DETI - 2023/2024

Compression-based method for Music Classification

TAI - Assignment 3

Diogo Paiva	103183
João Fonseca	103154
Rafael Gonçalves	102534

Algorithmic Theory of Information

Professor: Dr Armando J. Pinho

Professor: Diogo Pratas

June 10, 2024

Contents

Chapter 1

Introduction

1.1 Purpose of the report

The purpose of the report is to document the work developed by our group, for the third assignment of the Algorithmic Theory of Information course (2023/24), where we were tasked with the exploration of a compression-based method for music classification.

1.2 Repository on GitHub

The source code of the program `what_the_music` and the scripts used for the analysis and validation of the results can be found at our GitHub repository [1].

1.3 Measuring information distance between strings

The Kolmogorov complexity of a string x, denoted as $K(x)$, is defined as the size of the smallest program that, on a universal reference machine, produces x and halts. An information distance based on the Kolmogorov complexity [1] can be defined as

$$E(x,y) = max\{K(x|y), K(y|x)\}$$

and represents the length of the shortest binary program that computes x from y and y from x; however $E(x,y)$ is an absolute measure, not suitable to assess similarity. Therefore, a Normalised Information Distance (NID) [2] can be defined as

$$NID(x,y) = \frac{max\{K(x|y), K(y|x)\}}{max\{K(x), K(y)\}}$$

[1] https://github.com/joaompfonseca/tai-assignment3-group8

Another problem arises since the Kolmogorov complexity is not computable. A computable approximation to the NID is the Normalised Compression Distance (NCD) [2, 3] defined as

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

where $C(x)$ represents the number of bits required by a compressor C to represent x and $C(x,y)$ the number of bits needed to compress x and y together (we considered the concatenation of the strings). For this assignment, we considered the NCD as the indicator of the information distance between the signatures of two music samples.

1.4 Application in music identification

The main goal of this assignment was to develop a tool capable of performing the automatic identification of music, by using small samples for querying the database. The Normalised Compression Distance (NCD) was used to compute the information distance between the query samples and the music in the database - a smaller value of NCD indicates a bigger proximity or similarity. For compression, we considered four standard compressors: `gzip`, `bzip2`, `lzma` and `zstd`.

Chapter 2

The *What The Music* program

2.1 Description

The `what_the_music` program is a command-line tool whose purpose is to automatically identify musics using small samples for querying the database. It computes the Normalised Compression Distance (NCD) between a query sample and each music in the database. The query sample is assigned the name of the music in the database that yields the smallest value of the NCD. It is possible to query multiple samples in a single execution of the program, since it considers a whole folder that is passed as an argument.

Before executing the program, one should generate signatures for the music in the database and the query samples with the `GetMaxFreqs` program. It outputs a file with the `.freqs` extension for each file, which the only accepted extension type in the `what_the_music` program.

2.2 Relevant Features

Some relevant features present in this program include:

- **Selection of the compression method** to use in the NCD calculation. The supported compressors are `gzip`, `bzip2`, `lzma` and `zstd`.
- **Multiple queries in a single execution** since it considers all the valid samples (ending with `.freqs`) in an entire folder.
- **Caching into a binary file** of the number of bits of the music signatures for a certain compression method.
- **Retrieval of the top-k results** in a database query allows the retrieval of only the best matches for a given sample.
- **Support for logging of results** to a specific file allows us to test and analyse the program outputs for multiple parameter configurations.

2.3 Compile

It is optional since you can use the provided executable inside the `bin` directory.

- Run `cd bin` in root.
- Run `cmake .. && make` to compile the program.

2.4 Execute

- Run `cd bin` in root to change to the executable's directory.
- Run `./what_the_music REQUIRED OPTIONAL` to execute the program.

2.4.1 Required arguments

- `-d database_folder`: folder containing the music signatures to be inserted in the database (string).
- `-q queries_folder`: folder containing the music signatures to be queried on the database (string).
- `-c compression_method`: compression method of the music signatures (string). Supported compressors are `gzip`, `bzip2`, `lzma` and `zstd`.
- `-k top_k_results`: number of top-k resutls to be returned (int).

2.4.2 Optional arguments

- `-h`: shows how to use the program.
- `-l log_file_path`: path to the file where the log will be written (string, default is empty).
- `-t temp_folder`: folder to store temporary files (string, default is `queries_folder`).

