Assignment 3 - Music and Genre Likeness Using Information distances

Algorithmic Information Theory (2023/24)

Vicente Barros	Filipe Silveira	Mariana Andrade
97787	97981	103823

June 11, 2024

1 Introduction

This report explains the third and final project for the *"Algorithmic Information Theory"* course, aimed at developing a music similarity system to automatically identify music based on a sample. The core concept of this project is to employ information distance functions as an approach to solving this problem, by also using different genres of music to create more variations and have more results for our conclusions.

We will first describe our methodology and how we organized our code, our datasets and how we organized them, the base behind the Normalized Compression Distance (NCD), how our main program operates, and the results and conclusions of our approach.

2 Methodology

For the development of the automatic system to detect music, due to its simplicity we created multiple single Python files for each task which facilitate the implementation and testing processes.

The source code for this project was developed using various resources, including those provided in the course materials, such as ***GetMaxFreq***, and others referenced during lectures.

Regarding the repository structure, we organized our project into several key directories to maintain clarity and manageability:

- The **src** directory serves as the main container for all our source files. It includes several key Python scripts that handle various aspects of our audio processing and analysis pipeline.
- The **database** folder contains our database with our audio files and their signatures.

- The **datasets** folder contains all of our music files organized by source and their sub-folders organized by genre.
- The **cache** folder is used to avoid recalculating the compression length.
- The **analysis** folder is used for the analysis of the results and to store the analysis.
- The **deliveries** folder is used to store the report and link to the video of this project.
- The script **generate_data.sh** is used to generate the database for different parameters using the *GetMaxFreqs* program.
- The script **generate_best_data.sh** is used to generate the database for different noise levels using the best parameters found in the previous step using the *GetMaxFreqs* program.

3 Dataset

3.1 Gathering Initial Data

To start the development of this project, we decided to use copyright-free music provided by *YouTube Studio* because of its straightforward usage. From the beginning, it was settled that we would use music from genres that do not overlap and thus, we decided to select 16 songs from each of the genres of *Rock*, *Jazz*, *Hip-hop* and *Classic*.

The usage of this first dataset validated that our implementation was working properly. With that, we decided to increase the data complexity by using songs from *Spotify* which add voice - the previous dataset is mainly used as background music. Since *Spotify* does not allow music download on their platform, we found a workaround using the ***Spotifydl***[spo24] Python Library that searches in other platforms such as *Youtube* and *Soundcloud*, music with similar names and downloads them. The songs we selected from *Spotify* came from playlists with the Top 100 from each genre [spoa] [spoc] [spob] [spod] where we randomly selected 12 songs from them which 10 are to be used in our database and 2 are unseen for future study.

3.2 Manipulating the audio files

Following the selection of the music to be used from each genre, it was necessary to manipulate them to be used in our application. Firstly, all the music was converted to **44100Hz** bitrate and the format **WAV**, since the program was given to generate the signature, which will be delved in the next section of the report. Two of the pieces of music present in the database and the two left out from it were split into different sizes 5, 2 and 1 files to study the impact of varying the music size in identifying the music genre.

3.3 Making the signature

To create a signature for our audio files, we used a program developed by our professors called ***GetMaxFreq***[Pin24]. This program converts audio files into a frequency-domain "signature". The purpose of this signature is to capture the most significant frequency components of the audio.

To make the most out of the program, we first analyzed its parameters, which include:

- **-v (verbose):** When this flag is set, the program provides additional information during its execution, which is useful for debugging or understanding the processing steps.
- **-w freqsFile:** Specifies the file where the frequency signature will be written.
- **-ws winSize:** Determines the length (in number of audio samples) of the block (window) used for frequency analysis. The default value is 1024 samples.
- **-sh shift:** Sets the right shift (in number of audio samples) from one block to the next. The default value is 256 samples, meaning there is overlap between blocks.
- **-ds downSampling:** Defines the down-sampling factor from the original 44100 samples per second. The default value is 4, resulting in a sampling rate of 11025 samples per second.
- **-nf nFreqs:** Indicates the number of the most significant frequency components retained for each block. The default value is 4 components.
- **AudioFile:** The audio file to be processed should be in .wav or .flac format, stereo, sampled at 44100 Hz, 16 bits per sample.

By using our *generate_data* program in the *database* folder, we can generate and prepare the necessary data for our experiments. The Python script integrates the ***GetMaxFreq*** into a data processing pipeline.

