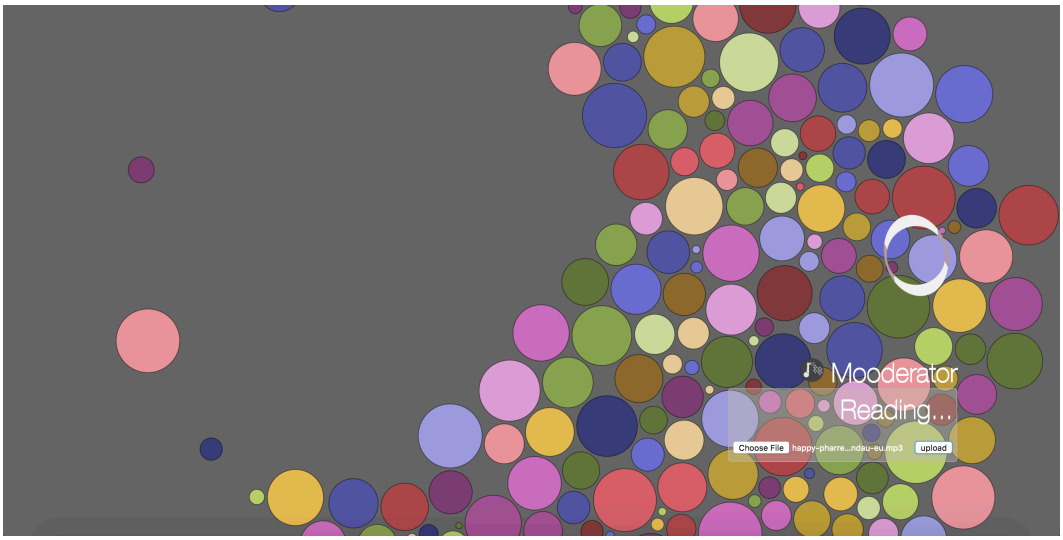
Deconstructing "Mooderator"

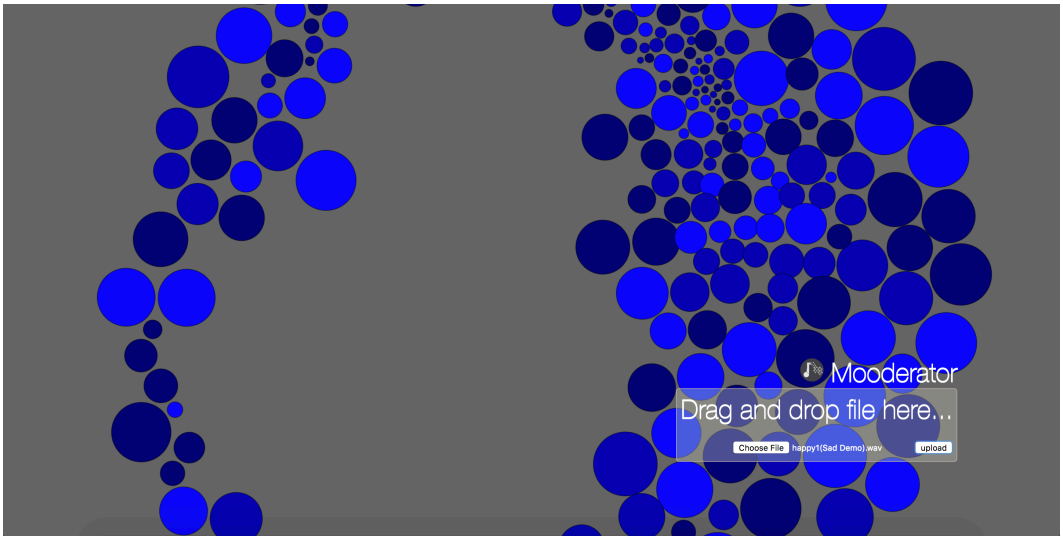


A plaintext explanation and analysis of "`moderator`", a web app for music visualization and categorization by mood.

Marshall Briggs, UID: 304417630
(with Kate Cook and Noam Grebler)
University of California, Los Angeles
PSYCH 186B, Winter 2018
Friday, March 23, 2018

1: Introduction and Problem Definition

Mooderator is an attempt at using data visualization to graphically express qualitative information about music in an interactive manner.