2.4.3 Example

```
./what_the_music\
-d ../example/database\
-q ../example/queries\
-c lzma\
-k 10
```

2.5 Flow of execution

1. Parse program arguments and print program configuration.
2. Initialise the compressor with the chosen compression method and temporary folder.
3. Initialise the database with the music folder and the compressor.
4. Load the database with the music from the music folder.
   (a) Load the number of bits of the compressed music signatures from cache.
   (b) Compute the number of bits of each music signature in the database folder.
      i. Check if the file extension is `.freqs`.
      ii. Read the music signature from the file.
      iii. Check if the number of bits for the compressed music signature is already cached.
      iv. If not, compute it by compressing the music signature using the compressor.
         A. Create a temporary file `content.temp` with the signature to compress.
         B. Run the compressor on `content.temp` and store in another temporary file `bytes.temp` the size of the compressed `content.temp` in bytes.
         C. Read the number of bytes B from `bytes.temp`.
         D. Remove the temporary files `content.temp` and `bytes.temp`.
         E. Return the number of bits $B * 8$.
      v. Add a tuple (signature, bits) to database storage, with music file name as the key.
   (c) Save the number of bits of the music signatures to cache, if there is not a cache file yet.
5. Query the database with each sample, determining the NCD in relation to each music, in order to find the closest matches.
   (a) Check if the file extension is `.freqs`.
   (b) Read the music signature from the file.
   (c) Query the database with the signature and the desired number of top-k results.
      i. Compute $C(q)$, the number of bits of the query signature q using compressor C.
      ii. For each music in the database storage.
         A. Retrieve the signature e and the number of bits $C(e)$.
         B. Compute $C(q,e)$, the number of bits of the concatenation of the query signature q and the music signature e using compressor C.
         C. Compute $NCD(q,e) = \frac{C(q,e) - minC(q),C(e)}{maxC(q),C(e)}$, the Normalised Compression Distance between the query sample and the music in the database.
         D. Save a tuple (Label(e), NCD) in a vector.
      iii. Sort the vector with the results by increasing order of NCD.
      iv. Return only the top-k results from that vector.
   (d) If no logger file was specified, print the results (music name and NCD) to the standard output; otherwise, print the query music name, followed by the top-k music name results, in increasing order of NCD.
6. Exit the program.

Chapter 3

Validation

In order to assess the robustness of the `what_the_music` program, we chose a diverse playlist of songs and conducted several experiments, relying on accuracy as the main metric, but also on retrieval metrics, using as reference the performance on raw audios. In our benchmarks, we started by analysing the influence of each test sample's duration, and applied a wide range of effects to the audio samples (ambience sound, reverb, reverse and different noises with varying intensities). For each one of the mentioned transformations, we tested the program with 4 compressors (gzip, bzip2, lzma and zstd). Then, we picked the best configuration and studied the impact of the database size on the program's performance.

3.1 Metrics

Although the problem returns a list of songs ordered by the NCD, we are most interested in the first entry, as it is the one that the program assigns to the query sample. Therefore, we can see the problem as a binary classification task, where there is no concept of true positives, or true negatives unlike problems such as cancer detection or spam filtering. In this context, we just have true and false predictions, hence accuracy is the most suitable metric to evaluate the performance of the program. It is defined as the ratio of the number of correct predictions to the total number of predictions.

Nevertheless, if we consider the first Top-K musics returned by the program for the raw audios as the ground truth, i.e, the relevant samples, the problem can be faced as a retrieval task. In this case, we could use common metrics, such as Precision@K, Recall@K and AP@K (Eq. 3.1), and average them over all test samples. However, as the program returns always the K most probable songs, the denominator of the Precision@K (all retrieved songs) and of the Recall@K (all relevant songs) is always K, which makes the metrics less informative. To overcome this limitation, we decided to use only the Mean Average Precision (MAP) at top-K, which is the mean of the AP@K over all test samples (Eq. 3.2).

$$AP@K = \frac{1}{K} \sum_{i=1}^{K} Precision@i \times rel_i \tag{3.1}$$

$$Map@K = \frac{1}{N}\sum_{s=1}^{N} AP@K_s \tag{3.2}$$

where k is the number of retrieved items, i is the position in the top-k list, rel_i is a binary value that indicates if the item at position i of the top-k list is relevant or not, s is the index of the test sample, and N is the number of test samples.

For plotting the results, we give preference to the accuracy metric, as it is more intuitive and easier to interpret. However, in chapter 4, we will also present the retrieval metrics, as they provide a more detailed view of the program's performance, when comparing to the one obtained with the raw audios.

