



UNIVERSIDADE D
COIMBRA

Faculdade de Ciências e Tecnologias

ANÁLISE E TRANSFORMAÇÃO DE DADOS - LICENCIATURA EM
ENGENHARIA INFORMÁTICA

IDENTIFICAÇÃO DE DÍGITOS ATRAVÉS DE CARACTERÍSTICAS EXTRAÍDAS DE SINAIS DE ÁUDIO

Trabalho Realizado por:
Nuno Batista e Francisco Lapa Silva

Março, 2024

Dados usados: 16

Meta 1.....	2
1/2 - Importação dos sinais de áudio e representações visuais.....	2
3 - Análise de características temporais.....	4
4 - Identificação das melhores características para discriminação de dígitos.....	6

META 1

1/2 - Importação dos sinais de áudio e representações visuais

De modo a importar os sinais de áudio com a função `audioread`, foi criada a função `getFeatures(exampleID, plt)`, que recebe como argumentos `exampleID`, que indica o índice das amostras a analisar e `plt`, que decide se os dígitos analisados devem ou não ser *plotted* (são *plotted* caso `plt == 1`). `getFeatures` é chamada, pela primeira vez, com o argumento `plt = 1`, ou seja, com o intuito de criar um *plot* de cada um dos sinais.

Dentro dessa função, um primeiro *for loop* itera pelos dígitos de 0 a 9, a cada iteração, a função `audioread` guarda os valores das amplitudes de cada um dos dígitos na variável `y` e o número de elementos de `y` em `rows`, deste modo, é possível concluir qual dos 10 sinais de áudio tem o maior número de amostras e guardar esse número na variável `maxRows` que, posteriormente, será usada para homogeneizar a duração dos sinais.

```
Ficheiro: getFeatures.mlx
% Find the duration of the Longest audio
maxRows = 0;
for i = 0:9
    [y, ~] = audioread(sprintf("samples/%d_16_%d.wav", i, exampleID));
    [rows, ~] = size(y);

    if(maxRows < rows)
        maxRows = rows;
    end
end
```

De seguida, inicia outro *for loop* que também itera por todos os dígitos com o intuito de fazer os *plots*, no entanto, para efeitos de melhor análise posterior, este *plot* exclui o silêncio inicial de cada um dos exemplos e adiciona silêncio ao final para todos os sinais terem a mesma duração. Este processo é feito através da análise da energia em janelas de tempo. A energia de cada janela é guardada no array `frameEnergy`, que é preenchido pela função `getFrameEnergy`, de acordo com a fórmula da energia.

$$E = \sum_{n=-\infty}^{\infty} |y[n]|^2$$

```
Ficheiro: getFrameEnergy.mlx
function [frameEnergy] = getFrameEnergy(y, frameSamples, numFrames)
% Calculate frame energy, the sum of squares of the samples in the frame
(not the integral because we are in the discrete domain)
    frameEnergy = zeros(numFrames, 1);
```

```

    % Iterate through every frame
    for j = 1:numFrames
        % The frame range is from the (j-1)th*frameSamples frame to
        the (j+1)th*frameSamples frame
        frame = y((j - 1)*frameSamples + 1:j*frameSamples);
        % Get the frame's energy
        frameEnergy(j) = sum(frame .^ 2);
    end
end
end

```

Basta, então, definir um *threshold* de energia e cortar todos as amostras antes da primeira que cumpre a restrição do *threshold* definido, bem como concatenar no final do sinal um vetor coluna cheio de elementos de valor 0 cujo número de elementos é igual à diferença entre o número de amostras do sinal de maior duração e o número de amostras do sinal atual (após o corte).

```

Ficheiro: getFetures.mlx
% frameEnergy is an array with the energy value of every frame
frameEnergy = getFrameEnergy(y, frameSamples, numFrames);

energyThreshold = 0.5;

% Find first index of the first frame with energy above threshold
startFrame = find(frameEnergy > energyThreshold, 1);
% Get the TIME index of the first frame with energy above threshold
startSample = (startFrame - 1) * frameSamples + 1;

% Trim y in order to remove low energy values
y = y(startSample:end);
% Add silence to the end of y
[curRows, ~] = size(y);
concatY = zeros(maxRows - curRows, 1);
y = [y; concatY + 0.5];

