2º Ano – Licenciatura em Engenharia Informática

Teoria da Informação

Trabalho Prático nº1

Entropia, Redundância e Informação Mútua

· U C ·

2º Ano – Licenciatura em Engenharia Informática

Teoria da Informação

Trabalho Prático nº1

Entropia, Redundância e Informação Mútua

Trabalho realizado por:

Gonçalo J. Rodrigues Ferreira, 2019219213

João A. da Silva Melo, 2019216747

Miguel A. G. de Almeida Faria, 2019216809

· U C ·

Índice

Introdução

Este trabalho prático foi realizado no âmbito da disciplina de TI (Teoria da Informação).

No intuito de aprofundar conhecimentos de Entropia, Redundância e Informação Mútua, este trabalho permitiu aplicar estes tópicos conjuntamente com o uso de bibliotecas de Python.

Além disso, permitiu-nos aplicar os conceitos numa componente mais prática, através da aplicação em imagens e ficheiros de texto e som.

Para uma melhor compreensão do trabalho, apresentamos o significado de algumas das noções mais relevantes:

- Informação – dados obtidos através da ocorrência de um evento.
- Fonte – local onde está contida toda a informação do evento a ser estudado.
- Entropia – medida de incerteza para uma variável aleatória; no âmbito da disciplina representa o limite mínimo teórico do número de bits por símbolo; matematicamente representa-se por:

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

- Informação Mútua – dependência entre duas fontes, ou no caso mais geral, entre duas variáveis, a quantidade de informação que uma variável contém relativamente à outra: matematicamente representa-se por:

$$I(X,Y) = \sum_{x \in A_x} \sum_{y \in A_y} P(x,y) \log_2 \frac{P(y|x)}{P(y)} = H(Y) - H(Y|X)$$
$$= H(X) + H(Y) - H(X,Y)$$

- Query – neste trabalho em específico trata-se de um trecho sonoro, tendo como finalidade o estudo e comparação da informação contida no mesmo, para que seja efetuado o cálculo da informação mútua com outros trechos sonoros de forma a encontrar "um match" entre um e outro, que será o que tiver a informação mútua maior.
- Target - neste trabalho em específico trata-se de um ficheiro sonoro utilizado para comparar com a informação contida na query. Vai ser dividido em várias janelas com a mesma quantidade de informação da query.

Os histogramas, valores calculados e gráficos podem ser consultados no anexo.

Exercício 1 – histograma de ocorrência de símbolos

No exercício 1 usamos as funções histograma e *numOcorrencias*. A função *histograma* recebe como argumento a Fonte, o Alfabeto e o nome do ficheiro. Com o auxílio da função *numOcorrencias* obtém-se o número de vezes que cada símbolo do alfabeto ocorre na fonte, conseguindo assim fazer o histograma.

Na função *numOcorrencias* optámos por utilizar um dicionário dado que foi a maneira mais eficiente que encontrámos para realizar o papel da função.

```
11  # Função que calcula o número de ocorrências de cada símbolo
12  # Parâmetros: P - lista de símbolos ; A - alfabeto com todos os símbolos
13  def numOcorrencias(P, A):
14      no = [0] * len(A) # inicialização lista que vai guardar o número de ocorrências
15      aux = dict(zip(A, no)) # dicionário - símbolos do alfabeto : número de ocorrências
16      for i in P: # aumentar o número de ocorrências dos símbolos da fonte
17          aux[i] += 1
18      no = list(aux.values()) # lista final com o número de ocorrências atualizado
19      return no
```

```
23  # Função que faz uma imagem do histograma da ocorrência dos seus símbolos
24  # Parâmetros: P - fonte de informação ; A - alfabeto com todos os símbolos ; N - nome do ficheiro
25  def histograma(P, A, N):
26      no = numOcorrencias(P, A) # lista com o número de ocorrências
27      # criação do histograma
28      plt.bar(A, no)
29      plt.xlabel('Alfabeto')
30      plt.ylabel('N° de Ocorrências')
31      plt.title(N)
32      plt.show()
33
```

Exercício 2 – Limite mínimo teórico para o número médio de bits/símbolo (Entropia)

No exercício 2 usamos a função *Entropia* e novamente a *numOcorrencias*. A função *Entropia* tem como argumentos a Fonte e o Alfabeto. Com o número de ocorrências, calcula-se a probabilidade de cada um dos valores e aplica-se a fórmula da entropia.

```
36  # Função que calcula o limite mínimo teórico para o número médio de bits por símbolo
37  # Parâmetros: P - fonte de informação ; A - alfabeto com todos os símbolos
38  def entropia(P, A):
39      no = numOcorrencias(P, A) # lista com o número de ocorrências
40      p = [] # lista onde vai ser guardada a probabilidade de cada símbolo
41      for j in range(len(no)): # cálculo da probabilidade
42          p += [no[j]/len(P)]
43      p = np.array(p)
44      p = p[p>0] # retirar casos excecionais
45      H = -sum(p * np.log2(p)) # cálculo da entropia
46      return H
47
```

Exercício 3 – Distribuição estatística (histograma) e limite mínimo para o número médio de bits por símbolo (entropia)

