# Classification problem with data compression

Identify if an audio segment belongs to a song

TAI - 2023/2024

João Mourão Nº 102578
João Rodrigues Nº 102487

# Introduction

Automatic music identification from short audio clips is a desired capability for various applications. This project explores using Normalized Compression Distance (NCD) for this task. NCD approximates the similarity between two strings by analyzing their compressibility. We will build a system that calculates NCD between a short music clip (query) and songs stored in a database. The song with the minimum NCD (most similar compressed representation) will be identified as the match. To assess the effectiveness of different compression algorithms and the robustness to noise, we will employ various compressors and introduce noise to the query clips during testing. The project will culminate in a short video demonstrating the developed system and its functionalities.

# Classification problem

In this project, the classification problem is to identify the most similar song in a database given a short audio clip (query).

We can frame this as a multi-class classification problem where the classes are the individual songs in the database. The system's task is to analyze the query clip and assign it the class label (song) that corresponds to the most similar music based on the NCD calculation.  The NCD is calculate through the following formula :

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

where C(x) represents the number of bits required by compressor C to represent x and C(x, y) indicates the number of bits needed to compress x and y together (usually, the two strings are concatenated). Distances close to one indicate dissimilarity, while distances near zero indicate similarity.

Our database consists of a full album of Pink Floyd called “The Wall”, with 25 songs distinct from each other.

# Model implementation

The code was developed in python and consists of a CLI (command line interface ) tool that provides a suite of commands for processing audio files. It supports extracting segments, creating music signatures, compressing databases, making predictions, cleaning up files, adding noise to audio files, performing grid searches. The options are :

**seg**: Extract a segment from audio files
This function handles the extraction of audio segments efficiently using SoX commands. It takes parameters for start time, duration, input directory, output format, and output directory. It iterates through each audio file in the data bases of music, extracting the specified segment.

**sig**: Create a music signature.
This option creates a “signature” of all music in the database, i.e., through the file GetMaxFreqs, it’s possible to get the highest frequencies of the music.

**compress**: Compress a database of complete music tracks
It's designed to compress files found in a specified directory, generally we want to compress the signatures of the database. It takes parameters for the compression method, the input directory containing files to be compressed, the output directory where the compressed files will be saved and optional compression level.

**pred**: Predict the music given a segment
Here the program will predict which music is our dataset. We need to provide the path to our segment the type of compression and the level of compression. These last two should be the same used in the database to avoid a degradation of the model’s performance. Then, the segment will be transformed to the signature as well as the concatenation between the segment and all music of the database separately and compressed after Now we have all the material to apply the NCD formula. The lowest value will be the predict

**clean**: Clean all music files except the database
This will eliminate all files created while executing these functions.

**add_noise**: Add noise to all files in a directory
For each file, it constructs a command to generate noise using the `sox`, specifying the type and volume of the noise based on the provided in the interface. Another command is constructed to mix the original audio file with the generated noise. The noise must be done before segmenting the music.

**grid_search**: Perform a grid search for optimal parameters
The grid search function iterates through all combinations of these parameter values. For each combination, it processes the music data according to the chosen settings. This involves adding noise, extracting segments, creating signatures from the segments, and compressing the entire music database.

The music identification itself is performed by a pred (referenced before) . While the current implementation doesn't explicitly track the identification accuracy (correct/incorrect) for each parameter set, it creates a dictionary to store the predictions made by the `predict` function. Finally, it logs the combination of parameters used along with the predictions for each segment to a text file for further analysis.

## Experiments to improve results

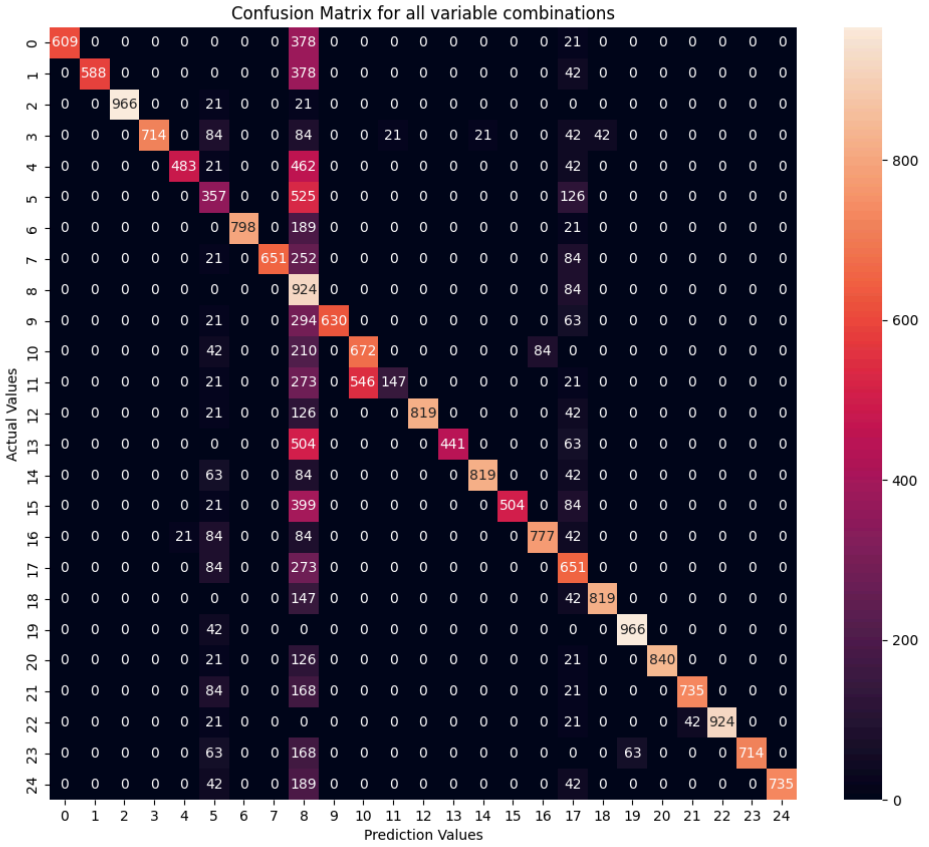
A grid search was implemented to comprehensively evaluate the robustness of the music identification system under various conditions. This approach involves systematically testing all possible combinations from predefined sets of parameter values. The parameters explored in this grid search were:

