Similarity Detection in Audio Data

Zach Chase, Adam Fidler, Erik Hannesson

December 12, 2019

1 Motivation and Overview

In an age of trends and algorithms, it has been increasingly difficult for musicians and producers to create unique, chart-topping songs. Many music writers have been shoehorned by popular demand into delivering a small variety of recycled music to sell a hit single. This leads to more songs of increasing similarity, further complicating music production in light of copyright laws. In order to prove that a song is infringing upon a copyrighted work, the plaintiff must prove two things: access (opportunity to view or copy a copyrighted work[1]) and substantial similarity[2] (a level of similarity that shows improper appropriation of the plaintiff's work[3]).

The purpose of this paper is to address the latter of these requirements by mathematically analyzing the similarity between songs. We will build upon previous audio similarity work to develop an legally admissible, objective, similarity metric. Ideally, this metric will be used to clearly and accurately measure the similarity between audio similarity lawsuits. At this point we do not claim it is possible to decide a case purely based on numeric analysis, but rather it can provide additional insight to help the decision.

Many modern musicologists use the "ordinary observer test," to prove substantial similarity between songs, in which an average person (one without musical training or experience) is asked to listen to the disputed pieces. If the individual can identify copied elements between the musical pieces, then the piece is considered infringing.[4] As a result, many careers, reputations, and occasionally millions of dollars depend on a subjective case ruling; whereas an unbiased metric would provide greater clarity and objectivity in ruling against true copyright infringements.

We see a such objective methods of similarity detection in technologies like the mobile app Shazam. This app can identify specific audio from music, movies, advertising, and television shows by analyzing recorded samples. While this and other apps are useful in identifying audio, they fall short of viable application in audio copyright cases.

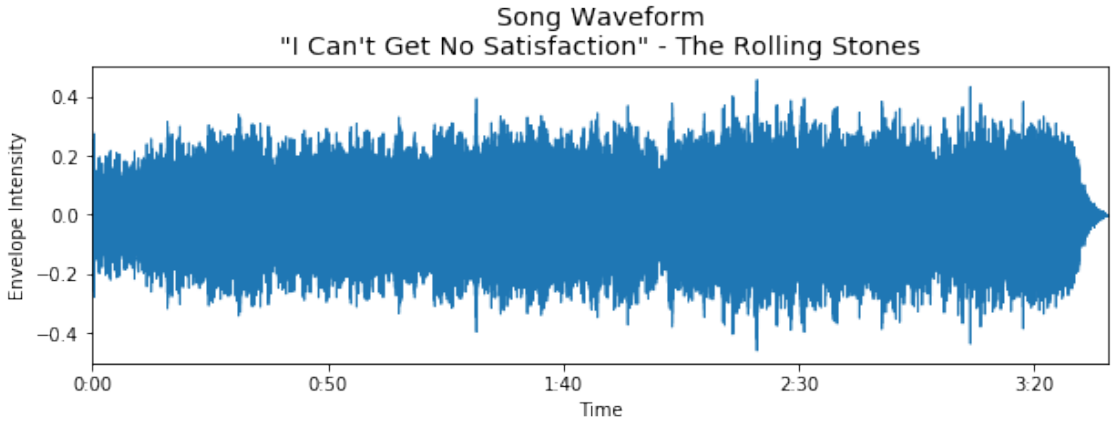


Figure 1: An audio time-series is most fundamental data structure in audio analysis; every figure we display originated with something like this.

To conduct our analysis and develop this metric, we sought datasets that we could both test and train on. We found the Covers80 dataset from the Columbia Academic Quality Fund, which contains numerous cover songs and their corresponding originals,[5] originally used by Ellis and Poliner.[6] This provided our training data, which we then compared to the song data from the Music Copyright Infringement Resource (MCIR), compiled by Charles Cronin, a JD, Ph.D., and musician at University of Columbia. The MCIR contains hundreds of musical copyright cases dating back to the late 19th century and excerpts of the songs in question.[7]

This data should provide a baseline for comparing similarity, allowing us to compare a song, via various spectral features, to its cover and another arbitrary song.

2 Data Collection and Cleaning

2.1 Covers80 Dataset

To proceed with the aims of this project, we must first gather the data. The Covers80 dataset is available for download at http://labrosa.ee.columbia.edu/projects/coversongs/covers80/. The songs came partitioned into subdirectories containing audio files of a song and its cover. As a result, little cleaning was necessary. However, all of the audio files were saved as mp3 file format, and LibROSA (a Python library for audio analysis) requires wav format. We converted the dataset from mp3 to wav as a batch using the following Bash command:

```
find -name "*.mp3" -exec ffmpeg -i {} {}.wav \;
```

This command located and converted all mp3 files using ffmpeg. Once converted, we evaluated and explored the cover dataset. Each song is about 3-5 minutes in length; some are studio recorded, while others are live performances. The live performances, which were almost categorically low quality recordings, contained non-musical background noise. We attempt to address the noise in our eigenfeature method, explained in greater detail in Section 5.

