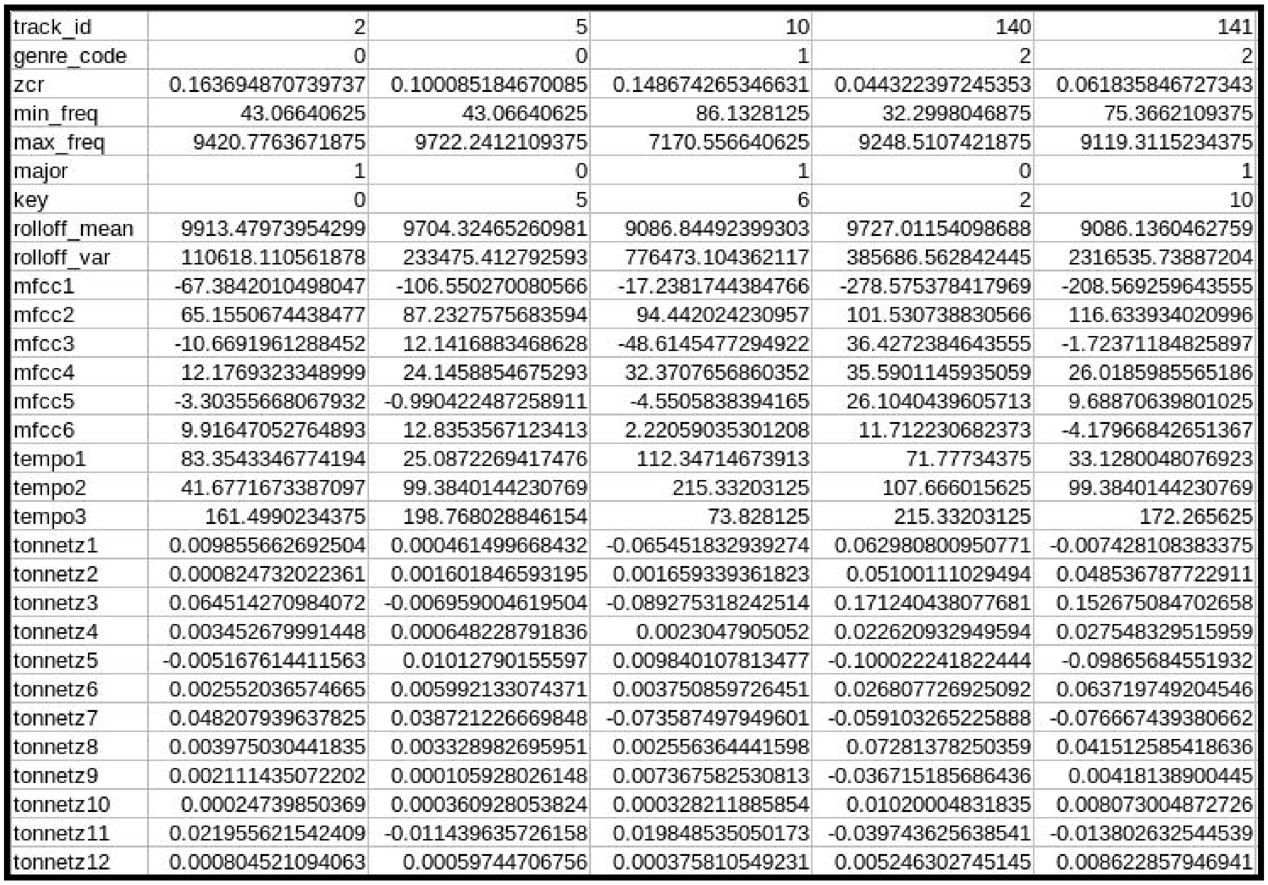
Clark Brown, Sam Cochran, Daniel Swingle

