FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

# Entropia, Redundância e Informação Mútua

Teoria da Informação

Licenciatura em Engenharia Informática

2020/2021

Duarte Emanuel Ramos Meneses – 2019216949

Inês Martins Marçal – 2019215917

Patrícia Beatriz Silva Costa - 2019213995

# Índice de conteúdos

Introdução ..... 3
Exercício 1 ..... 4
Exercício 2 ..... 5
Exercício 3 ..... 6
Será possível comprimir cada uma das fontes de forma não destrutiva? Se Sim, qual a compressão máxima que se consegue alcançar? ..... 8
Exercício 4 ..... 9
Será possível reduzir-se a variância? Se sim, como pode ser feito e em que circunstância será útil? ..... 10
Exercício 5 ..... 10
Exercício 6 ..... 12
Exercício 6a ..... 12
Exercício 6b ..... 14
Exercício 6c ..... 15
Conclusão ..... 17
Bibliografia / Webgrafia ..... 18

# Introdução

Este trabalho prático tinha como objetivo que os alunos que o desenvolveram adquirissem sensibilidade para as questões fundamentais da cadeira de Teoria da Informação, tais como os conceitos da informação, redundância, entropia e informação mútua.

Para o realizar, aplicamos os conhecimentos adquiridos nas aulas teóricas utilizando a linguagem de programação *Python* para desenvolver os programas necessários ao cálculo e visualização de informação.

De entre os muitos conceitos abordados ao longo deste trabalho, os mais importantes são os de entropia e informação mútua. Neste sentido, passamos a explicar cada um deles.

A entropia consiste no limite mínimo teórico para o número médio de bits por símbolo. Este símbolo está presente numa fonte de informação *P* e, em simultâneo, num alfabeto *A* = $\{a_1,\ldots, a_n\}$. Quanto mais dispersos forem os símbolos do alfabeto na fonte, maior será a entropia.

Ainda no âmbito da entropia, existe a entropia agrupada. Esta consiste em, tal como a "normal", representar o limite mínimo teórico para o número médio de bits. A diferença é que em vez desta medida ser em relação a um símbolo, diz respeito a vários símbolos contíguos.

Já a informação mútua define-se como a informação que uma variável contém de outra. Tal como vamos ver mais adiante, esta medida é muito utilizada para comparar ficheiros de áudio. Um dos casos mais populares é o do programa *Shazam* que apenas com um trecho de uma música, consegue identificar de que som se trata. Isto acontece uma vez que o programa vai comparar aquele trecho de som com os já existentes na sua base de dados e devolve aquele que possui maior valor de informação mútua (maior valor de informação mútua ⇔ som mais próximo do original).

Todos estes e outros conceitos serão abordados e explicados mais aprofundadamente ao longo do trabalho, à medida que forem aparecendo em cada exercício.

## Exercício 1

Neste exercício era-nos solicitado que escrevêssemos uma função que dada uma fonte de informação P com um alfabeto *A* = {$a_1$,…, $a_n$} determinasse e apresentasse o histograma de ocorrência dos seus símbolos. Para isso desenvolvemos a função *histograma()*.

Esta função começa por chamar a função *ocorrencias()* passando-lhe como parâmetros a fonte de informação e o alfabeto. Essa função, ao receber esses parâmetros, vai criar e devolver um dicionário em que as chaves são os símbolos do alfabeto e os valores são o número de ocorrências de cada símbolo. Para gerar esse dicionário, a função vai percorrer todos os símbolos do alfabeto e incrementar 1 unidade o valor da chave correspondente (tendo estes valores sido inicializados a 0 anteriormente).

Tendo este dicionário, na função *histograma()* basta utilizar a função *bar()* do pacote *matplotlib.pyplot* para gerar o histograma. No eixo das coordenadas ficam os valores do alfabeto, enquanto no eixo das ordenadas ficam os valores correspondentes ao número de ocorrências de cada símbolo.

```
def histograma(fonte, alfabeto):
    ocorrencia=ocorrencias(fonte, alfabeto)

    plt.xlabel("Alfabeto")
    plt.ylabel("Número de Ocorrências")
    plt.bar(alfabeto, ocorrencia.values(), color='skyblue')
```

*Fig 1 - Código da função histograma()*

, onde a função *ocorrencias()* tem o seguinte código:

```
def ocorrencias(fonte, alfabeto):
    ocorrencias=dict()
    for alf in alfabeto:
        ocorrencias.update({alf:0})
    for i in fonte:
        if (i in ocorrencias.keys()):
            ocorrencias[i]+=1
    return ocorrencias
```

*Fig 2 - Código da função ocorrencias()*

## Exercício 2

Nesta parte do trabalho prático criamos a função *entropia()* que dada uma fonte de informação *P* com um alfabeto *A* = {$a_1$,…, $a_n$} determina e apresenta o limite mínimo teórico para o número médio de bits por símbolo, ou seja, a entropia.

