• U C •

UNIVERSIDADE DE COIMBRA

Faculdade de Ciências e Tecnologias

Departamento de Engenharia Informática

Teoria da Informação

Trabalho Prático Nº1 2017/2018

Entropia, Redundância e Informação Mútua

Denzel Nascimento • 2016151477 • uc2016151477@student.uc.pt
Inês Amaro • 2016226582 • uc2016226582@student.uc.pt
Madalena Santos • 2016226726 • mcsantos@student.dei.pt

Exercício 1

Para a resolução deste exercício, foi criada a rotina *histograma*, inserida no rotina *pl1_ex1*. A primeira "desenha" o gráfico de barras, através do número de ocorrências que a fonte dada tem no alfabeto dado.

Exercício 2

Através da função *fileparts* do Matlab guardamos em variável a extensão do ficheiro que pretendemos usar para teste. Passamos como argumento para a rotina criada *readFonte* essa mesma extensão, que vai devolver a fonte em forma de matriz, e o respetivo alfabeto. Depois, através da rotina *entropy*, que devolve o valor da entropia para determinada fonte, calculamos o vetor de ocorrências da fonte no seu alfabeto, e calculamos o valor da entropia através da seguinte fórmula:

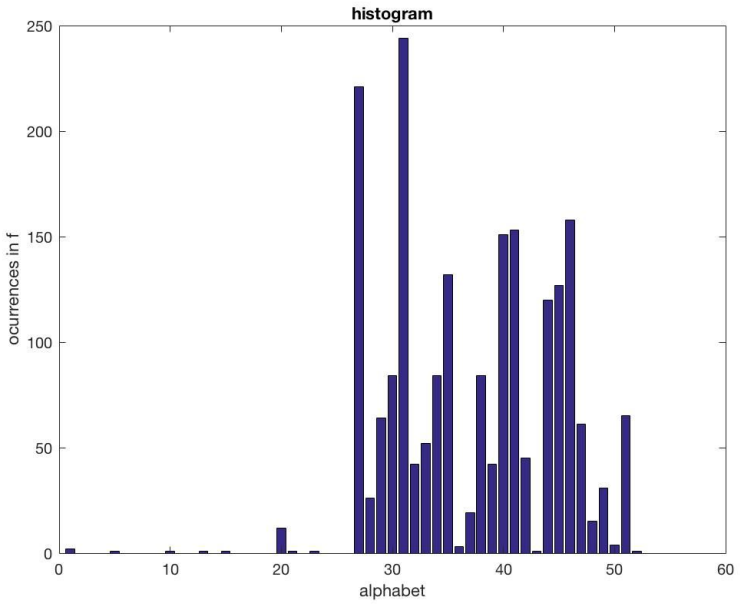
$$H = - \sum_{i \in Alphabet} p(i) * log(p(i))$$

Este valor corresponde ao limite mínimo teórico para o número médio de bits por símbolo.

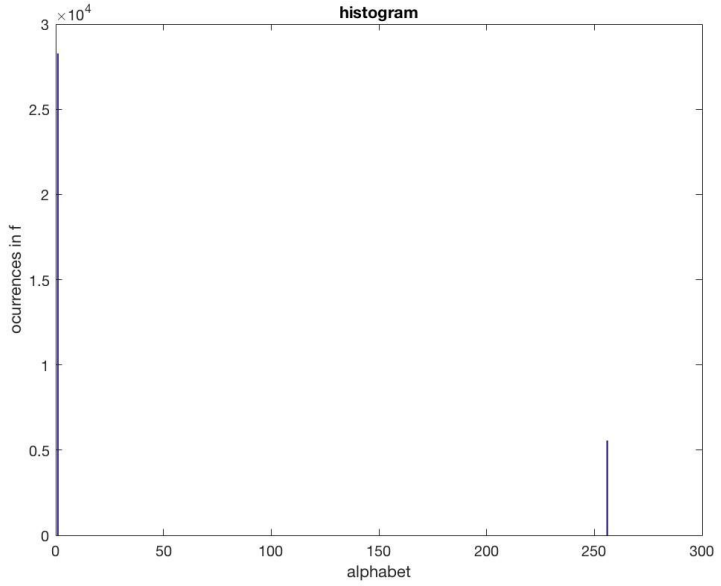
Exercício 3

Mais uma vez utilizamos a função *fileparts* e a rotina *readFonte* para obtermos a matriz da fonte e o vetor alfabeto correspondente. Para obtermos a distribuição estatística da fonte, recorremos à rotina *histograma*, e para calcular a entropia da fonte servimo-nos da rotina *entropy.*
Os símbolos de cada ficheiro são representados por 8 bits. Ainda assim, é possível representar cada um destes símbolos com um número inferior de bits sem perda de informação (compressão não destrutiva). O valor da compressão máxima coincide com os valores da entropia calculados para cada ficheiro, porque estes são os valores mínimos teóricos para a codificação de cada ficheiro.