2.2 Copyright Dataset

To collect the copyright case data, we used BeautifulSoup to scrape the MCIR source's website https://blogs.law.gwu.edu/mcir/cases/. The web page is neatly organized with a table of relevant data, including year, case name, complaining and defending works, as well as complaining and defending authors. In addition, each case has a link to audio samples for both the plaintiff and the defendant. Our scraper collected these metadata and audio samples, ignoring cases without audio files. The scraping algorithm was such that it eliminated the need to organize and clean the data. As with the Covers80 data, the audio was mostly provided as mp3 format. Running the same Bash command converted them into wav format as a batch.

Each audio sample in the MCIR is about 40-50 seconds in length and contains only the parts of the plaintiff and defendant works being investigated for infringement. Detailed code for the scrapping of cleaning of this dataset are found in the Appendix.

2.3 Reason for Two Datasets

Again, we reiterate the purpose of our two datasets. As the purpose of this paper is to mathematically analyze the similarity between songs in copyright infringement cases, we require a baseline. We hope to determine this baseline by investigating a song compared with two others: a cover and different, random song from Covers80.

We anticipate that songs and their covers should be very similar, while songs compared with random songs should be noticeably different, which provides the motivation for calling Covers80 the training dataset. After establishing the baseline, we analyze the lawsuit songs; hence, the copyright data is our testing set. Plagiarized songs should exhibit higher similarity than random songs, but potentially lower similarity than covers. Moreover, we hope to find that the two distinct datasets will mitigate the potential bias of a single dataset.

3 Initial Data Exploration

3.1 LibROSA

In order to analyze the data from Covers80 and MCIR we utilize the Python library LibROSA. This library is described as "a Python package for music and audio analysis, [which] provides the building blocks necessary to create music information retrieval systems." [8] It provides tools for loading, visualizing, transforming, and otherwise manipulating audio data.

The data engineering and analysis discussed throughout this paper rely heavily on LibROSA's feature extraction tools. For more detailed information on the our implementation of LibROSA, see the Appendix.

3.2 Exploratory Visualization

For our preliminary analysis, we focused on a few main audio features using LibROSA. Most of these were spectral features obtained via transformations such as the discrete Fourier transform and the discrete Cosine transform.

The following is a brief overview of the chosen spectral features, how they are processed, and how to interpret them.

3.2.1 Chromagram

Chromagrams transform audio time-series into pitch class distributions, typically based on the 12 notes of the chromatic scale. Using a short-time Fourier (STFT) or constant-Q transform (CQT) on many subsequent intervals of a song produces a harmonic fingerprint that can be used to compare with other songs. The three chromagram variants we use are the STFT, energy normalized (uses STFT), and the CQT.

The constant-Q chromagram for the song "I Can't Get No Satisfaction", by The Rolling Stones, is shown in Figure 2; brighter colors correspond to higher harmonic intensities.

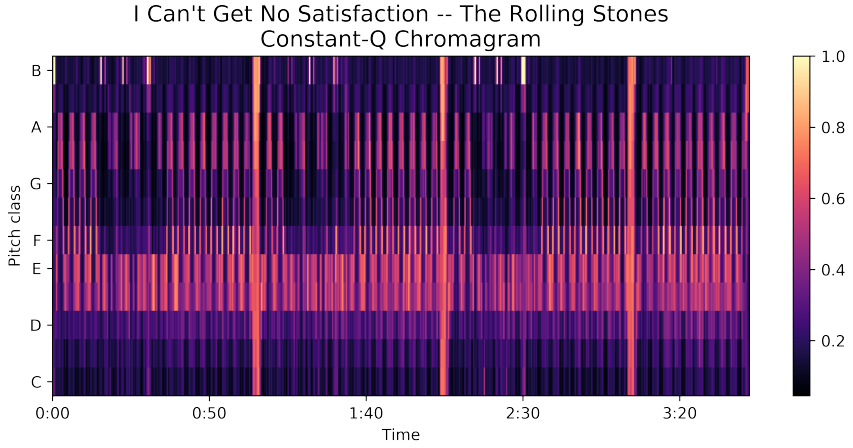


Figure 2: The constant-Q chromagraph of "I Can't Get No Satisfaction" by The Rolling Stones reveals much of the song structure. The repeated darker regions in the top half of the figure coincide with the chorus.

Chromagrams are one of the most useful features in song key identification. We discuss later how we leverage this to help process and transform our data into a more usable form.

