JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

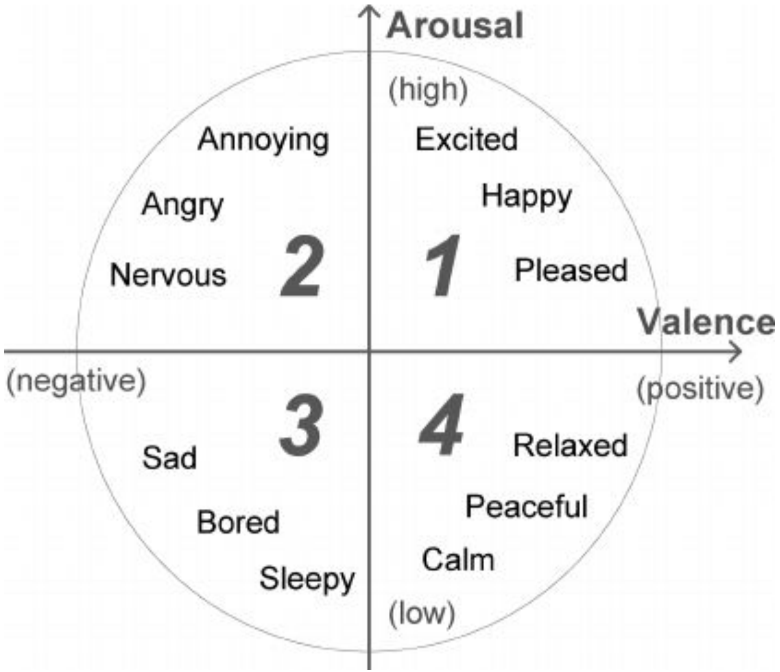
Music Emotion Predictions Using Neural Network and ABC Algorithm

PRESENTATION BY:
XINYU CHANG, XIANGYU ZHANG, YULU RAN, HAORUO ZHANG

EN.553.602.01.SP24

Brief Review

- Through this project, we want to explore a way to better predict a piece of random audio into a specific emotion region located in Russell's Emotion Quadrant.



Procedure Overview

1. Data Featurization - DONE
2. Data Preparation for Model - DONE
3. Model Training – Next Step
4. Evaluation – Next Step

What We Have Done

Feature Selection

- know what are the top-100 features Panda R. and his group used

```
# join features and feature_lookup to see what exactly are the top 100 features.
features_df = pd.DataFrame(features, columns=['Feature'])
merged_df = pd.merge(features_df, feature_lookup, how='left', on='Feature')[["Feature", "Name", "Toolbox", "Category", "Description"]]
merged_df.head()
```

	Feature	Name	Toolbox	Category	Description
0	F0525	MFCC1 (mean)	Marsyas	Tone Color	to be added later
1	F1152	FFT Spectrum – Average Power Spectrum (median)	PsySound 3	Tone Color	to be added later
2	ORIGINAL–TEXTURE–Musical Layers (Mean)	NaN	NaN	NaN	NaN
3	F1166	FFT Spectrum – Spectral 2nd Moment (median)	PsySound 3	Tone Color	to be added later
4	F0133	Spectral Skewness (std)	MIR Toolbox 1.6.1	Tone Color	Coefficient of skewness. The third central mom...

- Examined features: Panda used in his paper vs. Librosa package
- there are only a few overlapping features b/w Panda and his teams' work and Librosa
- **Decision**: Using Librosa to convert our own features

Featurization

- Using Librosa to convert all 900 30-sec audio clips into the 19 features
- Including 15 spectral features and 4 rhythm features.

Rhythm features

tempo	Estimate the tempo (beats per minute)
tempogram	Compute the tempogram: local autocorrelation of the onset strength envelope.
fourier_tempogram	Compute the Fourier tempogram: the short-time Fourier transform of the onset strength envelope.
tempogram_ratio	Tempogram ratio features, also known as spectral rhythm patterns.