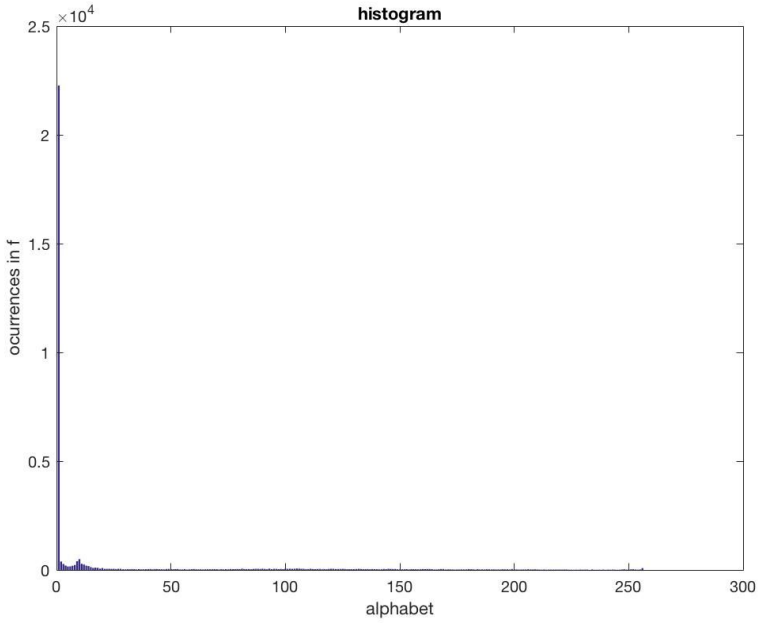
Resultados obtidos:



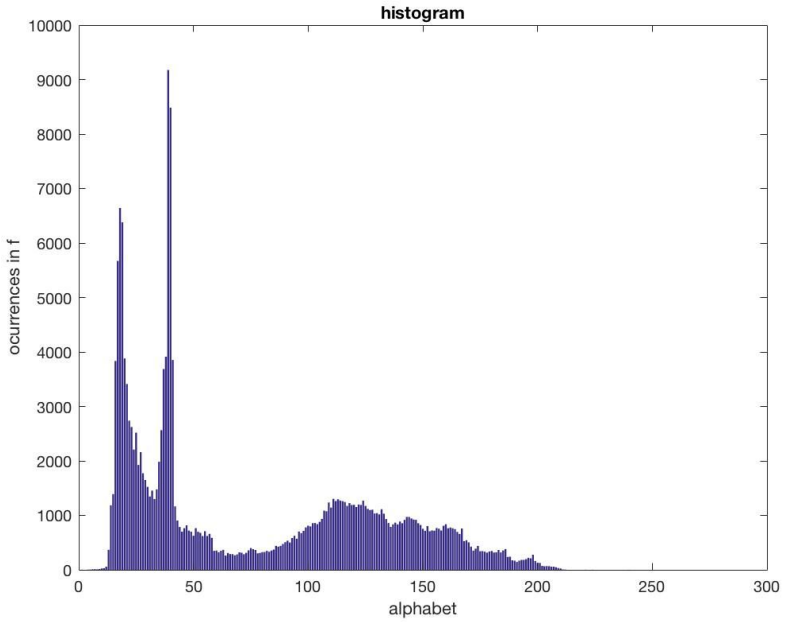
Histograma “english.txt”
Valor para a entropia (bits/símbolo): 4.1681



