Teoria da Informação

Trabalho prático nº1
Entropia, Redundância e Informação Mútua

Professor: Marco Simões

Alunos: Simão Correia Santos
Francisco José Coelho

Simão Correia Santos 2021216084
Francisco José Coelho 2021226534

Introdução

Este trabalho tem como objetivo a resolução de exercícios para aplicar os conceitos de informação, entropia, redundância e informação mútua. Deste modo, os exercícios propostos permitem-nos aplicar o material teórico numa situação prática e fazer uma análise e comentários face aos resultados obtidos.

Simão Correia Santos 2021216084
Francisco José Coelho 2021226534

Exercícios 1-3

O objetivo principal dos 3 primeiros exercícios é criar algumas funções que nos permitam obter o número de ocorrências de cada símbolo de um alfabeto numa determinada fonte (texto, som ou imagem) e visualizar graficamente os resultados obtidos (exercício 1), como também calcular a entropia associada à fonte (exercício 2), por fim aplicamos o código desenvolvido nestas alíneas a situações "reais" [MRI.bmp, MRIbin.bmp, landscape.bmp, soundMono.wav, lyrics.txt] (exercício 3).

Funções

'*getInfo*'

De forma a obtermos uma fonte e um alfabeto, através de uma imagem, som ou texto usamos a função *getInfo,* no caso do texto o alfabeto começa em ord('A') até ord('Z') reunido com ord('a') até ord('z'), para a fonte iteramos sobre a cada letra e convertemos para o respetivo valor da tabela ASCII, caso seja um símbolo regular do alfabeto será adicionado à fonte; no caso da imagem e do som criamos um alfabeto de 0 a 255, pois ambas são codificadas em 8 bits, para obtermos a fonte no caso do som usamos wavfile.read() da biblioteca scypi.io e no caso da imagem usamos mpimg.imread() da biblioteca matplotlib.image.

'getOcorrencia'

Para determinar o número de ocorrências de uma fonte num determinado alfabeto usamos a função *getOcorrencia* que recebe como parâmetro a fonte e o alfabeto, ao percorrermos o alfabeto usamos o np.where da biblioteca numpy que nos devolve num tuplo as posições onde cada um dos símbolos aparece na fonte, logo o comprimento do tuplo será o número de ocorrências.

Simão Correia Santos 2021216084
Francisco José Coelho 2021226534

'histograma'

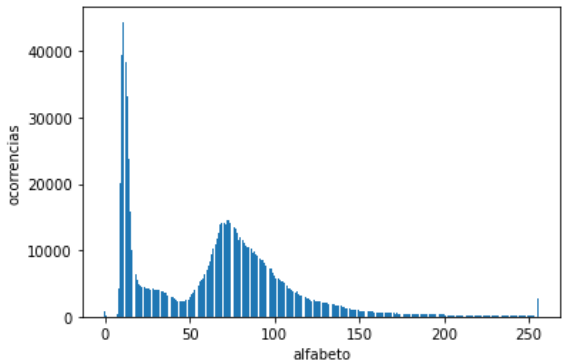
Através da função *histograma,* que recebe como parâmetros o array de ocorrências e o alfabeto, visualizamos graficamente o número de ocorrências (eixo yy) em ordem ao alfabeto (eixo xx), com auxilio do plt.bar() e plt.show() da biblioteca matplotlib.pyplot.

'*entropia*'

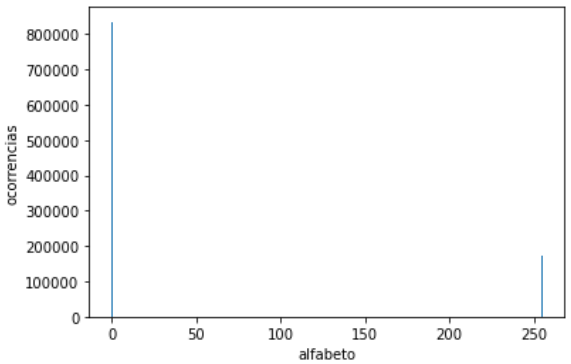
Função usada para calcular o limite mínimo para o número médio de bits por símbolo, por outras palavras, a entropia, que recebe como parâmetro o array de ocorrências e a fonte de informação. Através da fórmula da entropia [Σ P(x) * log2(1/P(x))] obtemos o valor da entropia.

