Teoria Algorítmica da Informação

Armando J. Pinho
Diogo Pratas

Lab Work Nº 3
Automatic Identification of Musics

Pedro Marques, 92926
Rui Fernandes, 92952
Inês Leite, 92928

DETI
Universidade de Aveiro
January 31, 2022

1 Introdução

Através deste relatório, explicaremos o processo de criação do nosso programa de identificação de uma música ou ficheiro de áudio sobre uma determinada base de dados. De modo a que seja possível comparar os ficheiros de teste fornecidos com a base de dados, é necessário criar uma representação de todos os ficheiros de áudio. Assim sendo, desenvolvemos um programa que gera uma assinatura/*fingerprint* de cada ficheiro. Igualmente, utilizaremos a *Normalized Compression Distance* para calcular a semelhança, assim sendo recorreremos a vários compressores de modo a analisar também como o desempenho do nosso programa de identificação muda consoante o compressor selecionado.

2 Contexto

O nosso programa está composto no ficheiro *compare.py*. Este recebe um ficheiro *sample* de teste e retorna o nome da música que considera ser a correta. Por outro lado, a função *create_signatures* em *generateAudioSignatures.py* é responsável por chamar a função de geração de assinaturas que criámos em *getMaxFrequencies.py*.

Finalmente, o script *compressor.py* apresenta a função que, escolhendo um dos métodos de compressão, retorna dados comprimidos utilizando esse método.

3 Compare

O programa é iniciado pela função compare.py:

- **compare.py -s sample_name -c compression_method <-reset>**

O primeiro argumento refere-se à amostra que queremos descobrir a música, depois temos o tipo de compressão a usar, [gzip, bzip2, zlib]. O último, *-reset*, é opcional e indica a necessidade de recomeçar o programa, criando assinaturas para todos os áudios.

O objetivo deste programa, como já foi referido, é encontrar a música a que a amostra de aúdio pertence. Para o fazer existem várias funções importantes, que serão referidas de seguida.

3.1 Obter as assinaturas dos áudios

Com a função *getSignatureFromAudio()* do ficheiro *getMaxFrequencies.py* são criadas as assinaturas de todos os aúdios, tanto das músicas, como das amostras.

Esta função tem em atenção o facto de cada música ter dois canais de áudio, *stereo*. Assim sendo, começamos por converter o sinal em *mono*. A biblioteca *AudioSegment* permite então dividir o áudio *stereo* em dois segmentos *mono*. Para obter a música final apenas fazemos a média entre os valores dos dois segmentos *mono*.

De seguida, são criadas janelas de tamanho *f_rate* (1 segundo) a partir sinal *mono*. A cada uma destas partes será aplicada a Transformada Discreta de Fourier, para observarmos as frequências com maior amplitude. A frequência com amplitude máxima é guardada e será usada para criar a assinatura do ficheiro.

Após a obtenção das frequências com amplitude máxima para cada janela do sinal, obtemos um *array* cujos elementos são tuplos do tipo: (frequência, amplitude, i), no qual a variável i corresponde à posição da janela na música, ou seja, ao momento da música a que a frequência com amplitude apresentada pertence. Finalmente, este *array* é guardado num ficheiro sendo este a assinatura de cada áudio.

3.2 Normalized Compression Distance

A *normalized compression distance* permite analisar a similaridade entre diferentes objetos utilizando a compressão.

$$\mathrm{NCD}(x,y) = \frac{C(x,y) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}},$$

Nesta equação, *C(x)* representa o tamanho, em *bits*, que o *compressor* C necessita para representar x e *C(x,y)* referente a *x* e *y* juntos. Quanto mais próximo do 0, mais similares serão.

Para efetuar esta compressão utilizámos 3 compressores desenvolvidos em python: gzip, bzip2 e zlib. A função *compress* disponível em *compressor.py* recebe como argumentos um string de dados e um método de compressão da lista apresentada anteriormente. Posteriormente, retorna a versão comprimida deste string.