The Python script orchestrates the usage of GetMaxFreqs by:

- **Generating a Shell Script Template:** The script dynamically creates a shell script (params.sh) to invoke GetMaxFreqs with the specified parameters (ws, sh, ds, nf).
- **Processing Audio Files:** It adds noise to the audio files to simulate different conditions and splits the audio into parts for training and testing.
- **Splitting Audio Data:** The script categorizes audio files into training and test sets. It processes each file, adding noise and splitting it into chunks of specified durations.

- **Metadata Management:** It manages metadata by tagging the processed audio files with relevant information, such as genre and file identifiers, using the taglib library.
- **Executing GetMaxFreqs:** After preparing the audio files, the script runs the shell script (params.sh) to generate the frequency signatures for the training audio files.

These steps ensure that our audio data is appropriately prepared and the frequency signatures are generated systematically for further analysis.

4 The Application

For the project, we decided to separate our main application into three phases: **Information Distances** to first understand our problem more in-depth, **Computational Model** to explain our implementation and **UI** to facilitate our final user to view results.

4.1 Information Distances

In our project, we utilize the concept of Information Distances to measure the similarity between audio files. We focus on the Normalized Compression Distance (NCD), an approximation of the non-computable Normalized Information Distance (NID).

The NID between two strings x and y is defined as

$$NID(x,y) = \frac{max\{K(x|y), K(y|x)\}}{max\{K(x), K(y)\}}, \tag{1}$$

where $K(x)$ is the Kolmogorov complexity of string x and $K(x|y)$ is the Kolmogorov complexity of x when y is given as additional information.

Since Kolmogorov complexity is non-computable, we approximate it using

$$NCD(x,y) = \frac{C(x,y) - \ min\{C(x), C(y)\}}{max\{C(x), C(y)\}}, \tag{2}$$

where $C(x)$ denotes 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. Distances close to one indicate dissimilarity, while distances near zero indicate similarity.

To test the differences and gain more insights, we used multiple compressors such as bz2, gzip, lzma, zstd, and zlib.

4.2 Computational Model

Our computational Model consists of a Python class which is initialized with the dataset that will read all the signature files and get the genre and the music's name from the corresponding WAV file.

The core method of the Model class is the *predict* method which receives the file to classify and calculates the Normalized Compression Distance using the different compressors ("*bz2*", "*gzip*", "*lzma*", "*zstd*" and "*zlib*").

When the results of the NCD are processed and available. For each result, the score is added to the total score for the corresponding method and genre, and the count of scores for the method and genre is incremented. If the score is the best (lowest) seen so far for the method, it is stored along with the associated data. The top 10 scores and associated data for each method are also kept track of and finally, the average score for each method and genre are computed.

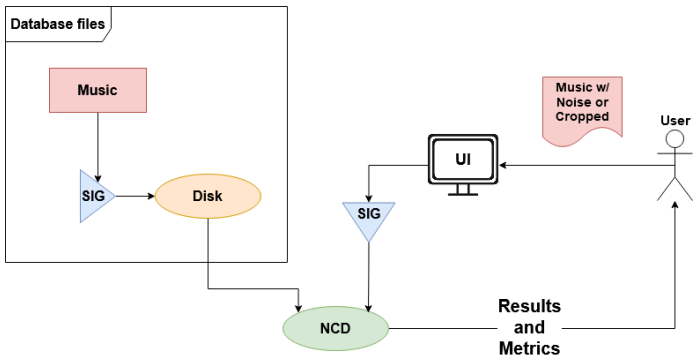

Figure 1: Example of our flow Diagram.

4.3 UI

The user interface for our application is implemented using Streamlit, providing an interactive and user-friendly experience for analyzing and visualizing the results of our audio signature processing and classification.

The sidebar contains buttons and navigation options for running the analysis and generating reports. One key feature is the "*Run All and Create Report*" button, which triggers the execution of the entire analysis pipeline on the dataset. This includes generating detailed reports and visualizations of the classification results. Additionally, there is a recursive tree structure in the sidebar that allows users to navigate through the dataset directory, enabling easy selection and interaction with individual audio files.

