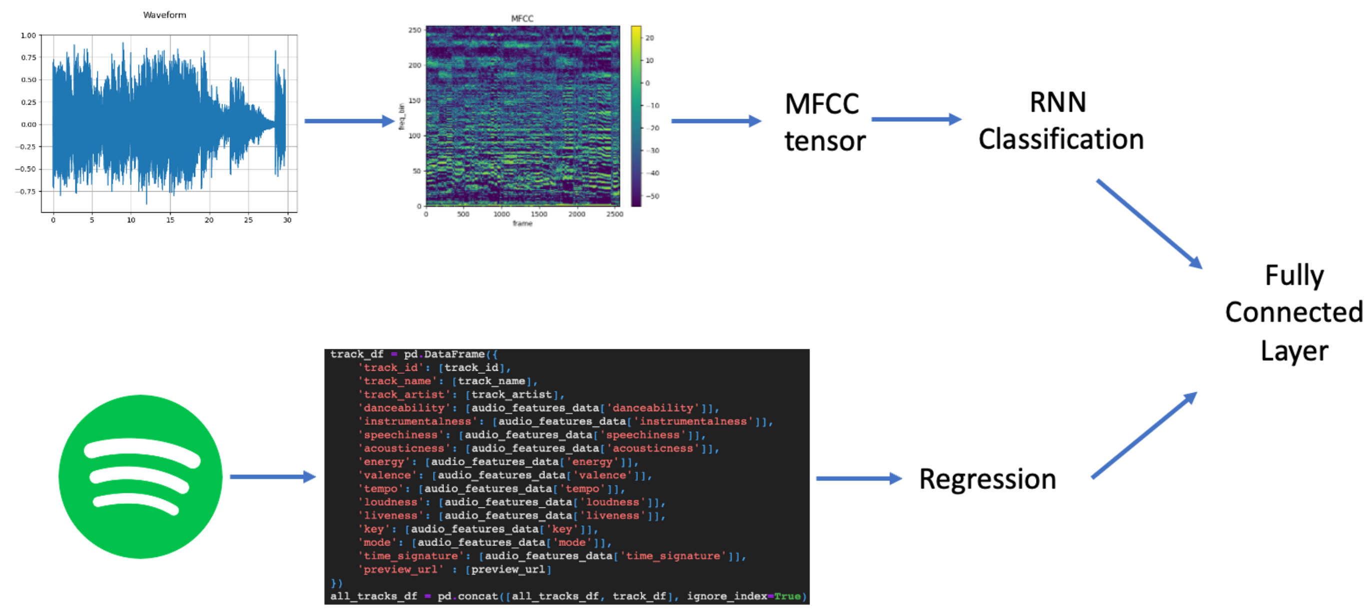
Song Feature Prediction: Final Report

Nafis Abeer, Erfan Khalaj, Risheet Nair, Bahar Rezaei
{nafis, ekhalaj, risheetn, rezaei}@bu.edu



```
track_df = pd.DataFrame({
    'track_id': [track_id],
    'track_name': [track_name],
    'track_artist': [track_artist],
    'danceability': [audio_features_data['danceability']],
    'instrumentalness': [audio_features_data['instrumentalness']],
    'speechiness': [audio_features_data['speechiness']],
    'acousticness': [audio_features_data['acousticness']],
    'energy': [audio_features_data['energy']],
    'valence': [audio_features_data['valence']],
    'tempo': [audio_features_data['tempo']],
    'loudness': [audio_features_data['loudness']],
    'liveness': [audio_features_data['liveness']],
    'key': [audio_features_data['key']],
    'mode': [audio_features_data['mode']],
    'time_signature': [audio_features_data['time_signature']],
    'preview_url' : [preview_url]
})
all_tracks_df = pd.concat([all_tracks_df, track_df], ignore_index=True)
```

Figure 1. Combined model architecture. The final linear layer(s) of the model combines the outputs of a regression-based multitask neural network with the outputs of an MFCC based multitask LSTM model.