Esta calcula-se através da fórmula:

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

com $i(a_i) = log_2 \frac{1}{P(a_i)}$, $P(a_i) = \frac{n}{N}$, n = número de ocorrências do símbolo $a_i$ e $N$ = número total de ocorrências

```
def entropia(fonte, alfabeto):
    ocorrencia = ocorrencias(fonte, alfabeto)
    prob=probabilidade(ocorrencia)
    entr=np.sum(prob*np.log2(1/prob))

    return entr
```

*Fig 3 - Código da função entropia()*

, onde a função *ocorrencias()* já é conhecida do exercício anterior e a função *probabilidade()* tem o seguinte código:

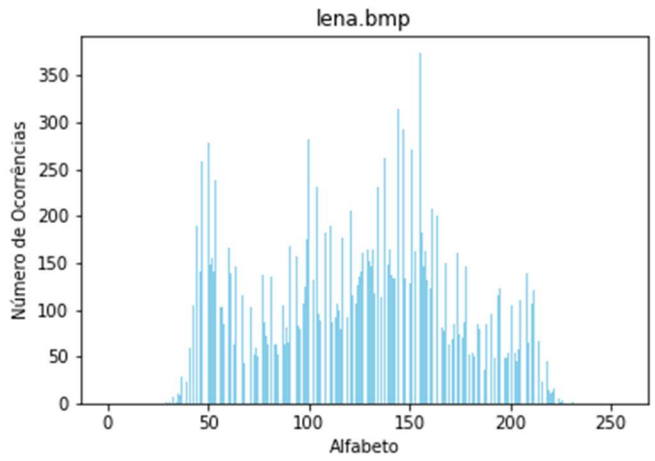
```
def probabilidade(ocorrencias):
    valores=np.array(list(ocorrencias.values()))
    valores = valores[valores>0]

    no_total=np.sum(valores)

    prob=valores/no_total
    return prob
```

*Fig 4 - Código da função probabilidade()*

Resumidamente, a função *entropia()*, após receber a fonte de informação e o alfabeto como parâmetros, vai calcular o número de ocorrências de cada símbolo (chamando a função *ocorrencias()* tal como explicado no exercício 1). Após isso, chama a função *probabilidade()* que vai calcular a probabilidade de cada símbolo ocorrer (número de ocorrências do símbolo $a_i$ a dividir pelo número total das ocorrências dos símbolos), ignorando os símbolos que não têm ocorrências. Tendo estas probabilidades, basta aplicar a fórmula da entropia já acima mencionada.

## Exercício 3

Nesta questão era-nos proposto que a partir das funções desenvolvidas nas alíneas anteriores determinássemos a distribuição estatística (histograma) e o limite mínimo para o número médio de bits por símbolo das fontes de informação *lena.bmp*, *CT1.bmp*, *binaria.bmp*, *saxriff.wav* e *texto.txt*. Os resultados foram os seguintes:




*Graf 1 - Número de ocorrências dos símbolos de lena.bmp*

Pela análise do gráfico verificamos que os valores do alfabeto variam entre 0 e 255.

Esta imagem foi a que, de todas as fontes, apresentou o valor mais alto de entropia:

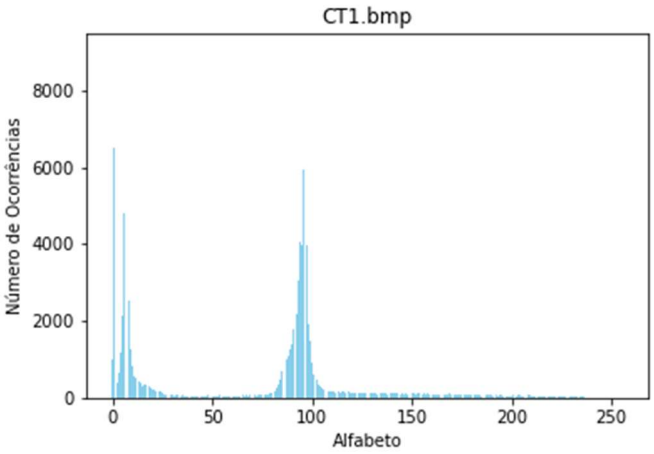
$$H(A) = 6.9153359855589525$$

Este valor já era esperado pois ao analisar a fórmula da entropia se percebia que quanto maior fosse a dispersão dos símbolos na fonte, maior seria esta medida. Como esta imagem é a que, no seu histograma, apresenta uma maior dispersão de símbolos, é natural que tenha a maior entropia.