3.2.2 Mel-frequency Cepstrum (MFC)

The mel-frequency cepstrum (or spectrogram) has no direct interpretation. The trouble is the data is log-scaled *after* it is transformed into frequency space; so the result is neither temporal nor harmonic. However, the MFC is generally considered to correlate strongly with how humans perceive sound, an important aspect when considering audio similarities.

The mel-frequency cepstrum is calculated as follows:

1. Transform to frequency domain via FFT
2. Apply a mel-filter
3. Log-scale the data
4. Transform using a discrete cosine transform.

Note that while it is traditional to plot the MFC with Hz on the y-axis, there is no *direct* relationship to the specific values displayed.

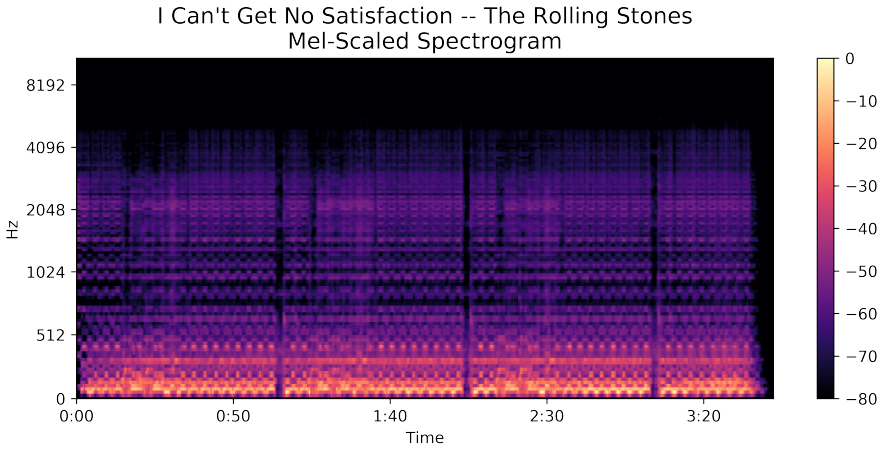


Figure 3: It is from the mel-scaled spectrogram that we derive the mel-frequency cepstral coefficients, or MFCC's, which are mentioned in Section 5

3.2.3 Spectral Bandwidth

The spectral bandwidth is defined as the width of the band of light at one-half the peak maximum. This particular function from LibROSA computes the pth-order spectral bandwidth and is useful in that it provides the difference between upper and lower frequencies.

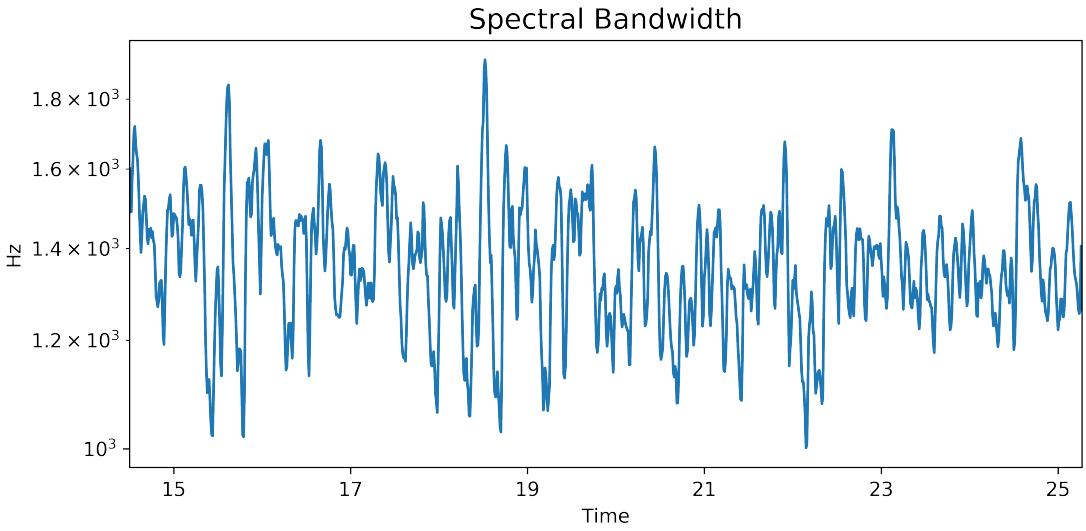


Figure 4: The spectral bandwidth can be loosely thought of as the momentary varience of a song's frequencies.

3.2.4 Spectral Contrast

Each frame of a spectrogram S is divided into sub-bands. For each sub-band, the energy contrast is estimated by comparing the mean energy in the top quantile (peak energy) to that of the bottom quantile (valley energy). High contrast values generally correspond to clear, narrow-band signals, while low contrast values correspond to broad-band noise.