1. Task

Our goal is to predict various song features using neural networks. The features we want to predict are the danceability, instrumentalness, acousticness, energy, and speechiness of songs. The danceability of a song is a measure of how compatible the song is with a dancing environment, like a club. This feature is very similar to the energy feature, with the difference being that the 'energy' is a direct measurement of the intensity of a track and is meant to be more predictable from the audio analysis. The instrumentalness quantifies if the track contains no vocal content. The speechiness is similar except it measures how lyrical a song is. Old-school Hip Hop songs would likely have a high value for this metric. Lastly, the acousticness assesses whether the track is composed of organic instruments or electronic music. We chose these labels to essentially replicate the feature extraction mechanisms present in many song recommender systems. Spotify already has all of these, but it is a black box. We try to discover/estimate what happens in the black box. We acquire the ground truths for these labels from Spotify [1], as the company is a leader in the music streaming industry; and therefore, their audio analysis team is trustworthy. The inputs of our neural networks would vary, with some inputs being numerical song features, while other inputs would directly analyze the audio to extract frequency information using MFCCs. We experiment with both single-task models that predict one of the labels at a time and multitask models that predict all the labels at once. Our success is determined by the neural networks' abilities to output values for the labels that closely resemble the Spotify ground truths.

2. Related Work

We build upon various existing approaches to develop models that predict our desired labels. We leverage the strengths of different feature extraction methods and model architectures to address the limitations of previous studies and achieve our goals. Something we had to consider deeply was the fact that we were limited in computational resources, which caused us to abandon some very effective methods.

Some of our MFCC model inspiration comes from Nasrullah and Zhao's work in [2], which presents a Convolutional Recurrent Neural Network (CRNN) for music artist classification. While their approach shows promise in capturing both spectral and temporal information, it suffers from the mentioned computational complexity issues. Their model requires generating spectrograms. While we initially attempted this method, we realized that the spectrograms we generated for our 30-second clips appeared much too condensed. The correct way to apply their approach would be to divide our audio clips into milliseconds-based clips and convert them to spectrograms in that state. Unfortunately, given our timeline, this would have been too ambitious. As a result, we choose to use Mel-frequency cepstral coefficients (MFCCs) as our primary input, which can capture key and timbral information in a more computationally efficient manner.

To combine different types of features as inputs, we draw inspiration from Zangerle's work [3]. They use a wide and deep neural network architecture to jointly exploit low- and high-level audio features for hit song prediction. While their objective differs from ours, their method of combining different architectures using linear layers is relevant to our approach to combining numerical and MFCC-based models. We essentially just used different input model architectures but drew from their idea that combining these models may yield success.

Indorf's article [5] provides a practical example of using Spotify data to extract audio from preview URLs and perform feature extraction. Their work demonstrates the feasibility of using neural networks to predict song popularity based on Spotify labels. We follow a similar approach in our work, focusing on predicting other song features using the provided labels from Spotify. He essentially helped pioneer our data pipeline.

In our updated report, we mentioned that we could use a beat-aligned method presented by Bertin-Mahieux et al. [8]. This paper suggested a complex preprocessing pipeline and tempo estimation from mini audio clips, and we instead opted to work with the entire 30-second clips. This decision simplifies our preprocessing steps while still allowing us to capture relevant information from the audio data. We were still able to fulfill our hopes of capturing temporal data in the songs using 20 MFCCs every one of the 160 frames in an audio clip.

We base our MFCC input and model architecture on Gessle and Åkesson's work [7]. They compare CNN and LSTM models for music genre classification using MFCCs. We adopt their approach for our MFCC-based multitask LSTM model, with the notable difference being our use of higher-quality audio and a focus on predicting song features rather than genre classification.

Overall, our approach combines the strengths of various existing methods while addressing our own limitations in being able to implement their methods. We develop a few comprehensive models for predicting song features based on audio data. By integrating different types of features and models, we aim to create a robust and accurate predictor for a wide range of song characteristics.

3. Approach

We first collected data from Spotify using the Spotipy library [4]. We had one of our team members create a Spotify developers account and use the Spotify API to grab all the songs in their library, along with the desired labels for those songs. We then had one team member work on collecting all the audio of these songs using the collected preview URLs. This was done using the Python requests library.

We initially converted the audio into spectrograms using Pytorch [6] before realizing that the images were far too compressed to be useful. Another team member created a data loader to associate the spectrograms with desired labels. Lastly, we had a team member build an initial simple CNN model that trained on predicting the danceability of the songs based on the spectrogram images. This model was to have a threshold and if the output were above the threshold, the prediction would be “danceable” and below would mean “non-danceable”. This was our initial binary classification approach that turned out to be a bit naive as the spectrograms were not capturing any useful information about the songs.

Our next step was to ensure that we had sufficient data and were preprocessing the data to extract useful information from songs. The preprocessing steps will be described in more detail in section 4.

