Relatório

Entropia, redundância e Informação mútua

Realizado por:

Rodrigo Figueiredo - 2020236687

Rui Santos - 2020225542

Bruno Sequeira - 2020235721

PL4

Introdução

▪ Primeiro trabalho no âmbito da cadeira de Teoria da Informação, realizado em **python**, realizado nas aulas práticas da mesma.

▪ Este trabalho tem como objetivo aplicar temas como a entropia, a redundância e a informação mútua, temas estes abordados nas aulas teórico-práticas da disciplina.

Exercício I

O primeiro exercício tinha como objetivo criar um **histograma** a partir de uma fonte e de um alfabeto, onde o eixo dos **y** irá ter as ocorrências de cada um dos símbolos da fonte pertencentes ao alfabeto e o eixo dos **x** cada um dos símbolos.

NOÇÕES:

Um **alfabeto** é um conjunto de símbolos que formam uma linguagem. Neste caso o alfabeto é o conjunto de símbolos usado para codificar a nossa **fonte de dados** de cada ficheiro.

A **fonte de dados** são os dados retirados de cada ficheiro que pertencem ao seu devido alfabeto.

Tipos de ficheiros:

Na realização deste trabalho formam utilizados três tipos de ficheiros:

- “txt” - corresponde aos ficheiros de texto e tem uma escala de valores de 65 a 122, onde de 65 a 90 encontram-se as letras minúsculas e de 97 a 122 as maiúsculas (na tabela **ASCII**) .

- “bmp” - corresponde aos ficheiros de imagem e utiliza-se a escala **0 a 255** uma vez que essa é a gama de valores onde o espectro de cores está definido em RGB.
- “wav” - corresponde aos ficheiros de som, e tal como nos ficheiros “bmp” tem uma escala de valores de **0 e 255**

Funções

- **hist():** Recebe um dicionário e a partir das suas chaves e valores cria um histograma, onde as chaves equivalem aos símbolos e os valores as suas respetivas ocorrências.
- **makeHist():** identifica que tipo de ficheiro estamos a ler e usa a função hist() para fazer o seu histograma, esta função recebe apenas o nome do ficheiro.
- **readText():** lê o ficheiro de texto e devolve todos os caracteres pertencentes ao alfabeto, essa será a fonte/data do ficheiro de texto.
- **ocorrencias():** recebe um alfabeto e uma fonte e devolve um dicionário com as ocorrências de cada símbolo, sendo estes as chaves e as suas respetivas ocorrências, os valores.

Para resolução deste exercício foram criadas 4 funções, uma para tratar de gerar o histograma, uma para verificar que tipo de ficheiro estamos a ler(uma vez que vamos trabalhar com 3 tipos de ficheiros), outra que guarda os caracteres, pertencentes ao alfabeto, presentes no texto, numa lista e outra que conta as ocorrências de um símbolo.

Precisamos também de gerar os alfabetos que cada tipo de ficheiro vai utilizar, no caso dos ficheiros de texto é o alfabeto de maiúsculas e minúsculas, e no caso dos ficheiros “wav” e “bmp” são os números de 0 a 255 (em wav um símbolo é codificado em 8 bits e 2**8 = 256, em bmp cada valor também é guardado em 8 bits).

Depois de abrir o ficheiro e guardar os dados de cada ficheiro numa variável, registamos as ocorrências de cada símbolo do alfabeto na data e geramos o gráfico de barras correspondente.

Exercício II

Neste exercício é pedido para calcular o limite mínimo teórico para o número médio de bits por símbolo, este calculo pode ser feito a partir da **fórmula de entropia.**

Fórmula da entropia:

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

Para este exercício foi criada a função **entropia()** que recebe a "data*" do ficheiro como parâmetro. Esta função conta as ocorrências dos símbolos da "data", dividimos esse número de ocorrências pelo comprimento da "data" para obter as probabilidades e aplicamos a fórmula da entropia para obter o resultado

*data – dados de um ficheiro / fonte / conjunto de símbolos presentes num ficheiro

Exercício III

Neste exercício pedem-nos para usar as funções criadas nos exercícios 1 e 2 e aplicá-las a 5 ficheiros, 3 do tipo "bmp", 1 do tipo "wav" e outro do tipo "txt".

english.txt

Entropia: 4.22796792353279

Pela análise do gráfico podemos concluir que existem mais letras minúsculas que maiúsculas no texto, e que, dada a diversidade de valores a entropia é alta.

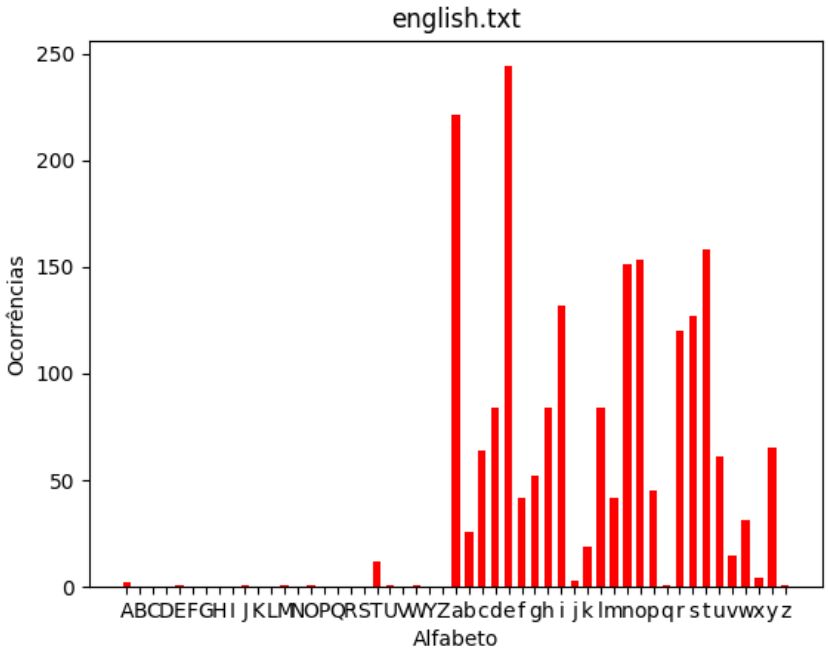


Figura 1.1 – Gráfico de ocorrências de cada letra na fonte do ficheiro english.txt

