro a fonte (informação), nela foi criado um array com as ocorrências de cada um

dos símbolos da fonte e utilizamos a fórmula da entropia $-\sum_{i=0}^n p_i \log_2(p_i)$ para calcular a entropia.

Exercício 3:

Para resolver este exercício foi utilizada a função **analyseFile**, que recebe como parâmetros a variável src (source do ficheiro que se pretende analisar), nela obtemos a variável fonte (através da função **getFonte**) e a variável alfabeto (através da função **getAlfabeto**), sendo que estas serão utilizadas para obter o histograma e a entropia correspondente a cada src, através das funções **histograma** e **entropia** criadas anteriormente (Exercícios 1 e 2). A função **getFonte** devolve uma variável fonte, que corresponde à leitura do ficheiro src dependendo da sua extensão (.bpm/.wav/.txt), e a função **getAlfabeto** devolve o alfabeto correspondente a cada tipo de src.

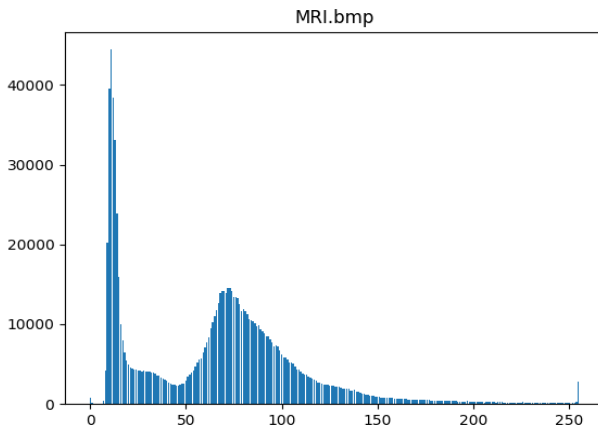
- landscape.bmp



Devido a dispersão não igualitária da quantidade de pixels por tom de cinza, a entropia da imagem é bastante elevada.

Entropia = 7.606914077203874 bits/símbolo

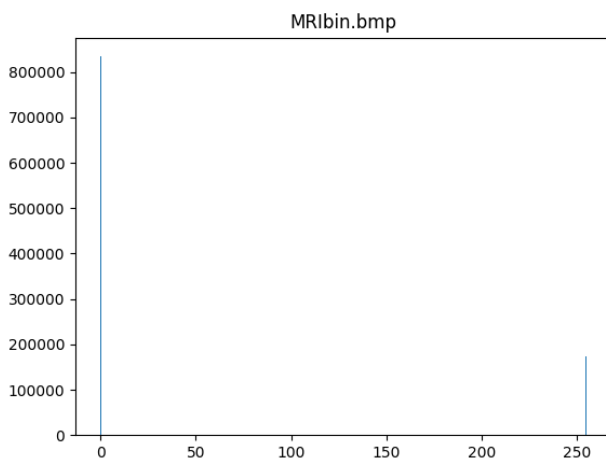
- MRI.bmp



Devido a quantidade elevada de pixels pretos, a entropia dos símbolos da imagem é elevada, havendo um pico enorme no histograma.

Entropia = 6.8605419847961215 bits/símbolo

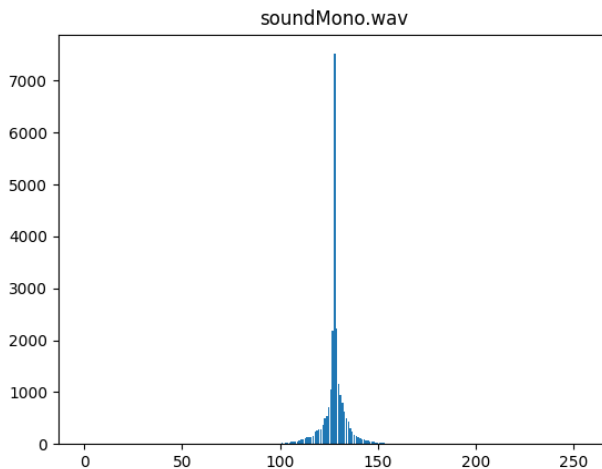
- MRIBin.bmp



Pelo facto de apenas haver 2 cores na imagem, a entropia da imagem é quase mínima, seria mínima caso o número de ocorrências de pixels pretos fosse igual ao número de ocorrências de pixels brancos.

