Using brain-computer interfaces and Spotify's music data to recommend music and predict user engagement - progress in Hilary Term and future plans

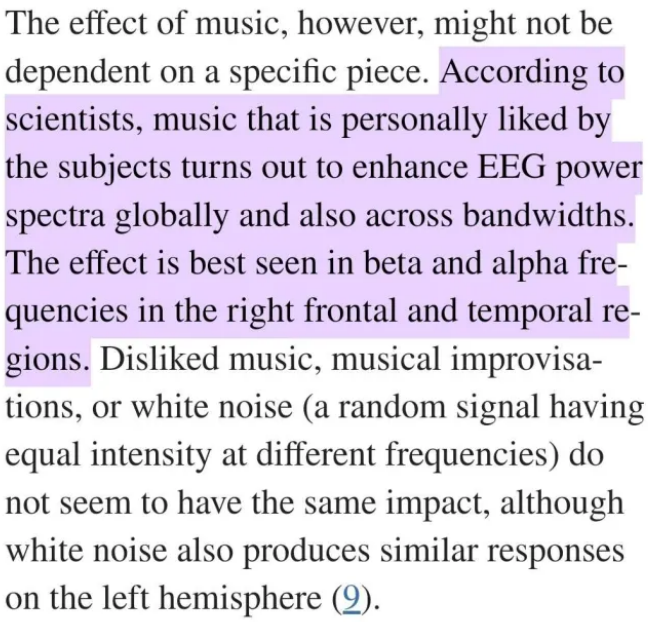
By Aditya Khanna, April 2024 - For OxNeurotech

Preliminary knowledge

Spotify holds an expansive and extremely detailed music database. For every song you can listen to on Spotify, there is a set of attributed "audio features", these include Energy, Valence, Key, Tempo, Danceability, Liveness, Acousticness, Speechiness, Instrumentalness and more. These features can come in extremely handy for a variety of analysis tasks. Spotify uses this data, alongside an obviously massive database of user engagement with its app, to bring people remarkable recommendations for music it thinks users will enjoy. Many users on the internet have reported things like "Spotify knows me better than I know myself!" because of this, and this has been my experience too, considering a lot of the music I have explored and that deeply resonated with me, was infact recommended to me by Spotify. Spotify uses various user-engagement metrics to determine these recommendations, as well as of course it's large database on lyrics, news and audio features. My aim with this project was to try to use one extra data source as a user-engagement metric for music recommendation - and that would be EEG signals. EEG headsets work by measuring the electric fields caused by postsynaptic dendritic currents (i.e. E-fields caused by ion flow in dendrites) with respect to a ground electrode. The EEG headset we have used for this project so far has been the Neurosity crown.

Motivating research

There is clear evidence that music that is personally liked by people tends to result in higher brain activity, especially in right frontal and temporal regions.

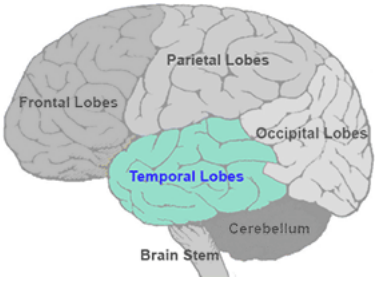


The effect of music, however, might not be dependent on a specific piece. According to scientists, music that is personally liked by the subjects turns out to enhance EEG power spectra globally and also across bandwidths. The effect is best seen in beta and alpha frequencies in the right frontal and temporal regions. Disliked music, musical improvisations, or white noise (a random signal having equal intensity at different frequencies) do not seem to have the same impact, although white noise also produces similar responses on the left hemisphere (9).

Taken from
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6130927/#:~:text=According%20to%20scientists%2C%20music%20that,ri

Therefore, we can use this in combination with trained Machine Learning algorithms to determine whether or not someone likes a song, and perhaps be able to tell the degree to which they like a song!

This data would be an extra user satisfaction metric that Spotify could use to better inform recommendations! So far, we have used songs that people like and dislike in combination with audio features to predict whether someone would like or dislike an unfamiliar extra song. The technical details and difficulties we faced in doing this will be explored in the next few sections.



Extracting and analysing Spotify Data

The first two code blocks in this section are rather boring, so feel free to skip them.

