Multi-modal Analysis of Audio, Artist, and Lyrical Features to Predict Music Popularity

Jennifer Chen, Sophia Dai, Leslie Kim, Rose Zhao

May 3, 2025

1 Introduction

The music industry has experienced significant growth due to technological globalization, with individuals spending an average of 20.7 hours per week listening to music on streaming platforms such as Spotify, Apple Music, and Soundcloud. This growth has led to increased competition among emerging artists, creating a challenging landscape where the ratings of a new song can significantly impact an artist's career trajectory. As students with peers aspiring to be musicians and understanding the challenges of entering the music industry, we identified the need for better knowledge of what characteristics contribute to a song's popularity in specific contexts.

In this project, we developed a multi-modal model that integrates diverse data inputs such as audio features, metadata, and lyrics to predict a song's popularity score. Our goal was to create a tool that would allow artists to understand their song performance and career trajectory better, providing them with insights to better plan for the future. Through this feature analysis, we aimed to identify which aspects contribute most significantly to a successful song, giving artists actionable information to guide their creative decisions.

The primary challenges we faced included data collection and integration, handling multilingual content, and developing effective neural network architectures that could process both numerical and textual data. We approached these challenges through careful data preparation, feature selection using ridge regression, and the implementation of both fully connected neural networks for audio features and unsupervised topic modeling (LDA) and supervised sentiment classification models for lyrical data.

1.1 Instruction On Running Colabs Files:

To run our code, follow the steps below. However, to see the model results and comparisons, it is only necessary to run the third Google Colab notebook, which is explained in step 3.

1. The translated data can be spotifydata_translated_combined.csv
2. Run the Google Colab notebooks in the order that is marked within the filename. In the first notebook, there are instructions on when to run the second notebook, Sentiment Analysis with Pytorch. The only dataset that could not be included in our

zip due to size is the Sentiment140, which will need to be separately gathered from Kaggle.

3. Finally, run the third Google Colab notebook, which uses the FINAL_CLEAN_SPOTIFY_DATA.csv. This dataset has been cleaned and made numeric so that it can be readily fed into training our models.

2 Data

2.1 Dataset Selection

After evaluating several options, we selected the "Spotify HUGE database - daily charts over 3 years" from Kaggle which contains data on the top 200 songs across 35 countries from 2017-2020. We chose this dataset over our initially planned dataset (which contained a hard limit of 1000 songs per genre) to ensure a more representative sample of music industry trends.

The dataset includes comprehensive features: (1) metadata (country, URI, title, album/single, release date, track number), (2) artist information (followers, explicitness), (3) audio characteristics (danceability, energy, key, loudness, mode, speechiness, acoustics, instrumentalness, liveliness, valence, tempo, duration), and (4) additional metadata (time signature, genre, days since release, language, LDA topic, and more).

Importantly, the dataset includes a custom popularity score calculated by assigning values from 1-200 to songs based on their chart position (200 for #1, 1 for #200), with scores aggregated over the three-year period per country. This approach provides a more robust popularity metric than Spotify's time-biased internal scoring.

2.2 Data Preprocessing and Enrichment

Our preprocessing pipeline involved multiple steps:

- **Cleaning and Standardization**: We removed duplicate information across columns, encoded categorical variables (e.g., converting country names to integers), and standardized column formats.
- **Lyric Collection**: Since the dataset lacked lyrics, we scraped them using the Genius.com API. This was challenging due to the volume (60k+ unique songs) and multilingual nature of the content. We retained only songs for which we could successfully retrieve lyrics.
- **Translation Handling**: For non-English lyrics, we implemented translation processes to enable consistent sentiment analysis across all songs, navigating API rate limits and character constraints.

After preprocessing, our final dataset contained 121,064 entries representing 41,729 unique songs across 26 regions. This represents approximately 72% of the initial dataset, with removal of entries primarily due to missing values or unavailable lyrics.

3 Methodology

Our methodology involved a multi-stage approach to developing models that could effectively predict song popularity based on a multitude of features:

3.1 Exploratory Data Analysis

We began with exploratory data analysis to understand the distribution of the data and the relationships between variables. We generated correlation matrices to identify which features were most strongly associated with popularity scores and with each other. This exploration helped guide our feature selection and model development process.

