Advanced Machine Learning Final Project - From Emotions To Music: AI powerd music generation based on LSTM and FIGARO

Kamil Sabbagh & Rafik Hachana & Mohammad Shahin
k.sabbagh@innopolis.university,r.hachana@innopolis.university,m.shahin@innopolis.university

1 Motivation

In today's fast-paced digital age, there's a growing appetite for personalized content, especially in the realm of music creation. Many individuals aspire to craft their own unique musical compositions, whether for personal enjoyment or commercial endeavors. The beauty of modern AI-powered tools lies in their ability to empower users regardless of their expertise in a specific field. This project aims to capitalize on that accessibility, opening doors for anyone to delve into music production and customize their creations to evoke the emotions they desire.

2 Data

In order to achieve our goal, we need to break it into smaller parts to better understand each one.

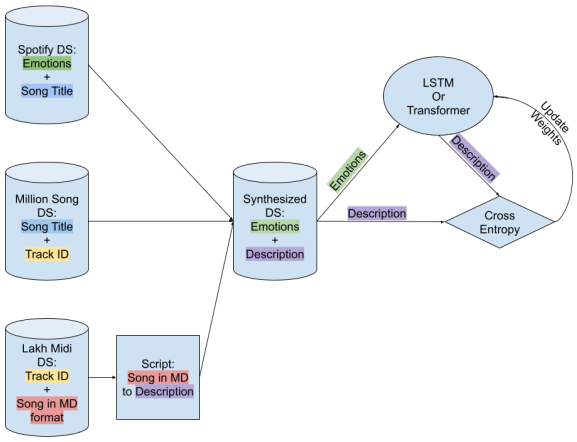

Figure 1. The Training pipeline from data to music

2.1 Emotions to Music Files to Instruments

The first step to achieve our goal is to transform the emotions (hereafter referred to as the input) into musical information which dictates how a specific music piece will be generated. To accomplish this task, intersections of several publicly available datasets were utilized, mainly:

- The Spotify Tracks dataset[1]: This dataset contains a list of songs from the Spotify streaming service. For each song, it includes its name, artist, album name, as well as emotion-related features that are used by the Spotify algorithm for recommendation.
- The LakhMIDI dataset[2]: This dataset contains the MIDI files of songs. We use the MIDI files to extract "high-level description sequences" which are the target of our trained-model, and will be the input of the FIGARO music generator. This MIDI files are named with a track ID from the Million Song dataset.
- The Million Song Dataset[3]: This dataset is used to match the previous dataset and find their intersection. It contains both an ID of the song, which corresponds to the name of a MIDI file in the LakhMIDI dataset, and contains the song title and artist which we use to match with the Spotify dataset.

By taking the intersection of the Spotify Tracks dataset and the LakhMIDI dataset, through the Million Song Dataset as a matching mediator, we obtain a dataset that maps emotion features to MIDI files, we further extract the FIGARO description from these MIDI files to obtain the dataset that we use to train our model.

2.2 Part 2: Instruments to Music

The second step of our pipeline leverages the FIGARO[1] pre-trained model. We don't have any training data here.

3 Exploratory data analysis

After taking the intersection of the Spotify Song Dataset and the LakhMIDI dataset. We obtain a tabular dataset of 10742 rows containing the following columns:

1. FIGARO description: A list of tokens representing a 12-bar segment from the corresponding song, expressed in terms of the high-level features (description) that the FIGARO model takes as input.
2. Musical features: This is the part taken from the Spotify dataset. These features will be used to generate a FIGARO description, which we will subsequently use to generate music. We are using ten predictor

[1] https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset
[2] https://colinraffel.com/projects/lmd/
[3] http://millionsongdataset.com/

features: *Energy, Valence, Loudness, Instrumentalness, Danceability, Acousticness, Key, Mode, Time Signature, Tempo*

3. Other Spotify features that won't be used for prediction but can be used for exploratory data analysis, such as the song's popularity, or its musical genre.