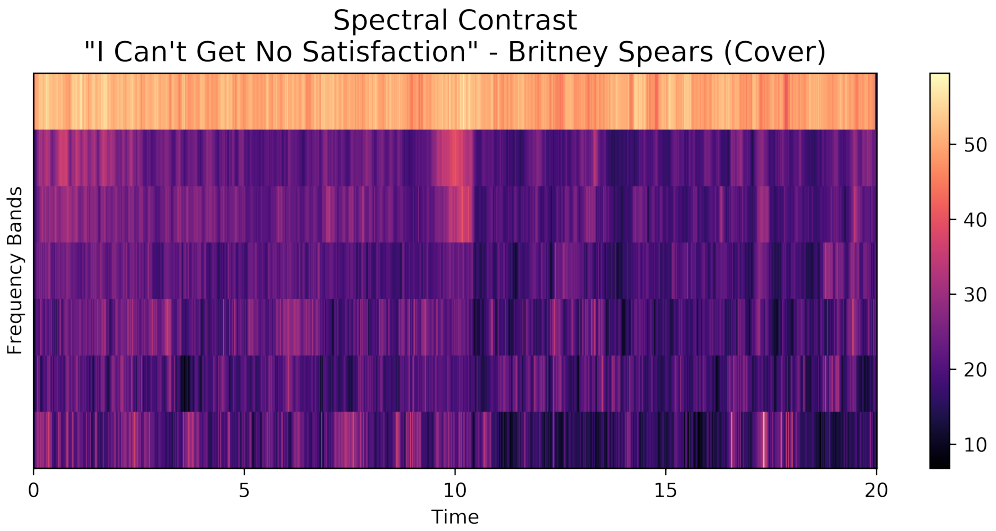


Figure 5: Spectral contrast for the Britney Spears cover of "I Can't Get No Satisfaction".

3.2.5 Zero Crossing Rate

This determines the rate of a signal changing from positive to negative or vice versa. Because of it's usefulness in classifying percussive sounds, this feature has been heavily used in both speech recognition and music information retrieval. Additionally, one of the simplest ways to distinguish between voices and no voices in audio samples is by analyzing the zero crossing rate.

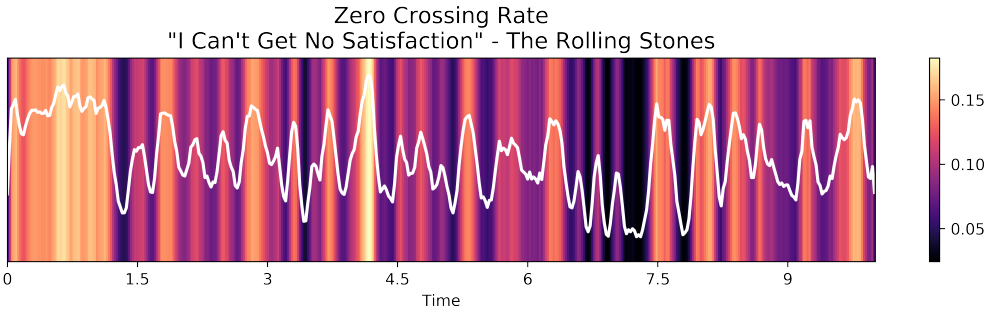


Figure 6: The zero crossing rate is often used to identify percussion, since it captures the sharp change in a signal.

4 Data Engineering

4.1 Aligning Data

Despite the fact that we can easily extract these spectral features from a song, abnormalities inherent in audio data can weaken a one-to-one comparison. The most significant challenge with automated detection of similarities in audio files is data alignment.

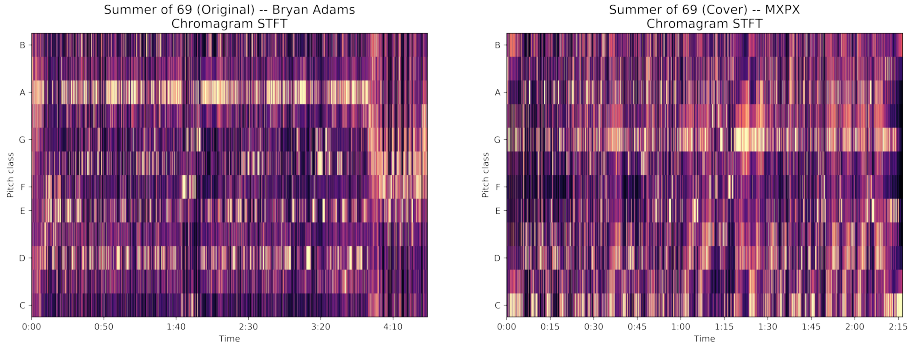
Songs are played in different keys and at different tempos. Furthermore, songs are not of uniform duration. As such, a naive attempt to directly compare two songs without significant data engi-

neering is either impossible or meaningless. We have made significant efforts to combine previous research efforts to create innovative solutions to these challenges.

