Lyric Classification and Genre Style Transfer

Tiffany Chen tiffc@mit.edu
Sohini Kar skar@mit.edu
Julia Wagner jnwagner@mit.edu
Crystal Wang crystw@mit.edu

Abstract

In this paper, we apply a variety of model architectures to the problems of song lyric classification and generation. For the multi-class classification task, we use a baseline RNN model with Word2Vec embeddings and a more complex transformer-based model to classify the genre of a song based on its lyrics. The resulting models have drastically different performances and we see a significant improvement in the transformer model over the RNN model. Meanwhile, for song lyric generation, we apply techniques from other text style transfer papers in order to generate new song lyrics based on the original lyrics and the desired genre of the output. Unfortunately, while these techniques do well in terms of style transfer, they lack content retention and are not usually well-formed sentences. The results from both of these problems suggest that there are distinct semantic differences between songs in different genres that can be detected by machine learning models. [1]

1 Introduction

New techniques and advancements within the field of natural language processing (NLP), such as attention-based transformer models like Google's BERT, have been highly successful within a variety of classic NLP tasks. We apply these methods in juxtaposition with traditional machine learning architectures in order to tackle the problems of genre classification and song lyric generation.

Classification involves prediction from some set of labels for a given text input. Within the context of this paper, we will be predicting song genres given their lyrics as the input using a variety of deep learning architectures such as recurrent neural networks (RNNs) and transformers, sometimes in tandem.

Text generation is the task of outputting text that is indistinguishable from regular human-generated language. Here, we will study a subset of text generation problems called text style transfer. Text style transfer is the changing of the qualitative properties of a body of text while maintaining the original content and context. In this paper, we will apply known text style transfer techniques in order to produce new song lyrics that preserve the content of the original input lyrics while introducing the style of the input genre.

This paper is structured as follows. Section 2 discusses prior work and research within the areas of classification and text-style transfer problems. Section 3 describes the data set used to train our models and explains the steps taken to clean and process this data. Section 4 walks through the architecture of the baseline and improved classification models, as well as implementation details and evaluation results. Section 5 mirrors the layout of section 4 but addresses the generation models instead. We conclude with a discussion of conclusions and future work in section 6.

2 Previous Work

2.1 Lyrical Classification

Previous lyrical classification research focused on supervised learning techniques such as support vector machines, k-nearest neighbors, and naive Bayes. Fell and Sporleder (Fell and Sporleder, 2014) classify among 8 genres using n-grams with hand selected features to help represent vocabulary, style, structure, and semantics. Ying et al. (Ying et al., 2012) use part-of-speech tags to classify among 10 genres using SVMs, k-NN, NB with a highest accuracy of 39.94%. McKay et al. (McKay et al., 2010) use hand selected features in lyrics to produce classification accuracies of 69% amongst 5 genres and 43% amongst 10 genres. More recently, Bužić and Dobša (Bužić and Dobša, 2018) use naive Bayes to predict song performer based solely on song lyrics.

With the development of deep-learning networks such as convolutional neural networks, RNNs, and transformers, lyric classification has highly benefitted, allowing for higher accuracy with more genres. Sigtia and Dixon (Sigtia and Dixon, 2014) builds a random forest classifier on the hidden states of

[1] Code can be found at https://github.com/sohinik/6.864-lyric-analysis

a neural network, reporting an accuracy of 83% among 10 genres. Li et al. (Li et al., 2018) use a CNN to embed features, achieving an accuracy of 84% over 10 genres. Last, Yang et al. (Yang et al., 2016) uses hierarchical attention networks to classify documents; when applied to lyric classification via Tsaptsinos (Tsaptsinos), the model achieves 49% accuracy with 20 genres.

Our classification model builds on the previous work done on deep-learning models, using hidden states from an RNN to determine the genre in our baseline and building onto it with transformer hidden states.

2.2 Text Style Transfer

Methods used in the problem of style transfer generally include encoder-decoder frameworks, where the encoder maps the input into a style-independent vector representation and the decoder outputs text with the desired style inputted. Many previous solutions utilize disentanglement, which uses a discriminator to extract style features from the training text and input these features into unbiased text to include that style. Shen et al. (Shen et al., 2017) use a cross-aligned autoencoder with adversarial training to learn a shared latent content distribution and a separate latent style distribution. Hu et al. (Hu et al., 2017) propose a different solution: a neural generative model that combines variational autoencoders and holistic attribute discriminators to impose semantic structure.

Alternatively, a discriminator can remove style features from a styled input; the new style is then added in the decoding stage. John et al. (John et al., 2019) use a deterministic autoencoder and a variational autoencoder to get disentangled latent representations.

There are also various techniques that use pretrained discriminators and thus do not manipulate the latent representation of inputs. Xu et al. (Xu et al., 2018) utilize a cycled reinforcement learning method for unpaired sentiment-to-sentiment translation. Li et al. (Li et al., 2018) extract content words by deleting phrases a strong attribute value, retrieve new phrases associated with the target attribute, and use a neural model to combine these into a final output.

However, because of the difficulty of fully disentangling inputs, there are also several proposals that do not use disentanglement. Lample et al. (Lample et al., 2019) reduce text style transfer to an unsupervised machine translation problem by using denoising autoencoders (Vincent et al., 2008) and back-translation (Sennrich et al., 2016). Dai et al. (Dai et al., 2019) use transformers to train a style transfer system. Kim and Sohn (Kim and Sohn, 2020) separate the model into two tasks, sentence reconstruction using transformers and a classifier style module, to allow for variable style strength.

Our lyric style transfer model builds on the encoder-decoder architecture that previous solutions use, with the encoder acting as the discriminator.

3 Data

3.1 Dataset

We use a Kaggle dataset called "Multi-Lingual Lyrics for Genre Classification" (Bejan, 2021) which contains over 290,000 datapoints, each representing a song with metadata such as the genre, artist, language, and lyrics. Each song's genre is taken from the respective artist's most dominant genre. Although the dataset offers two files of data, "train.csv" and "test.csv", we used only "train.csv" as the test data set was missing the genre of each datapoint.

3.2 Data Cleaning

We filter out the data in the following ways:

Language

We use only English songs for the sake of simplicity and for performance purposes. Our models are unlikely to perform well when given datapoints across several languages and our pre-trained word embedding models are limited to the English vocabulary.

Features

We delete all other features except the genre and the lyrics, then filter out datapoints that are missing either the genre or the lyrics.

Genre

For each problem, we select a different subset of genres to examine. For classification, the genres are selected based on the number of datapoints within each genre. For generation, we select genres based on our classification model's ability to distinguish between them. More detail about the genre selection process is included in sections 4 and 5.