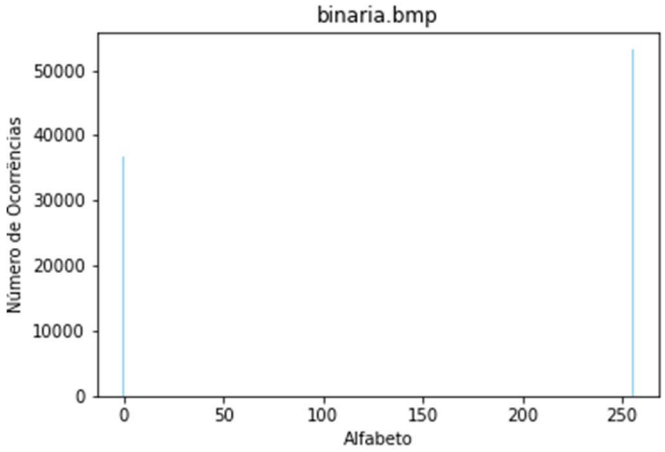
Pela análise do gráfico verificamos que os valores do alfabeto variam, tal como o anterior, entre 0 e 255.
A entropia desta imagem é de:

$$H(A) = 5.972233535340102$$

A justificação para tal é semelhante à da fonte *lena.bmp*, uma vez que a entropia é apenas um pouco menor que a dessa imagem. Isto explica-se pelo facto de existir igualmente dispersão, mas já não ocorrer na mesma quantidade (em vez de os valores estarem mais ou menos uniformemente dispersos, existem picos de concentração mais acentuados, como é o caso dos intervalos [0, 25] e [80, 105] aproximadamente).




*Graf 2 - Número de ocorrências dos símbolos de CT1.bmp*




*Graf 3 - Número de ocorrências dos símbolos de binaria.bmp*

O histograma ao lado indica o esperado. Sendo a fonte uma imagem binarizada, era expectável que o gráfico apenas apresentasse ocorrências dos valores 0 (preto) e 255 (branco).
Esta imagem foi a que, de todas as fontes, apresentou o valor mais baixo de entropia:

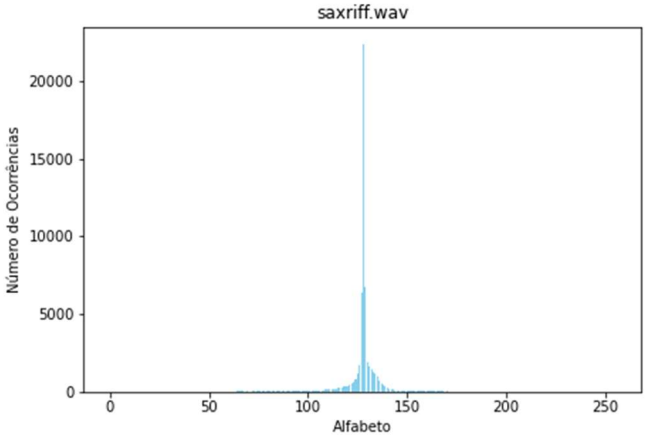
$$H(A) = 0.9755265590544998$$

Este valor também era expectável uma vez que, só existindo 2 símbolos distintos na fonte, a sua dispersão é mínima. Concluímos ainda que a imagem tem mais símbolos correspondentes a branco do que a preto (notório no histograma).

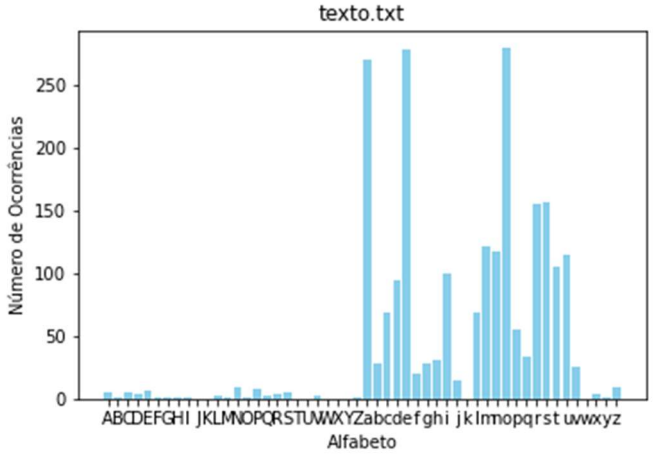
Podemos verificar no gráfico ao lado que existe uma grande concentração de valores em torno do 128, havendo inclusive um pico de concentração neste mesmo valor. Isto acontece uma vez que o som se trata de informação do tipo *uint8* e, portanto, não há valores negativos. Deste modo é o 128 (valor intermédio) que "serve de 0" (valor de repouso). Os restantes valores dispõem-se de forma aproximadamente simétrica tendo esse valor como eixo de referência.
A entropia desta fonte é:

$$H(A) = 3.530988733765133$$

Isto indica que existe ligeira dispersão de valores.




