Music Pace Compatibility Project
Consolidated Report

Andrew Harless
Springboard

2019-04-19
revised 2019-08-09

I. The Problem

A. Overview

A project to estimate multiple musical tempos from an audio segment. For example, a song marked at ♩=160, in 4/4 time with secondary stress on beat 3 and a lot of ♫, could be timed as 40, 80, 160, or 320 beats per minute depending how you define a beat, and depending on the particular sound of the song, which may or may not lend itself to any or all of these timings.

Beat detection and tempo estimation are standard problems, for which (imperfect) solutions exist. But one thing I've found trying to jog to music is that a single tempo estimate (even if it's accurate) doesn't necessarily solve the relevant problem. For example, if a song has ♩=105 but has a lot of 8th notes, you can jog with very quick steps (210 per minute) or with very slow steps (105). And a song compatible with a normal jogging pace (say 160) may not be recognizable as such from a single tempo estimate (e.g. maybe 80).

The problem is complicated by the fact that, while most songs have an even time signature (2 or 4), many have odd (usually 3), and some have both even and odd factors (6 or 12). So the relation of the official tempo to the possible alternatives differs from song to song.

B. Applications

Tempo estimation has numerous applications, but what particularly interests me is keeping pace for jogging, walking, marching, running, and so on. The present project adds a new dimension by allowing for multiple tempos.

A hypothetical client could be someone developing smartphone software to choose music for jogging/walking/running. Once a model is developed, it can be used to produce an ever-expanding database of songs with compatible tempos. The database could then be used to choose music compatible with an intended pace.

II. The Data

The raw data consisted of a set of audio files, each containing a single song, and a human-labeled database specifying the tempos that are compatible with each song. I provided the initial labels myself. (Presumably if the project continues, it could include other labelers.)

The data issues for this project concerned two things:

1. Getting audio files into a form that could be used to address the question of concern
2. Getting label data into a form convenient to train a model

The original "raw" input files came in MP3 format (from my own music collection). Much of the processing was done using the LibROSA Python library, which initially converts an MP3 file directly to a simple time series represented by 1-dimensional NumPy floating-point array. (By default—which I accepted—the stereo channels, if they exist, are mixed to mono and sampled at a rate of 22050 Hz, which is less precise than the typical 44100 Hz used for WAV files.)

An immediate issue was that many songs are "dirty" from a tempo point of view, in the sense that the tempo is not consistent throughout the song. Particularly, at the beginning or end of many songs, there is an acceleration or deceleration, or a special section at a different tempo than most of the song. I dealt with this issue by simply removing the first 512K elements and the last 512K elements of each array (corresponding to about 24 seconds of music each: 512K/22050). This isn't a perfect solution, since it loses a lot of data, and in some cases the remaining data may still be "dirty," but I wanted to follow a simple and consistent procedure. (Songs that contained noteworthy shifts in tempo in the middle of the song, I simply discarded from the training data.)

The next issue was how to represent sound data for analysis. To keep the project simple, I chose a single representation, the Mel spectrogram, which is a common way of representing audio data to emphasize human-relevant features. By default, LibROSA calculates a spectrogram of 128 frequencies at hops of 512. For greater precision I chose a hop length of 256 instead, meaning that there are about 86 (22050/256) spectral samples per second, or about 5200 per minute. This rate allows for 16 spectral samples per beat at my (arbitrarily chosen) maximum tempo of 320 beats per minute, thus permitting me to extract a reasonable amount of information to be associated with each beat for any tempo in the range I was considering.

Before proceeding with further wrangling of the audio data, I had to make a decision about how to deal with the label data. In the form that I initially recorded my labels, a song can have up to 3 “compatible paces,” but some songs have only 1 or 2. Initially I had thought I would use a whole song as the unit of analysis for machine learning and come up with some loss function to describe how well the model’s outputs approximated the set of labels for a given song. But there were two problems with that approach. First, the required loss function would be quite complicated and arbitrary. Second, using a whole song as the unit of analysis would severely limit the amount of training data available. So I settled on a different approach.

Instead of using a song as the unit of analysis, I use a random clip from a song. And instead of creating a model that would estimate a set of tempos (paces) directly, I generate a set of “candidate tempos” and classify each as “wrong” or “right” depending on whether it is close to one of the human (“ground truth”) labels or not.