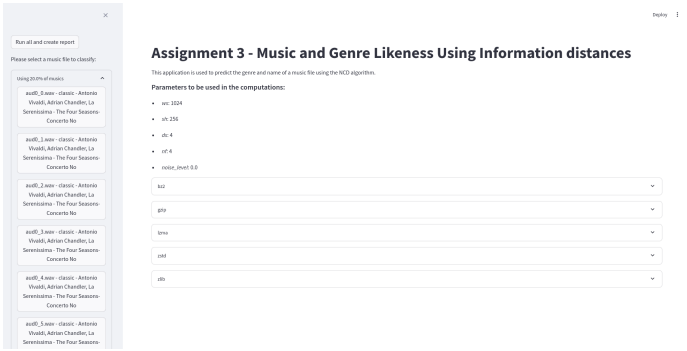

Figure 2: Our application with the sidebar.

When an audio file is selected through the UI, the *on_audio_click* function creates a temporary directory to store the uploaded audio file and converts it into a signature file using the *GetMaxFreqs* program. The pre-trained model then predicts the genre of the audio file based on its signature, with the results stored in the session state for display.

The application maintains placeholders for displaying prediction results for different compression methods. These placeholders are dynamically updated with the best cases and average NCD scores for each method. The best-case display shows the best matching genre and file name for each compression method. Graphical representations are created using *Plotly* to visualize bar charts displaying the average NCD scores for each genre and the top 10 NCD scores for each method.

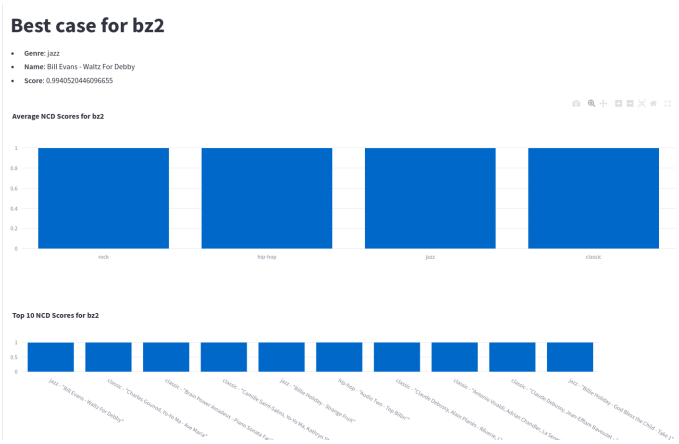

Figure 3: Example of our NCD Results

The *run_all* function generates detailed reports on the classification performance. It computes and saves classification metrics (accuracy, precision, recall,

f1-score) for each compression method and genre. CSV files summarizing the classification results are generated, including parameters like window size, downsampling rate, and the number of significant frequencies used.

By integrating these components, our application offers an interactive platform for analyzing audio files, running predictions, and visualizing results, making it accessible and informative.

5 Results

For the results, we decided to create a run to identify the best parameters for the **GetMaxFreqs**. Depicted in Fig. 4, it is possible to verify how the accuracy behaves with each parameter. In the end, we decided to go with a Window Size of 4096, a Shift of 1024, a Downsampling of 16, and a number of the most significant frequency components (NF) of 2.