Spectral features

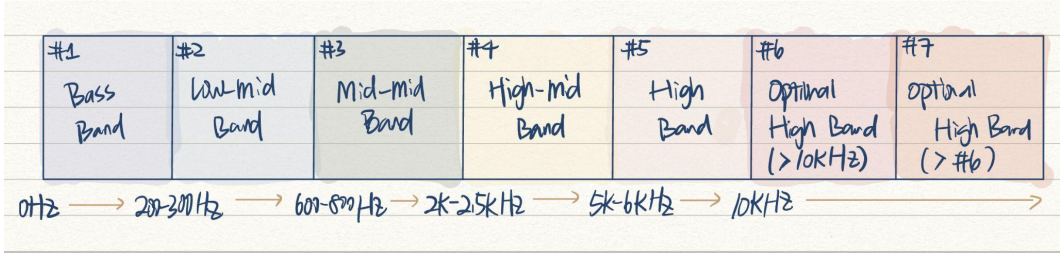
chroma_stft	Compute a chromagram from a waveform or power spectrogram.
chroma_cqt	Constant-Q chromagram
chroma_cens	Compute the chroma variant "Chroma Energy Normalized" (CENS)
chroma_vqt	Variable-Q chromagram
melspectrogram	Compute a mel-scaled spectrogram.
mfcc	Mel-frequency cepstral coefficients (MFCCs)
rms	Compute root-mean-square (RMS) value for each frame, either from the audio samples y or from a spectrogram S.
spectral_centroid	Compute the spectral centroid.
spectral_bandwidth	Compute p'th-order spectral bandwidth.
spectral_contrast	Compute spectral contrast
spectral_flatness	Compute spectral flatness
spectral_rolloff	Compute roll-off frequency.
poly_features	Get coefficients of fitting an nth-order polynomial to the columns of a spectrogram.
tonnetz	Compute the tonal centroid features (tonnetz)
zero_crossing_rate	Compute the zero-crossing rate of an audio time series.

Adding Features

- Adding features allows us having more data to train NN
- Larger dataset can feed NN to train better, in general (BUT not guaranteed).
- **Decision**: Calculate meaningful statistical values of each feature
  - E.g. Min, Max, Mean, Std, Range, Skewness, Kurtosis
  - Kurtosis: shape of the tails and peak of the distrb. compared to normal distrb.
- **Actions**: Examine each feature to decide features we want to generate

Adding Features-Example

- “**Spectral_contrast**” – shape: (7, 1295) (n_bands, n_frames)
- n_bands - # of frequency bands analyzed (usually 6 or 7, including bass band)
- Each element in array: the contrast val. in dB for a specific freq. band and time frame
- Higher val. indicate greater contrast b/w spectral peaks and valleys within that band.
- **Decision**: Add mean, std, min, max, skewness, kurtosis, range
- Range: difference b/w the max and min across all freq. bands and time frames



Data Preparation for Model-Part 1

- After adding features, current dataset shape: (900, 101)
- Check: 1.Useless data; 2. Needs to do Padding; 3. Any missing data

```
# drop all columns that are not 1d or scalars: we already have their mathematical features
df_q1 = df_q1.drop(columns=cols_to_drop)
df_q2 = df_q2.drop(columns=cols_to_drop)
df_q3 = df_q3.drop(columns=cols_to_drop)
df_q4 = df_q4.drop(columns=cols_to_drop)
```

```
# After inspection, we will also drop xs and srs, they are only useful for featurization
df_q1 = df_q1.drop(columns=['xs', 'srs'])
df_q2 = df_q2.drop(columns=['xs', 'srs'])
df_q3 = df_q3.drop(columns=['xs', 'srs'])
df_q4 = df_q4.drop(columns=['xs', 'srs'])
```

```python
# decide whether need padding
# Example for a list of arrays (if `data` is a list or an array of arrays)
lengths_q1 = [seq.shape[0] for seq in mod_data_q1]  # Assuming sequences
                                                   # are the first dimension
lengths_q2 = [seq.shape[0] for seq in mod_data_q2]
lengths_q3 = [seq.shape[0] for seq in mod_data_q3]
lengths_q4 = [seq.shape[0] for seq in mod_data_q4]

# Check if all sequences have the same length
if len(set(lengths_q1)) > 1:
    print("Q1 Padding needed. Sequence lengths vary.")
else:
    print("Q1 No padding needed. All sequences have the same length.")
# Check if all sequences have the same length
if len(set(lengths_q2)) > 1:
    print("Q2 Padding needed. Sequence lengths vary.")
else:
    print("Q2 No padding needed. All sequences have the same length.")
# Check if all sequences have the same length
if len(set(lengths_q3)) > 1:
    print("Q3 Padding needed. Sequence lengths vary.")
else:
    print("Q3 No padding needed. All sequences have the same length.")
# Check if all sequences have the same length
if len(set(lengths_q4)) > 1:
    print("Q4 Padding needed. Sequence lengths vary.")
else:
    print("Q4 No padding needed. All sequences have the same length.")
```

```
Q1 No padding needed. All sequences have the same length.
Q2 No padding needed. All sequences have the same length.
Q3 No padding needed. All sequences have the same length.
Q4 No padding needed. All sequences have the same length.
```