Here's some simple code to extract audio features data:

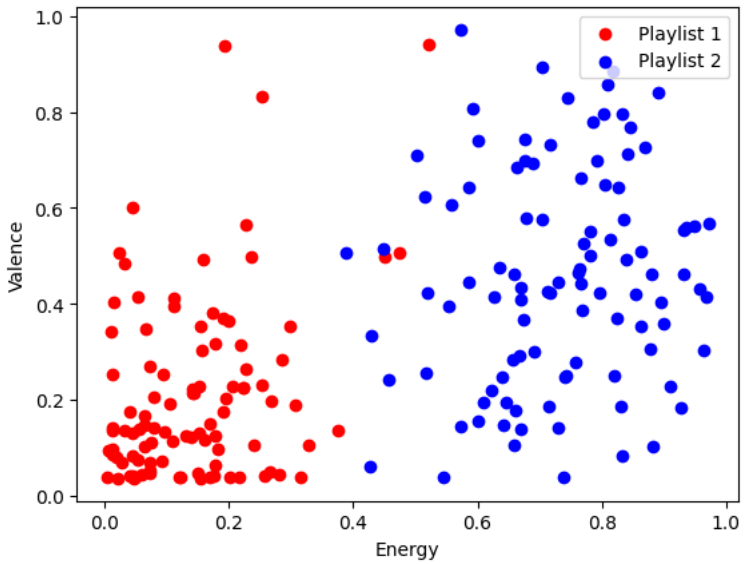
```
!pip install spotipy
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from statistics import mean
# <https://developer.spotify.com/documentation/web-api/reference/get-audio-features>
# in case any seem relevant

client_id = "SPOTIFY API CLIENT ID"
client_secret = "SPOTIFY API CLIEANT SECRET"


client_credentials_manager = SpotifyClientCredentials(client_id=client_id, client_secret=clien
t_secret)
spoti = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

# url of the playlist in question, can change into an input type thing whatever
# put your playlist here
playlist_url1 = 'https://open.spotify.com/playlist/1iscqLcesqSwjVSuc7pJLM'
playlist_url2 = 'https://open.spotify.com/playlist/37i9dQZF1DX76Wlfdnj7AP'




def get_playlist_tracks_with_audio_features(playlist_url1):
    playlist_id1 = playlist_url1.split('/')[-1]
    #get the url id which in this case would be whats after the final slash
    results = spoti.playlist_tracks(playlist_id1)
    playlist_data = {}

    for track in results['items']:
```

```
        track_name = track['track']['name']
        track_id = track['track']['id']

        audio_features = spoti.audio_features(track_id)

        if audio_features:
            playlist_data[track_name] = audio_features[0]

    return playlist_data

playlist_data1 = get_playlist_tracks_with_audio_features(playlist_url1)
playlist_data2 = get_playlist_tracks_with_audio_features(playlist_url2)
#print playlist tracks and audio features
audio_feats_for_each_track = {}
mean_relevant_data = {}
tracks = []
means = []
#print(playlist_data)
def get_mean_value(playlist_data, audio_feature):
    all_feats = []
    for track_name, audio_features in playlist_data.items():
        playlist_feat = audio_features[audio_feature]
        all_feats.append(playlist_feat)
    return sum(all_feats)/len(all_feats)

def get_scatter(playlist_data, audio_feature1, audio_feature2):
    feat1 = []
    feat2 = []
    for track_name, audio_features in playlist_data.items():
        playlist_feat1 = audio_features[audio_feature1]
        playlist_feat2 = audio_features[audio_feature2]
        feat1.append(playlist_feat1)
        feat2.append(playlist_feat2)

    return [feat1,feat2]
```

These functions can then allow us to plot and compare these two playlists in terms of their features, like so:

```
import matplotlib.pyplot as plt

scatter1 = get_scatter(playlist_data1,'energy','valence')
scatter2 = get_scatter(playlist_data2, 'energy','valence')

plt.scatter(scatter1[0], scatter1[1], color='red', label='Playlist 1')
plt.scatter(scatter2[0], scatter2[1], color='blue', label='Playlist 2')

# Adding labels and legend
plt.xlabel('Energy')
plt.ylabel('Valence')
plt.legend()
```