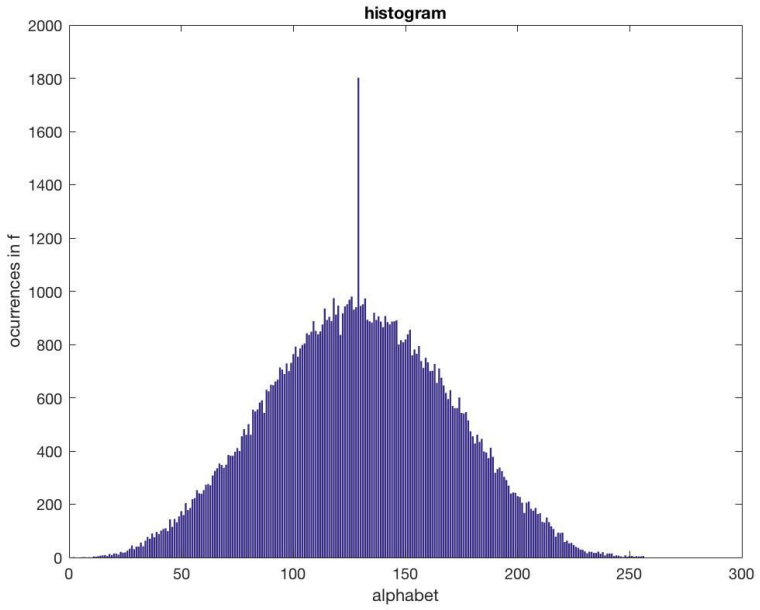
Histograma “homerBin.bmp”
Valor para a entropia (bits/símbolo): 0.6448



Histograma “homer.bmp”
Valor para a entropia (bits/símbolo): 3.4659



Histograma “kid.bmp”
Valor para a entropia (bits/símbolo): 6.9541



Histograma “guitarSolo.wav”
Valor para a entropia (bits/símbolo): 7.3580

Exercício 4

Neste exercício usamos *fileparts* e *readFonte* para transformarmos a fonte e obtermos o alfabeto. Depois, utilizamos a rotina fornecida, *hufflen*, que recebe o vetor de ocorrências dos símbolos do alfabeto na fonte, para calcular a matriz do número de bits utilizado para cada símbolo desse mesmo alfabeto. Ao calcular o somatório e a média dos valores desta matriz, obtemos o valor médio para o número de bits por símbolo utilizado para codificar a fonte. O valor calculado para cada ficheiro é superior ao valor da sua entropia, visto que esta corresponde ao limite mínimo teórico para o número médio de bits por símbolo, e não ao valor real obtido por *hufflen*.

Ao codificar os ficheiros, como o número de bits é diferente para cada símbolo, o valor da variância é maior. A informação mantém-se a mesma, ou seja, estamos na presença de uma codificação não destrutiva. Se a codificação de Huffman for feita com um número mais uniforme de bits para cada símbolo, conseguimos reduzir a variância. Para valores de variância mais baixos, o tempo de codificação para cada símbolo torna-se mais uniforme.

Ficheiro	**Nº médio de bits/símbolo**	**Variância**
“kid.bmp”	6.9832	14.9594
“homer.bmp”	3.5483	1.5566
“homerBin.bmp”	1	0.0078
“english.txt”	3.4640	15.9744
“guitarSolo.wav”	7.3791	8.2338

Exercício 5

No quinto exercício, é nos pedido que calculemos a entropia utilizando agrupamento de símbolos dois a dois. Para isso, foi nos sugerido pelo professor que agrupássemos diretamente os símbolos já existentes na fonte, para melhorar o desempenho do algoritmo. É isso que faz a rotina groupSymbols. Depois, tornamos a fonte numa string sem espaços, e contamos o número de ocorrências dos símbolos do novo alfabeto na fonte já modificada. A partir desta matriz de ocorrências calculamos a entropia para cada ficheiro. Os resultados apresentados já dizem respeito a apenas 1 símbolo.

Fonte	Número médio de bits/símbolo
“english.txt”	1.4756
“guitarSolo.wav”	3.5296
“kid.bmp”	4.9557
“homer.bmp”	3.0589
“homerBin.bmp”	0.6481

Estes valores são bastante inferiores comparativamente aos valores da entropia calculados em (3) ou os valores do número médio de bits por símbolo calculados em (4). Isto deve-se graças ao agrupamento dos símbolos no alfabeto.

Exercício 6 (A)

Para chegar ao resultado proposto no enunciado, utilizamos a rotina *informacaoMutua*, à qual fornecemos os parâmetros (query, target, step, alfabeto) que nos são dados como exemplo.
Nesta rotina começamos por calcular a quantidade de steps que vamos ter que fazer para percorrer todo o target e armazenamos o seu valor na variável “tam”. Recorrendo à função *histc()*, com o” query” e o “alf” como parâmetros, calculamos o número de ocorrências de cada um dos elementos do query (“occurs_query”), por sua vez, para calcular a probabilidade dos elementos, dividimos as ocorrências pela soma de todos os elementos do query (obtendo “probs_query”). De cada vez que andamos um step, obtemos uma nova janela do target, para cada uma dessas janela calculamos as variáveis “ocurrs_janela” e “probs_janela” da mesma maneira que fizemos para o query. Para calcularmos a probabilidade conjunta de dois símbolos, sendo o x do query e y da janela, criamos uma matriz com o tamanho do “alf” e guardamos nela o número de ocorrências de cada par. Para que não tenhamos que percorrer toda a matriz à procura de ocorrências diferentes de zeros usamos a rotina find(), guardando o seu resultado no vetor “ind_nao_zeros”. Para sabermos a linha e a coluna de cada um dos valores do vetor na matriz calculamos o módulo desse elemento e do comprimento do “alf”, para ter a linha, e arredondamos para cima o valor da divisão destes últimos, para ter a coluna.

