Predicting Song Similarity with Deep Learning

Aimee Langevin, FNU Annanya, Weiqian Zhang
11-785: Introduction to Deep Learning
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Understanding why two songs feel similar is a central challenge in music recommendation. Many platforms rely on user behavior to drive recommendations, but this introduces popularity biases and often misses intrinsic content-based similarities. In this project, we re-implemented the *DeepPlaylist* models that use lyrics and audio inputs to classify whether two songs are similar [1]. Our re-implementation matched the reported classification performance, achieving approximately 80% accuracy on lyrics-based models. Building on this foundation, we reframed the task from binary classification to regression, aiming to predict similarity scores on a continuous scale. This shift better captures the nuanced nature of musical similarity and opens up more flexible recommendation strategies. Our results show that regression-based models using both lyrics and audio can more effectively represent the spectrum of song similarity compared to binary classification.

1 Overview and context

1.1 Motivation

Music recommendation systems today are typically powered by collaborative filtering, which relies on patterns in user listening behavior. While effective at scale, these systems reinforce popularity bias, making it difficult for lesser-known or niche songs to reach the right audiences. This not only limits user discovery but also disadvantages emerging artists who may have highly relevant content but lack widespread exposure.

Building more inclusive and accurate recommendation systems requires shifting focus from user data to the intrinsic properties of the music itself. By directly analyzing lyrics and audio, content-based models have the potential to surface a broader and more personalized range of recommendations. This project revisits *DeepPlaylist*, a content-based similarity framework, and extends its lyrics-based classification baseline into a regression approach while also incorporating audio representations [1]. Improving content-based similarity prediction can benefit listeners by enabling deeper discovery experiences, support artists by leveling the playing field, and empower platforms to deliver more meaningful recommendations.

1.2 Objective

The primary objective of this project is to evaluate and extend content-based approaches for predicting song similarity. Specifically, we aim to:

- Replicate the *DeepPlaylist* lyrics-based BiLSTM classification model and achieve comparable classification accuracy (80%) on song similarity prediction [1].
- Validate the effectiveness of content-based models by matching or exceeding the reported baseline performance using lyrics alone.

- Extend the task from binary classification to regression, predicting continuous similarity scores to better capture nuanced relationships between songs.
- Incorporate audio features alongside lyrics to explore potential gains from multi-modal input representations.
- Quantitatively benchmark models using accuracy (for classification) and mean absolute error (MAE) (for regression).
- Lay the groundwork for future improvements in music recommendation systems that reduce popularity bias and promote better discovery.

2 Related work

DeepPlaylist by Balakrishnan and Dixit presents a neural architecture for classifying song similarity using lyrics or audio content [1]. Their work highlights a key limitation of collaborative filtering methods in recommendation systems: the lack of intrinsic content-based modeling. Instead, *DeepPlaylist* proposes a BiLSTM-based framework where the lyrics of two songs are processed and compared to output a binary similarity label [1]. We aim to re-implement their lyrics-based classification model and validated its performance against the original reported accuracy (around 80%).

The design of *DeepPlaylist* was influenced by vector combination strategies proposed by Mou et al., who explored different methods—concatenation, subtraction, and elementwise product—for comparing sentence representations in natural language tasks [1][5]. Similarly, we evaluated these combination methods in our own implementation to understand their impact on model accuracy.

While *DeepPlaylist* primarily used classification accuracy as the evaluation metric, our project extends the task to a regression setting. Prior work in similarity learning, such as by Bromley et al. with Siamese networks and more recent work in metric learning (Schroff et al.), has demonstrated the value of predicting continuous similarity scores rather than discrete labels [4][6]. In line with these approaches, we use mean absolute error (MAE) and root mean squared error (RMSE) as quantitative metrics, which are standard for evaluating regression models predicting similarity (e.g., in face verification and recommendation systems). This shift in evaluation metrics allows for a finer-grained analysis of how closely predicted similarity scores align with ground-truth data.

3 Methodology

3.1 Lyrics model

3.1.1 Model description

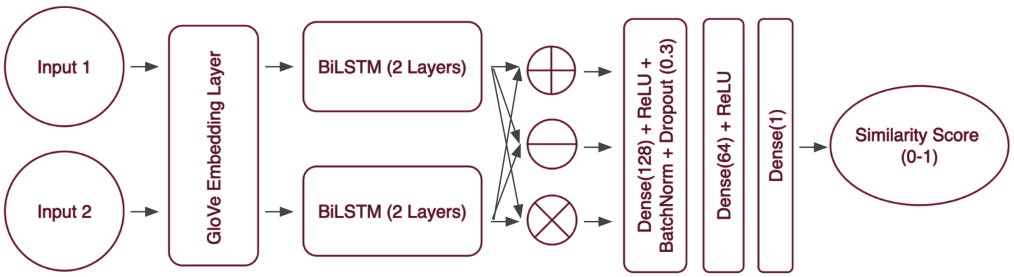

Figure 1: Architecture of lyrics-based regression model.