I generated candidate tempos as the peaks of the timewise mean of LibROSA’s tempogram. For each song, I divided it into 24-second sections and generated a set of candidate tempos for each section; then I took the union of those sets and associated that whole set with the song, as a set of candidate tempos. To generate a training (or validation) case, I choose a random song, choose a random clip from the song, and choose a random candidate tempo from the song’s associated set. Then I classify the case as either positive or negative according whether the particular candidate tempo is within 5% of one of the “ground truth” labels.

The next issue was how long to make each random clip. One possibility would have been to make all clips the same length in time, but that would mean that some clips would contain many beats at the candidate tempo and some would contain only a few. Instead I decided to make all clips the same length in terms of hypothetical beats. By “hypothetical beats” I mean beats that would occur at the candidate tempo. I chose each clip to be a length of 16 hypothetical beats. So, for example, if a song has both 80 and 160 beats per minute as candidate tempos, a clip associated with the 80 candidate tempo would last one fifth (16/80) of a minute (i.e., 12 seconds), and a clip associated with the 160 candidate tempo would last one tenth (16/160) of a minute (i.e., 6 seconds).

Since the clips (as decided above) had different lengths in time, but the spectral data were all sampled at the same time rate, I had to resample the data to put them in a consistent form. (An alternative would have been to use features that could be generated in a consistent way from different time lengths, but I chose not to go that route.) Resampling turned out to be the hardest part to code. I used Pandas time series methods to upsample and then downsample the data. The result always has 16 samples per (hypothetical) beat, but the initial upsample goes to the least common multiple of 16 and the original number of samples per beat, so as to avoid unnecessarily losing information in the resampling process. (It turns out that Pandas time series methods are not a very efficient way of doing this, but at least they work.)

After resampling, the “X” data for each case consist of a 256x128 floating point array where one dimension represents time and the other represents audio frequency (pitch). The next wrangling step was to reshape the array to 16x16x128. One dimension then represents the immediate progression of time over the time slice associated with one hypothetical beat, and the other dimension represents the time between successive hypothetical beats. For positive cases, we should expect to see a repetitive pattern across this second dimension, as each actual beat will be similar to the previous beat. For negative cases, patterns along this dimension should seem random, as they do not represent actual beats.

Finally there was the question of how to generate mini-batches of training (and validation) cases. My original idea was to generate batches on demand by sampling random clips during the training process. But the resampling code ran very slowly, which made the training process very slow. This was inconvenient, since I wanted to watch the training process in real time. So instead of generating training data on demand, I generated a huge file of training cases (which I call a “megabatch” since it reused the code originally to designed to create small batches on demand) overnight and sampled cases randomly from that file during training. (There was also a separate file for validation, generated from a separate set of songs.)

Specifically, I generated 16,384 training cases (and 4,096 validation cases). Since the training cases were generated from a single set of about 100 songs, there was not as much variety as the number 16,384 might seem to suggest. However, since the clips were sampled randomly, the phase relative to the beat was different in different clips from the same song. So there was still a lot for a machine learning model to learn.

Currently, code for the data wrangling steps is contained in the following files in the repo:

`SongsSliceDiceSetsWork2.ipynb`:
code to process each song into spectrogram and candidate tempos

`get_training_data.py`:
code to sample clips from songs and associate each clip with a candidate tempo

`SaveMegabatch.ipynb`:
code to standardize clips for use as training/validation data and produce large files thereof