Once we have our data in the aforementioned format, we do some exploratoray data analysis to get feature distributions, correlations, ... etc.

3.1 Feature distribution

We analyze the distribution of both predictor and non-predictor features using histograms and Kernel Density Estimation. Our predictor musical features have varying degrees of biases, we have included examples in figures 2, 3, 4. As we can see, some features like Danceability are balanced and can be fitted with a normal distribution, others like Loudness are more skewed and have a larger tail on one side. There are some features like Time Signature, which are defined as discrete values, but are highly skewed because of tendencies in popular music. For instance most popular music tends to be written in the 4/4 time signature, which we can observe in figure 4. We have observed that our features fall into one of these categories, the results for all features can be found on our GitHub repository.

We have also explored non-predictor features. An example is the musical genre which we show in figure 5. As we can see, there is a huge bias towards disco and pop in the dataset, with genres like jazz, techno, metal, and opera being under-represented. We think that this will affect the qualitative results of the final pipeline, skewing it more to using elements from pop music rather than less popular genres.

3.2 Feature correlation

We visualize the correlation between our predictor features in figure 6. Overall, most features are not correlated at all except for a few pairs. Valence and danceability are highly correlated, which means that happier songs tend to be more danceable. Loudness and energy are highly correlated as well. Acousticness is negatively correlated with loudness and energy, which means that the more a song is "Acoustic" (using natural instruments and less electronic sounds), the less loud and energetic it tends to be. We think that these correlations are intuitive. And we prefer to keep all features for the training pipeline, which will give the user more freedom to choose and give our model more features to use for prediction.

4 Task

Our goal is to build the pipeline shown in figure 7. The pipeline has two main steps: The conversion of the emotional and music features into a description sequence of the desired music, and then the generation of music using the obtained description. We will discuss each step separately here. The first step is where we train our machine learning model from

