Identifying Songs using Information Distance

Bruna Simões *brunams21@ua.pt*
Daniel Ferreira *danielmartinsferreira@ua.pt* Tiago Carvalho *tiagogcarvalho@ua.pt*

***Abstract*—The normalized compress distance (NCD) is an universal measure which uses compression methodologies to find the least distance between two objects, that is, how similar they are between themselves, by compressing information of an object, using the other. Therefore, a lesser normalized compressed distance denotes a greater similarity between two objects.**

Nowadays, most song detectors use technologies such as audio fingerprinting and spectogram analysis to match a sample and identify it in a database. However, it is proposed that an approach using the normalized compressed distance between a given sample and a song in a database can be used as a method to detect the song it belongs to, since the sample will, in theory, have a lesser value for the songs it is more similar to.

***Index Terms*—Normalized Compressed Distance, Normalized Information Distance, Kolmogorov Complexity, Song Identification**

I. Introduction

NORMALIZED compressed distance is a metric used to denote the similarity between two different objects by comparing their compression and the compression of their concatenation. This distance is based on the notion of normalized information distance, which uses the Kolmogorov complexity. This Normalized Information Distance is given by the formula:

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

where $K(x)$ is the Kolmogorov complexity of x, and $K(x, y)$ is the Kolmogorov complexity of x given y, representing the length of the shortest program that computes x when y is provided as input, that is, as additional information.

This suffers, however, from the problem of non-computability due to the Kolmogorov complexity parameters, in this case, the halting problem, which consists on determining the shortest program that outputs a given string, associated with Turing Machine's halting problem, in which it is impossible to determine when a Turing machine running a given algorithm will stop.

Therefore, in order to deal with this problem, it is used the Normalized Compressed Distance approximation, which uses the compression of strings to determine their distance, that is, their similarity, between x and y. It is given by:

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

where $C(x)$ denotes the number of bits required by the compressor C to represent x, and $C(x, y)$ denotes the compression of x concatenated with y using the same compressor C. In this case, if x and y are not similar at all, then the normalized compressed distance will be close to $C(x) + C(y)$.

Therefore, by analyzing each song, y, and the segment to be detected x, we can calculate the normalized compression distance for all segments and, therefore, infer about the least normalized compressed distance, which will correspond to the song it is more similar to.

However, for this strategy to be efficient, both x and y will have to be converted to their set of predominant frequencies, that is, their signature, which is mostly unique to each song.

Therefore, a converter from songs to their corresponding maximum frequencies is needed, and the compressor will compress not the songs themselves, but their corresponding signature, with x being the signature of the song segment to be analyzed, y being the signature of the song that the segment is being compared to, and (x, y) being the concatenation of the files where the signatures will be stored.

II. Methodology

The developed methodology consists of a Python program which analyzes a file to be compressed, within a certain folder. Additionally, given a list of song signatures, the normalized compressed distance will be calculated for all elements of the signatures folder, including their concatenation with the signature of the song excerpt, and posterior compression using several standard compressors. The output will be a list, which denotes all songs of the dataset, and their normalized compressed distance facing the song segment being analyzed.

A. Song dataset

For the implementation to be reliable, several songs were used. The implemented dataset consists of 35 different songs, divided into two main genres: ”Classical” and ”Modern”.

The used songs can be visualized in Table II.

B. Song conversion

Upon obtaining the specified dataset, its conversion to the `.wav` format was made, in order to then obtain the set of signatures.

To do so, a shell script was created, which iterates through each `.mp3` song in the designated `/songs` folder (which stores all the songs previously mentioned), and converts each one to the designated `.wav` file, while changing the sample rate to a maximum of 44100Hz, which is the maximum accepted by the program that converts the `.wav` files to their corresponding signatures. For this conversion, the `Sound eXchange` (or `sox`) [1] program was used.

