UNIVERSIDADE D COIMBRA FACULDADE DE CIÊNCIAS E TECNOLOGIA

RELATÓRIO DO TRABALHO PRÁTICO Nº1

LICENCIATURA DE ENGENHARIA INFORMÁTICA – TECNOLOGIA DA INFORMAÇÃO

CARLOS MATOS 2020245868

MARIANA MAGUEIJO 2020246886

JOÃO PINO 2020210945

ÍNDICE

Página

INTRODUÇÃO

Neste trabalho pretende-se adquirir conhecimento nas questões fundamentais da Teoria de Informação.

Com esse intuito realizámos exercícios para desenvolver ideias como a informação, informação mútua, entropia e redundância.

Utilizamos ferramentas como o numpy e o matplotlib para processar e representar os dados fornecidos pelos ficheiros: english.txt, guitarSolo.wav, homer.bmp, homerBin.bmp e kid.bmp

EXERCÍCIO 1

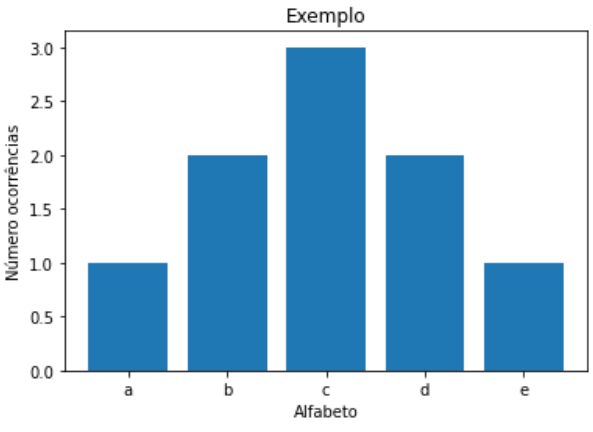
O primeiro exercício tem com objetivo a criação de um histograma que represente a ocorrência dos símbolos todos de uma fonte de informação P.

Para completar este exercício procedemos à criação de uma função chamada *histograma* que admite como argumentos P, a fonte de informação, e file, que é o nome do ficheiro.

Para representação dos dados é preciso uma variável que guarde o Alfabeto de uma amostra de símbolos. Para obter o alfabeto recorremos à função *numpy.unique* que devolve um array com todos os elementos únicos da fonte de informação que é introduzida como argumento. Ao introduzir nesta função de numpy o argumento "return_counts = True" a função também retorna o número de ocorrências de cada símbolo respetivamente.

O histograma consiste num plot construído pela biblioteca matplotlib com os elementos do alfabeto no eixo 0x e com a quantidade do respetivo símbolo, presente na fonte de informação, no eixo 0y.

Exemplo de um histograma:



Exemplo.txt:
"abbcccdde"

EXERCÍCIO 2

No segundo exercício pretende-se obter a entropia, que consiste no limite mínimo teórico para o número médio de bits por símbolo.

Para obter a entropia usámos a fórmula:

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

H(A) = entropia da fonte de informação A
P(ai) = Probabilidade de um acontecimento ai

Para aplicar a fórmula no programa prosseguimos com a criação de uma função chamada *entropia* que recebe a fonte de informação, *P*, e o array com o número de ocorrências, *no*, de cada símbolo devolvido pela *numpy.unique* da função do exercício anterior.

Para fazer o somatório com as probabilidades criámos um array de probabilidade de cada elemento. Começando por eliminar todos os zeros no array '*no*' pois estes têm uma probabilidade nula e não são contabilizados. A probabilidade de cada elemento é igual ao número de ocorrências desse elemento a dividir pelo número total de ocorrências, sum(no).

Por fim, através do numpy.sum pode se calcular o somatório de todos os elementos do array probabilidade multiplicados pelo logaritmo de base 2 deles mesmos e o seu número simétrico é o limite mínimo teórico para o número médio de bits por símbolo.