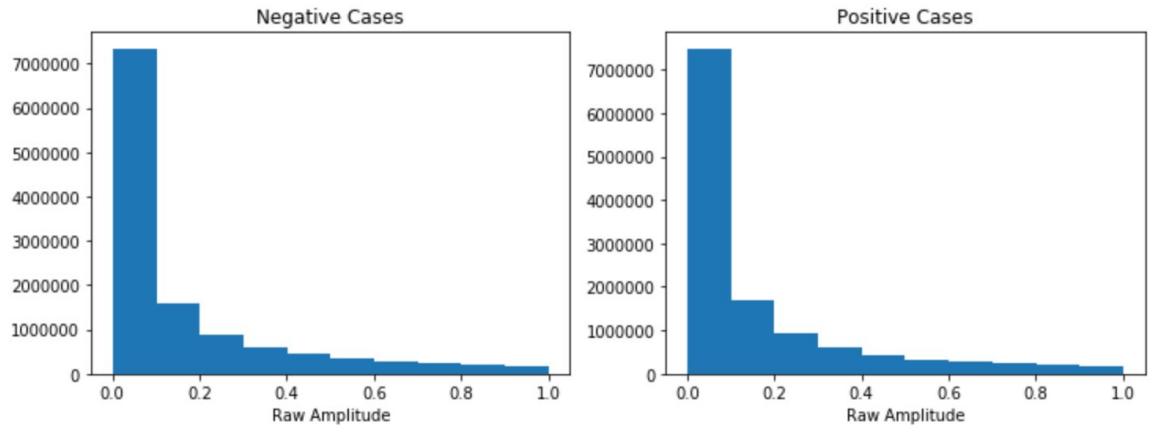
`pace_params.py`:
some of the parameters used in the above files

As of now (2019-03-29), the internal documentation of these files still needs considerable improvement, and much of the code itself could do well with some cleaning up.

III. Initial Findings

A. Raw Data Distribution

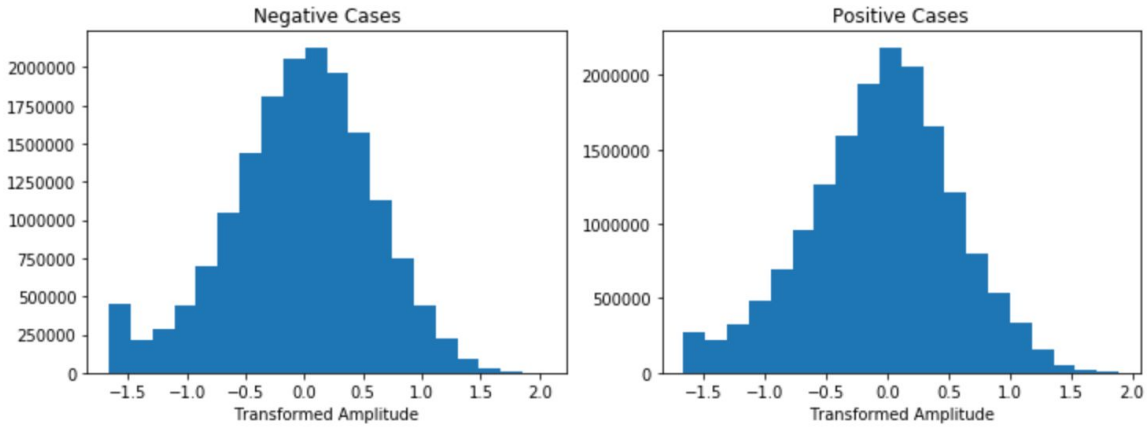
The raw data (taken as a “bag of numbers,” rather than as the tensors that represent whole classification cases) are extremely skewed:



To make them more amenable to analysis, I transformed them by adding a small constant, taking a logarithm, and then centering (approximately) and scaling them:

```
transformed_data = (log(data+1e-5) +2.5) / 5
```

The resulting histograms:

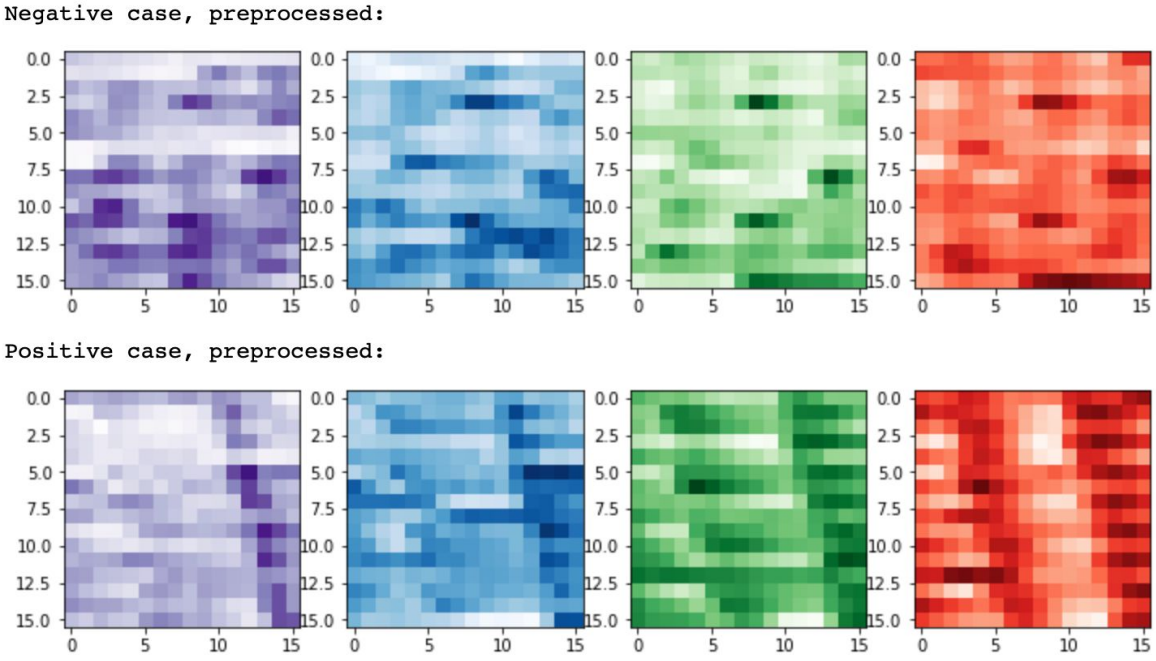


B. Appearance of Positive and Negative Cases

Given the way the data were constructed, positive cases—the ones where an acceptable tempo has been correctly identified—are those in which the relevant (subjective) beat occurs at the same point in each row of the 16x16x128 tensor representing that case. (In other words, the beats should correspond to some particular value of the second subscript.) Whatever acoustic feature is responsible for that subjective beat, that feature should be repeated in each row and form a vertical column. Thus, in principle, positive cases should be identifiable as those which contain some sort of consistent vertical column, although the nature of that column remains vague.

Since there are 128 readings for each cell in the 16x16 matrix, it's hard to say exactly how the relevant "consistent columns" in that matrix might present themselves. To simplify for visualization purposes, I divided the 128 readings into 4 groups and took a mean (at each point in the matrix) for each of these 4 pitch groups. Thus a set of 4 heatmaps represents each training case. I used a different color for each map and represented amplitude (or its transform) by the intensity of the color, with pure white representing the lowest value.

Here are a typical negative and a typical positive example (using the transformed data):



In general, the negative cases look pretty random, as they should. There does seem to be a pattern in most of the positive cases, but it's not exactly the expected one. Rather than the

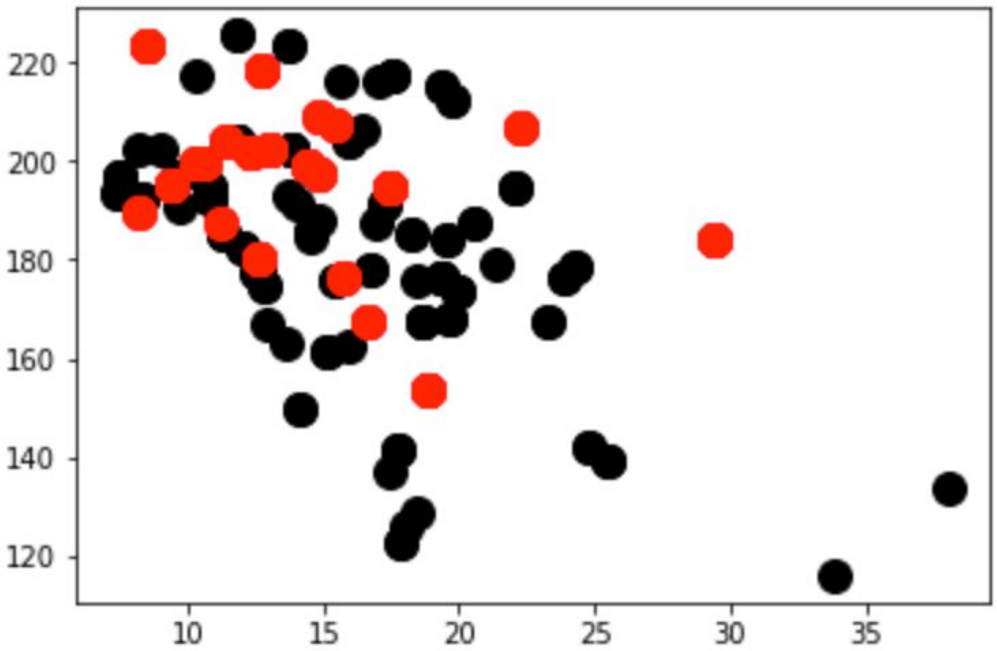
expected vertical lines, there seem to be diagonal ones—steeply sloped (and always negatively sloped) but not quite vertical. Whatever acoustic feature reflects the subjective beat, that feature seems typically to occur later in time at each successive beat. In other words, the period of the acoustic phenomenon seems a little bit longer than that of the corresponding subjective phenomenon. (Or it's also possible that there's something a little bit off about how I constructed the data.)