After converting all files to the `.wav` format, limiting the sample rate to the defined maximum and placing them in a

[1]https://sourceforge.net/projects/sox/

new folder `/converted_songs`, the script starts iterating again through the created folder of `.wav` files and converting them to their respective set of maximum frequencies, which in this case, work as a fingerprint, specific for each song.

To do so, the provided `GetMaxFreq` program, written in C++, was used, which given a file, transforms it to a signature of maximum frequencies and stores them in the designated signatures folder.

C. Song analysis and identification

After pre-processing the database and obtaining the dataset with the signatures of the songs used, it is possible to obtain the results of the normalized compressed distance, given a sample `.mp3` song, and infer the song that is more similar to the provided sample.

This verification is made assuming that the sample to be tested is already parsed to its maximum frequencies, and used as an input to the python program. However, if the file is yet to be converted, it is possible to convert it automatically, using the `convert_song_to_be_analyzed.sh` shell script, which given all samples in the `/analyze` folder, converts them and parses them to their signature counterparts.

Then, after all files are converted to their signatures frequencies, they can be used by the main python program, `generate_compression.py`.

The program takes several arguments, which can be divided into two categories:

- Arguments that affect how the program works.
  - **-c** or **–compress**:
  - **-gzip**: Compress the given sample using the `gzip` standard compressor.
  - **-bzip2**: Compress the given sample using the `bzip2` standard compressor.
  - **-lzma**: Compress the given sample using the `lzma` standard compressor.
  - **-zstd**: Compress the given sample using the `zstd` standard compressor
  - **-folder-test**: Folder where the file to be analyzed, more specifically, its signature, is located. It's default is `analyze`
  - **-test-file**: Name of the file to be analyzed, more specifically, its signature. Its default value is `signature_test`.
  - **-dataset-signatures**: Folder where the signatures are located. Its default value is `song_signatures/`.
  - **-dataset-compressions**: Folder where the compressed signatures are located. Its default value is `compressed/`
- Arguments used for testing purpose, which do not affect the internal operations. These arguments are used when writing tests to `.csv` files to be analyzed later, and indicate additional information about the files being tested.
  - **-noise-type**: Type of noise used in the song segments. Can be either brown-noise, pink-noise or white-noise.
  - **-noise-percentage**: Percentage of noise applied to the song segments.
  - **-test-start**: Time, in seconds, where the song segment being tested started in the original song it belongs to.
  - **-test-duration**: Duration of the song segment.
  - **-csv-file**: Name of the `.csv` file the results should be written to.

  The program allows four different compressors: `gzip`, `bzip2`, `lzma` and `zstd`, where at least one of these compressors must be specified in the program arguments. However, more than one compressor can be submitted as input for the program, with the results displaying the normalized compressed distance being divided for each compressor used.

D. Chromaticism

In Western music, the key in which a song is in determines which notes are permissible for use. Chromaticism is a technique in which notes foreign to the native key are used. In terms of compression, this means that the universe of possible frequencies is expanded, and, in theory, makes it harder to compress a section with sporadic foreign notes (Figure 1).

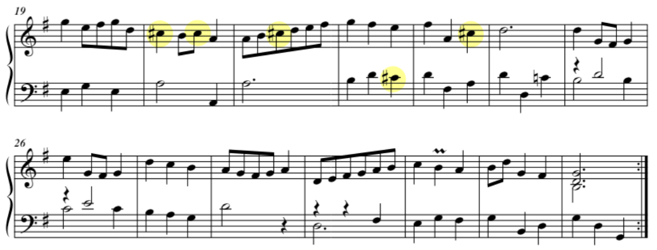

Fig. 1: Excerpt from Menuet in G Major by J.S. Bach, with foreign notes anotated

III. Results

The analysis was conducted to test the effect of various parameters of compression on the performance metrics. For this, the overall metrics were evaluated across the chosen 4 compressors and the effects of the independent variables were studied on the compression value, allowing us to conclude if a given variable would or would not impact the metrics.

As seen in Figure 2, when exposed to an arbitrary sample, all chosen compressors have satisfactory performance, with *gzip* being notoriously out-of-tune with the rest, lagging slightly behind in all measured aspects.