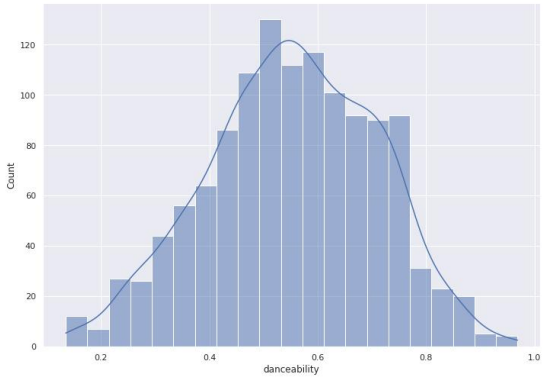

Figure 2. Distribution of the Danceability feature

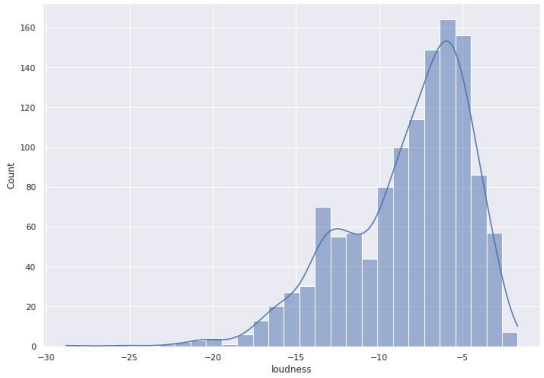

Figure 3. Distribution of the Loudness feature

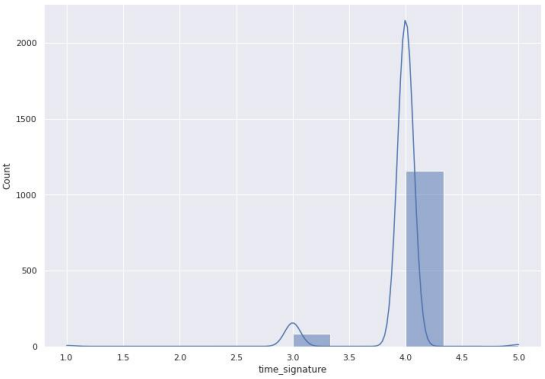

Figure 4. Distribution of the Time Signature feature

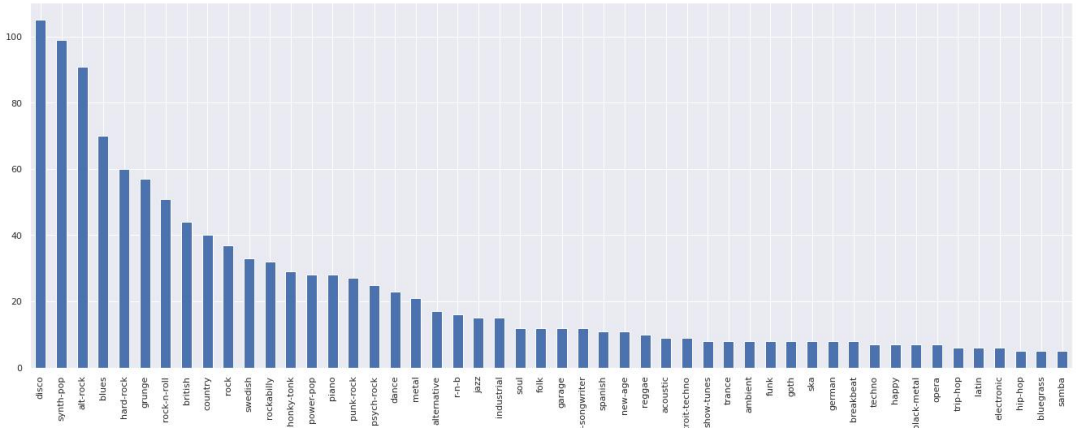

Figure 5. Distribution of song genres

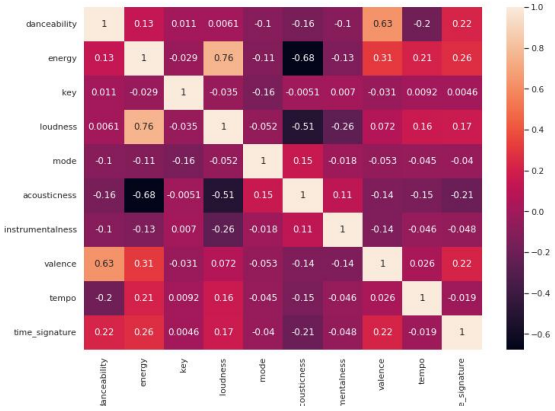

Figure 6. Correlation between the predictor features

scratch. Meanwhile, we use the pretrained FIGARO model for the second step.

4.1 Mapping emotions to music description

This is our main contribution. We want to map features like danceability and energy into a FIGARO description sequence, which makes FIGARO more accessible to end users who don't know its input vocabulary.

In order to learn this mapping, we train multiple model architectures on our collected dataset. We then compare the results and choose a model to keep for the final pipeline. The main architectures we use are Transformers and LSTMs.

4.1.1 Transformer Decoder. Our first option for the mapping is to use a Transformer decoder. We train the decoder to predict the next token for a description sequence, while providing it with the emotional features embedded as an "encoder" hidden-state.

4.1.2 LSTM model. We train an LSTM model on the next token prediction task. In this case, the LSTM takes the previous description tokens concatenated with a vector of features each. We use a 6-layer LSTM.

4.1.3 Variants of the task. We have observed that FIGARO descriptions follow a regular-grammar and can be broken down into different token classes that can be recombined programmatically. Therefore, given that our dataset is limited in size, we also attempt to train multiple LSTM models to generate a certain description token class each, or sub-descriptions. Which can be combined later to give a better result compared to the single model. With this approach, we ensure that the output description is always valid, even if the performance of the individual models is not great. The models will always output tokens that are within the vocabulary of their class of interest.