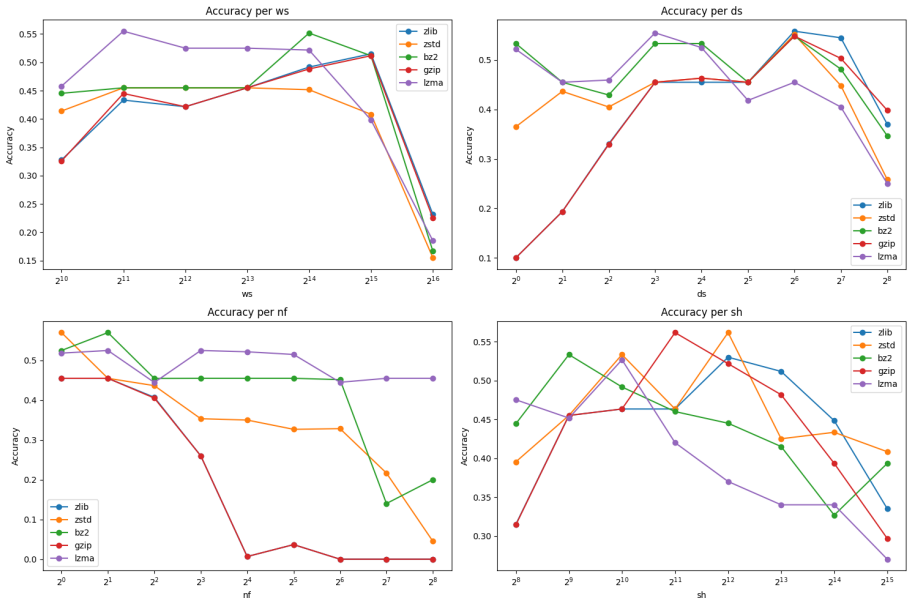

Figure 4: Accuracy varying the different parameters

5.1 Accuracy per Parameter Analysis

5.1.1 Window Size (ws)

The accuracy of genre identification varies with the window size. Initially, as the window size increases from 2^{10} to 2^{13}, there is a noticeable improvement in accuracy, reaching a peak. However, beyond a window size of 2^{14}, the accuracy tends to decline sharply. This trend indicates that while larger windows capture

more data, they might also introduce noise or redundancy that hampers the model's performance.

5.1.2 Shift (sh)

For the shift parameter, the results show that accuracy fluctuates with different shift values. Smaller shifts generally maintain a more stable accuracy, but as the shift increases, there are significant peaks and troughs. This suggests that the optimal shift value balances between too frequent updates, which might introduce noise, and too infrequent updates, which might miss important transient features in the music.

5.1.3 Downsampling (ds)

The downsampling parameter has a mixed impact on accuracy. Lower downsampling values tend to start with lower accuracy, which improves as the downsampling rate increases, peaking at mid-range values. However, very high downsampling rates again lead to a drop in accuracy. This indicates that while some reduction in data helps in focusing on significant features, excessive downsampling might result in loss of critical information.

5.1.4 Number of Most Significant Frequency Components (NF)

The number of most significant frequency components shows a clear pattern where accuracy decreases as the number of features increases. This is evident across most compression methods, with notable drops in methods like bz2 and zstd at higher feature counts. The chosen value of 2 for NF represents a balance, capturing essential frequency information without overwhelming the model with too much data.

5.2 Mean Accuracy vs. Noise Level by Compression Method

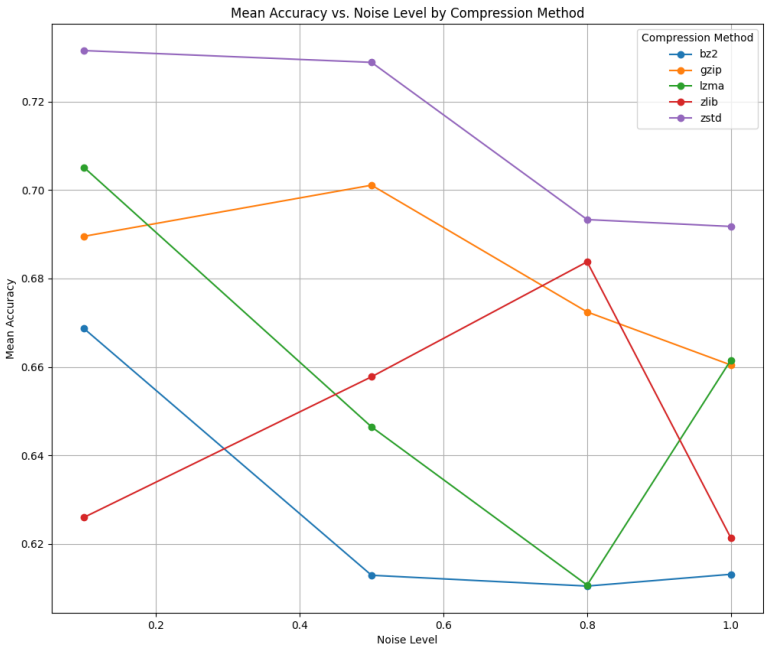

Figure 5: Mean Accuracy vs. Noise Level by Compression Method

The plot in Fig. 6 illustrates the mean accuracy of music genre identification across different noise levels for each compression method. The analysis is as follows:

Noise Level 0.1

At the lowest noise level of 0.1, all compression methods exhibit high mean accuracy. **Zstd** shows the highest accuracy, followed by **Lzma** and **Gzip**. **Zlib** has the lowest accuracy, although it still performs reasonably well.

Noise Level 0.5

At a noise level of 0.5, there is a slight decline in accuracy for most methods. **Gzip** and **Zstd** maintain high accuracy, while **Lzma** and **Zlib** show moderate decreases. **Bz2** experiences a noticeable drop in performance.