Data Preparation for Model-Part 2

- We stored all data after previous steps into .npy files.
- **Needs**: Transfer loaded .npy into tensor. (will use pyTorch for model)
- Results: The prepared data X – data for training and Y – labels having shape

```
demo_x.shape: torch.Size([900, 1187])
demo_y.shape: torch.Size([900])
```

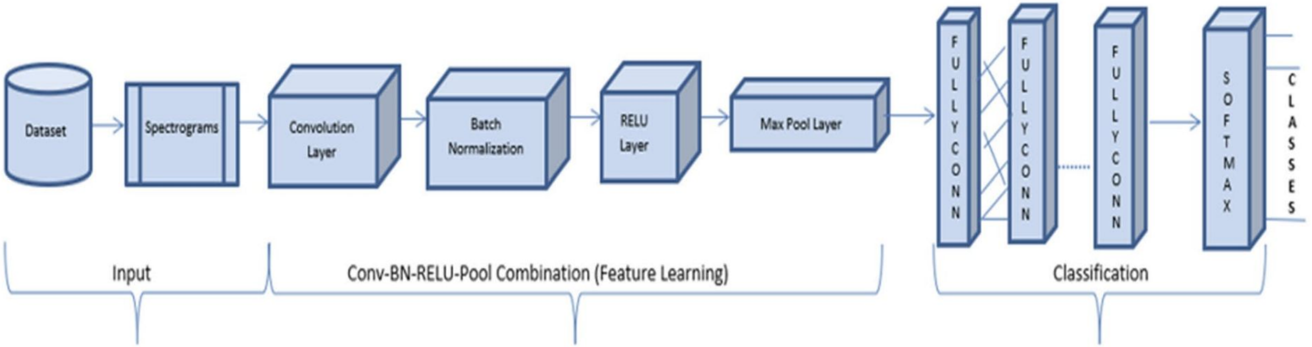
Next Step

Model Comparison

- Classical Machine learning models for music emotion recognition (classification):
    - Support Vector Machines (SVM)
    - k-Nearest Neighbors (k-NN)
    - Gaussian Mixture Models (GMM)