kid.bmp

Figura 1.2.1 – imagem do ficheiro kid.bmp

Entropia:
6.954143307171645

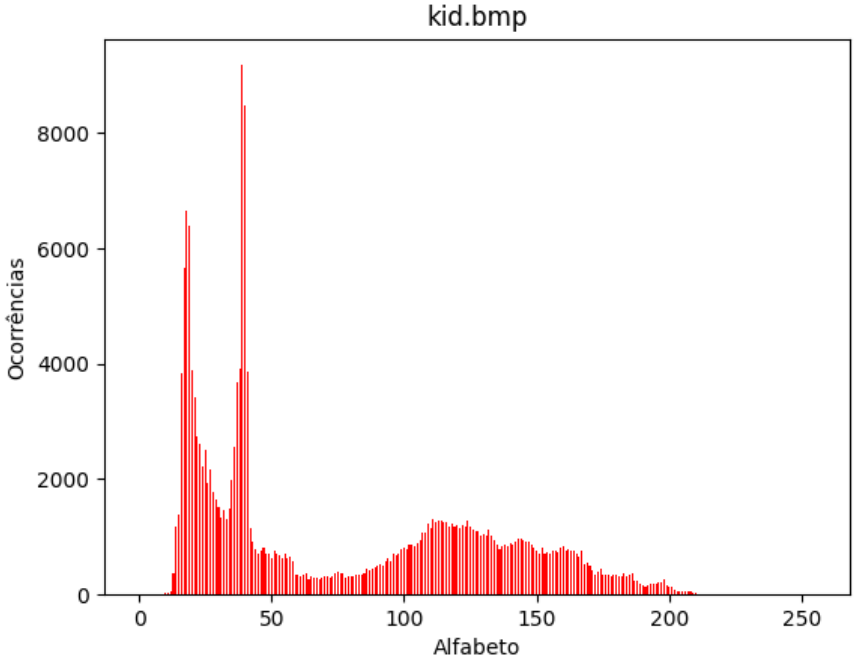
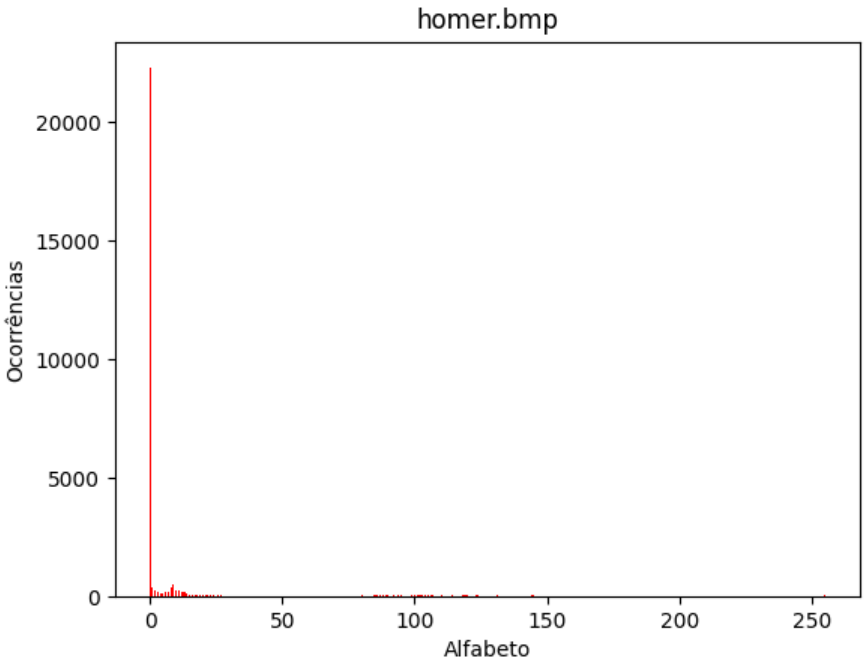


Figura 1.2.2 – gráfico de ocorrências das cores da figura 1.2.1

Pela análise do gráfico observamos uma grande diversidade de valores, que observando a figura podemos assumir que são várias tonalidades da cor cinza e daí a entropia ser alta.

Figura 1.3.2 – gráfico de ocorrências das cores da figura 1.3.1



Homer.bmp

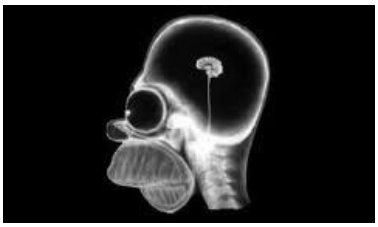


Figura 1.3.1 – imagem do ficheiro Homer.bmp

Entropia: 3.465865162708984

Pela análise do gráfico, observamos uma grande quantidade de ocorrências no valor 0 o que indica uma grande maioria da cor preta na imagem e então a entropia terá um valor consideravelmente menor aos dos ficheiros anteriores de imagem.

homerBin.bmp

Figura 1.4.1 - – imagem do ficheiro homerBin.bmp

Entropia:0.6447813982002277

Pela observação do gráfico, percebemos que só existem 2 valores, o que implica que a entropia será muito inferior , comparado com outras fontes. Pelo facto do ficheiro ser binário, implica que o 0 corresponde ao preto e o 255 corresponde ao branco.