Noise Level 0.8

At a noise level of 0.8, accuracy continues to decline. **Zlib** shows an improvement compared to noise level 0.5, indicating some robustness. **Gzip** and **Zstd** still perform well, though slightly reduced. **Lzma** and **Bz2** have the lowest accuracy, indicating higher sensitivity to noise.

Noise Level 1.0

At the highest noise level of 1.0, the trend of declining accuracy stabilizes for some methods. **Zstd** and **Gzip** maintain the highest accuracy. **Lzma** shows a slight recovery, and **Bz2** and **Zlib** remain the lowest but with minimal further decline.

In conclusion, as we expected with more noise inserted the worse our model will perform.

5.3 Accuracy Distribution Analysis

The plot in Fig. 6 presents the accuracy distribution of the music genre identification task across different noise levels and compression methods. This analysis aims to understand how the accuracy of genre identification is affected by varying levels of noise and to compare the performance of different compression methods under these conditions. The box plots illustrate the spread of accuracy values, highlighting the median, interquartile range, and potential outliers for each combination of noise level and compression method.

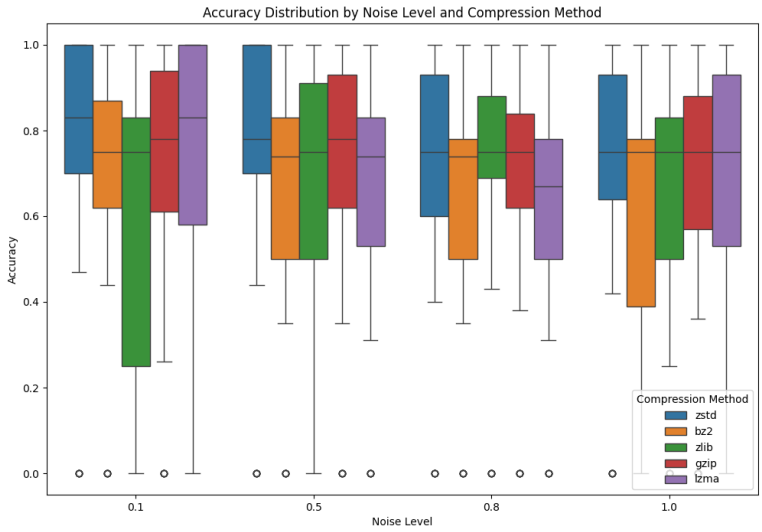

Figure 6: Box-plot showing the accuracy distribution from the different compressors

5.3.1 Noise Level 0.1

At a noise level of 0.1, the accuracy distribution is relatively high for all compression methods. The median accuracy values are generally above 0.6, indicating that the models perform well with minimal noise interference. Specifically: - **Zstd** and **lzma** show the highest median accuracies, with **lzma** demonstrating a more consistent performance as indicated by a smaller interquartile range (IQR). - **bz2** and **gzip** exhibit slightly lower median accuracies but still maintain reasonable performance. - **zlib** shows a wider range of accuracy values, suggesting some instability in performance at this noise level.

5.3.2 Noise Level 0.5

Increasing the noise level to 0.5 leads to a noticeable decrease in accuracy across all compression methods. The distributions show more variability, indicating less consistent performance: - **gzip** and **lzma** continue to perform relatively well, with higher median accuracies compared to other methods. - **Zstd** and **bz2** experience significant drops in their lower quartiles, reflecting poor performance in several instances. - The overall spread of accuracy values increases, highlighting the growing impact of noise on the model's ability to correctly identify music genres.

5.3.3 Noise Level 0.8

At a noise level of 0.8, the trend of decreasing accuracy persists. The medians drop further, and the IQRs widen, indicating greater variability in the results: - **zlib** and **lzma** maintain better performance, with **lzma** showing relatively high median accuracy and consistency. - **bz2** and **Zstd** continue to struggle, with further declines in accuracy and increased performance variability. - The higher noise level exacerbates the challenges in accurately identifying music genres, as seen by the broader distributions and lower median accuracies.

5.3.4 Noise Level 1.0

At the highest noise level of 1.0, the accuracy distributions show a further decline in performance for all compression methods: - **lzma** and **gzip** still manage to maintain higher median accuracies, suggesting some robustness to noise. - **bz2** shows the lowest performance, with the widest IQR and numerous outliers, indicating high variability and poor reliability under these conditions. - Overall, the increased noise level significantly impacts the accuracy of genre identification, with all methods showing decreased performance and greater variability.