Resultados

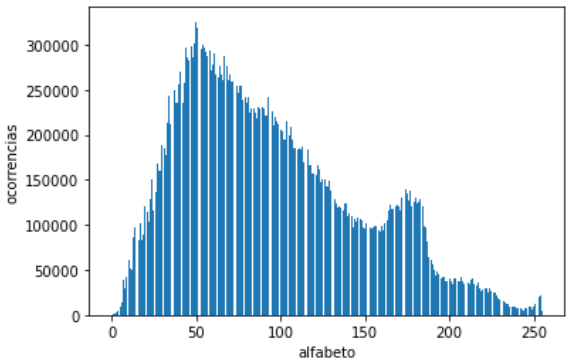
'MRI.bmp'



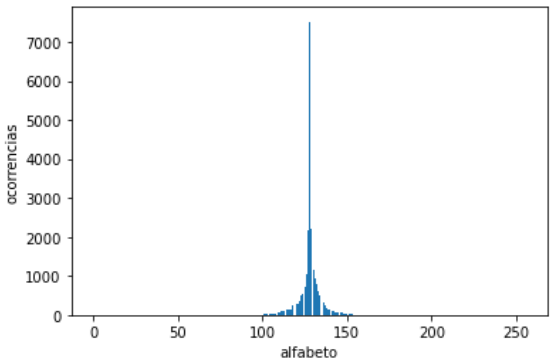
'MRIbin.bmp'



'landscape.bmp'

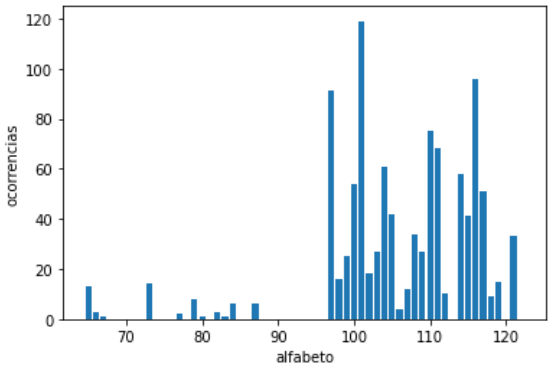


'soundMono.wav'



Simão Correia Santos 2021216084
Francisco José Coelho 2021226534

'lyrics.txt'



Ficheiro	Entropia	log2(|Ax|)
MRI.bmp	6.860541984796121	8
MRIbin.bmp	0.6610803299918833	1
landscape.bmp	7.606914077203874	8
soundMono.wav	4.065729167841344	8
lyrics.txt	4.41070522496444	5.70

⇒ |Ax| é o módulo do alfabeto!

Como nos é sabido, **a entropia é uma medida de dispersão da distribuição,** quer isto dizer que ao visualizarmos graficamente as ocorrências em ordem à fonte, quanto mais disperso for o nosso histograma maior será a entropia dessa fonte. Analisemos agora cada caso do nosso trabalho:

⇒ MRI.bmp: ao analisarmos o gráfico vemos que apresenta uma dispersão relevante, menor quando comparado com landscape.bmp e maior em relação a sounMono.wav (comparamos estes 3 pois ambos são codificados a 8 bits), logo é de esperar uma entropia que esteja entre estas duas;

⇒ MRIbin.bmp: esta imagem é binária quer isto dizer que pode ser codificada apenas com 1 bit, cada pixel da imagem é 0 ou 1, como no histograma o número de '0' é maior que '1' é de

esperar uma entropia menor que 1 (se estes fossem iguais, a entropia seria 1, pela seguinte propriedade: H(X) = log2|Ax|, se P(xi) = 1/|Ax|);

⇒ landscape.bmp: esta é a imagem que apresenta a maior dispersão, consequentemente também será a que apresentará maior entropia;

⇒ soundMono.wav: ao visualizar o gráfico é de esperar que este seja o que tenha menor entropia quando comparado com as fontes que são também codificadas a 8 bits;

⇒ lyrics.txt : este é o caso mais atípico apresentado, |Ax| = 52, (26 letras maiúsculas e minúsculas), logo log2(|Ax|) = 5.7, pelo histograma apresenta grande dispersão logo a entropia será elevada;

Por fim podemos concluir que é possível comprimir cada uma das fontes de forma não destrutiva, umas mais do que outras, sendo a compressão máxima que se consegue alcançar em cada uma das fontes a sua entropia.

