FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Teoria da Informação

Diogo André de Freitas Alves-2020214197

Rodrigo Borralho-2021233455

Saulo José Piccirilo Mendes-2021235944

INDICE

Conteúdo

Introdução

No interesse da disciplina de Teoria de Informação realizámos os exercícios de 1 a 6 da ficha tp1. A linguagem introduzida nestes exercícios é Python, e o IDE utilizado é pyCharm. Todas as bases de conhecimentos aplicados neste projeto foram adquiridos a partir da disciplina POO e POAO.

Criámos um Function.py para cada exercício que contém todas as funções utilizadas nos exercícios, uma vez que existem várias funções comuns entre os exercícios realizados e desta forma podemos gerar códigos mais compactos.

EX1

Bibliotecas:

- Matplotlib.pyplot as plt: biblioteca utilizada para produção de gráfico 2D estáticos

Funções:

- Calculo de ocorrências: criamos um array que irá guardar as ocorrências de cada símbolo, com o tamanho do alfabeto. Criamos um dicionário de ocorrências com agrupamentos do alfabeto com o array ocorrências(zip()).O programa percorre toda a fonte e no dicionário de ocorrências é adicionado 1 valor na posição corresponde à ocorrência do símbolo da fonte. Esta função retorna o dicionário de ocorrencias

- Grafico: A função cria_gráfico_barra tem como parâmetros de entrada a fonte, ocorrências, legendas do eixos x, y e o titulo. Utilizámos a função .bar() já incluída na biblioteca importada, que nos permite produzir gráficos de barras 2D, cujos parâmetros utilizados são o alfabeto e a função de calculo de ocorrências. Utilizamos também as funções .xlabel(), .ylabel() e .title() para escrever as legendas dos eixos x, y e titulo respetivamente.

- No gráfico seguinte podemos observar os resultados obtidos para os seguintes parâmetros: Fonte=[1,1,1,2,3] e Alfabeto=[1,2,3]
  Concluindo que os símbolos 1, 2 e 3 ocorrem 3, 1, 1 vezes respetivamente.

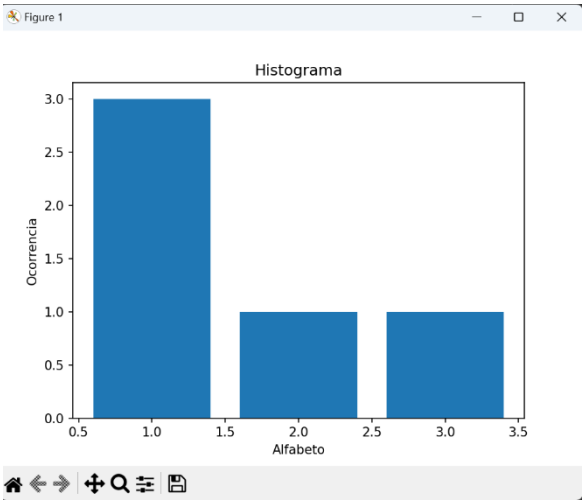


Figura 1 Histograma de ocorrências

EX2

Bibliotecas:

- Matplotlib.pyplot as plt: biblioteca utilizada para produção de gráfico 2D estáticos

Funções:

- Calculo de ocorrências: função importada de Function.py (calcular_ocorrencia)
- Calculo de entropia:
    - Parâmetros: Dicionário de ocorrências
    - *Procedimento*: criámos um array ocorrência com o tamanho e valores do parâmetro de entrada. A Entropia consiste na multiplicação (np.multiply) da probabilidade pelo logaritmo de base 2 da probabilidade inversa. A probabilidade consiste na divisão das ocorrências individuais de cada símbolo pelo tamanho total da fonte(soma de todas as ocorrências).
      Para calcular a probabilidade inversa dividimos 1 pela probabilidade e guardámos o resultado em prob_inver(np.divide()). Em prob_log guardámos o logaritmo de base 2 da probabilidade inversa(np.log2). Após a multiplicação o resultado é guardado no array entropia_individual1 e é retornada a soma dos seus valores

EX3

Bibliotecas:

- Matplotlib.pyplot as plt: biblioteca utilizada para produção de gráfico 2D estáticos
- Scipy.io: fornece funções para otimização, estáticas e processamento de sinal
- PIL: disponibiliza funções da manipulação de imagem
- Numpy: utilizada na criação de arranjos, matrizes e funções matemáticas

Funções:

- Funções de leitura:

  NOTA: todas as funções de leitura retornam uma lista com a fonte, ocorrências, alfabeto e entropia, por esta ordem.