	Popularity	Album/Single	Artist_followers	Explicit	danceability	energy	key	loudness	mode	speechiness	acoustics	instrumentalness	liveliness	valence	tempo	duration_ms	time_signature	Genre	Released_after_2017	senti_score	Sentiment_Range	dominant_topic
Popularity	1.000000	-0.121457	0.106889	-0.007428	0.102841	-0.011607	0.006734	0.064748	-0.026709	0.020192	0.030660	-0.075453	-0.026932	0.031299	-0.013819	-0.023793	0.023289	-0.016707	0.063201	0.000922	-0.002734	0.054593
Album/Single	-0.121457	1.000000	0.193889	0.111718	-0.049873	-0.096464	-0.014494	-0.150729	0.022759	0.030156	0.062251	0.022157	0.021895	-0.042876	0.015403	0.098585	-0.018704	-0.078276	-0.176305	-0.024645	-0.016643	-0.019299
Artist_followers	0.106889	0.193889	1.000000	-0.003598	-0.065415	-0.057036	-0.025592	0.073284	0.050126	-0.062375	-0.061734	0.002524	0.022665	-0.100504	-0.004823	0.167775	-0.020720	0.060523	-0.017323	-0.067513	-0.076855	-0.116079
Explicit	-0.007428	0.111718	-0.003598	1.000000	0.292213	-0.053518	-0.007917	-0.062486	-0.066848	0.447136	-0.006055	-0.111421	0.044217	-0.041501	0.074228	-0.095397	0.059347	-0.310082	0.211913	-0.113832	-0.064764	0.191528
danceability	0.102841	-0.049873	-0.065415	0.292213	1.000000	-0.004757	0.007505	0.052726	-0.087292	0.306994	-0.086035	-0.034547	-0.096960	0.310553	-0.000657	-0.197744	0.123509	-0.162326	0.168419	0.042535	0.025849	0.226298
energy	-0.011607	-0.096464	-0.057036	-0.053518	-0.004757	1.000000	0.016763	0.687221	-0.045186	0.077534	-0.422831	-0.048191	0.086028	0.346810	0.075567	0.022943	0.103882	0.094719	-0.026236	0.050559	0.031974	0.066465
key	0.006734	-0.014494	-0.025592	-0.007917	0.007505	0.016763	1.000000	-0.022637	-0.168208	0.035972	0.006159	-0.006156	-0.003992	0.043728	-0.005560	-0.010126	0.000825	-0.002705	0.002684	0.005490	-0.003637	0.009827
loudness	0.064748	-0.150729	0.073284	-0.062486	0.052726	0.687221	-0.022637	1.000000	-0.017202	-0.015564	-0.323322	-0.190040	0.037139	0.263914	0.044765	-0.001520	0.083035	0.120229	0.039562	0.036449	0.021151	0.044245
mode	-0.026709	0.022759	0.050126	-0.066848	-0.087292	-0.045186	-0.168208	-0.017202	1.000000	-0.105549	-0.009496	-0.007793	-0.008288	-0.047588	0.006760	0.051542	-0.032948	0.036985	-0.083681	0.007206	-0.005047	-0.078978
speechiness	0.020192	0.030156	-0.062375	0.447136	0.306994	0.077534	0.035972	-0.015564	-0.105549	1.000000	-0.013197	-0.172779	0.035760	0.112224	0.096188	-0.167204	0.118798	-0.283160	0.224378	-0.040370	-0.040001	0.237418
acoustics	0.030660	0.062251	-0.061734	-0.006055	-0.086035	-0.422831	0.006159	-0.323322	-0.009496	-0.013197	1.000000	-0.058640	-0.051824	-0.052518	-0.100150	-0.032654	-0.093115	-0.040114	-0.000755	-0.035253	-0.024854	0.001072
instrumentalness	-0.075453	0.022157	0.002524	-0.111421	-0.034547	-0.048191	-0.006156	-0.190040	-0.007793	-0.172779	-0.058640	1.000000	-0.027863	-0.132869	0.008350	0.074376	-0.027746	0.087652	-0.044559	0.051356	0.046137	-0.050746
liveliness	-0.026932	0.021895	0.022665	0.044217	-0.096960	0.086028	-0.003992	0.037139	-0.008288	0.035760	-0.051824	-0.027863	1.000000	-0.022368	0.008637	0.004643	-0.009341	-0.034176	0.006754	-0.018668	-0.004312	0.006381
valence	0.031299	-0.042876	-0.100504	-0.041501	0.310553	0.346810	0.043728	0.263914	-0.047588	0.112224	-0.052518	-0.132869	-0.022368	1.000000	0.021338	-0.141531	0.086117	0.040717	-0.074242	0.109033	0.083376	0.087706
tempo	-0.013819	0.015403	-0.004823	0.074228	-0.000657	0.075567	-0.005560	0.044765	0.006760	0.096188	-0.100150	0.008350	0.008637	0.021338	1.000000	0.002102	-0.004610	0.021852	0.025817	0.008669	0.024135	0.004472
duration_ms	-0.023793	0.098585	0.167775	-0.095397	-0.197744	0.022943	-0.010126	-0.001520	0.051542	-0.167204	-0.032654	0.074376	0.004643	-0.141531	0.002102	1.000000	-0.036312	0.045739	-0.172240	-0.057688	-0.064034	-0.084300
time_signature	0.023289	-0.018704	-0.020720	0.059347	0.123509	0.103882	0.000825	0.083035	-0.032948	0.118798	-0.093115	-0.027746	-0.009341	0.086117	-0.004610	-0.036312	1.000000	-0.026646	0.041812	-0.005234	-0.003786	0.052960
Genre	-0.016707	-0.078276	0.060523	-0.310082	-0.162326	0.094719	-0.002705	0.120229	0.036985	-0.283160	-0.040114	0.087652	-0.034176	0.040717	0.021852	0.045739	-0.026646	1.000000	-0.127170	0.038847	0.025529	-0.145169
Released_after_2017	0.063201	-0.176305	-0.017323	0.211913	0.168419	-0.026236	0.002684	0.039562	-0.083681	0.224378	-0.000755	-0.044559	0.006754	-0.074242	0.025817	-0.172240	0.041812	-0.127170	1.000000	-0.074837	-0.048619	0.147880
senti_score	0.000922	-0.024645	-0.067513	-0.113832	0.042535	0.050559	0.005490	0.036449	0.007206	-0.040370	-0.035253	0.051356	-0.018668	0.109033	0.008669	-0.057688	-0.005234	0.038847	-0.074837	1.000000	0.656555	0.030963
Sentiment_Range	-0.002734	-0.016643	-0.076855	-0.064764	0.025849	0.031974	-0.003637	0.021151	-0.005047	-0.040001	-0.024854	0.046137	-0.004312	0.083376	0.024135	-0.064034	-0.003786	0.025529	-0.048619	0.656555	1.000000	0.029560
dominant_topic	0.054593	-0.019299	-0.116079	0.191528	0.226298	0.066465	0.009827	0.044245	-0.078978	0.237418	0.001072	-0.050746	0.006381	0.087706	0.004472	-0.084300	0.052960	-0.145169	0.147880	0.030963	0.029560	1.000000