*Graf 4 - Número de ocorrências dos símbolos de saxriff.wav*




*Graf 5 - Número de ocorrências dos símbolos de texto.txt*

Pela visualização do gráfico, facilmente percebemos que, tal como era de esperar de uma fonte de texto deste tipo, existem mais letras minúsculas que maiúsculas.
A entropia é:

$$H(A) = 4.196888541803248$$

Isto indica que há uma dispersão assinalável de valores. É muito mais provável ocorrer uma letra minúscula do que uma maiúscula, sobretudo se for o *a*, *e* ou *o*.

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

A compressão de dados é uma estratégia utilizada para armazenar informação em menos espaço. Para a compressão ser bem efetuada temos de garantir que não ocorre perda de informação. Deste modo, ao descomprimir, as informações iniciais têm de se manter inalteradas.

A taxa de compressão máxima é dada por:

$$TaxaCompressão_{max} = \frac{Entropia_{max} - Entropia}{Entropia_{max}} * 100$$

A entropia máxima é dada também pela fórmula:

$$H(A)_{max} = log2\ (\mathrm{M})$$ , sendo M o número de elementos do alfabeto.

No caso de as fontes de informação serem imagens e sons, o valor do M é 256, uma vez que o seu alfabeto vai de 0 a 255. Deste modo, a entropia máxima é de:

$$log_2(256) = 8$$

Já no caso da fonte de informação se tratar de um texto, estando nós a considerar apenas o alfabeto sem caracteres especiais (que tem 26 letras minúsculas e 26 letras maiúsculas), a entropia máxima nestes casos será de:

$$log_2(52) \approx 5.70$$

Deste modo:

| Fonte | Entropia obtida | Entropia máxima | Taxa de compressão máxima (%) |
|---|---|---|---|
| lena.bmp | 6.915336 | 8.0 | 13.56 |
| CT1.bmp | 5.972234 | 8.0 | 25.35 |
| binaria.bmp | 0.975527 | 8.0 | 87.81 |
| saxriff.wav | 3.530989 | 8.0 | 55.86 |
| texto.txt | 4.196889 | 5.7 | 26.37 |

Pela análise do quadro anterior, verificamos que é expectável conseguir-se comprimir cada uma das fontes de forma não destrutiva. A taxa de compressão máxima varia de fonte para fonte (resultados no quadro acima), estando os valores dessa taxa compreendidos num intervalo entre 13.56% e 87.81%.

## Exercício 4

Neste exercício era-nos pedido que calculássemos o número médio de bits por símbolo para cada uma das fontes de informação disponibilizadas. Para isso, foi-nos fornecido o código *huffmancodec.py*, o qual utilizamos. Através da fórmula da média ponderada (adaptada a este contexto), calculamos o pretendido.

$$Média\ ponderada = \frac{\sum_{i=1}^{n} P(a_i)*l_i}{\sum_{i=1}^{n} P(a_i)}$$ . Como $\sum_{i=1}^{n} P(a_i) = 1$:

$$Média\ ponderada = \sum_{i=1}^{n} P(a_i) * l_i$$

,com $l_i$ número de bits necessário para codificar o símbolo $a_i$

Ainda nesta pergunta, foi-nos pedido que comentássemos os valores da variância dos comprimentos dos códigos resultantes. Os valores da média ponderada e da variância encontram-se no quadro abaixo. Estes últimos foram calculados através da fórmula:

$$Variância = \sum_{i=1}^{n} P(a_i) * (l_i - nº\ médio\ de\ bits)^2$$

| Fonte | Entropia | N.º médio de bits por símbolo | Variância dos comprimentos |
|---|---|---|---|
| lena.bmp | 6.915336 | 6.942505 | 0.639395 |
| CT1.bmp | 5.972234 | 6.007546 | 5.201652 |
| binaria.bmp | 0.975527 | 1.0 | 0.0 |
| saxriff.wav | 3.530989 | 3.584290 | 7.718197 |
| texto.txt | 4.196889 | 4.217295 | 1.882716 |

Comparando os resultados do número médio de bits por símbolo com a entropia anteriormente calculada, verificamos que são muito semelhantes, ainda que os primeiros sejam ligeiramente mais elevados. Com isto constatamos que se verifica esta fórmula:

$$H(S) \leq \bar{l} < H(S) + 1\ ,$$

sendo H(S) a entropia calculada e $\bar{l}$ o número médio de bits por símbolo

A variância indica o quão díspares são os valores do número médio de bits necessário para codificar cada símbolo e o número de bits necessários para codificar os

símbolos da fonte. Com isto, percebemos que a fonte *binaria.bmp* está já codificada da melhor forma e a *lena.bmp* está quase ótima. Já o som *saxriff.wav* e a imagem *CT1.bmp*, tendo uma variância elevada, não se encontram codificados da forma mais otimizada.