No exercício 3 usamos as funções criadas nos exercícios anteriores. Como temos de analisar diferentes tipos de ficheiro, utilizamos a função *Ex3_4*, que tem como parâmetro o seu nome, e, de acordo com o seu tipo, redireciona para a função correspondente: *image*, *sound* ou *text*. Em cada uma destas funções é retirada a fonte e o alfabeto, de modo a poder ser calculada a entropia e apresentado o respetivo histograma.

```python
# Função que distribui os diferentes ficheiros
# Parâmetros: ficheiro - nome do ficheiro
def Ex3_4(ficheiro):
    tipo = ficheiro.split('.')[1]
    if tipo == 'bmp':
        image(ficheiro)
    if tipo == 'wav':
        sound(ficheiro)
    if tipo == 'txt':
        text(ficheiro)
```

```python
# Função que retira a fonte da imagem
# Parâmetros: I - imagem
def image(I):
    img = mpimg.imread(I) # lista com valores dos pixeis da imagem
    if img.ndim > 2: # retirar apenas o canal R caso tenha mais que dois canais
        img = img[:,:, 0]
    P = img.flatten() # transformar o array para uma dimensão
    A = np.arange(0, 256) # o alfabeto corresponde aos 8 bits de cor do canal
    histograma(P, A, I)
    print("Ficheiro: %s" %I)
    print("Entropia: %.6f" %(entropia(P, A)))
    print("Número médio de bits por símbolo: %.6f" %(huffman(P)))
    print("Variância dos comprimentos dos códigos: %.6f" %(variancia(P)))
```

```python
# Função que retira a fonte do fichero de som
# Parâmetros: S - ficheiro de som
def sound(S):
    data = spiowf.read(S)[1] # lista com as samples por segundo(fs) e informação(data)
    P = data[:, 0] # ficar apenas com a informação do canal esquerdo
    P = np.array(P)
    numBits = int(str(data.dtype)[4:]) # número de bits a usar no alfabeto
    A = np.arange(0, 2**numBits) # array com o alfabeto
    histograma(P, A, S)
    print("Ficheiro: %s" %S)
    print("Entropia: %.6f" %(entropia(P, A)))
    print("Número médio de bits por símbolo: %.6f" %(huffman(P)))
    print("Variância dos comprimentos dos códigos: %.6f" %(variancia(P)))
```

```python
# Função que retira a fonte do fichero de texto
# Parâmetros: T - ficheiro de texto
def text(T):
    # leitura do ficheiro de texto
    f = open(T, 'r')
    dados = f.read()
    f.close()
    P = [] # lista da fonte
    for i in range(len(dados)):
        if (65 <= ord(dados[i]) <= 90) or (97 <= ord(dados[i]) <= 122): # condição para restringir a letras minúsculas e
                                                                        # maiúsculas
            P += [dados[i]]
    P = np.array(P)
    A = np.array(list(map(chr, range(65,91))) + list(map(chr, range(97,123)))) # array que contém o alfabeto (letras
                                                                               # minúsculas e maiúsculas)
    histograma(P, A, T)
    print("Ficheiro: %s" %T)
    print("Entropia: %.6f" %(entropia(P, A)))
    print("Número médio de bits por símbolo: %.6f" %(huffman(P)))
    print("Variância dos comprimentos dos códigos: %.6f" % (variancia(P)))
```

- binaria.bmp

A imagem é composta apenas por pixéis com valor 0 e 255, preto e branco, respetivamente. Este é o ficheiro com menor entropia, uma vez que possui apenas duas cores e consequentemente são necessários menos bits para a representar.

- ct1.bmp

A imagem é idêntica à imagem *binaria.bmp*, no entanto, como possui diferentes tons de cinzento, irá necessitar de mais bits, aumentando o valor da entropia.

- lena.bmp

Este é o ficheiro que tem a maior entropia, as ocorrências dos valores dos pixéis encontram-se melhor distribuídas pela imagem.

- saxriff.wav

Neste ficheiro de som existe um grande número de ocorrências no intervalo 100 a 150. Como o alfabeto vai de 0 a 255, este intervalo representa uma pequena fração dele. Assim, a entropia não é muito alta.

- texto.txt

Este ficheiro possui como alfabeto apenas as letras (maiúsculas e minúsculas, excluindo caracteres especiais). Podemos verificar que existe uma grande distribuição nas letras minúsculas, porém, como nas maiúsculas isto não se verifica a entropia tem um valor intermédio.

Os histogramas encontram-se no Anexo.

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

Como todos os ficheiros possuem uma entropia inferior ao número de bits necessários para os representar (8 bits para imagem e som e 6 para texto), é possível comprimir as fontes de forma não destrutiva. Assim, consegue-se reduzir o tamanho ocupado pelos ficheiros, eliminando elementos redundantes da fonte.

A compressão máxima é dada pela expressão:

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

Como referido anteriormente, a entropia máxima ocorre quando são utilizados 8 bits, no caso da imagem e som, e 6, no caso do texto.