Para calcular a probabilidade conjunta “pxy” dividimos o valor da matriz nos índices que calculámos em cima pelo comprimento do query. Para obter a probabilidade no query “px” e na janela “py” basta ir buscar os valores que temos nos vetores probs_query e probs_target, nos respetivos índices.
Por fim calculamos a informação mútua:

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

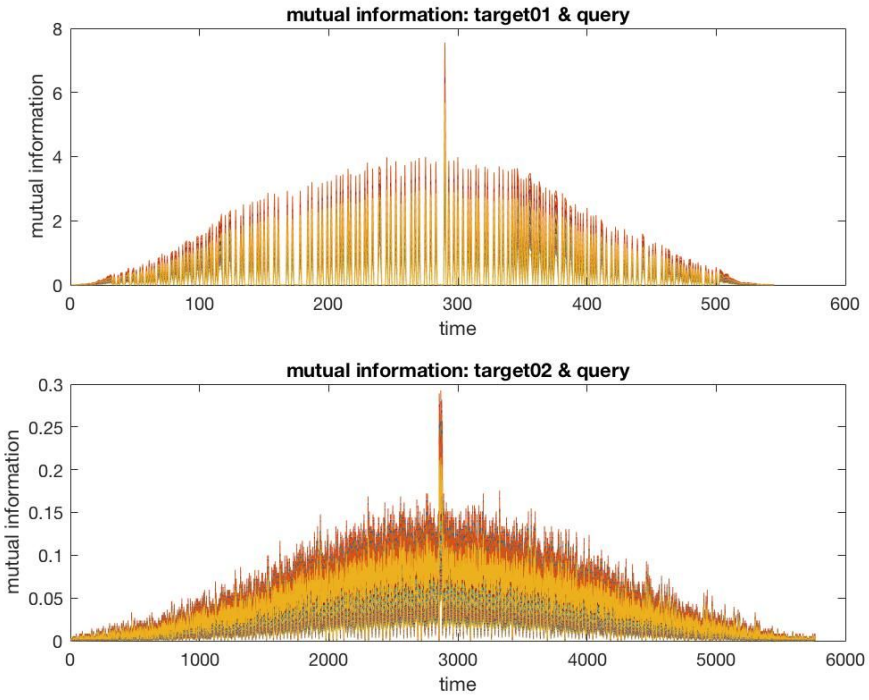
Exercício 6 (B)

Através da rotina infoSom, obtemos a matriz da query e de cada target utilizado. Depois, com a rotina *infoMutuaSom*, calculamos a matriz das frequências conjuntas da seguinte maneira: percorremos a target de n em n elementos (onde n é o step definido no enunciado). Ao contrário daquilo que acontece na alínea (a), não estamos a trabalhar com números inteiros (os valores de som variam de -1 a 1). Então, para cada elemento da query temos que calcular o índice que queremos endereçar na matriz das frequências conjuntas. Depois de algumas contas, chegámos à conclusão que o índice nessa matriz de um determinado elemento podia ser calculado através da equação:
$index = \frac{value + 1}{0.0078}$ onde value é o valor atual da query e do target. 0.0078 é a diferença entre valores consecutivos do alfabeto para ficheiros de som. Com este valor de índice, endereçamos a matriz de frequências na posição (index target, index query), e incrementamos o valor dessa posição em 1 unidade. No final do ciclo, a matriz das frequências representa o número de vezes que cada par de elementos query-target aconteceu.