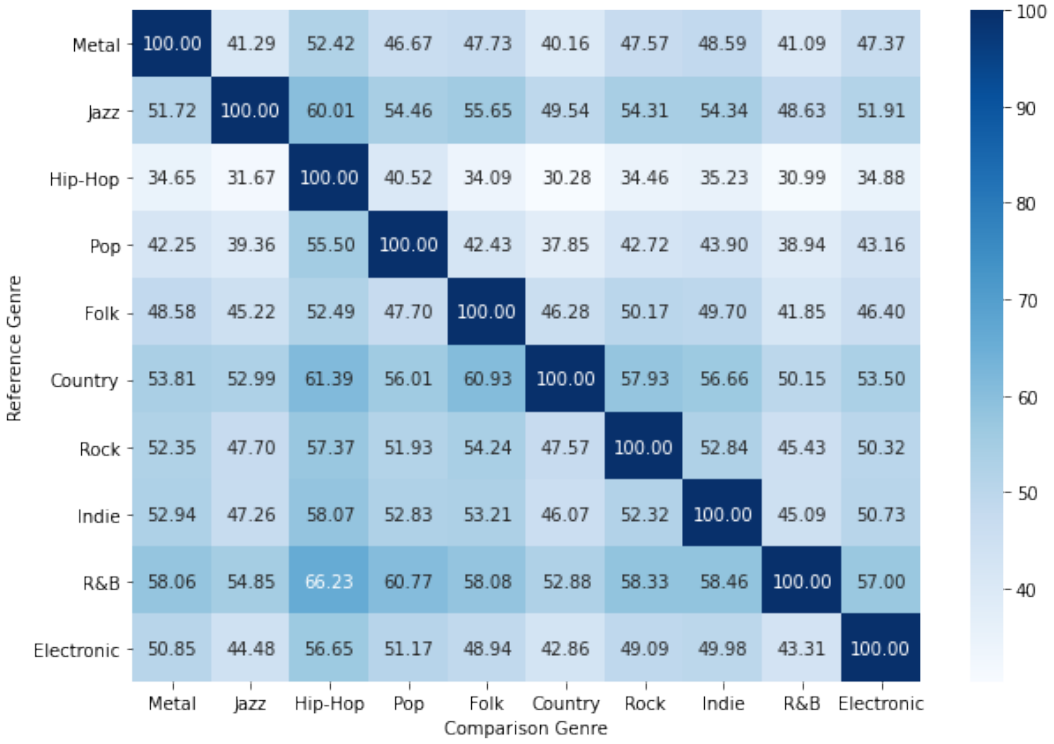

Figure 1: This matrix stores the word similarity of each pair of genres. The value at row i and column j is the percentage of words in the vocab of genre i that are shared by genre j

Lyrics

There are some datapoints where the song lyrics contain some additional text, such as guitar chords, that were probably a side effect of the data collection process for this dataset. We remove these datapoints before conducting any further data processing.

When splitting lyrics into individual words for vocabulary creation, we separate based on whitespace (e.g. a space or a newline) and we strip the ends of any of the following symbols: ( ) , . / — { } – _ ‘ ’ ” :; ?!#@$%& ∗ .

3.3 Data Processing

In addition to data cleaning, we also process the data in the following ways:

Training and Testing Split

After cleaning our data, we randomly split the data into training and testing at a ratio of 4:1 (i.e. 80% of the data is training data and the remaining 20% is testing data).

Classification Song Separation

For the classification task, we split each song up into ”stanzas” of 150 words per stanza in order to address two main problems: vanishing gradients and token limits. By dividing each song into more datapoints, we could avoid the problem of vanishing gradients that occurs when long sequences are fed in RNNs and we can circumvent the token limit for DistilBERT, a transformer we use in the classification problem.

A potential problem with splitting a song into more datapoints arises when these datapoints are in both the training and testing sets. If there are stanzas from the same song in both datasets, our models could end up performing well because it recognizes words and themes in a specific song instead of the overall lyrical style of the genre. Therefore, we perform the song splits after separating the data into training and testing data.

Generation Song Separation

For the generation task, we can no longer split each song into 150 word excerpts because we need to maintain the grammatical accuracy and sentence structure of our input in order to achieve accurate outputs. In addition, we no longer tokenize our inputs for DistilBERT, since the token limit prevents us from properly generating longer sentences.

Therefore, we separate each song based on newline characters in order to divide the lyrics into individual lines, removing duplicates. We then treat

Genre Distribution	
Genre	Datapoints
Rock	107145
Pop	86298
Metal	19133
Jazz	13314
Folk	8169
Indie	7240
R&B	2765
Hip-Hop	2238
Electronic	2005
Country	1890
Total	250197

Table 1: Genre distribution for all datapoints in our original data set

each line as a datapoint and we also "translate" future inputs line by line and construct the final output by appending the line outputs. As a side effect of this new splitting policy, we are limited in the genres that are available to us since some genres do not have songs that are newline-delimited. Our final list of available genres is as follows: folk, jazz, metal, pop, and rock.

Genre Uniformity

The final data processing involves making the genre distribution uniform within the training and testing datasets. We do so in order to prevent the models from learning the frequency of each genre and having an incorrectly high accuracy as a result. A side effect of this limitation and the stanza separation is that the resulting training and testing data sets will not have an exact ratio of 4:1 but the actual ratios are relatively close at 3.92:1 and 4.04:1 for the classification models.

3.4 Data Analysis

In this subsection, we'll analyze our data set distribution and some of the features of each genre.

Table 1 shows the distribution of datapoints across the genres after data cleaning, sorted from most datapoints to least. We can see that the most common genre is rock and the least common is country, but we can increase the number of datapoints (and avoid vanishing gradients) by splitting each song up into multiple datapoints as mentioned before.

Figure 1 demonstrates the vocabulary similarity across genres. The value of the similarity matrix at row i and column j is calculated using the following formula

$$\text{Similarity}_{ij} = \frac{|S_i \cap S_j|}{|S_i|}$$

where S_i is the set of words used in songs belonging to genre i.

As we can see from the figure, the row of similarity values for hip-hop is the lightest in color, demonstrating that hip-hop's vocabulary has the highest percentage of unique words even compared across all other genres. Meanwhile, the highest vocabulary similarity (excluding the values along the diagonal that represent trivial 100% self-similarity) occurs between R&B and hip-hop, where R&B has the lowest percentage of unique words compared against hip-hop. We therefore leave out R&B from our classification models and instead select the genres with the lowest vocabulary similarity.

4 Classification Models

We construct two main classification models: a simple model using RNNs to serve as a baseline and a more complex transformer-based model with several layers. For this problem, we focus on the following six genres: metal, jazz, hip-hop, pop, folk, and country. These genres were selected because they either had the most datapoints in our cleaned dataset or they were empirically known to have lyrical differences.