4.1.1 Key

Differences in key provide an immediate challenge when comparing song audio. For example, it is reasonable to expect that a song and cover of that song should be quite similar; however, if the original was recorded in the key of A but the cover is transposed to the key of D, there is little hope of detecting the similarity based on spectral (harmonic) features. This issue is readily seen in Figure 7, where the chromagram for Bryan Adams' original song, "Summer of 69", has the strongest (i.e. brightest) harmonic features in the pitch A, while the cover by MXPX shows the strongest harmonic features in the pitch G.

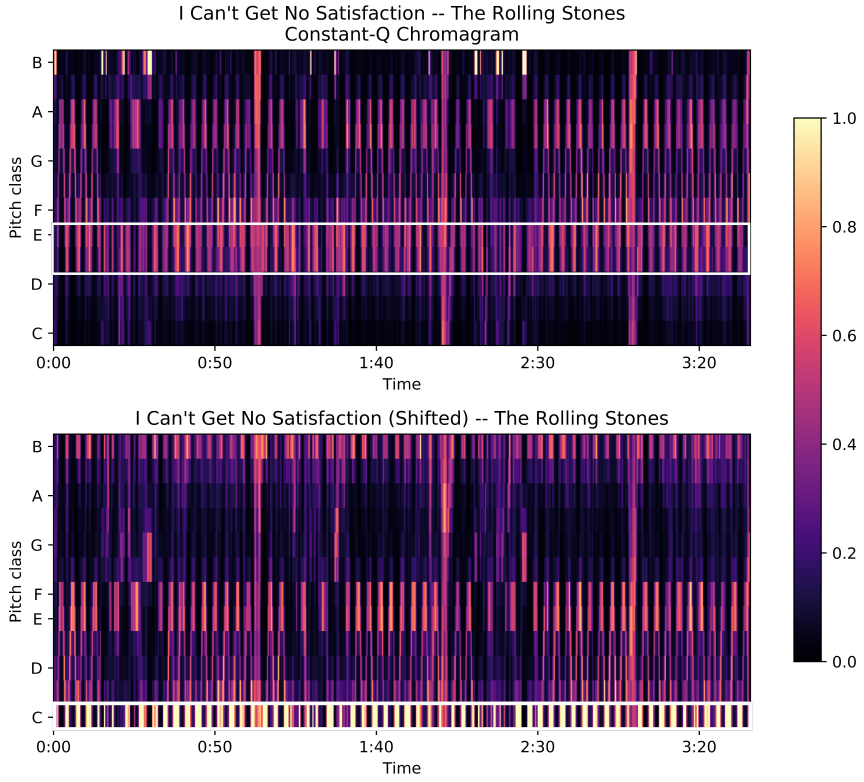
Figure 7: The orignal version of the song, as performed by Bryan Adams, is in the key of A; the cover performed by MXPX, however, is in the key of G.



One straightforward approach to solve this problem is to shift each audio file to a standard key. We decided to transform each song to the key of C (C minor and C major). To minimize the introduction of artificial audio artifacts, the direction of this transformation was chosen to minimize the number of half-step int~ervals required.

The results can be seen in Figure 8, where we the original chromagram of The Rolling Stones' "I Can't Get No Satisfaction" displayed above the key-adjusted chromagram. In the original (top), it is clear that the strongest harmonic feature throughout the song is the pitch D# or E; after shifting, the strongest harmonic feature is the pitch C, as desired.

Figure 8: (Above) Prior to normalizing for key, the song was sung in E; after we applied the transposition, the audio is centered around C.