In addition to this, Figure 3 provides a glimpse of the rest of the analysis: the type of compressor used has a significant impact on the compression obtained and, therefore, on the reliability of this method as a song identification tool.

A. Noise

Various types of noise were introduced with different prevalences in order to test the reliability of this solution in more life-like scenarios. In general (Figure 4) the prevalance of noise has little to no impact on the NCD result, regardless of the type of noise applied (Figure 5) or compressor used (Figure 6).

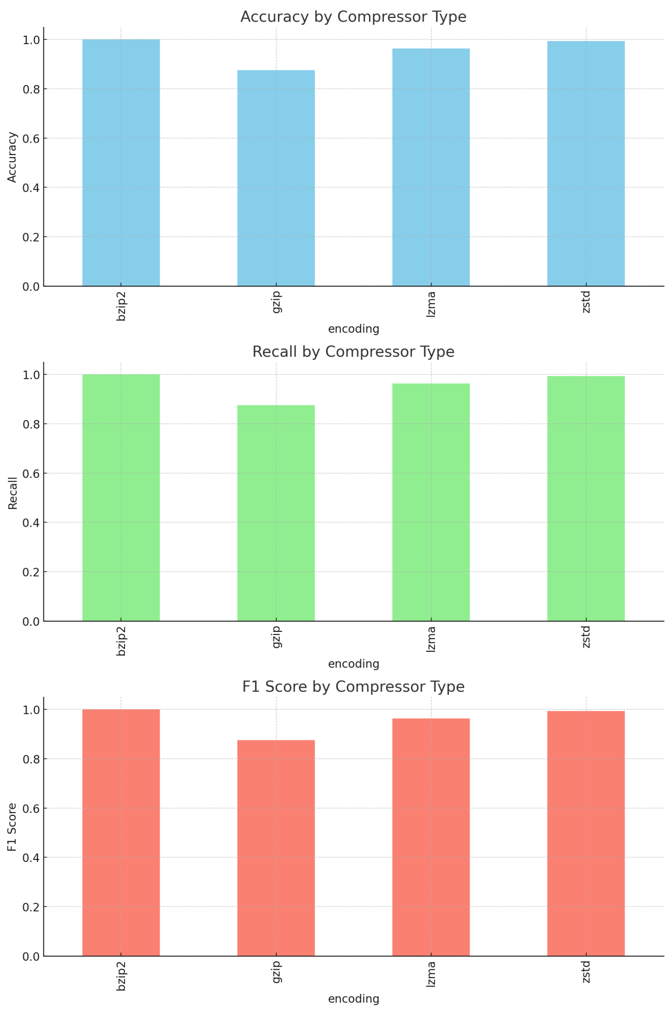

Fig. 2: Overall metrics across all samples tested

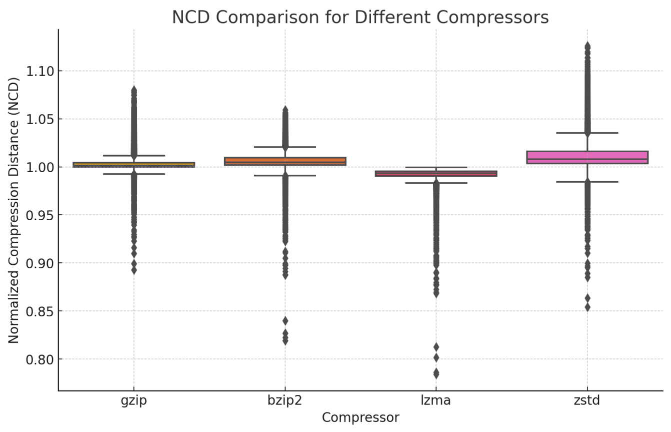

Fig. 3: Compression value distribution across compressors

B. Duration

Regarding the duration of the analyzed sample, again, the results are compressor dependent. In Figure 7, each compressor has its own behaviour as sample duration increases, for instance, *zstd* shows a NCD increase while *lzma* is stable.