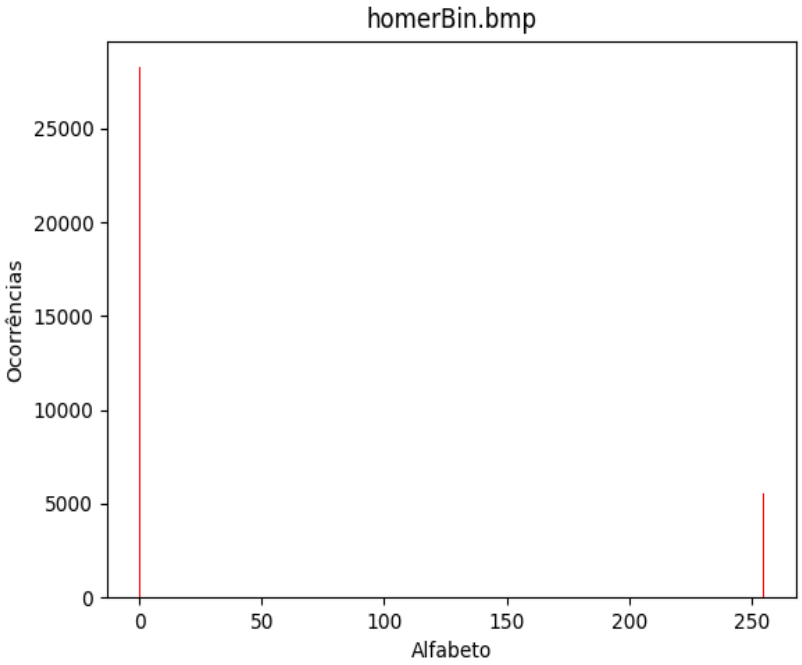


Figura 1.4.2 - gráfico de ocorrências das cores da figura 2.4.1

guitarSolo.wav

Entropia: 7.329201669454549

Pela análise do gráfico, observamos uma enorme diversidade de dados, o que justifica o valor alto da entropia dos dados deste ficheiro. Observa-se também uma grande centralização dos dados neste gráfico.

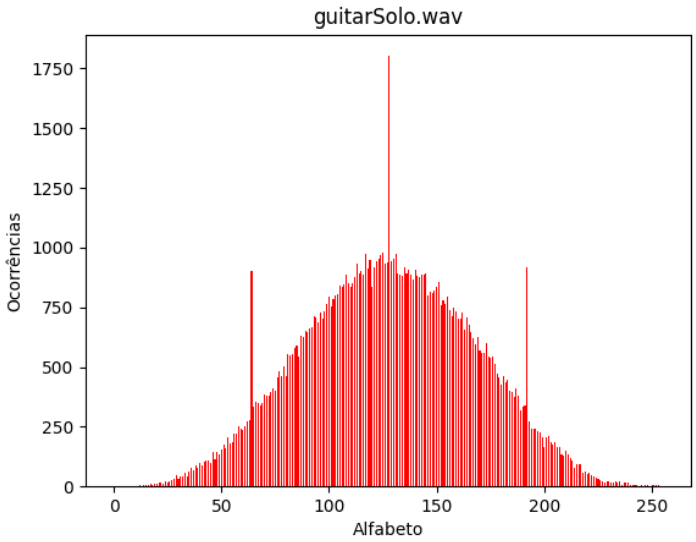


Figura 1.5.1 - gráfico de ocorrências dos dados do ficheiro guitarSolo

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

Sim, é possível comprimir cada uma das fontes de forma não destrutiva.

Para primeiro calcular o valor de compressão não destrutiva de cada um dos ficheiros (em percentagem) é necessário realizar o seguinte cálculo:

- **(Entropia máxima – Entropia) / Entropia máxima*100**

  (a entropia máxima corresponde ao número de bits necessário para codificar cada símbolo)

Nome do ficheiro	**valor de compressão não distrutiva maxima** *(Entropia máxima – Entropia) / Entropia máxima x 100*
english.txt	(8-4.22796792353279) / 8 * 100 = 47,15%
kid.bmp	(8- 6.954143307171645) / 8 * 100 = 13,07%
homer.bmp	(8-3,465865162708984) / 8 * 100 = 56,68%
homerBin.bmp	(8- 0.6447813982002277)/ 8 * 100 = 91,94%
guitarSolo.wav	(8- 7.329201669454549) / 8 * 100 = 8,38 %

Pela análise da tabela e dos respetivos cálculos podemos concluir e justificar que é possível a compressão de cada uma das fontes de forma não destrutiva pois os seus valores de compressão não estão no seu máximo.

Exercício IV

- **Codificação de Huffman:** É um método de compressão que utiliza as probabilidades das ocorrências de cada símbolo para determinar códigos de tamanho variável para cada símbolo.

