UNIVERSITY AVEIRO

Lab work nº 3

Algorithmic Information Theory

Pedro Silva (93011)
Miguel Almeida (93372)
João Soares (93078)

Department of Electronics, Telecommunications and Informatics

2022

Contents

Chapter 1

Introduction

A common real life use case that is often studied is how to identify a piece of media by taking into account a randomly sampled piece of said media, potentially with noise applied to it. For example, identifying a song by taking into account an excerpt with noise heard on the radio.

One way of doing this is by using the Normalized Compression Distance, an approximation of the information distance between two strings by way of compression, on a binary signature of various audio files.

In this work, we resort to this approach combined with a database of 25 songs and various samples of varying size with and without noise to develop and validate a program capable of identifying songs.

Chapter 2

Implementation

2.1 FindMusic

FindMusic is a class and main part of the program that allows the automatic identification of musics from a sample, using the NCD (Normalized Compression Distance).

The program receives as input the path to the sample audio file to analyze in *.wav* format, the type of compressor to use,the level of noise and respective the type to apply in the sample audio.

The class loads the database using the function *load_db()*, that compose of a dictionary of the music and the respective signature calculated beforehand and sets the compressor to use.

If indicated, the program applies noise to the sample audio file using the function *add_noise()* that will be explained later in the report.

The program can be divided in several functions and classes:

find - Finds the name of the music in the database more similar to the sample, that corresponds to the one with smallest NCD value.

The function starts by getting the most significant frequencies of the audio file sample, using the function *applyGetMaxFreqs()* from utils.py that will be explained further in the report. Then it computes the Normalized Compression Distance (NCD) between the sample audio file and each music in the database, returning the one with the smallest value, in other words, returns the music in the database more similar to the sample audio file.

In the end deletes all the temporary files generated during the process.

ncd - Computes the Normalized Compression Distance between the signature of two musics, following the formula:

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

Where C represents the number of bytes necessary to compress some signature using type of compressor that was passed as input to the program.

2.2 Compressors

To perform compression, were used the gzip, bzip2 and lzma compressors.

the compressors implemented in the program were gzip, bzip2 and lzma, *compressor.py* contains the implementation of each compressors, which contains the function *compress()* that computes the number of bytes to compress the data.

2.3 Music Signature

To get the signature (most significant frequencies) of an audio file, it was developed the function *applyGetMaxFreqs()* from *utils.py*.

The function receives as input the path of music file and then executes in a sub process *GetMaxFreqs*[1] module, available in the website of the course, that internally uses the Fast Fourier Transform to obtain the frequencies of the audio file. Finally, it returns the content of the file in which the "signature" was written.

2.4 Noise

The process to add noise to the samples is done in run-time using the function *add_noise()* from *utils.py*.

This function receives as input the audio sample, the type of noise (*whitenoise* or *brownnoise*) and the amount of that noise to add to the audio file, that must be between -0.4 and 0.4.

By deafault, the whitenoise will be used as type of noise, if this is not defined.

The function runs internally the SoX[2] program in a sub process, to generate the effect of noise and apply it to the audio sample. In the end, it returns the path to the audio sample file with noise added, that will be used later instead the original sample.

[1]Developed by Armando J. Pinho, ap@ua.pt, 2016-2020

[2]Cross-platform command line utility that can convert various formats of audio files into other formats and apply various effects to these sound files - http://sox.sourceforge.net/

Chapter 3

Results and Discussion

3.1 Music Configuration

The musics selected for the database (25) and samples were chosen taking into account a diversity in the genre of music to validate our solution in different types of music and also in different languages.

The samples were generated using the library SoX[1] and have sizes of 10, 20 and 30 seconds starting in random moments of the music to better analyse the accuracy and replicate a real world usage of the program.

3.2 Accuracy by Variation of Noise

For noise variation, considered samples with 30 seconds, and experimented varying noise level between -0.4 and 0.4 for both types of noise available in Sox, *brownoise* and *whitenoise*. For all results the compressor used was BZIP2.

As we can observe in Figure 3.1, the program was shown to be resilient to the use of white noise, maintaining a perfect accuracy for all noise levels tested it.

While for brown noise, as can be seen in Figure 3.2, results were less encouraging, with noise levels bigger in absolute value of 0.1 leading to sharp decreases in accuracy registered.