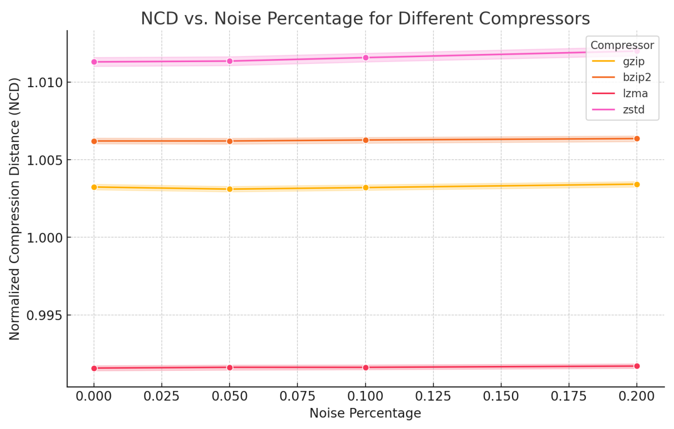

Fig. 4: Variance of the the NCD value in function of the percentage of the excerpt covered with noise.

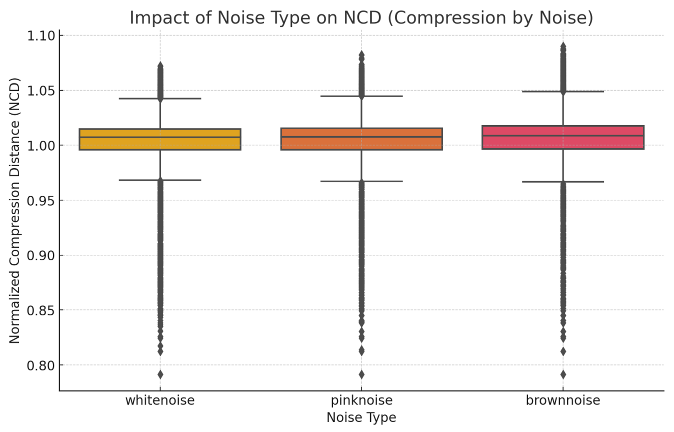

Fig. 5: Compression value distribution when applying different types of noises.

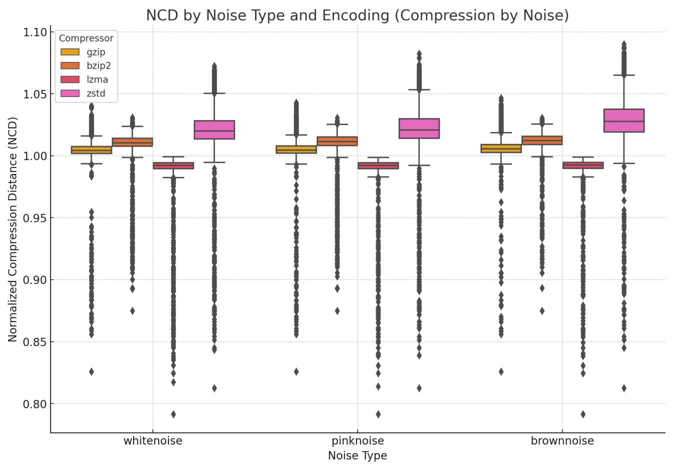

Fig. 6: Compression value distribution when applying different types of noises across all compressors.