One team member converted audio clips into MFCC tensors and used them to build various models for predicting the desired features. Initially, they predicted only one feature (danceability), but later adapted a model to predict all five labels simultaneously. The models were all built using the Pytorch built-in neural network library. The architectures of these models were kept simple as we cared more about the outcomes of using a CNN vs LSTM to confirm the findings from [7], at first. Interestingly, as seen in our results section, our LSTM performed better than CNN. We built three models using MFCC tensors, namely:

1. A simple 2D CNN model.
2. A simple 1D LSTM model.
3. A multitask 1D LSTM model.

Another team member worked on correlating high-level numerical features of songs with the desired labels. They trained neural networks to output values based on specific input feature combinations. They also used the Pytorch built-in neural network library and their architecture only used linear layers with Relu activations since they were essentially solving a regression problem.

A third team member focused on building a more complex CNN model, aiming to improve upon the existing multitask model that predicted all five labels at once. They introduced several more convolutional layers, some dropouts because we had MFCCs for many of the frames and the information could have been a bit overwhelming. They also introduced average pooling into this model for similar reasons.

Finally, one team member combined the most successful MFCC-based model with the numerical-based model by concatenating the outputs of the two pretrained models and adding intermediate fully connected layers. They then performed hyperparameter tuning on this combined architecture to optimize its performance. The hyperparameter tuning decisions are described in section 6.

4. Datasets

To obtain the dataset of songs with various features (danceability, energy, instrumentality, acousticness, etc.), we queried Spotify using the Spotipy library. We initially used a mini batch of 700 songs to create the spectrogram images from the 30 second preview URLs that we extracted from the initial query. We collected more data by querying Spotify using the Spotipy library for eight different genres. Their goal was to gather a total of 20,000 tracks, distributing them evenly across genres. For each genre, a dataframe containing relevant tracks was compiled and then combined into a single dataframe, which was saved to a CSV file for further processing.

Mel-Frequency Cepstral Coefficients (MFCC) were used for feature extraction from the audio signals. MFCC is a popular technique for audio processing, particularly in speech and music applications. It represents the spectral shape of an audio signal in a compact and informative manner, which can be used for tasks like speech recognition, speaker identification, or music genre classification.

The MFCC tensors obtained had a shape of [21325, 20, 160]. The first argument represents the batch size, the second argument is the number of MFCC coefficients, and the last argument represents the number of frames in the MFCC representation of the image. Given the hop_length of 8192, and clip length of 30 seconds sampled at 44100 Hz, the team expected about (30 * 44100)/8192 frames. The edge effects of windowing and padding possibly contributed to this number being slightly lower than the actual value. To keep the frame count low and avoid excessive data in the transform tensor, we carefully selected the parameters for the MFCC transform, such as the number of FFT points, window length, hop length, and the number of Mel bands and MFCCs.

We created a custom dataset class called AudioDataset to handle the audio data and its associated labels. This class inherits from the torch.utils.data.Dataset class, which is part of the PyTorch library. It takes an MFCC tensor and a dataframe as inputs during initialization. The class calculates the mean and standard deviation of the MFCC tensor and standardizes it by subtracting the mean and dividing by the standard deviation. This preprocessing step ensures that the MFCC features have zero mean and unit variance.

The labels are extracted from the dataframe, and they include danceability, instrumentalness, acousticness, energy, and speechiness. These labels are then converted to a PyTorch tensor of type torch.float32. The method returns a tuple containing the standardized MFCC tensor and the corresponding label tensor. This is just one example of the kind of dataset classes we were able to create given the CSV file containing our labels and the MFCC tensor file we compiled using the described methods.

5. Evaluation Metrics

The metrics we used to evaluate our models are as follows:

Mean Squared Error (MSE): a commonly used measure of the average squared difference between the predicted values and the actual values in a regression problem. However, MSE can also be sensitive to outliers which can cause issues especially with datasets that use the type of precision that was used in our data. The lower the MSE, the better the model. In our case, since the labels fell between an interval of 0 and 1, squaring the differences between the actual values and the predicted values will actually lower the MSE giving a false sense that our models performed well.

Mean Absolute Error (MAE): another commonly used measure of the average difference between the predicted values and the actual values in a regression problem. Unlike MSE, this metric does not square the difference but rather takes the absolute difference. We went with MAE for our final metric because it is much more robust to outliers, and it does not give a false sense that our model is performing well. An average MAE of 0.110, would indicate that if an actual value is 0.234, our model would predict somewhere between 0.124 and 0.344 on average, which is still low relative to the entire range of possible output values.

