# Dynamic Range Compression and Its Effect on Music Genre Classification

Arlyn Reese Madsen III

September 2024

## 1 Introduction

Can audio compression improve music genre classifier accuracy? Compression, also known as Dynamic Range Compression (DRC), brings the loudest and quietest parts of an audio signal closer together. It is called dynamic because the signal is only affected after passing a specific threshold in decibels. Thus everything above this threshold is compressed, and everything below the threshold remains untreated. Suppose you are recording vocals and you scream and whisper in the same take. Once compression is added, the screaming will be reduced in volume and the whispering will be brought up in volume (once makeup gain is added). As you can see in Figure 1, the louder parts of the signal are decreased and the quieter parts of the signal (like the 4th transient) are increased due to makeup gain. Thus the resulting signal is more even and balanced since now the peaks and valleys of the signal are closer together.

When making music, I use a compressor to make my beats sound clearer and more dynamic. Perhaps a compressor will make a music genre classifier able to "read" the songs clearer. Will adding compression to the test set increase the accuracy of the classifier compared to the uncompressed test set? First, I trained the support vector machine classifier with the base song data. Then I applied different compression effects to the base song data. Finally, I acquired accuracy measurements for each test set with a different compression setting. Then the accuracy measurements were compared to see which of the 90 compression settings increases the accuracy the most.

## 2 Dynamic Range Compression

We follow the paper "Digital Dynamic Range Compressor Design - A Tutorial and Analysis" by Giannoulis, Massberg, and Reiss for theory on audio compression. Dynamic range compression is the method of changing the dynamic range of a signal to a smaller range. More precisely a compressor is a "nonlinear time-dependent system with memory". Thus it cannot be described by linear equations, the system response changes over time, and the system retains memory of the past behavior. Ultimately, we are changing the volume of the signal dynamically.

Let's discuss the different settings on a compressor. First, the **threshold** must be chosen in decibels. The threshold defines at what point compression starts. Any part of the audio signal above the threshold will be reduced/compressed according to the **ratio**. The ratio controls the input/output ratio for signals over the threshold. The ratio is unitless. If the threshold is 5 decibels, the ratio is 3, and the signal is 8 decibels (i.e. 3

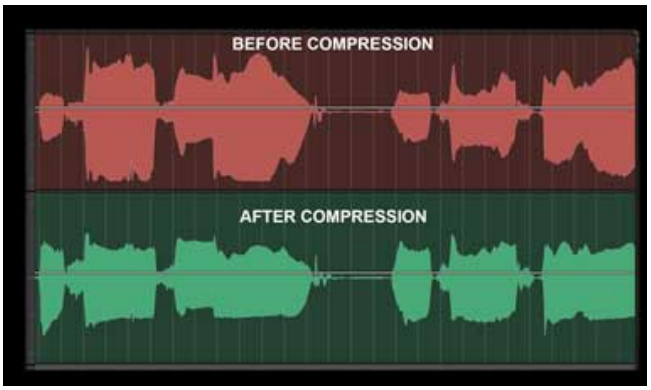


Figure 1: Before and After Compression

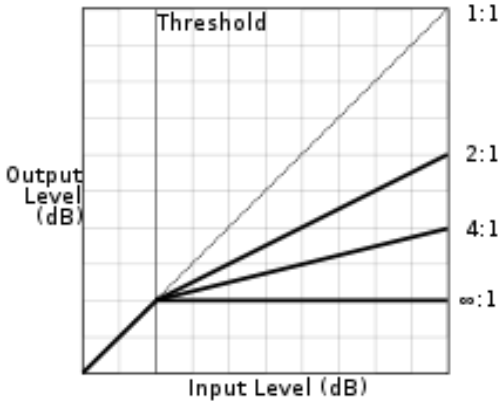


Figure 2: Compression Ratio and Threshold Graph

decibels over the threshold), the resulting output will be 6 decibels. A graphical representation of the threshold and ratio is shown in figure 2. The **kneewidth** in decibels controls whether the bend in the compression characteristic is a sharp change or a more smooth and rounded one. The bend in the compression characteristic is the change from the ratio being 1 to the specified ratio. For a soft or rounded knee, the compression effect is less perceptible. **Attack and release times** determine how quickly a compressor kicks in and returns to the base signal. The attack time determines how fast the compressor kicks in once the threshold is passed. The release time determines how fast the signal returns to the original volume once the signal falls below the threshold. The **makeup gain** is used to bring the signal back up to near the input volume since the compressor is reducing the gain overall.