Figure 1: Covariance Matrix for numerical data and sentiment data.

Our covariance matrix included both our numerical data and data from lyrical analysis. In our project proposal, our initial goal was to develop two models simultaneously: one for the numerical audio features and one for the analysis of lyrical features, then combine them in some form of ensemble model. However, upon further inspection, it became incredibly difficult to predict song popularity solely based off of lyrical features and analysis. For example, we didn't see any strong correlations between the lyrical analysis data and popularity in our covariance matrix. Therefore, our goal became to incorporate these lyrical features into a singular model with the numerical features, creating our multi-modal model in this fashion instead.

3.2 Traditional Regression Methods

Before implementing more complex models, we established a baseline using traditional regression techniques:

- **Linear Regression**: We performed multivariate linear regression as an initial benchmark for predicting popularity scores. The model minimizes the mean squared error:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 \tag{1}$$

where β represents the model weights, x_i the feature vector for song i, and y_i the popularity score.

- **Regularized Regression**: We implemented Ridge, Lasso, and ElasticNet regression to perform feature selection and combat overfitting. Ridge regression adds L2 regularization:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + \alpha \|\beta\|_2^2 \tag{2}$$

Lasso uses L1 regularization:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + \alpha \|\beta\|_1 \tag{3}$$

ElasticNet combines both approaches:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + \alpha \left(\rho \|\beta\|_1 + (1-\rho) \|\beta\|_2^2\right) \tag{4}$$

where α is the regularization strength and ρ is the L1 ratio in ElasticNet.

- **Kernel Ridge Regression**: We utilized Kernel Ridge Regression with various kernels (linear, RBF, polynomial) to capture non-linear relationships in the data. This extends Ridge regression by applying the kernel trick:

$$\min_{\alpha} \|y - K\alpha\|_2^2 + \lambda \alpha^T K \alpha \tag{5}$$

where K is the kernel matrix with elements $K_{ij} = k(x_i, x_j)$ representing the similarity between data points, and λ is the regularization parameter.

3.3 Neural Network Approaches

For more complex modeling, we implemented neural network architectures optimized for our specific task:

- **Optimized Neural Network (PyTorch)**: We developed a custom neural network in PyTorch specifically designed for music popularity prediction. The network architecture included batch normalization, expanded width in early layers, and dropout for regularization:

$$\text{Input} \to \text{BatchNorm1d} \to \text{FC}(input\_size, 512) \to \text{BatchNorm1d} \to \text{ELU} \to \text{Dropout}(0.4) \tag{6}$$

$$\to \text{FC}(512, 256) \to \text{BatchNorm1d} \to \text{ELU} \to \text{Dropout}(0.4) \tag{7}$$

$$\to \text{FC}(256, 128) \to \text{BatchNorm1d} \to \text{ELU} \to \text{Dropout}(0.3) \tag{8}$$

$$\to \text{FC}(128, 64) \to \text{BatchNorm1d} \to \text{ELU} \to \text{Dropout}(0.2) \tag{9}$$

$$\to \text{FC}(64, 1) \tag{10}$$

We used a combination of MSE and Huber loss for training:

$$\mathcal{L} = 0.7 \cdot \mathcal{L}_{\text{MSE}} + 0.3 \cdot \mathcal{L}_{\text{Huber}} \tag{11}$$