When we play music, there are elements of that specific listening experience that make it different from any other; qualitatively, you and I file our music into different categories like *genre*, *danceability*, or *mood*. Like so many of our attempts to glean insight from our cognitive processes, it is difficult for us to pin down what exactly separates these disparate groups.

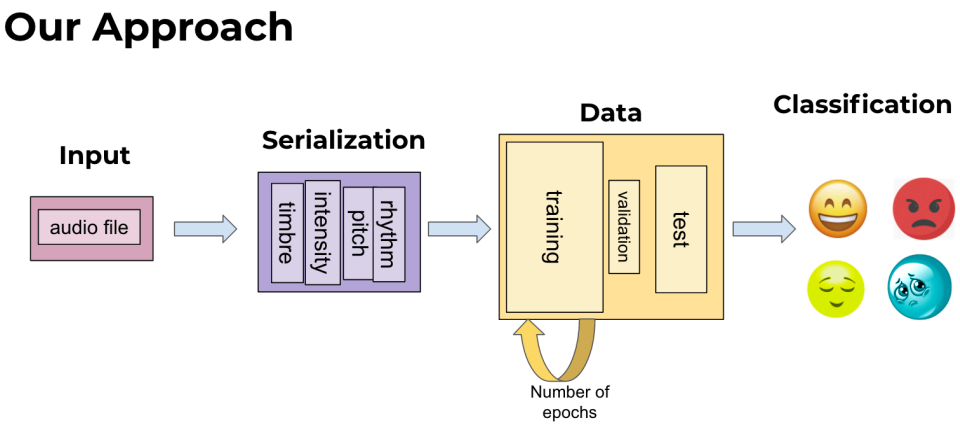
We may be able to identify like characteristics within groups, such as "*Country music often uses a banjo,"* or *"Danceable music is always uptempo,"* but those, rather than being real unifying factors, are likely indicators of more specific quantitative properties (*banjos are high pitched, uptempo music has high rhythm*). But what are those properties with which we group music, exactly, and where do we draw the lines between groups?

As is often done in the pursuit of insight into human cognitive processes, recent work has been done through the use of neural networks to analyze and categorize music within genres (Kozakowski & Michalak, 2016), moods (Liddy & Schindler, 2016), levels of danceability (Cloudhary & Agarwal). Though they differ in their research question, the referenced researchers are united in their common approach to a similar problem; they attempt to model a music analysis engine by breaking down input sounds into their frequency time series and analyzing quantitative properties of said series.

The objective of this project was to do the same.

`Problem Definition:` Create a data visualization application backed with a music recognition engine which, when trained through the use of a sophisticated neural network, can correctly interpret the mood of an input audio file and output a meaningful graphic representation of said audio.

In the following pages, I will outline our Python implementation of the stated problem definition, all the while avoiding writing *actual code* like the plague. The objective of this report is that you, the reader, regardless of your experience with Python, the Flask, Tensorflow and Keras modules, upon completion of this document will conceptually understand how the application accepts, processes, and predicts input sounds.



2: Dissecting Mooderator

I. The front end: `colors.html`

I will begin by briefly explaining the function of the application's front end, as this is the least conceptually difficult piece. As a preface, though a visual, graphical front end did nothing to aid the efficiency or the intelligence of our neural network model, it was important to us to *use* the model we created for a useful purpose. It is our hope that, at worst, the front end web app is an entertaining way to listen to music, and at best helpful in understanding how the network model categorizes music.

In `colors.html`:

1. We initialize an svg canvas on which to display an animation.
2. We indicate a region in the bottom right of the window, for the title and in which to drop input music files.
3. We then populate the empty canvas with 350 circular nodes, each of which is given a default color.
4. When a user uploads a file, a loading animation is shown, while the backend app is processing the file.
5. On completion, the browser receives a response payload containing the beats of the input song, and an array containing mood predictions for the input song, one for each beat and one for the song as a whole.
6. 2/3 of the circular nodes on the canvas are colored according to the whole-song-mood (red=aggressive, green=calm, blue=sad, yellow=happy). The final 1/3 of the nodes are given the color individual beat-moods, changing with each consecutive beat.
7. The browser then begins to play the song. With each beat, 1/3 of the nodes change color and a point on the canvas is chosen at random to "push" away all nearby nodes. This animation visually simulates the song's beat track.