Now let's look at the mathematics of a compressor. Let $y$ be the output signal, $x$ be the input signal, $T$ be the threshold, and $R$ be the ratio. Once the signal reaches above the threshold value, the signal is attenuated as per the ratio. The ratio is defined as

$$R = \frac{x - T}{y - T} \tag{1}$$

Thus the following piecewise function can be used to describe the output signal:

$$y = \begin{cases} x, & \text{if } x \leq T \\ T + (x - T)/R, & \text{if } x > T \end{cases} \tag{2}$$

# 3 Music Genre Classification

## 3.1 Raw Data

For music genre classification we use the GTZAN dataset available on Kaggle. This is a dataset of 1000 thirty-second song clips with 10 genres including blues, classical, country, disco, hip hop, jazz, metal, pop, reggae, and rock.

## 3.2 Tranformed Data

We determine which compressor setting increases the music genre classifier accuracy the most. There are three base transformations: high compression, medium compression, and low compression. High compression is -20db threshold, 8 ratio value, 0 db knee width, 1 ms attack time, 10 ms release time, and 7db of makeup gain. This is considered high compression because a significant part of the signal will pass the low threshold, the ratio is very high so for every 8 decibels of input, there will only be 1 decibel of output, and there is a very sharp knee, and there is a very fast attack and release time. The medium compression setting is -10 db threshold, 5 ratio value, 5 db knee width, 5 ms attack time, 50 ms release time, and 5 db of makeup gain. The low compression setting is -5 db threshold, 2 ratio value, 20 db knee width, 10 ms attack time, 100 ms release time, and 3 db of makeup gain.

For every base transformation, we vary each setting and create a new transformation. For example for the threshold we have 5 values: -40 db, -20 db, -10 db, -5 db, and -3 db. We have 5 values for each setting which gives us a total of 90 unique transformations.

### 3.3 Preprocessing

We must conduct data preprocessing to transform the songs into numbers that are readable by a support vector machine classifier. There are several features we use for each song including 13 MFCCs, tempo, zero crossings, and 5 tonal centroids.

**Mel frequency cepstral coefficients** (MFCCs) are one of the most widely used features in music genre classification tasks. MFCCs can be used for audio similarity features and timbral descriptions for music. Timbre is the character of a sound and is represented by overtones on a frequency spectrum. A guitar and a piano sound different even if they play the same frequency say 440Hz. MFCCs will be able to differentiate the guitar note from the piano note. This is useful for music genre classification because the overall timbre will be different for a hip-hop song versus a jazz song. There are several steps in computing MFCCs:

1. Take the discrete Fourier transform of a signal.
2. Take the log amplitude of the power spectrum.
3. Change to mel scale.
4. Take the discrete cosine transform and compute power.
5. MFCCs are the amplitude of the resulting spectrum (cepstrum).

We take the discrete Fourier transform to change from the time domain to the frequency domain. The reason for log amplitude is as follows. Let $y(n) = x(n) * h(n)$ be a signal where $y(n)$ is the convolution of two signals. To separate $y(n)$ we need to use the log amplitude after we take the discrete Fourier transform:

$$log[F(y(n))] = log[F(x(n) * h(n))] = log[F(x(n)) \cdot F(h(n))] = log[F(x(n))] + log[F(h(n))] \tag{3}$$

Thus we have separated the signal $y(n)$. Separating the signal is important because we can separate the timbre from the base frequency for example. Now mel scale is meant to mimic the human auditory experience. Here is the conversion from Hertz to mels:

$$m = 2595\log(1 + \frac{f}{700}) \tag{4}$$

where $f$ has hertz units. This conversion is meant to resemble the fact that humans hear things logarithmically. The discrete cosine transform is a type of Fourier transform. The benefit over a normal discrete Fourier transform is that it only gets real-valued coefficients, reduces the dimensionality, and is computationally inexpensive. Finally, the cepstrum is a play on letters of the word spectrum. We use cepstrum to resemble the spectrum of a spectrum since we take the discrete Fourier transform and then the discrete cosine transform.

**Tempo** tells us how many beats in a minute are in a song. Beats are a unit of time in a song. The more upbeat and uptempo a song is the faster it sounds.

