The Mooderator: an AI algorithm that understands the mood of a song

Kate Cook, Marshall Briggs and Noam Grebler Farras

University of California, Los Angeles

Unlock your iPhone, plug in your earphones, go into Spotify and blast some music. This has become an everyday habit for a large amount of people around the world. Music is a way to escape the present and feel something different from our mundane lives. However, a general habit in the world of music is to classify the songs by genre (e.g. Jazz, Hip Hop, etc…). This clearly contradicts the idea of listening to music to feel something as a jazz song can bring us to feel the same mood than a hip-hop song, but two jazz songs might awaken very different moods in us. Through our research of past literature on this matter we found two papers that inspired our project. The first paper (Kim and André) investigates a musical induction method that spontaneously leads subjects to real emotional states. The second paper (Yang and Lee) we found explains introduces an algorithm that tries to understand the mood of a song by analyzing its lyrics. Knowing that a song can change our mood and that an algorithm can understand the mood of a song through its lyrics we decided to create an algorithm that understands the mood of a song by analyzing all of its components (e.g. timber, pitch, etc..) but lyrics. This idea seemed interesting to us as, by simply using this app we could help people feel the way they want. From this idea we create The Mooderator, an neural network that is able to understand the mood of a song by simply analyzing its musical components and ignore the lyrics.

Our Solution

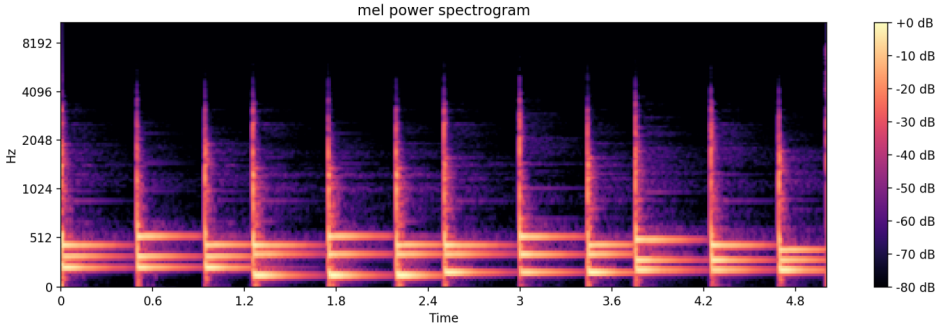
Our Solution is based in several steps:

1) The code directly intakes a list of 30 seconds audio files which we took from a library of songs that were classified by genre and we manually classified them by mood and then names each file by their mood and number e.g. happy079.wav. The code does this for training and validation data.

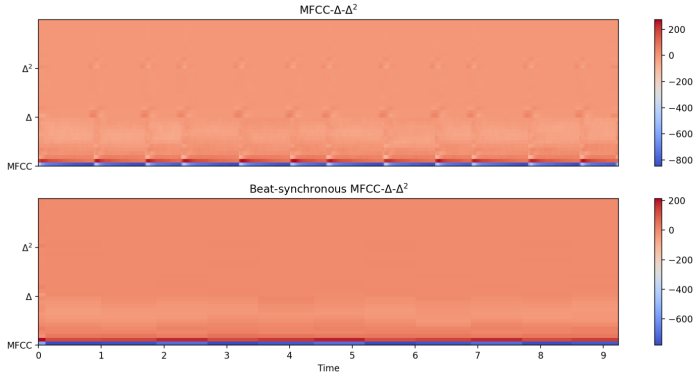
2) It then grabs each file individually and serializes it by breaking it up into four basic musical features. We used librosa (Python audio analysis lib.), to extract four identifying features from various pieces of music: pitch (Chromagram), intensity (loudness, by Constant Q-Transform), timbre (MFCC), and tempo for each beat. It then takes an average of all the beats to give a categorization for the whole file for each feature. We end up saving this numbers in arrays.

   These are some spectrograms showing our work:

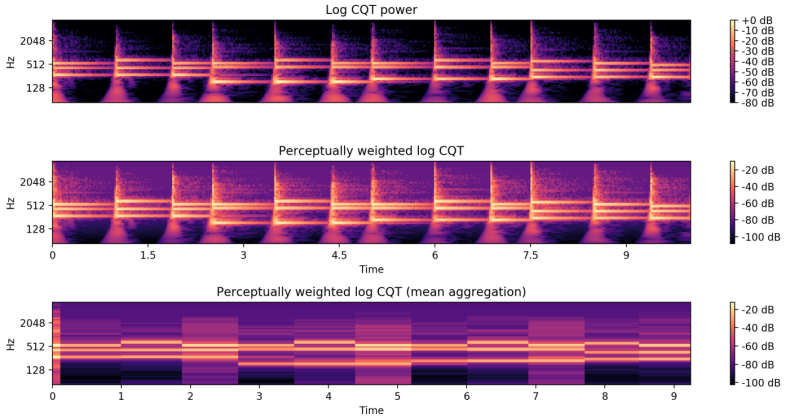
   a) Mel power spectrogram



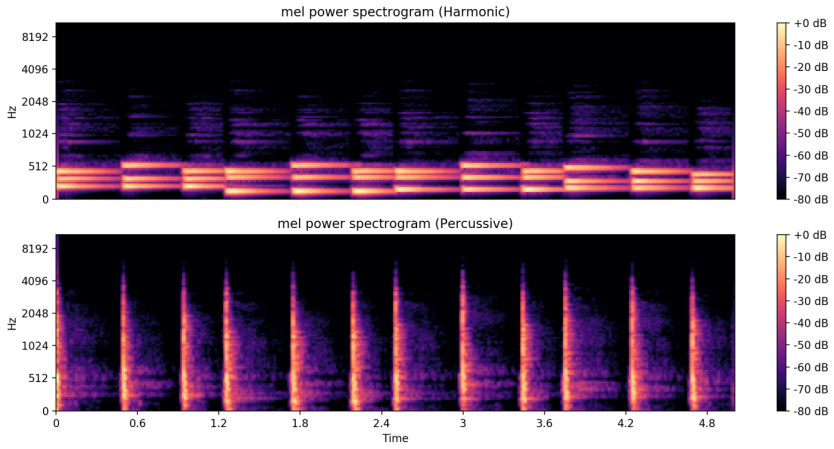
   b) Beat-synchronous MFCC (Timbre)



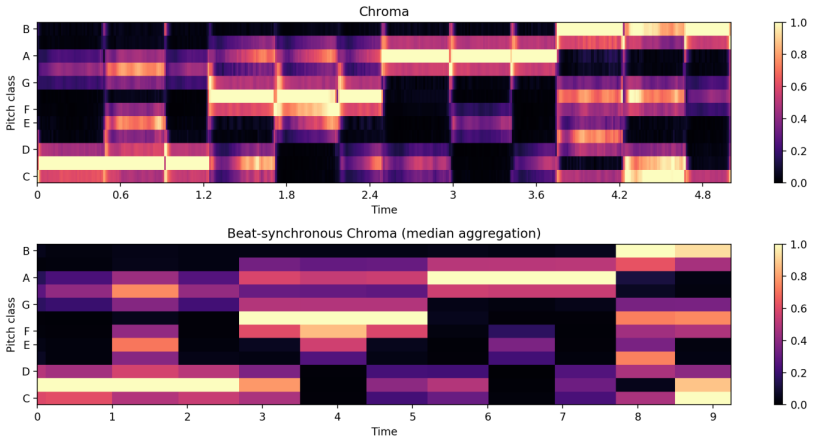
c) Beat-synchronous CQT (Intensity)



d) Percussive vs. Harmonic components



e) Beat-synchronous Chroma (Pitch)

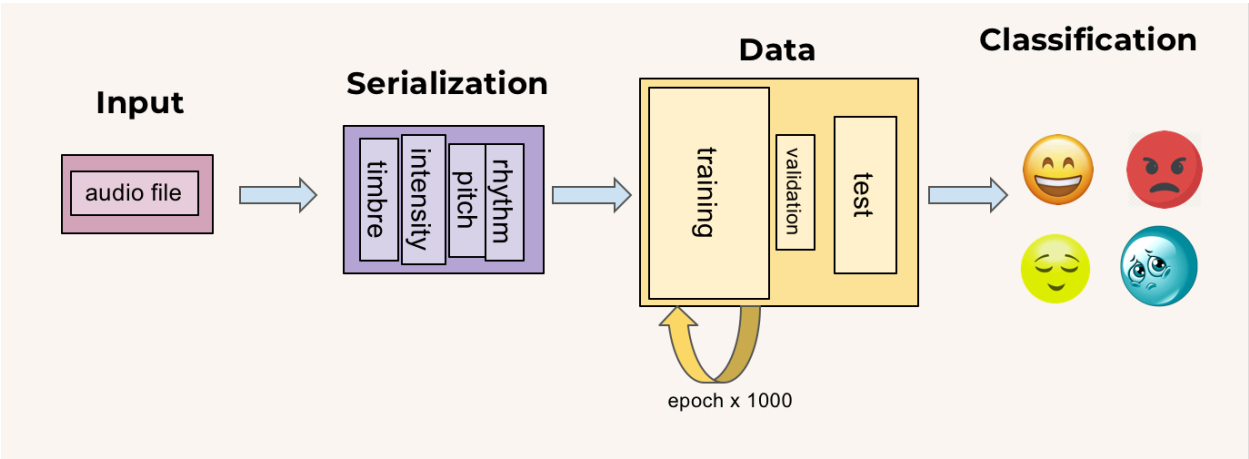


f) This tables helped us his classify how each feature of a song translates to its mood:

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

3) Pass this serialization through our model and then use it as a training/validation data which mean that the f vector is the serialization and the expected result, g vector, is the name of the song.
4) Then once the model is ready we take an audio file and drop it into our Web-App
5) Same serialization as in step 2.
6) Use this serialization to figure out the mood of each beat and the mood of the whole song.
7) This information is then passed to the front end which outputs some graphics that are dependent on mood/beat.

Here is a graph that helps us picture this process



Our Model

For the sake of this project we decided to use two different Neural Network models one is a Long Short-Term Memory and the other one is a Deep Neural Network. The one with which we got the best results is the LSTM for which we used a Keras module to implement it. An LSTM is an improved neural network that has the capacity to decipher relevant information and additionally has a sort of memory. This memory that LSTM networks have make them a great fit for sequential information such as songs thanks to its capacity to remember what happened at the beginning of a song all the way through the end. This is how our LSTM works:

1) The LSTM has the capacity to update a cell state
    a) At each iteration it decides what information it will remember/forget
    b) This system works with 3 "gates":
        i) input gate
        ii) forget gate
        iii) output gate
    c) A gate is composed of:
        i) sigmoid layer which chooses
            (1) If to update a cell
            (2) And if we update it, it says by how much we need to update it.
        ii) A function that does a pointwise multiplication to update the state of a cell
2) We used as input every beat of our audio files
3) Memory of the cell also called cell state which updates for each beat of our files
4) Sigmoid layer also known as forget layers that determines:
    a) What part of the information should the LSTM forget
    b) What part of the information should not be forgotten
5) Input gate which guarantees the importance of the data that is added to a cell.
    a) This gate creates a vector of units to add to the cell state with the help of the tanh function
6) Output gate:
    a) Makes use of the both the input gate and the sigmoid layer to make sure the LSTM outputs relevant information
    b) This information is then sent to the hidden layers of the subsequent cell.
7) Hidden layer:
    a) This layer consists of a matrix that changes
        i) It receives inputs
        ii) Adapts itself to provide better output
        iii) Resends information to input:
            (1) Can be new input
            (2) OR previously used input that is fed back

On the other side we made a second model that used a Deep Neural Network. I won't get too much into detail as this model performed worse than our LSTM and therefore is less interesting for our project. This model is much more similar to the neural networks we learned in class. We used Tenserflows library to implement it. As it doesn't count with the memorization part our results in accuracy of test did not reach the 80%.