C. Basic Features that Help to Classify Cases

Another way to look at the data is to look at the correlations of the spectrum across corresponding elements in successive rows. Whereas the squares shown above divide the spectrum into four ranges to facilitate presentation, the correlation takes the whole spectrum into account. We should expect this correlation to be high for positive cases, as it should reflect repetition of pattern for each beat.

Also, for each point in the spectrum, we can calculate the variance within columns, which should be relatively low in positive cases. To display this, we can then take a mean across the spectrum.

The following scatterplot shows sums of variances on the X-axis and sums of correlations on the Y-axis, for a sample of training cases. The red dots represent positive cases, and the black dots represent negative ones:



It does seem that there's some information here: the positive cases are much more likely to occur in the upper left quadrant, as expected. However, there are also a lot of negative cases in that quadrant.

Anyhow, the advantage of using a deep learning model with these data is that we don't need to figure out exactly what features contain the information we need. There's clearly something in the data that we can see. The model will see more.

D. Statistical Test

I performed a statistical test of whether there is identifiable objective information in the audio files that corresponds to my subjective tempo assignment from listening to songs and tapping out the beat. To do this, I looked at songs where `librosa.beat.tempo()` (using its default parameters) identified a tempo that could not be reconciled with any of the ones I identified subjectively. My null hypothesis was designed to correspond to the idea that LibROSA's identified tempo was as good an objective candidate as the first one I identified subjectively. My alternative hypothesis corresponded to the idea that objective information favored my subjective tempo.

Specifically, for each song, my test was as follows:

1. Cut off the beginning and end of the song.
2. Transform the mel spectrogram data to make the distribution look more Gaussian (using the transformation discussed in the “Raw Data Distribution” section above).
3. Take the first K principal components.
4. For each component:
    - Sample each Nth observation, where N corresponds to the period associated with the hypothesized tempo.
    - Take the standard deviation. (The smaller the standard deviation, the more evidence there is of a repeated pattern at the associated periodicity.)
    - Repeat the two substeps above N-1 more times, with each repetition offset by one time unit (spectrogram hop) from the previous one.
    - Take the mean of these N standard deviations.
5. Do step 4 once for the "alternative hypothesis tempo" and once for the "null hypothesis tempo."
6. For each of the K components, calculate the difference between the two values calculated in step 5.
7. For each of the K components, note whether the difference is negative. (Negative differences favor the alternative hypothesis, because they imply more evidence of periodicity at the alternative hypothesis tempo.)

8. Since principal components are, by construction, independent (at least linearly), treat the result (negative vs. non-negative) for each component as an independent Bernoulli trial. Under the null hypothesis, they are like coin tosses: for each one, the probability of a negative value is 0.5.
9. Perform a binomial test of the hypothesis that p=0.5, against the alternative that p>0.5, where p is the probability that any given component will show a negative difference. The statistic is the probability of getting X or more "heads" in K tosses of a fair coin, or `1-binom_cdf(X-1, K, .5)`.

The choice of K is arbitrary: higher values of K introduce more noise by including components that may contain little residual information, but lower values give fewer "trials" to test, so it's not clear what value would maximize the power of the test. As a heuristic, I took every component that individually explains more than 1% of the variance.