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

O algoritmo do Código de *Huffman*, tal como foi lecionado nas aulas teóricas, ocorre em 2 etapas:

1- Ordena os símbolos por ordem crescente;
2- Constrói uma árvore binária (combinando os 2 símbolos menos frequentes num único símbolo e acrescentando esse símbolo resultante à lista em que a frequência de ocorrência é a soma das frequências individuais).

A variância é mínima quando se constroem as árvores binárias a partir das probabilidades de ocorrência dos símbolos ordenadas de forma decrescente. Deste modo, como o nosso código não ordena as probabilidades desta forma, mas sim pela ordem dos símbolos que definimos no alfabeto, a variância não é mínima. Com isto, fica claro que basta ordenar as probabilidades da maior para a menor para reduzir a variância.

Esta redução é útil quando se pretende contornar congestionamentos de rede. Para o fazer, utilizam-se *buffers* e para isso convém que a taxa de enchimento seja relativamente constante. Deste modo, o número médio de bits para codificar cada símbolo deve ser de igual forma aproximadamente constante (variância baixa).

## Exercício 5

Neste exercício foi-nos proposto determinar a entropia agrupada das fontes de informação disponibilizadas, isto é, calcular a entropia, tal como no exercício 3, admitindo agora que cada símbolo é na verdade uma sequência de dois símbolos contíguos.

Deste modo, o alfabeto de cada fonte passou a ser o conjunto de todas as combinações possíveis de 2 elementos que constam no alfabeto utilizado no exercício 3. No caso da imagem e som, o alfabeto é calculado a partir do código seguinte:

```
def alfa_som_img_par():
    alfabeto = list()
    for i in range (256):
            for j in range(256):
                alfabeto.append(str(i)+"/"+str(j))
    return alfabeto
```

*Fig 5- Código da função que cria o alfabeto dos ficheiros de som e imagem para a entropia agrupada*

Já no caso de a fonte ser um texto, como o alfabeto original só tem 52 elementos, o alfabeto para a entropia agrupada é calculado com o seguinte código (sendo *alfabeto_ant* o alfabeto original utilizado no exercício 3:

```
def alfa_txt_par(alfabeto_ant):
    novo=list()
    for i in range (len(alfabeto_ant)):
            for j in range(len(alfabeto_ant)):
                novo.append(str(alfabeto_ant[i])+"/"+str(alfabeto_ant[j]))
    return novo
```

*Fig 6 - Código da função que cria o alfabeto dos ficheiros de texto para a entropia agrupada*

Os resultados da entropia agrupada de cada fonte estão presentes no quadro abaixo:

| Fonte | Entropia | Entropia agrupada |
|---|---|---|
| lena.bmp | 6.915336 | 5.596516 |
| CT1.bmp | 5.972234 | 4.481268 |
| binaria.bmp | 0.975527 | 0.542405 |
| saxriff.wav | 3.530989 | 2.889866 |
| texto.txt | 4.196889 | 3.754269 |

Era de esperar que a entropia agora calculada fosse menor que a determinada no exercício 3, uma vez que agrupamos dois símbolos contíguos. Geralmente, a estimação da entropia melhora com o agrupamento de símbolos.

No entanto, não se pode dizer que este é o método mais eficaz para reduzir o número de bits uma vez que o alfabeto aumenta exponencialmente, o que torna o código mais complexo e leva a que demore mais tempo a ser executado.

# Exercício 6

## Exercício 6a

Neste exercício foi-nos proposto que criássemos uma função que, dada a query, o target, um alfabeto *A* = $\{a_1,\ldots, a_n\}$ e o passo, devolvesse o vetor de valores de informação mútua em cada janela.

Para isso, tal como exemplificado no enunciado, teríamos de ir percorrendo passo a passo a query ao longo do target criando janelas para as comparar com a query. Quanto mais semelhante fosse a janela da query, mais próximo esses excertos de som seriam e, por conseguinte, maior seria o valor da informação mútua.