Assim, os valores da entropia, entropia máxima e compressão máxima para cada um dos ficheiros são:

Ficheiro	H(X)	$H_{max}(X)$	TC_{max}
binaria.bmp	0.975527	8	87.81
ct1.bmp	5.972234	8	25.35
lena.bmp	6.915336	8	13.56
saxriff.wav	3.530989	8	55.86
texto.txt	4.196889	6	30.05

Exercício 4 - Número médio de bits por símbolo

No exercício 4 usamos as funções *huffman* e *variância*. Ambas as funções recebem como argumento a fonte e são chamadas na função *Ex3_4*, para não repetir código. Na função *huffman* começa-se por obter o "codec", que vai permitir saber o array de símbolos e o array dos seus comprimentos respetivos. Tendo em conta o pedido, é calculada a média de bits por símbolo. Na função *variância*, faz-se uma abordagem idêntica à da função anterior, aplicando a fórmula da variância:

$$V(X) = E(X^2) - (E(X))^2$$

```
115 # Função que determina o número médio de bits por símbolo
116 # Parâmetros: P - fonte
117 def huffman(P):
118     codec = hc.HuffmanCodec.from_data(P)
119     symbols, lenghts = codec.get_code_len()
120     mediaBits = 0 # variável onde vai ser armazenado o número médio de bits
121     for i in range(len(P)):
122         mediaBits += lenghts[symbols.index(P[i])]/len(P)
123     return mediaBits
```

```
126 # Função que determina a variância dos comprimentos dos códigos
127 # Parâmetros: P - fonte
128 def variancia(P):
129     codec = hc.HuffmanCodec.from_data(P)
130     symbols, lenghts = codec.get_code_len()
131     EX = 0
132     for i in range(len(P)):
133         EX += lenghts[symbols.index(P[i])]/len(P)
134     EX2 = 0
135     for i in range(len(P)):
136         EX2 += lenghts[symbols.index(P[i])]**2/len(P)
137     # arredondamento de casas decimais excessivas
138     EX = round(EX, 6)
139     EX2 = round(EX2, 6)
140     V = EX2 - EX**2
141     return V
```

Os valores do número médio de bits por símbolo e da variância são:

Ficheiro	Número médio de bits	Variância
binaria.bmp	1.000000	0.000000
ct1.bmp	6.007546	5.201658
lena.bmp	6.942505	0.639393
saxriff.wav	3.584290	7.718198
texto.txt	4.217295	1.882716

Podemos constatar que os valores obtidos através do cálculo da entropia e do número médio de bits por símbolo do código de Huffman são bastante semelhantes, como é de esperar, segundo a fórmula:

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

Com a análise dos resultados obtidos, verifica-se que a dispersão dos comprimentos de Huffman é maior nos ficheiros que possuem uma variância mais elevada.

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

É possível reduzir a variância. Para tal, é necessário que a taxa de enchimento seja aproximadamente constante e que se coloquem os símbolos combinados na lista usando a ordem mais elevada possível (código de variância mínima). Assim sendo, tem-se uma árvore binária com menor profundidade e uma distribuição dos comprimentos dos símbolos mais uniforme, tornando a variância mais pequena.

A utilização deste método é útil porque diminui a possibilidade de erros na descodificação e torna o código mais eficiente.

Exercício 5 – Limite mínimo para o número médio de bits por símbolo aplicando agrupamentos de símbolos