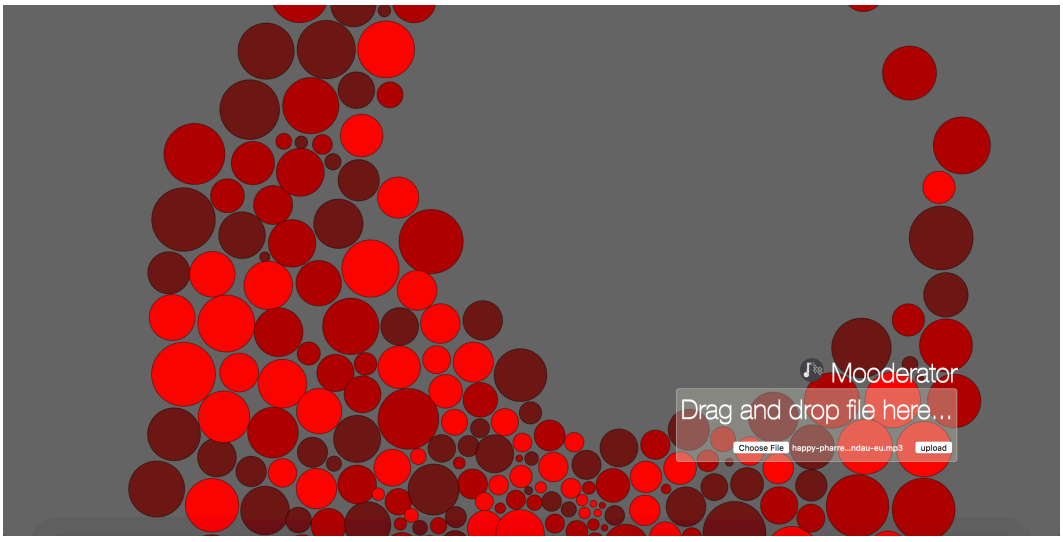
II. The back end: `server.py`

This is the basic server endpoint for the web app. It fields three kinds of requests, (*i.* GET /favicon.ico) the favicon for display in a web browser, (*ii.* POST /song) for uploading and processing songs, and (*iii.* GET /colors) renders the main page of the web app.

After starting up, upon receiving the first request, `server.py` initializes the neural network model and trains it on the built in training set.

When it receives a POST /song request carrying an audio file as form data, it saves the file locally, and passes it to the neural network model, to analyze and use for prediction.

When the network finishes prediction, it returns the resulting mood predictions and the beats of the song in an array, which the server sends back to the front end in JSON format, to display.



III. The intelligence: `classify_mood.py`

The first model we implemented was a Deep Neural Network, by use of the Tensorflow library's 'DNNClassifier' module. Ultimately, it neither performed as well or as consistently as the LSTM model, but it was left connected to the front end app because of its ease of use and *relative* boot speed (the LSTM, in order to reach maximum accuracy, must be trained through upwards of 1000 epochs, which does not make for a desirable user experience). Thus, the DNN implantation is a 'naïve', entertaining and educational foray into the world of mood classification before animation is abandoned for the comfort of accuracy.

The Deep Neural Network model is resembles a traditional neural network, with two hidden layers, each with 250 nodes. Layers are connected to subsequent nodes through a sigmoid activation function. This particular network was trained with 5000 training steps, and was both trained and tested using a batch size of 100.

One major area by which the DNN differs from the LSTM is in its input format. Inputs to the DNN are read from .csv files which contain details of the distribution of input song's pitch, intensity, timbre, and tempo. Specifically, a single row of input to the DNN resembles the following:

```
features = {max_intensity, min_intensity, mean_intensity,
      median_intensity, max_pitch, min_pitch, mean_pitch,
      median_pitch, max_timbre, min_timbre, mean_timbre,
      median_timbre, max_tempo, min_tempo, mean_tempo,
      median_tempo}
```