Exercício 4

O objetivo deste exercício é criarmos algumas funções para obtermos o número médio de bits por símbolo para cada uma das fontes, ou seja o comprimento dos códigos de Huffman, como também a sua variância.

Funções

'mediaHuffman'

Função que recebe como parâmetro a fonte e o array de ocorrências e com auxílio das funções de codificação de Huffman fornecidas permite calcular a média ponderada de bits por símbolo através da sua fórmula [Σ((array de ocorrencias > 0)/len(fonte)) * comprimento];

'varianciaHuffman'

O cálculo da variância dos comprimentos dos códigos de Huffman é calculado neste função que recebe como parâmetro a fonte, a média de comprimentos, array com os comprimentos e o array de ocorrências através da fórmula da variância, neste caso ponderada [∑((array ocorrências > 0)/len(fonte)) * (xi - xmédio)];

Resultados

Ficheiro	Média	Variância
MRI.bmp	6.890995928904777	2.1930808768576195
MRIbin.bmp	1.0	0
landscape.bmp	7.629300712836217	0.7516160479312144
soundMono.wav	4.110713829651386	4.3549582143224255
lyrics.txt	4.443486590038314	1.0820552766400962

Numa primeira análise superficial, podemos concluir que os nossos resultados estão de acordo com o esperado, pois verificam a seguinte propriedade: $H(X) \leq Lmédio \leq H(X) + 1$

Analisemos agora cada caso:

⇒ MRI.bmp: para esta imagem a média dos comprimentos dos códigos de Huffman é praticamente igual à entropia, quanto à variância esta imagem tem ainda um valor considerável o que era previsível, o que era previsível pelo histograma onde vemos uma incidência elevada nos primeiros valores;

⇒ MRIbin.bmp: o valor esperado para uma imagem binária da média de comprimento é 1, e da variância é 0, ou seja, estando comprimido ou não cada símbolo ocupa 1 bit (o algoritmo das árvores de Huffman está sempre limitado ao mínimo de 1 bit por símbolo);

⇒ landscape.bmp: esta imagem é a que tem maior entropia, então seria de esperar que também tivesse maior média de comprimento e uma variância baixa, ou seja, não obtemos um aproveitamento muito elevado quando usamos os códigos de Huffman;

⇒ soundMono.wav: quando comparado com as duas imagens codificadas a 8 bits, a média de bits está também muito próxima da entropia, no entanto este é o que apresenta maior variância, o que era de esperar pela análise do histograma, onde há um pico acentuado no centro;

⇒ lyrics.txt: o resultado do comprimento médio de bits, à semelhança do som e das 2 imagens não binárias aproxima-se também à entropia e apresenta uma variância perto de 1;

Por fim podemos concluir que é possível reduzir a variância dos códigos de Huffman, para isso, tentamos concentrar os comprimentos dos símbolos perto do seu valor médio, ou seja, um símbolo que tenha um comprimento grande pode ser diminuído se um símbolo que tenha um comprimento pequeno for aumentado, mas para que isto aconteça nem a entropia nem o comprimento médio de bits pode ser alterado. Este processo pode ser útil no caso de uso de buffers, caso haja um congestionamento de rede aplica-se um buffer e é conveniente que a taxa de enchimento seja constante, ou seja, variância mínima.

Exercício 5

Este exercício tem como objetivo repetir o exercício 3, mas desta vez aplicando o agrupamento de símbolos, ou seja, cada símbolo é na verdade uma sequência de dois símbolos contíguos.