Deste modo, tal como o nosso código na função *infoMutua()* indica, cada janela terá exatamente o mesmo número de valores que a query. Já o número de janelas (número de deslocamentos da query sob o target) é dado pela aproximação às unidades de:

$$Número\ de\ janelas\ = \frac{Tamanho_{target}\ - Tamanho_{query}}{Passo} + 1$$

A informação mútua de cada janela é dada pela expressão:

$$Informação\ mútua\ = Entropia_{query}\ +\ Entropia_{janela}\ -\ Entropia_{query,janela}$$

Chegamos a esta conclusão a partir da demonstração seguinte:

Da matéria lecionada nas aulas teóricas sabe-se que:

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

Ou seja:

$$H(Y|X) = H(X,Y) - H(X)$$

e

$$H(X|Y) = H(X,Y) - H(Y)$$

Deste modo, e analisando o diagrama seguinte fica claro que:

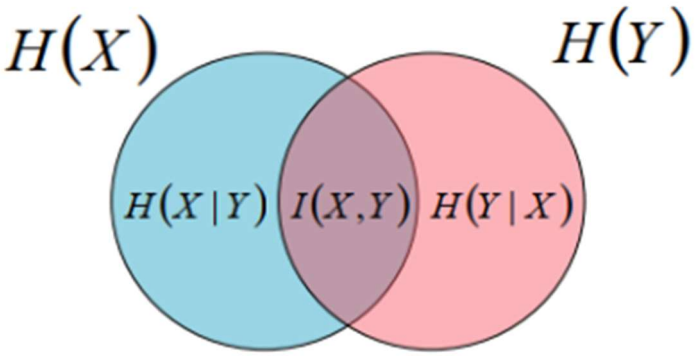



*Fig 7 - Diagrama de Venn que representa a Informação Mútua*

$$2 * I(X,Y) = H(X) + H(Y) - H(X|Y) - H(Y|X) \Leftrightarrow$$
$$\Leftrightarrow 2 * I(X,Y) = H(X) + H(Y) - H(X,Y) + H(Y) - H(X,Y) + H(X) \Leftrightarrow$$
$$\Leftrightarrow 2 * I(X,Y) = 2 * (H(X) + H(Y) - H(X,Y)) \Leftrightarrow$$
$$\Leftrightarrow I(X,Y) = H(X) + H(Y) - H(X,Y)$$

Para calcular as entropias usamos a função *entropia()* já referida anteriormente.

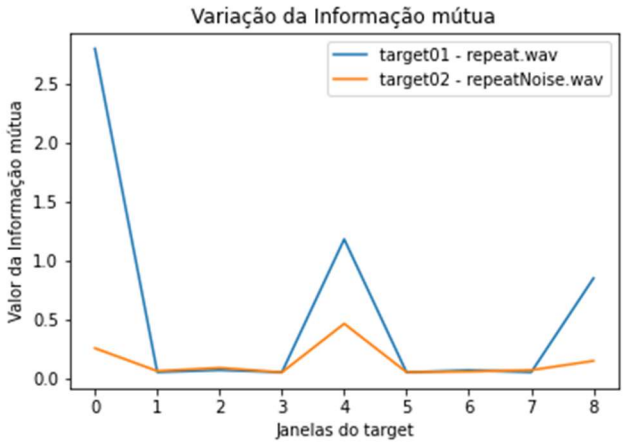
```
def infoMutua(query, target, alfabeto, passo):

    janela = np.zeros(len(query), dtype='int')

    entr_query = entropia(query, alfabeto)

    Lista_infomutua = np.zeros(int(((len(target)-len(query))/passo)+1))

    alfab = list()

    for alfa in alfabeto:
        for alfa2 in alfabeto:
            alfab.append(str(alfa)+"/"+str(alfa2))

    for i in range(0, int(((len(target)-len(query))/passo)+1)):
        listaMutua=list()
        for j in range (len(query)):
            janela[j] = target[i*passo+j]
            listaMutua.append(str(janela[j])+"/"+str(query[j]))

        entr_conj = entropia(listaMutua,alfab)

        entr_janela = entropia(janela, alfabeto)
        Lista_infomutua[i] = entr_query + entr_janela - entr_conj

    return Lista_infomutua
```

*Fig 8 - Código da função infoMutua()*

## Exercício 6b

Nesta questão foi-nos pedido que utilizássemos a função criada na alínea anterior para calcular a informação mútua utilizando dois *target's* diferentes (*target01 – repeat.wav* e *target02 – repeatNoise.wav*) com a mesma *query* (*saxriff.wav*) sendo o valor do passo igual a ¼ do comprimento da *query*.