5.4 Effect of Data Partition

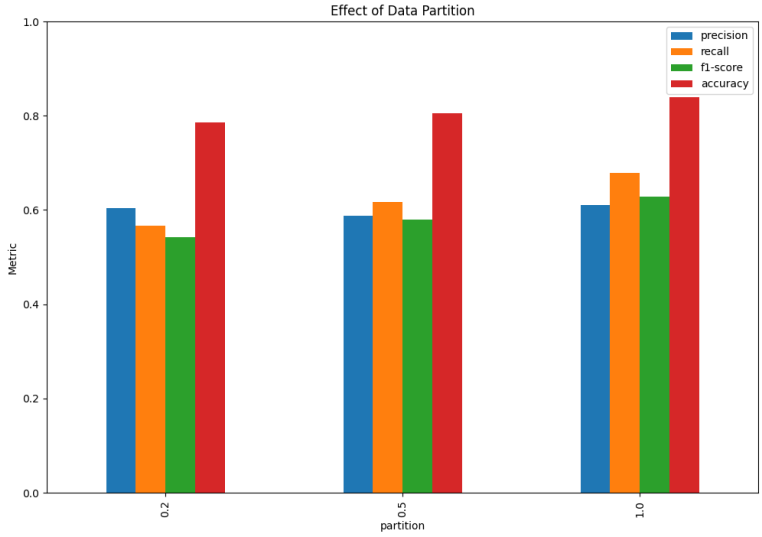

Figure 7: Effect of data partition

The plot in Fig. 7 illustrates the performance metrics of precision, recall, f1-score, and accuracy for different data partition ratios. These partitions represent

the percentage of the music data analyzed to determine the genre. The analysis is detailed as follows:

Partition 0.2

With a partition ratio of 0.2, the model shows balanced precision and recall, with a moderate f1-score and accuracy. This indicates a stable performance, although there is some room for improvement in recall and f1-score to achieve better overall performance.

Partition 0.5

Increasing the partition ratio to 0.5 leads to an improvement in recall and f1-score, as well as a slight increase in accuracy. This suggests that using more data for analysis helps the model to identify genres more accurately and with fewer errors.

Partition 1.0

At the highest partition ratio of 1.0, the model achieves the best performance metrics across all categories. The precision, recall, and f1-score are the highest, and the accuracy improves significantly. This indicates that analyzing the entire dataset provides the model with the most comprehensive information, leading to the most accurate genre identification.

5.5 Performance by Genre (Test Data)

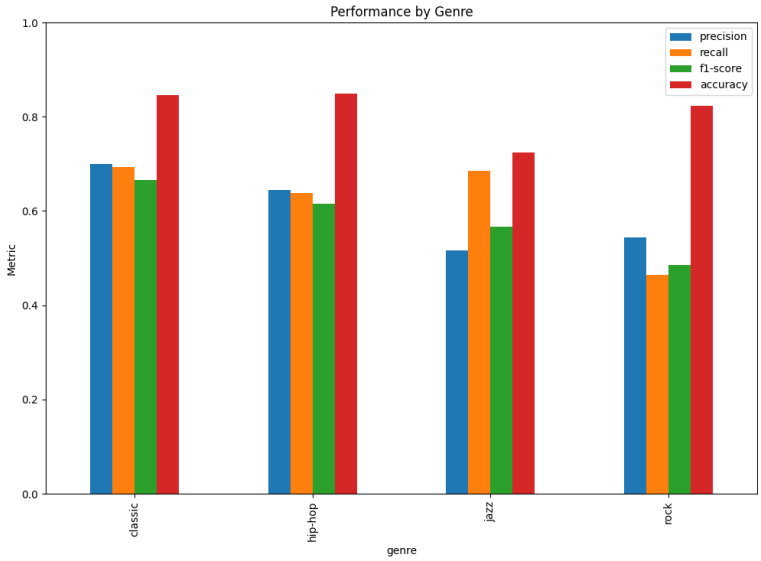

Figure 8: Performance by Genre (Test Data)

The plot in Fig. 8 shows the performance metrics of precision, recall, f1-score, and accuracy for each music genre using the test data. The analysis is summarized as follows:

Classic

Classic music exhibits high accuracy and balanced performance across precision, recall, and f1-score, indicating that the model performs well in identifying this genre.

Hip-hop

Hip-hop also shows high accuracy with relatively balanced precision and recall, but a slightly lower f1-score compared to classic, suggesting good identification performance with minor room for improvement.