Our lyrics-based model extends the DeepPlaylist baseline by adapting it for continuous similarity prediction. The model takes two sequences of lyrics as inputs, each represented as a padded sequence of word indices after tokenization. A shared embedding layer, initialized with GloVe pre-trained word vectors, maps these indices to dense word embeddings. The embedding layer is trainable to allow fine-tuning during training.

Each embedded lyrics sequence is processed by two stacked bidirectional LSTM layers, capturing sequential dependencies in both forward and backward directions. The outputs of the two branches are compared using four operations: direct concatenation, elementwise subtraction, elementwise product, and direct encoded representations. These comparison vectors are concatenated into a single feature vector.

The merged feature vector is passed through two dense layers with L2 regularization, batch normalization, ReLU activations, and dropout for regularization. Finally, a single output neuron produces a scalar prediction corresponding to the predicted similarity score between the two input songs. The model is trained using mean squared error (MSE) loss and evaluated primarily using mean absolute error (MAE).

This architecture preserves the sequential modeling strengths of the original DeepPlaylist framework while adapting it for a regression setting, allowing for more nuanced and fine-grained similarity predictions compared to binary classification.

Output shapes and parameter counts

The model processes inputs of shape $(\text{batch size}, 300)$, corresponding to padded sequences of word indices. After embedding into a 100-dimensional space, the input has shape $(\text{batch size}, 300, 100)$. The first BiLSTM layer (256 units, bidirectional) produces output of shape $(\text{batch size}, 300, 512)$, and the second BiLSTM layer (128 units, bidirectional) reduces this to $(\text{batch size}, 256)$. After combining representations through concatenation, subtraction, and elementwise product, the merged feature vector has shape $(\text{batch size}, 1024)$. Subsequent dense layers progressively reduce this to a final scalar similarity score.

Key parameter counts:

- Embedding layer: 2,000,000 parameters
- First BiLSTM (256 units): 731,136 parameters
- Second BiLSTM (128 units): 656,384 parameters
- Dense layers ($128 \rightarrow 64 \rightarrow 1$): 139,521 parameters

The overall model contains approximately 3.5 million trainable parameters.

3.1.2 Dataset

Similar to *DeepPlaylist*, our dataset combines similarity scores from the LastFM dataset with lyrics from the musiXmatch dataset, both linked through the Million Song Dataset (MSD) [2][3]. We constructed pairs of songs, each labeled with a continuous similarity score ranging from 0 to 1.

We first extracted song pairs and similarity scores from the LastFM database, where each source song lists a set of target songs with associated similarity scores. To align these with lyrics, we parsed the musiXmatch dataset, which provides bag-of-words (BOW) representations of song lyrics based on a controlled vocabulary of frequent words.

For each song pair, we reconstructed approximate text representations by expanding the BOW format back into word sequences. We filtered out pairs where either song lacked lyric information, ensuring both inputs contained valid reconstructed lyrics. Positive (similar) and negative (dissimilar) pairs were included to maintain a balanced dataset for training.

The final dataset consists of approximately 1 million pairs of reconstructed lyrics, each associated with a continuous similarity score. We split the resulting data into training (64%), validation (16%), and test (20%) sets, ensuring consistent distribution across similarity scores.

3.1.3 Evaluation metrics

Since our task is to predict a continuous similarity score between two songs, we evaluate model performance using regression metrics rather than classification accuracy. Specifically, we report Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) on the held-out test set.

- **Mean Absolute Error (MAE)** measures the average absolute difference between the predicted similarity scores $\hat{y}_i$ and the true similarity scores y_i across all n examples. It is

defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

- **Root Mean Squared Error (RMSE)** measures the square root of the average squared difference between the predicted and true similarity scores. It penalizes larger errors more strongly than MAE. It is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Where:

- n is the number of song pairs in the test set,
- $\hat{y}_i$ is the predicted similarity score for the i-th song pair,
- y_i is the ground-truth similarity score for the i-th song pair.

Both metrics are minimized when the model's predictions closely match the ground-truth similarity labels. Lower MAE and RMSE values indicate better performance in accurately predicting how similar two songs are based on their lyrics.

We monitor validation MAE during training to select the best model checkpoint and report both MAE and RMSE on the final test set to assess generalization performance.

3.1.4 Loss function

The model is trained to minimize the Mean Squared Error (MSE) loss between the predicted similarity scores and the ground-truth labels. The MSE loss is defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

where:

- n is the number of song pairs in a batch,
- $\hat{y}_i$ is the predicted similarity score for the i-th song pair,
- y_i is the ground-truth similarity score for the i-th song pair.