3.2 Dataset

In this stage, we intended to gather a diverse set of songs with different rhythms, but also with at least two songs of the same genre to test the program's ability to distinguish between them. Harnessing the Spotify playlist 'Top 100 Most Streamed Songs of All Time' [1], we were able to collect 93 songs, leaving out 7 tracks due to copyright issues when downloading. The playlist contains a variety of sub-genres, mostly from Pop, Hip-Hop and Rock.

3.3 Transformations

For creating segments of the songs and adding effects, we relied on the `sox` command-line utility. For automatising the whole pipeline, which consists of trimming the audio, adding an effect, generating the signature and querying the database, we created 2 Python scripts: `transform.py` and `benchmark.py`. The first script is responsible for applying the transformations to the audio samples, leveraging whenever possible `pysox`, a Python wrapper for `sox`. Some effects were not available in the wrapper, so we had to use the command-line utility directly through the `os` module. The second script is responsible for running the program with the transformed samples and collecting the results with the well-known `pandas` library. In both files, we took advantage of the `multiprocessing` module to create a pool of workers, speeding up the execution.

3.3.1 Trimming

The default sample duration is 30 seconds. However, we wanted to study how the duration of the sample affects the program's performance. In this sense, we created random segments with durations of 5, 10, 15, 30 and 60 seconds and starting at least 30 seconds from the beginning of the song. We then ran the program with each one of them. Fig. 3.1 shows the accuracy obtained for the tested sample duration's and compression algorithms.

[1] https://open.spotify.com/playlist/5ABHKGoOzxkaa28ttQV9sE

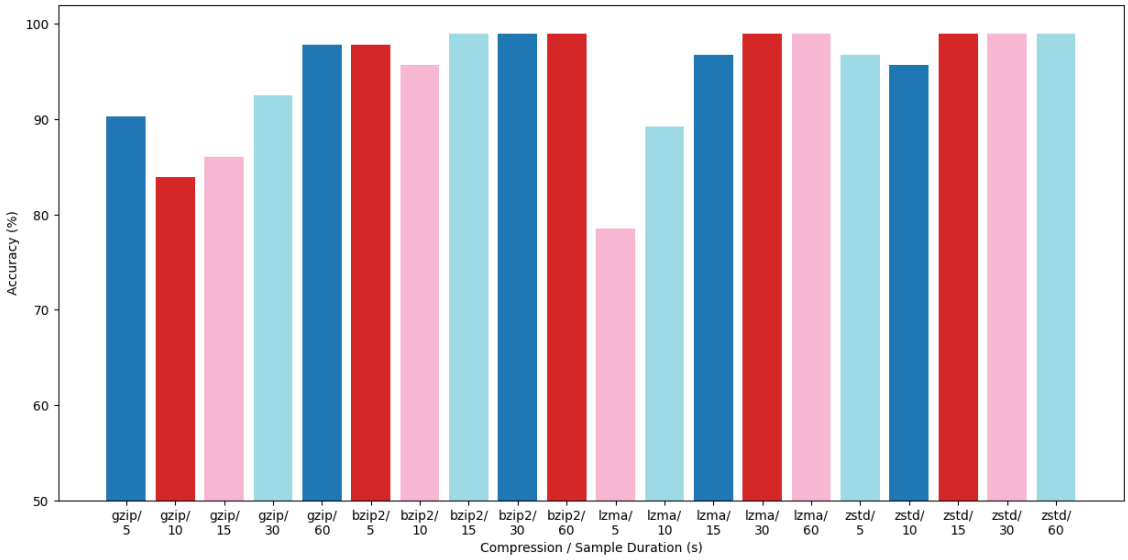

Figure 3.1: Accuracy of the raw data varying the compressor and the duration

As we can see by Fig. 3.1 the compressor that achieve the best results was the bzip2 and the compressor with the worst results was the gzip.

3.3.2 Ambience sound

When using popular song recognition apps, such as Shazam, the users can often identify the song even when there is a lot of noise in the background. We managed to get a recording of a coffee shop, where people are talking and the sound of cups and plates can be heard, and mixed it with the raw audio samples (30 seconds), using the `sox` utility. The accuracy as seen in Table. 4.1 is the same or even better in one case than in the raw audio samples.

3.3.3 Reverse

The reverse effect is a simply playing the audio backwards. We applied this effect to the 30-second audio samples and, as expected, the accuracy as seen in Table. 4.1 was very similar to the one obtained with the raw songs, since the frequency content of the audios is preserved, with the exception of the compressor lzma that decreases significantly the results.