With this variant of the task, some of the token classes have an ordering to their tokens .i.e their vocabulary has a unique mapping to integers. For example, the `MeanPitch` token class has all its tokens numbered (`MeanPitch0`, `MeanPitch1`, `MeanPitch2` ...) with each number denoting a mean pitch value bin. In the case of these token classes, we can tolerate model predictions that are close to the hard label given by the dataset. Therefore we use *soft labels* to tolerate close predictions of the model. We calculate the soft labels by replacing the hard label with a Gaussian distribution centered at the hard label value.

4.1.4 Evaluation. In order to evaluate this part of the task, we track the training loss of each model, and we calculate the Accuracy score on the test dataset (10% of the whole dataset size) every 10 epochs. For the LSTMs trained to predict sub-descriptions of "ordered" token classes, we also use the KL-divergence as an evaluation metric, comparing the output

probabilities of the model and the soft labels from the target.

4.2 Mapping music description to audio music

Once we obtain a music description after the first step of our pipeline, we need to convert the high-level description into audio. For this purpose, we leverage the pre-trained FIGARO model, and we use its `figaro_expert` checkpoint which converts description sequences into MIDI files. Description sequence only have the high-level features of music (an example is shown in the pipeline in figure 7), while MIDI is a symbolic representation of music that contains all the musical information of the song, down to the individual notes, their instruments, loudness, and timing.

Therefore, after converting the description to MIDI using FIGARO, we have already obtained the desired song, and we just need to convert it to audio. This can be done simply with an audio library. In our case, we `FluidSynth`.

The FIGARO checkpoint that we use leverages a sequence-to-sequence Transformer architecture, as opposed to other variants of the model that also use a VQ-VAE with samples from its latent space in addition to the input description. For more details, you can check the demo notebook published by the authors of FIGARO[4]. We use the same code for our final pipeline.

5 Results

Here we present the training results and outcomes, and how they served our full pipeline.

5.1 Model training

In total, we train 7 sequence models, for 100 epochs each: One Transformer decoder, one LSTM that is trained to predict the the whole description sequence, and 5 LSTM models that work jointly to predict the sequence, with each one of them predicting a subsequence of a certain token class, as described in the previous section.

The final training results are shown in Table 1. In terms of overall accuracy, the Transformer performed the best with a test Accuracy of 68%. While most of the specialized LSTMs came second with their accuracy of generating sub-descriptions. Only one specialized LSTM struggled in performance, which is the Misc. LSTM, trained to predict Bar, Chords and Instrument tokens. This LSTM has the biggest vocabulary to predict from, which explains its poor performance.

We also show some examples of the evolution of loss and evaluation metrics over training in figures 8, 9, 10, 11, 12, 13. For the full results, please check our GitHub repository.

[4] https://colab.research.google.com/drive/1UAKFkbPQTfkYMq1GxXfGZOJXOXU_svo6

5.2 Full pipeline

Looking at the Accuracy score, it might seem that the Transformer is the most suited model for usage in our pipeline with FIGARO. However, after trying the combined sub-descriptions from the specialized LSTMs, we observed that they give a better result with FIGARO, even though they have a lower performance individually. This performance was judged qualitatively from the generated sound file.

We attribute the advantage of using multiple specialized LSTMs to the following fact: *The FIGARO description sequence is defined using a regular grammar*, .i.e it has a very defined structure that can be expressed as a regular expression. All the training samples of FIGARO are following this grammar. On the other hand, a Transformer trained to predict the whole sequence will output grammatically-invalid descriptions, which are very far from the distribution that FIGARO was trained on. When we switch to an ensemble of specialized LSTMs, the combined output description is guaranteed to follow the regular description grammar. The description might not be as representative of the input emotions and features, but it is closer to what FIGARO expects. This results in better quality music that is close enough to what the user desires. In practical cases, the music quality itself is non-negotiable, while variations in the emotions expressed can be tolerated.