While this is a conceptually straightforward process, there are some significant technical challenges when implementing an automated transformation pipeline. The main challenge is that we need to know the key of the song in order to shift it properly, but this information is typically unavailable. As such, we have implemented a key detection algorithm, based one the work of Kumhansl and Schmuckler[9]. The algorithm accepts a pitch-class energy distribution (i.e. an array of the row-means of chromogram matrices), and computes a similarity score with each major and minor key profile. The key with the highest similarity score is taken to be the key of the song.

```
def detect_key(X=None):
        """Estimate the key from a pitch class distribution.
        Parameters
        ----------
        X : np.ndarray, shape=(12,)
            Pitch-class energy distribution; need not be normalized.
        Returns
        -------
        major : np.ndarray, shape=(12,)
        minor : np.ndarray, shape=(12,)
            For each key (C:maj, ..., B:maj) and (C:min, ..., B:min),
```

```
        the correlation score for X against that key.
    """
    X = scipy.stats.zscore(X)
    # Coefficients from Kumhansl and Schmuckler
    # as reported here: http://rnhart.net/articles/key-finding/
    major = np.asarray([6.35, 2.23, 3.48, 2.33, 4.38, 4.09,
                        2.52, 5.19, 2.39, 3.66, 2.29, 2.88])
    major = scipy.stats.zscore(major)

    minor = np.asarray([6.33, 2.68, 3.52, 5.38, 2.60, 3.53,
                        2.54, 4.75, 3.98, 2.69, 3.34, 3.17])
    minor = scipy.stats.zscore(minor)

    # Generate all rotations of major/minor (i.e. circle of fifths)
    major = scipy.linalg.circulant(major)
    minor = scipy.linalg.circulant(minor)

    return major.T.dot(X), minor.T.dot(X)
```

We further adapted this algorithm to include information from multiple pitch-class decompositions (e.g. chroma stft, chroma cqt, etc.), and use them in concert to yield higher classification accuracy.

Once the key has been identified, we use the `pitch_shift` function available in the `librosa` library to transform the audio data to be C normalized. The `pitch_shift` function basically uses the Fast Fourier Transform (FFT) to transform the audio to the harmonic domain, shift the entire harmonic spectrum, and then use the Inverse FFT, to convert back to an audio time-series.

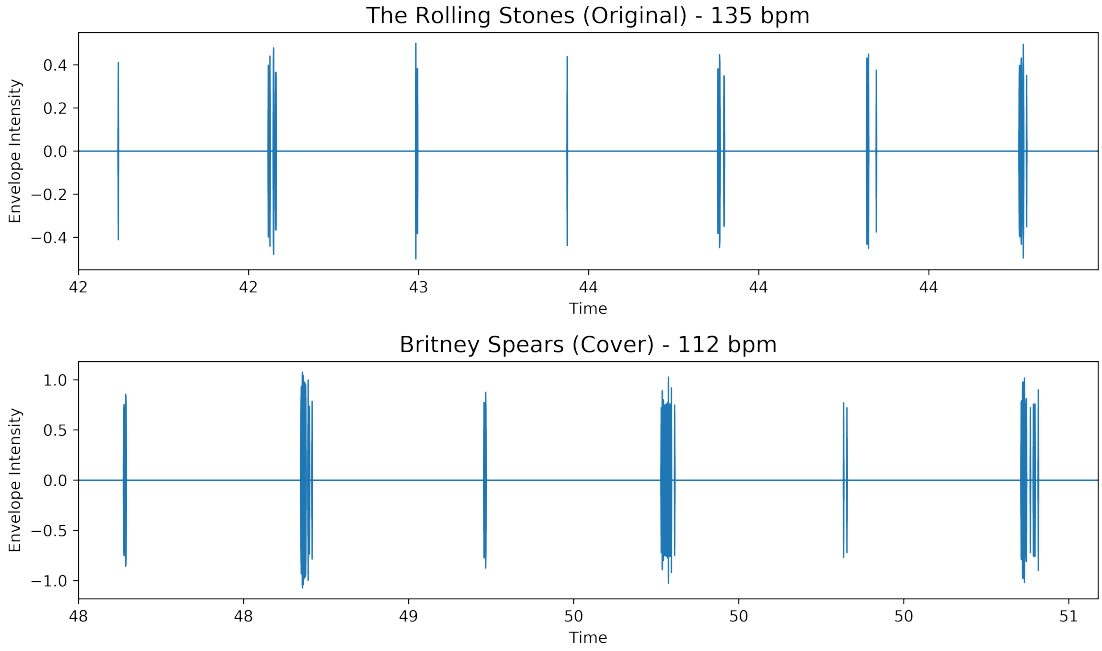
Other Challenges with Key While shifting the audio data to a normalized key solves many comparison problems, it also introduces some potential issues of its own. In particular, we have not transformed all of the audio to the *same* C key; that is, the transformed songs may be separated by an octave. While this does not impose significant issues when comparing certain spectral features, it does introduce some loss in accuracy when comparing two songs in different octaves.

This can be avoided by adding an additional step to the original key transformation process that would make sure to send each song to middle C, for example.

4.1.2 Tempo

Once song data has been key normalized, the next challenge that naturally arises is that of differing tempos. Two songs may be very alike, but if one song is played at 135 beats per minute (bpm), and the other at 112 bpm, the similarities will quickly fall out of sync. For example, in Figure 9, we plot the percussive beats during the chorus of "I Can't Get No Satisfaction", as performed originally by The Rolling Stones, and in a cover by Britney Spears.