4.1 Baseline Model

4.1.1 Architecture

To represent the words in our text, we use Google's pretrained Word2Vec representations, pretrained on over 100 million words in the Google News dataset. The model contains a continuous bag-of-words of 30 million words as 300-dimensional vectors.

In our baseline model, seen in figure 2, we define a GRU with the hidden size being the number of genres we want to classify (6). This GRU acts as an encoder that transforms our text into a vector representation, where each number in the vector is a logit for the likelihood that the inputted text is the corresponding genre. We use two layers and bidirectionality to create a fuller picture of our input.

4.1.2 Implementation Details

Table 2 shows the genre distribution in our training and testing data. We can see from the table that

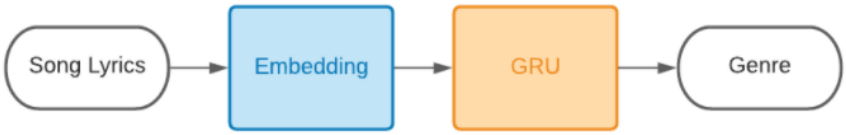

Figure 2: Architecture for the baseline classification model

Baseline Model		
Genre	Train	Test
Country	2625	650
Hip-Hop	2625	650
Pop	2625	650
Jazz	2625	650
Metal	2625	650
Folk	2625	650
Total	15750	3900

Table 2: Genre distribution for training and testing data for the baseline classification model

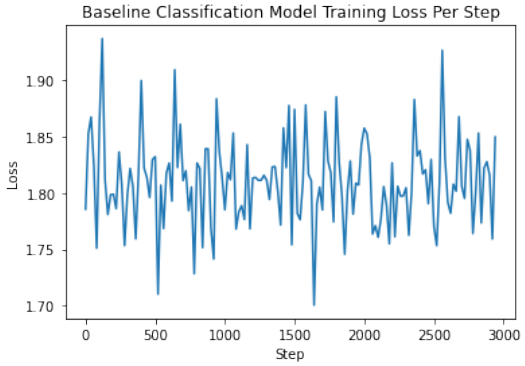

Figure 3: Training loss per step for the baseline classification model

the ratio of training to testing data is $4.04 : 1$ so we know that the stanza separation step did not significantly alter the data split.

Figure 3 shows the training loss over the 2955 steps, each with a batch size of 16, covering 3 epochs. We use an Adam optimizer with a weight decay of $1e-5$, and epsilon of $1e-6$ and a warmup_rate of 0.05.

To calculate loss, we convert every label to its corresponding numerical ID and then use cross-entropy loss in order to softmax the output logits and compare it against the correct label ID.

As we can tell from the training loss, the model is not improving over time even on the training data set.

4.2 Transformer Model

4.2.1 Architecture

To represent the words in our text, we use DistilBERT's pretrained tokenizer. Additionally, we use DistilBERT's pretrained model as our language model. These are trained on the cased English language and then distilled from Google's BERT model. The model has 6 layers and a hidden size of 768 by default.

In our model, we put our inputs through the pretrained DistilBERT model, which gets us the hidden states of each token in our input. We take the last hidden state to maximize the amount of context in our hidden states. We then put the last hidden state through a dropout layer to reduce overfitting in our model.

With this, we have the hidden states of all words in our inputs (i.e. `seq_len x hidden_size`). We want to flatten this into a hidden encoding of the sentence (i.e. `1 x hidden_size`). To do this, we put our hidden states into a GRU that acts as an encoder. We use two layers and bidirectionality to get a fuller picture of our input.

Lastly, we put our encoded input into a linear layer that maps the encoding into a vector with the length being the number of genres we want to classify (6). This outputted vector represents the logits of the input being that genre.

4.2.2 Implementation Details

Table 3 shows the genre distribution in our training and testing data. We can see from the table that the ratio of training to testing data is $3.92 : 1$ so we know that the stanza separation step did not significantly alter the data split.

Due to the nature of our bidirectional, bilayered GRU encoder, we get hidden states of dimension

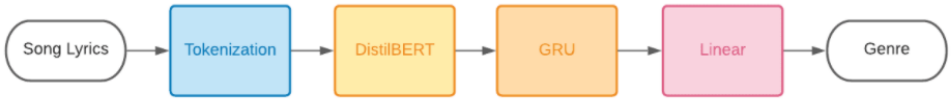

Figure 4: Architecture for the transformer classification model

Transformer Model		
Genre	Train	Test
Country	2609	666
Hip-Hop	2609	666
Pop	2609	666
Jazz	2609	666
Metal	2609	666
Folk	2609	666
Total	15654	3996

Table 3: Genre distribution for training and testing data for our transformer classification model

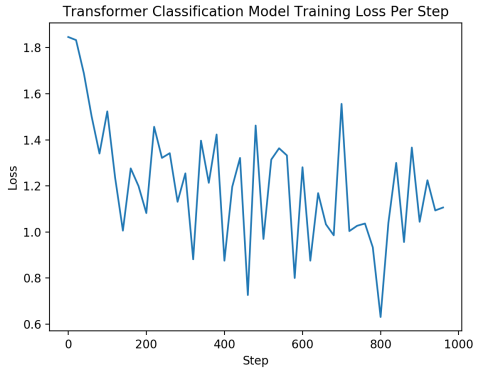

Figure 5: Training loss per step for the transformer classification model

(`2 * 2, batch_size, 768`). Since we only care about the hidden state of the last layer, we grab the 2 hidden states from the two final layers (one for each direction) and we sum them to get an aggregate hidden state. We then pass this into our linear layer to generate our final logits.

Figure 5 shows the training loss over the 979 steps, each with a batch size of 16, covering 1 epoch. We use an Adam optimizer with a weight decay of $1e-5$, and epsilon of $1e-6$ and a warmup_rate of 0.05. We also use a learning rate scheduler from the transformers library.

To calculate loss here, we also convert every label to its corresponding numerical ID and then use cross-entropy loss in order to softmax the output logits and compare it against the correct label ID.

As we can tell from the training loss, the model is improving over time and we can attribute some of the volatility to the small batch size.

4.3 Results

We evaluate the success of our model through accuracy, precision, recall, and F1 scores.