Table 1. Final training results for all models. Only models trained on soft-labels were evaluated with the KL-divergence

Model	Accuracy	KL-Divergence
Transformer	0.68	-
LSTM full	0.05	-
LSTM Pitch	0.49	0.47
LSTM Density	0.53	0.36
LSTM Velocity	0.46	0.60
LSTM Duration	0.66	0.50
LSTM Misc.	0.07	-

6 End-user music generator

As discussed previously, the pipeline setup with multiple specialized LSTMs is more suited for practical cases. We use this setup along with FIGARO and the FluidSynth library to implement an emotion-to-music tool for end-users. The tool is user-friendly and does not require any knowledge of music theory to use it.

The tool has two parts: The frontend user interface, which is a single-page application implementation using Typescript, React, and the Chakra components library. The main page of the user interface is shown in figure 14.

The second part is the backend JSON REST API, implemented in Python with the Flask framework. The API loads all the necessary models on startup and has three endpoints:

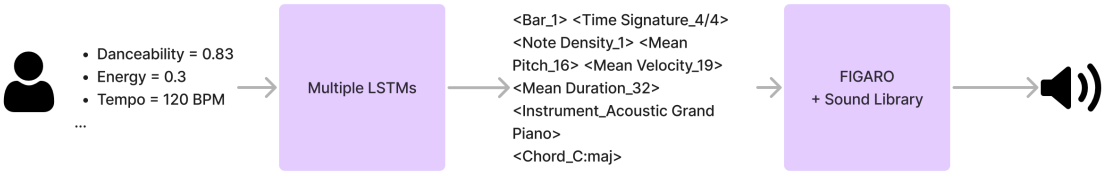

Figure 7. Final pipeline

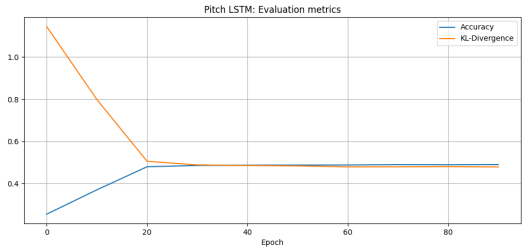

Figure 8. Performance of the Pitch LSTM over training

Figure 11. Training Loss of the Transformer

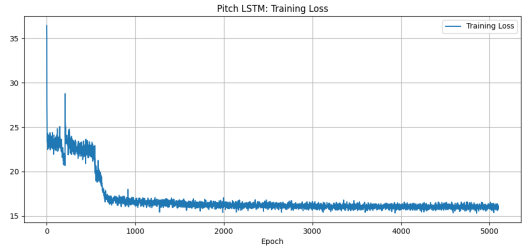

Figure 9. Training Loss of the Pitch LSTM

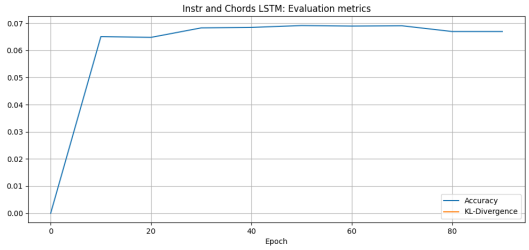

Figure 12. Performance of the Misc. LSTM over training

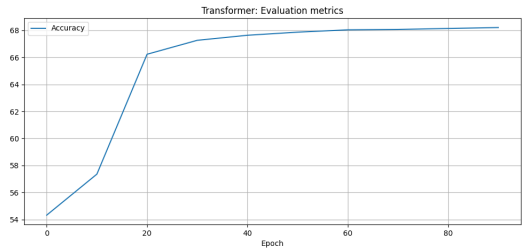

Figure 10. Performance of the Transformer over training

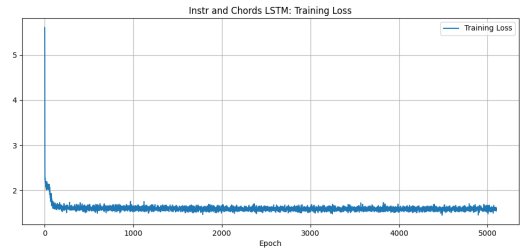

Figure 13. Training Loss of the Misc. LSTM

