Ear Training by Algorithm
A Study of Supervised and Unsupervised Learning of Musical Intervals

Overview
I began with the goal of applying my music theory knowledge to machine learning. The discipline of ear training – identifying notes, intervals, and chords by ear – is difficult for humans. Can a computer do better? To give my program some standard to compete with, I had a friend test my interval identifying ability, and I scored 67.6% accuracy.

My data consisted of two-note intervals recorded from a piano, each stored in an individual .wav file. I used only files where the notes were within one octave, since this is what a person practicing ear training would normally hear, with the intent of using the full dataset if I later needed more training data.

With this dataset, I implemented two experiments. The first used the unsupervised learning algorithm k-means clustering. I wanted to see if, without context, the algorithm would choose to group intervals according to their types (distinguished on an even-tempered piano as the number of half steps between the two notes), or if not, how it would choose to cluster them. My second experiment was to apply supervised learning using a neural network with labelled data, to see if my program could correctly classify the 12 and/or 7 different types of intervals.

Feature Extraction
I spent the most time in this project trying different methods of feature extraction to try to get meaningful information out of my audio data. The two techniques I tried were various functions from the library pyAudioAnalysis\textsuperscript{ii} and performing Fast Fourier Transform analysis directly, using the fft functions built into NumPy.

My first attempt was to extract all features provided by pyAudioAnalysis, which includes a pitch analysis but also other analysis of the signal, hoping that given a reasonable representation of what a human might hear, the algorithm would decide what features were useful. When this proved unhelpful, I tried using FFT analysis, both taking the raw output and compressing/truncating it in various ways to avoid having 50,000 features, and taking simply the min, max, and mean values of the (absolute values of the) fft output. When this also gave very low accuracy in the neural network, I returned to pyAudioAnalysis, this time extracting only the chroma vector, a 12-feature vector for each frame representing the pitches, and using both the time-averages values directly or considering only the maximum two values or the values over a
certain threshold. This last strategy proved the most successful, although it was not nearly as good as I had hoped.

**Experimental Results**

I. K-Means Clustering

Graph of J vs. K for averaged chroma values:

![Graph of J vs. K for averaged chroma values](image)

The graph does not show a clear elbow, but I used 7 for the clustering as there are 7 distinct types of intervals (some, such as major3 and minor6, invert to one another and contain the same notes).

Cluster output, showing the notes and interval types of each cluster (as sets):

**Files in cluster 0:**

```plaintext
```

**Interval types in cluster 0:**

```
{'major2', 'minor3', 'minor7', 'major6', 'minor2', 'octave', 'major7', 'fifth', 'major3', 'tritone', 'minor6', 'fourth'}
```

**Files in cluster 1:**

```plaintext
```

**Interval types in cluster 1:**

```
{'major2', 'minor3', 'minor7', 'major6', 'minor2', 'octave', 'major7', 'fifth', 'major3', 'tritone', 'minor6', 'fourth'}
```

**Files in cluster 2:**
Observations: The sets of interval types show that every cluster contained every type of interval, so clearly did not group them based on the interval. Examining the actual notes contained in each file, we can see that cluster 3 contains mostly intervals with a root note of Eb 6 or 7, and clusters 4 and 6 have a similar trend with A and C, respectively. Clusters 1, 2, and 5 contain mostly lower notes, and cluster 0 has many of the midrange notes. So the algorithm is drawing some pitch information from the data, but finds the general pitch more important than the relative pitch within each audio clip.

II. Neural Network

My biggest issue with the neural network was that I could not even fit the training data well, and many combinations of hyperparameters resulted in the network converging to low values for the training accuracy, sometimes as low as random-guess accuracy (8.3% when I ran with 12 classes, the slightly better 16.7% for 7 classes). The best training accuracy I achieved was 31%, using the ReLU activation function, the architecture [12, 150, 150, 7], mini batch size 5, alpha between 0.02-0.05, and 100-200 epochs (hoping it
might keep increasing with enough time). Because I never managed to even fit the training data well, I did not proceed to using a validation or test set.

After even direct analysis of the pitch didn't work well, I examined the visualization of the chromagram, provided by pyAudioAnalysis. For example, a file C3F3 gave:

![Chromagram of C3F3](image1)

and A2G#3 gave:

![Chromagram of A2G#3](image2)

We can see that the strongest pitch in each file (D and B, respectively) is not either of the pitches played in that file. So, I can only conclude that my data analysis code wasn't giving me meaningful data anyway.

**Conclusions**

To reiterate, the K-means algorithm, while it did not identify the intervals, had enough sense of pitch to group some files by similar range and some by similar root notes. The neural network did not manage to fit the training data well, but upon examination of the chroma analysis, which gave my best results and logically seemed like the best features, it was giving an inaccurate representation of the data.

I conclude that extracting audio features is more difficult than it seems, given that using what based on quick Google searches appeared to be the most prominent audio analysis library, I could only achieve a fairly low training accuracy. With a better way of extracting different pitches from audio files, I hope that I could achieve much better results. Until then, I will rely on the human ear for my interval identifying needs.

---


\[^ii\] Library by Theodoros Giannakopoulos. Downloaded at: [https://pypi.python.org/pypi/pyAudioAnalysis/](https://pypi.python.org/pypi/pyAudioAnalysis/)