Title: **TAI - Lab work n°3**

Author: **Bruno Bastos, Eduardo Santos, Pedro Bastos**

Date: **21/01/2022**

Contents

1. INTRODUCTORY NOTE 2
2. STEPS AND DECISIONS DURING THE IMPLEMENTATION 2
2.1. HOW TO RUN 3
2.1.1 main.py 3
2.1.2 tests.py 3
3. RESULTS AND ANALYSIS 4
4. CONCLUSION 6

1. Introductory Note

Normalized Information Distance (NID) gives for two inputs x and y, the similarity between them. However, NID is not computable so instead it is used an approximation, using compression, called Normalized Compression Distance (NCD). NCD uses the approximation

$$NCD(x,y) = \frac{C(x,y) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}} \quad (1)$$

where C(x) represents the number of bits to compress a string using the compressor C. The closer the distances are to zero, the higher the similarity between the two strings, x and y, is.

The objective is to use the NCD, with multiple compressors, in order to obtain the similarity between a small music sample and every complete music in the dataset. The sample will then be assigned to the one which has the smaller average distance for all the compressors.

2. Steps and Decisions During the Implementation

The program accepts a complete music sample from the dataset, which will be trimmed, starting at a defined position. This is done by using a sox transformer that creates a new sample by trimming the original file.

After, the program needs to calculate the NCD using the complete music dataset for multiple compressors. It is expected that the smallest distance corresponds to the original file from where the sample was taken.

However, compressors require another sample format in order for them to perform better. Using the GetMaxFrequencies program provided by the professor, the sample audio is transformed into the set of its most significant frequencies. This way, the data is more suitable for compression for the compressors used and will give better results.

Now, the NCD is applied using different compressors for every file in the dataset. This is done by compressing the small sample, the complete music from the dataset and the concatenation of both the sample and the music. Then, it is used the size of the resulting compressions to calculate NCD and the one with the smallest distance will correspond to the file where the sample was taken from.

To test the robustness of the program it is possible to add white noise to the sample and see if it is able to predict correctly the music.

2.1. How to run

2.1.1. main.py

```
•100% ➜ python3 main.py -h
usage: main.py [-h] [--sample file] [--start_trim START_TRIM] [--threshold THRESHOLD] [--compressor COMPRESSOR] [--noise NOISE] [--st] [--n]

Recognize music.

optional arguments:
  -h, --help            show this help message and exit
  --sample file         Sample file
  --start_trim START_TRIM
                        Seconds to start trim
  --threshold THRESHOLD
                        Percentage of the song to test
  --compressor COMPRESSOR
                        Compression type (gzip, bzip2, lzma).
  --noise NOISE         Noise value. Default is no noise
  --st                  Plot graphic with variation of sample size.
  --n                   Plot graphic with variation of noise.
```

Figure 1: Main program menu.

As you can see, there are several options in our Main program. It is possible to:

- Select the music file to test;
- Change the timestamp to start the trim;
- Change the percentage of the song;
- Select the compressor to be used (supports gzip, bzip2 and lzma);
- Add noise to the sample;
- Plot a graphic with the variation of the sample size;
- Plot a graphic with the variation of the noise.

2.1.2. tests.py

Besides our main program, we have a separate file that gives a graphic presentation on the precision of each compressor given different sample sizes.

3. Results and Analysis

In theory, if a sample is taken from a music in the dataset, the program should always identify the correct file from which the sample is taken. The initial tests meant to test that. For each music in the dataset, having a sample size of 50% of the complete music, the program was able to identify correctly every single music.

However, half of the reference music might be a lot, so it was tested if the program had the same performance with much smaller sizes for the sample file. Figure 2 shows the precision of the program when selecting a sample for every music in the dataset. The results were observed for each of the compressors.

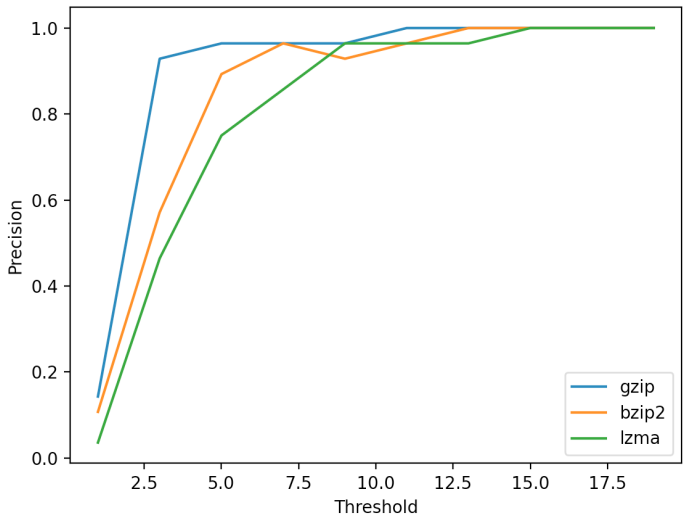

Figure 2: Global precision for different sample sizes.

Here, it is possible to see that for much smaller values of the threshold the program performs considerably worse. However, even if it correctly predicts the music where the sample was taken from, the level of similarity between the sample and the full music isn't good. To show that, tests were performed by changing the sample size and calculating the NCD. Figure 3 presents the evolution of the NCD value, between the sample and the music it was taken from, as the size of the sample increases.