Initial Feature Engineering Report

Features we've added:



track_id	2	5	10	140	141
genre_code	0	0	1	2	2
zcr	0.163694870739737	0.100085184670085	0.148674265346631	0.044322397245353	0.061835846727343
min_freq	43.06640625	43.06640625	86.1328125	32.2998046875	75.3662109375
max_freq	9420.7763671875	9722.2412109375	7170.556640625	9248.5107421875	9119.3115234375
major	1	0	1	0	1
key	0	5	6	2	10
rolloff_mean	9913.47973954299	9704.32465260981	9086.84492399303	9727.01154098688	9086.1360462759
rolloff_var	110618.110561878	233475.412792593	776473.104362117	385686.562842445	2316535.73887204
mfcc1	-67.3842010498047	-106.550270080566	-17.2381744384766	-278.575378417969	-208.569259643555
mfcc2	65.1550674438477	87.2327575683594	94.442024230957	101.530738830566	116.633934020996
mfcc3	-10.6691961288452	12.1416883468628	-48.6145477294922	36.4272384643555	-1.72371184825897
mfcc4	12.1769323348999	24.1458854675293	32.3707656860352	35.5901145935059	26.0185985565186
mfcc5	-3.30355668067932	-0.990422487258911	-4.5505838394165	26.1040439605713	9.68870639801025
mfcc6	9.91647052764893	12.8353567123413	2.22059035301208	11.712230682373	-4.17966842651367
tempo1	83.3543346774194	25.0872269417476	112.34714673913	71.77734375	33.1280048076923
tempo2	41.6771673387097	99.3840144230769	215.33203125	107.666015625	99.3840144230769
tempo3	161.4990234375	198.768028846154	73.828125	215.33203125	172.265625
tonnetz1	0.009855662692504	0.000461499668432	-0.065451832939274	0.062980800950771	-0.007428108383375
tonnetz2	0.000824732022361	0.001601846593195	0.001659339361823	0.05100111029494	0.048536787722911
tonnetz3	0.064514270984072	-0.006959004619504	-0.089275318242514	0.171240438077681	0.152675084702658
tonnetz4	0.003452679991448	0.000648228791836	0.0023047905052	0.022620932949594	0.027548329515959
tonnetz5	-0.005167614411563	0.01012790155597	0.009840107813477	-0.100022241822444	-0.09865684551932
tonnetz6	0.002552036574665	0.005992133074371	0.003750859726451	0.026807726925092	0.063719749204546
tonnetz7	0.048207939637825	0.038721226669848	-0.073587497949601	-0.059103265225888	-0.076667439380662
tonnetz8	0.003975030441835	0.003328982695951	0.002556364441598	0.07281378250359	0.041512585418636
tonnetz9	0.002111435072202	0.000105928026148	0.007367582530813	-0.036715185686436	0.00418138900445
tonnetz10	0.00024739850369	0.000360928053824	0.000328211885854	0.01020004831835	0.008073004872726
tonnetz11	0.021955621542409	-0.011439635726158	0.019848535050173	-0.039743625638541	-0.013802632544539
tonnetz12	0.000804521094063	0.00059744706756	0.000375810549231	0.005246302745145	0.008622857946941

Since our original data was made up only of track IDs corresponding to wav files, and their genre labels, our feature extraction makes up all of our useful data. We created a dataframe that has the following features as its columns. The first five rows of this dataframe (transposed) are shown above. Below, we discuss the meaning of each added feature column.

Track ID: each wav file corresponds to a number, and we have a function that generates the file path to access each track if needed.

Genre Code: We have encoded our eight genres by a 1:1 mapping to integers 0-7.

Zero Crossing Rate: Indicates the average rate at which the sign of the signal changes. Higher zero crossing rates match with higher percussiveness in the song. We added this feature because genres often have a certain feel relative to beat and percussive sound.