Root Mean Squared Error (RMSE): this metric is extremely similar to the MSE but the square root is taken to convert the units back to their original scale. This makes the results extremely easy to interpret. Additionally, it penalizes larger errors because the errors are squared before averaging and then the square root is taken. In this case, the RMSE was a better metric for our purposes than the MSE was.

R-squared: a statistical measure that represents the proportion of the variance in the dependent variable that is explained by the independent variables in a regression model. It is a measure of how well the regression model fits the observed data. R-squared does not provide any information about the goodness of individual predictions, only about the overall fit of the model. Since we did care about the accuracy of individual predictions, we did not put too much stake on this metric, but it was useful to see whether our model was fitting the observed data well.

Pearson's Correlation Coefficient: measures the linear relationship between two variables, with values ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation). We wanted to see whether a change in certain ground truth values had a direct linear effect on features we were trying to predict. Thus Pearson's correlation coefficient was a valid metric for us to use. This allowed us to see whether our linear model was effectively finding a linear association between the numerical aspects of the songs and the song features.

6. Results

1D RNN Model: This simple 1D model uses an LSTM layer with input size of 20 and hidden layer size of 64. It then has a linear layer of size 64. This particular model was only tested on the danceability feature and yielded an MAE of 0.079 and a Pearson correlation coefficient of 0.830 as can be seen from the results in the tables. This seemed to perform better than the 1D CNN model which predicted values for the same label, which is contrary to the findings in [7]

2D CNN Model: This model used two 2D convolutional layers followed by two linear layers. The Relu activation function was applied to both convolutional layers and then we applied max pooling to the result of each layer. The first linear layer also used a Relu activation function and then finally, after the second linear layer was applied, we used the sigmoid activation function. Once again, this model was only tested on the danceability feature which yielded an MAE of 0.080 and a Pearson correlation coefficient of 0.811 as can be seen in the following tables.

Multi-task RNN Model: This model has a very similar architecture as the 1D RNN model except the purpose of this model is to predict all 5 desired features simultaneously. One difference is that the sigmoid activation function is applied after the LSTM and linear layers. As can be seen from the tables, the danceability feature was predicted with an MAE of 0.111 and a Pearson correlation coefficient of 0.615.

Linear Combiner: Since the linear combination model was our final architecture, the hyperparameter tuning was crucial. The following are the different settings we worked with:

a. Removing the final fully connected (fc) layer from the pre-trained LSTM and numerical model, but keeping the previous pretrained layers to preserve the features learned in the networks and provide more flexibility in combining them. Final linear layers contained two 32 size hidden layers and one 64 size hidden layer, and were trained using backpropagation.
Here we got an average MSE of 0.042 and average MAE of 0.1455 across our 5 predicted variables.

b. Keeping the final fc layer in the LSTM and numerical model, and adding three extra fully connected layers to combine (concatenate) the outputs. These three extra layers were treated as parameters and trained via backpropagation. The size of the layers was changed during hyperparameter tuning. The sigmoid activation function in the LSTM model was removed. Final linear layers contained two 16 size and one 32 size hidden layer. Here we got an average MSE of 0.043 and average MAE of 0.1472.

c. Keeping all layers from the pretrained models, and adding three extra fully connected layers to combine (concatenate) the outputs. These three extra layers were treated as parameters and trained via backpropagation. The size of the layers was changed during hyperparameter tuning. Final linear layers contained two size 5 and one size 10 hidden layer. Here we got an average MSE of 0.044 and an average MAE of 0.1497.

d. Changing learning rate: adaptively changed the learning rate when training loss plateaued, starting from 0.01 to 0.005, 0.002, 0.001. As for the architecture, the final fc layer in the LSTM and regression model was removed, keeping the previous pretrained layers to preserve the features learned in the networks and provide more flexibility in combining them. Here we got an average MSE of 0.041 and an average MAE of 0.1413.

As seen in the table, the best performing combined model was the one in (d), where we removed the final fully connected layers in the LSTM and used an adaptive learning rate.

Simple model: Our first test attempt at using a simple CNN to predict danceability. We used it to predict danceability, then measured its accuracy using a binary classification metric (i.e classify song as danceable if danceability > 0.5). We got an accuracy of 83% however it became clear that the trained network was outputting one number for every class. We are not sure why this could be the case, but it may be due to poor treatment of our dataset (for instance extra white spaces in the image).