Minimizing MSE encourages the model to predict similarity scores that are close to the true labels on average, with larger errors penalized more heavily due to the squared term. This is appropriate for our task, where fine-grained similarity prediction is desired.

Optimization is performed using the Adam optimizer, an adaptive gradient-based method that maintains separate learning rates for each parameter. The parameter update rule at each iteration t is:

$$\theta_t = \theta_{t-1} - \alpha\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where:

- θ_t are the model parameters at step t,
- α is the learning rate,
- $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected estimates of the first and second moments (mean and uncentered variance) of the gradients,
- ϵ is a small constant for numerical stability.

We use an initial learning rate of $\alpha = 0.001$ as recommended for Adam, and optimize the model by minimizing the MSE loss over the training data.

3.2 Audio model

3.2.1 Model description

In our fused model, the audio branch is handled by a dedicated AudioEmbeddingCNN. This model is designed to extract rich, compact embeddings from the raw audio representations. Specifically, we use features like timbre (which captures tone color, texture, and quality) and chroma (which captures harmonic and pitch-related aspects) to represent different nuanced characteristics of the audio signal.

The AudioEmbeddingCNN processes the input through a series of convolutional layers with batch normalization and ReLU activations, followed by max-pooling to progressively capture local and global structures in the audio. After the convolutional stages, the feature maps are flattened and passed through fully connected layers to produce a 128-dimensional audio embedding. This embedding is L2 normalized to ensure it lies on a unit hypersphere, making it suitable for downstream similarity comparisons.

The resulting embedding captures detailed aspects of the audio's texture (via timbre) and musical content (via chroma), ensuring the model learns not just broad characteristics, but also fine-grained differences important for music similarity tasks. We also tried using trasfer learning from ResNet. Both models aim to project raw audio features into a shared embedding space, where similar songs are close together, and dissimilar ones are far apart.

3.3 Audio Feature Extraction

To capture the nuanced characteristics of audio signals critical for music similarity, we design a custom convolutional neural network termed **AudioEmbeddingCNN**. The architecture consists of three convolutional layers with progressively increasing channel dimensions: 32, 64, and 128 channels, respectively. Each convolutional layer is followed by batch normalization to stabilize training dynamics, a ReLU activation to introduce non-linearity, and a max-pooling operation to downsample the feature maps while preserving important frequency and temporal structures. After the convolutional blocks, the resulting feature maps are flattened and passed through two fully connected (dense) layers. A dropout layer with a 30% dropout rate is applied to mitigate overfitting. Finally, the audio embedding output is projected onto the unit hypersphere using L2 normalization. Formally, if f denotes the flattened feature vector, the final embedding e is computed as:

$$e = \frac{W_2(\text{ReLU}(W_1 f + b_1)) + b_2}{\|W_2(\text{ReLU}(W_1 f + b_1)) + b_2\|_2}$$

where W_1, W_2 and b_1, b_2 are the weights and biases of the dense layers. This design allows the network to effectively learn fine-grained differences in timbre (color or quality of sound) and chroma (pitch-related information) across diverse musical recordings.

3.4 Siamese Network Architecture

To measure pairwise similarity between two music inputs, we employ a **Siamese Network** structure with shared weights. In this architecture, two different audio inputs, Audio_1 and Audio_2, are independently passed through the same AudioEmbeddingCNN encoder to generate embeddings:

$$e_{\text{audio1}} = \text{AudioEncoder}(\text{Audio}_1), \quad e_{\text{audio2}} = \text{AudioEncoder}(\text{Audio}_2)$$

Similarly, their corresponding lyrics, Lyrics_1 and Lyrics_2, are processed through a LyricsEncoder to produce:

$$e_{\text{lyrics1}} = \text{LyricsEncoder}(\text{Lyrics}_1), \quad e_{\text{lyrics2}} = \text{LyricsEncoder}(\text{Lyrics}_2)$$

Thus, we obtain four embeddings: two audio-based and two lyrics-based.

3.5 Feature Merging Strategies

After obtaining the embeddings, we model the relationship between the two music pieces using a feature merging strategy. Several methods were explored:

- **Concatenation:**
$$\text{Merged} = [e_{\text{audio1}}; e_{\text{audio2}}; e_{\text{lyrics1}}; e_{\text{lyrics2}}]$$
where $[\,;\,]$ denotes vector concatenation.
- **Subtraction:**
$$\text{Merged} = e_{\text{audio1}} - e_{\text{audio2}}$$
- **Multiplication:**
$$\text{Merged} = e_{\text{audio1}} \times e_{\text{audio2}}$$
- **Mean:**
$$\text{Merged} = \frac{e_{\text{audio1}} + e_{\text{audio2}}}{2}$$
- **Cosine Similarity:** Instead of explicit merging, we directly compute the cosine similarity between the two embeddings:
$$\cos(e_{\text{audio1}}, e_{\text{audio2}}) = \frac{e_{\text{audio1}} \cdot e_{\text{audio2}}}{\|e_{\text{audio1}}\|_2 \|e_{\text{audio2}}\|_2}$$

In our main setup, we adopt **concatenation** as the merging strategy, allowing richer feature fusion by preserving both individual characteristics and relational patterns between audio and lyrics embeddings.