  .*BMP*:

  - Parâmetros: ficheiro
  - Procedimento:
    1. leitura do ficheiro, .open(ficheiro, "r")
    2. armazenamento de data: data=list(img.getdata(0)), criámos uma lista que armazena a matriz R de cada pixel, uma vez que se trata de uma greyscale.
    3. Alfabeto: array com tamanho de 256
    4. Calculámos a ocorrência(função importada de function.py)
    5. Calculámos a entropia(função importada de function.py)

- .*WAV*:
    1. Parâmetros: ficheiro
    2. 1. Leitura: [fs, data]=wavfile.read(ficheiro), onde fs é um array que armazena as frequências e data um array que armazena a informação
    3. Calculámos o numero de bits a quantificar multiplicando o nº de canais(obtido através de len(data.shape) por 8(bits)
    4. Alfabeto: criámos um alfabeto com um tamanho equivalente a 2 levantado ao nº de bits a quantizar, uma vez que, para cada bit existem dois valores possíveis
    5. Ocorrencia: função importada de function.py
    6. Entropia: função importada de function.py

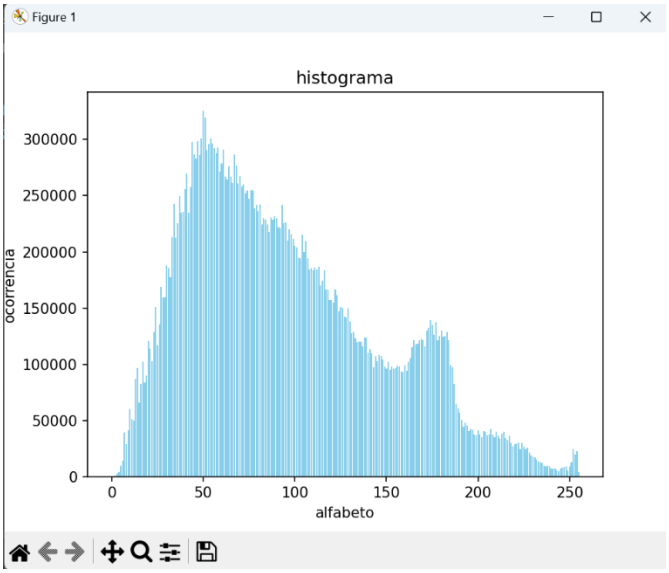
- *.TXT*

1. Ficheiro
2. Alfabeto: o alfabeto é composto por todas a letras Maiúsculas e minúsculas, para tal criámos dois ciclos com limites, sendo o numero inicial equivalente à letra “A” ou “a” e o numero final equivalente a “Z” ou “z”(função ord()), adicionando ao alfabeto as letras correspondentes aos números utilizados(função chr())
3. Fonte: a fonte(texto) fica com o conteúdo do ficheiro através da função .read(-1) (lê ficheiro inteiro)
4. Ocorrência: criámos um dicionário com agrupamentos do alfabeto e das respetivas ocorrências, é percorrida a fonte e caso o elemento da fonte esteja no alfabeto é adicionado um valor na posição correspondente à ocorrência do símbolo