I tested 3 songs, and my results were as follows:

For *Collide* I conclusively rejected the null hypothesis, with a p-value of 0.0000019

For *Sally Go Round the Roses* I found no evidence for the alternative, with a p-value of 0.87

For *Without You* I could not reject the null (p=0.11), but the differences for the first four components were negative, so the data seem consistent with the alternative hypothesis, and a differently constructed test might well have rejected the null.

Collectively, applying a correction for multiple hypothesis testing, the results imply a rejection of the null hypothesis with a p-value of 0.0000057

III. Machine Learning

A. Structure of the Neural Network

Output: Sigmoid

Dense(20)

Dropout(.8)

1x4 Convolution, 3 filters

Batch Norm

4x6 Convolution, 25 filters

Dropout(.3)

1x4 Convolution, 12 filters

Batch Norm

1x1 Convolution, 32 filters

Dropout(.1)

1x1 Convolution, 64 filters

Input: 16x16, 128 channels

Note salient features:

- Narrow convolutions (1x4 and 4x6) instead of square convolutions
- Alternating batch norm and dropout between successive convolutions
- Very high dropout before top layer

B. Results

Validation accuracy 0.91 (after 60 epochs using final version of model)

This is quite good, considering that many cases are ambiguous

Example Results:

53rdAnd3rdMP 45
Predicted: 0.00090704486 Actual: False

AHardRainsAGonnaFall 56
Predicted: 0.021726133 Actual: False

AHardRainsAGonnaFall 62
Predicted: 0.017301498 Actual: True

InTheHillsOfShiloh 64
Predicted: 0.13423629 Actual: False

CaliforniaDreamin2MP 112
Predicted: 0.93683857 Actual: True

IThinkWereAloneNow2MP 44
Predicted: 4.0176445e-05 Actual: False

FugueForTinhorns2 161
Predicted: 0.6075436 Actual: True

BloodyMerryMorning 83
Predicted: 0.0055997227 Actual: False

ChatanoogaChooChoo 58
Predicted: 0.10576297 Actual: False

HoldMeTight 45
Predicted: 0.0042094495 Actual: False

See https://github.com/andyharless/paces/blob/master/code/BestStage0Model.ipynb

IV. Recommendations

- The model can begin to be used to create a database, but since testing has thus far been limited, it's important to apply strict quality control initially. At first, each item in the database should probably be reviewed by a human judge before inclusion. As we become more confident, the rate of review can gradually be reduced, until eventually the hope is to have the model functioning essentially autonomously with only a few reviews of random items.

- Since there has thus far only been a single human labeler, tests should be performed with other labellers, and if necessary, the model should be revised to produce results more broadly consistent with human labels.

V. Conclusions and Next Steps

Thus far, the project can be considered a success. As noted earlier, the accuracy of 0.91 is a favorable result: given that many cases are ambiguous, near-perfect accuracy would not be a reasonable outcome. Subjectively, 0.91 seems pretty close to the ideal level of accuracy to be expected from a model that is performing optimally.

However, some additional evaluation is required before the project can be unambiguously declared successful. Thus far, conclusions have been based on performance on the validation data. The validation data are "clean" in the sense that none of the songs they include were used to produce training data, but they still face a risk of overfitting in that the process of developing a model and choosing hyperparameters may have resulted in decisions that are overly specific. To be confident about generalizability, it's important to test with songs that are new to the entire process.

Also at this point we can't be certain that the model is "right at the right times." In other words, although the overall level of accuracy is close to ideal, we don't know yet if the apparent failures are genuinely ambiguous cases or if some significant fraction of the failures are real failures. To answer this question properly will require making additional subjective judgments based on listening to songs in the light of the current predictions of the model.

There is considerable work to be done going forward. Numerous possible next steps are noted in the "Issues and Ideas" section of the repository's README file. Perhaps the most important possibilities are those discussed in the above paragraphs and in the Recommendations section above. Other things that come to mind as important:

- Get more data from underrepresented genres and underrepresented time signatures, and a greater variety of artists.

- Set up the model to run with publicly available audio inputs (which need to be labeled first), so the project could be forked and run immediately by someone else.