Funções

numMedBits(): Recebe os dados de um ficheiro (data) e calcula o número médio de bits utilizados para codificar o ficheiro. Para isso conta

variancia(): Recebe também a "*data*" de um ficheiro e com a função numMedBits(), após se calcular a média, aplica-se a fórmula da variância (E(X**2) – E(X)**2)

Ficheiros

english.txt

- Variância: 1.1908519457764974
- Número médio de bits: 4.251830161054173

Média de bits: tem uma média de bits baixa devido à baixa variedade de caracteres

Variância: apresenta uma variância baixa devido à distribuição de ocorrências por símbolo não muito inconstante.

Kid.bmp

- Variância: 2.099295925824727
- Número médio de bits: 6.983223014256619

Média de bits: tem uma média de bits alta devido à alta variedade de tons entre branco e preto, o que aumenta o tamanho dos dados da fonte logo o número médio de bits será maior também

Variância: apresenta uma variância baixa pois a probabilidade de ocorrência, da maioria dos símbolos, não varia muito.

homer.bmp

- Variância: 13.19683817935466
- Número médio de bits: 3.5483156028368796

Média de bits: tem uma média de bits bastante baixa pois não há muita variedade de símbolos

Variância: apresenta uma variância bastante alta pois a probabilidade de ocorrência de cada símbolo é muito diferente

homerBin.bmp

- Variância: 0.0
- Número médio de bits: 1.0

Média de bits: tem uma média de bits igual a 1 pois apenas existem 2 cores , preto e branco logo é preciso apenas 1 bit para representar cada uma delas.

Variância: dado o número médio de bits ser igual a 1 é esperado que a variância seja 0.

~

guitarSolo.wav

- Variância: 0.7274273982765322
- Número médio de bits: 7.350157949456174

Média de bits: tem um número médio de bits bastante alto pois tem uma grande variedade de símbolos, logo, necessita de mais bits para os codificar

Variância: apresenta uma variância extremamente baixa pois a discrepância entre as probabilidades de ocorrência de cada símbolo e o valor esperado não varia muito.

- Não deixámos de notar que o número médio de bits de cada um dos ficheiros é próximo ao valor da sua entropia.

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

Sim, é possível reduzir a variância, basta apenas **aumentar o número de ocorrências dos símbolos**.

É bastante útil obter uma variância baixa em situações que requerem uma transmissão de uma grande quantidade de bits **constantes**, como por exemplo, streaming.

Exercício V

No quinto exercício, é pedido para calcular a estropia recorrendo ao agrupamento dos símbolos da fonte.

Para resolver este exercício fizemos uma função que vai calcular o valor da estropia do agrupamento de 2 símbolos.

Funções

- **entropia2S():** Esta função calcula a entropia de dois símbolos seguidos, logo recebe "data"/dados da fonte, agrupando assim os símbolos destes, colocando-os como chaves (tuplos de 2 símbolos seguidos ex: "(a,b)") de dicionários quando estes ocorrem. Se ocorrerem pela primeira vez, inicializamos a 1, mas se a chave já pertencer ao dicionário este acrescenta-lhe 1 ao valor que tinha.
  Pelo facto de serem dois símbolos, o comprimento do ficheiro terá de ser metade do original logo no final dividimos a entropia por 2.

Análise dos ficheiros

english.txt:

Entropia de 2 símbolos: 3.6507191991851626

O valor é mais baixo em relação à entropia para 1 símbolo, porque existe uma redundância entre letras (vogais seguidas a consoantes).

kid.bmp:

Entropia de 2 símbolos: 4.909097962176646

O valor é mais baixo em relação à entropia para 1 símbolo, pois existe uma redundância entre símbolos.

homer.bmp:

Entropia de 2 símbolos: 2.412733070535293

O valor é mais baixo em relação à entropia de 1 símbolo, pois existe uma redundância entre símbolos.

homerBin.bmp:

Entropia de 2 símbolos: 0.39782409226242005

O valor é mais baixo em relação à entropia de 1 símbolo, porque existe uma redundância entre pixéis.

guitarSolo.wav:

Entropia de 2 símbolos: 5.754384158334452

O valor é mais baixo em relação à entropia de 1 símbolo, porque existe uma redundância entre símbolos.

Observações finais:

Comparando os valores da entropia de 1 símbolo com a entropia de 2 símbolo, observamos que é sempre inferior.

A entropia de 2 símbolos mais alta e mais baixa, tal como na entropia normal, é a do guitarSolo.wav e homerBin.bmp, respetivamente, tal como já esperávamos.

Exercício VI

Noções

- **query –** Sinal a pesquisar
- **Target –** Sinal onde pesquisar
- **Passo –** Intervalo entre janelas consecutivas
- **Informação mútua –** A informação mútua de duas variáveis aleatórias é a medida de dependência mútua entre as duas variáveis. Mais especificamente, a informação mútua quantifica a informação (em bits) que uma variável aleatória contém acerca de outra.

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} w(x,y) p(x,y) \log \frac{p(x,y)}{p(x)\,p(y)},$$

Funções

getData(): recebe um ficheiro e retorna uma lista com conjunto de símbolos que compõe o ficheiro

linearGraph(): desenha os gráficos. Recebe uma lista com a informação mútua entre dois ficheiros e o título que vai corresponder ao nome do ficheiro que estamos a comparar.

infoMutua(): recebe a query, o target, o passo e o alfabeto e retorna uma lista com informação mútua. Para calcular a informação mútua precisamos da probabilidade conjunta para podermos aplicar a fórmula da informação mútua.