Entropia = 0.6610803299918834 bits/símbolo

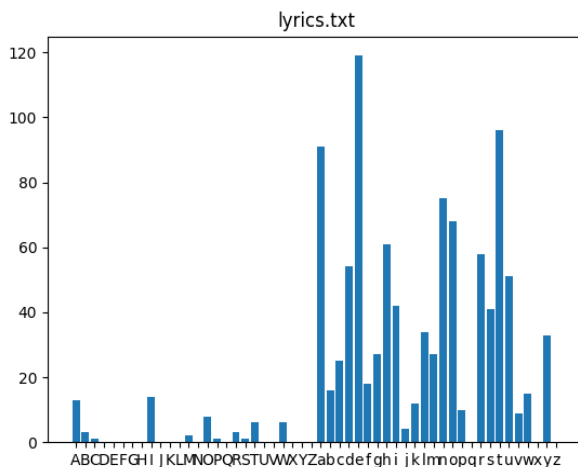
- soundMono.wav



Devido ao elevado número de ocorrências dos símbolos entre 125 e 133, há um grande pico no histograma, que é a causa do valor maior que necessário para a entropia.

Entropia = 4.065729167841344 bits/símbolo

- lyrics.txt



Obtemos valores mais elevados no lado correspondente às letras minúsculas sendo que estas são as mais frequentes durante as lyrics.txt.

Devido à elevada diversidade de caracteres a entropia terá um valor mais elevado que o necessário.

Entropia = 4.41070522496444 bits/símbolo

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

Resposta: Sendo a entropia máxima igual ao \log_2 do tamanho da fonte, o valor da entropia máxima para as imagens seria $\log_2 256$ igual a 8, e para o texto seria $\log_2 52$ igual a 5,7004.

Podemos reparar que nenhuma das entropias obtidas foi igual ao valor da entropia máxima, assim confirma-se que as fontes podem ser comprimidas de forma não destrutiva.

Inicialmente cada símbolo é guardado em 8 bits. A entropia do ficheiro *landscape* é de 7,606914 bits/símbolo que arredondado seria também 8 bits/símbolo ou seja não daria para comprimir.

A entropia do ficheiro *MRI* é aproximadamente 6,860542 bits/símbolo e arredondado seria 7 bits/símbolo, o que neste caso significa que é possível efectuar a compressão deste ficheiro. Usando assim o mesmo processo, o ficheiro *MRIbin* passaria 0.66 bits/símbolo para apenas 1 bits/símbolo, o ficheiro *soundMono* de 4.066 bits/símbolo para 5 bits/símbolo, e por fim o ficheiro *lyrics* de 4.4107 bits/símbolo para 5 bits/símbolo.

Exercício 4:

Para determinar o número médio de bits por símbolo para cada uma das fontes de informação utilizando as funções de codificação de Huffman, foi usada a função ***analyseHuffman***, esta função recebe como parâmetro a variável src (source), nela foi criado um dicionário em que as keys são o alfabeto e os values são o número de ocorrências das keys, guardado na variável fonte (utilizando a função ***getFonte***), de seguida utilizando a codificação de Huffman obtemos as variáveis symbols e length, em que a variável symbols é utilizada para ordenar as keys da variável fonte, sendo que os valores correspondentes a estas keys serão guardados num array chamado ocorrencias. Seguidamente utilizamos mais duas funções com o intuito de calcular a média ponderada e a variância ponderada, estas funções (***mediaP*** e ***variânciaP***), recebem como parâmetros os arrays ocorrencias e length e calculam os valores pretendidos através das fórmulas:

$$E(X) = \sum_{x \in X} \frac{x(p_x)}{Total_{ocorrencias}} \text{ e } V(X) = E(X^2) - [E(X)]^2$$

- *landscape.bpm*

Média Ponderada: 7.6293007128362165

Variância Ponderada: 0.7516160479312148

A variância do comprimento do código de Huffman da imagem é pouca, pois o desvio médio dos valores é pequeno.