Model Comparison

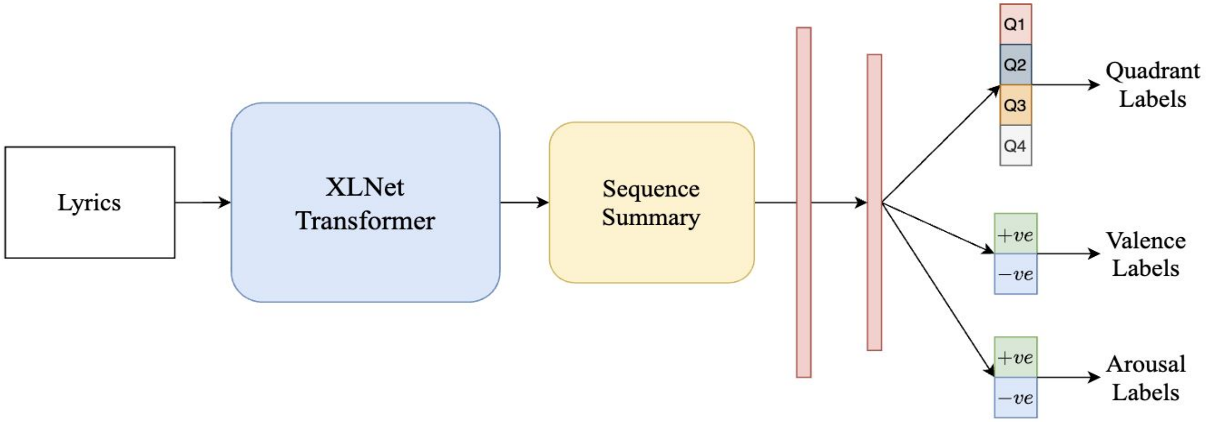
- CNN based model



https://link.springer.com/content/pdf/10.1007/s10772-020-09781-0.pdf

Model Comparison

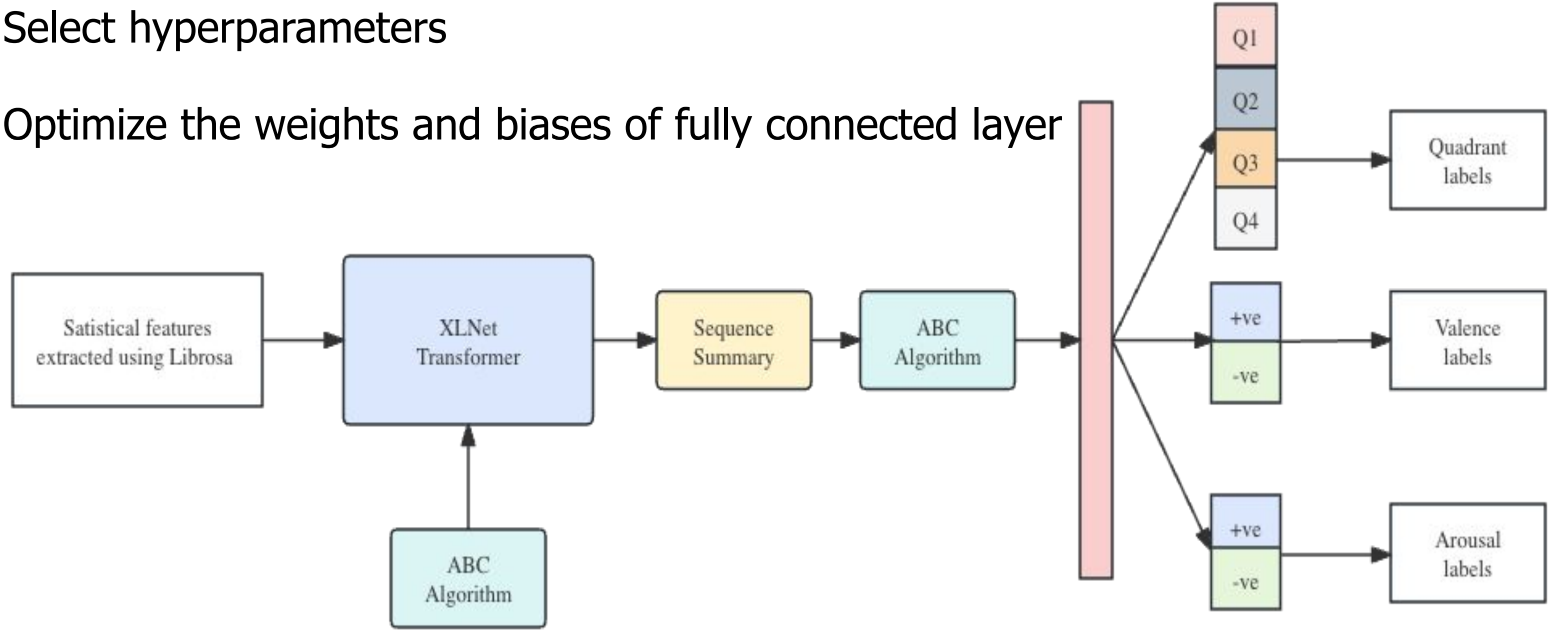
- Transformer-based approach model + XLNet architecture
  - Transformer is a good choice in mining affective connotations
  - XLNet: a large bidirectional transformer



https://arxiv.org/pdf/2101.02051.pdf

Our Model(tentative)

- Incorporate Artificial Bee Colony (ABC) algorithm
  - Select hyperparameters
  - Optimize the weights and biases of fully connected layer



Mid-Checkpoint

- We wanted to know whether we can replicate our datasets' authors' works.
- After examine features, we decide to explore our own way.

- Done with all the previous work for training a NN, have a neat dataset
- Will continue exploring multiple NN structures, compare their performance for predicting the quadrants of audio clips.

Later Possible Works

- Chunk the audios into only 15 secs or 10 secs instead of 30 secs to see whether we still got nice predictions.
- Run some classical ML models(SVM, K-NN, GMM) and possibly CNN for comparison.
- Visualize the distribution of emotion embedding (to specify the label)