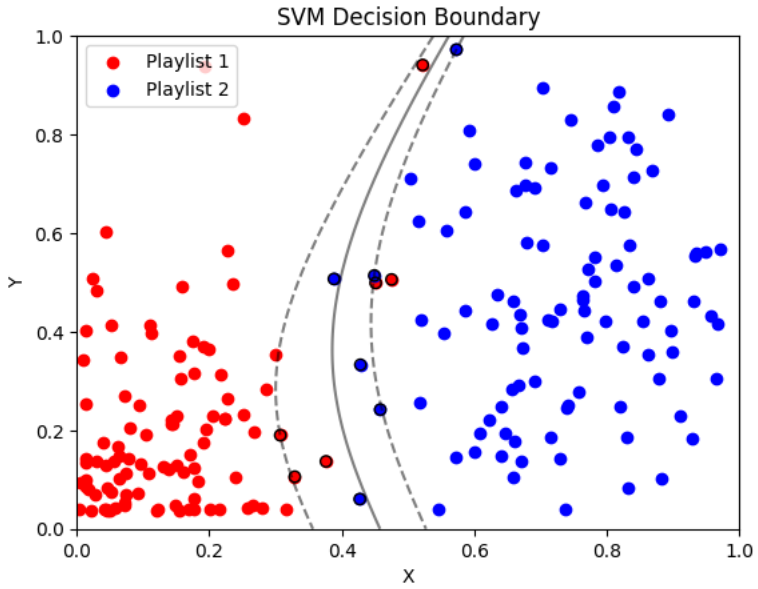
This allows us to get the plot below, showing a two distinct clusters in energy and valence. If you were to go to the playlist links we put in the first code block, you'll see why :).



We can then use a Support Vector Machine (a basic supervised machine learning algorithm) to draw a nonlinear decision boundary between these two clusters, allowing us to then pick which playlist a new song with a given Energy and Valence belongs in!

```
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt

# data
x_a = np.array(scatter1[0])
y_a = np.array(scatter1[1])

x_b = np.array(scatter2[0])
y_b = np.array(scatter2[1])

# feat matrix x, labels Y
X = np.vstack((np.column_stack((x_a, y_a)), np.column_stack((x_b, y_b))))
Y = np.hstack((np.zeros(len(x_a)), np.ones(len(x_b))))

# the model itself
svm_model = svm.SVC(kernel='poly', C=1.0)

# train it
svm_model.fit(X, Y)


xx, yy = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))
Z = svm_model.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
# plot the data points
plt.scatter(x_a, y_a, color='red', label='Playlist 1')
plt.scatter(x_b, y_b, color='blue', label='Playlist 2')

# plot the decision boundary
plt.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

# highlight support vectors
plt.scatter(svm_model.support_vectors_[:, 0], svm_model.support_vectors_[:, 1], facecolors='none

plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('SVM Decision Boundary')
plt.show()
```



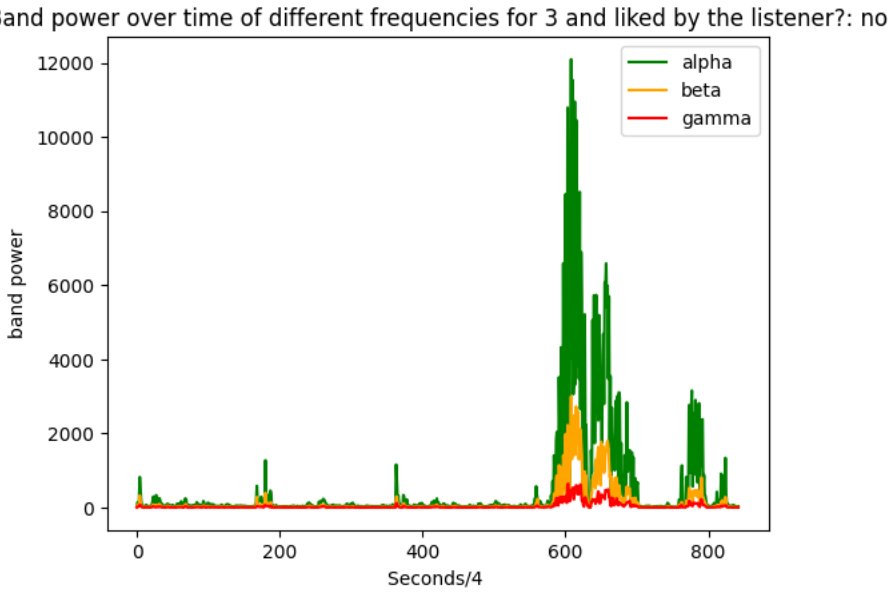
Then finally, to predict:

```
new_data_point = np.array([[0.8, 0.0]])
#we can predict which playlist an arbitrary song belongs in using this.
predicted_class = svm_model.predict(new_data_point)
class_labels = ['Playlist_1', 'Playlist_2']
predicted_class_label = class_labels[int(predicted_class)]
print(f'The predicted class for {new_data_point} is: {predicted_class_label}')
```