Accuracy is the percentage of test inputs that were correctly labelled or, in other words, the number of correctly labelled inputs summed across all genres divided by the total number of inputs. On our confusion matrix, accuracy is the sum of the numbers on the diagonal divided by the sum of every number in the matrix.

$$\text{Accuracy} = \frac{\sum_{\text{genre}} \text{\# of true positives in genre}}{\text{\# of datapoints}}$$

Precision is a metric that is different for each genre, and it is the percentage of the songs predicted to be in that genre that were actually in that genre. On our confusion matrix, precision is the number in each column of the diagonal divided by the sum of every number in that column.

$$\text{Precision}_{\text{genre}} = \frac{\text{\# of true positives in genre}}{\text{\# of datapoints in the genre}}$$

Recall is also a metric that is different for each genre, and it is the percentage of songs in the genre that were predicted to be in that genre. On our confusion matrix, recall is the number in each row of the diagonal divided by the sum of every number in that row.

$$\text{Recall}_{\text{genre}} = \frac{\text{\# of true positives in genre}}{\text{\# of datapoints predicted in genre}}$$

Finally, F1 scores are the harmonic mean of the precision and the recall.

$$\text{F1}_{\text{genre}} = 2 * \frac{\text{Recall}_{\text{genre}} * \text{Precision}_{\text{genre}}}{\text{Recall}_{\text{genre}} + \text{Precision}_{\text{genre}}}$$

4.3.1 Baseline Model Analysis

Table 4 shows the accuracies of the baseline model. Our baseline model's accuracy is not better than randomly assigning genres and thus it fails at classifying lyrics.

In fact, since the distribution across predicted genres, shown in Figure 6, is the same in every row, it appears that the genre of the input has little to no impact on the predicted genre.

The model may then be making its predictions for each genre based on a feature that is uniformly random across all genres. For example, one possible feature could be that the model predicts "country" when the length of the sequence is less than 20 words, but this is unlikely to be correlated with the genre and rather is dependent on the input.

We also notice that, despite being given the same number of genres in the training data, the model disproportionately predicts the genres folk, hip-hop, and jazz. It is possible that the last few inputs received by the model were disproportionately in those three genres and therefore the model was significantly altered in the last few batches.

Another explanation could be that the testing data set is skewed towards the features that the RNN has learned. However, repeated training over several different randomized splits of training/testing data demonstrates that the model continues to disproportionately guess a handful of genres over other genres. Since repeated testing should decrease the probability of a bad train/test split, we conclude that this explanation is unlikely to be true.

Additionally, Table 5 shows the precision, recall, and F1 scores for the baseline classification model. Note that the recall for each genre (and therefore the F1 score) is positively correlated with the likelihood that the model predicts said genre (i.e. the darker the corresponding column in the confusion matrix is, the higher the recall and the F1 score are). Since the recall is the number of true positives divided by the total number of data points in that genre, if the model is disproportionally predicting a certain genre uniformly, then the number in the diagonal section would be larger compared to everywhere else in the row.

Classification Accuracies

Model	Accuracy
Baseline	15.64%
Transformer	56.56%

Table 4: Accuracy for the baseline and transformer classification models

Baseline Classification Model

Genre	Precision	Recall	F1
Country	0.115385	0.041538	0.0611
Folk	0.146259	0.198462	0.1684
Hip-Hop	0.179666	0.198462	0.1886
Jazz	0.163324	0.263077	0.2015
Metal	0.170686	0.141538	0.1548
Pop	0.129167	0.095385	0.1097

Table 5: Precision, recall, and f1 scores for the baseline classification model

4.3.2 Transformer Model Analysis

Table 4 shows the accuracies of the transformer model. Our transformer model's accuracy is significantly better than randomly assigning genres, and we can see a clear diagonal on the confusion matrix in Figure 7. Thus, our transformer model was moderately successful at lyric classification.

Using transformers aided in our model's success due to its self-attention layer, allowing our model to focus on words that best indicate genre. Specifically, using the pretrained DistilBERT model gives additional context to the lyrics, with pre-trained attention giving focus to words that are most likely to be genre indicators.

From our confusion matrix, we can see that folk is often confused for country. In contrast, country is not confused as folk nearly as often, although folk is the genre that country is misclassified as the most. This is perhaps explained by country music's roots, which originated partially from various American folk music. However, country has since evolved to include more regional cues from the southern U.S., which folk does not have. (Malone, 2003)

As a result, when these cues appear in lyrics (as they usually do for country music), it is easy to identify the genre as country but when they are absent (as they usually are in folk music and sometimes are in country), it becomes harder to differentiate between country and folk. We offer this hypothesis as a potential explanation for why folk is confused often for country but not vice versa.