EXERCÍCIO 3

INTRODUÇÃO

Neste exercício aplicamos as funções desenvolvidas nos exercícios 1 e 2 aos ficheiros fornecidos, havendo 3 tipos de ficheiros: texto (.txt), áudio(.wav) e imagem(.bmp)

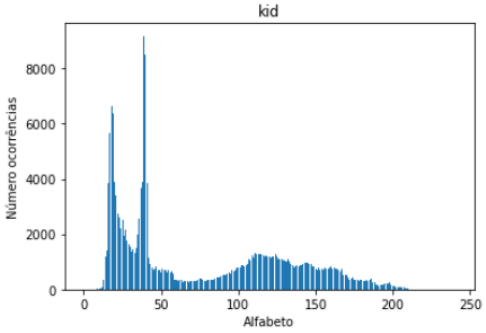
Para saber qual o tipo de ficheiro usamos a função .endswith() que devolve o número inteiro 1 se o argumento for igual ao sufixo do ficheiro introduzido. É importante saber qual o tipo de ficheiro para poder abrir com os módulos scipy.io.wavfile, no caso de um ficheiro .wav, e matplotlib.image, no caso de um ficheiro .bmp. No de texto usa-se as funções open() e read() do python.

ANÁLISE DAS FONTES

- **Imagem**

Como os ficheiros de imagens podem vir com várias dimensões, se o ficheiro tiver mais de 2 dimensões, como no caso de imagens RGB, utilizamos apenas as 2 primeiras dimensões e transformamos o array com a fonte de informação num array unidimensional através da função flatten() do numpy.

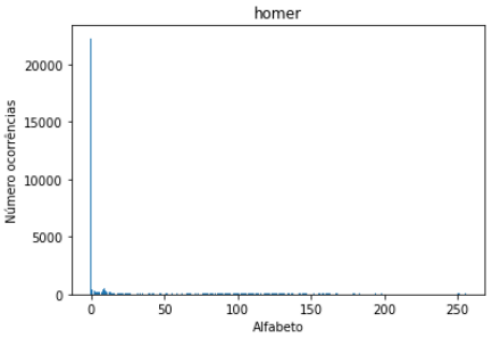
- Kid.bmp



Valor da entropia = 6.954143307171648 bits por símbolo.

Esta é a fotografia com a entropia mais alta visto que tem um número de ocorrências mais uniformemente distribuído.

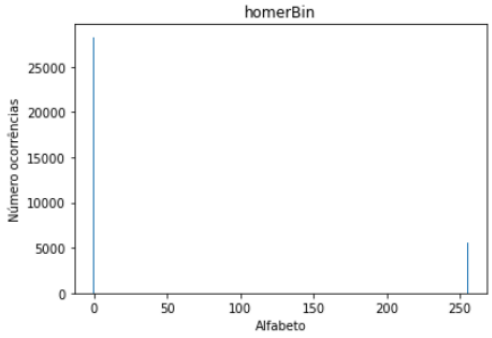
- Homer.bmp



Valor da entropia = 3.465865162708985 bits por símbolo.

Esta imagem por outro lado tem uma menor distribuição de ocorrências, tendo um ápice claro perto do 0.

- homerBin.bmp



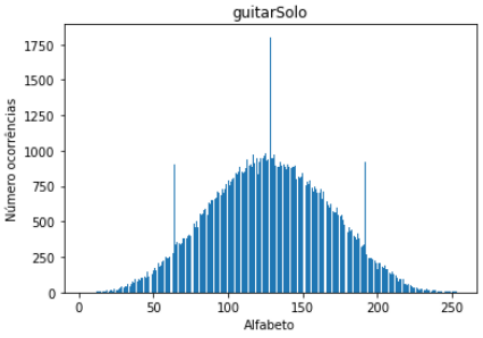
Valor da entropia = 0.6447813982002277 bits por símbolo.

Como esta é uma imagem com apenas dois valores possíveis, "binarizada", então observam-se 2 picos no histograma e vai ter uma entropia muito menor devido à reduzida distribuição das ocorrências.

- **Som**

Como as fontes de informação de ficheiros .wav podem vir com 1 ou 2 canais (mono ou stereo), fazemos uma condição e se essa for verdadeira usamos o primeiro canal como a fonte de informação.

- guitarSolo.wav



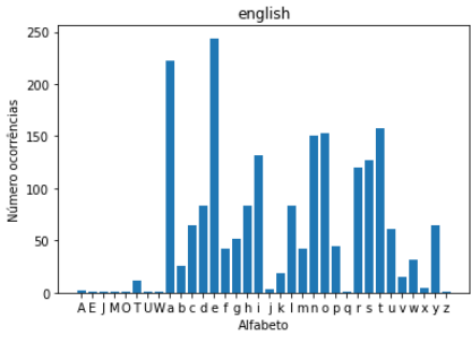
Valor da entropia = 7.329201669454548 bits por símbolo.

O ficheiro tem uma entropia alta pois vê-se uma distribuição alta das ocorrências.