Por fim, calculamos os valores de informação mútua para cada “momento” através da fórmula

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

Para cada target, calculamos a sua evolução de informação mútua com o query:

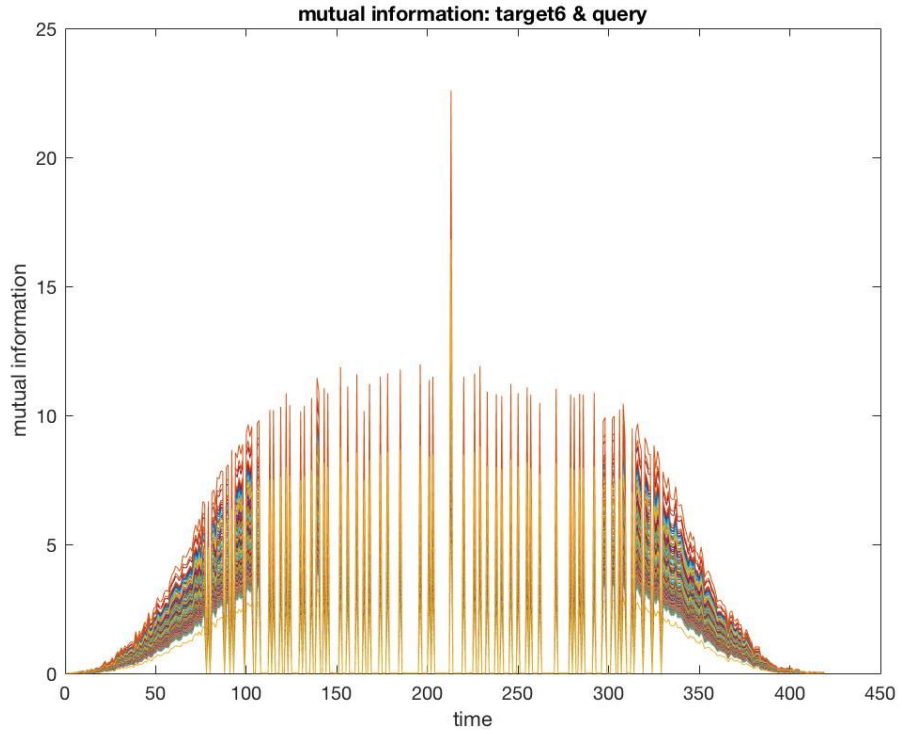


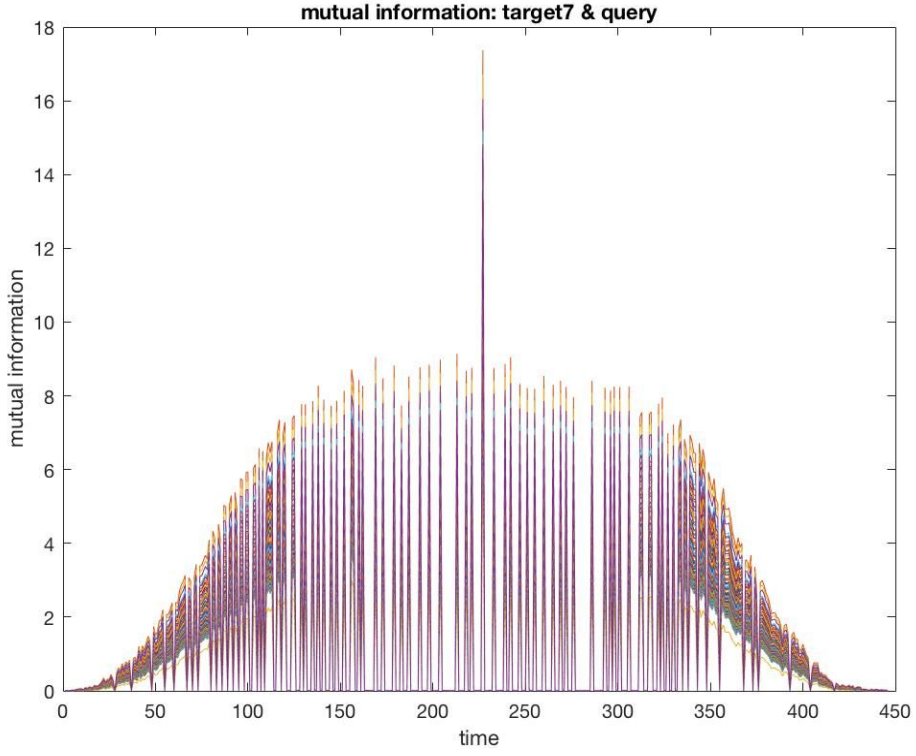
Exercício 6 (c)