3.3.4 Reverb

Reverb is a common effect used in music production to give the impression that the music is being played in a large room, and it has a shorter bounce time than echo. We applied this effect to the 30-second audio samples and varied the reverberance from 10% to 100% (`sox` default is 50%).

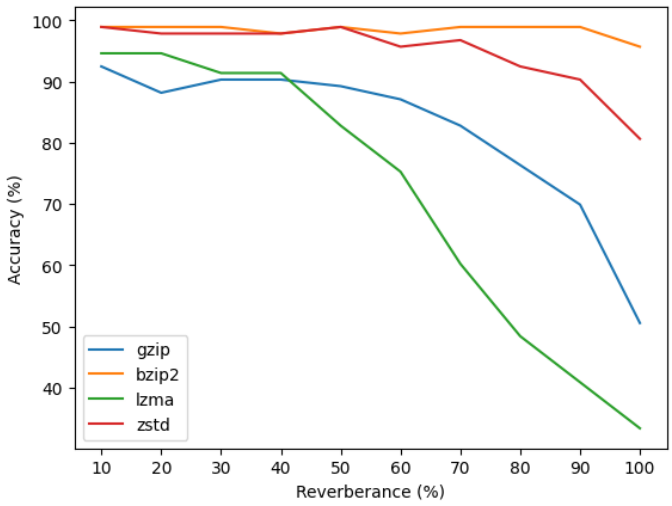

Figure 3.2: Accuracy based on the compressor and noise level for reverb

As expected, the accuracy decreases when the percentage of reverberance increases as we can see in Fig. 3.2. The compressor that achieves the best results for this type of transformation is the bzip2 and the worst compressor is the lzma.

3.3.5 White noise

In audio engineering, noises are often categorised by different colours (white, pink, brownian, blue, violet and grey), according to their power spectral density [4]. White noise is the most basic type of noise, as it is simply a random signal with a flat power spectral density. It is often used in music production to mask unwanted sounds.

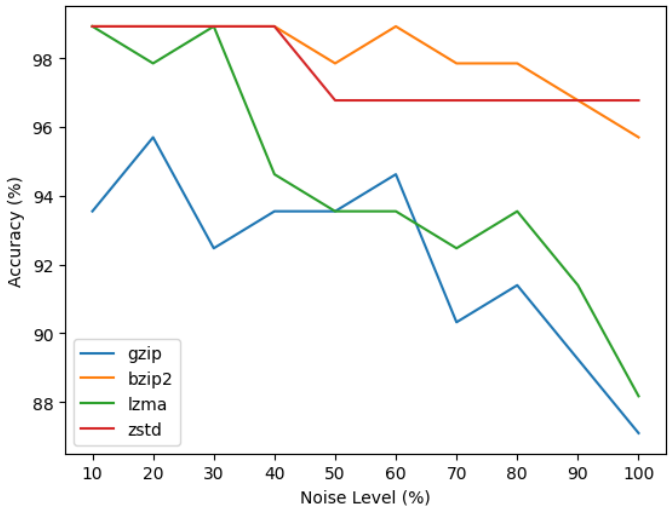

Figure 3.3: Accuracy based on the compressor and noise level for white noise

Fig. 3.3 shows the accuracy obtained for the tested intensities of white noise and compression algorithms. And, as we can see, for 3 compressors, the accuracy decreases when the percentage

of noise level increases. On the other hand, for zstd, the accuracy is constant with only a small drop from the 40% noise level to the 50% noise level and zstd is the best compressor for this transformation.

3.3.6 Pink noise

According to [5], pink and brownian noises are considered natural noises, as they have a spectral distribution similar to that of many natural sounds. As we had a large configuration space, we decided to test only one of them: pink noise. This type of noise has more power at lower frequencies and less at higher frequencies, and its spectral distribution is similar to that of many natural phenomena, such as heartbeats and ocean waves. Fig. 3.4 shows the accuracy obtained for the tested intensities of pink noise and compression algorithms. The accuracy once again decreases when the percentage of noise level increases, as expected, and the best compressor for this transformation is the bzip2 and the worst one is the gzip.