Os resultados da evolução da informação mútua ao longo do tempo foram os seguintes:




*Graf 6 - Variação da Informação Mútua de target01 - repeat.wav e target02 - repeatNoise.wav*

| Fonte | Vetor de informação mútua | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| target01 - repeat.wav | 2.802683 | 0.061817 | 0.079310 | 0.061220 | 1.188246 | 0.060780 | 0.078063 | 0.061069 | 0.857290 |
| target02 - repeatNoise.wav | 0.264677 | 0.071612 | 0.098595 | 0.060610 | 0.472681 | 0.062073 | 0.067259 | 0.078971 | 0.157641 |

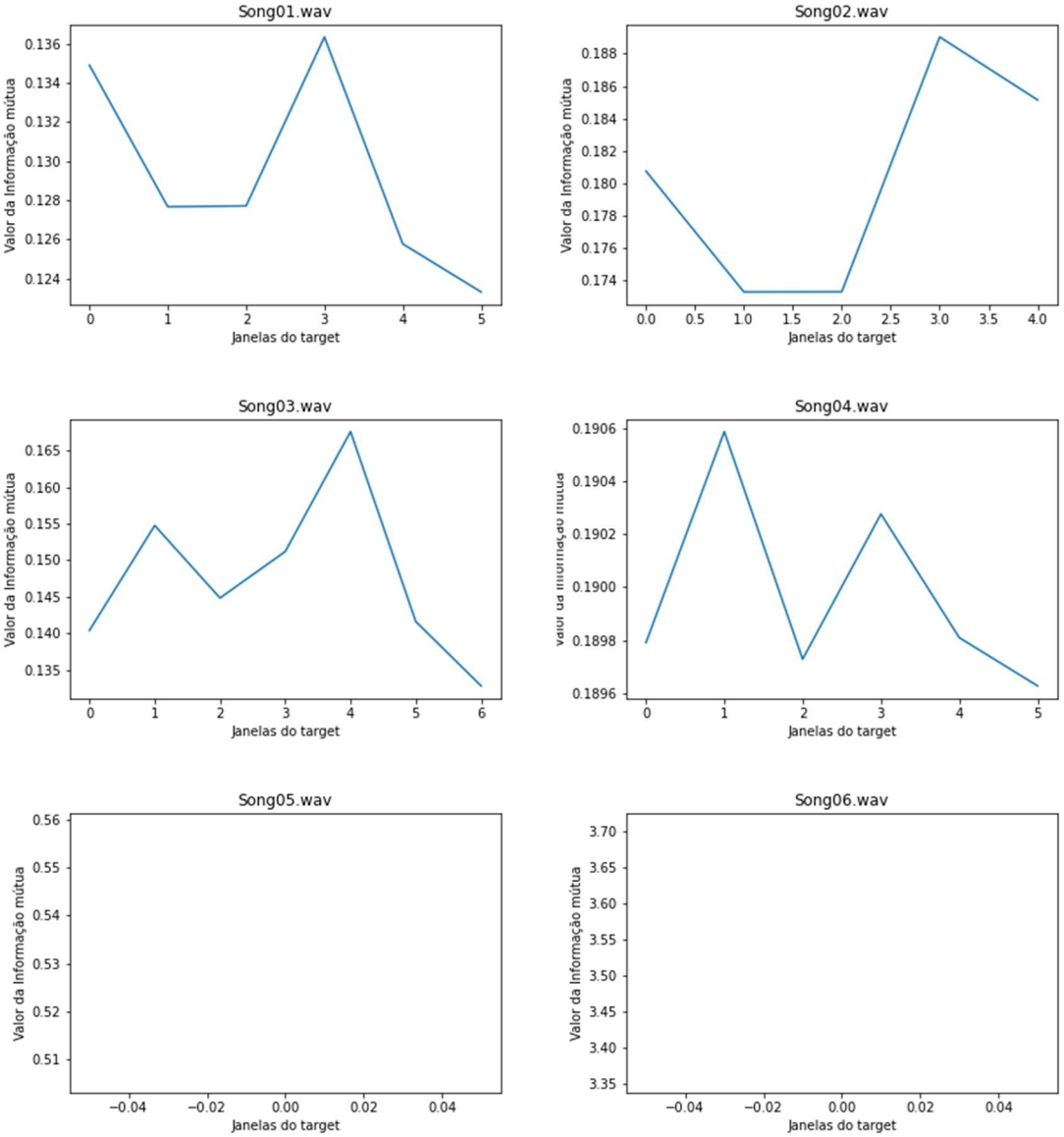
Com estes resultados, fica claro que o *target01 – repeat.wav* tem muito mais semelhanças com o ficheiro *saxriff.wav* do que o *target02 – repeatNoise.wav.* Constatamos isto devido aos maiores valores presentes no vetor de informação mútua de *target01 – repeat.wav.* Mais facilmente se verifica tal facto visualizando o gráfico da sua variação ao longo do tempo já que se compara graficamente os valores de ambos os ficheiros. Aí fica notório que os valores de informação mútua entre *target01 – repeat.wav* e a *query* são mais elevados (sobretudo nas janelas 0, 4 e 8).