Funções

'getEntropiaAgrupado'

Esta função recebe como parâmetros uma fonte, um alfabeto e o nome do ficheiro, caso o ficheiro seja um ficheiro de texto criamos uma matriz de 122 por 122 (122 ==> ord(z)), caso contrário criamos uma matriz de len(alfabeto) por len(alfabeto), que funciona como o nosso alfabeto agrupado , ou seja, a posição (1, 2) corresponde à sequência 1 2. Caso o comprimento da fonte seja ímpar ignoramos o último símbolo, para que assim seja possível

razer agrupamentos 2 a 2. Por fim iteramos sobre a fonte de 2 em 2 e incrementamos na matriz criada inicialmente em 1, a posição correspondente aos elementos i, i + 1;

Ficheiro	**Entropia Agrupada**
MRI.bmp	5.693602970171973
MRIbin.bmp	0.9028703654483534
landscape.bmp	6.704059693097203
soundMono.wav	3.8108642
lyrics.txt	4.1521797

O uso de agrupamento de 2 símbolos contínuos permite em geral reduzir a entropia da fonte associada, pois este procura o reconhecimento de padrões na fonte o que faz com que a entropia diminui, apesar de o alfabeto aumentar exponencialmente. Analisando agora os resultados obtidos, podemos concluir de uma forma geral que o uso deste algoritmo permitiu diminuir a entropia de cada fonte à exceção da imagem binária, pois como já referido anteriormente o comprimento mínimo de um símbolo na árvore de Huffman é 1.

Exercício 6

Este exercício tem como objetivo criar algumas funções para procurar uma onda sonora conhecida num sinal genérico. Para esse efeito temos um sinal a pesquisar (query) e um sinal onde pesquisar (target), iremos "deslizar" a query pelo target usando uma janela deslizante com um determinado passo e assim iremos calcular a informação mútua.

Funções

`targetQueryAgrupado´

Função que recebe como parâmetro a query, o target e o início (o início será o primeiro elemento do target a ser agrupado) e retorna um array com todos os elementos da query e do target agrupados. Para agrupar a query e o target iremos percorrê-los, consoante o tamanho da query, e assim agrupá-los elemento a elemento.

`entropyConjunta´

Função que recebe como parâmetro a query, o target e o alfabeto e retorna a entropia conjunta da query e do target. Para calcular a entropia conjunta criamos uma matriz quadrada do tamanho do alfabeto pois os valores da query e do target variam entre 0 e 255 e de seguida é percorrida a query e o target ao mesmo tempo e na matriz incrementa-se 1 na posição query[i]target[i]. Após isto são calculadas as probabilidades e de seguida é calculada a entropia conjunta,

`getInformacaoMutua´

Função que recebe como parâmetro a query, o target, o alfabeto e o array com a query e o target agrupados e retorna o valor da informação mútua. Para esse efeito foi calculada a entropia da query e do target e a entropia conjunta da query e do target com auxílio das funções anteriormente descritas, após ter os valores das entropias foi calculado o valor da informação mútua fazendo a soma da entropia da query e do target subtraindo a entropia conjunta [I(X,Y) = E(X) + E(Y) - E(X,Y)].

`percorreTarget´

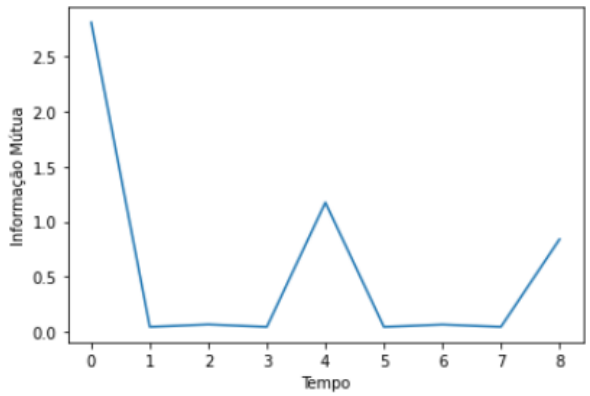
Função que recebe como parâmetro o passo, o target, a query e o alfabeto e retorna um array com os valores todos da informação mútua e um array com os valores para criar um gráfico com a informação mútua. Para isso foram inicializadas 2 variáveis, uma a zero denominada de início, cuja sua função é dizer onde começa o agrupamento no target com a query, e outra denominada de fim com o tamanho da query cuja função é dizer o onde acaba o agrupamento no target com a query (a ambas as variáveis é incrementado o valor do passo para criar a janela deslizante da query sobre o target). Após isto cria-se o agrupamento da query e do target e adiciona-se a uma lista os valores da informação mútua, com auxílio das funções anteriormente descritas. Em cada ciclo é incrementado a variável tempo que será usado no eixoXX do gráfico de informação mútua.