This hybrid loss function provided greater robustness to outliers while maintaining the benefits of MSE. We optimized using AdamW with cyclical learning rates, which greatly improved convergence speed and model performance:

$$\text{OneCycleLR} : \text{max\_lr} = 0.01, \text{pct\_start} = 0.3, \text{div\_factor} = 25.0 \tag{12}$$

- **Neural Network with MLPRegressor (scikit-learn)**: We implemented a fully-connected neural network using scikit-learn's MLPRegressor with optimized hyperparameters:

$$\text{Input} \to \text{FC}(input\_size, 512) \to \text{ReLU} \tag{13}$$

$$\to \text{FC}(512, 256) \to \text{ReLU} \tag{14}$$

$$\to \text{FC}(256, 128) \to \text{ReLU} \tag{15}$$

$$\to \text{FC}(128, 64) \to \text{ReLU} \tag{16}$$

$$\to \text{FC}(64, 1) \tag{17}$$

The model was trained using the Adam optimizer with adaptive learning rates and early stopping to prevent overfitting. The objective function was mean squared error:

$$\min_{w,b} \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i; w, b))^2 \tag{18}$$

where $f(x_i; w, b)$ represents the neural network output for input x_i with weights w and biases b.

3.4 Natural Language Processing of Lyrics

We initially planned to use LSTMs to capture sequential language patterns. However, we ultimately realized that LSTM's were too complex, so we decided to use logistic regression and LDA to get our sentiment and topic analysis. To uncover both the thematic structure and the affective tone of our lyrics, we implemented a Latent Dirichlet Allocation (LDA) model to extract recurring topics and a logistic-regression-based sentiment analyzer to extract the mood of a song.

Preprocessing Language Data. One of the largest challenges we faced when preparing the lyrics data was the number of non-English lyrics. To maintain a consistent meaning for our sentiment analysis, the data must first be translated into English. However, this was an incredibly time-consuming and finicky process with various API's that limited the number of songs that could be requested per second and the number of characters that could be input. Due to ambiguous error codes from our initial translation API, we learned to navigate GitHub documentation to understand these nuances and code solutions around them to avoid paying exorbitant prices to translate using other services such as Google Cloud's Translation API.

After the lyrics were translated, We begin by pre-processing each lyric with the following:

1. Lowercasing & Punctuation Removal: using a regular expression, we striped any unnecessary extraneous headers and punucation.
2. Stop-word Filtering: We maintain a custom English stop-word list and drop any token in that list.
3. Vocabulary Pruning: Each remaining word is assigned to a unique integer ID; we count its total occurrences across the entire corpus and discard any word whose global frequency was below 10.

After these steps, the lyrics of each song are represented as a sparse 'bag-of-words' vector where each word has an ID count.

Topic Modeling with LDA. On top of our pruned vocabulary, we run a gensim LdaModel to discover 8 latent topics. Each topic is clustering of words that appear across all songs in a sufficient frequency, and each song includes a probability distribution of each topic. The dominant topic for each song, the one with the highest probability, is assigned as the song's topic.

Sentiment Analysis with Logistic Regression. In parallel, we built a sentiment classifier from first principles scoring sentiment from 0 to 1, where 0 is the most negative and 1 is the most positive. We trained the model on Sentiment140, a dataset of 1,600,000 tweets that have pre-labeled sentiment scores. This was done because our own songs do not have sentiment labels, so we were required to outsource textual data with sentiment analysis scores in order to create a training and validation dataset for our model.

1. Extract the same bag-of-words features as above.
2. Initialize weights $\mathbf{w}$ and bias b to zero.
3. Optimize the logistic-loss

$$\min_{\mathbf{w},b} \sum_{i=1}^{N} \log\bigl(1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x}_i + b))\bigr)$$

over 200 iterations.

4. At convergence, apply the learned model to every song to produce a real-valued "sentiment score" (the signed distance from the decision boundary).

These sentiment scores, together with the topic-posterior distributions from LDA, feed directly into our downstream popularity-prediction models—enabling us to quantify how thematic content and emotional tone jointly influence a track's success.

4 Implementation Details

4.1 Data Preprocessing

We prepared our data through several sequential steps: converting string columns to numeric values using label encoding; cleaning data by removing parentheses and whitespace; dropping rows with missing values and standardizing features with StandardScaler. The data was split into training (80%), validation (10%) and testing (10%) sets with a random state of 72 for reproducibility.