Multi-task Regression Model: This model initially had 3 linear layers of which the first two used the Relu activation function and the last layer had a linear activation. The model was compiled with the MSE loss and the Adam optimizer. It was trained for 100 epochs with a batch size of 32. This initial regression model yielded an average MSE of 0.031 and an average MAE of 0.116 which was already a very good performance. However, there were some ways we thought we could try to improve the model. The first change that was made was an extra linear layer in the architecture. All other hyperparameters were kept the same. This yielded an identical average performance as the previous model. So, in the next attempt, the extra linear layer was removed and instead, the Relu activation for the first 2 layers were changed to sigmoid. The 3rd layer remained a linear activation. All other hyperparameters were kept the same. Our results in the following tables show that this model did in fact do marginally better than the initial model. The average MSE was 0.030 and the average MAE was 0.114. The danceability MAE was 0.103 and the danceability Pearson correlation coefficient was 0.699.

Summary: Based on the danceability feature alone(since these two models only tested danceability rather than all five), the simple 1D RNN model performed better than the 2D CNN model based on the MAE and Pearson correlation coefficient metrics.

For the multi-task models, it makes more sense to analyze the average MAE and Pearson correlation coefficients as all the features were predicted. The multi-task RNN model when compared to the multi-task regression model performed better based on the same metrics as above.

Table 1:

Model	Danceability (MSE/MAE)	Energy (MSE/MAE)	Speechiness (MSE/MAE)
Multi-task Regression	0.017/0.103	0.045/0.159	0.070/0.166
Multi-task 1D RNN	0.019/0.111	0.018/0.107	0.007/0.054
Complex CNN	X	X	X
Simple 2D CNN	0.001/0.080	X	X
Simple 1D RNN	0.010/0.079	X	X
Linear Combiner	0.022/0.117	0.030/0.134	0.008/0.058

Table 2:

Model	Acousticness (MSE/MAE)	Instrumentalness (MSE/MAE)	Average (MSE/MAE)
Multi-task Regression	0.008/0.053	0.013/0.089	0.030/0.114
Multi-task 1D RNN	0.024/0.113	0.052/0.141	0.024/0.105
Complex CNN	X	X	0.026/0.111
Simple 2D CNN	X	X	X
Simple 1D RNN	X	X	X
Linear Combiner	0.066/0.196	0.082/0.200	0.042/0.141

Table 3:

Model	Danceability (Pearson)	Energy (Pearson)	Speechiness (Pearson)
Multi-task Regression	0.699	0.801	0.644
Multi-task 1D RNN	0.615	0.876	0.363
Complex CNN	X	X	X
Simple 2D CNN	0.811	X	X
Simple 1D RNN	0.830	X	X
Linear Combiner	0.549	0.771	0.311

Table 4:

Model	Acousticness (Pearson)	Instrumentalness (Pearson)	Average (Pearson)
Multi-task Regression	0.321	0.900	0.673
Multi-task 1D RNN	0.901	0.772	0.705
Complex CNN	X	X	X
Simple 2D CNN	X	X	0.811
Simple 1D RNN	X	X	0.830
Linear Combiner	0.694	0.574	0.580

7. Conclusion

We realized how simple it could be to collect song audio and features from Spotify. Their API made this entire project possible to complete without having any prior datasets available for the task. We also noticed that simpler models seem to perform better than more complex models as the multitask LSTM model performed better than the combined model despite having less input information. This could also be attributed to the fact that none of the models are providing perfect results, so adding more complexities just stacks the imperfections. Overall we saw all the models we trained, beside the simple model, perform better than before training, as evidenced by the loss functions. This indicates that our suggested methods of learning song features from both numerical features and MFCCs using LSTM, CNNs, combined architectures, and linear regression models are all valid.

As part of future work, we could try using pre-trained neural networks such as Spleeter to first separate the audio into different tracks, and then input those tracks (as spectrograms) into a neural network. This would potentially lower the complexity of our inputs and help with feature learning.