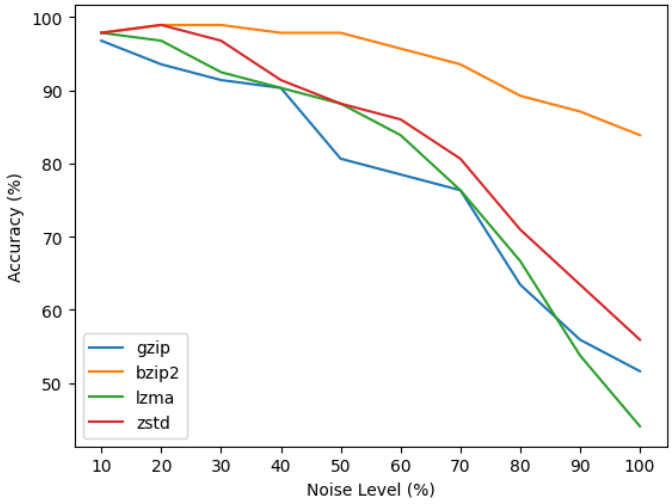

Figure 3.4: Accuracy based on the compressor and noise level for ping noise

3.3.7 Database size

In order to understand how the database size affects the program's performance, we created 3 databases with 10, 50 and 100 songs, and ran the program with the raw audios, using another script named `db_size_benchmark.py`. Fig. 3.5 shows the accuracy obtained for the tested database sizes and compression algorithms. The accuracy varies a lot in all the cases, but in general the better compressor is the lzma. Probably due to the presence of songs that are hard to identify, the results also suggest that the number of incorrect predictions is not that different as we increase the database size. We think that the accuracy increase is related to its denominator increase (total number of predictions), but the number of incorrect predictions is relatively constant. As future work, with less hardware constraints, one should test the program with other databases.

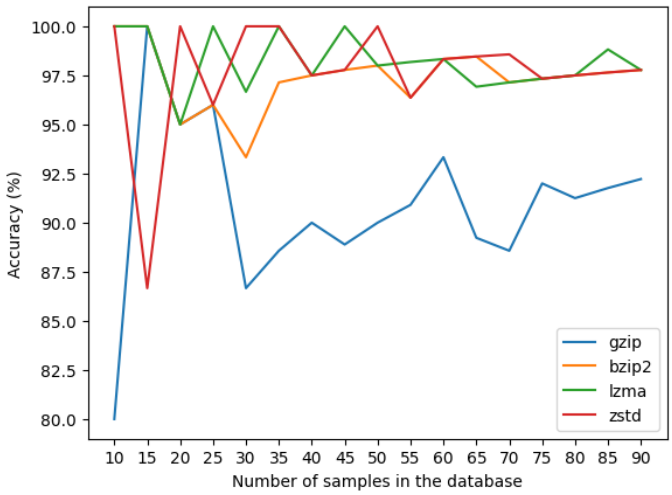

Figure 3.5: Accuracy based on the compressor and database size

Chapter 4

Discussion

Combining different compression algorithms with the described effects and the associated arguments, apart from the database size tests, we were able to test 148 configurations for each one of the 93 songs, totalling 13,764 program runs. After all jobs were completed, our first priority was to understand which musics were the most difficult to identify. Fig. 4.1 shows the 10 songs with the least number of correct predictions. This can be related to several factors, as the existence of similar rhythms in the dataset and consequently similar frequencies.