- **Texto**

Como transformamos os caracteres em números temos que limitar os caracteres no texto para os que importam (a-z e A-Z). Para isso usamos a função trata_texto que apenas adiciona os caracteres desejados.

- english.txt



Valor da entropia = 4.227471302397896 bits por símbolo.

Através do histograma podemos ver que as letras minúsculas são mais usadas pois têm um maior número de ocorrências e os dois maiores picos são em vogais.

Pergunta 1

A compressão de imagens baseia-se na remoção de informação redundante existente nas imagens.

Existem dois tipos de compressão:

Destrutiva - no processo de compressão são perdidas características das imagens, o que permite obter graus de compressão mais elevados;
Não destrutiva - é possível reconstruir exatamente a imagem original antes de ter sido efetuada a compressão;

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

Para uma fonte ser comprimida de forma não destrutiva, o número médio de bits por símbolo tem de ser inferior ao valor da entropia máxima. Esta provém do tamanho do nosso alfabeto A, que no caso do áudio e da imagem é 256, e pela fórmula seguinte é 8. Para o ficheiro de texto.

Como todos os valores da entropia são inferiores a 8, é possível comprimir as fontes de forma não destrutiva.

$$H_{max}(X) = \sum_{i} \log_2 |A_x|$$

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

Para calcular o valor da taxa de compressão utilizamos a seguinte fórmula:

$$\frac{H_{\max}(X) - H(X)}{H_{\max}(X)} * 100$$

A partir da fórmula podemos observar os seguintes resultados:

Ficheiro	Entropia (bit/símbolo)	Compressão máxima (%)
kid.bmp	6.954143307171648	13.0732086603544
homer.bmp	3.465865162708985	56.6766854661376875
homerBin.bmp	0.6447813982002277	91.94023252249715375
guitarSolo.wav	7.329201669454548	8.38497913181815
english.txt	4.227471302397896	16.89657357189117

EXERCÍCIO 4

No exercício 4 utilizámos as funções de codificação Huffman fornecidas e determinámos a média e variância de cada ficheiro. A função média recebe a probabilidade, calculada anterior e o comprimento de cada um dos símbolos da fonte e calcula a média de bits por símbolo. A função variância admite a probabilidade, a média e o comprimento de cada um dos símbolos da fonte como argumentos e retorna a variância.

A partir dessas funções obtemos os seguintes resultados:

Ficheiro	Entropia	Média (bit/símbolo)	Variância
kid.bmp	6.954143307171648	6.983223014256619	2.0992959258247232
homer.bmp	3.465865162708985	3.548315602836879	13.19683817935466
homerBin.bmp	0.6447813982002277	1.0	0.0
guitarSolo.wav	7.329201669454548	7.3501579494561735	0.7274273982765376
english.txt	4.227471302397896	4.251219512195122	1.19103509815586

Comparando os resultados da entropia e da média podemos verificar que os valores são muito semelhantes, sendo os da média superiores, comprovando a fórmula:

$$H(S) \leq \ l \ \leq H(S) + 1$$

A variância de uma variável aleatória é uma medida de dispersão, indicando "o quão longe" em geral os seus valores se encontram do valor da média.

Pergunta 2

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

Colocando os símbolos por ordem crescente de ocorrências e representando os mais frequentes pelo menor número de bits, é possível reduzir a variância, tornando-se útil para otimizar o programa.

EXERCÍCIO 5

INTRODUÇÃO

Para este exercício repetimos as funções do 3 mas agora com os símbolos agrupados.

Com o intuito de agrupar criámos uma função chamada agrupamento de símbolos que admite a fonte de dados e nome do ficheiro como argumentos.

Se o primeiro elemento da fonte de informação for um caracter então o ficheiro é um ficheiro de texto. Nesse caso transformamos os elementos do array da fonte de informação em números.

Como temos de agrupar os elementos, se for um número ímpar de elementos teremos de ignorar o último elemento. Para ignorar esse membro, ao criar uma variável size subtraímos 1 ao tamanho da fonte de informação fornecida e dividimos tudo por 2 para obter o tamanho que a fonte, depois de agrupada, terá.

Para agrupar usamos a função reshape do numpy com 2 argumentos: size, que obtemos na condição anterior, e o número 2 que indica à função que queremos fazer um array de arrays bidimensionais, P.