4.2 Model Implementation

4.2.1 Linear and Regularized Regression

We implemented the following regression models from scikit-learn with the following configurations: Linear Regression with default parameters; Ridge with $\alpha = 1.0$; Lasso with $\alpha = 0.1$; and ElasticNet with $\alpha = 0.1$ and $l1\_ratio = 0.5$.

4.2.2 Kernel Ridge Regression

To handle our large dataset efficiently, we developed a memory-optimized Kernel Ridge implementation. We used stratified sampling to select 15,000 training examples, tested multiple kernels (linear, RBF, polynomial) with various hyperparameters, and performed 3-fold cross-validation. The best configuration (RBF kernel, $\alpha = 0.1$, $\gamma = 0.1$) was selected based on validation R^2 and trained on a subset of 30,000 examples due to computational limits.

4.2.3 Neural Networks

PyTorch Implementation Our custom `OptimizedSpotifyNN` architecture featured five blocks with decreasing width (input→512→256→128→64→1). Each block included a linear layer, batch normalization, ELU activation, and dropout (rates from 0.4 to 0.2). We initialized weights using Xavier uniform distribution and trained with AdamW optimizer ($lr = 0.01$, weight_decay=1e-3, amsgrad=True). The learning rate followed a OneCycleLR schedule (max_lr=0.01, pct_start=0.3), and we used a combined loss function (70% MSE, 30% Huber). Training proceeded with batch size 512 for 400 epochs with early stopping (patience=40), employing gradient clipping (max_norm=1.0) for stability.

Scikit-learn Implementation The MLPRegressor used a similar decreasing-width architecture (hidden_layer_sizes=(512, 256, 128, 64)) with ReLU activation. We configured the Adam solver with $\alpha = 0.001$, batch_size=256, and adaptive learning rate (initial_learning_rate=0.01). Training continued for up to 500 iterations with early stopping (validation_fraction=0.15, n_iter_no_change=20, tol=1e-4).

Model Type	Parameter	Value
Regularized	Ridge α	1.0
	Lasso α	0.1
	ElasticNet α, $l1\_ratio$	0.1, 0.5
Kernel Ridge	Best kernel	RBF
	Best parameters	$\alpha = 0.1$, $\gamma = 0.1$
PyTorch NN	Architecture	input→512→256→128→64→1
	Optimizer	AdamW(lr=0.01, weight_decay=1e-3)
	Loss function	0.7·MSE + 0.3·Huber
	Batch size, Epochs	512, 400 (with early stopping)
	Regularization	BatchNorm, Dropout(0.4-0.2), gradient clipping
Scikit-learn NN	Architecture	input→512→256→128→64→1
	Solver, Activation	Adam, ReLU
	Alpha, Batch size	0.001, 256
	Learning rate, Max iter	adaptive (init=0.01), 500

Table 1: Summary of model configurations and hyperparameters

Both neural network implementations were monitored during training to ensure proper convergence and prevent overfitting, with metrics tracked across training, validation, and testing phases.

4.3 Lyrical Analysis Implementation

4.3.1 LDA_topic Implementation

To implement the LDA model, we first preprocessed data by removing unecessary conjugates and punctuation. Then, we grouped words from lyrics into 8 LDA topic categories depending on the word popularity and similarity to other words. Each topic includes the 10 most representative words discovered by the LDA model:

- Topic 0: low, high, bang, boom, de, yo, woop, tempo, bounce, power
- Topic 1: light, christmas, year, home, king, star, sky, ring, tonight, sun
- Topic 2: heart, thing, away, ever, even, mind, said, find, hold, head
- Topic 3: n****, b****, f***, s***, money, man, huh, lil, gang, new
- Topic 4: dance, taste, going, body, crazy, move, turn, girl, stop, hot

- Topic 5: girl, good, bad, better, boy, bout, feelin, nothin, somethin, work
- Topic 6: everything, people, u, much, many, money, two, year, city, bomb
- Topic 7: little, alone, leave, heart, call, every, stay, forget, wait, nobody

4.3.2 Sentiment Analysis Implementation

We implemented sentiment analysis using TF-IDF vectorization and Sklearn's logistic regression on all the data, tokenizing similar to our methodology for the LDA topics. Our resulitng model was trained on Sentiment140 before utilizing it on our own dataset of lyrics.

5 Results

We evaluated all models using two primary metrics: Mean Squared Error (MSE) and coefficient of determination (R^2). The results are summarized in the table below:

Model	**Train MSE**	**Test MSE**	**Train R^2**	**Test R^2**
Linear Regression	0.2212	0.2214	0.0949	0.0938
Neural Network (scikit-learn)	0.1572	0.1702	0.3567	**0.3034**
Neural Network (PyTorch)	0.1599	0.1733	0.3456	0.2905
Ridge	0.2212	0.2214	0.0949	0.0938
Lasso	0.2429	0.2429	0.0059	0.0057
ElasticNet	0.2315	0.2316	0.0527	0.0519
Kernel Ridge	0.1539	0.1905	**0.3703**	0.2201

Table 2: Performance comparison of different models. Bolded are the models with highest training, testing performance.