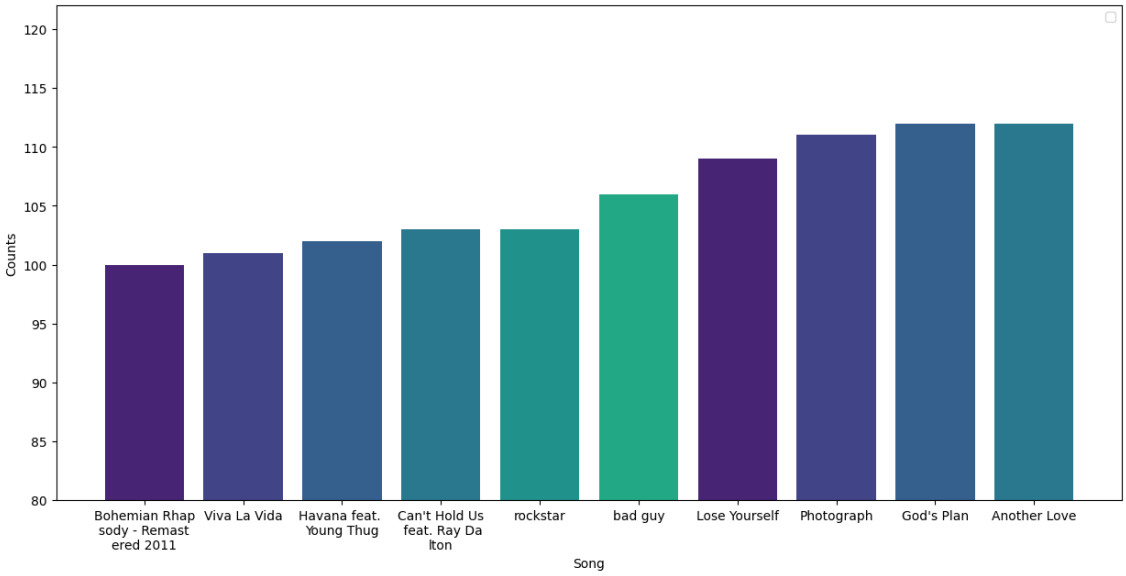

Figure 4.1: Songs with the least number of correct predictions

To have a holistic view of the tested configurations, we summarised them in two tables, one for each evaluation metric: accuracy 4.1 and Map@10 4.2. We emphasise that the retrieval metrics were calculated using the top-10 songs returned by the program for the 30-second raw audios.

Table 4.1: Accuracy comparison

Compression	**Raw (30s)**	**White (80%)**	**Pink (80%)**	**Reverb (80%)**	**Reverse**	**Ambience**
gzip	0.9247	0.914	0.6344	0.7634	0.828	0.9462
bzip2	0.9892	0.9785	0.8925	0.9892	0.9247	0.9892
lzma	0.9892	0.9355	0.6667	0.4839	0.6237	0.9892
zstd	0.9892	0.9677	0.7097	0.9247	0.914	0.9892

The lzma compression technique is dictionary-based and takes advantage of pattern repetition. The reverb effect repeats parts of the audio and that leads to the creation of patterns not present in the original audio, which can explain the poor accuracy of this compressor for the reverb effect. Reversing the audio leads to a similar decay in the accuracy for lzma, since it changes the patterns in the original audio completely. The pink noise effect was not an issue for the bzip2 compressor, when comparing its performance against the other compressors.

In general, we obtained robust accuracy results with all the compressors; however, bzip2 was the best performing compressor on average.

Table 4.2: Map@10 comparison

Compression	**White (80%)**	**Pink (80%)**	**Reverb (80%)**	**Reverse**	**Ambience**
gzip	0.0898	0.0927	0.0772	0.0758	0.0683
bzip2	0.0588	0.0708	0.0536	0.0486	0.0463
lzma	0.0116	0.0088	0.0074	0.0057	0.0052
zstd	0.1012	0.0936	0.0685	0.0904	0.0833

Even with a good accuracy, the top-k order - when compared with the raw sample - is very distinct. In simple terms, this means the predicted song is frequently the same, but the subsequent guesses vary greatly. In some cases, the Mean Average Precision is near zero, revealing that the top-k results are almost completely distinct from the raw samples.

Chapter 5

Final considerations

This project successfully demonstrated the potential of a compression-based method for music classification through the development and evaluation of the `what_the_music` program. Through the usage of the Normalised Compression Distance (NCD) and various standard compression algorithms, we achieved promising results in identifying music from small audio samples. Among the tested compressors, bzip2 consistently yielded the highest accuracy, particularly with longer sample duration, even when faced with various audio transformations such as noise and reverb. These findings underscore the robustness and effectiveness of the NCD approach in music identification scenarios.

However, several challenges remain, including the computational complexity associated with compression processes and the sensitivity to audio quality variations. Future work should focus on optimising algorithms for efficiency, exploring additional audio features to enhance robustness, and conducting real-world testing to validate the tool's applicability in diverse environments.

Overall, this report highlights the viability of a compression-based music classification and sets the stage for further advancements in the field of music information retrieval.

References

[1] C.H. Bennett, P. Gacs, Ming Li, P.M.B. Vitanyi, and W.H. Zurek. Information distance. *IEEE Transactions on Information Theory*, 44(4):1407–1423, 1998.

[2] Ming Li, Xin Chen, Xin Li, Bin Ma, and P.M.B. Vitanyi. The similarity metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, 2004.

[3] R. Cilibrasi and P.M.B. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

[4] Shih-Yi Lu, Yuan-Hao Huang, and Kuei-Yi Lin. Spectral content (colour) of noise exposure affects work efficiency. *Noise and Health*, 22(104):19–27, 2020.

[5] Jaymee-Lee Tolliday. The Colours of Noise: What are they and what do they mean? https://www.cirrusresearch.co.uk/blog/2023/04/the-colours-of-noise/, April 2023.