- *MRI.bpm*

Média Ponderada: 6.890995928904776

Variância Ponderada: 2.193080876857614

A variância do comprimento do código de Huffman da imagem é consideravelmente grande pois o desvio médio dos valores é elevado.

- MRIbin.bpm

Média ponderada: 1.0

Variância ponderada: 0.0

A variância do comprimento do código de Huffman da imagem é zero pois ao utilizar a codificação de Huffman, necessariamente cada um dos símbolos será representado com um bit (0 ou 1), logo não haverá desvio em relação ao comprimento do código.

- soundMono.wav

Média ponderada: 4.110713829651385

Variância ponderada: 4.354958214322426

A variância do comprimento do código de Huffman da imagem é maior que a média ponderada, pois o desvio médio dos valores é enorme devido a elevada quantidade de símbolos entre 125 e 133.


- lyrics.txt

Média ponderada: 4.443486590038314

Variância ponderada: 1.0820552766400944

A variância do comprimento do código de Huffman da imagem é pouca, pois o desvio médio dos valores é pequeno.

Após aplicar a codificação de Huffman em cada um dos ficheiros, a média ponderada de bits por símbolo é quase semelhante à entropia calculada no exercício anterior, o que significa que o número de bits a ser utilizado em média é quase mínimo.

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

Resposta:

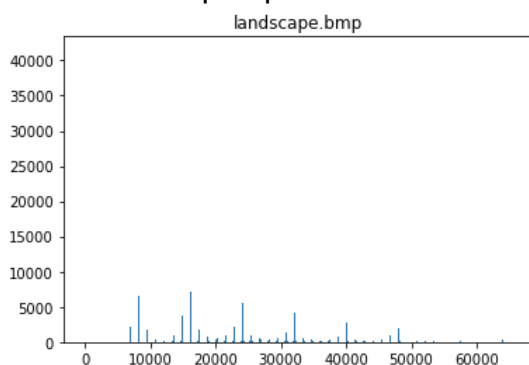
O código fornecido para realizar este exercício já tem uma variância mínima, o que significa que não é possível reduzir ainda mais.

Reduzir a variância é útil na forma em que fica melhor para transmitir informação.

Exercício 5:

Para resolver o exercício 5 foi usado duas funções, a função **getAlfabetoPairs** para obter o alfabeto com o agrupamento de símbolos da respectiva src através de uma condição *if* irá diferenciar as src que terminam em .txt das que terminam em .bpm ou em .wav, sendo que para as terminadas em .txt irão devolver a junção em pares do alfabeto anterior através do Unicode (numpy.unicode_), enquanto as terminadas em .bpm ou .wav irão devolver um alfabeto possível para números de 16 bits, 0 até 256^2 . E a função **analyseFilePairs** que para as *src* que acabam em .txt irá juntar os seus caracteres em pares pelo seu Unicode, enquanto as terminadas em .bpm ou .wav irão fazer a junção em pares utilizando *shifts* de 8 bits para a esquerda (<<) e adicionando aos 8 bits que ficarão à direita o segundo número, mas independente da sua extensão em ambos casos estas junções serão guardadas na variável fonte, de seguida irá utilizar a função anteriormente descrita para obter o alfabeto agrupado em pares e utiliza a função **histograma** e **entropia** para obter os resultados pretendidos para estes alfabetos e fontes.

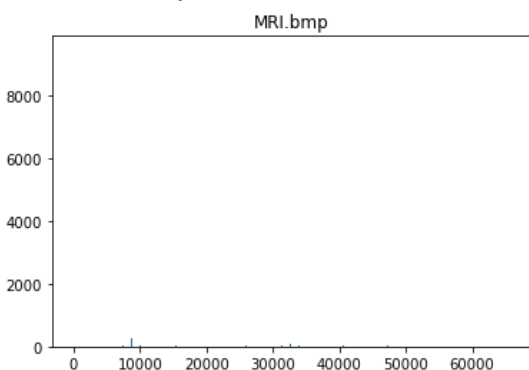
- landscape.bpm



Dado a melhor distribuição de símbolos (ao juntar os valores de 2 a 2 temos maior dispersão de símbolos) a entropia é menor que a entropia da imagem com os símbolos analisados 1 a 1.