```

Finalmente, resta fazer o próprio *plot* do dígito na iteração atual. Obtendo assim o resultado final.

```

if plt == 1
    % Ts is the sampling period
    Ts = 1 / Fs;
    % t is the time vector
    t = (0:maxRows-1);
    % t is the time vector in seconds (starts at the first frame that
    exceeds the energy threshold)
    t = (t .* Ts);
    subplot(5, 2, i + 1);

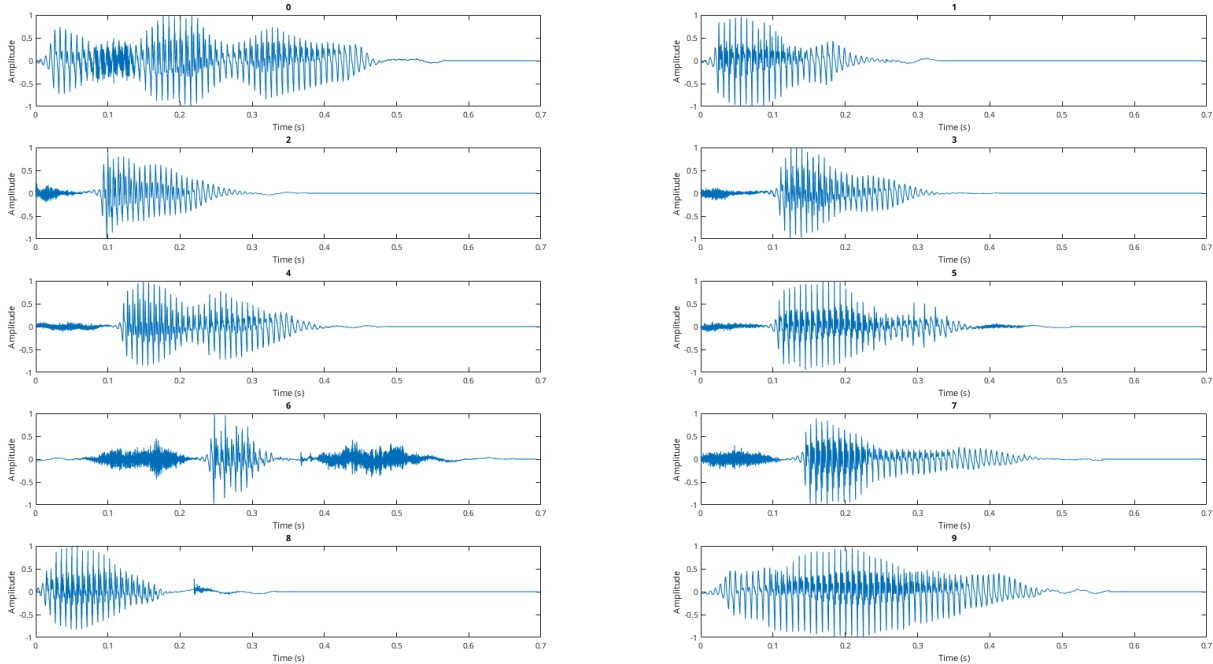
```

```

plot(t, y');
xlabel('Time (s)');
ylabel('Amplitude');
label = sprintf("%d", i);
title(label);
end

```

Signal examples



3 - Análise de características temporais

A função `getFeatures` devolve a matriz 10x5 `resultFeatures`, que contém os resultados das análises de 5 características diferentes de cada um dos 10 dígitos (`resultFeatures(i + 1, x)` é o valor da característica x do dígito i). As características escolhidas foram:

- 1 - Energia total
- 2 - Desvio Padrão
- 3 - Amplitude máxima
- 4 - Taxa de cruzamento por zero
- 5 - Duração em segundos

Para calcular a energia total, basta utilizar a fórmula anteriormente referida, mas desta vez, para todos os elementos de y.

```
resultFeatures(i + 1, 1) = sum(abs(y) .^ 2)
```

É de referir que no início da análise de cada dígito, a amplitude é normalizada de modo a todos os valores entrarem no raio $[-1, 1]$ tornando assim o valor do desvio padrão muito reduzido, portanto, o cálculo desta característica é realizado pela função `std` antes da normalização.

```
resultFeatures(i + 1, 2) = std(y);
```

O mesmo acontece com a amplitude máxima, que é dada pelo valor máximo de `y`.

```
resultFeatures(i + 1, 3) = max(y);
```

A taxa de cruzamento por zero é calculada pela função `zerocrossrate`, é importante ter em conta que é por causa da análise desta característica que foi tomada a decisão de fazer uma normalização no raio $[-1, 1]$ e não $[0, 1]$.

```
resultFeatures(i + 1, 4) = zerocrossrate(y);
```