No exercício 5 usamos as funções criadas nos Exercícios 1 e 2. Como temos de analisar diferentes tipos de ficheiro, utilizamos a função *Ex5*, que tem como parâmetro o seu nome, e, de acordo com o seu tipo, redireciona para a função correspondente: *imageEx5*, *soundEx5* ou *textEx5*. Em cada uma destas funções é retirada a fonte e o alfabeto e realiza-se o agrupamento de símbolos, fazendo com que cada símbolo seja uma sequência de dois símbolos contíguos, de modo a poder ser calculada a entropia.

```
223 # Função que distribui os diferentes ficheiros
224 # Parâmetros: ficheiro - nome do ficheiro
225 def Ex5(ficheiro):
226     tipo = ficheiro.split('.')[1]
227     if tipo == 'bmp':
228         imageEx5(ficheiro)
229     if tipo == 'wav':
230         soundEx5(ficheiro)
231     if tipo == 'txt':
232         textEx5(ficheiro)
```

```
159 # Função que retira a fonte da imagem
160 # Parâmetros: I - imagem
161 def imageEx5(I):
162     img = mpimg.imread(I) # lista com valores dos pixeis da imagem
163     if img.ndim > 2: # retirar apenas o canal R caso tenha mais que dois canais
164         img = img[:,:, 0]
165     P = img.flatten() # transformar o array para uma dimensão
166     if len(P) % 2 != 0: # remoção do último elemento caso o comprimento da lista seja ímpar
167         P=P[:-1]
168     PAlterado = P.reshape(int(len(P)/2), 2) # transformação da fonte em sequências de dois símbolos contíguos
169     AAlterado = np.unique(PAlterado, axis=0) # excerto do alfabeto total, contendo apenas os elementos existentes também
170                                              # na fonte
171     # transformação dos arrays em listas de tuplos de modo a poderem ser utilizados no dicionário
172     PA = list(map(tuple, PAlterado))
173     AA = list(map(tuple, AAlterado))
174     #histogramaEx5(PA, AA)
175     print("Ficheiro: %s" %I)
176     print("Entropia: %.6f" % (entropia(PA, AA) / 2)) # divisão da entropia por dois porque são pares
```

```
179 # Função que retira a fonte do fichero de som
180 # Parâmetros: S - ficheiro de som
181 def soundEx5(S):
182     data = spiowf.read(S)[1] # lista com as samples por segundo(fs) e informação(data)
183     P = data[:, 0] # ficar apenas com a informação do canal esquerdo
184     if len(P) % 2 != 0: # remoção do último elemento caso o comprimento da lista seja ímpar
185         P=P[:-1]
186     PAlterado = P.reshape(int(len(P)/2), 2) # transformação da fonte em sequências de dois símbolos contíguos
187     AAlterado = np.unique(PAlterado, axis=0) # excerto do alfabeto total, contendo apenas os elementos existentes também
188                                              # na fonte
189     # transformação dos arrays em listas de tuplos de modo a poderem ser utilizados no dicionário
190     PA = list(map(tuple, PAlterado))
191     AA = list(map(tuple, AAlterado))
192     #histogramaEx5(PA, AA)
193     print("Ficheiro: %s" %S)
194     print("Entropia: %.6f" % (entropia(PA, AA) / 2)) # divisão da entropia por dois porque são pares
```

```
197 # Função que retira a fonte do fichero de texto
198 # Parâmetros: T - ficheiro de texto
199 def textEx5(T):
200     # leitura do ficheiro de texto
201     f = open(T, 'r')
202     dados = f.read()
203     f.close()
204     P = []
205     for i in range(len(dados)):
206         if (65 <= ord(dados[i]) <= 90) or (97 <= ord(dados[i]) <= 122): # condição para restringir a letras minúsculas e
207                                                                         # maiúsculas
208             P += [dados[i]]
209     P = np.array(P)
210     if len(P) % 2 != 0: # remoção do último elemento caso o comprimento da lista seja impar
211         P = P[:-1]
212     PAlterado = P.reshape(int(len(P)/2), 2) # transformação da fonte em sequências de dois símbolos contíguos
213     AAlterado = np.unique(PAlterado, axis=0) # excerto do alfabeto total, contendo apenas os elementos existentes também
214                                              # na fonte
215     # transformação dos arrays em listas de tuplos de modo a poderem ser utilizados no dicionário
216     PA = list(map(tuple, PAlterado))
217     AA = list(map(tuple, AAlterado))
218     #histogramaEx5(PA, AA)
219     print("Ficheiro: %s" %T)
220     print("Entropia: %.6f" % (entropia(PA, AA) / 2)) # divisão da entropia por dois porque são pares
```

Os valores da entropia e da entropia agrupada são:

Ficheiro	Entropia	Entropia do agrupamento de símbolos
binaria.bmp	0.975527	0.542405
ct1.bmp	5.972234	4.481268
lena.bmp	6.915336	5.596516
saxriff.wav	3.584290	2.889866
texto.txt	4.196889	3.754269

Conclui-se que, com o agrupamento de símbolos, obtém-se um valor de entropia mais baixo comparativamente ao calculado no Exercício 3.

Assim, torna-se vantajoso utilizar agrupamento de símbolos como forma de reduzir o valor de entropia em qualquer um dos ficheiros.

Exercício 6 – Informação Mútua

a) Função que devolve o vetor de valores de informação mútua em cada janela

No Exercício 6a usamos as funções *informacaoMutua*, *entropiaConjunta* e *entropia*. Tanto a função *entropiaConjunta* como a função *informacaoMutua* têm como parâmetros a query, o target e o alfabeto, sendo que a última recebe ainda o passo (intervalo entre janelas consecutivas). Na função *entropiaConjunta* é criada uma matriz quadrada onde vai sendo incrementado o número de ocorrências de cada par, para calcular a sua probabilidade e

posteriormente a sua entropia. A função *informacaoMutua* calcula cada uma das janelas do target. Obtém também o valor da entropia da query, da janela e a entropia conjunta destas, de modo a obter a informação mútua: $H(X) + H(Y) - H(X,Y)$

```
# Função que calcula a entropia conjunta
# Parâmetros: Q - query ; T - target ; A - alfabeto
def entropiaConjunta(Q, T, A):
    a2=np.zeros((len(A), len(A))) # matriz com a dimensão do comprimento do alfabeto, com elementos inicializados a 0,
                                  # onde se vai incrementando o número de ocorrências de cada par
    for i in range(len(Q)): # procura da posição de cada par na matriz para incrementar
        indQ = np.where(A == Q[i])
        indT = np.where(A == T[i])
        a2[indQ, indT] += 1
    a2 = a2/len(Q) # cálculo da probabilidade
    a2 = a2.flatten() # transformar o array para uma dimensão
    a2 = a2[a2>0] # tirar casos excecionais
    H = -sum(a2*np.log2(a2)) # cálculo da entropia
    return H
```

```
# Função que calcula a informação mútua entre a query e o target
# Parâmetros: Q - query ; T - target ; A - alfabeto ; passo - intervalo entre janelas
def informacaoMutua(Q, T, A, passo):
    informacao = [] # lista onde vão ser guardados os valores da informação mútua para cada janela
    for i in range(0, len(T)-len(Q)+1, passo): # ciclo que permite percorrer o target com o passo
        janela = np.copy(T[i:i+len(Q)]) # array com a janela
        # cálculo das entropias
        HX = entropia(Q, A)
        HY = entropia(janela, A)
        HXY = entropiaConjunta(Q, janela, A)
        # cálculo e adição da informação mútua
        inf = HX + HY - HXY
        informacao += [inf]
    return informacao
```