Assim sendo, após termos gerado/carregado a assinatura da *sample x* que queremos testar, carregamos/criamos a assinatura de cada música *y* da base de dados. Tendo agora estes dados, percorremos *y* utilizando uma janela deslizante de dimensão igual à de *x* e calculamos a NCD entre essa janela e *x*. A junção de *x* e *y* consiste na concatenação do primeiro string ao último. Tendo agora todas as NCD entre todas as janelas e *x*, a *normalized compression distance* final entre a música da base de dados e a *sample* é o valor mínimo calculado.

4 Resultados

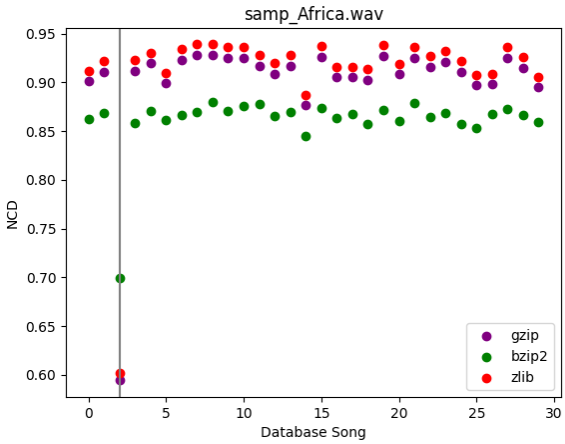
Para testagem do algoritmo, criámos ficheiros de áudio que correspondem a partes de músicas da base de dados. Igualmente, utilizando estes excertos, aplicámos ruído e e outras técnicas de transformação de sinal para ver o quão tolerável é o nosso algoritmo. Estes ficheiros encontram-se todos dentro da pasta *samples*.

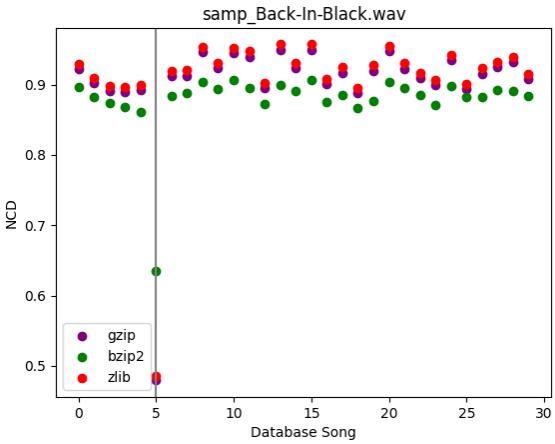
A nomenclatura destes ficheiros obedece a uma certa regra: **effect_audioname**

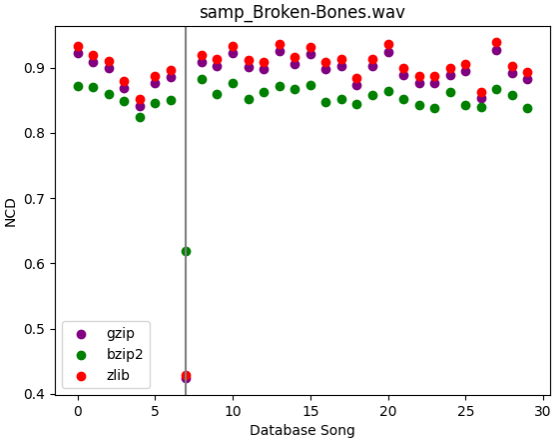
- "samp_": partição do ficheiro de áudio sem efeitos
- "br_noise_number": aplicação de *brown noise* com amplitude *number*
- "wh_noise_number": aplicação de *white noise* com amplitude *number*
- "bass_": *bass boosting* de 15dB
- "treb_": *treble boosting* de 15dB
- "freq_": aumento de 10% de *pitch* (frequência)
- "clip_": *distortion: hard cliping at* -6dB (transforma com as harmónicas altas)
- "saw_": adição de um ruído agudo crescente