Para calcular a duração, é necessário remover o silêncio no final do sinal, para isso, procura-se a última janela com um valor de energia superior ao do *threshold* definido e divide-se a diferença entre a posição da última e da primeira amostra que cumprem a restrição do *threshold* pela taxa de amostragem `Fs`.

```
Ficheiro: getFeatures.mlx
energyThreshold = 0.1;
% Find first index of the first frame with energy above threshold
startFrame = find(frameEnergy > energyThreshold, 1);
% Get the TIME index of the first frame with energy above threshold
startSample = (startFrame - 1) * frameSamples + 1;

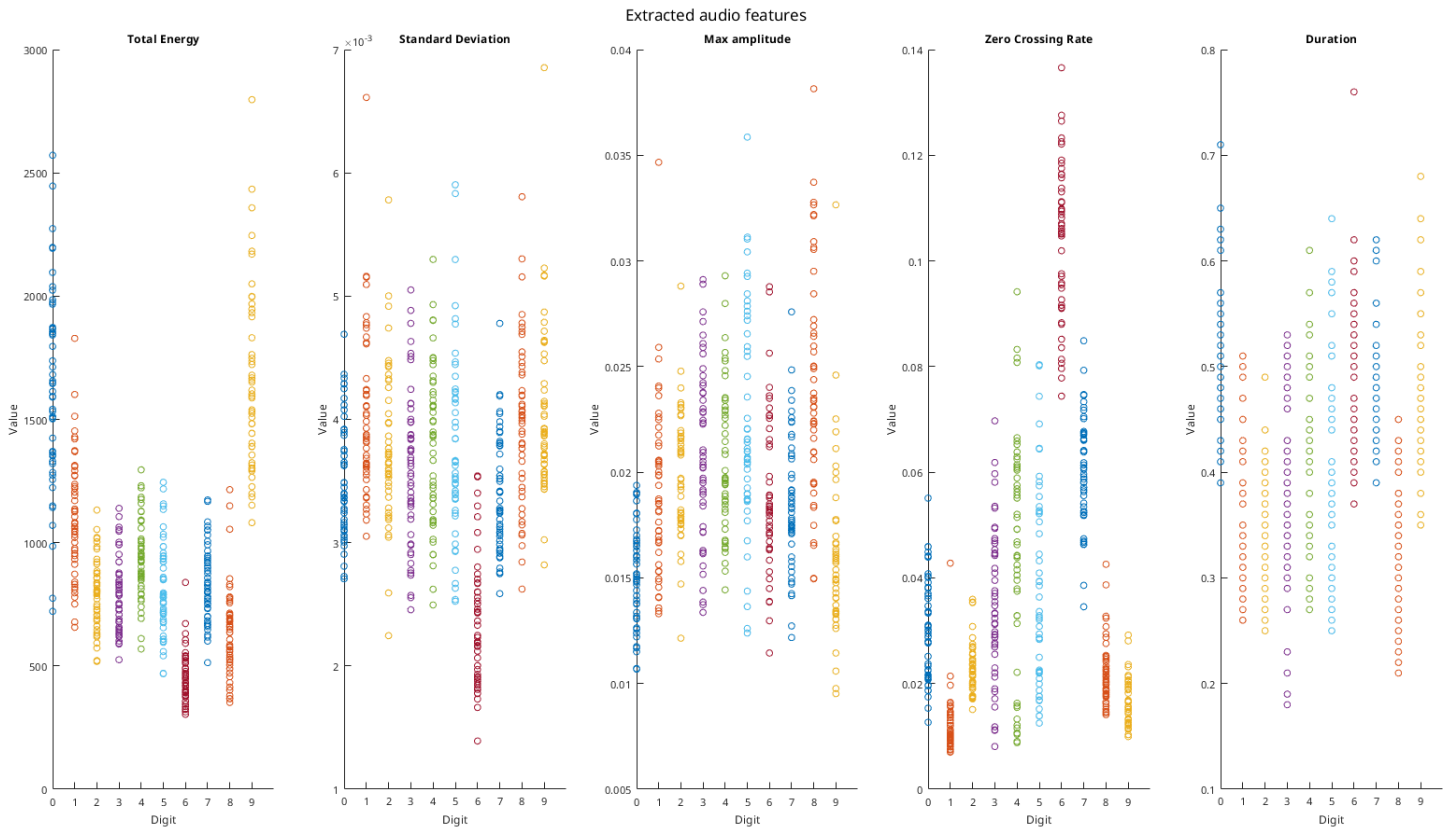
% Find the index of the Last frame with energy above threshold
endFrame = find(frameEnergy > energyThreshold, 1, 'last');
% Get the TIME index of the Last frame with energy above threshold
endSample = endFrame * frameSamples;

% Store the audio duration without silences in seconds
resultFeatures(i + 1, 5) = (endSample - startSample) / Fs;
```

Evidentemente, `resultFeatures` tem apenas as características de um conjunto de dígitos, no entanto, há 50 conjuntos, portanto, na função `main`, um *for loop* chama esta função com os IDs dos conjuntos restantes, ou seja, de 1 a 49 e guarda essas características na matriz 3D `audioFeatures`, onde `audioFeatures(x, y, z)` é o valor da característica `z` do dígito `y-1` do conjunto `x-1`.

```
% i + 1 because MATLAB uses 1-based indexing but the audio file names
use 0-based indexing
for i = 1:49
    audioFeatures(i+1, :, :) = getFeatures(i, 0);
end
```

Por último, resta apresentar os resultados visualmente, obtendo o seguinte resultado com *scatter plots* 2D.



4 - Identificação das melhores características para discriminação dos dígitos

Tendo em conta os resultados obtidos no ponto anterior, a energia total, a taxa de cruzamento por zero e a duração foram consideradas as três características cujos valores mais diferem entre dígitos, levando a uma melhor discriminação dos mesmos, para representar estas três características visualmente, usam-se *box plots* 2D, obtendo assim o seguinte resultado.

