Ratio Prediction for Dynamic Range Compressors

Wesley Yu, José Díaz Rohena

December 16, 2019

Abstract

Automatic parameter control can improve the usability of dynamic range compressors. MIR techniques can assist in this task, allowing for parameter settings to be predicted using acoustic features. Presently, this task is attempted by training a K-Nearest Neighbors classifier using MFCC and Crest Factor features to recognize the compression ratio applied to an audio file. The classifier performed better than chance on an evenly distributed dataset, but was overall better at distinguishing compressed/not compressed than between different compression ratios. It is hypothesized that a larger and differently conceptualized dataset, one that uses compressed and uncompressed versions of the same audio, would improve performance.

1 Introduction

Dynamic range compression is an important tool for audio mixers and producers. It has many uses—rhythmic enhancement, spectral shaping, harmonics generation, and of course dynamic range management. It can also be a challenging effect to properly implement in an aesthetically pleasing way. Parameter automation can streamline the mixing workflow by allowing for higher level control of a dynamic range compressor. It can also make an effect more robust to poor user input. This type of automation currently appears in products such as iZotope Ozone and Landr, both of which take input audio along with high level user input to set parameters and apply dynamic compression and limiting. For our final project, we set out to accomplish a task that would be crucial to such endeavors—training a nearest neighbors classifier to predict the compression ratio used on an audio clip.

2 Related Work

Automatic/intelligent parameter setting is a richly studied part of the compression literature, encompassing both heuristic and data-driven approaches. One strategy is to use time-domain processing to allow higher level control of parameters [3]. This also allows those parameters to automatically adjust to the signal in real time. Or, users can be given new ways of specifying the compressor's behavior, such as using a graphical display to apply sidechain filtering in the frequency domain [4]. Another way is to allow users to attach semantic descriptors to audio features and the settings used in those contexts [8]. In more abstract applications, features can be extracted from the input signal and a reference source in order to match the compression used in the reference [1]. Neural networks have also been used for this task [5]. Developing a method that uses automatic feature extraction is sought, in this case, towards the development of effects that enable automatic control using high-level parameter input from the user.

3 Dataset

For the present case, the goal was modest: correctly predict the ratio of compression applied to the audio file. The ratio could be either 1:1, 8:1, or 100:1 (a.k.a.: not compressed, compressed, or really compressed). Other parameters, chosen to demonstrate audible compression, remained static. The compressor was adapted from Reiss and McPherson's C++ implementation [7].

Ratio	Threshold	Attack	Release
1:1, 8:1, 100:1	-40.0dB	1.3ms	300ms

The GuitarSet "mic" dataset was chosen for this task [9]. This set has 360 examples. In order to create more examples, the recordings were cut into 5-second segments and compressed at the above ratios. This increased the dataset to 2004 examples. 1332 examples from that set were then randomly chosen, compressed at 8:1 or 100:1, and relabeled as such. This left us with a dilemma: since our dataset of 2004 examples was still relatively small (especially for our chosen task), should we try to maintain an even distribution of different compression ratios, or would it be more effective to include duplicates (all 2004 unprocessed examples, along with our 1332 processed examples) for a total of 3336 examples? We decided to try both options and compare our results. For our evenly distributed dataset, our 1332 processed examples created 666 examples of 8:1 and 666 examples of 100:1 labels. The remaining 672 examples were left unprocessed, and thus given a 1:1 ratio label. In order to have a round total of 2000 examples, 4 examples were randomly taken off of the unprocessed example set, leaving 668 unprocessed examples. Using a ratio of 80:10:10, we split our example set so that the training set would contain 1600 examples, and the validation and test sets would contain 200 examples each. These were randomly selected from our example set, but were done so in a way that left an even distribution of ratio labels. For our unevenly (UE) distributed dataset, there were still 666 examples each of 8:1 and 100:1, but now also 2004 unprocessed examples with 1:1 ratio labels. Using the same ratio as above (80:10:10), our UE training set contained 2670 examples, with our UE validation and UE test sets containing 333 examples each. These were also randomly selected while still preserving an even distribution of compression ratio labels.

4 Features

MFCC and Crest Factor features were used as features to train the classifier. MFCC coefficients were chosen because of their success in similar projects [2], especially in capturing timbral changes. For each example, summary statistics were conducted and normalized across the dataset. Crest factor is proposed as an additional feature to capture a loudness-invariant look at how the dynamic range has been affected. For each example, the crest factor was taken over a 2048 sample window with no overlap.

$$x_{crestdB} = 20 * log10(\frac{x_{peak}}{x_{RMS}}) \tag{1}$$

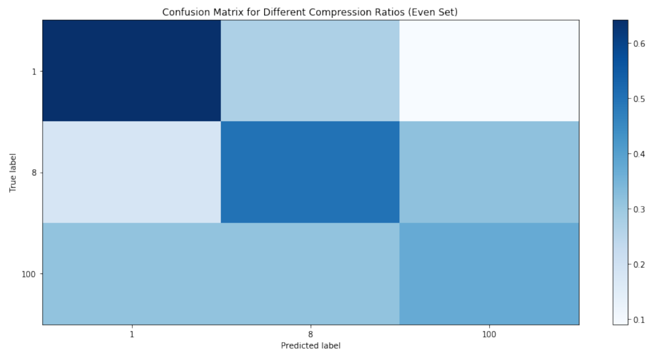
As with the MFCC, summary statistics were conducted for all the crest factor frames in a given example, and normalized across the dataset. Using a this feature slightly improved classification accuracy.