Entropia em pares: 6.2040596930972045 bits/símbolo

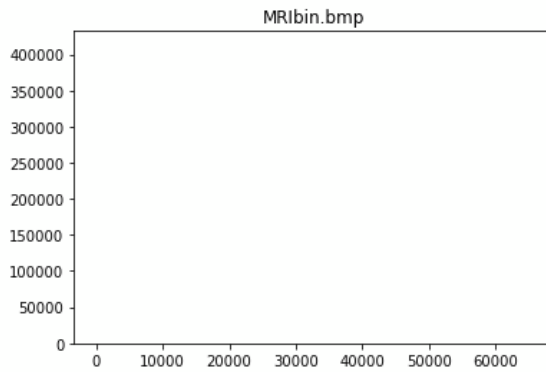
- MRI.bpm



Não é possível ver bem no histograma devido ao número elevado de duplas de bits com zero ocorrências, mas ele tecnicamente está melhor distribuído que o ficheiro quando os símbolos são analisados individualmente, logo a sua entropia é muito menor.

Entropia em pares: 5.193602970171974 bits/símbolo

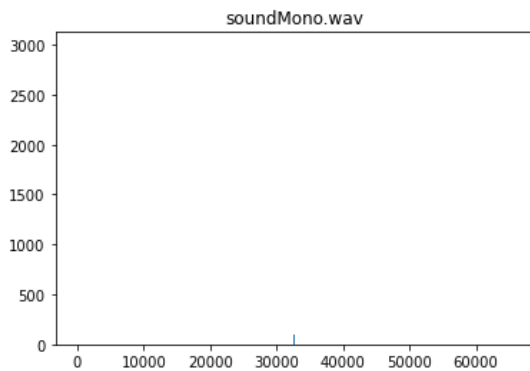
- MRIbin.bpm



É impossível de ver no histograma, mas este possui apenas valores em 0 (ambos os símbolos são pixels pretos [0]), em 255 (símbolo da esquerda é um pixel preto [0] e símbolo da direita é um pixel branco [255]), em 65280 (símbolo da esquerda é um pixel branco [255] e símbolo da direita é um pixel preto [0]) e em 65535 (ambos os símbolos são pixels brancos [255]), sendo eles mais bem distribuídos que na imagem com os símbolos analisados 1 a 1, logo a sua entropia é menor.

Entropia em pares: 0.4028703654483534 bits/símbolo

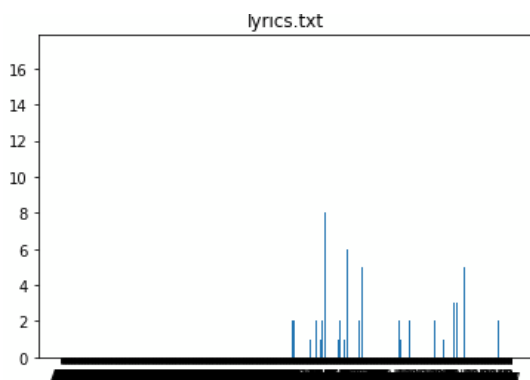
- soundMono.wav



Não é possível ver pelo histograma, mas dada a melhor distribuição de símbolos (ao juntar os valores de 2 a 2 temos maior dispersão de símbolos) a entropia é menor que a entropia do áudio com os símbolos analisados 1 a 1.

Entropia em pares: 3.3109957723669114 bits/símbolo

- lyrics.txt



No caso do ficheiro *lyrics.txt*, a diferença de entropia não é tão grande devido a distribuição dos símbolos no histograma, provavelmente causado por haver maior frequência de símbolos com vogais (ex: do ou da).

Entropia em pares: 3.6521800559086772 bits/símbolo

Exercício 6:

A função ***mutualInformation*** recebe como parâmetros a query, target, alfabeto, step e retorna um array com todas as informações mútuas entre a query e o target. Esta função percorre um loop que vai enchendo a lista de informações mútuas com a fórmula $I(X|Y) = H(X) + H(Y) - H(X,Y)$.

Para fazer $H(X,Y)$ juntou-se a informação da mesma forma que no exercício 5 e depois chamou-se a função entropia criada anteriormente para o X, Y e a junção de (X, Y).