3.6 Similarity Decoder

The merged feature vector is subsequently passed through a **Similarity Decoder**, designed to predict a similarity score between 0 and 1. The decoder comprises three fully connected layers with progressively decreasing dimensions, each followed by batch normalization, ReLU activation, and dropout (dropout rate = 0.3). The final output layer applies a sigmoid activation to produce the probability score. Mathematically, the output s is:

$$s = \sigma(W_4(\text{ReLU}(W_3(\text{ReLU}(W_2(\text{ReLU}(W_1\text{Merged} + b_1)) + b_2)) + b_3)) + b_4)$$

where $\sigma(\cdot)$ denotes the sigmoid function, W_i and b_i are the weights and biases of the fully connected layers. This design allows the model to capture complex non-linear interactions between the features, enabling accurate prediction of similarity between musical pieces.

3.6.1 Dataset

Data preparation

To support our multi-modal similarity regression model, we curated a dataset that integrates both audio and lyrics representations for pairs of songs with known similarity scores. The dataset was constructed by aligning and filtering the LastFM similarity data, musiXmatch lyrics data, and precomputed audio features from the Million Song Dataset (MSD).

Lyrics Pair Construction: We began by loading song pairs and their similarity scores from pre-filtered CSV or pickle files. The lyrics data was parsed to extract individual lyrics for each track in a pair. For each valid pair (track1, track2), we checked whether both songs had corresponding lyric entries. We dropped any pair where lyrics were missing or not reconstructible, ensuring all samples had complete lyric information.

Audio Feature Alignment: We also ensured both track IDs in a given pair were found in our preloaded audio dictionary. This dictionary maps track IDs to waveform-based embeddings extracted via pre-trained ResNet or CNN models. We removed any entries with missing or None values in the audio dictionary to prevent model input errors during training.

Exporting the Dataset: Our project, just like previous works done on the MSD and LastFM, is very severely constrained by the computing resources and especially the memories, given that audio files are typically very large, making the resulting features large. To streamline model development and reuse, we serialized each processed audio using Python's pickle module. Each output pickle file contains roughly 1,000 songs and takes up approximately 1 GB of storage. Full datasets can therefore reach sizes of 30 GB or more, and remember we were talking about clipped audios, not the whole songs. This project uses 4 thousand rows as a batch.

Merging and Splitting: After filtering and balancing, we merged the positive and negative samples into a single dataset. The dataset was randomly shuffled and split into training (80 %), validation (10%), and test (10%) subsets. Each data point in the final dataset includes:

Two lyrics inputs (as tokenized sequences),

Two audio embeddings (as vectors),

A continuous similarity score between 0 and 1.

This final dataset was used to train and evaluate our Siamese-style model incorporating both audio and lyrics features.

3.6.2 Evaluation metrics

Our combined model also uses the **Mean Squared Error (MSE)** loss during training, consistent with the lyrics-only regression baseline. However, the results of the combined model cannot be directly compared with the lyrics model because we were relying on a subset of the lyrics dataset, as the above Dataset section illustrates:

Let $\mathcal{L}$ denote the full set of lyrics pairs with valid similarity scores, and let $\mathcal{C}$ denote the set of audio-lyrics pairs where both audio embeddings are available. Then:

$$\mathcal{A}_{\text{available}} \subset \mathcal{L}$$

$$\mathcal{C} = \mathcal{L} \cap \mathcal{A}_{\text{available}}$$

Where $\mathcal{A}_{\text{available}}$ is the subset of tracks for which valid audio embeddings exist. Thus, our evaluation for the combined model is limited to a resulting $\mathcal{C}$, the intersection of lyrics and audio-compatible data.

The results and analysis will be presented below in the corresponding section.

3.6.3 Loss function

The model is trained to minimize the Mean Squared Error (MSE) loss between the predicted similarity scores and the ground-truth labels. The MSE loss is defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

where:

- n is the number of song pairs in a batch,
- $\hat{y}_i$ is the predicted similarity score for the i-th song pair,
- y_i is the ground-truth similarity score for the i-th song pair.