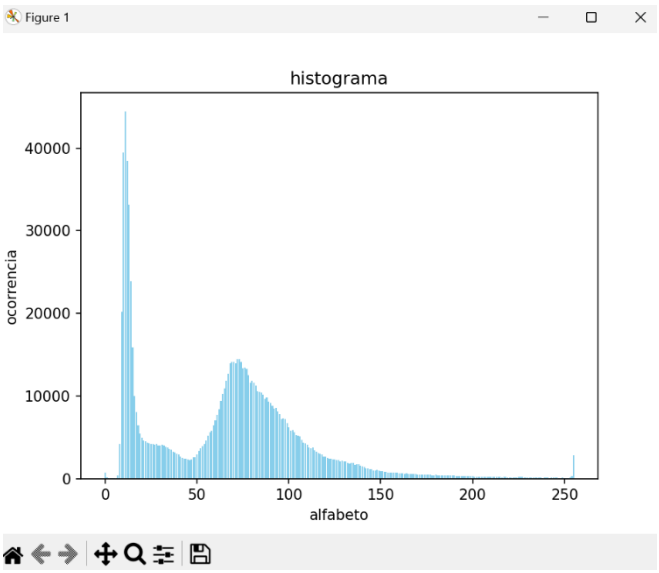
Gráficos:

Nota: Para todos os Histogramas produzidos foi importada a função cria_graficos de function.py, onde os parâmetros fonte e ocorrências são a primeira e segunda posição, respetivamente, da lista retornada pelas funções de leitura

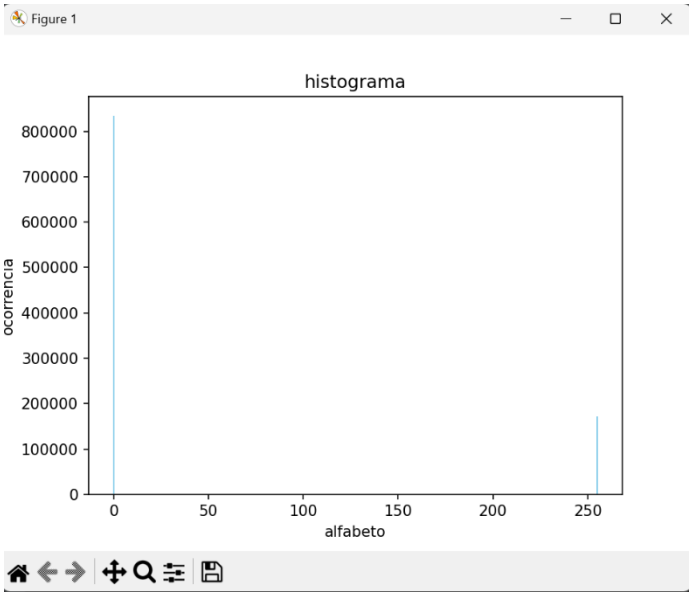
1. Landscape.bmp:



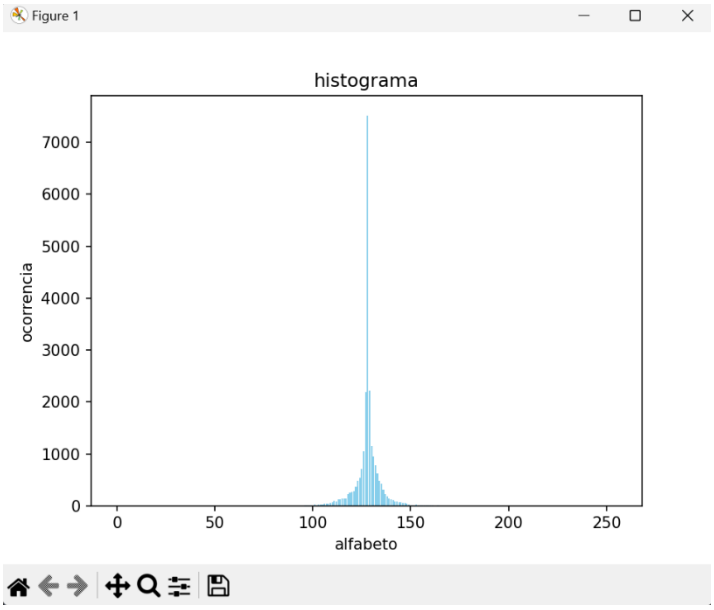
2. Mri.bmp:



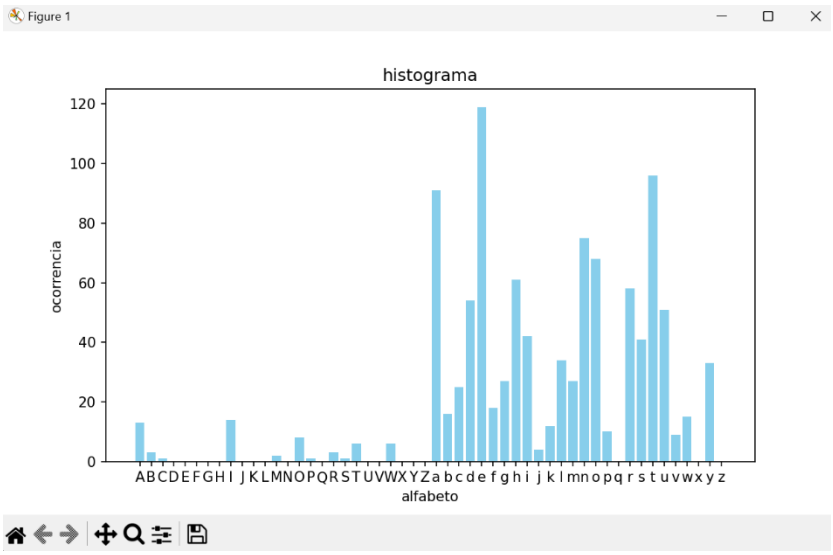
1. MRIbin.bmp



2. soundMono.wav



3. Lyrics.txt



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 usada para armazenar informação em menos espaço, contudo para ela ser bem efetuada não podem existir perdas de informação

A taxa de compressão máxima é dada por:

*TaxaCompressaoMax=((EntropiaMax-Entropia)/EntropiaMax)*100*

A entropia máxima é dada por:

EntropiaMax=log2(N), onde N corresponde ao numero de elementos do alfabeto

Caso estejamos a lidar com ficheiros de áudio ou som a Entropiamax será log2(256), caso seja ficheiros de texto a Entropiamax será log2(52)

Assim sendo temos:

Fonte	Entropia	EntropiaMax	TaxaCompressaoMax (%)
Landscape.bmp	7.606914077203874	8.0	4.91
Mri.bmp	6.860541984796121	8.0	14.24
Mribin.bmp	0.6610803299918833	8.0	91.74
soundMono.wav	4.065729167841344	8.0	49.17
Lyrics.txt	4.410705224964439	5.7	22.62

Verificamos assim que é possível comprimir todas as fontes de forma não destrutiva

EX4

Bibliotecas:

- Math
- Huffmancodec
- Scipy.io
- PIL

Funções:

calcular_n_bits_por_simbolo:

Neste exercício era nos pedido para calcular o número medio de bits por símbolo para todas as fontes de informação, assim sendo e com a ajuda do huffmancodec.py e da formula da media ponderada conseguimos calcular a mesma com sucesso

- calcular_variancia:

  - Passamos como argumentos o array com o comprimento respetivo de cada um dos símbolos da fonte e o dicionário de ocorrências da mesma
  - Em seguida calculamos o somatório das ocorrências
  - a variância ponderada é mais exata porque leva em consideração o peso de cada variável, ou seja, leva em consideração a frequência/ocorrência

Resultados:

Fonte	Entropia	Nº medio de bits por símbolo	Variância
Landscape.bmp	7.606914077203874	7.629300712836211	0.7516160479312144
Mri.bmp	6.860541984796121	6.8909959289047755	2.1930808768576195
Mribin.bmp	0.6610803299918833	1.0	0.0
soundMono.wav	4.065729167841344	4.110713829651388	4.3549582143224255
Lyrics.txt	4.480694531865603	4.52014652014652	1.736773604905473

Há uma grande semelhança entre os valores obtidos na entropia e no numero médio de bits por símbolo, no entanto a variância indica o quão diferentes são os valores necessários para codificar cada símbolo e o numero de bits necessários para codificar os símbolos da fontes. Assim sendo podemos verificar que o Mribin.bmp esta codificado da melhor forma possível e o Landscape.bmp esta com uma codificação quase perfeita ao contrário das outras fontes utilizadas neste exercício que podiam estar melhor codificadas

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

R: Para conseguir reduzir a variância bastava ordenar as probabilidades de um símbolo ocorrer de forma decrescente, contudo neste trabalho não fizemos esta organização justificando assim o porque da nossa variância não ser ótima.