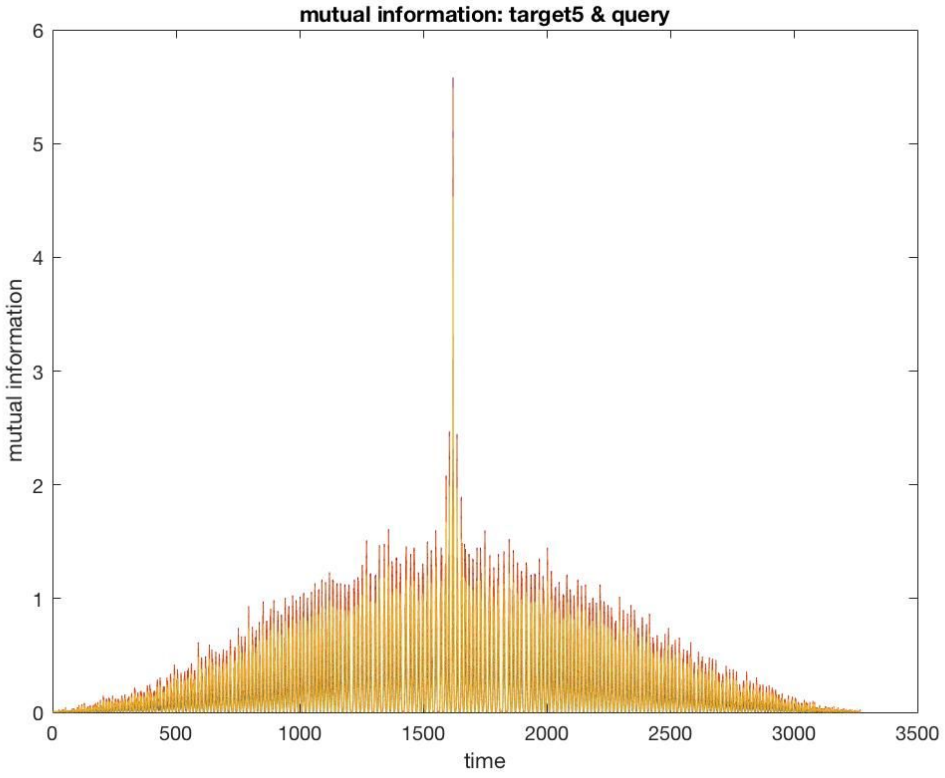
Para cada par target-query, onde a query é o ficheiro “guitarSolo.wav” e os targets são “Song01.wav”, “Song02.wav”, “Song03.wav”, “Song04.wav”, “Song05.wav”, “Song06.wav”ou “Song07.wav”, calculamos o vetor da informação mútua.
Depois, calculamos o máximo de cada vetor (ou seja, a informação mútua máxima para cada par query-target). Obtemos os seguintes resultados:

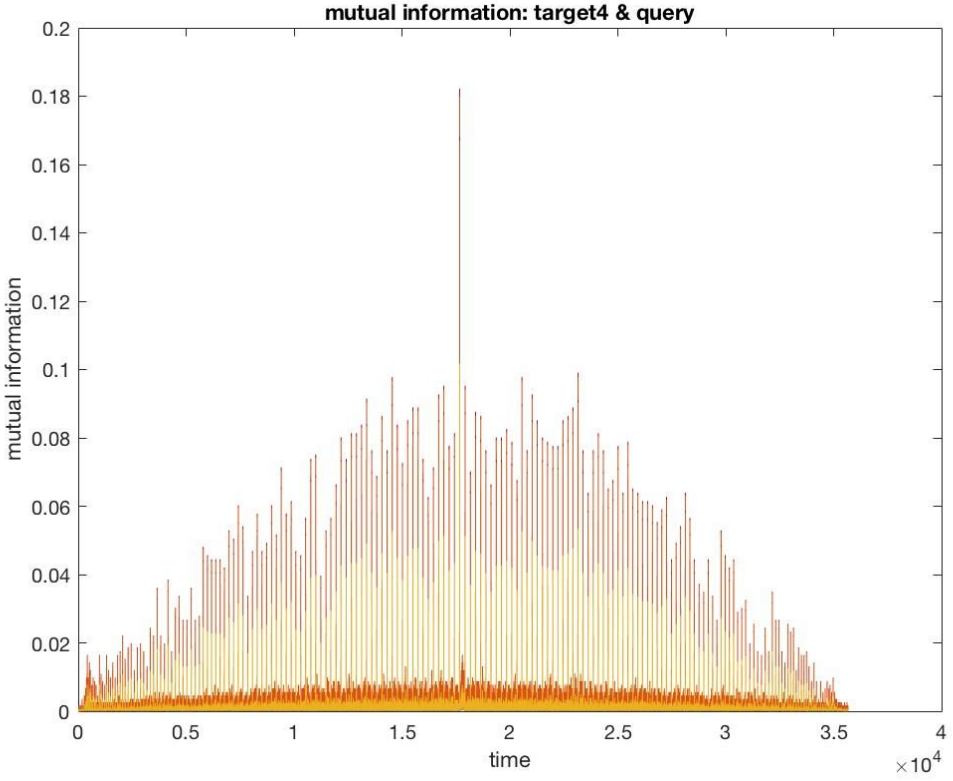
Par query-target	Valor de informação mútua máxima (bits)
“guitarSolo.wav” & “Song01.wav”	0.0632
“guitarSolo.wav” & “Song02.wav”	0.0272
“guitarSolo.wav” & “Song03.wav”	0.0660
“guitarSolo.wav” & “Song04.wav”	0.1823
“guitarSolo.wav” & “Song05.wav”	5.5811
“guitarSolo.wav” & “Song06.wav”	22.5811
“guitarSolo.wav” & “Song07.wav”	17.3723