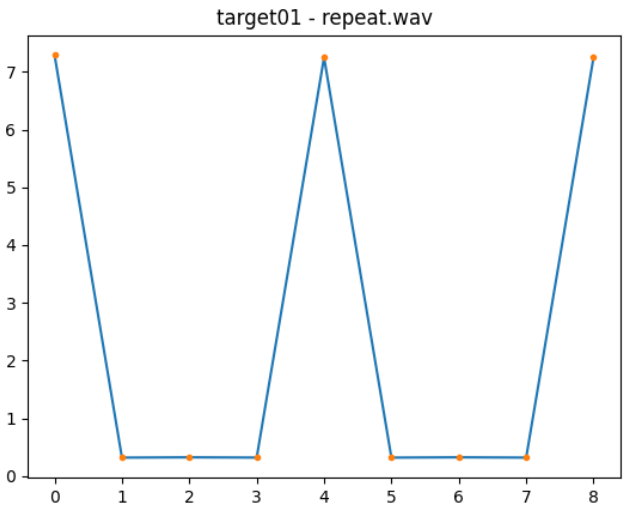
simulador(): recebe uma lista de sons e uma query e calcula a informação mútua (com uso da função infoMutua) entre a query e todos os sons da lista em que, para cada um dos sons, guarda o valor mais alto entre todos os passos numa lista. Essa lista vai representar a evolução da informação mútua.

a)

Função **infoMutua()** previamente explicada

b)

Nesta alínea temos de usar o ficheiro **"guitarSolo.wav"** e comparar com os ficheiros **"target01 – repeat.wav"** e **"target02 – repeat.wav"** e determinar a variação da informação mútua.



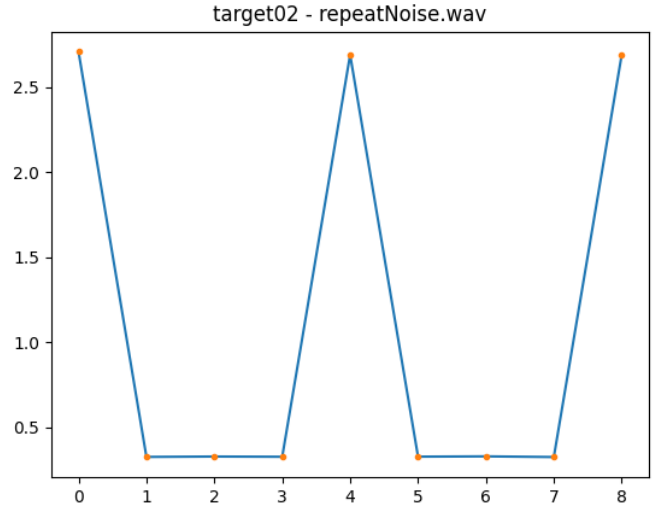


Figura 2.1 e 2.2 – informação mútua entre os ficheiros guitarSolo.wav e target01 – repeat.wav e guitarSolo.wav e target02 – repeat.wav, respetivamente.

"target02 - repeatNoise.wav" é um som semelhante ao som do ficheiro **"guitarSolo.wav"**, mas com muito mais ruído do que **"target01 - repeat.wav"**, e por esse motivo o valor de informação mútua é mais baixo.

"target01 - repeat.wav" é um som semelhante ao som do ficheiro **"guitarSolo.wav"**, mas em loop com algum ruído, por isso temos uma variação mútua elevada, mas nunca máximo pelo facto de possuir algum ruído, sendo o máximo da informação mútua quando o ruído é menos audível

c)

Nesta alínea é pedido para criarmos um simulador para identificar uma música comparando o ficheiro “guitarSolo.wav” a outros 7 ficheiros de som. Devemos também determinar a evolução da informação mútua para cada um dos ficheiros.

Para resolver o exercício usamos todas as funções que falamos no início do exercício.

Vamos dispor cada gráfico por ordem decrescente de informação mútua.