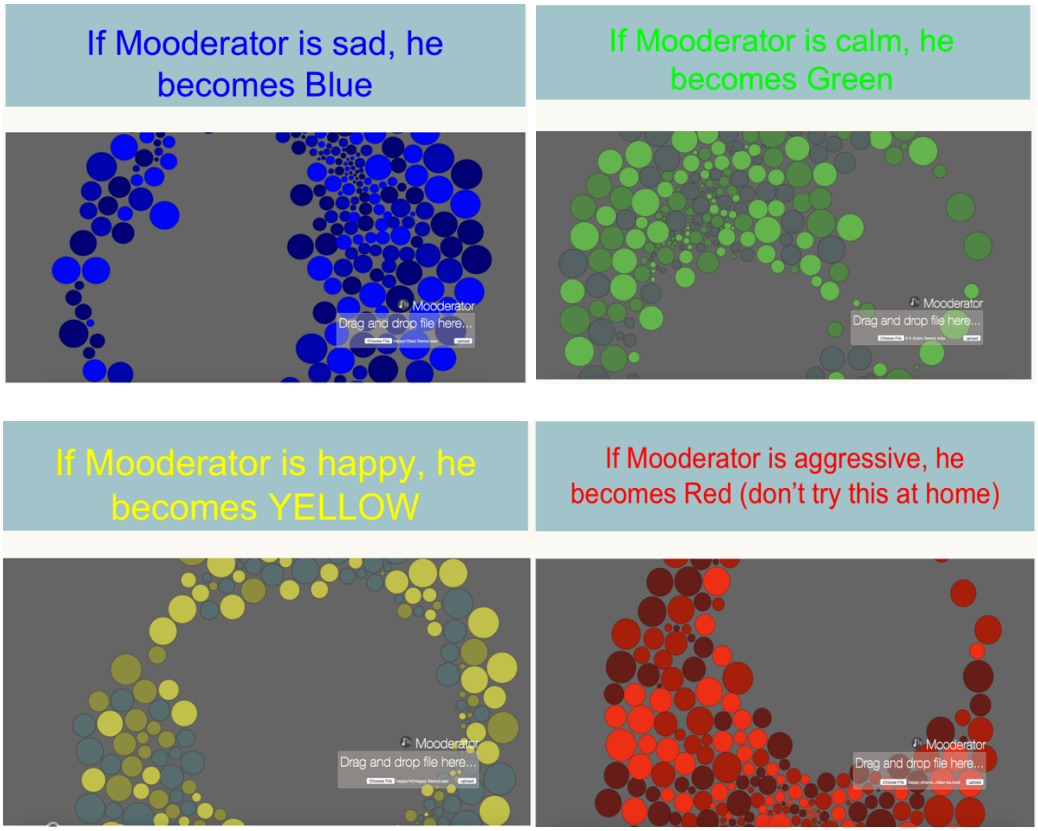
Front End

In order to translate our algorithm into a tangible outcome we created a front end in the form of a web app. This app generates graphics according to the mood of the song. It creates a number of balls that are colored depending on the mood:

- 2/3 of the balls are colored depending on the general mood of the song
- The other 1/3 is colored depending on the mood of the specific beat

At the same time as the balls appear they move with a random movement at each beat and the song plays as background.

The colors are the balls are filled with are following:



Results

These are some of the tables we built with our group from the results we got:

Number of Epochs, Batch size 150

	100	150	300	600	1000
Training	Accuracy: 68% Loss: 0.82	Accuracy: 70% Loss: 0.75	Accuracy:78 % Loss: 0.63	Accuracy: 87% Loss: 0.42	Accuracy: 94% Loss: 0.29
Test	Accuracy: 63% Loss: 0.85	Accuracy: 73% Loss: 0.69	Accuracy: 77% Loss: 0.63	Accuracy: 86% Loss: 0.47	Accuracy: 85% Loss: 0.48

Batch size = 150, Optimization function = Adam, 2 hidden layers, one dense layer with softmax activation function

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

Batch size

	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

Activation functions for hidden layer:

	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

We use softmax to get probabilities for each classification Softmax provides a probability

Optimization Functions:

	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

These results show that our model ran with a 93% accuracy for our training data and 88% for our test and only 0.22 training loss and 0.51 test loss. These are good numbers that not only show the possibility of classifying music by mood but also that our model is well built.

During test our code was able to classify:

- all 15 aggressive songs correctly
- all calm songs expect for one which it classified as sad
- it classified 2 sad songs as calm
- it classifies 3 happy songs as aggressive

Limits and possible improvements

The main limits we faced were during the classification of the songs. Agreeing of what is a sad song versus what is a calm song without taking into account the lyrics was somewhat complicated. We had different opinions in many cases and when we divided the songs, and each sorted some we clearly felt it affected our model as there was no consistency. We then decided to go through all of the songs together and only select the ones where we all agreed. Our accuracy immediately increased by a significant amount.

We found several ways we could improve our model as increasing the amount of songs in each category, training, validation and testing for example. This would not only make our model more accurate but also increase the number of genres its able to recognize and classify in a mood. We could also increase the number of different moods in order to be more specific and allow the people to select exactly what they want to feel. Finally, one of the key features we could add to our model to make it more precise is the recognition of lyrics as they did in the paper Yang and Lee.

Conclusion

This experiment shows that a mood classification of music is possible and can be done accurately. Our LSTM model worked well and gave us promising results. We believe this could be applied in large scales and help people guide their mood.

As we saw one of the limits is that even though we have “similar profiles”, all students at the same university with the same major and of same age, we still greatly differed in our mood classification. We could only imagine how much it would change with someone much more different. This might mean that there might not be a universal definition of what a happy, sad or aggressive song is and therefore makes our model hard to be applied universally. However, to solve this problem we could make every person create their training data and then people would have their personalized algorithm to choose songs by their mood. Which gives us hope in terms of the utility of the mooderator.

We think that this experience opens the door to a new way of listening music, a personalized way…

Be happy, be mooderated!

References

D. Yang and W. S. Lee, "Music Emotion Identification from Lyrics," 2009 11th IEEE International Symposium on Multimedia, San Diego, CA, 2009, pp. 624-629. doi: 10.1109/ISM.2009.123 http://ieeexplore.ieee.org/abstract/document/5363083/

J. Kim and E. André, "Emotion recognition based on physiological changes in music listening," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 12, pp. 2067-2083, Dec. 2008. doi: 10.1109/TPAMI.2008.26 http://ieeexplore.ieee.org/abstract/document/4441720/