The EEG headset

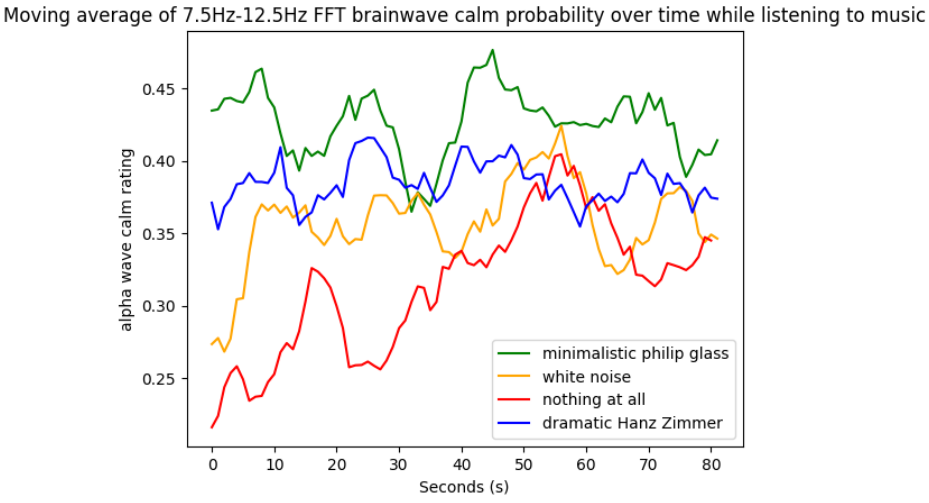
The EEG headset we have used for this project so far has been the Neurosity crown, a consumer grade headset with 8 channels.

The EEG headset was where we began noticing our first difficulties. Despite using numerous filters and algorithms applied to filter noise (e.g. a notch filter set at 50 Hz to avoid the UK mains alternating current acting as a source of interference), we still ended up with noisy and rough data. For example, extracting alpha, beta and gamma brainwave band powers using the Neurosity API:

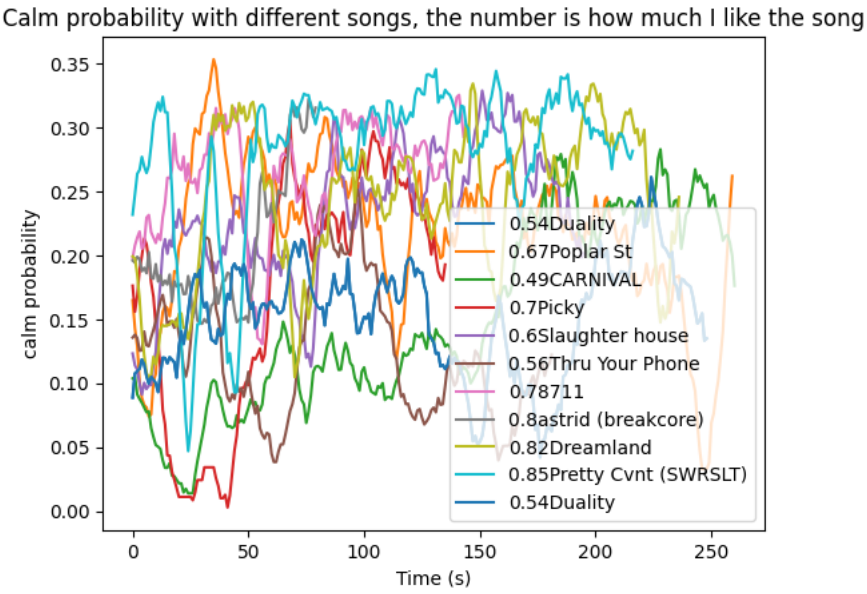


There is clearly a lot wrong with this graph, and it's almost impossible to extract meaningful data out of it. We therefore tried a different tack, and attempted to use one of Neurosity's inbuilt metrics determined using their own proprietary machine learning algorithms. This was the calm data, and we found that for songs enjoyed more by the user, there is an overall higher mean calm as well as often less variance. We thus thought that standard deviation and mean were adequate features to use.