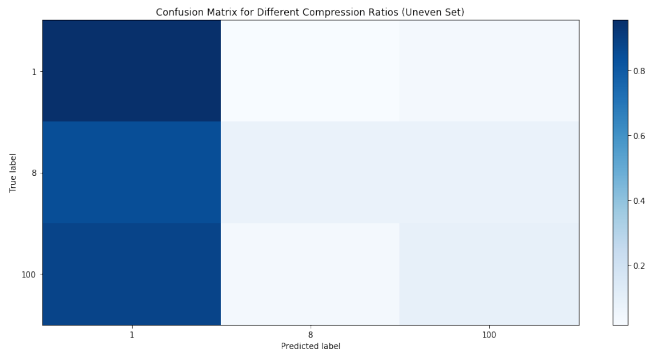
5 Classifier

The model was trained and evaluated using the scikit-learn k-nearest neighbors classifier, using balanced accuracy score to select the best hyperparameter [6]. Features were used to train a nearest neighbors classifier at k = [1, 5, 10, 35, 45, 50, 75, 100].

6 Results

The evenly distributed dataset had a marked improvement in performance over the unevenly distributed (but larger) dataset. For the former, the balanced accuracy score and accuracy score were identical at 0.505, with the highest performing k-nearest neighbor value at k=50. The UE dataset had a balanced accuracy score of 0.373, and an accuracy score of 0.607, at k=10. Though the unbalanced accuracy score of the UE dataset is high, this in itself is deceiving, as shown in the confusion matrices below:





Metric	Even	Uneven
Accuracy	0.505	0.607
Balanced Accuracy	0.5049	0.374
f1	0.50	0.508

As seen in the evenly distributed set, the classifier was relatively accurate in determining the correct ratio label, performing better than chance (0.33). However, the same cannot be said for the UE dataset–the UE classifier almost always predicted a 1:1 ratio regardless of the input. The UE balanced accuracy score demonstrates this – that the UE classifier performed about the same as chance.

This discrepancy could be a result of over-fitting – since the 1:1 ratio (unprocessed) examples were over-represented in our UE dataset, the classifier tended towards that label rather than accurately detecting significant features of compression. This solves our earlier dilemma: that an evenly distributed dataset yields much better classification accuracy.

7 Discussion

The fact that the classifier performed better than chance on the evenly distributed set is promising. However, it seems that it was generally better at distinguishing between compressed/not compressed than it was at correctly identifying the ratio. This is in important distinction. The ratios were selected in order to provide a dramatic contrast. In actual use, the ratios to be distinguished would be much closer, e.g. 4:1 vs 10:1. Based on our results, it seems that the classifier would perform poorly under such conditions.

Expanding the dataset should be considered. Though there are only 360 examples of playing in the dataset, there are 6 different capture methods. Since we are training on acoustic features, the different capture methods would be a great set of extra examples. Traditional augmentations (reverb, etc.) should also be considered as another way to expand the dataset. Furthermore, the rich metadata associated with the GuitarSet were not retained during the creation of the compression dataset. Doing so could offer opportunities to better analyze the classifier's performance, by genre or tempo, for example. Then again, the fact that the examples where cut up into short segments would have made these designations less useful. These segments could be cut according to onsets in order to maximize the compression action seen in each example.

We also could have approached dataset creation differently. Sheng and Fazekas took a relatively small number of examples and applied many different variations of the same transformation to each one. We could have, as they did, taken 50 tracks from guitar set and applied 60 compression augmentations on each one, creating 3000 examples. Using processed and unprocessed versions of the same audio was also successful in other studies [5].

References

[1] György Fazekas Di Sheng. "Automatic Control of the Dynamic Range Compressor Using a Regression Model and a Reference Sound". In: *Proceedings of the 20th International Conference on Digital Audio Effects (DAFx-17)* (Sept. 2017).

[2] György Fazekas Di Sheng. “Feature Selection for Dynamic Range Compressor Parameter Estimation”. In: *AES 144th Convention* (May 2018).

[3] Michael Massberg Dimitrious Giannoulis and Joshua D. Reiss. “Parameter Automation in a Dynamic Range Compressor”. In: *Journal of the Audio Engineering Society* 61.10 (Oct. 2013).

[4] Vesa Välmäki Leo McCormack. “FFT-Based Dynamic Range Compression”. In: *Proceedings of the 14th Sound and Music Computing Conference* (July 2017).

[5] Stylianos Ioannis Mimilakis et al. “Deep Neural Networks for Dynamic Range Compression in Mastering Applications”. In: *Audio Engineering Society Convention 140*. May 2016. URL: `http://www.aes.org/e-lib/browse.cfm?elib=18237`.

[6] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[7] Joshua D. Reiss and Andrew McPherson. *Audio Effects: Theory, Implementation and Application*. CRC Press, Oct. 2014.

[8] Ryan Stables et al. “SAFE: A System for the Extraction and Retrieval of Semantic Audio Descriptors”. In: *15th International Society for Music Information Retrieval Conference* (2014).

[9] Q. Xi et al. “Guitarset: A Dataset for Guitar Transcription”. In: *19th International Society for Music Information Retrieval Conference* (Sept. 2018).