References

[1] *Web API reference: Spotify for developers*. Spotify Song Audio Features. (n.d.). Retrieved February 27, 2023, from https://developer.spotify.com/documentation/web-api/reference/get-audio-features

[2] Nasrullah, Z., & Zhao, Y. (2019, March 14). *Music artist classification with Convolutional Recurrent Neural Networks*. arXiv.org. Retrieved February 27, 2023, from https://arxiv.org/abs/1901.04555

[3] Zangerle, E. (2021, August 20). *Hit song prediction: Leveraging low- and high-level audio features*.

evazangerle.at. Retrieved February 27, 2023, from https://evazangerle.at/publication/zangerle-ismir-2019

[4] *Welcome to spotipy!*¶. Welcome to Spotipy! - spotipy 2.0 documentation. (n.d.). Retrieved February 27, 2023, from https://spotipy.readthedocs.io/en/2.13.0/#

[5] Indorf, N. (2022, January 28). *Using deep learning to predict hip-hop popularity on Spotify*. Medium. Retrieved February 27, 2023, from https://towardsdatascience.com/using-deep-learning-to-predict-hip-hop-popularity-on-spotify-1125dc734ac2

[6] Audio feature extractions¶. Audio Feature Extractions - Torchaudio 2.0.1 documentation. (n.d.). Retrieved April 6, 2023, from https://pytorch.org/audio/stable/tutorials/audio_feature_extractions_tutorial.html

[7] Gessle, Gabriel, and Simon Åkesson. "A Comparative Analysis of CNN and LSTM for Music Genre Classification." Digitala Vetenskapliga Arkivet https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1354738&amp;dswid=1183

[8] Bertin-Mahieux, Thierry; Weiss, Ron J.; Ellis, Daniel P. W. Clustering beat-chroma patterns in a large music database, ISMIR 2010: Proceedings of the 11th International Society for Music Information Retrieval Conference, August 9-13, 2010, Utrecht, Netherlands

Appendix A. Detailed Roles

Table 1. Team member contributions

Name	Task	File names	No. Lines of Code
Nafis	● Project management (task division, planning etc.) ● Spotify Data retrieval ○ Initial small datasets for model testing (csv files in Sample_data directory) ○ Large dataset for improved models (formation of /Data/track_features.csv using Spotipy algorithms to collect 20000+ songs with desired features) ○ Spectrogram tests ● MFCC tensor from audio research and conversion (formed [21325, 20, 160] tensor to be used as input for all MFCC based models) ● Created AudioDataset classes for various MFCC based models ● MFCC based 2D CNN model to predict danceability ● MFCC based 1D RNN (LSTM actually) model to predict danceability ● MFCC based 1D RNN (LSTM) multitask model to simultaneously predict all 5 desired features ● Evaluation metrics for all models mentioned above ● Handled presentation planning and presenting with support from team ● Handled bulks of the reports (sections 1, 2, 3, 4 of previous and 1, 2, 3, 4, 7 & appendixes of final) due to understanding project overview and suggesting project goals. Completed all of the project proposal.	mfcc_model.ipynb Sample_Data/spotipy-test.ipynb Data/spotipy_labels.ipynb Data/spectrograms_tests.ipynb load_models.ipynb	9247
Erfan	● Architecture support ● Initial binary classification model	Linear_models_combine.ipynb	1808

	• Detailed spectrograms of smaller time steps for more accurate representation of Audio in image form (visuals) • 2D CNN model to predict Valence • Linear model that combines outputs from Regression based model and MFCC based multi task LSTM model • Created AudioDatasets for models mentioned above • Performed hyper parameter tuning on Linear model • Evaluation metrics for all models for hyperparameter tuned linear models as well as Valence model • Came up with steps of doing a temporal model in the project update report.	simple_model.ipynb valence_mfcc_model.ipynb	
Risheet	• Data collection support • Initially gathered spectrograms for all audio files given preview URLs • Performed multiple audio gathering runs to take preview URLs and provide links to Google Drive folders containing 30 second audio clips • Created numerical based regression model that takes multiple audio features as input to output our desired 5 labels – multi task model • Created tracks dataset to process numerical inputs • Provided evaluation metrics for all predicted labels • Created evaluation table for presentation as well as final report • Completed Evaluation metrics section of reports as well as results section of final report	numerical_model.ipynb Data/spectrogram_saving.ipynb	2544
Bahar	• Outlook support • Ensured that project goals were comprehensive • Improved cohesiveness of all reports by questioning laid out plans and project goals • Initially came up with base data loaders to feed into simple model • Performed spectrogram formation tests as well as MFCC tensor formation tests • Created AudioDataset for complex CNN multitask model • Designed complex CNN multitask model to predict all labels simultaneously • Provided evaluation metrics for average performance of complex CNN model on all labels • Proofreading and cohesiveness checks on all reports • Diagrams for class presentation	Complex_cnn_mfcc.ipynb Sample_Data/dataloader Baseline cnn Trying transfer learning by experimenting with ResNet18	

Appendix B. Code repository

GITHUB: https://github.com/Naaafis/Song_Feature_Pred