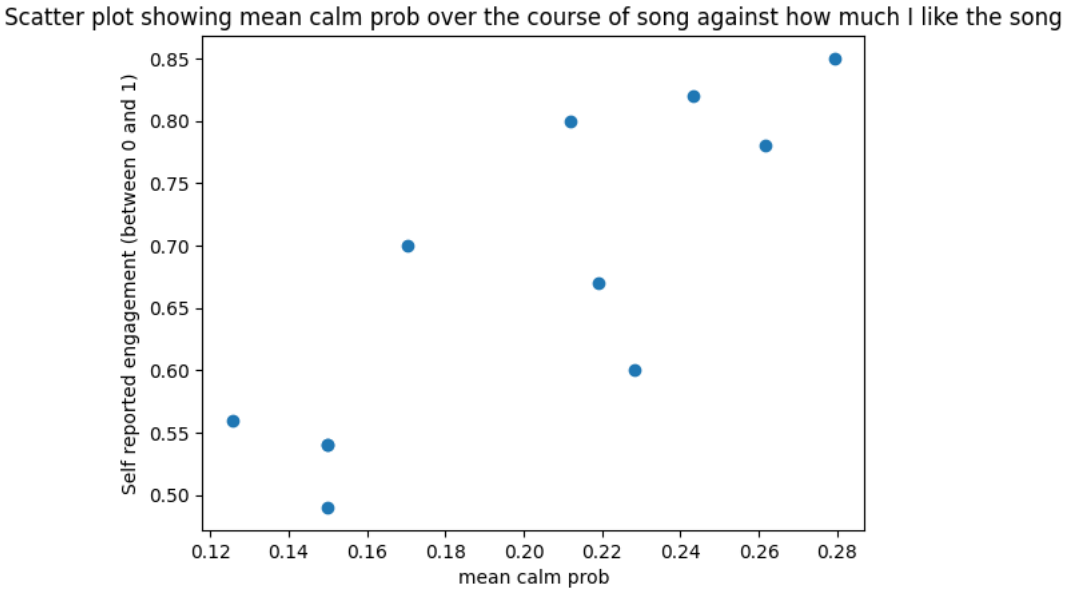
This was our first experiment using this data, with our society president, John Lee as the test subject:



Building on top of this and with more songs, now trying this out on myself:



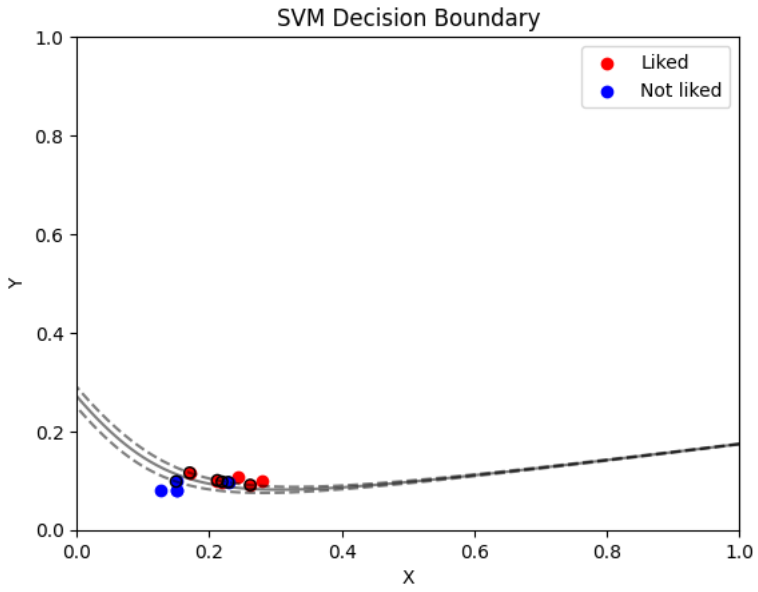
Plotting the mean calm probability wrt how much I like the song, we get a scatter plot with a very weak positive correlation:



But we could still use this to find an imprecise measurement of musical engagement using extremely simple linear regression.