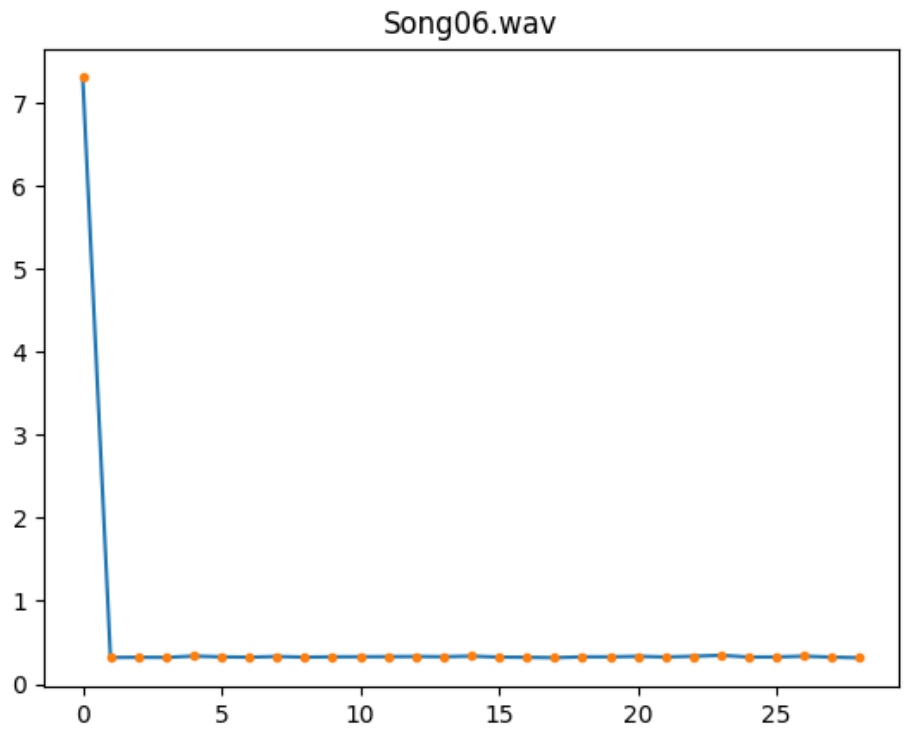


Figura 3.1 – Informação mútua entre os ficheiros song06.wav e guitarSolo.wav

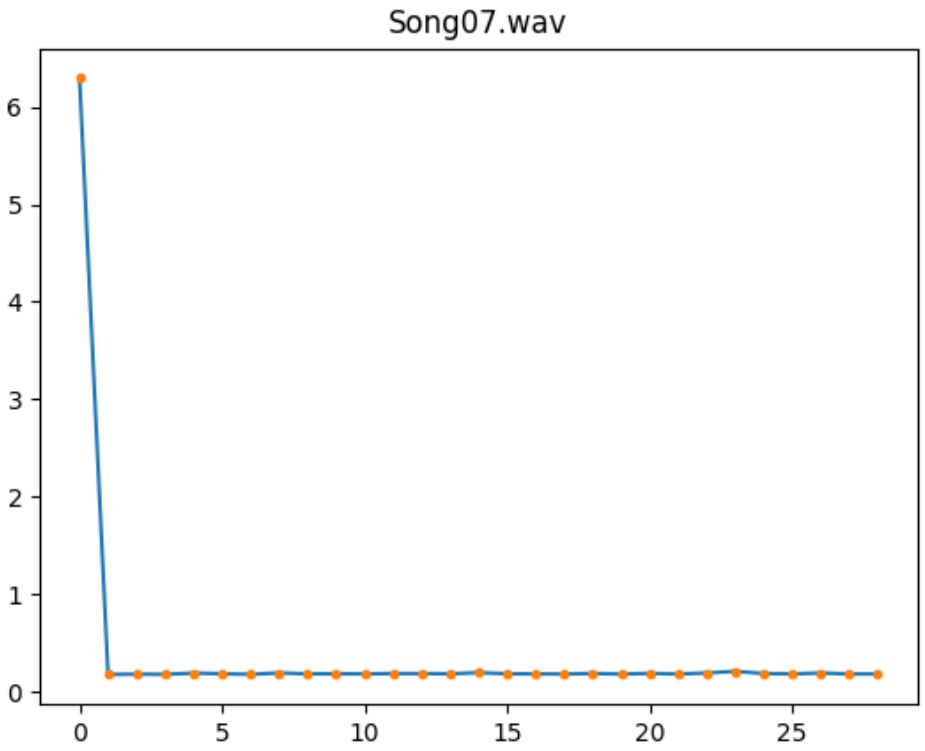


Figura 3.2 – Informação mútua entre os ficheiros song07.wav e guitarSolo.wav

Song06.wav e **Song07.wav** são sons semelhantes ao **guitarSolo.wav**, mas com mais ruído, a única diferença entre os dois sons é que o volume do **Song06.wav** é superior ao do **Song07.wav**, por esse motivo o ruído fica mais disfarçado, assim o maior valor de informação mútua entre os dois é o do **Song06.wav**.

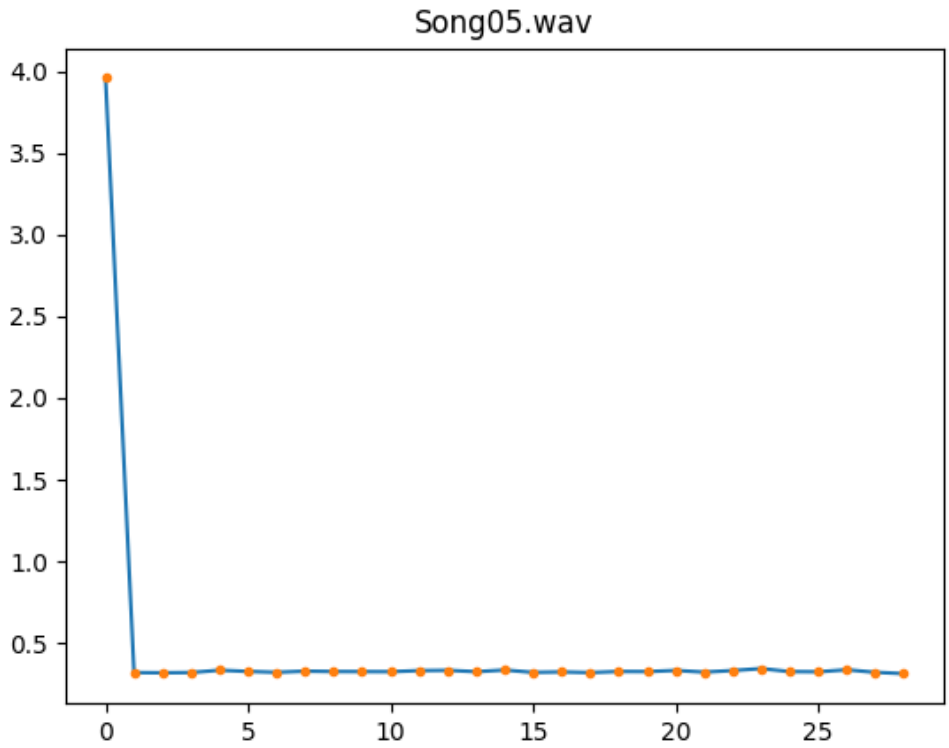


Figura 3.3 – Informação mútua entre os ficheiros song05.wav e guitarSolo.wav

Song05.wav é o som do ficheiro **guitarSolo.wav**, mas com mais ruído comparado com o **Song06.wav** e o **Song07.wav**. Por essa razão, podemos observar no gráfico que o valor da informação mútua é menor comparado com os mesmos.