- Noise Type: The type of noise added to the music data (e.g., white noise, pink noise).
- Noise Level: The percentage of noise introduced into the music (e.g., 0%, 10%, 50%).
- Segment Length: The duration of the music segments used for identification (e.g., 10 seconds, 30 seconds).
- Compression Mode: The compression algorithm employed to compress the music data (e.g., gzip, bzip2).
- Compression Level: The compression level used by the chosen compression algorithm (e.g., none, level 2, level 9).

This grid search serves to identify the parameter combinations that lead to the most accurate music identification. By analyzing the results file, we can determine how well the system handles different noise types, segment lengths, and compression configurations. This information can be crucial for optimizing the system's performance in real-world scenarios.

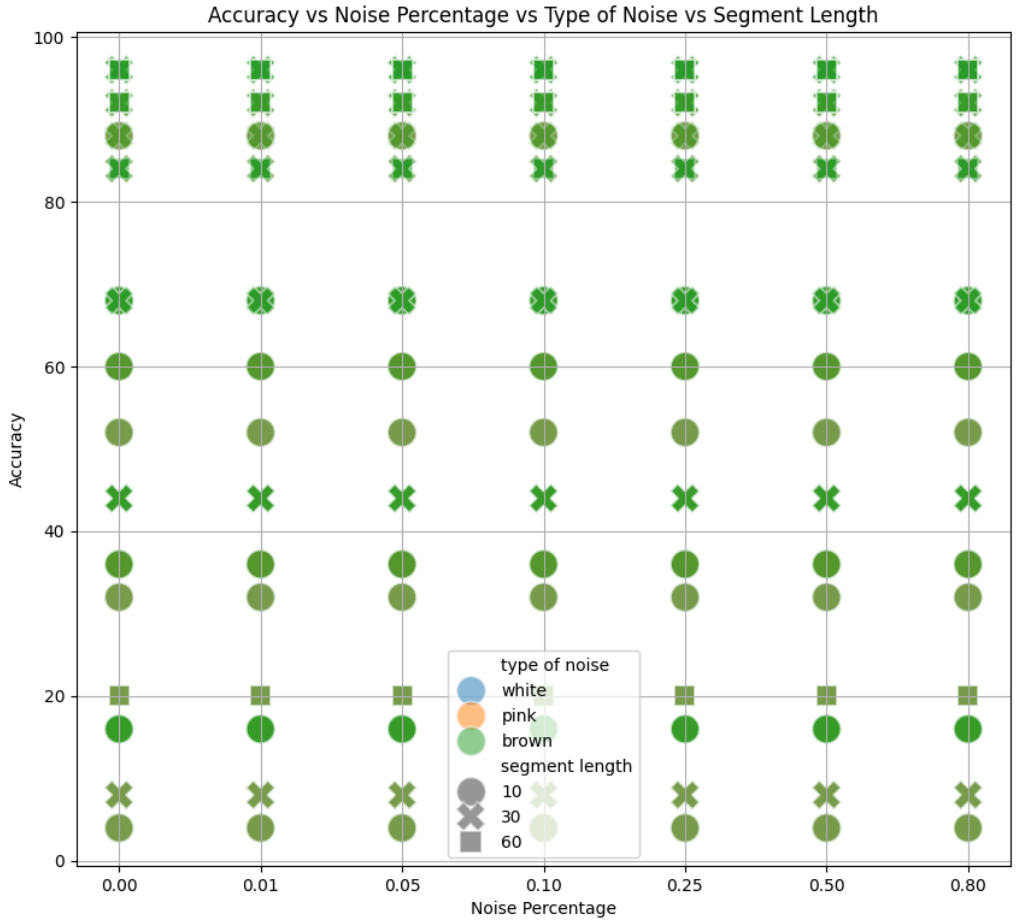
## Results

***Classification problem with data compression***



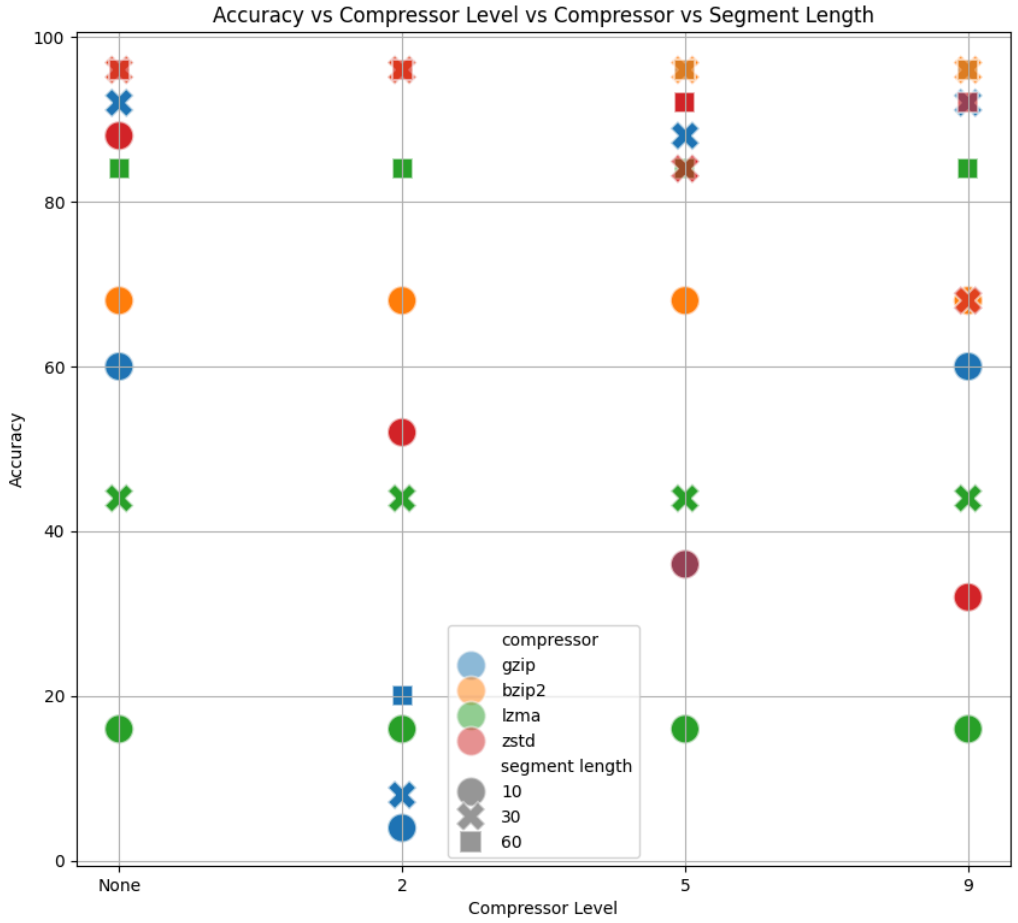

- Overall, the model predicts correctly for the majority of the cases. However, there are some classes that display some kind of oddness.
- The class 8, corresponding to the song “Goodbye Cruel World”, has a high tendency of being  predicted even when it is not the actual class. In most cases, if the class is incorrectly predicted, the prediction is class 8. Classes 5, corresponding to the song “Don’t Leave Me Now”, and 17, corresponding to the song “Run Like Hell”, also show this property, although on a smaller scale. This may happen due to common patterns that exist throughout the album and are simply more common in these songs.
- Class 11 is often confused with class 10 and we are very confident that this happens due to these classes corresponding to the songs “In the Flesh” and “In the Flesh?”, which are very similar and have some equal sequences.