A machine learning algorithm to determine a boolean “song liked” or “song not liked”

This boolean classification task seems like the perfect job for an SVM. I have already used the SVM once (and I will do so again, it’s the simplest choice with the data we have).



We can use exactly the same algorithm as before.

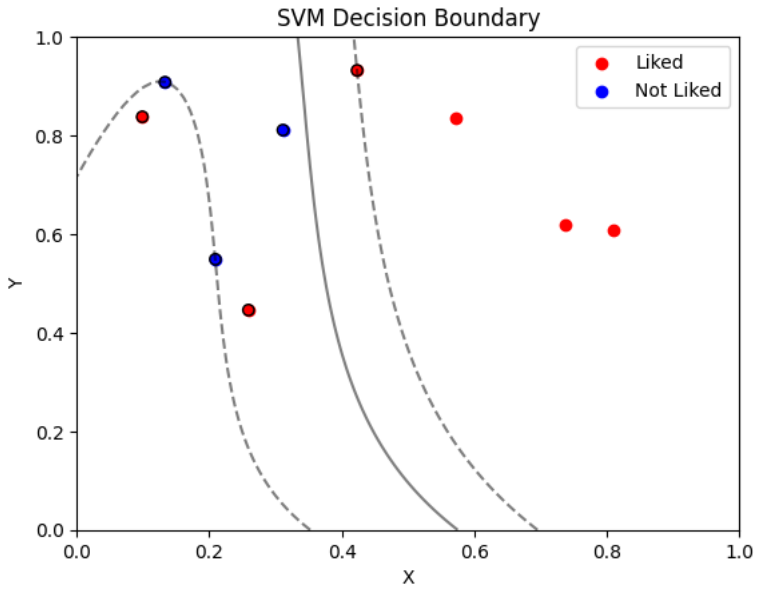
```
new_data_point = np.array([[0.1, 0.1]])
# [0] refers to mean of my calm data, [1] refers to standard deviation of my calm data
# we can predict whether I will like or dislike an arbitrary song based on this
predicted_class = svm_model.predict(new_data_point)
class_labels = ['Liked', 'Disliked']
predicted_class_label = class_labels[int(predicted_class)]
print(f'The predicted class for {new_data_point} is: {predicted_class_label}')
```

And here we have a simple classifier to tell if someone likes a song or not.

Given that we have the data of songs that someone likes or dislikes, can we predict whether they will like or dislike an arbitrary other song?

We have the Spotify audio features for all the songs, this is something we could theoretically plot on an n dimensional graph, I.e. we can store the songs as n-dimensional vectors with labels “liked” or “disliked”. Then to classify into these categories we could draw an n-dimensional nonlinear hyperplane to separate them. Do you see where I’m going with this? That’s right! Another SVM. We could also have used a multilayer perceptron, but I think considering how basic the data we have is, an SVM would work better.

Considering energy and valence (the 2D case, this can be extended to as many dimensions as there are audio features, but we can’t display that):



The issue with this is, people's preferences for energy and valence for example tend to depend on their mood. So to turn this into a viable system, it's not enough to simply train the musical engagement predictor once. What we'd require is that it trains during a music listening session with the headset on. So someone is listening to music and we use EEG to check whether they like it or not. If they skip the song, that immediately tells us they definitely don't like it.

As you can see, our data is a little sparse. I did experiment on a friend, but over there I am displaying only data from myself.

But either way, we need a lot more data to improve this prediction model.