Based on these results, the scikit-learn MLPRegressor neural network achieved the best performance, with a test MSE of 0.1702 and test R^2 of 0.3034. The PyTorch neural network followed closely with a test MSE of 0.1733 and test R^2 of 0.2905. Both neural network implementations significantly outperformed the traditional regression methods, indicating that the relationship between the features and popularity score is non-linear and requires more complex modeling approaches. However, compared to Gulmatico et. al's state of the art model HitMusicNet that predicts song popularity, we are far off of their best test MSE values of 0.0112 [2].

The Kernel Ridge Regression model with RBF kernel also performed reasonably well, achieving a test R^2 of 0.2201, demonstrating the importance of capturing non-linear relationships in the data. However, it showed signs of overfitting with a larger gap between training and testing performance compared to the neural networks.

The traditional linear methods (Linear Regression, Ridge, Lasso, and ElasticNet) performed relatively poorly, with test R^2 values ranging from 0.0057 (Lasso) to 0.0938 (Linear Regression and Ridge). This poor performance further confirms that simple linear models are inadequate for capturing the complex patterns in music popularity data.

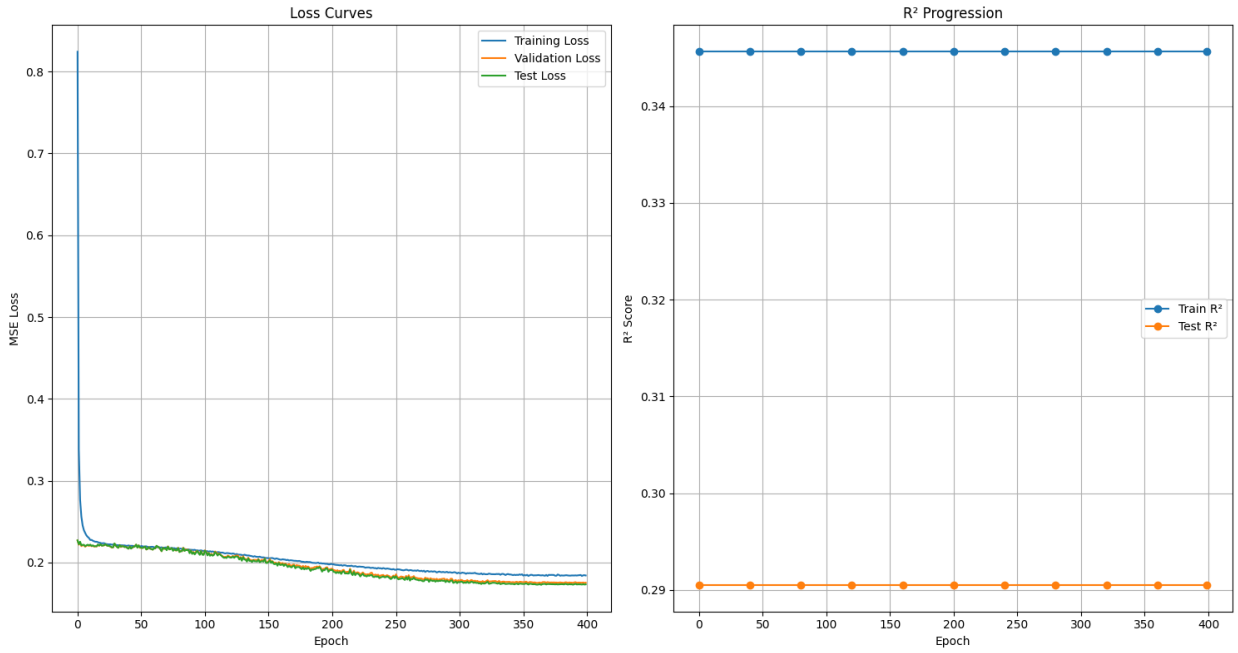

Figure 2: Training and evaluation metrics for the PyTorch neural network. Left: Loss curves for training, validation, and test sets. Right: R^2 progression throughout training.