Minimizing MSE encourages the model to predict similarity scores that are close to the true labels on average, with larger errors penalized more heavily due to the squared term. This is appropriate for our task, where fine-grained similarity prediction is desired.

Optimization is performed using the Adam optimizer, an adaptive gradient-based method that maintains separate learning rates for each parameter. The parameter update rule at each iteration t is:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where:

- θ_t are the model parameters at step t,
- α is the learning rate,
- $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected estimates of the first and second moments (mean and uncentered variance) of the gradients,
- ϵ is a small constant for numerical stability.

We use an initial learning rate of $\alpha = 0.001$ and optimize the model using the Adam optimizer. To improve convergence, we apply a cosine annealing learning rate scheduler with parameters $T_{\text{max}} = 30$ and $\eta_{\text{min}} = 10^{-6}$. This scheduler gradually reduces the learning rate following a cosine decay pattern, helping the model avoid local minima and improve generalization across epochs. The model is trained to minimize the MSE loss over the training data.

4 Baseline and extensions

4.1 Baseline selection & evaluation

To establish a reference point for our similarity prediction task, we re-implemented the lyrics-based classification model proposed in the *DeepPlaylist* paper [1]. The original work achieved approximately 80% validation accuracy by using a BiLSTM-based architecture to classify whether two songs were similar based solely on their lyrics [1]. We selected this architecture as our baseline due to its simplicity, strong reported performance, and relevance to content-based recommendation research.

Our baseline model consists of a shared embedding layer initialized with pre-trained GloVe word vectors, followed by a bidirectional LSTM encoder for each input song. The two encoded representations are combined using simple concatenation, passed through two fully connected layers with ReLU activations, batch normalization, and dropout for regularization. A final dense layer with a sigmoid activation outputs the predicted probability of similarity.

Training was conducted using the binary cross-entropy loss function and optimized with the Adam optimizer ($\alpha = 0.001$). We evaluated model performance using classification accuracy on the validation set.

Although the original *DeepPlaylist* paper explored various comparison methods between the encoded vectors—including concatenation, subtraction, and elementwise product—our baseline implementation used only concatenation [1]. After achieving approximately 80% validation accuracy with this simpler setup, we chose to prioritize extending the model to a regression task rather than exhaustively tuning the classification baseline. This decision was driven by the broader project goal of predicting continuous similarity scores, which required shifting away from binary classification.

To further improve model stability and generalization, we expanded the training set by incorporating a larger number of song pairs than initially used, increasing from 500,000 to 1,000,000. This adjustment allowed us to reach the target accuracy early in training, confirming that our re-implementation faithfully reproduced the core performance of the original *DeepPlaylist* baseline.

Baseline results:

- Validation Accuracy: **80.53%**
- Key modification: expanded dataset size to improve training stability
- Combination method: concatenation only

These closely resemble the results of the paper, which achieved an accuracy of 78.3% when concatenating and using two fully-connected layers. These results validated our baseline replication and provided a strong foundation for subsequent experiments involving regression and multi-modal extensions.

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.78      0.71      0.74     80745
         1.0       0.82      0.87      0.84    123349