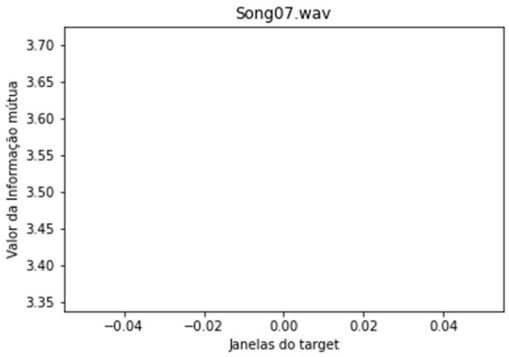
Ainda antes de calcular os valores do vetor de informação mútua de cada ficheiro de som, estes resultados já eram expectáveis uma vez que, ao ouvir as fontes de informação, se notava claramente que o ficheiro *target01 – repeat.wav* era mais parecido com *saxriff.wav* do que o ficheiro *target02 – repeatNoise.wav* (que apresentava mais ruído).

## Exercício 6c

Nesta alínea era pedido que utilizando a mesma *query* da alínea anterior (*saxriff.wav*) analisássemos a sua informação mútua com os ficheiros *Song01.wav*, *Song02.wav, Song03.wav, Song04.wav, Song05.wav, Song06.wav, Song07.wav* respetivamente. Deste modo, era solicitado que criássemos um mini simulador de identificação de música, sendo o passo igual a ¼ da duração da *query*.

A evolução da informação mútua de cada ficheiro está presente nos gráficos seguintes.






*Grafs 7 a 13 - Resultados da variação da Informação mútua ao longo do tempo*

Pela visualização dos gráficos percebemos que ao longo do tempo existe variação da informação mútua nos 4 primeiros ficheiros de som analisados. Tal não acontece nos seguintes, uma vez que a sua duração é bem mais inferior (semelhante à da *query*), o que leva a que não haja mais do que 1 janela (não há deslocamentos da *query* sob o *target*). Deste modo, nesses gráficos não se vislumbram oscilações de valores, uma vez que o vetor de informação mútua desses ficheiros apenas contem 1 valor.

A informação mútua máxima de cada ficheiro (já ordenados de forma decrescente) é:

| Fonte | Informação mútua máxima |
|---|---|
| Song07.wav | 3.530989 |
| Song06.wav | 3.530989 |
| Song05.wav | 0.532242 |
| Song04.wav | 0.190587 |
| Song02.wav | 0.189038 |
| Song03.wav | 0.167568 |
| Song01.wav | 0.136342 |

Os sons estão ordenados sendo o que está no topo da tabela o mais parecido com a query em questão e o que está em baixo o menos semelhante. Analisando os valores do quadro, percebe-se que os ficheiros de áudio *Song07.wav* e *Song06.wav* são os mais semelhantes com a *query*. Tal como na alínea anterior, estes resultados eram esperados, já que, ouvindo todos os ficheiros WAV, estes eram os mais parecidos sonoramente.

## Conclusão

Este trabalho prático tinha como objetivos o desenvolvimento de capacidades por parte de quem o realizava, tais como a sensibilização para as questões fundamentais da cadeira de Teoria da Informação e a aprendizagem de alguns conceitos fundamentais.

Ao longo da realização do trabalho, e com as observações do docente da turma teórico-prática, fomo-nos deparando que o nosso código nem sempre estava o mais otimizado possível. Além de demorar muito tempo a ser executado, a própria organização não era a melhor. Com isto, tentamos, por exemplo, utilizar as ferramentas que o pacote *numpy* nos oferece para reduzir o número de ciclos *for* que tínhamos implementado. Após as otimizações, o código ficou não só mais fácil de interpretar como também mais rápido na execução.

Tal como referimos na introdução, este primeiro trabalho prático baseava-se principalmente nos conceitos de entropia e informação mútua, mas também no de redundância e informação. Na nossa opinião, conseguimos implementar eficazmente as suas definições, abordadas nas aulas teóricas, nos exercícios práticos propostos.

Em suma, este foi um trabalho que nos permitiu ficar mais cientes do contexto prático e da utilidade dos conceitos abordados na componente teórica da cadeira. Ajudou-nos ainda a compreender algumas definições que, apenas teoricamente, não eram tão percetíveis como agora.

## Bibliografia / Webgrafia


- Materiais da cadeira disponibilizados pelos docentes
- https://pt.wikipedia.org/wiki/Compress%C3%A3o_de_dados [22/10/2020]
- http://users.isr.ist.utl.pt/~vab/FTELE/cap1.pdf [27/10/2020]
- http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-estatistica/algoritmo-de-huffman/ [05/11/2020]
- https://pt.wikipedia.org/wiki/Teoria_da_informa%C3%A7%C3%A3o [06/11/2020]