- `POST /submit` which accepts requests to generate music. The request should include the emotion features. This enpdoint launches the generation process in a separate thread and returns a task ID. `GET /status` which returns that status of a task given its ID (in progress or finished).

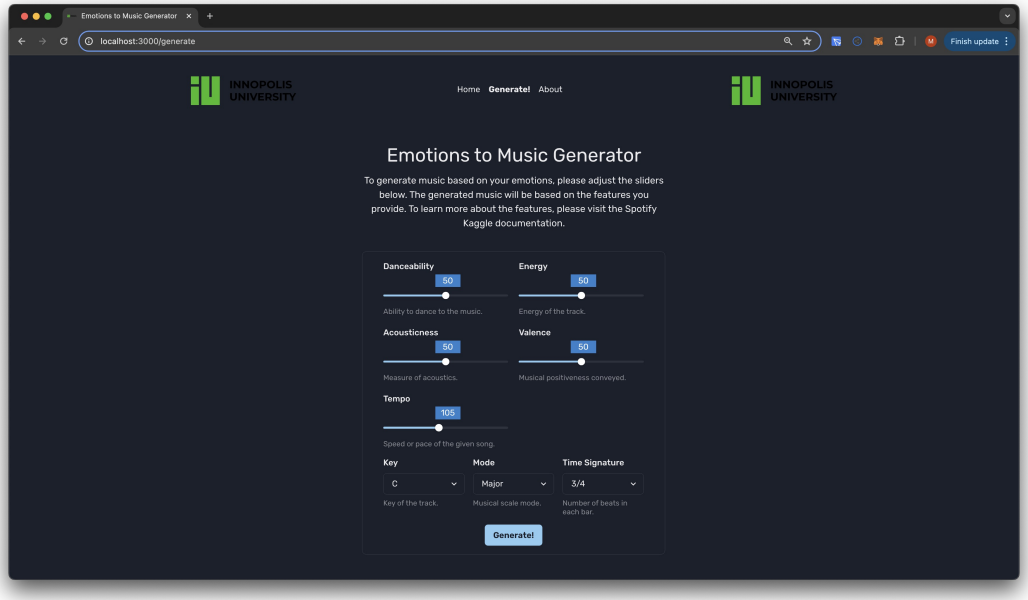

Figure 14. User Interface for our emotion-to-music tool

- `GET /result` which returns the audio file for a completed task.

7 Conclusion

In this work, we aimed to create an accessible music generation tool using multiple machine learning. We have achieved this goal by leveraging the FIGARO pretrained Transformer for music generation, and by combining it with our own trained mapping from emotional features to description sequences, the latter being the input of FIGARO. We experimented with multiple models and approaches in our learned mapping, including Transformers, LSTMs, and the usage of soft-labels. Our results show that the best qualitative results are achieved using multiple specialized LSTM models, which map features to "sub-descriptions". We then combine the sub-description to construct an input for FIGARO and generate the final result.

To fulfill our initial motivation, we made the tool accessible to end-users through a web interface.

For the exhaustive code, numerical results, audio results, and visualizations used in this project, please check out our GitHub repository at https://github.com/RafikHachana/emotion-based-music-gen.

References

[1] D. von Rütte, L. Biggio, Y. Kilcher, and T. Hofmann, "FIGARO: Generating Symbolic Music with Fine-Grained Artistic Control," Mar. 2022, arXiv:2201.10936 [cs, eess, stat]. [Online]. Available: http://arxiv.org/abs/2201.10936