    accuracy                           0.81    204094
   macro avg       0.80      0.79      0.79    204094
weighted avg       0.80      0.81      0.80    204094
```

Figure 2: Confusion matrix for classification baseline results.

4.2 Extensions & experiments

After moving from classification to regression, we conducted a series of controlled experiments to evaluate how model choices influenced similarity prediction performance. Table 1 in the following section summarizes the main ablations we performed.

Key aspects varied across experiments included:

- **Embedding Layer:** allowing the GloVe-initialized embedding layer to be trainable versus fixed.
- **LSTM Hidden Size:** increasing the size of LSTM units from 128 to 256.
- **Dropout Rate:** testing different dropout probabilities (0.3, 0.4, 0.5) for regularization.
- **Weight Regularization:** adding L2 kernel regularization to dense layers.
- **Data Size:** augmenting the training dataset with additional song pairs.

5 Results and analysis

5.1 Final results

5.1.1 Lyrics model

The regression model was trained for 10 epochs, with training progress monitored using mean squared error (MSE) loss and mean absolute error (MAE) metrics. Figure **??** shows the training and validation MSE loss curves over epochs. Both curves decrease steadily, indicating that the model effectively learned to predict song similarity without severe overfitting.

During training, the model achieved a final training MAE of **0.0720** and a validation MAE of **0.0670** at the best checkpoint. These values suggest good generalization from training to unseen validation data.

We conducted several ablation experiments to study the effect of key architectural and training choices on performance. Table 1 summarizes the validation MAE achieved for each experimental setting.

Table 1: Summary of Regression Model Ablations

Exp. ID	Changes	Train MAE	Val MAE	Test RMSE
1	Frozen embeddings, 128 LSTM, 0.3 dropout	0.0761	0.0715	0.1661
2	Trainable embeddings, 128 LSTM, 0.3 dropout	0.0758	0.071	0.157
3	256 LSTM units, 0.3 dropout	0.0711	0.0723	0.1561
4	256 LSTM, 0.5 dropout	0.0799	0.0754	0.1562
5	256 LSTM, 0.4 dropout, L2 regularization, more data	0.688	0.0712	0.1541
6	0.3 dropout	0.0702	0.06701	0.1543

Across experiments, we found that increasing the LSTM hidden size, fine-tuning embeddings, and applying moderate dropout with L2 regularization led to consistent improvements in validation performance. Experiment 5 yielded the best overall results.

After selecting the best model based on validation MAE, we evaluated final performance on the test set. The results are summarized in Table 2.

Table 2: Final test performance of the regression model.

Metric	Test Value
Mean Absolute Error (MAE)	0.0678
Root Mean Squared Error (RMSE)	0.1543

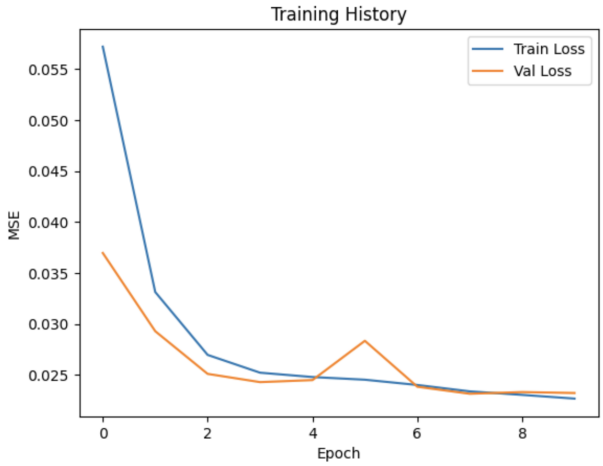

Figure 3: Training and validation loss of the lyrics regression model over 10 epochs.

The model achieved a test MAE of **0.0678** and a test RMSE of **0.0238**. These results indicate that on average, predicted similarity scores deviated from the ground truth by less than 0.07, confirming the model's ability to perform fine-grained similarity prediction.

While direct comparison to the original classification baseline is not straightforward due to the change in task formulation (classification to regression), our results demonstrate a strong ability to predict continuous similarity scores. The low MAE and RMSE values achieved highlight that the regression model captures more nuanced relationships between songs, providing a richer and more flexible foundation for content-based recommendation compared to binary classification.

5.1.2 Lyrics & Audio model

Figure 4 presents the training and validation loss of the combined model. Figure 5 presents the absolute errors between our predictions to the true labeled scores in the test datasets. Remember that we only choose to include labeled ones into the datasets for the combined models, so we do not have any inserted zeros. If a song pair does not have a label (which is the majority of the original Million Son Datasets), unlike the DeepPlaylist paper[1] where half of their dataset is all zeros either due to rounding or lack of label, we do not include this pair.

From the Figure 5 we can see that our model performs legitimately well that most of the predictions have less than 0.2 errors with the true labels, and occassionally it gets completely wrong, those are the places, upon looked up, where the true labels have extreme values like 0 or 1. So the model performs well in the range where the labels are not extreme to the poles. The final MSE score is 0.0863.

We have to admit that our training is severely constrained by the computing resources we had at hand. We are limited to work with Kaggle because mostly of our data processing was done at there and it's hard to migrate the whole databases and environments we've built, note that we have multiple datasets to work with: MSD, LastFM, Musixmatch, our pickles we zipped as decribed in the dataset section. Besides, the most constrainding condition caused by the Kaggle is its working memory limit. Unlike Colab before its deprecation of the instance market, we can choose the specs and RAM to work with by paying more, we were often restricted to use only one pickle or at most two at a time,

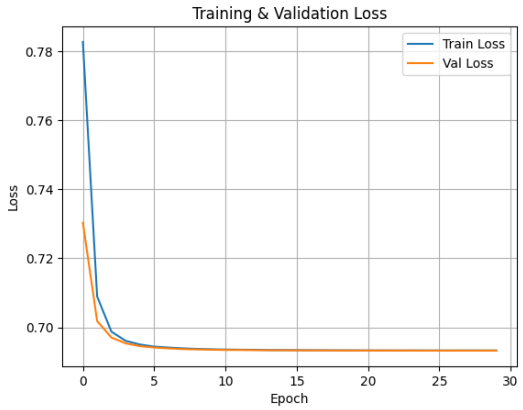

Figure 4: Training & Validation Loss for the Combined Model

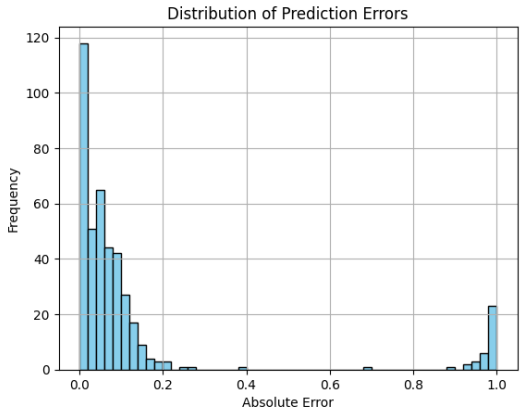

Figure 5: Distribution of Prediction Errors