To explore the validity of this hypothesis, we

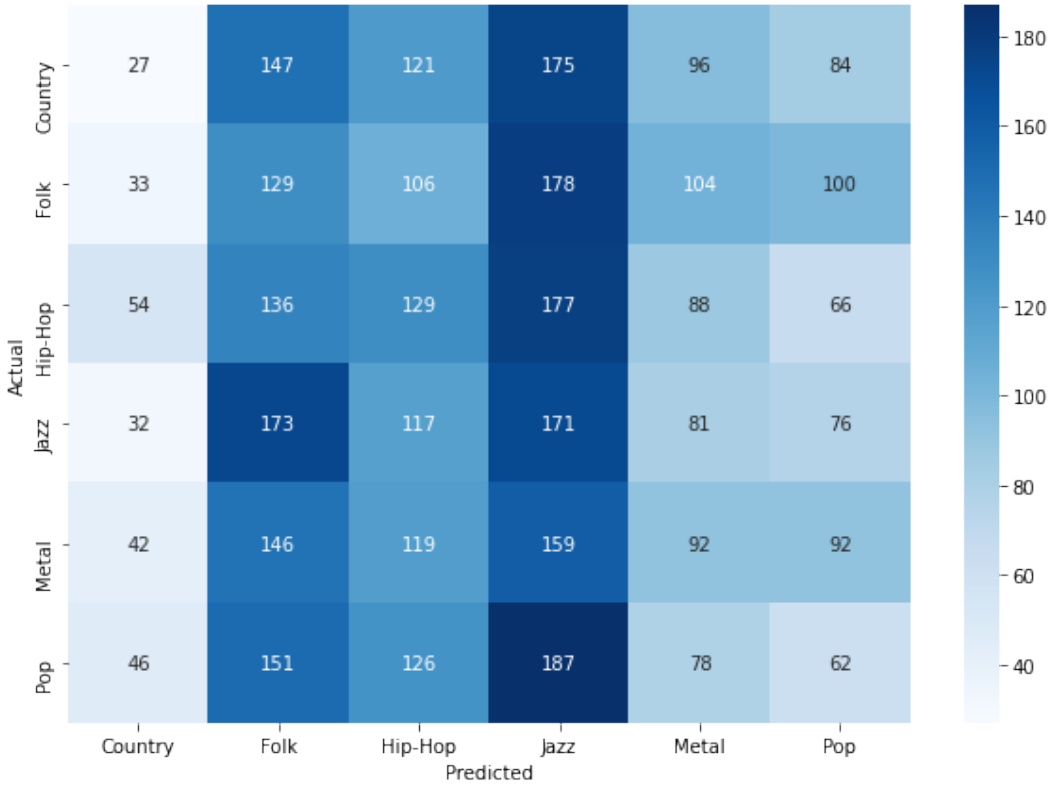

Figure 6: Confusion matrix for the baseline classification model

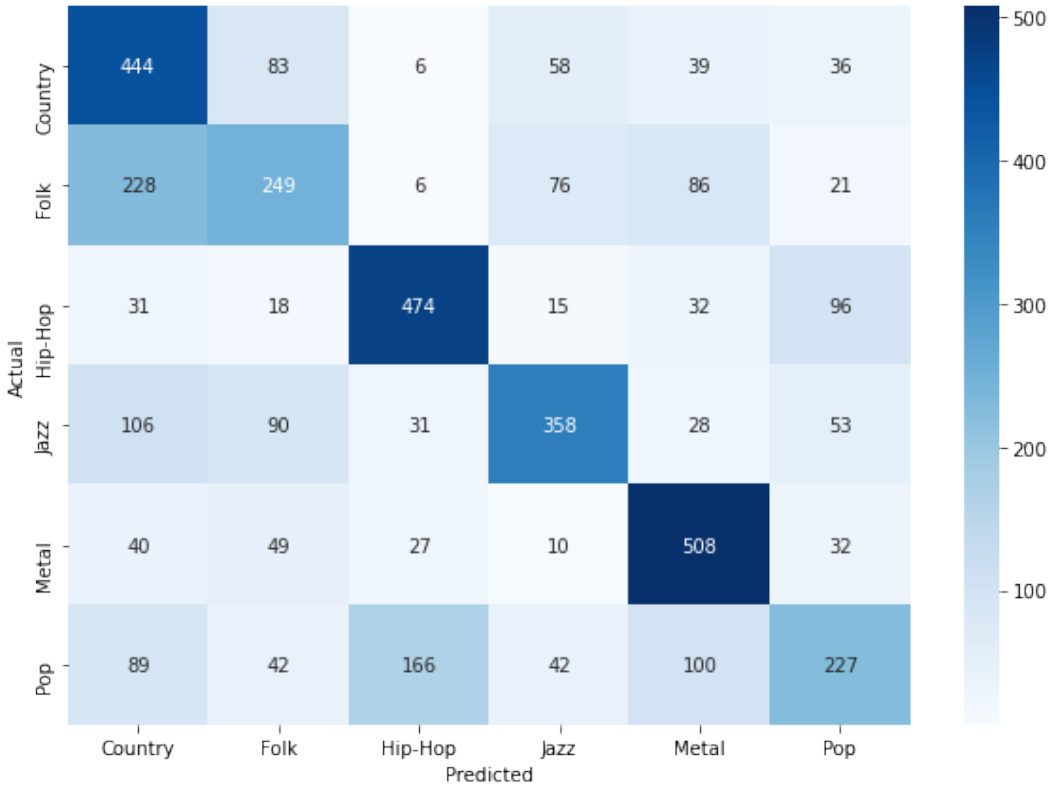

Figure 7: Confusion matrix for the transformer classification model

analyze the vocabulary of the folk song lyrics that were misclassified as country songs by calculating their similarity (as defined in section 3.4) with actual country songs. The calculated similarity value is 87.1%, meaning about 87% of the words in the misclassified folk songs appear in real country songs as well. This is significantly higher than the similarity for all folk against all country (instead of just misclassified folk against all country), which is 46.3% as seen in figure 1.

In contrast, the similarity value with country songs as the denominator was vastly smaller than normal, at 17.0%. This tells us that while the misclassified folk songs share most of their vocabulary with country songs, country songs have a much larger vocabulary that is not dominated by the intersection with those folk songs. Therefore, our hypothesis appears to hold up under similarity analysis.

Similarly, pop is often confused for hip-hop. In contrast, hip-hop is not confused as pop nearly as often, although pop is the genre that hip-hop is misclassified as the most. A potential reason for this is because of hip-hop's influence on popular music, as hip-hop is one of the most popular genres in modern American music. (Rojek, 2011) Another related phenomenon is hip-hop's unique vocabulary that makes it easily differentiable from other genres, as we saw in section 3.4.

To continue on this point, pop is the most misclassified genre; it has the lowest F1 score, as shown in Table 6. There are several potential reasons that this happens. First, pop often borrows features from other genres. Additionally, pop's largest identifying feature is catchy musical hooks, which isn't often captured in the text; adding some form of audio features to our classification model may increase pop's recall and precision. (Rojek, 2011)

Meanwhile, hip-hop and metal have the highest F1 scores, all above 0.68. As stated before, hip-hop has a highly unique vocabulary, often using slang and shortened words, and is easier to differentiate via text-based analysis as a result. Similarly, metal typically focuses on darker lyrical theme, especially compared to the other genres we included in our classification model, and thus the model most likely distinguishes metal with works with darker subjects (Kahn-Harris, 2006).

Generally, we observe that the transformer model performed much better than the baseline

Transformer Classification Model

Genre	Precision	Recall	F1
Country	0.473348	0.666667	0.5536
Folk	0.468927	0.373874	0.4160
Hip-Hop	0.667606	0.711712	0.6890
Jazz	0.640429	0.537538	0.5845
Metal	0.640605	0.762763	0.6964
Pop	0.488172	0.340841	0.4014

Table 6: Precision, recall, and f1 scores for the transformer classification model

Generation Models

Genre	Train	Test
Jazz	12000	3000
Metal	12000	3000
Total	24000	6000

Table 7: Genre distribution for training and testing data for our generation models

model and it achieved its best scores on highly specialized or differentiated genres like country, hip-hop, and metal. More specifically, genres with common lyrical themes, such as the ones aforementioned, proved to be the easiest for the model to classify. Meanwhile, those that span a wide range of potential topics, such as pop, or that share roots with other genres, such as folk, provide more of a challenge for our model.

5 Generation Models

We construct two main generation models: a set of autoencoder models using GRUs to serve as a baseline and a more complex many-to-many translation model with an encoder-decoder scheme. For this problem, we focus on the following genres: metal and jazz. These genres were selected because of their success in the classification model.

Table 7 shows the genre distribution in our training and testing data. We are capped at around 15,000 data points due to RAM capacities. Thus, we choose to have 12,000 training data points and 3,000 testing data points, to maintain a 4:1 ratio for training to testing data.

5.1 Autoencoder Model

5.1.1 Architecture

For each genre, we train an autoencoder. These autoencoders build our encoders and decoders for prediction, where we take the encoder of our input

genre and the decoder of the desired genre to get our output, as shown in figure 8.

To represent the words in each genre, we use an embedding layer that we train ourselves. We use a different embedding layer for the encoder and decoder; thus, altogether, each genre model has 2 embedding layers.