Reduzir a variância pode ser vantajoso para contornar o congestionamento de rede, para o fazer recorremos aos buffers contudo é necessário que a taxa de enchimento seja constante assim sendo o numero medio de bits deve ser aproximadamente constante.

Ex5

Bibliotecas:

- numpy as np
- Scipy.io
- PIL

Funções:

criar_alfabeto_contiguo:

- Dados dois alfabetos, vamos criar um alfabeto novo que corresponde a todas as combinações do primeiro com o segundo

Resultados:

Fonte	Entropia	Entropia agrupada
Landscape.bmp	7.606914077203874	6.20405969097203
Mri.bmp	6.860541984796121	5.193602978171974
Mribin.bmp	0.6610803299918833	0.428703654483534
soundMono.wav	4.065729167841344	3.3109957723669114
Lyrics.txt	4.480694531865603	5.560978102335898

Era expectável os resultados da entropia agrupada serem menores do que a entropia calculada anteriormente. Os resultados tendem a melhorar quando se agrupam dois símbolos contíguos, contudo este não é o melhor método para reduzir o numero de bits pois o alfabeto com este método aumenta exponencialmente

Ex6:

Bibliotecas:

- numpy as np
- Scipy.io
- matplotlib

Ex 6a:

Funções:

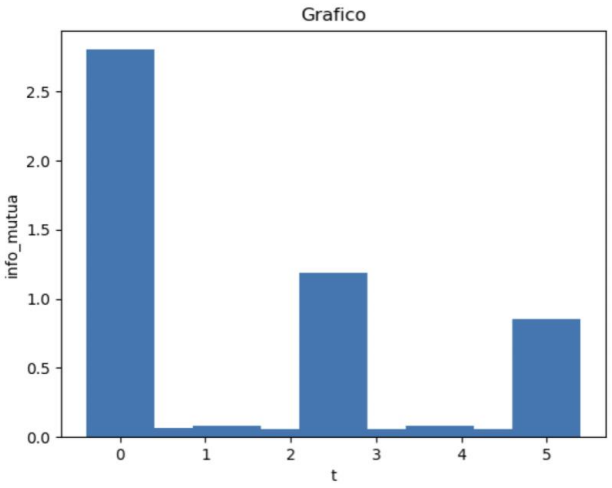
calcular_info_mutua:

- A formula para o calculo da informação mutua é dada pela somas das entropias da query e da janela subtraindo a entropia da query, janela. Nesta função vamos percorrer a query passo a passo ao longo do target criando janelas comparando-as com a query.

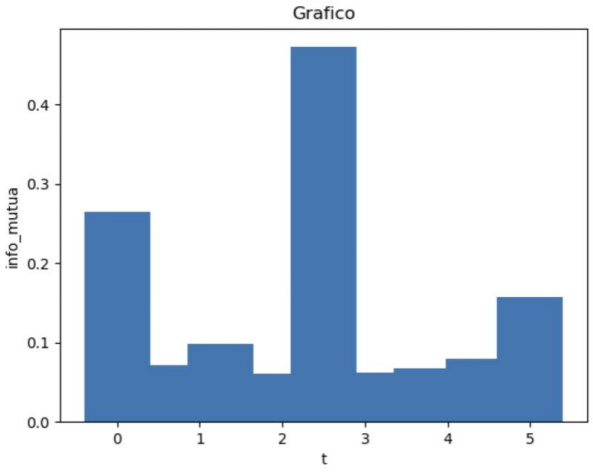
Ex 6b:

Neste exercício é nos pedido para usarmos as funções criadas em a para calcular a informação mutua dos dois targets diferentes (target01-repeat;target02 - repeatNoise) com a mesma query(saxriff.wav)