***Classification problem with data compression***




- Noise percentage does not seem to significantly influence accuracy. The accuracy values remain consistent across different noise percentages.
- Segment length appears to have a more noticeable impact on accuracy, with longer segment lengths (60) generally showing higher accuracy.
- All types of noise (white, pink, brown) affect the accuracy similarly, with no significant variation among them.

***Classification problem with data compression***




- Higher compression levels (5 and 9) generally correspond to lower accuracy across most compressor types and segment lengths.
- No compression ("None") consistently shows the highest accuracy, suggesting that higher compression levels lead to a potential negative impact on accuracy.
- The gzip compressor seems to have a much lower accuracy for, specifically, compressor level 2 and mostly average accuracies for other values. This compressor also seems to overall have better results when the segment length is 30.
- The bzip2 compressor has very high accuracy results on average, and seems to maintain its accuracy independently of the compressor level. It has worse results for low segment lengths.
- The lzma compressor seems to also be independent of compression levels and has better results for higher segment lengths.
- The zstd compressor has very high and consistent accuracies for segment lengths of 30 and 60 but for segment lengths of 10, the accuracy decreases with the increase of the compression level.

# Conclusion

Overall, this method works really well on music detection  and shows really good robustness to any type of noise and any percentage of it. However, the segment of audio must have a decent size to increase probability of getting the right prediction. The method also works better with certain compressors and certain levels of compression,so it should be chosen carefully.