C. Genre

When it comes to the Modern/Classical split, Classical entries are harder to compress (Figure 8), as expected, since there is, general, more elements (i.e. frequencies and sound

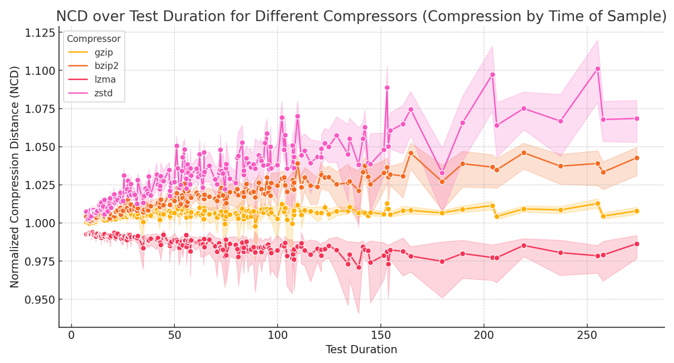

Fig. 7: Overall metrics across all samples tested

signatures) to take into account. Because of this they are also more prone to be noise-sensitive, due to the likelihood of the noise obstructing a used frequency is higher (Figure 9). Even though in terms of start time, both classes observe similar behaviours (Figure 11), duration has a significant effect on Classical entries (Figure 10), again, likely due to the more heterogeneous frequency and melodic composition of this type of music.