which seriously limits the sample size we arrived at in the end. All of these contribute to a rather poor performance for predicting the extremes.

5.2 Discussion

Our final regression model using lyrics achieved a mean absolute error of 0.0678 on the test set, demonstrating the ability to predict continuous song similarity scores with fine granularity. This capability is significant because it enables more nuanced recommendation strategies compared to traditional binary similarity classification. Predicting a spectrum of similarity scores allows for more personalized playlist generation and recommendation diversity.

One risk in lyrics-based models is overfitting to superficial patterns, such as frequent shared vocabulary, without capturing deeper thematic or emotional similarity. Additionally, reconstructed bag-of-words lyrics may omit important context, limiting the model's ability to fully assess song meaning.

The performance of the lyrics-based model is sensitive to input noise, such as incomplete or inaccurate lyric reconstructions. Changes in vocabulary size, embedding initialization (e.g., switching from GloVe to contextual embeddings), or quality of textual input could significantly impact model performance.

For the combined model, our main contribution is a complete pipeline for how to process both the lyrics and audio features, and to combine the two in an effective way with the Siamese architecture.

Another important consideration is the data coverage trade-off introduced by the combined model. While integrating audio features adds expressive power and captures musical qualities not present

in lyrics (e.g., rhythm, instrumentation, vocal tone), the requirement for valid audio embeddings reduced the usable dataset size significantly. As discussed earlier, we only retained samples for which both lyrics and audio embeddings were available, leading to a reduced training set compared to the lyrics-only model. This exclusion—though principled—likely contributed to the lower performance at the extremes of the similarity range.

Despite these limitations, the combined model demonstrated robust error behavior in mid-range similarity scores, suggesting it captures shared structure and trends across modalities. This is encouraging for real-world recommendation systems where mid-range similarities (i.e., "somewhat similar") are more useful than extreme cases for playlist diversification.

Moreover, our results reaffirm that content-based features, even in isolation from user behavior data, can drive meaningful similarity estimation. In particular, our design showcases how lyrical semantics and audio timbre can be brought together in a unified embedding space, opening the door to future multimodal retrieval systems that are less dependent on historical co-listening patterns.

Lastly, our project underlines the practical engineering challenges in multi-source dataset alignment. Coordinating the Million Song Dataset, LastFM similarity labels, musiXmatch BOW lyrics, and audio embeddings required substantial preprocessing, filtering, and balancing to ensure quality input to the models. This highlights that, beyond model architecture, careful data engineering is a key determinant of deep learning performance, particularly in recommendation settings where missing or noisy data is common.

6 Future directions

Several areas remain open for further improvement and exploration.

- **Full Multi-Modal Integration:** Future work will focus on building and evaluating a unified model that combines lyrics and audio representations. Integrating semantic and acoustic features has the potential to significantly improve similarity predictions, especially for genres where lyrical content and musical style diverge.
- **Improved Lyrics Representations:** Our current lyrics inputs are reconstructed from bag-of-words (BOW) vectors, which omit important word order and context. Future experiments could use full-text lyrics, when available, or leverage pre-trained contextual embeddings (e.g., BERT, RoBERTa) to better capture semantic and emotional content.
- **Robustness to Input Quality Variations:** Our model's sensitivity to incomplete or noisy lyrics and audio suggests the need for explicit robustness techniques. Future work could include data augmentation, noise injection during training, and ensemble methods to stabilize predictions across varying input quality levels.
- **Fine-Grained Similarity Evaluation:** While MAE and RMSE provide aggregate error measures, future evaluation should include ranking-based metrics (e.g., Spearman correlation, nDCG) to better capture how well predicted similarity scores preserve the relative ordering of similar songs.
- **Scalability and Deployment Considerations:** As models grow to include multi-modal inputs, training and inference costs may become prohibitive for large-scale recommendation systems. Future work should explore model distillation, quantization, or other efficiency techniques to ensure practical deployability.

Addressing these areas would not only improve the accuracy of song similarity predictions, but also enhance model robustness, generalization, and usability in real-world music recommendation systems.

Last but not the least, for the performance limitations and data coverage trade-offs introduced earlier, we outline several key directions for improvement:

Method 1: Scaling with More Powerful Computing Resources