Ultimately, after training on over 700 different songs (evenly split between happy, sad, aggressive, calm) and testing over 60 songs, mean training accuracy was found to be **0.74**, and mean testing accuracy was found to be **0.72**. The high rate of transference between these two results indicates that, given the level of granularity of the input (distribution characteristics rather than time series inputs), that this model may generalize just as well as it can. Through case by case testing, it was also interesting to see where the DNN made the most mistakes. This model most incorrectly labels 'happy' songs as 'aggressive', an intuitive result, as both 'happy' and 'aggressive' songs tend to share similar levels of both pitch and tempo

IV. *Real* intelligence: `LSTM/lstm_genre_classifier_keras.py`

Our second attempt at mood classification with neural networks came on the shoulders of the keras library's LSTM, which stands for 'long short term memory' network. The LSTM is similar to a Recurrent Neural Network, but additionally, through the use of a discretionary sigmoid neural net layer, it has the ability to add or remove additional information to the cell state, like what it saw in the recent past. Thus, it is 'better' than an RNN (particularly for analyzing time series data) because it can both decipher relevant information *and* keep ahold of some memory.

For example, if the model reads a single 'happy' moment within a 'sad' song, the network has the ability to prevent this data from altering the cell state, given what it's seen so far, i.e., that the song is majority sad. This memory becomes highly relevant when analyzing a time series, such as a piece of music, which is likely to have a similar sound or mood throughout.

The input to this model takes the form of specific features (mfcc, spectral centroid, chroma, and spectral contrast) extracted from time series data.

We took greater care to correct the hyperparameters of this model than of the DNN model. We monitored training set and test set accuracy while altering number of epochs, batch size, and optimization function before settling on the combination of hyperparameters that best generalized on our data. Ultimately, we settled on a batch size of 100, 1000 epochs, and the Adam() optimization function (an adaptation of gradient descent that computes individual adaptive learning rates for different parameters).

The LSTM did exceedingly well at generalizing both training and test data (far better than the DNN model); mean training accuracy was found to be **0.93**, and mean testing accuracy was found to be **0.90**. Similar to what we saw from the DNN, the LSTM shows a high level of transference from training to test accuracy.

Number of Epochs, Batch size 100

	100	150	300	600	1000
Training	Accuracy: 66% Loss: 0.80	Accuracy: 70% Loss: 0.71	Accuracy: 79% Loss: 0.54	Accuracy: 82% Loss: 0.44	Accuracy: 92% Loss: 0.28
Test	Accuracy: 65% Loss: 0.76	Accuracy: 63% Loss: 0.77	Accuracy: 78% Loss: 0.56	Accuracy: 82% Loss: 0.49	Accuracy: 90% Loss: 0.37

Optimization function = Adam, 2 hidden layers, one dense layer with softmax activation function

	35	100	150
Training	Accuracy: 76% Loss: 0.58	Accuracy: 84% Loss: 0.65	Accuracy: 79% Loss: 0.60
Test	Accuracy: 75% Loss: 0.65	Accuracy: 80% Loss: 0.54	Accuracy 76% Loss: 0.58

Epochs= 300, Optimization function = Adam, 2 hidden layers, one dense layer with softmax activation function

	Softmax	Sigmoid	Tanh
Training	Accuracy: 92% Loss: 0.28	Accuracy: 93% Loss: 0.20	Accuracy: 45% Loss: 4.36
Test	Accuracy: 90% Loss: 0.37	Accuracy: 90% Loss: 0.41	Accuracy: 43% Loss: 5.85

Batchsize = 100, Epochs=1000, Adam optimization function, everything else constant

	Adam	SGD	Adamax
Training	Accuracy: 93% Loss: 0.20	Accuracy: 48% Loss: 1.22	Accuracy: 83% Loss: 0.43
Test	Accuracy: 90% Loss: 0.41	Accuracy: 44% Loss: 1.94	Accuracy: 78% Loss: 0.65

Batchsize = 100, Epochs=1000, Softmax activation, everything else constant

Specifically, results analysis showed that, on the 60 test songs, (i) the LSTM network correctly classified all 15 aggressive songs, (ii) correctly classified 14/15 calm songs (1 mislabeled as sad), (iii) correctly classified 12/15 happy songs (3 mislabeled as aggressive), and (iv) correctly classified 13/15 sad songs (2 mislabeled as calm).

Similar to the DNN, it made the most mistakes by mislabeling happy songs as aggressive. Also similar to the DNN, songs that were mislabeled were by and large given a 'similar' label, i.e., into a category which shares like features to the correct one (sad & calm, both share low tempo and medium/low pitch; aggressive & happy, both share high tempo and high pitch).