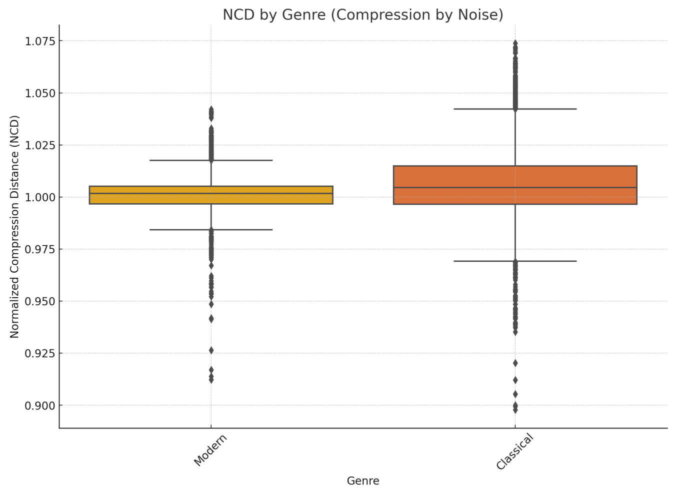

Fig. 8: Overall metrics across all samples tested

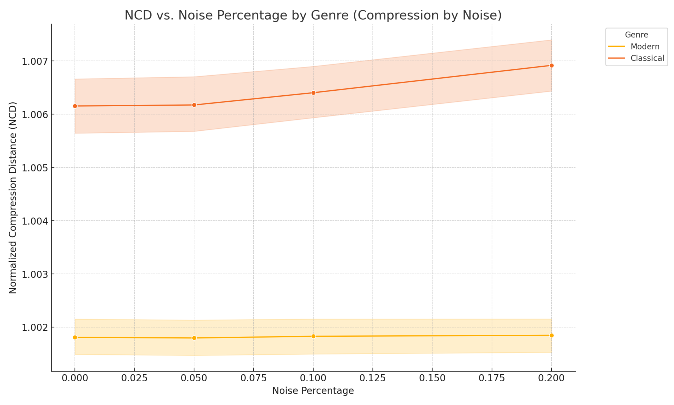

Fig. 9: Overall metrics across all samples tested

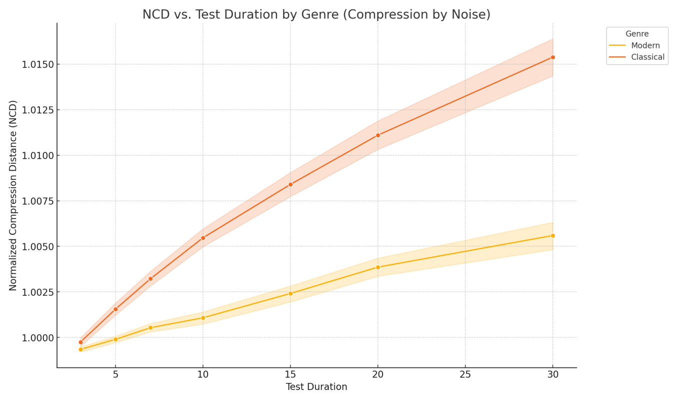

Fig. 10: Overall metrics across all samples tested

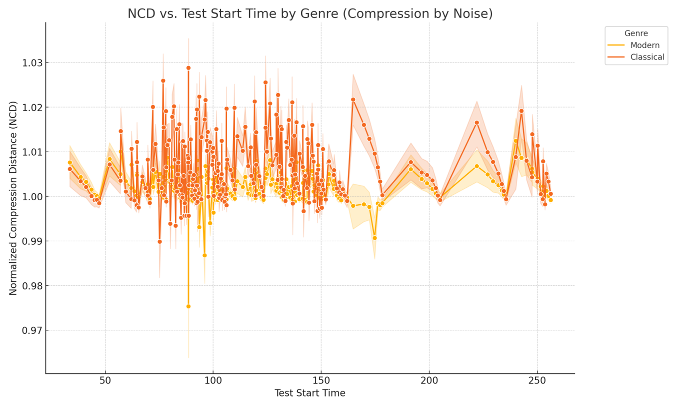

Fig. 11: Overall metrics across all samples tested

D. *Chromaticism*

The use of chromaticism was also studied. Metrics for chromaticized and non-chromaticized tests are displayed in Table 12. Surprisingly, the differences are statistically insignificant, even though most compressors (except *gzip*, which was the worst overall performer) are actually more efficient when chromaticism is in place. This might be due to the fact that the foreign notes in use are of frequencies that can be approximated to in-key notes, making compression more efficient, while in-key notes are, by definition, a certain interval apart, making their frequencies clearly distinct.

TABLE I: Consolidated Compressor Performance Metrics

Encoding	Accuracy (Not Chrom)	Recall (Not Chrom)	F1 Score (Not Chrom)	Precision (Not Chrom)	Accuracy (Chrom)	Recall (Chrom)	F1 Score (Chrom)	Precision (Chrom)
bzip2	0.740000	1.0	0.850575	0.740000	0.753333	1.0	0.859316	0.753333
gzip	0.623333	1.0	0.767967	0.623333	0.618333	1.0	0.764161	0.618333
lzma	0.503333	1.0	0.669623	0.503333	0.506667	1.0	0.672566	0.506667
zstd	0.656667	1.0	0.792757	0.656667	0.676667	1.0	0.807157	0.676667

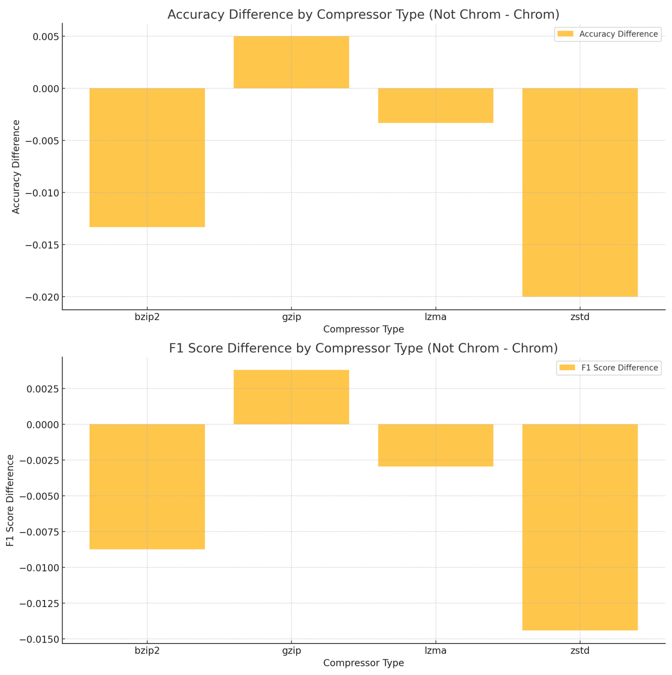

Fig. 12: Overall metrics across all samples tested

IV. Conclusion

This report investigates the feasibility of using Normalized Compressed Distance (NCD) as a metric for song identification. The traditional methods for song identification, such as audio fingerprinting and spectrogram analysis, rely on matching samples to a database using various features of the audio signal. The proposed method leverages NCD to measure the similarity between the compressed representations of audio segments and songs.