‘*graficoInformacaoMutua*’

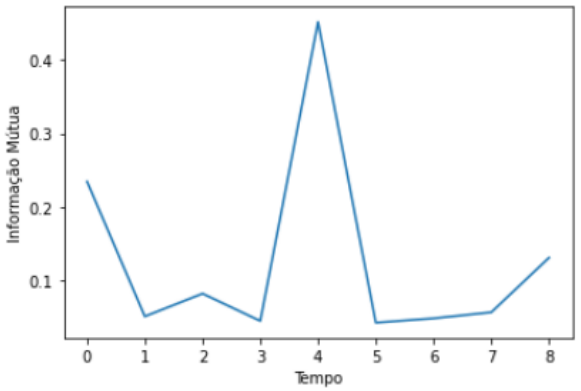
Função usada para visualizar graficamente a evolução da informação mútua em ordem ao tempo, recebe como parâmetro o array da informação mútua e o tempo.

Resultados

`target01 - repeat.wav´



`target02 - repeatNoise.wav´



Ficheiro	**Variação da Informação Mútua**
target01 - repeat.wav	[2.8077; 0.0453; 0.0672; 0.0451; 1.1739; 0.0450; 0.0662; 0.0455; 0.8396]
target02 - repeatNoise.wav	[0.2343; 0.0506; 0.0816; 0.0444; 0.4520; 0.0420; 0.0479; 0.0562; 0.1306]

O target01 e o target02 são representações do mesmo som, a única diferença é o ruído, ao analisar os gráficos podemos reparar que o gráfico que representa o target01 tem uma informação mútua maior, ou seja, tem um ruído menor, já o target02 tem um ruído maior entre os 0 a 3 e 5 a 8 unidades de tempo, já entre os 3 e 5 unidades de tempo vemos que há um pico na informação mútua, momento esse em que não há a presença de ruído tão forte.

Simão Correia Santos 2021216084
Francisco José Coelho 2021226534

Alínea c)

Neste alínea foi-nos pedido para comparar o som “saxriff.wav” com os 7 sons que nos eram fornecidos, usando o som “saxriff.wav” como query e os restantes sons como target, e desta comparação obtivemos as seguintes informações mútuas:

Ficheiro	Informação Mútua
Song 06.wav	3.5356
Song 07.wav	3.5356
Song 05.wav	0.5193
Song 04.wav	0.1165
Song 02.wav	0.1116
Song 03.wav	0.0958
Song 01.wav	0.0779

Ao analisar a tabela observamos que os valores do `Song06.wav´ e do `Song 07.wav´ são bastante elevados comparadamente aos restantes, deste modo podemos concluir que o som que mais se assemelha ao som da “saxriff.wav” é o som 6 e o som 7, pois estes são os que têm valores que mais se assemelham à query. Já os restantes sons têm uma informação mútua muito baixa, o que nos leva a concluir que os valores do seu respectivo target diferem muito dos da query.

Simão Correia Santos 2021216084
Francisco José Coelho 2021226534

Conclusão

Numa primeira análise superficial, podemos concluir que os resultados obtidos foram os esperados, tanto no cálculo da entropia, como no cálculo da entropia conjunta e também no cálculo das informações mútuas. Estes estão de acordo com os histogramas obtidos (no caso das entropias, quanto mais disperso o gráfico, maior a entropia e no caso das informações mútuas, quanto maior o seu valor maior a homogeneidade das fontes).

Por fim, a realização deste trabalho permitiu-nos consolidar melhor a matéria teórica e prática sobre conceitos de informação, entropia, redundância e informação mútua, para além disso, ainda nos ajudou a expandir o nosso conhecimento em relação ao python e às suas bibliotecas.

Simão Correia Santos 2021216084
Francisco José Coelho 2021226534