A primary bottleneck we encountered was the constrained working memory on Kaggle, which restricted us to loading only one or two pickle files (each roughly 4GB) at a time. This severely limited the number of training samples we could access simultaneously, reducing model robustness and generalization. Migrating our workflow to a more capable environment—such as a high-RAM

Colab Pro+ instance, an on-premise GPU server, or a cloud VM with scalable memory (e.g., AWS EC2, Google Cloud Compute)—would allow us to batch-load the full dataset and train on a much larger sample pool. This alone would improve the model's coverage and reliability, especially for rare or extreme similarity labels.

Method 2: Hard Negative Mining or Label Smoothing
Since the majority of unused pairs in the MSD are unlabeled (not explicitly dissimilar), another approach is to selectively include them using a form of weak supervision. For instance, we could apply label smoothing or assign soft similarity scores (e.g., ≈ 0.1) to pairs with no known similarity, instead of discarding them entirely. This would allow us to introduce more training pairs while acknowledging uncertainty, providing the model with more gradient signal and improving robustness to unseen or ambiguous cases. This is not accomplished by this course project due to the same computing budget issue, where we don't have the time and computation powers to re-sample through the entire database in achieving a more balanced positive and negative division.

Method 3: Curriculum Learning and Progressive Training
A final strategy is to structure training using curriculum learning—starting with high-confidence labeled examples (e.g., scores near 0 or 1), then gradually introducing noisier or weakly labeled data. This allows the model to first learn robust decision boundaries before tackling ambiguous or less certain cases, improving generalization and stability in low-data regimes. But since the label is provided as given by the LastFM, this improvement stratey may rely on some manual labeling to ensure the ground truth labels truthfully reflect the similarity, in another word, a similarity score of 1.0 should mean "a hundred percent similar", which is not the case for the MSD and LastFM.

Taken together, these improvements aim to expand dataset usability, enhance feature quality, and make training more resilient to the multimodal coverage gap that currently limits performance.

7 Conclusion

In this project, we explored content-based approaches for predicting song similarity, moving beyond traditional collaborative filtering methods. Our initial objective was to replicate the lyrics-based classification baseline from the *DeepPlaylist* paper, which we achieved by reaching approximately 80% validation accuracy using a BiLSTM model with concatenation of encoded lyrics features.

Building on this foundation, we extended the task from binary classification to regression, enabling continuous similarity prediction. Our best lyrics-based regression model achieved a test mean absolute error of 0.0678 and a root mean squared error of 0.1543, demonstrating the ability to capture nuanced relationships between songs based solely on their lyrics.

Overall, we made significant progress toward our original objective of developing a content-driven similarity model that reduces reliance on user interaction data and enables richer, more inclusive music recommendations. Future work will focus on refining multi-modal models, improving robustness to input variations, and enhancing evaluation methods to further strengthen song similarity prediction systems.

In addition to the lyrics-only model, we successfully developed a combined lyrics and audio model using a Siamese architecture. This model processes both modalities in parallel and merges their embeddings to produce a unified similarity score. Despite facing compute and memory limitations that reduced training scale, our combined model showed promising results—particularly in mid-range similarity predictions—highlighting the value of multimodal inputs. We also contributed a complete pipeline for integrating and aligning heterogeneous datasets (lyrics, audio, similarity labels), setting a foundation for future multi-source music similarity research.

8 Administrative details

8.1 Team contributions

FNU Annanya - Implementing the Lyrics model

Aimee Langevin - Implementing the Combined model

Weiqian Zhang - Dataset preparations and feature engineering

8.2 GitHub repository

GitHub Repository

References

[1] Janani Balakrishnan and Aashish Dixit. Deepplaylist: Modelling music listening sessions with recurrent neural networks. In *NIPS Workshop on Machine Learning for Creativity and Design*, 2016. URL https://cs224d.stanford.edu/reports/BalakrishnanDixit.pdf.

[2] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The last.fm dataset (part of the million song dataset). http://millionsongdataset.com/lastfm/, . Accessed: 2025-04-28.

[3] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The musixmatch dataset (part of the million song dataset). http://millionsongdataset.com/musixmatch/, . Accessed: 2025-04-28.

[4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Sackinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, volume 6, 1993. URL https://proceedings.neurips.cc/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf.

[5] Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Recognizing entailment and contradiction by tree-based convolution. *arXiv preprint arXiv:1512.08422*, 2015. URL http://arxiv.org/abs/1512.08422.

[6] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *arXiv preprint arXiv:1503.03832*, 2015. URL https://arxiv.org/abs/1503.03832.