b) Variação da informação mútua entre "saxriff.wav" e os ficheiros "target01 - repeat.wav" e "target02 – repeatNoise.wav"

No Exercício 6b usamos as funções *Ex6b*, *Ex6* e *graficoEvolucaoEx6b*. Como temos de calcular a variação da informação mútua de dois ficheiros e visualizá-la graficamente, utilizamos a função *Ex6b*, que recebe o nome dos ficheiros. A função *Ex6* tem como parâmetros o ficheiro utilizado como target, o ficheiro utilizado como query e o passo. Começa por retirar o target, a query, o alfabeto e o passo (sendo que neste caso tem o valor de ¼ do comprimento do vetor da query) e calcula o array da informação mútua. A função *graficoEvolucaoEx6b* recebe como argumento dois vetores de informação mútua e cria os seus gráficos correspondentes.

```
# Função que calcula a variação da informação mútua para dois targets diferente e as compara
# Parâmetros: St1 - nome do ficheiro de som de um target ; St2 - nome do ficheiro de som de outro target ; Sq - nome do
# ficheiro de som da query
def Ex6b(St1, St2, Sq):
    infM1 = Ex6(St1, Sq, 0.25) # cálculo da informação mútua de um ficheiro de som
    infM2 = Ex6(St2, Sq, 0.25) # cálculo da informação mútua de outro ficheiro de som
    graficoEvolucaoEx6b(infM1, infM2)  # visualizar a evolução da informação mútua ao longo do tempo em ambos
```

```python
# Função que retira a informação dos ficheiros de som e calcula a informação mútua
# Parâmetros: S1 - nome do ficheiro de som de target ; S2 - nome do ficheiro de som de query ; passo - intervalo entre
# janelas
def Ex6(St, Sq, passo):
    data1 = spiowf.read(St)[1] # lista com as samples por segundo(fs) e informação(data1) do target
    if data1.ndim > 1:
        T = data1[:, 0] # ficar apenas com a informação do canal esquerdo
    else:
        T = data1
    data2 = spiowf.read(Sq)[1] # lista com as samples por segundo(fs) e informação(data2) da query
    Q = data2[:, 0] # ficar apenas com a informação do canal esquerdo
    numBits = int(str(data2.dtype)[4:])  # número de bits a usar no alfabeto
    A = np.arange(0, 2**numBits) # array com o alfabeto
    passo = int(passo*len(Q)) # cálculo do passo consoante o comprimento da query
    infM = informacaoMutua(Q, T, A, passo) # cálculo da informação mútua
    infM = np.array(infM)
    infM = np.around(infM, decimals=6)  # arredondamento dos valores da informação mútua
    print("Evolução da informação mútua (%s): " %St + str(infM))
    return infM
```

```python
# Função que faz uma imagem do gráfico da evolução da informação mútua ao longo do tempo para cada target, de dois
# ficheiros de som
# Parâmetros: informacao1 - lista com os valores da informação mútua de um ficheiro de som ; informacao1 - lista com os
# valores da informação mútua de outro ficheiro de som
def graficoEvolucaoEx6b(informacao1, informacao2):
    # criação do gráfico
    plt.plot(informacao1)
    plt.plot(informacao2)
    plt.xlabel('Janelas')
    plt.ylabel('Informação Mútua')
    plt.title('Evolução da Informação Mútua')
    plt.show()
```

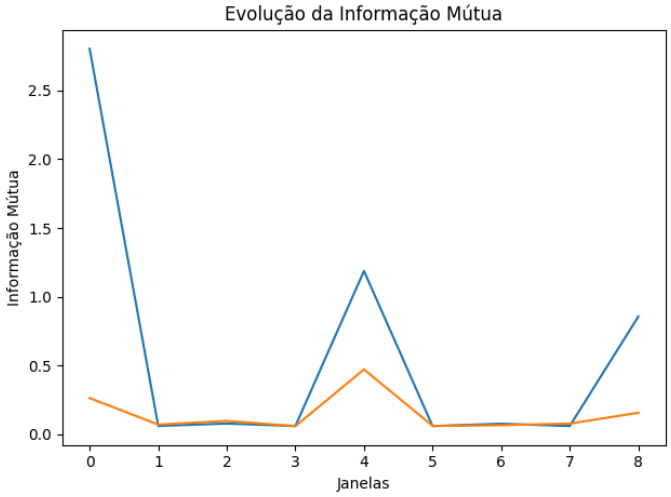
Resultados da evolução da informação mútua:

- "target01 - repeat.wav":

[2.802683 0.061817 0.07931 0.06122 1.188246 0.06078 0.078063 0.061069 0.85729]

- "target02 - repeatNoise.wav":

[0.264677 0.071612 0.098595 0.06061 0.472681 0.062073 0.067259 0.078971 0.157641]



Conclui-se que há mais informação mútua entre o ficheiro "target01 – repeat.wav" e o ficheiro "saxriff.wav" do que no caso do ficheiro "target02 - repeatNoise.wav" e "saxriff.wav". Isto poderá dever-se à inserção de ruído no "target02 - repeatNoise.wav", o que provoca uma menor dependência entre o ficheiro target e query.

c) Simulador de identificação de música usando o ficheiro "saxriff.wav" como query e os ficheiros "Song*.wav" como target

No Exercício 6c usamos as funções *Ex6c*, *Ex6* e *graficoEvolucaoIM*. Como temos de determinar evolução e o valor máximo da informação mútua, utilizamos a função *Ex6c*, que recebe o nome dos ficheiros de target e query. A função *graficoEvolucaoIM* recebe como argumentos o array da informação mútua e o nome do ficheiro target e cria o seu gráfico.

```
323 # Função que calcula a variação da informação mútua e a variação mútua máxima
324 # Parâmetros: St - nome do ficheiro de som do target ; Sq - nome do ficheiro de som da query
325 def Ex6c(St, Sq):
326     infM = Ex6(St, Sq, 0.25) # cálculo da informação mútua
327     infM = np.array(infM)
328     infM = np.around(infM, decimals=6)  # arredondamento dos valores da informação mútua
329     IMmax = np.amax(infM) # descobre a informação mútua máxima
330     print("Informação mútua máxima: " + str(IMmax))
331     graficoEvolucaoIM(infM, St) # visualizar a evolução da informação mútua ao longo do tempo
```

```
282 # Função que faz uma imagem do gráfico da evolução da informação mútua ao longo do tempo para cada target
283 # Parâmetros: informacao - lista com os valores da informação mútua de todas as janelas ; S - nome do ficheiro de som
284 def graficoEvolucaoIM(informacao, S):
285     # criação do gráfico
286     plt.plot(informacao)
287     plt.xlabel('Janelas')
288     plt.ylabel('Informação Mútua')
289     plt.title('%s' %S)
290     plt.show()
```

Os gráficos e os valores da informação mútua encontram-se no Anexo.

A tabela indica os ficheiros por ordem decrescente de informação mútua máxima:

Ficheiro	Informação Mútua Máxima
Song06.wav	3.530989
Song07.wav	
Song05.wav	0.532242
Song04.wav	0.190587
Song02.wav	0.189038
Song03.wav	0.167568
Song01.wav	0.136342

Verifica-se que os ficheiros "Song06.wav" e "Song07.wav", como possuem uma informação mútua superior ao resto dos ficheiros, são os mais parecidos ao ficheiro "saxriff.wav".

Nos ficheiros "Song05.wav", "Song06.wav" e "Song07.wav" constatámos que apenas existe um valor para a informação mútua. Isto deve-se ao facto de que os targets e a query têm o mesmo tamanho, podendo haver apenas uma janela.

Conclusão

Com a realização deste trabalho conseguimos aprofundar os conhecimentos não só sobre os temas abordados na aula, como a entropia e a informação mútua, mas também sobre o uso do *numpy*, *matplotlib* e *scipy*.

Concluímos que com a análise dos histogramas é possível prever o valor da entropia de acordo com a dispersão no histograma. Verificamos também que a entropia agrupada é menor que a entropia, dado que há uma otimização do código. Por último, conseguimos compreender o funcionamento de uma das aplicações mais utilizadas mundialmente no reconhecimento de músicas, a app "Shazam", ao nível mais técnico.

Bibliografia

PowerPoints fornecidos pelo professor (PPT Cap.II s.17, s.51-52, s.110, s.104)

www.numpy.org

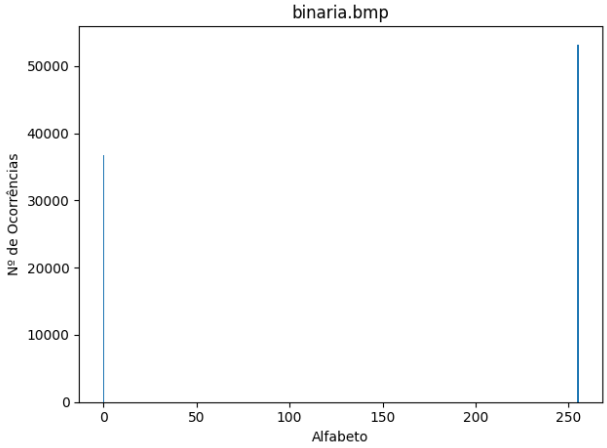
2º Ano – Licenciatura em Engenharia Informática