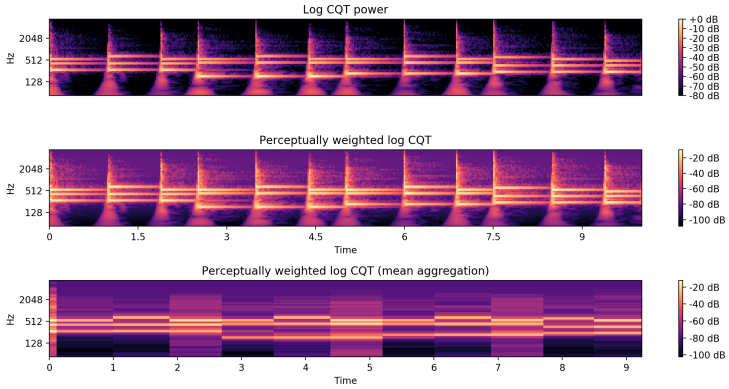
V. Feature extraction: `get_training_features.py`

	Intensity	Pitch	Rhythm
Happy	Medium	Very High	High
Sad	Medium	Very Low	Low
Calm	Very Low	Medium	Very Low
Aggressive	High	High	High

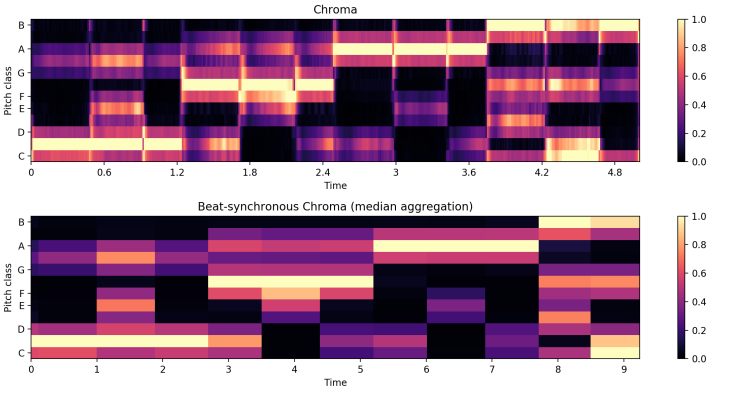
There are vast datasets available for music-genre classification. There are not such datasets, however, for mood classification. Thus, we were the creators of our own dataset. The above table illustrates the measures by which we categorized songs into our four categories, 'aggressive', 'calm', 'happy', 'sad'. We chose these categories because we were confident both that our assessment of these features across the moods were accurate and because they were features that could be easily extracted from time series data.

In `get_training_features.py`, using the librosa python library, we were able to extract said features from our training and test audio files. The DNN took in distribution data of individual beats, thus it synchronized intensity, pitch, and rhythm features with beats in the song's percussive track. In contrast, the LSTM model took in individual feature time series from the song as a whole.

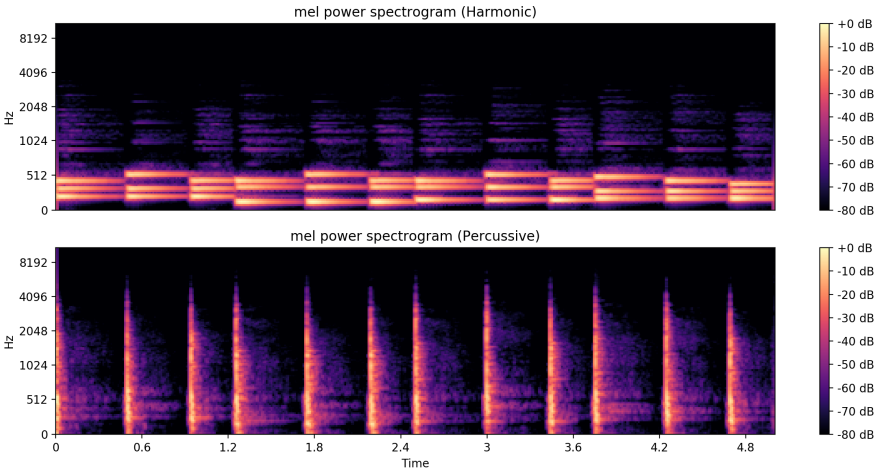
Beat synchronous CQT (intensity)