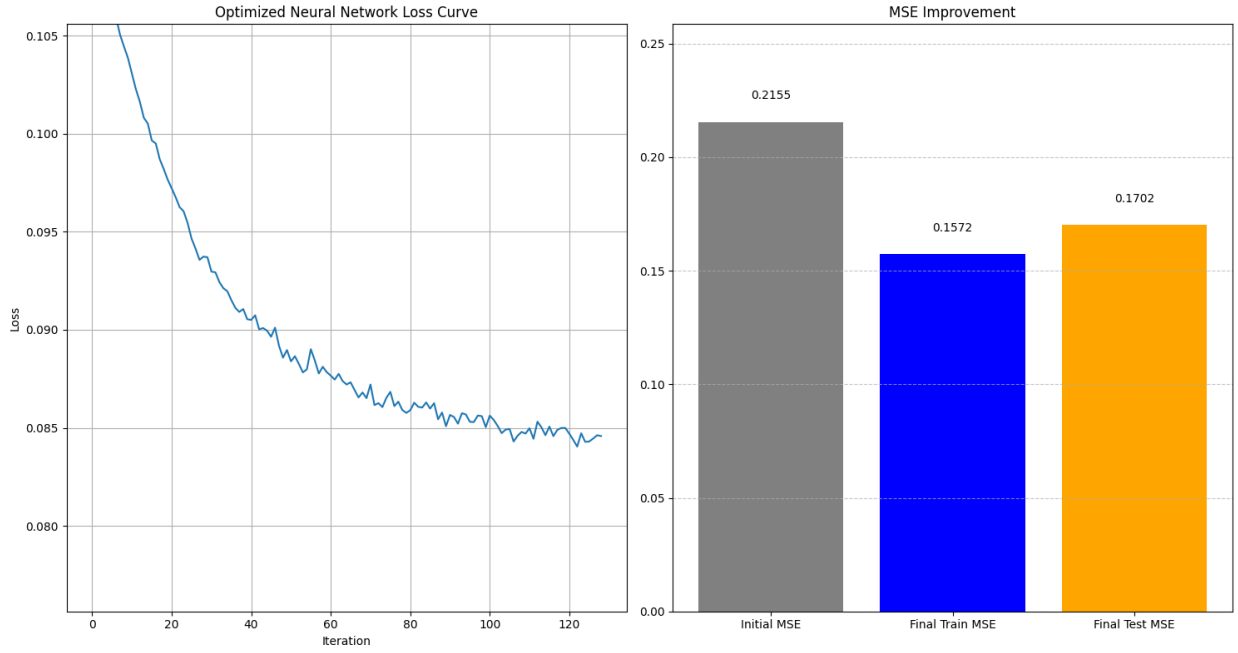

Figure 3: Performance of the scikit-learn neural network. Left: Loss curve showing convergence during training. Right: Comparison of initial and final MSE values.

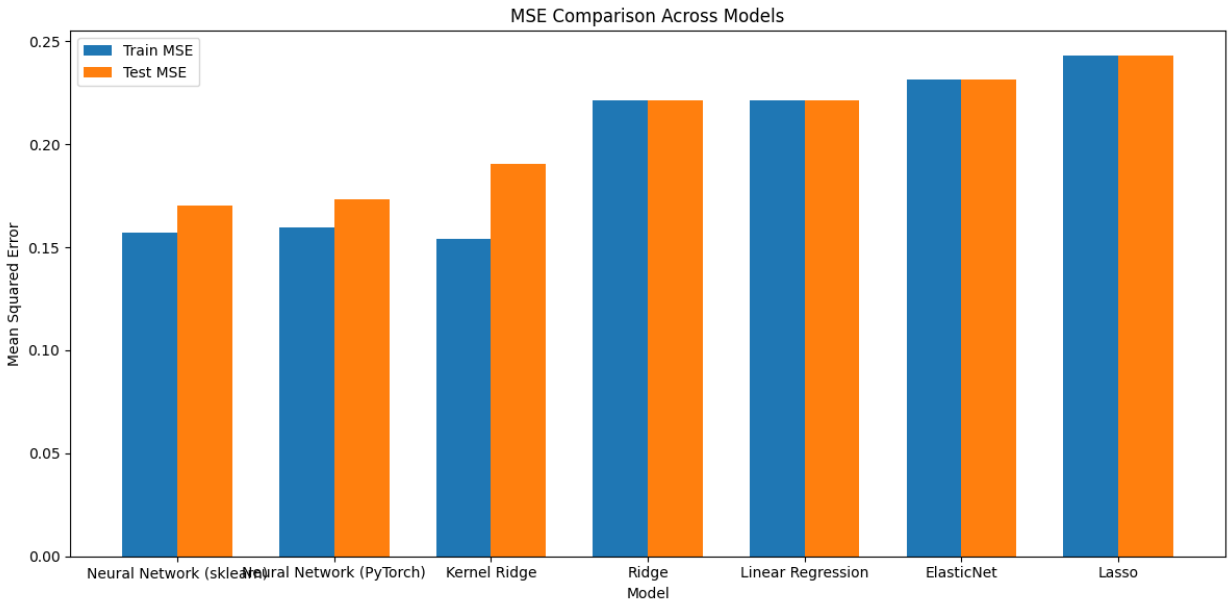

Figure 4: MSE comparison across all models for both training and test sets.