The methodology involved converting songs to their frequency signatures and using various compression algorithms (*gzip*, *bzip2*, *lzma*, and *zstd*) to compute the NCD. The choice of compressor significantly impacted the song identification process, with *gzip* performing the worst and *bzip2*, *lzma*, and *zstd* providing satisfactory results. Noise had little impact on NCD results, but the duration of the sample influenced NCD values differently for each compressor. Classical music, due to its complex structures, was more challenging to compress and more sensitive to noise and duration variations.

Additionally, the study examined chromaticism's impact on compression. Surprisingly, the differences between chromatic and non-chromatic tests were statistically insignificant, with most compressors performing better with chromaticism due to the approximation of out-of-key notes to in-key notes.

Future work should investigate the impact of the distance of these chromaticized notes to the native key and at which point due they aid compression.

TABLE II: Dataset used

Signature	Artist	Song
	Modern Music	
`ACDC_highway_to_hell`	AC/DC	Highway to Hell
`GNR_dunas`	GNR	Dunas
`aerosmith_dream_on`	Aerosmith	Dream On
`ai_coracao`	Mimicat	Ai Coração
`bee_gees_stayin_alive`	Bee Gees	Stayin' Alive
`clair_de_lune`	Claude Debussy	Clair de Lune
`coldplay_viva_la_vida`	Coldplay	Viva la Vida
`daft_punk_harder_better_faster_stronger`	Daft Punk	Harder, Better, Faster, Stronger
`daft_punk_one_more_time`	Daft Punk	One More Time
`djo_end_of_beginning`	Djo	End of Beginning
`elton_john_im_still_standing`	Elton John	I'm Still Standing
`eurythmics_sweet_dreams_are_made_of_this`	Eurythmics	Sweet Dreams (Are Made of This)
`guns_n_roses_sweet_child_o_mine`	Guns N' Roses	Sweet Child O' Mine
`iolanda_grito`	Iolanda	Grito
`message_in_a_bottle`	The Police	Message in a Bottle
`metallica_master_of_puppets`	Metallica	Master of Puppets
`muse_supermassive_black_hole`	Muse	Supermassive Black Hole
`nemo_the_code`	Nemo	The Code
`nirvana_heart_shaped_box`	Nirvana	Heart-Shaped Box
`nirvana_smells_like_teen_spirit`	Nirvana	Smells Like Teen Spirit
`queen_bohemian_rhapsody`	Queen	Bohemian Rhapsody
`so_um_beijo`	So Um	Beijo
`the_beatles_here_comes_the_sun`	The Beatles	Here Comes the Sun
`the_cranberries_zombie`	The Cranberries	Zombie
	Classical Music	
`ballade_no1_g_minor_op23`	Frédéric Chopin	Ballade No. 1 in G minor
`Beethoven_fur_elise`	Ludwig van Beethoven	Für Elise
`clair_de_lune`	Claude Debussy	Clair de Lune
`etude_op10_no12_c_minor_revolutionary`	Frédéric Chopin	Étude Op. 10, No. 12 in C minor Revolutionary
`fantaisie_impromptu`	Frédéric Chopin	Fantaisie-Impromptu
`gymnopedie_no_1`	Erik Satie	Gymnopédie No. 1
`gymnopedie_no_2`	Erik Satie	Gymnopédie No. 2
`liebestraume`	Franz Liszt	Liebesträume
`minuet_g`	Johann Sebastian Bach	Minuet in G
`moonlight_sonata_1`	Ludwig van Beethoven	Moonlight Sonata (1st Movement)
`nemo_the_code`	Nemo	The Code
`prelude_c_major`	Johann Sebastian Bach	Prelude in C Major
`symphony_no5_C_minor_op67`	Ludwig van Beethoven	Symphony No. 5 in C minor, Op. 67