Jazz

Jazz has the highest recall but lower precision and f1-score, indicating the model is better at identifying true positives but also has more false positives, resulting in a moderate overall accuracy.

Rock

Rock shows the lowest recall and f1-score among the genres, with moderate precision and accuracy, suggesting the model struggles more with this genre compared to others.

5.6 Performance by Genre (Unseen Data)

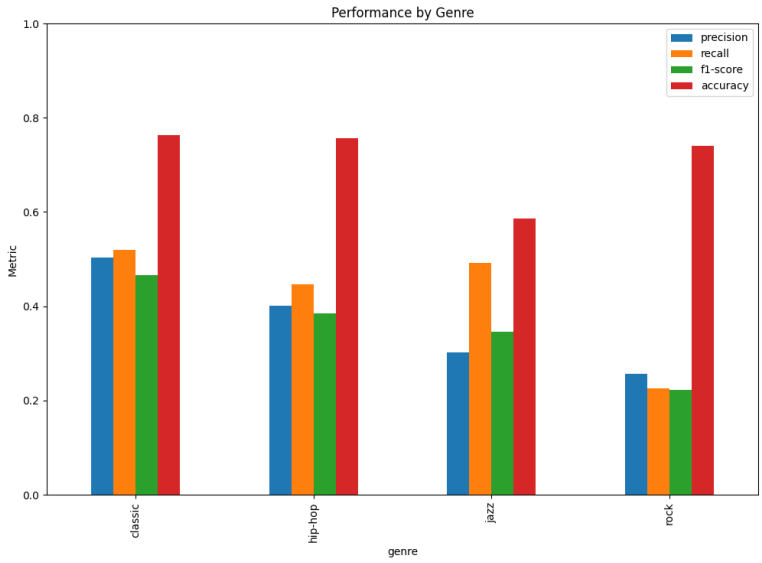

Figure 9: Performance by Genre (Unseen Data)

The plot in Fig. 9 shows the performance metrics of precision, recall, f1-score, and accuracy for each music genre using unseen data. The analysis is detailed as follows:

Classic

For unseen data, the model's performance on classic music shows a drop in all metrics compared to test data, but it still maintains relatively higher accuracy and balanced performance.

Hip-hop

Hip-hop sees a decrease in all performance metrics, indicating that the model has more difficulty identifying this genre correctly when exposed to new, unseen data.

Jazz

Jazz experiences a significant drop in precision and f1-score, with recall remaining moderately high. This suggests the model identifies some true positives but also a higher rate of false positives, leading to lower overall accuracy.

Rock

Rock shows the lowest performance metrics for unseen data, with substantial decreases in precision, recall, f1-score, and accuracy. This indicates the model struggles significantly with this genre when faced with new data.

6 Conclusion

This study validates the use of the information distance function for music and genre analysis. We simplified the similarity process by leveraging the Normalized Compression Distance (NCD) in our system. The results show that using NCD, with compressors such as bz2, gzip, lzma, zstd, and zlib, effectively differentiates between various music genres.

While challenges such as computational complexity and varying performance across different compressors persist, the outcomes suggest that the distance algorithmic functions hold potential for practical applications. Future research could explore integration with more complex compression programs to enhance both accuracy and efficiency.

In summary, our findings meet the academic requirements and contribute to the development of efficient, theory-driven music and genre similarity detection systems.

References

[Pin24] Armando J. Pinho. Getmaxfreqs. Elearning https://elearning.ua.pt/mod/resource/view.php?id=290156, May 2024.

[spoa] 100 greatest hip-hop. https://open.spotify.com/album/7gODFQhnrWl0teeQ7SoLzT?si=Pr1DpEbPSLe8f9zZh8wqZg. Accessed: 2024-06-10.

[spob] Jazz24 top 100. https://open.spotify.com/playlist/4f5dGFu8UzyuA5qvY3HZhk?si=6df2d0551a8c4282. Accessed: 2024-06-10.

[spoc] Top 100 classical music. https://open.spotify.com/playlist/30eSdgbIrWvRYb74flsFFI?si=2256506128014629. Accessed: 2024-06-10.

[spod] Top 100 most popular roc. https://open.spotify.com/playlist/7DgPQwzEoUVfQYBiMLER9Z?si=fc7e85e072a04b04. Accessed: 2024-06-10.

[spo24] spotDL. spotdl: Spotify downloader, 2024. Version 4, GitHub Repository.