[1]Python Library : pysox[1] - https://pypi.org/project/sox/

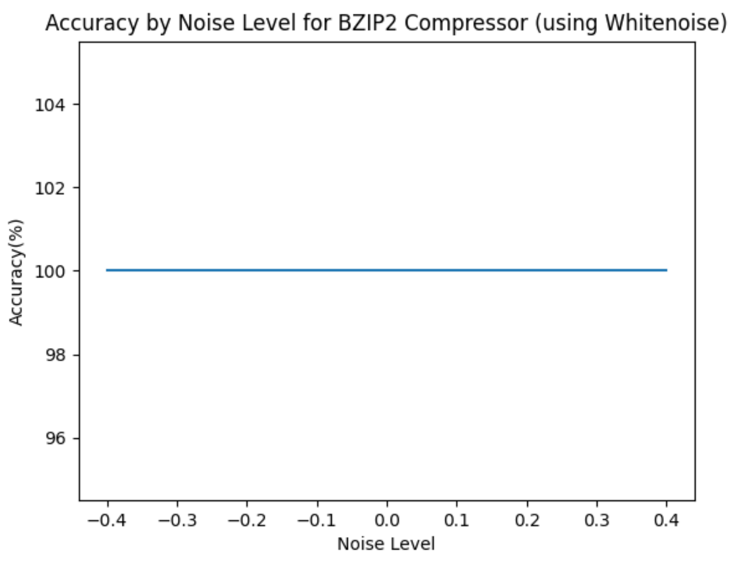

Figure 3.1: Accuracy by variation of noise for White noise

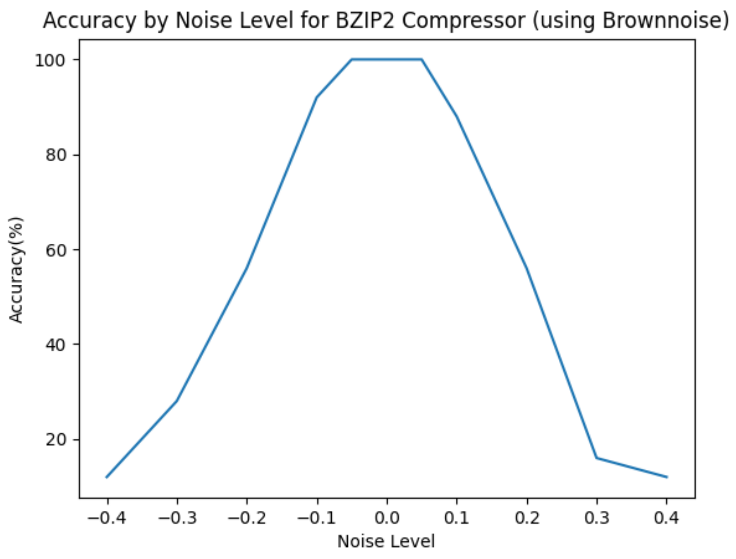

Figure 3.2: Accuracy by variation of noise for Brown noise

3.3 Accuracy by Variation of Sample Size

For sample size variation, we took into account samples with 10,20 and 30 seconds, and verified the accuracy for all types of compressors: LZMA, GZIP and BZIP2.

As can be seen in Figure 3.3, for the GZIP compressor, accuracy fluctuated between 92% and 96%, increasing for sample size of 30.

Meanwhile for the LZMA compressor we got a more mixed, but still very high on average accuracy result, as can be seen in Figure 3.4, as accuracy, while somewhat low for 84%, it increased to 92% for sample size of 20 and finally to a perfect 100% accuracy for sample size of 30.

Finally for the BZIP2, we can observe in Figure 3.5 a perfect score regardless of sample size.