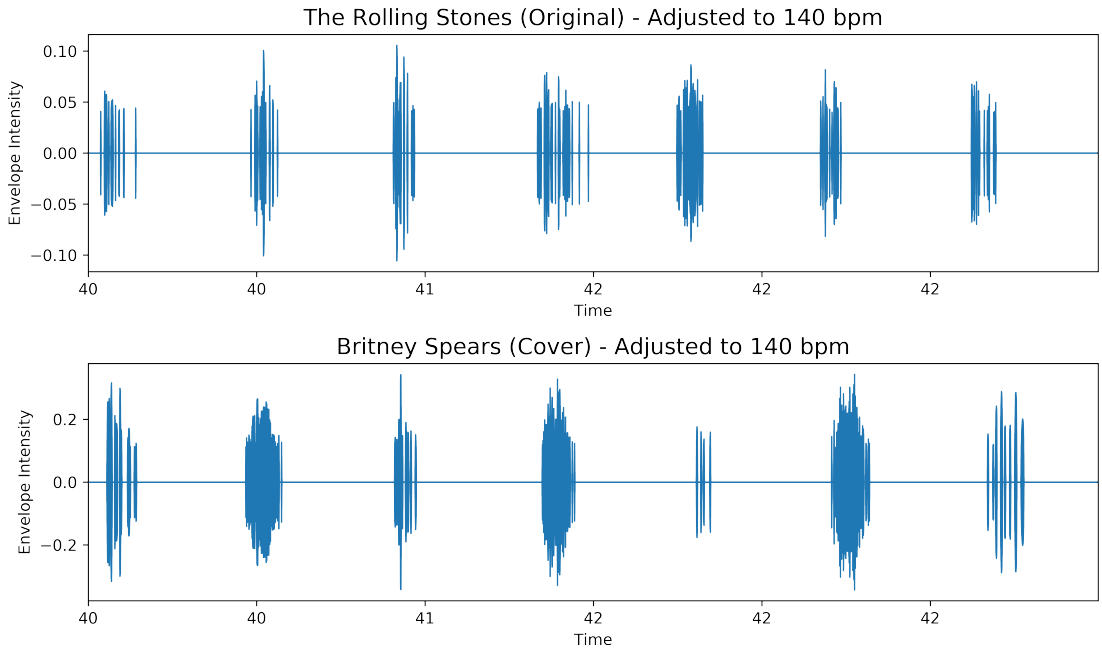
Figure 9: The beats in the cover by Britney Spears are spaced out slightly more than the original song by The Rolling Stones. Halfway through the chart you can see the two songs are totally out of sync.



Even though the plot spans only three seconds of audio, it is easy to see that the beats in the cover fall behind those of the original (as is expected with a lower tempo).

Luckily, this problem can be solved in much the same way that we dealt with differences in key: detect the tempo of each song, then transform the audio data to a standardized tempo. In our experience, sending every song to 140 bpm has proven successful. Figure 10 displays the same clips of "I Can't Get No Satisfaction" *after* we performed a tempo standardizing transformation. Each beat in the window lines up almost exactly, as desired.

Figure 10: After applying our tempo normalization method, the two songs play at a steady 140 bpm; we no longer have issues of off-beat confusion.



Tempo detection is typically done by filtering audio to include only percussive events, which tend to be uniformly spaced for much of the song. Finding the number of such events in a small window of time gives an estimate of the number of beats per minute. We combined multiple tools from `librosa`, to perform this task. It should be noted that, as with all transformations, some additional noise can be introduced.

5 Eigenfeature Method

In our analysis we have implemented ways to normalize key and tempo. Reasonable workarounds are in place to account for varying song length and additional noise from live-recorded songs.[8]

As indicated previously, the main spectral features we consider are CENS, spectral contrast, MFCC, spectral bandwidth, and zero crossing rate. In an effort to delve deeper into these spectral features, we implemented an eigenfeature method.