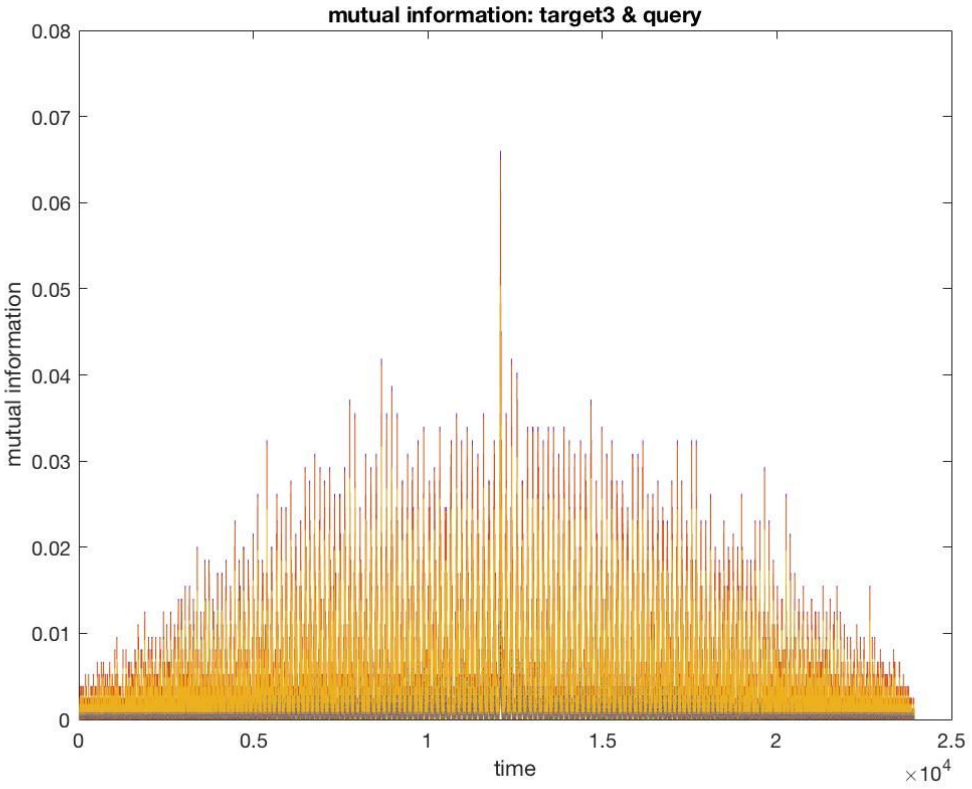
Variação da informação (seriados por ordem decrescente do valor máximo para cada par)