Target01-repeat:



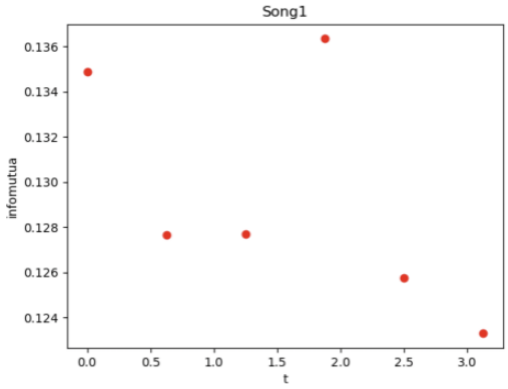
target02 – repeatNoise:

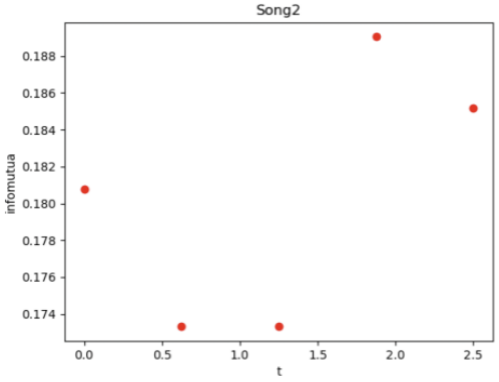


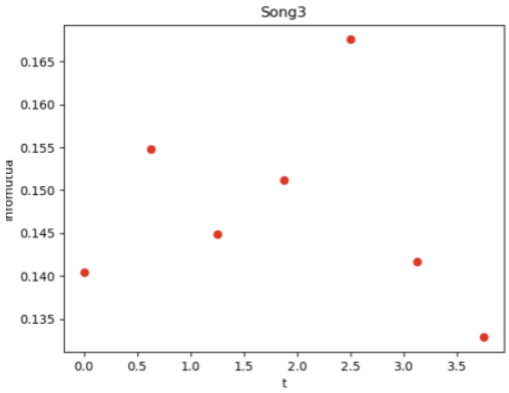
Após observar estes resultados chegamos a conclusão que o target01-repeat.wav tem muito mais em comum com o ficheiro saxriff.wav que o target02- repeatNoise.wav uma vez que os valores da informação mutua são maiores

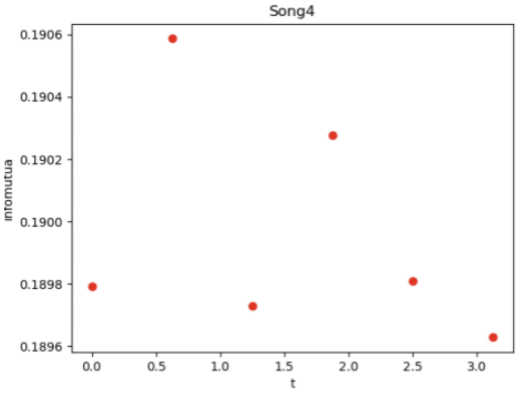
Ex 6c:

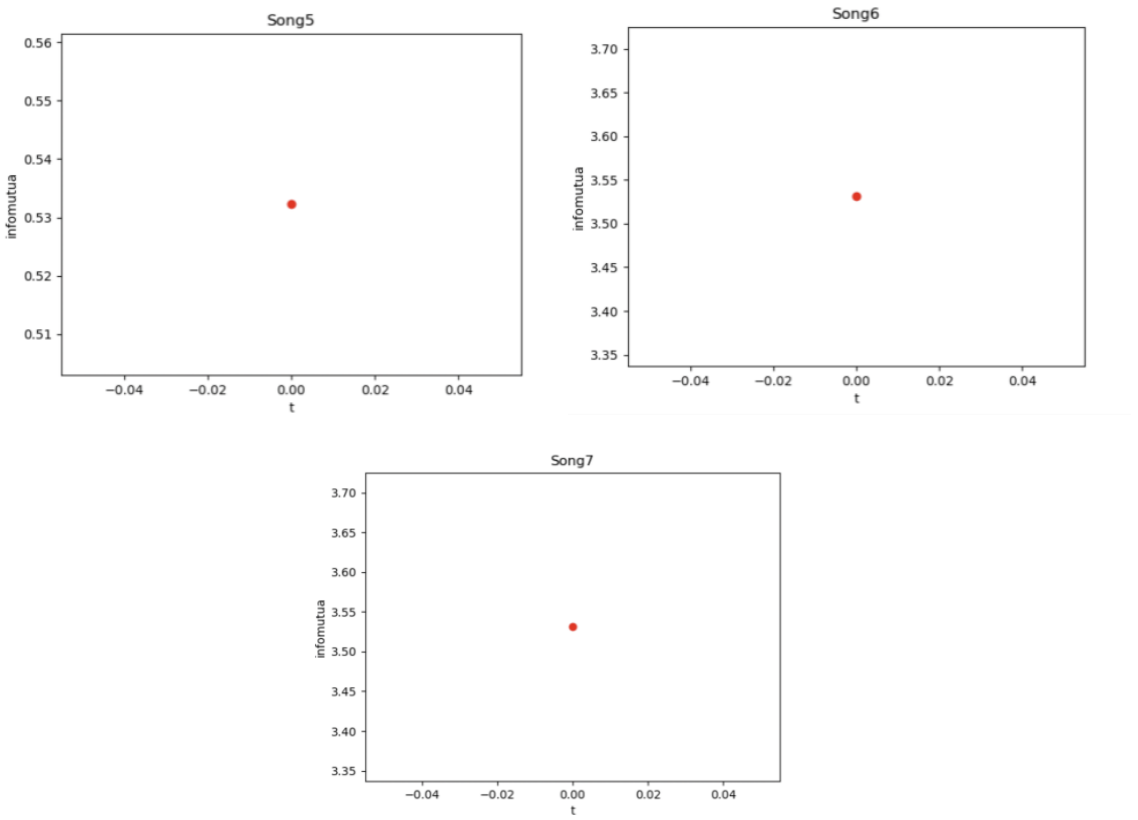
Neste exercício é nos pedido para analisar a informação mutua dos ficheiros: Song01.wav, Song02.wav, Song03.wav, Song04.wav, Song05.wav, Song06.wav, Song07.wav com a query usada no exercício anterior











Atraves da analise dos graficos podemos concluir que ao longo do tempo existem variações da informação mútua nas 4 primeiros ficheiros analisados.Nos 3 seguintes isso já não se verifica pois estes ficheiros tem um duraçao menor (semelhante a da query), assim sendo não vemos oscilações nestes graficos.

Em seguida podemos observar a informaçao mutua maxima de cada ficheiro:

Fonte	Informaçao mutua maxima
Song01.wav	0.1363424616299831
Song02.wav	0.18903790836002266
Song03.wav	0.16756787439474863
Song04.wav	0.19058719990804285
Song05.wav	0.5322424061045901
Song06.wav	3.530988733765132
Song07.wav	3.5309887337651333

Após analisar os resultados podemos verificar que a Song06.wav e a Song07.wav são as mais semelhantes a query o que era previsível uma vez que são as mais semelhantes sonoramente a query.

Conclusão

Este projeto foi bastante vantajoso uma vez que não só melhorou a nossa forma de trabalhar em equipa como também nos fez evoluir na linguagem Python.

Como exemplo da nossa evolução ao longo do nosso trabalho apresentamos de seguida alguns tópicos dos nossos melhoramentos:

1. Utilização de numpy ao invés de python math, tornando o código mais rápido e eficiente
2. No alfabeto Contiguo passámos a aplicar um separador em str(elemento)+ “separador” + str(elemento), ao fazer .append no alfabeto Contiguo
3. Alteração da formula da variância. Em vez de usarmos a variância normal, passámos a usar a variância ponderada, ou seja, passámos a levar em consideração a ocorrência dos símbolos/comprimentos.