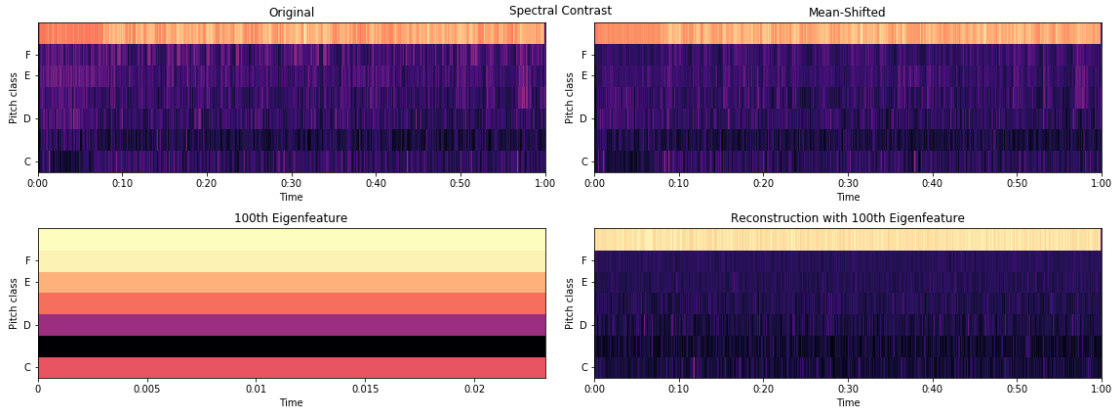
The eigenfeature method consists of the following elements:

1. Perform feature extraction or transformation on the audio data.
2. Calculate the mean feature given all the songs in the cover dataset.
3. Apply a mean-shift to each song.
4. Calculate corresponding eigenfeatures via singular value decomposition with variable rank for computational efficiency.

5. Match songs by calculating $\underset{i}{\text{argmin}}||\hat{\mathbf{f}}_i - \hat{\mathbf{g}}||$, where $\hat{\mathbf{f}}_i$ is the song (expressed as a column of the matrix of all songs) that is closest to a given song $\hat{\mathbf{g}}$.

Some of this process can be seen visually in Figure 11.

Figure 11: Spectral contrast of a song throughout various stages of the eigenfeature method.



We noted that the mean feature also included a significant amount of the noise from live recordings. Thus applying a mean-shift to each song reduces the effect of noise.

Moreover, we choose to take 60 second intervals of the songs, rather than load the entire song each time. Not only does this significantly reduce the computational complexity of the eigenfeature method, but it also provides enough audio for reasonable analysis while keeping the vector sizes consistent.

This approach proved only marginally effective at identifying similarities between two songs; similarities between the original song and its corresponding cover were often overlooked by the algorithm. We were, however, able to identify issues in the dataset itself, such as duplicate songs that were incorrectly listed as a cover/original pair. Moreover, the algorithm appeared to effectively match songs that were at least within the same genre.

6 Conclusion

Implementing a mathematical metric for copyright infringement cases is an ambitious goal, which our attempts demonstrate. We gathered audio data from the Covers80 and MCIR datasets, then determined their validity and usability. Using LibROSA, we explored the datasets, calculated spectral features, and visualized the data. Because inconsistencies between audio samples weaken a similarity analysis, we devised methods to align the data by key, tempo, and duration. Additionally, we instituted an eigenfeature method that estimates the most similar song by finding the mean-shift and SVD of specific song features. Combining these alignment and eigenfeature methods show promise in developing an objective and accurate metric for audio similarity with legal applications.

6.1 Bibliography

[1] Merriam-Websters Dictionary of Law. Springfield (US): Merriam Webster, 1996. Music Copyright Infringement Resource. Accessed December 13, 2019. https://blogs.law.gwu.edu/mcir/.

[2] "Substantial Similarity: Glossary." Research Guides. Accessed December 13, 2019. https://guides.lib.umich.edu/substantial-similarity/glossary#s-lg-box-wrapper-21057273.

[3] "Anatomy of a Copyright Infringement Case: Elements of a Copyright Infringement Claim." Fasthoff Law Firm - The Woodlands Lawyer, The Woodlands Business Attorney, Domain Name Lawyer, Trademark Lawyer, December 8, 2016. http://www.fasthofflawfirm.com/anatomy-copyright-infringement-case-elements-copyright-infringement-claim/.

[4] Copyright Law - Infringement - Ordinary Observer Test. Accessed December 13, 2019. http://www.tabberone.com/Trademarks/CopyrightLaw/Infringement/OrdinaryObserverTest.shtml.

[5] Ellis, Dan. The covers80 cover song data set. Accessed December 13, 2019. http://labrosa.ee.columbia.edu/projects/coversongs/covers80/.

[6] D. Ellis & G. Poliner (2007). Identifying 'Cover Songs' with Chroma Features and Dynamic Programming Beat Tracking *Proc. Int. Conf. on Acous., Speech, & Sig. Proc. ICASSP-07, Hawai'i, April 2007, pp.IV–1429-1432.*

[7] Music Copyright Infringement Resource. Accessed December 13, 2019. https://blogs.law.gwu.edu/mcir/.

[8] "LibROSA." LibROSA. Accessed December 13, 2019. https://librosa.github.io/librosa/index.html.

[9] Accessed December 13, 2019. http://rnhart.net/articles/key-finding/

6.2 Appendix: Code

See auxillary file for any additional code used.