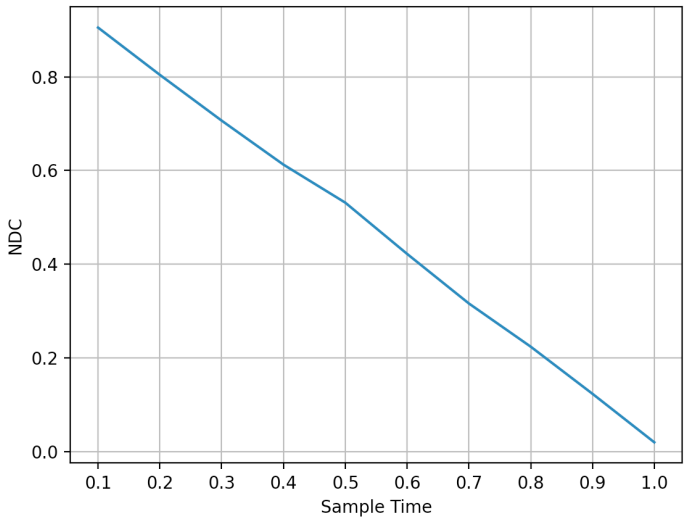

Figure 3: NCD for different sizes of the sample size.

As expected, for bigger samples, the program finds it more similar with the complete music it was taken from.

Although the results seem promising, the test cases that the program was put through do not represent the environment were it would be useful. A user might want to know which music is playing at a given point by making the program listen to it. The problem is, the program cannot just capture the music, it will also capture other sounds, which for our case is considered noise. In order to test this type of environments, the program can add white noise to the samples. The effect the noise has in predictions is pictured in Figure 4.

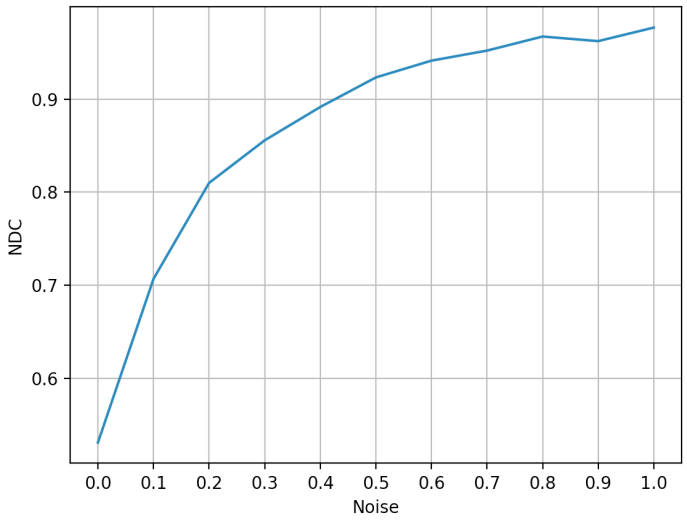

Figure 4: Effect of Noise.

To really see the impact of the noise, we tested the precision of the program, as shown before, but with a noise of 0.2. The results are presented in Figure 5.

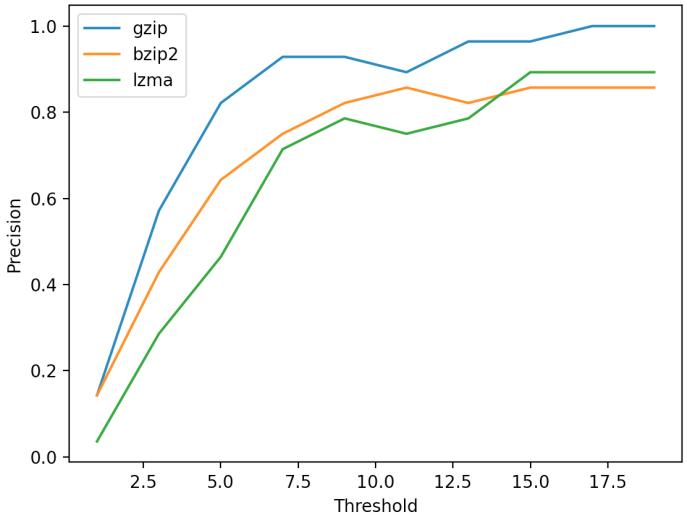

Figure 5: Effect of Noise.

As expected, the noise had a negative effect on the precision of the program. As you can see, the higher is the noise, the higher is the percentage of the song for the program to be precise. We can also conclude, from Figure 2 and 5, that gzip has slightly better results than the other 2 compressors.

4. Conclusion

Normalized Compression Distance can be very effective when the task to identify the correct file where the sample is taken from. The tests made tried to understand the relation between the size of the sample size and the noise used in the sample. We reached the conclusion that, the greater the size of the sample, the similar it will be with the original file, which was to be expected. Noise tries to replicate a use case where the program could be used in a environment where there are multiple sounds. The experiment proved that the noise does indeed decrease the similarity between the sample and the original file, but if the noise value isn't too high, the program can still predict the correct file from which the sample was taken from.