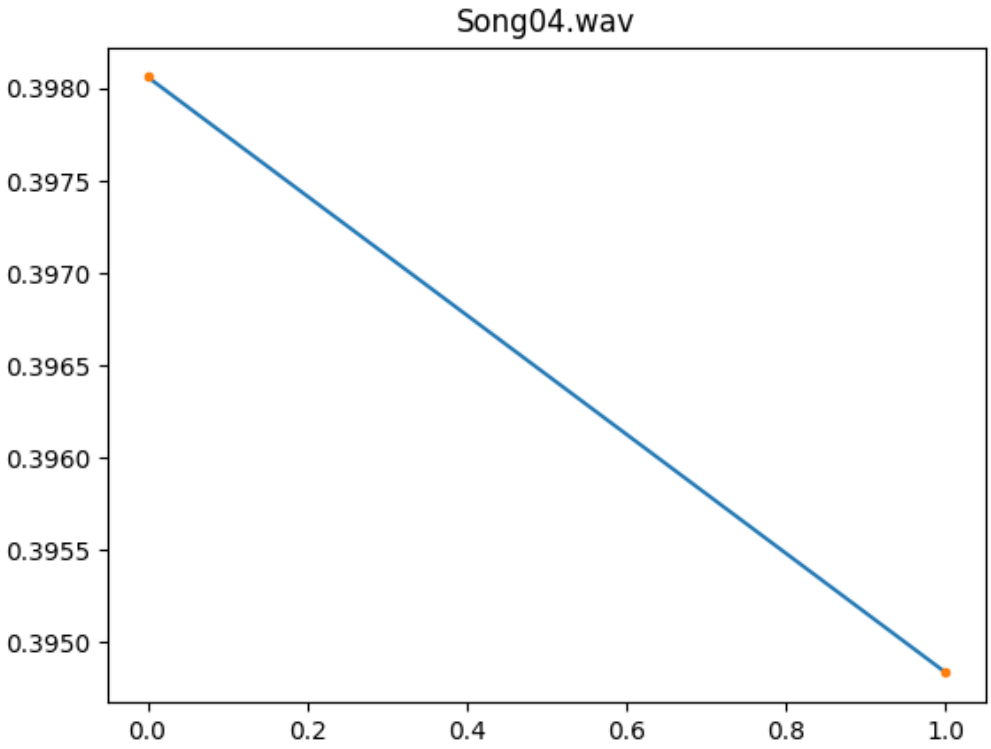


Figura 3.4 – Informação mútua entre os ficheiros song04.wav e guitarSolo.wav

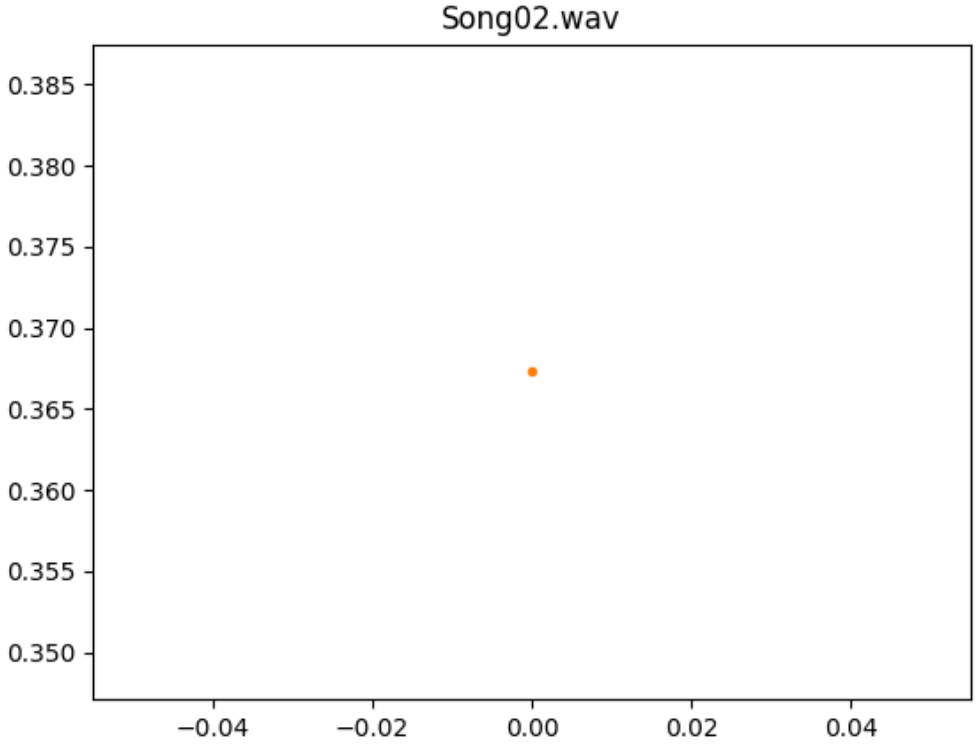


Figura 3.5 – Informação mútua entre os ficheiros song02.wav e guitarSolo.wav

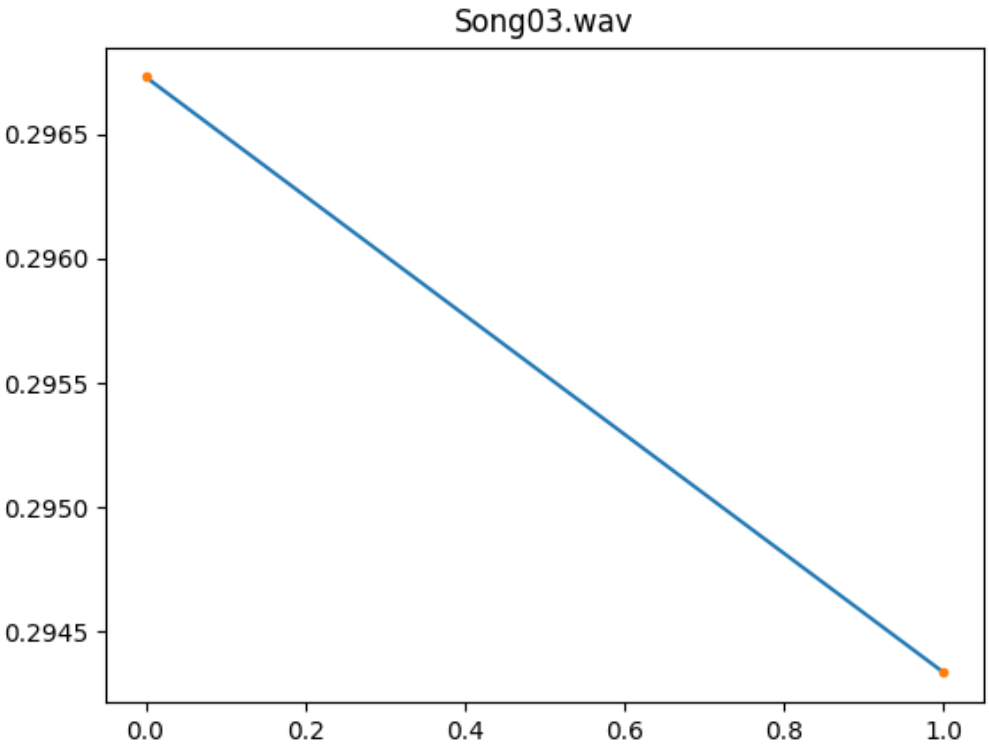


Figura 3.6 – Informação mútua entre os ficheiros song03.wav e guitarSolo.wav

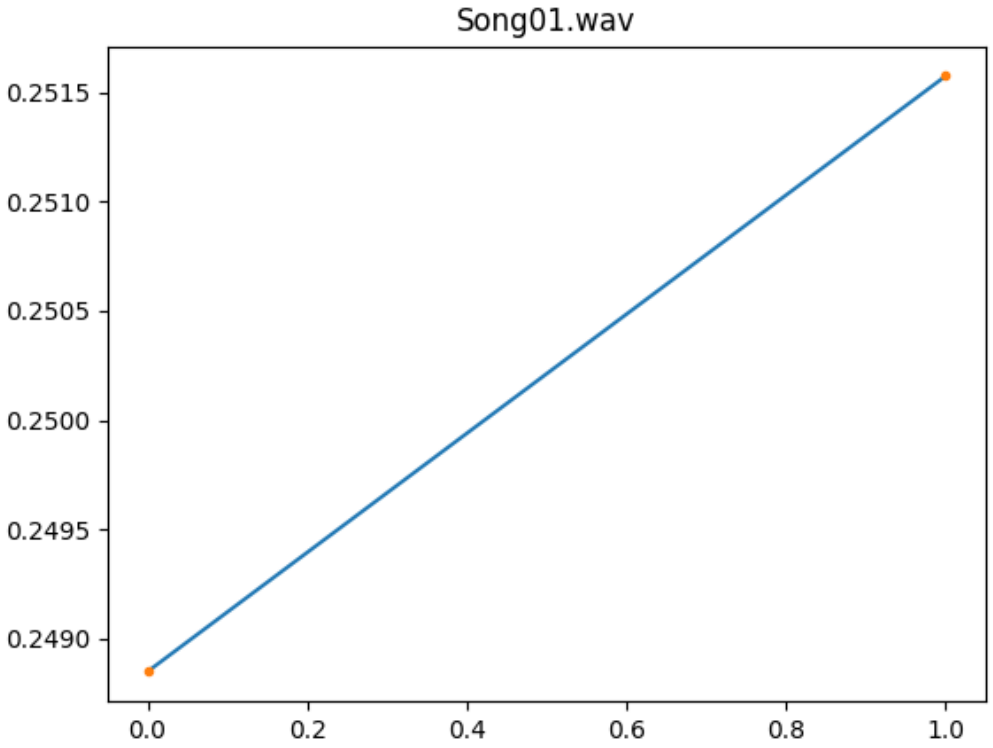


Figura 3.7 – Informação mútua entre os ficheiros song01.wav e guitarSolo.wav

Os sons 1,2,3,4 são sons completamente diferentes e, apesar do **Song04.wav** e **o Song02.wav** terem pontos onde o valor da informação mútua é mais elevado, não têm qualquer semelhança com o **guitarSolo.wav**.

Conclusão

- No final deste trabalho foi possível entender melhor as aplicações da entropia, redundância e informação mútua num **contexto real**.
- Foi também útil para aprendermos mais sobre python e as demais bibliotecas existentes e também para percebermos que tipos de aplicações têm os conceitos aprendidos na cadeira de IPRP do primeiro ano.

Webgrafia/bibliografia:

Slides dados nas aulas teóricas

https://pt.wikipedia.org/wiki/Informa%C3%A7%C3%A3o_m%C3%BAtua

https://pt.wikipedia.org/wiki/Entropia