**Zero crossings** tells us the rate at which an audio signal changes from positive to negative amplitude and from negative to positive amplitude i.e. crossing zero. It is useful for classifying percussive sounds. It can be characterized by the equation:

$$Z = \frac{1}{T-1} \sum_{t=1}^{T-1} \chi_{\mathbb{R}_{<0}}(x_t x_{t-1}) \tag{5}$$

where $x$ is the signal of length $T$ and $\chi_{\mathbb{R}_{<0}}$ is the indicator function i.e.

$$\chi_{\mathbb{R}_{<0}}(x) = \begin{cases} 1, & \text{if } x < 0 \text{ and } x \in \mathbb{R} \\ 0, & \text{if } x \geq 0 \text{ and } x \in \mathbb{R} \end{cases} \tag{6}$$

.

**Tonal centroids** and harmonic change detection functions can help us determine when there are chord changes in a song. This can be characteristic of the song itself since chords are a central part of the mood of the song. A tonal centroid for the chord A major is represented by the pitch A and its relation to the circle of fifths, minor thirds, and major thirds. We can represent the six-dimensional tonal centroid vector, $\zeta$, for time frame $n$ by the multiplication of the chroma vector, $\mathbf{c}$, and a transformation matrix $\psi$. The equation is given by

$$\zeta_n(d) = \frac{1}{||\mathbf{c}_n||_1} \sum_{l=0}^{11} \psi(d, l)\mathbf{c}_n(l) \tag{7}$$

for $0 \leq d \leq 5$ and $0 \leq l \leq 11$. Here $l$ is the chroma vector pitch class index and $d$ is which of the six dimensions of $\zeta_n$ are being computed. We can finally define the harmonic change detection function as the rate of change of the tonal centroid signal. This is what determines when chords are changing.

### 3.4 Classification

To classify songs into genres we use support vector machines. Support vector machines (SVMs) are a type of supervised learning method that in our case is used for multi-class classification. SVMs work well in high dimensional spaces, are memory efficient, and are versatile due to the use of kernel functions. For multi-class classification, we implement the one-versus-one approach. For the 10 classes, we must train 45 different 2-class SVMs on all possible pairs of classes and then classify the test points according to which class scores the highest based on correct predictions on all models.

Now let's explain a 2-class SVM classifier. SVMs maximize the margin between the two classes being predicted. The problem at hand can be formulated as a convex optimization problem where we find the global minimum of the objective function. For nonlinear SVM we use a nonlinear transformation to transform the data from the lower dimensional space to a higher dimensional space. We train the SVM in the higher dimensional space and then project the decision boundary back to the lower dimensional space.

We follow section 7.1 in the Sparse Kernel Machines Theory for two-class classification for SVM formulation. Let $x$ and $w$ be a 21-dimensional vector. We use the linear model

$$y(x) = w^T\phi(x) + b \tag{8}$$

where $\phi(x)$ is the fixed feature space transformation and $b$ is the bias parameter. Now we will move to a dual representation of this problem because it allows us to use kernel functions to prevent us from working in the feature space. The training data consists of $N = 1000$ input vectors $x_1, ..., x_N$ with corresponding target values $t_1, ..., t_N$ where $t_n \in \{-1, 1\}$. Now we want to find the best line or hyperplane that separates the two classes. By best, we mean the one that has the smallest generalization error. To do this we define a margin which is the smallest distance between the decision hyperplane and any of the samples. We want to maximize this margin. The perpendicular distance of a data point $x$ from a hyperplane defined by $y(x) = w^T\phi(x) + b = 0$ is given by $\frac{|y(x)|}{||w||}$. We are only interested in solutions that are correctly predicted so we have $t_n y(x_n) > 0$ ,$\forall n$ since $t_n$ and $y(x_n)$ will have the same sign. Thus the distance of a point $x_n$ to the decision hyperplane is

$$\frac{t_n y(x)}{||w||} = \frac{t_n(w^T\phi(x) + b)}{||w||} \tag{9}$$

The margin is the perpendicular distance to the closest point $x_n$ and we want to optimize $w$ and $b$ to maximize the margin. Thus we solve

$$\arg\max_{w,b} \left\{ \frac{1}{||w||} \min_n [t_n(w^T\phi(x_n) + b)] \right\} \tag{10}$$