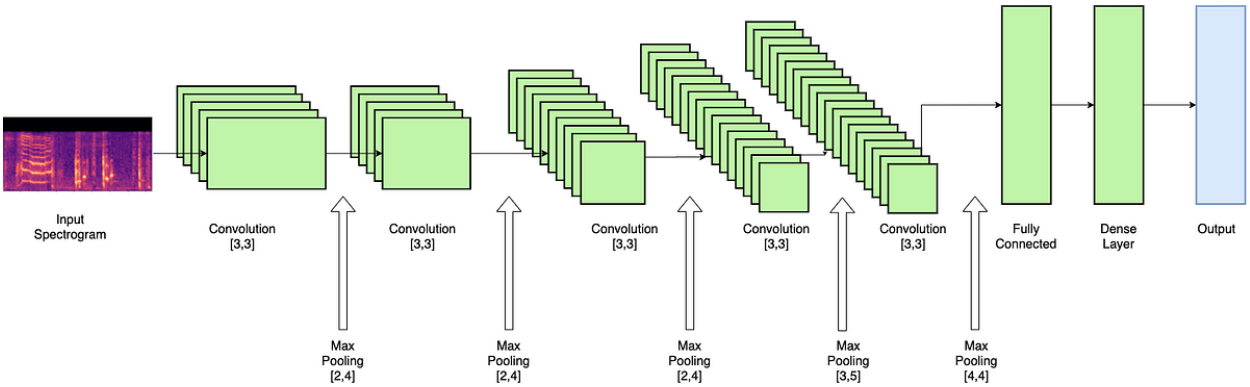
One interesting thing that came of experimenting on my friend was, at one point I noticed the calm data spike to a lot. Anything above 0.3 is considered significant. He was at around 0.85 at one point. And this was while listening to a song he really did not seem to like "dumbest girl alive - 100 gecs" (100 gecs is a rather polarising musical artist, I have noticed). Anyway I asked him "what on earth is going on??" and he replied that he'd just been thinking of something funny. I found that quite amusing and interesting. I felt that was worth mentioning because the only difference between science and messing around is with science you write stuff down. This entire project (so far) has just been me messing around and now I'm writing stuff down so it was a scientific project right?!

Evaluating things and future endeavours

So we have made *some* progress, however the machine learning algorithms we've implemented so far have been extremely basic (not to mention all being the exact same algorithm with different features and maybe extra dimensions). I think this is fitting however, given that the data we've got from the Neurosity crown has also been rather basic, since it is proprietary calm data and does not exactly represent how much a song is liked, the way using the band powers would directly correspond to the motivating research on bioelectrical oscillations in the brain due to music.

The biggest next step would therefore be collecting more data, and collecting *different* data. The data I would particularly like to collect is simply the raw EEG data in all channels, and especially corresponding to the channels nearest the right-frontal temporal lobe. We could then apply a convolutional neural network to the spectrogram of songs that are liked or disliked (this is infact how Spotify . We could further try regression algorithms to see if we can better estimate *how much* a song is liked. This is difficult to do properly with an 8-channel headset such as the Neurosity crown.

An example of CNNs being applied to song spectrograms (I think it could be handy to try to apply this to EEG spectrograms of a song too):



Taken from https://towardsdatascience.com/music-tagging-using-convolutional-recurrent-neural-networks-e26a9856f9f5

Another way to extend this project from here is to use emotion recognition datasets and try to find a value of the user's valence. We could then recommend a song to match this valence. I.e. happy songs when they're feeling happy, sad songs when they're feeling sad.

Furthermore, a much more ambitious extension of the project and something I doubt we have the ability to do currently is to isolate musical parts of the song that seem to carry the most engagement and see if other songs with similar parts can be found. I'm not entirely sure how we would practically implement this, but to give an example, at one point in my life I particularly liked this song called "Set Free" by an artist called Veorra. Later, I listened to another song called "711" by Emei. Listening to both of these songs one after another reveals a similarity that is difficult to put into words, and I believe if someone particularly likes one part of one song and we can find another song containing a somewhat similar part then that song will again be a song that they should like.

Another issue with the basic premise of this project that I've found is the lack of objectivity. It's really difficult to exactly pinpoint how much I like a song on a spectrum from 0 to 1. It's also very difficult to tell if the prediction of whether I will like a song or not is actually good because our system is working or because I simply have bad taste in music and will gladly listen to anything at all over the voices in my head.

Overall, I have really enjoyed working on this project given my interests in music and psychology/neuroscience. I have learned quite a lot due to this so far across a range of fields from neuroscience to machine learning, and I'm excited to begin work on it again next year, hopefully with a better knowledge of machine learning and artificial intelligence to improve the system.

Thanks for reading, and if you've made it this far that must mean you're quite interested in the project. If so, and you feel like you've got something to add (e.g. actual machine learning knowledge or any new ideas), please sign up!