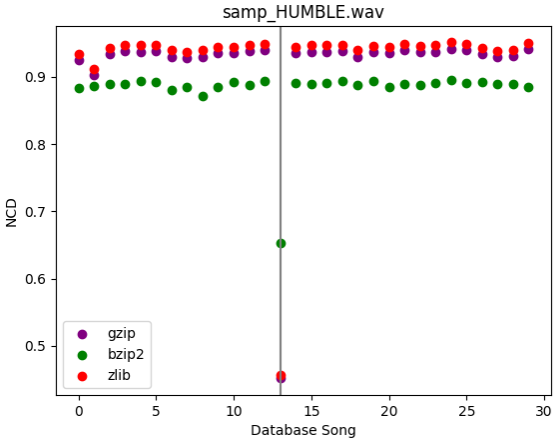
4.1 *Samples* Sem Efeitos Aplicados (*Raw*)

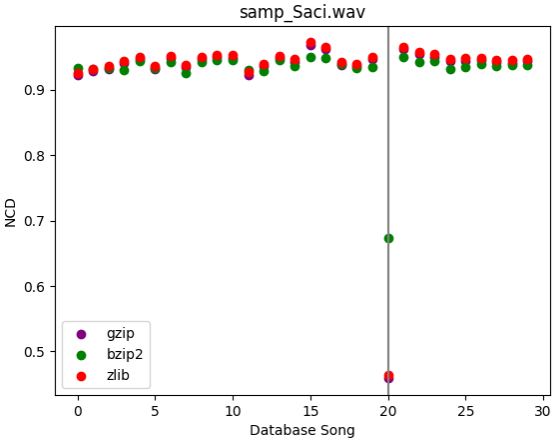
Como era de esperar, as comparações com amostras sem ruído eram as melhores, dando resultados muito bons. Assim sendo, apresentamos abaixo os resultados da *Normalized Compression Distance* para cada ficheiro de áudio com prefixo "samp_", ou seja, partições *raw* das músicas da base de dados (os títulos das figuras identificam as *samples*).