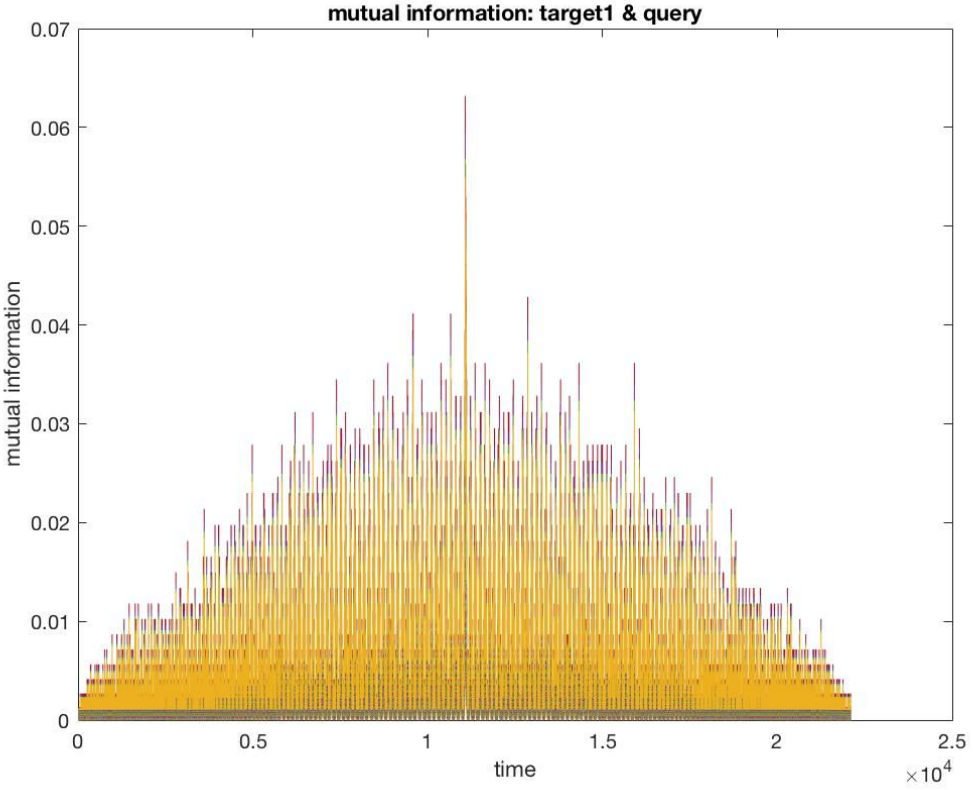

mutual information: target6 & query
mutual information
time
0
5
10
15
20
25
0
50
100
150
200
250
300
350
400
450

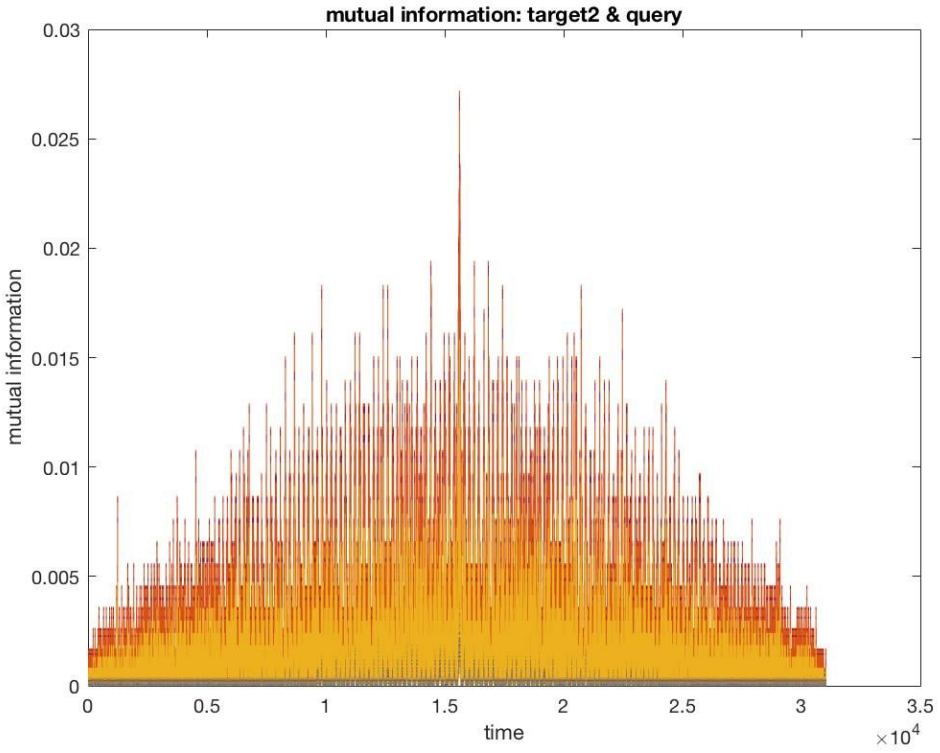

mutual information: target7 & query
mutual information
time
0
2
4
6
8
10
12
14
16
18
0
50
100
150
200
250
300
350
400
450


mutual information: target5 & query
mutual information
time
0
1
2
3
4
5
6
0
500
1000
1500
2000
2500
3000
3500


mutual information: target4 & query
mutual information
time
×10^4
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0
0.5
1
1.5
2
2.5
3
3.5
4


mutual information: target3 & query
mutual information
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
0.5
1
1.5
2
2.5
time
×10^4


mutual information: target1 & query
mutual information
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
0.5
1
1.5
2
2.5
time
×10^4


mutual information: target2 & query
mutual information
time
0.03
0.025
0.02
0.015
0.01
0.005
0
0
0.5
1
1.5
2
2.5
3
3.5
×10^4