When training our models, we put our given input through the encoding embedding layer, which gets us the embeddings for each word. Then, we put this embedding input through our encoder, which is a bidirectional bilayered GRU. This gives us encoded outputs and hidden states, which we pass along with the decoder embedded input to our decoder. The decoder, with teacher forcing, goes through each word in the given input. First, it puts the word input and the encoder states through our attention model, based on Bahdanau attention, to get the attention context and probabilities. With this, we input the attention context and the previous embedding into a bidirectional bilayered GRU, giving us an output and a hidden state. We get the pre_output by combining the GRU output, previous embedding, and the attention context into a dropout layer (to prevent overfitting) and a linear layer, mapping it to a hidden state. With these concatenated hidden states (one for each word), we put them through a generator, which gets the logits for each word being at that spot by putting the hidden states through a linear layer, expanding the states to the vocab size, and a softmax layer.

To text-style transfer on these models, we retrieve the encoder and its corresponding embedding layer from the input genre and the decoder, its corresponding embedding layer, and the generator from the output genre. We then run the same steps as with training.

5.1.2 Implementation Details

When getting the vocabulary for our genres, we take the 7000 most frequent words to prevent GRU memory issues.

Due to the nature of our bidirectional, bilayered GRU encoder, we get hidden states of dimension (`2 * 2, batch_size, hidden_size`). Since we only care about the hidden state of the last layer, we concatenate the 2 hidden states from the two final layers (one for each direction), resulting in a dimension of (`2 * 2, batch_size, hidden_size`). We then pass them through a linear layer bridge to get a final encoder hidden state of dimension (`2 * 2, batch_size, hidden_size`).

In the decoder, after retrieving the outputs from the GRU, we concatenate the outputs with the attention context and the previous word embedding before passing that through a linear layer, to take into account the most important information to get the logits.

To calculate loss here, we compare the original input to the output using negative log likelihood loss. Figures 12 and 10 show the training loss over steps for the autoencoders, each with a batch size of 64, covering 10 epochs. We use an Adam optimizer with a learning rate of $1e-3$. As we can see from the training losses, the models are improving over time.

When decoding our logits, we use a greedy decode to take the most likely word for each timestep until an EOS token is reached. We use greedy decode for memory purposes.

5.2 Genre-to-Genre Model

5.2.1 Architecture

For this model, we train a many-to-many translation model with several linear layers, encoders, and decoders, one for each genre. This is shown in figure 11.

Unlike our previous models, we pass in our lyrics as one-hot vectors instead of tokens, to allow our model to predict based on word logits. Thus, to embed the words in each genre, we use a linear layer mapping from vocab_size to embed_size. We use a different linear embedding layer for the encoder and decoder; thus, altogether, each genre has 2 linear embedding layers.

To use this model, we put our given input through the encoding linear embedding layer for our input genre, which gets us the embeddings for each word. Then, we put this embedded input through our input genre encoder, which is a bidirectional bilayered GRU. This gives us encoded outputs and hidden states.

Then, we pass along the encoder states to the output genre decoder, along with the output genre linearly embedded decoder input. The decoder, with teacher forcing, goes through each word in the given input. First, it puts the word input and the encoder states through our attention model, based on Bahdanau attention, to get the attention context and probabilities. With this, we input the attention context and the previous embedding into a bidirectional bilayered GRU, giving us an output and

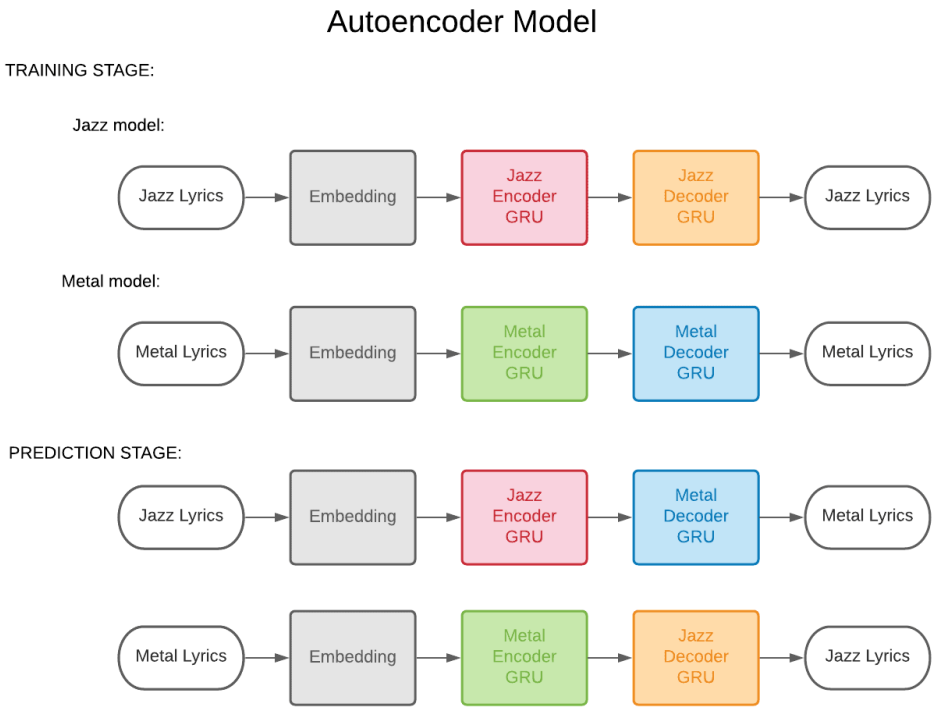

Figure 8: Architecture for the autoencoder generation model

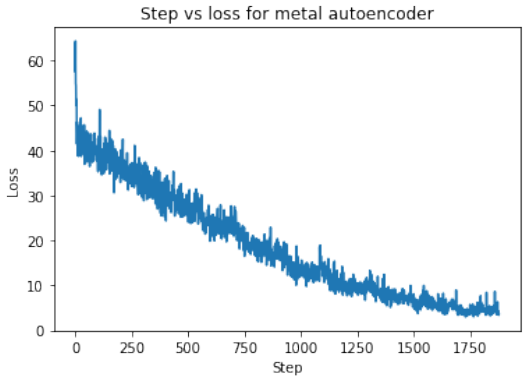

Figure 9: Training loss per step for the metal autoencoder generation model

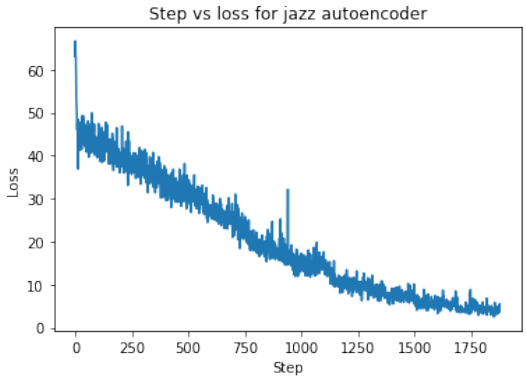

Figure 10: Training loss per step for the jazz autoencoder generation model

a hidden state. We get the pre_output by combining the GRU output, previous embedding, and the attention context into a dropout layer (to prevent overfitting) and a linear layer, mapping it to a hidden state.