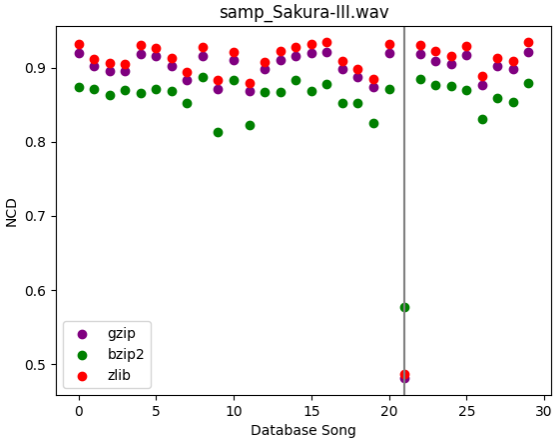












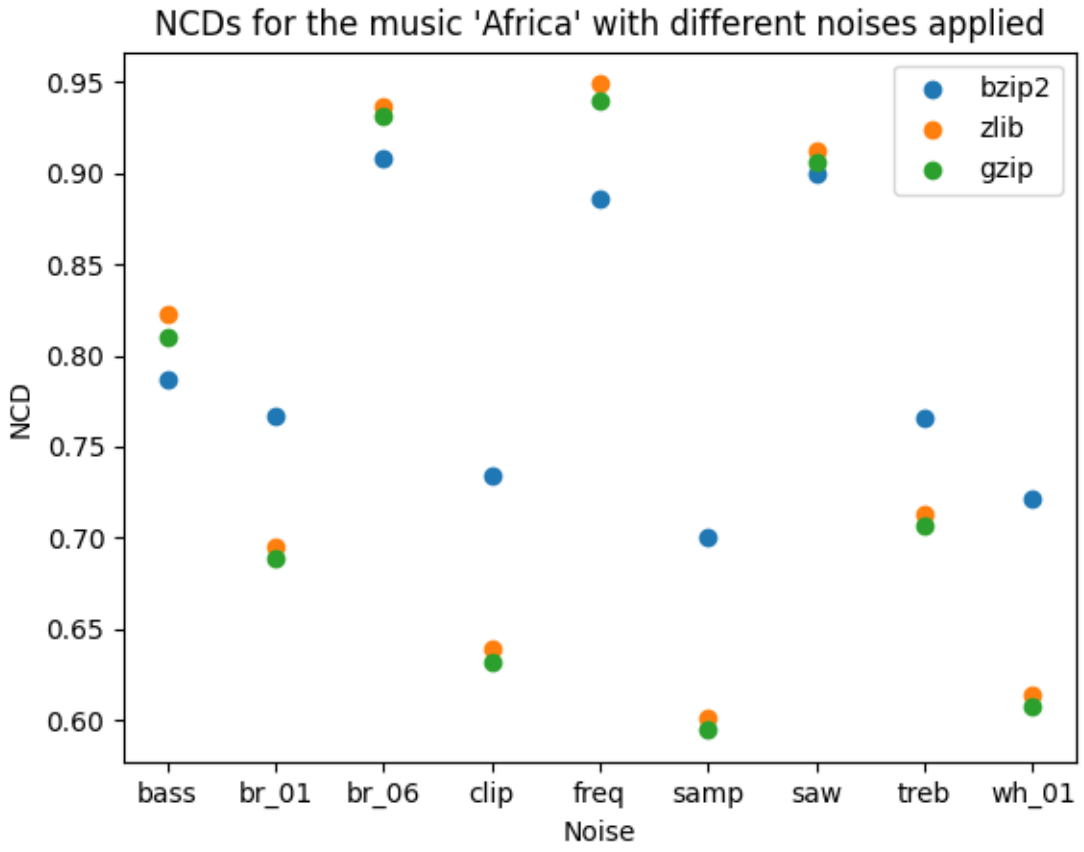
Nestas figuras, a linha vertical corresponde à posição da música correta no *x-axis*. Como podemos ver, não houve qualquer erro na previsão das músicas. Embora os resultados tenham sido diferentes, os algoritmos de compressão permitiram chegar a respostas corretas.

Podemos também observar que, enquanto os compressores *zlib* e *gzip* resultam quase na mesma *NCD distance*, o *bzip2* produz os piores resultados. Suspeitamos que isto seja porque o rácio de compressão deste último método é bastante superior aos restantes.

É também de notar que enquanto este foi o algoritmo mais lento, *zlib* foi bastante mais rápido que os restantes.

4.2 Comparação entre métodos de compressão

Para comparar os diferentes métodos de compressão com os diferentes efeitos aplicados fez-se um *script, tester.py*. Aqui, testámos todos os 9 tipos de distorções aplicadas na música 'Africa' juntamente com os 3 compressores disponíveis e guardámos a distância NCD resultante. Com todos estes valores gerámos o gráfico abaixo para visualização.



Como é possível verificar, usando zlib e gzip são obtidos valores NCD muito semelhantes e com bzip2 as distâncias revelam menor variação.

4.3 Observações interessantes

Um aspeto interessante que observamos ao testar o programa foi que para a versão da música 'Africa' que continha o efeito *brown noise* de amplitude 6 a previsão dada pelo programa era sempre errada. Ao ouvido humano esta amostra é bastante percetível, principalmente quando comparada com o *white noise*. No entanto, a amostra que tinha tido esta última distorção (*white noise*) aplicada obteve os melhores resultados, apenas superada pela amostra raw.

5 Conclusão

Em suma, o programa que desenvolvemos consegue, com alta confiança, averiguar qual a música que corresponde a um certo ficheiro de áudio dado. Conseguimos também garantir que consegue igualmente detetar com elevada confiança a música para ficheiros de áudio que sofreram alguma alteração, simulando, por exemplo, ruído que seria apanhado por um microfone.

Embora tenhamos alcançado este objetivo, tínhamos também outros em mente que pretendemos desenvolver numa futura iteração. Em termos de trabalho futuro, pretendemos desenvolver um método de aprendizagem, de modo a que o programa utilize os resultados dos ficheiros de áudio distorcidos que recebe para aumentar a confiança de previsão noutros ficheiros.