Para criar um valor para cada símbolo usamos a seguinte fórmula:

$$A[i] = (agrupado[i][0] * data.size) + agrupado[i][1]$$

ANÁLISE DE RESULTADOS

Ficheiro	Entropia agrupada (bit/símbolo)
kid.bmp	4.909097962175789
homer.bmp	2.412733070535473
homerBin.bmp	0.39782409226242005
guitarSolo.wav	5.754384158334467
english.txt	3.659258888196971

Como houve uma diminuição da entropia em todos os casos, podemos concluir que o agrupamento otimiza o código pois vai haver uma redução do número médio de bits necessários por símbolo.

No entanto, como aumentamos o numero de símbolos o programa vai correr mais lentamente.

EXERCÍCIO 6

EXERCÍCIO 6A)

No exercício 6a é pedido para calcular a informação mútua entre o sinal a pesquisar (***query***) e o sinal onde pesquisar (***target***).

A informação mútua é calculada através da seguinte fórmula:

$$I(X,Y) = \mathrm{H}(X) + \mathrm{H}(Y) - \mathrm{H}(X,Y)$$

Com a esta fórmula podemos calcular um vetor de valores de informação para cada ***passo***. A função inf_mutua6a criamos um ciclo *while* em que incrementamos o passo e avançamos a janela que estamos a comparar com o target, criando assim um vetor da informação mútua entre o *query* e o *target*.

EXERCÍCIO 6B)

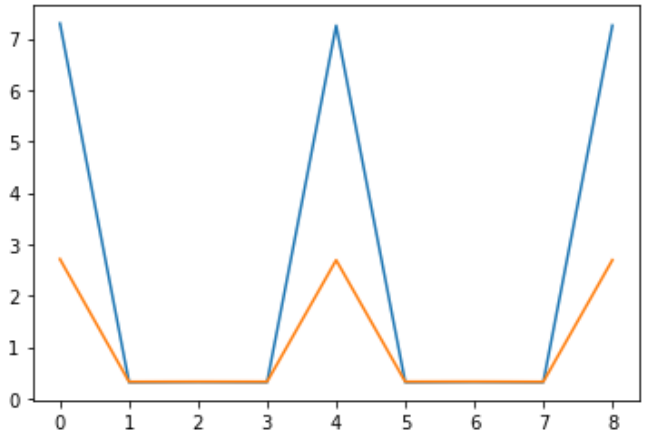
No 6b utilizamos o raciocínio anterior para procurar a informação mútua entre o ficheiro "guitarSolo.wav" e os ficheiros de target "target01 - repeat.wav" e "target02 – repeatNoise.wav" e determinar a variação de informação mútua entre eles. A função ler_audio retorna a fonte da query e do target para utilizar na função inf_mutua6.

Informação mútua de guitarSolo.wav e target01 - repeat.wav:

[7.2918073 0.31758715 0.32287857 0.31840882 7.25009521 0.31766061 .3230725 0.31801639 7.25181809]

Informação mútua de guitarSolo.wav e target02 – repeatNoise.wav:

[2.70912733 0.32609222 0.3283094 0.32694404 2.69088423 0.32761545 0.32936009 0.3253737 2.69194152]



Ao analisar o gráfico acima concluímos que a informação mútua (linha azul) entre o target01 – repeat.wav e o guitarSolo.wav é maior.

No ficheiro target02 – repeatNoise.wav a informação mútua é menor, o que pode dever-se há presença de ruído, levando à diminuição da informação mútua.

EXERCÍCIO 6C)

Nesta última alínea usamos os mesmos princípios dos dois últimos exercícios para comparar informação mútua do ficheiro guitarSolo.wav com os 7 ficheiros música. No final apenas imprimimos o valor máximo de informação mútua para cada música:

Ficheiro	Informação mútua máxima
Song01.wav	0.2515764966373677
Song02.wav	0.3672940926924131
Song03.wav	0.2967314337869684
Song04.wav	0.3980632976749803
Song05.wav	3.959875176803001
Song06.wav	7.309520323198752
Song07.wav	6.296821392318376

Podemos observar que os ficheiros Song06.wav e Song07.wav são os que têm mais parecenças com o ficheiro guitarSolo.wav, pois têm valores de informação mútua mais elevados.

CONCLUSÃO

Ao realizar este trabalho familiarizamos-mos com conceitos de Teoria da Informação bem como aprofundamos os nossos conhecimentos de numpy.

Compreendemos as limitações reais dos códigos de Huffman, como a sua incapacidade de atingir o limite mínimo teórico para o número de bits por símbolo. Também reconhecemos que ao agrupar os símbolos podemos otimizar o código ao reduzir a sua entropia.

Com este trabalho também ganhamos uma melhor compreensão de problemas reais, como a forma em que o ruído afeta os resultados de certos programas.