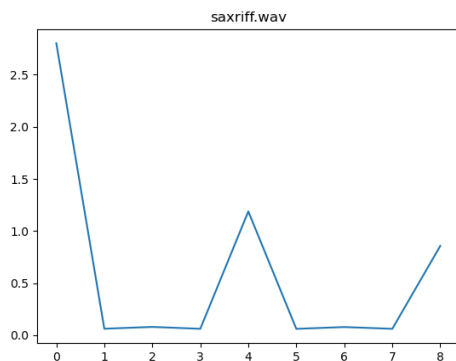
A função ***evolucaoInformacaoMutua*** recebe como parâmetros a info e o title, que seriam o array da informação mútua e o título para dar ao gráfico.

Alínea a):

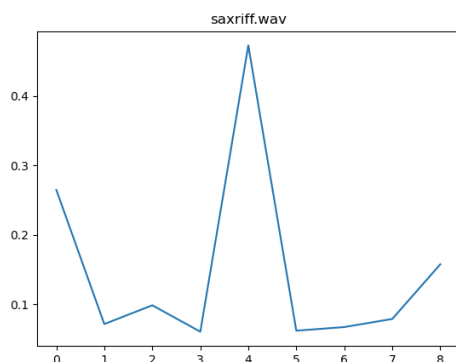
A função ***alineaA***, utilizando uma query, um target, um alfabeto, e um step, estando estes com valores predefinidos (hardcoded), vai usar a função ***mutualInformation*** e guardar a informação obtida na variável infoMutua, sendo que de seguida irá imprimir esta informação guardada.

Alínea b):

Na função ***alineaB***, foi criada uma query, um passo, um alfabeto, um target1 com a correspondente infoMutua01, um target2 com a correspondente infoMutua02, de forma a alcançar o resultado esperado. Nas variáveis query, target1 e target2 foi utilizado as funções ***numpy.hsplit*** e ***.flatten*** de modo a conseguir utilizar apenas um canal dos áudios utilizados. De forma a apresentar os resultados esperados foram imprimidos as variáveis infoMutua01 e infoMutua02 (variáveis obtidas através da utilização da função ***mutualInformation***) e apresentamos os gráficos correspondentes utilizando a função ***evolucaoInformacaoMutua***.



Ao utilizar o ficheiro “*target01 - repeat.wav*”, no instante 0 é onde a query (“saxriff.wav”) é idêntica ao target o valor de informação mútua é máximo e dado o valor elevado pode-se admitir que o áudio “saxriff.wav” pertence ao target.



Já ao utilizar o ficheiro “*target02 - repeatNoise.wav*”, o instante 4 é onde a informação mútua é máxima, mas o seu valor é demasiado pequeno para se poder admitir que a query pertence ao target, o que faz sentido pois o áudio “target2” possui bastante ruído, que dificulta a assimilação dos áudios.

Alínea c):

A função **infoMaximos** recebe como parâmetros songs, query, alfabeto, passo e retorna um dicionário com os máximos ordenados por ordem decrescente.

Nesta função há um loop que percorre os songs em que para cada song a função verifica se estes têm mais do que uma cadeia de entrada, e se sim filtra-se só para o primeiro canal depois faz-se a informação mútua para cada song e adiciona-se ao dicionário. No final do loop é usado uma lambda function para ordenar o dicionário.

Informação Mútua Máxima de cada target:

'Song06. wav': 3.53098 8733765 1333	Song07. wav': 3.53098 8733765 1333	'Song05. wav': 0.53224 2406104 5883	'Song04. wav': 0.19058 7199908 04285	'Song02. wav': 0.18903 7908360 02443	'Song03. wav': 0.16756 7874394 74863	'Song01. wav': 0.13634 2461629 9831
---	--	---	--	--	--	---

Como se pode analisar, os áudios “Song06.wav” e “Song07.wav” têm informação mútua igual e elevada, logo, pode-se admitir que são áudios semelhantes à query, *saxriff.wav*. Os restantes áudios possuem um valor máximo de informação mútua baixo, logo não se pode dizer que exista uma grande semelhança entre a query e estes.