Teoria da Informação

Trabalho Prático nº1

Entropia, Redundância e Informação Mútua

Anexo



Ficheiro - binaria.bmp:

Ex3

Entropia: 0.975527

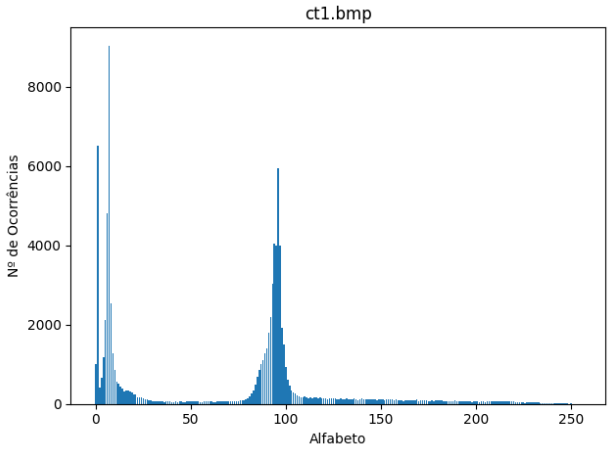
Ex4

Número médio de bits por símbolo: 1.000000

Variância dos comprimentos dos códigos: 0.000000

Ex5

Entropia: 0.542405



Ficheiro – ct1.bmp:

Ex3

Entropia: 5.972234

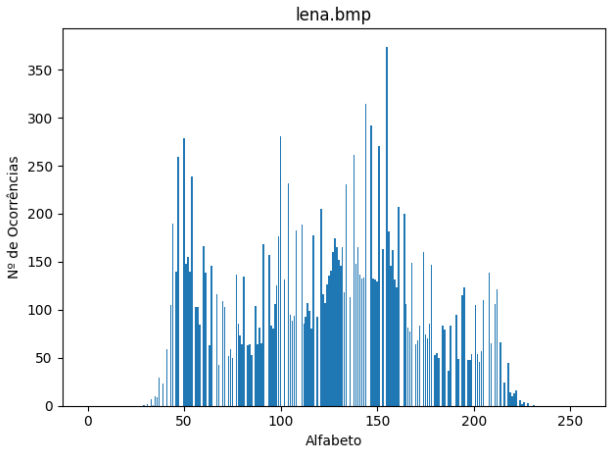
Ex4

Número médio de bits por símbolo: 6.007546

Variância dos comprimentos dos códigos: 5.201658

Ex5

Entropia: 4.481268



Ficheiro – lena.bmp:

Ex3

Entropia: 6.915336

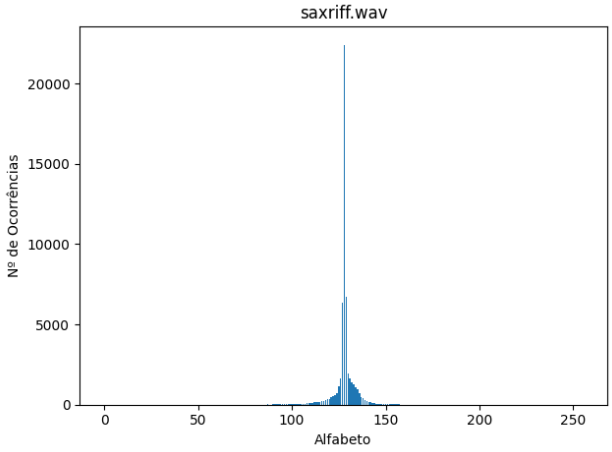
Ex4

Número médio de bits por símbolo: 6.942505

Variância dos comprimentos dos códigos: 0.639393

Ex5

Entropia: 5.596516



Ficheiro – saxriff.wav:

Ex3

Entropia: 3.530989

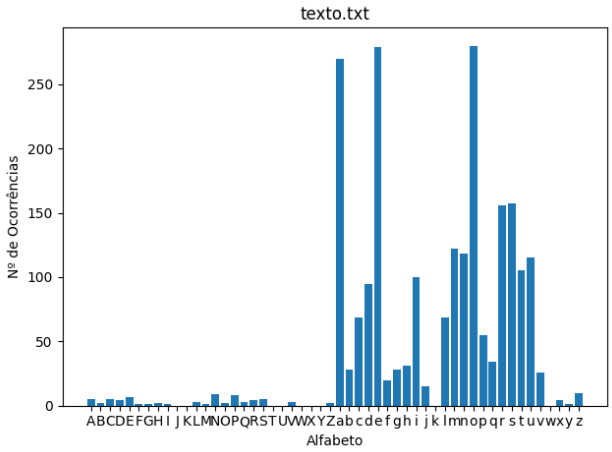
Ex4

Número médio de bits por símbolo: 3.584290

Variância dos comprimentos dos códigos: 7.718198

Ex5

Entropia: 2.889866



Ficheiro – texto.txt:

Ex3

Entropia: 4.196889

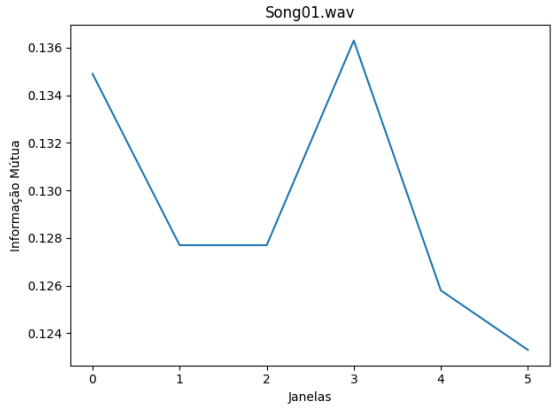
Ex4

Número médio de bits por símbolo: 4.217295

Variância dos comprimentos dos códigos: 1.882716

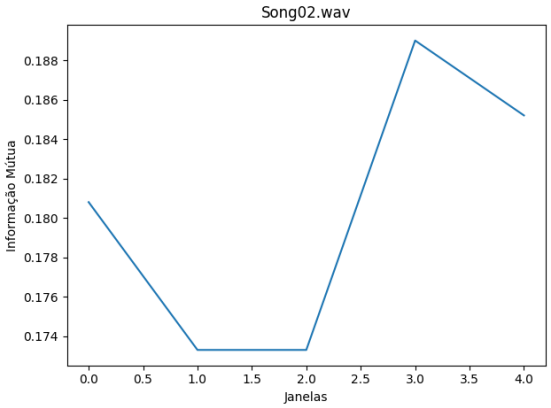
Ex5

Entropia: 3.754269



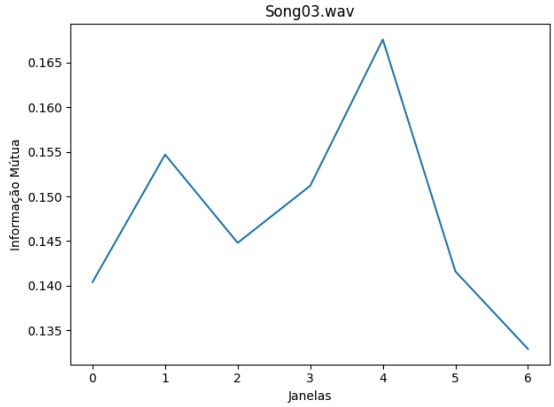
Evolução da informação mútua (Song01.wav): [0.134901 0.127661 0.127703 0.136342 0.12576 0.123305]

Informação mútua máxima: 0.136342



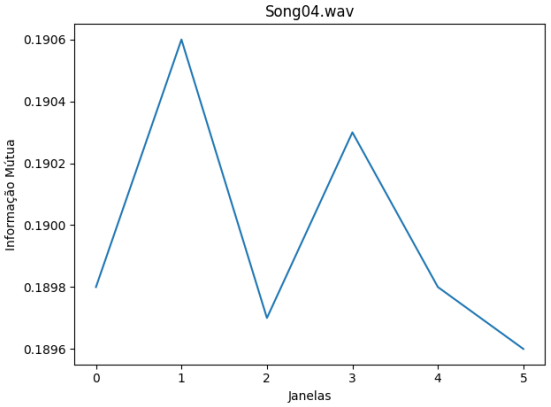
Evolução da informação mútua (Song02.wav): [0.180766 0.173313 0.173322 0.189038 0.185154]

Informação mútua máxima: 0.189038



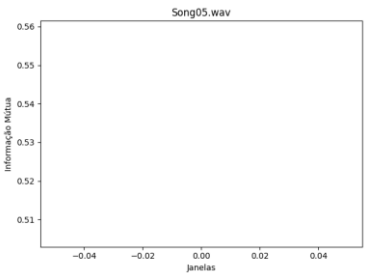
Evolução da informação mútua (Song03.wav): [0.140424 0.154747 0.14485 0.151197 0.167568 0.141633 0.132851]

Informação mútua máxima: 0.167568



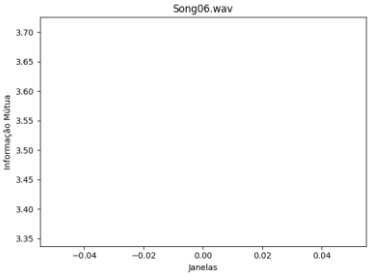
Evolução da informação mútua (Song04.wav): [0.189791 0.190587 0.189729 0.190276 0.18981 0.189628]

Informação mútua máxima: 0.190587



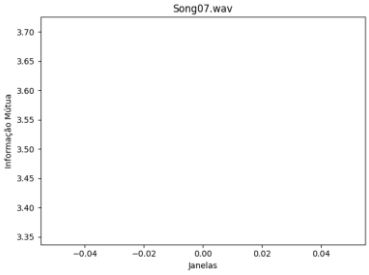
Evolução da informação mútua (Song05.wav): [0.532242]

Informação mútua máxima: 0.532242



Evolução da informação mútua (Song06.wav): [3.530989]

Informação mútua máxima: 3.530989



Evolução da informação mútua (Song07.wav): [3.530989]

Informação mútua máxima: 3.530989