Now solving this problem is very complex so we will find a dual problem to solve instead. By rescaling $w$ and $b$ we may set

$$t_n(w^T\phi(x) + b) = 1 \tag{11}$$

Thus the data points satisfy the following inequality

$$t_n(w^T\phi(x) + b) \geq 1, \qquad n = 1, ..., N \tag{12}$$

We call this the canonical representation of the decision hyperplane. Since we are maximizing $\frac{1}{||w||}$ we may instead minimize $||w||^2$. Thus we have the optimization problem

$$\arg\min_{w,b} \frac{1}{2}||w||^2 \tag{13}$$

subject to the constraints in equation (12). This is a quadratic programming problem because we are minimizing a quadratic function subject to linear constraints. To solve the optimization problem we must use Lagrangian multipliers $a_n \geq 0$ with one multiplier for each constraint in equation (12). We define the Lagrangian function

$$L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} a_n \left\{ t_n(w^T\phi(x) + b) - 1 \right\} \tag{14}$$

and solve for the stationary points and the Lagrange multiplier to solve the constrained optimization problem. Here $a = (a_1, ..., a_N)^T$. We then set the derivative of $L$ with respect to $w$ and $b$ equal to zero

$$w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \tag{15}$$

$$0 = \sum_{n=1}^{N} a_n t_n \tag{16}$$

Substituting for $w$ and $b$ we get the dual representation of the maximum margin problem. We maximize

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \tag{17}$$

subject to the constraints

$$a_n \geq 0 \text{ for } n = 1, ..., N \tag{18}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{19}$$

where $k(x, x') = \phi(x)^T \phi(x')$. This is called the kernel function. For our experiments, we use the radial basis function for the kernel. Thus we have $k(x, x') = e^{\gamma ||x - x'||^2}$ for a pre chosen $\gamma$. We choose the dual problem because we may use kernel functions which help if there are more dimensions than data points. Finally, we have our formula for predicting a test point using equations (8) and (15).

$$y(x) = \sum_{n=1}^{N} a_n t_n k(x, x') + b \tag{20}$$

### 3.5 Results

We averaged over 10 iterations of train test splits in the data. There was a 3.1% improvement in the compressed test data versus the base data over these 10 iterations for the best-performing compressor setting each time. If we take only one iteration then the compressed data can improve the score by up to 6.6%. Now I want to draw out the settings used for the higher-performing compression settings. For 1000 iterations the ranking was as follows for the best-performing compression settings: LM2, LM1, LT3, MM2, and LA1. The worst-performing compression settings were as follows: LT1, HA1, HRe1, MT1, and HT1. We can see that the best performers consisted mostly of the low compression base while the worst performers had a high compression base. Now we list the top 5 compressor settings:

1. LM2: -5 threshold, 2 ratio, 20 knee width, 0.01 attack time, 0.1 release time, 0 makeup gain
2. LM1: -5 threshold, 2 ratio, 20 knee width, 0.01 attack time, 0.1 release time, -1 makeup gain
3. LT3: -10 threshold, 2 ratio, 20 knee width, 0.01 attack time, 0.1 release time, 3 makeup gain
4. MM2: -10 threshold, 5 ratio, 5 knee width, 0.005 attack time, 0.05 release time, 0 makeup gain
5. LA1: -5 threshold, 2 ratio, 20 knee width, 0 attack time, 0.1 release time, 3 makeup gain

Thus taking one of these compressor settings and using it on a test set before computing the accuracy can improve model performance on average by 3.1%. Now we can improve music genre classifiers by using compressed song data sets. Thus in Spotify's music recommendation system, we could suggest compressing all songs in the best way possible to more accurately predict users recommended songs in a way that increases user retention.

# References


[1] “1.4. Support Vector Machines.” Scikit, scikit-learn.org/stable/modules/svm.html. Accessed 10 Oct. 2024.

[2] ”Best Analog Compressors for Studio-Quality Vocals.” Access Analog, accessanalog.com. Accessed 15 October 2024.

[3] Bishop, Christopher M. Pattern Recognition and Machine Learning. Springer New York, 2006.

[4] “Compressor.” Dynamic Range Compressor - MATLAB, www.mathworks.com/help/audio/ref/compressor-system-object.html. Accessed 10 Oct. 2024.

[5] Dunphy, Tim. ”The Audio Compressor: Breaking Down The Parameters.” MasteringBox, February 6, 2018, www.masteringbox.com/articles/the-audio-compressor-breaking-down-the-parameters. Accessed 14 March 2024.

[6] Harte, Christopher & Sandler, Mark & Gasser, Martin. (2006). Detecting harmonic change in musical audio. Proceedings of the ACM International Multimedia Conference and Exhibition. 10.1145/1178723.1178727.

[7] Giannoulis, Dimitrios, et al. “Digital Dynamic Range Compressor Design—A Tutorial and Analysis.” AES: Journal of the Audio Engineering Society, vol. 60, July 2012.