Beat synchronous Chroma (pitch)

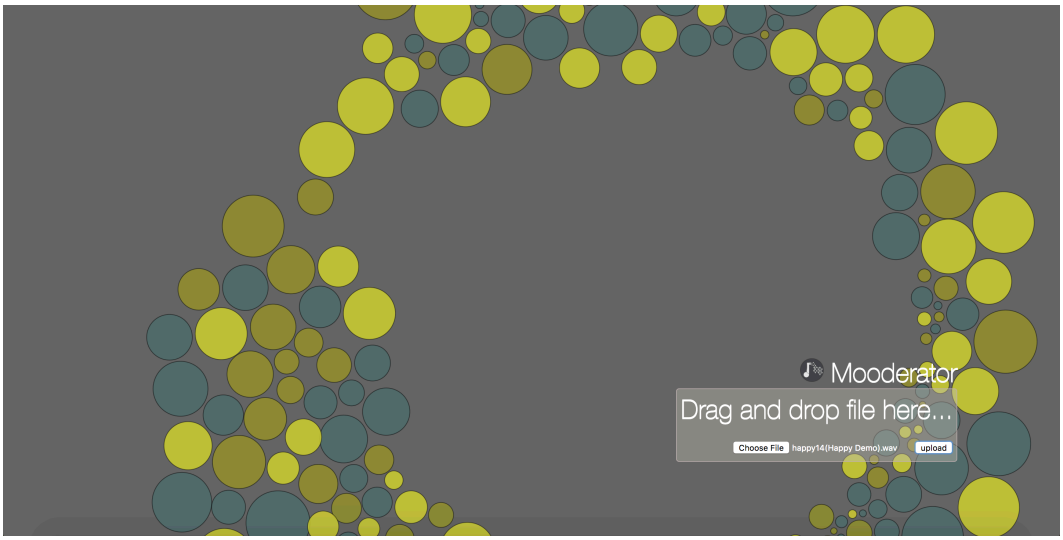


Harmonic/percussive source separation (rhythm)



3: Conclusions

Judging from our results, both of our neural network models (LSTM and DNN) give us similar, passable accuracy levels on an untrained music-mood classification dataset, though the 'memory' of the LSTM seems to perform better for this data specifically, which may be generalizable to its better performance on time series data than the traditional deep or convolutional neural networks. Overall, we have proven that a sophisticated neural network can be used to, at least trivially (among 4 categories), identify relevant features of music and classify songs with their appropriate mood.



Now that we have seen how well an LSTM model performs at classifying music across such a small number of moods, it would be interesting to return to the problem with a more ambitious goal (categorize music across all recognized moods), and a much larger and more proprietary dataset, or a scalable method of collecting song-mood data.

In the real world, reliable music-mood classification has a wealth of applications, from boosted user experience at concerts and listening venues (through procedurally generating pleasing animations) to mood transference from technology to user *("Alexa, play me a sad song"*).

4: References

Choudhary, A, and M Agarwal. "Music Recommendation Using Recurrent Neural Networks." People.cs.umass.edu/, Https://People.cs.umass.edu/~Ashutoshchou/Music_recommendation_using_RNN.Pdf.

Google. "Tensorflow API Documentation: tf.Estimator.DNNClassifier. https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier

Kozakowski, Piotr, and Bartosz Michalak. "Music Genre Recognition." DeepSound, 26 Oct. 2016, deepsound.io/music_genre_recognition.html.

Liddy, T, and A Schindler. Parallel Convolutional Neural Networks for Music Genre and ... 2016, www.bing.com/cr?IG=6EBE8580B26D4E3A93FCFA840339B340&CID=1747FF4697B86E283CE6F4FD96176FE3&rd=1&h=wEVb8k1aY14ebAYHUdc3bwqwLhnCpM2CHSs-5gvvDvI&v=1&r=http://www.ifs.tuwien.ac.at/~schindler/pubs/MIREX2016.pdf&p=DevEx,5067.1.

"Understanding LSTM Networks." Understanding LSTM Networks -- Colahs Blog, colah.github.io/posts/2015-08-Understanding-LSTMs/.