With these concatenated hidden states (one for each word), we put them through a generator, which gets the logits for each word being at that spot by putting the hidden states through a linear layer, expanding the states to the vocab size, and a softmax layer.

5.2.2 Implementation Details

When getting the vocabulary for our genres, we take the 7000 most frequent words to prevent GRU memory issues. We choose to share the vocabulary between metal and folk songs to have logits for each word correspond to the same word, allowing us to calculate cycle loss (discussed below).

Much of the implementation details for the autoencoder discussed in section 5.1.2 applies here, particularly the discussion of the encoder-decoder architecture.

For this model, we use two loss functions. Our first loss function is self loss. This is when we put the input through the model with the same source and target genre before comparing the original input to the output using negative log likelihood loss, like for our autoencoders. Our second loss is cycle loss. For this loss, we define an intermediate genre that we translate the input to. Once we get that output, we put it through the model again with our target genre being the original input genre. We then compare the original input to the final output using negative log likelihood loss. These both focus on preserving semantic meaning when translating text into different genres, with a hope that the self loss introduces some innate decoding that will correspond to style. They would be strengthened by the third type of loss discussed in future steps.

Figure 12 shows the training loss over steps for the genre-to-genre model, each with a batch size of 32, covering 5 epochs. We use an Adam optimizer with a learning rate of $1e-3$. For each epoch, we train on the jazz dataset before the metal dataset. Thus the spikes in the graph are when we switch the dataset we are using. Still, as we can see from the training loss, the model is improving over time.

Like our autoencoder, when decoding our logits, we use a greedy decode to take the most likely word for each timestep until an EOS token is reached. In this greedy decode, we also input the source and target genre to make sure our model uses the correct encoder, decoder, and linear layers.

Autoencoder BLEU Scores

Model	BLEU Score
Jazz-to-Jazz	54.10
Metal-to-Metal	46.92
Jazz-to-Metal-to-Jazz	2.97
Metal-to-Jazz-to-Metal	3.23

Table 8: BLEU scores for the autoencoder models and encoder-decoder models after training

5.3 Results

For text style transfer, our models should be able to hit two main goals: content retention and style transfer. In the following analyses, we discuss the performance of our model within these two objectives.

5.3.1 Autoencoder Model

Content Retention

Table 8 lists the BLEU scores for a variety of models. The first and second rows demonstrate the BLEU scores for our jazz and metal autoencoders, respectively. We can see that the autoencoders perform very well after training and are capable of reconstructing the original sentence very well. This is also shown in Table 9, where we show sample outputs from our autoencoder models. For the autoencoder tasks, the input and output are very similar, with differing words still being close in sentiment.

The third row demonstrates the BLEU scores for chained jazz-to-metal and metal-to-jazz encoder-decoders constructed by passing jazz lyrics through the jazz encoder, then the metal decoder, then the metal encoder, and finally the jazz decoder. In doing so, we aim to replicate the original jazz lyric input as our final output. However, we can see that the BLEU score falls drastically to 4.81, indicating that this cyclic model performs very poorly. Similarly, the fourth row represents passing metal lyrics through the metal encoder, then the jazz decoder, then the jazz encoder, and finally the metal decoder. We can see similarly poor results for this cycle. This is potentially because the latent states from the two encoders are very different and thus result in drastically different outputs when combining encoders and decoders from different autoencoders.

We use the BLEU scores here in order to measure content retention, since BLEU scores measure

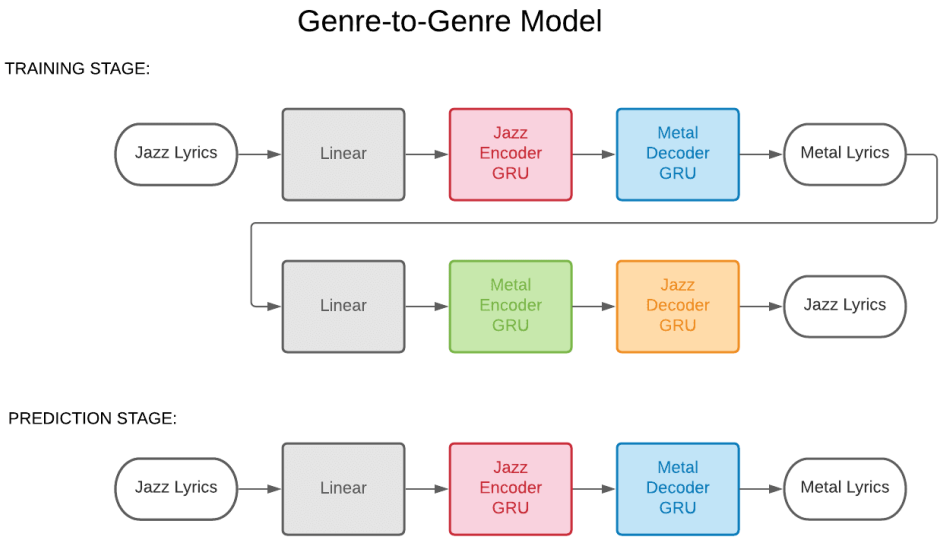

Figure 11: Architecture for the genre2genre generation model showing both cyclic loss (self loss is the same as autoencoder training) and the prediction for jazz to metal (metal to jazz not shown but is symmetrical)

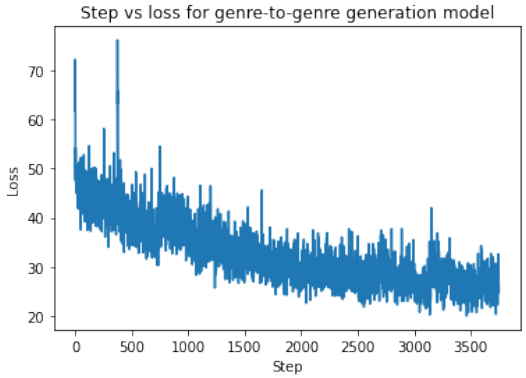

Figure 12: Training loss per step for the genre-to-genre generation model

the similarity of a predicted text against a reference text, and we pass in the reconstructed lyrics to compare with the original lyrics. The original autoencoders are able to preserve the content relatively well, but the chained encoder-decoders are unable to do the same.