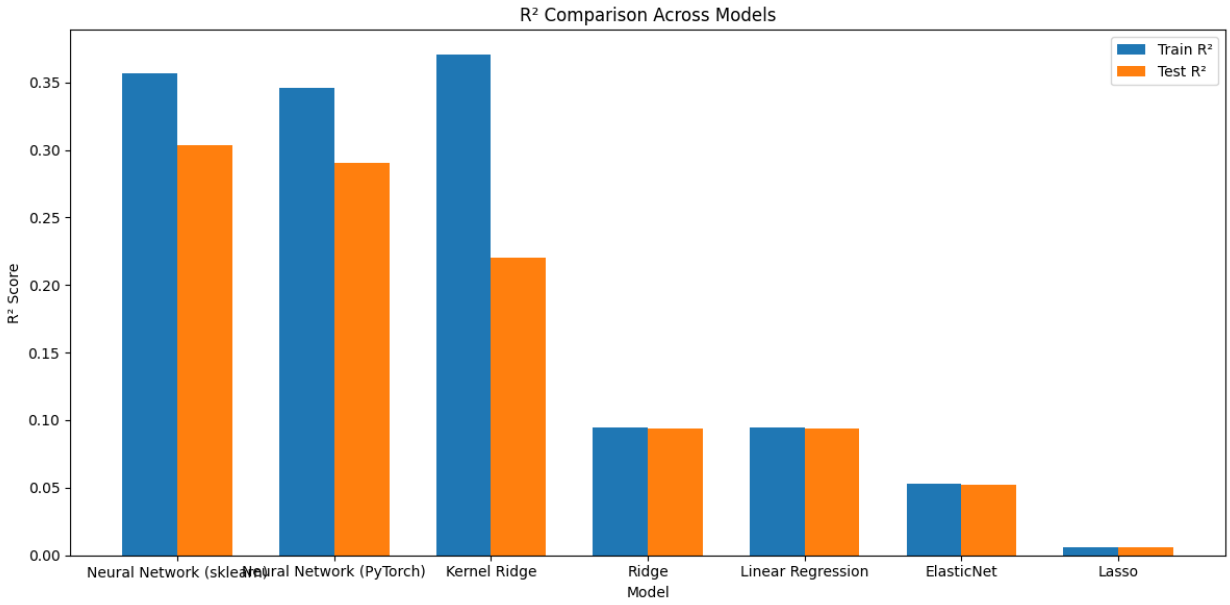

Figure 5: R^2 comparison across all models for both training and test sets.

Our results demonstrate that neural network models significantly outperform traditional regression methods for predicting song popularity based on audio features, metadata, and artist information. The performance improvements achieved by the neural networks suggest

that there are complex, non-linear relationships between song characteristics and popularity that simpler models cannot capture.

The PyTorch neural network showed substantial improvement throughout training, with the loss decreasing from an initial value of 0.7288 to a final training loss of 0.1842—a 74.72% reduction. This improvement, coupled with the network's ability to maintain good performance on the test set, indicates effective generalization.

Similarly, the scikit-learn neural network achieved a 27.04% improvement over its initial MSE, demonstrating steady convergence over approximately 130 iterations before early stopping was triggered.

6 Conclusion

Throughout this project, we explored various models, with the ultimate goal of learning how to best predict the popularity of a song using audio features, metadata, and lyrics. We began by analyzing our baseline numerical data using ridge regression and neural networks. For lyrical analysis, we initially planned to use LSTMs to capture sequential language patterns, however, we ultimately realized that LSTM's were too complex, so we decided to use logistic regression and LDA to get our sentiment and topic analysis. We found that these methods were more efficient to implement and also allowed us to translate into numerical features for further analysis.

In the end, we found that our most successful model was a neural network implemented using scikit learn. To test the contribution of our lyrical analysis, we ran this model with and without our sentiment and topic features. Interestingly, we found that there was no significant improvement to our results. While we initially expected that lyrics would play a stronger role in a song's popularity. Our results suggest that audio features and artist metadata may be more influential. This project highlights the challenges of modeling a subjective topic like song popularity, where the measure of 'success' can depend on many complex factors like cultural context and marketing.

References

[1] pepepython. (2023). *Spotify Huge Database - Daily Charts Over 3 Years* [Dataset]. Kaggle. `https://www.kaggle.com/datasets/pepepython/spotify-huge-database-daily-charts-over-3-years`

[2] Gulmatico, J. S., Susa, J. A. B., Malbog, M. A. F., Acoba, A., Nipas, M. D., & Mindoro, J. N. (2022). "SpotiPred: A Machine Learning Approach Prediction of Spotify Music Popularity by Audio Features," 2022 Second International Conference on Power, Control and Computing Technologies (ICPC2T), Raipur, India, pp. 1-5, doi: 10.1109/ICPC2T53885.2022.9776765.