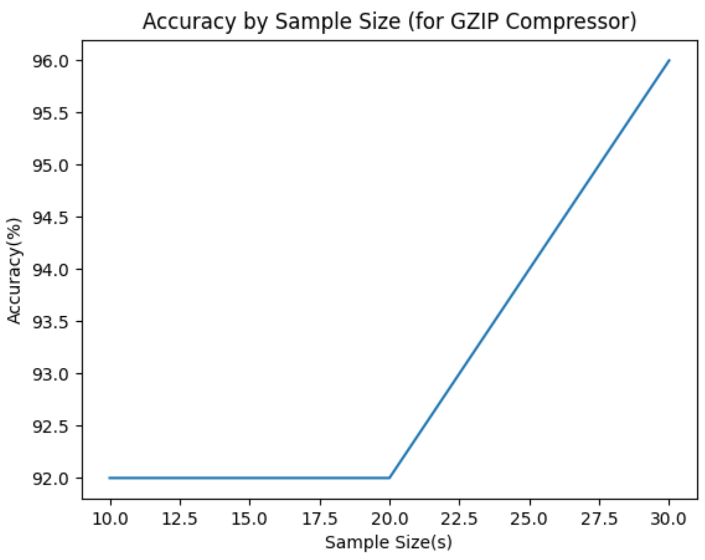

Figure 3.3: Accuracy by variation of Sample Size for GZIP Compressor

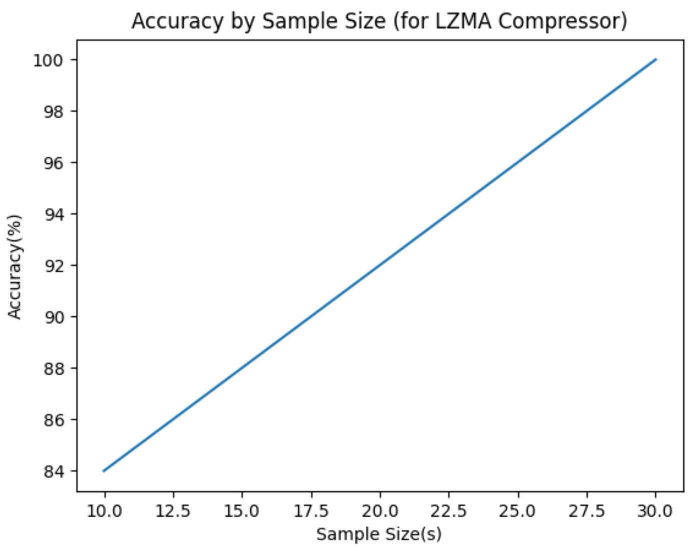

Figure 3.4: Accuracy by variation of Sample Size for LZMA Compressor

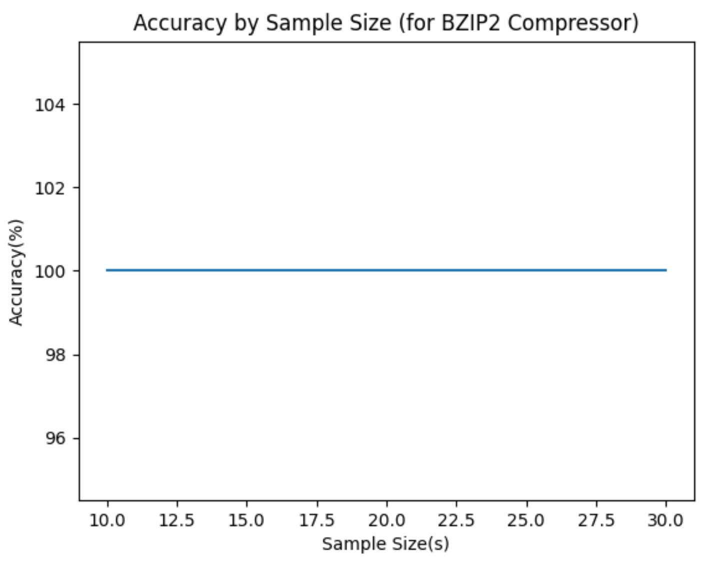

Figure 3.5: Accuracy by variation of Sample Size for BZIP2 Compressor

3.4 Accuracy by type of Compressor

For variation of type of compressor, we took into account only samples with 20 seconds and experimented with all compressors GZIP, LZMA and BZIP2.

As can be seen at Figure 3.6, BZIP2 obtained the best results, with a 100% accuracy mark on average when compared with the LZMA compressor who only got 88% and the GZIP compressors who also only got 88%.

While not occurring during experiments, it's worth mentioning that, depending on the compressor compression ratio, they might give values slightly above 1.

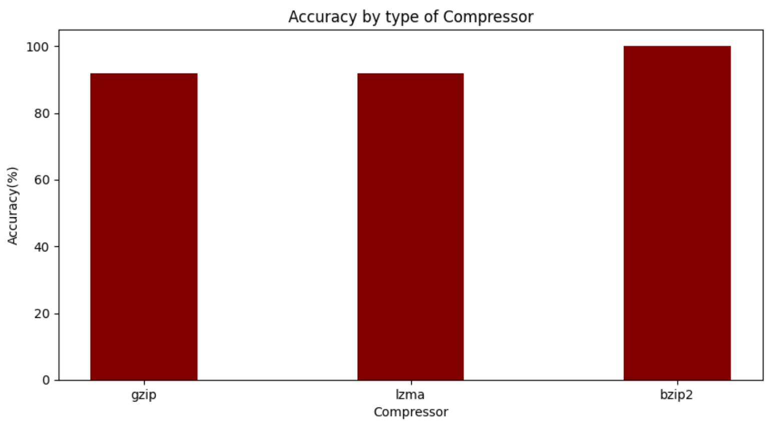

Figure 3.6: Accuracy of type of Compressor for sample size of 20 sec

Chapter 4

Conclusion

As we observed, by using the power of Normalized Compression Distance (NCD) it is possible to build a robust audio identification program capable of obtaining high levels of accuracy in identifying songs from small samples of audio.

By taking into account the results obtained, we were able to conclude that choice of compressor used is indeed important, because while all compressors gave high accuracy results on average, BZIP2 was able to get a perfect mark of 100% accuracy, we also observed the effects of the size of the sample used and notice how the increase in size led to better results on average.

Finally, by testing our program with both white noise and brown noise for various values we observed that while our program was shown to be resilient to all levels of white noise, it had difficulties when confronted with the effects of brown noise.

Appendix A

Percentage of participation each member of the group

- Pedro Silva (93011) 33.3%
- Miguel Almeida (93372) 33.3%
- João Soares (93078) 33.3%

References

[1] Juan P. Bello Rachel M. Bittner, Eric Humphrey. Pysox: Leveraging the audio signal processing power of sox in python. 2016.