Frequency Range: The max and min frequency the audio ignoring the top 20% and bottom 20%. Clipping the top and bottom was important because almost all of our audio files go from 10

Hz to 10000 Hz. But seeing the range in where most of the sound of a song is seems to be connected to genre. Some genres have greater ranges while others are in a small range.

Key and Tonality: We used the Krumhansl-Schmuckler algorithm to estimate the most likely key that the audio sample is in, and whether the key is major or minor. We chose this because even though most genres have songs in different keys, knowing the key will aid in normalizing pitch information for other features.

Mel Frequency Cepstral Coefficients (MFCCs): Represents the short term power spectrum of the sound. Aligns closely with the human auditory system's reception of sound. These 6 (for now) coefficients describe the sound of a song in a human way. MFCCs are being used more and more in Music Information Retrieval specifically with genre tasks because they encapsulate the human experience of sound. We feel this will improve accuracy.

Spectral Rolloff: The frequency below which a certain percent of the total spectral energy (pitches) are contained. When audio signals are noisy, the highest and lowest pitches present do not convey much information. What is more useful is knowing the frequency range that 99% of the signal is contained in, which is what the spectral rolloff represents.

The Three Highest Tempo Autocorrelation Peaks: Indicative of what we would guess the average BPM will be for this audio file (3 columns). This is a way of summing up the entire tempogram array in just a few numbers so that comparing tempo features for each track is tractable.

Average Tonnetz over all Time: The mean and variance of the x and y dimensions of the tonal centers for the major and minor thirds, as well as the fifths (this ends up being 6 means and 6 variances for a total of 12 columns). Here we take the means and variances to reduce the information down from a 6xt matrix (where t is the number of time values, about 1200) to just 12 numbers that sum up that matrix for each track.

We recognize that we have a lot of features, and as we build our model we will continue to evaluate which ones contribute most, and may do some feature reduction down the line. So far, however, we are on track with our original project plan, and feature engineering has not unearthed any massive errors in our methodology. For now then, we will continue with the plan outlined in our original proposal.

Below, we include some example code used to collect one of our features. Specifically, the code below is our implementation of the Krumhansl-Schmuckler algorithm for finding/estimating the key. The code for the rest of our features is found on our github.

Sample Code for Features:

```python
# coefficients from: http://rnhart.net/articles/key-finding/
major_coeffs = la.circulant(
    stats.zscore(
        np.array(
            [6.35, 2.23, 3.48, 2.33, 4.38, 4.09, 2.52, 5.19, 2.39, 3.66, 2.29, 2.88])
    )
).T
minor_coeffs = la.circulant(
    stats.zscore(
        np.array(
            [6.33, 2.68, 3.52, 5.38, 2.60, 3.53, 2.54, 4.75, 3.98, 2.69, 3.34, 3.17])
    )
).T
# map an index 0-11 to a key
keys = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"]

def map_key(is_major: bool, index: int) -> str:
    return keys[index] + " " + ("major" if is_major else "minor")

def find_key(y: np.ndarray, sr: int) -> Tuple[bool, int]:
    """
    Estimate the major or minor key of the input audio sample
    :param y: np.ndarray [shape=(n,)]
    Audio time series
    :param sr: number > 0
    Sampling rate of y
    :return: (bool, int)
    Whether the sample is in a major key (as opposed to a minor key)
    Key of the audio sample
    """
    # compute the chromagram of the audio sample
    chroma_cq = librosa.feature.chroma_cqt(y=y, sr=sr)

    # find the average of each pitch over the entire audio sample
    average_pitch = chroma_cq.mean(axis=1)

    # Krumhansl-Schmuckler algorithm (key estimation)
    x = stats.zscore(average_pitch)
    major_corr, minor_corr = major_coeffs.dot(x), minor_coeffs.dot(x)
    major_key, minor_key = major_corr.argmax(), minor_corr.argmax()

    # determine if the key is major or minor
    is_major = major_corr[major_key] > minor_corr[minor_key]

    return is_major, major_key if is_major else minor_key
```