Style Transfer

When evaluating style transfer, we feed the generated lyrics through to our transformer classification model to get the predicted genre. Figure 13 represents a confusion matrix demonstrating the results when taking argmax and allowing for all of the genres the classification model has been trained on. Figure 14 represents a confusion matrix demonstrating the results when taking argmax only over jazz and metal genres.

We can see from the general argmax confusion matrix that the jazz-to-metal encoder-decoder model is very successful at transferring over the style to metal, while the metal-to-jazz encoder-decoder produces lyrics that are more commonly mistaken as country or folk songs leading to an accuracy of 26.6%. When we limit the genres to just jazz and metal, the model performs much better and achieves an improved accuracy of 70.0%.

However, judging from the results shown in Table 9, we hypothesize that the high accuracy stems from the fact that the metal decoder outputs words from the metal vocabulary. As we saw from the classification task, metal songs tend to have a dark theme that makes it easy for our model to detect the genre, and since the decoder is limited to vocabulary from metal songs, its outputs score highly with the classification model.

5.3.2 Genre-to-Genre Model

Content Retention

Table 10 lists the BLEU scores computed for our genre-to-genre model, calculated in the same way as described in Section 5.3.1. Through the BLEU scores, we can see that the model does not do well in maintaining text style or applying text style transfer.

Through experimentation, we derived that the failure in our model most likely comes from us us-

Autoencoder Case Study

Jazz to Jazz

Input	Output
they got a crazy way of loving there	they got a crazy way of loving there
now ain't it peculiar that she's finally <unk> your	now ain't it couldn't that she's right <unk>

Metal to Metal

Input	Output
but she chose to push them away	but she chose to push them away
the right drug wrong time he'll be remembered	the right drug wrong time he'll be help

Jazz to Metal

Input	Output
and what we have is much more than they could see	one again through black every you loved every else
i may dream a million dreams	the breaking that i falling see what's

Metal to Jazz

Input	Output
my iron eyes tell the tale	street hold there do knows
be careful what you wish for when you dream	all rock so know small it your

Table 9: Examples of outputs produced by the autoencoder models

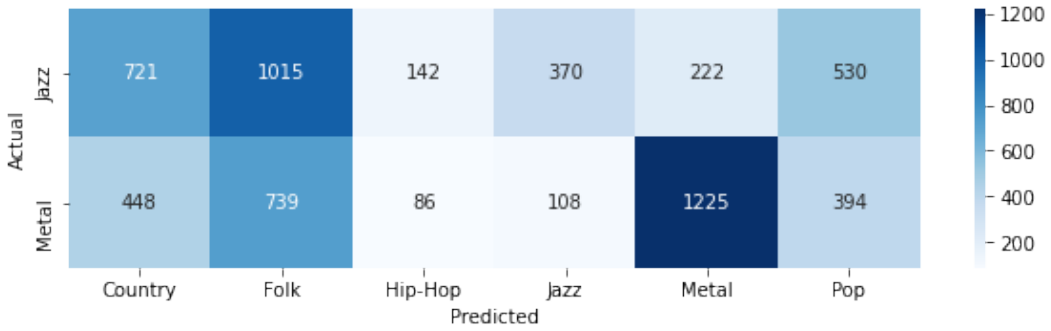

Figure 13: Confusion matrix for generated jazz and metal lyrics with general argmax

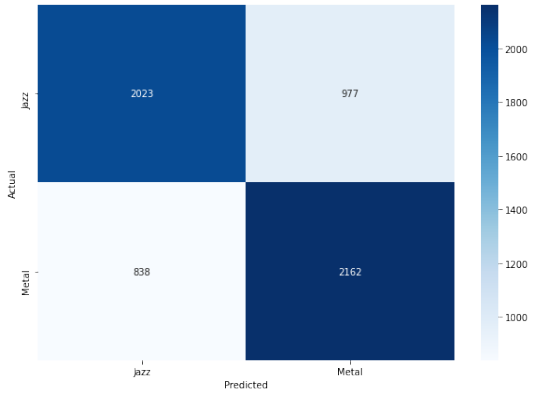

Figure 14: Confusion matrix for generated jazz and metal lyrics with selective argmax

Genre-to-Genre BLEU Scores

Model	BLEU Score
Jazz-to-Jazz	0.155
Metal-to-Metal	0.168
Jazz-to-Metal-to-Jazz	0.103
Metal-to-Jazz-to-Metal	0.101

Table 10: BLEU scores for the genre-to-genre model and encoder-decoder models after training

ing linear layers to embed the text which was necessary to implement cycle loss as the back propagation had to be compatible with our greedy decode. We did this by slowly building our genre-to-genre model starting with our autoencoder. This is likely because the `nn.Embedding` layer is a hash table, so it only backward-propagates on the words found in each input. In contrast, `nn.Linear` backpropagates on all words, which makes it hard to learn embeddings for less common words due to their absence in the majority of songs.

Table 11 shows examples of inputs and outputs that our model produces. Note that <unk>s represent words that are not part of our vocab size.

When we treat our genre-to-genre model as an autoencoder, words are repeated, but the word chosen to be repeated tends to keep the sentiment of the input. This is most clear with "feel my anguish," which results in "burn" being repeated. This shows moderate success in our model understanding sentiments of the lyric.

Style Transfer

Table 11 shows examples of inputs and outputs that our model produces. When attempting to text-style transfer with our model, our outputs are mostly useless, only repeating common words. We hypothesize that this is because of us using linear layers as embeddings without a much larger dataset. As described above, using a linear layer trains our model on every word regardless of its usage in the input while using an embedding layer only trains on words if they are used. Thus it is likely that our model simply predicts common words when text-style transferring, as these are the most likely words to appear in songs.

In addition to the logit input type being necessary for the cycle loss, it is necessary for style loss. Style loss is the loss taken from putting lyric logits through a source genre encoder, a target genre decoder and generator, greedily decoding, classifying using our classification model, and using cross entropy loss to compare predictions to the target genre. Currently, our self and cycle loss could be satisfied by simply copying input to output. Style loss forces our model to take text style into account. However, for back propagation reasons, it requires the input of the *classification* model to be logits. Additionally, since we implemented and trained our classifier with a specific tokenizer and embedding, we could not smoothly integrate this third type of loss. We would like to explore this loss function along with more computing capacity to hopefully result in a more well-rounded model with respect to our objectives.

6 Conclusion and Future Work

In this paper, we aim to solve the problems of song lyric classification and generation. For classification, we develop two models: one with an RNN as its main architecture and one with a transformer as its main architecture. While the former did not show any success, the latter was successful in our experiments in classifying lyrics into genres, showing that there are semantic differences detectable by deep-learning models.

For generation, we develop two models: one based on autoencoders and one based on a many-to-many translation model architecture. In our experiments, our autoencoder showed moderate success in text-style transfer, as our transformer classifier was able to classify our generated metal lyrics as such. However, our genre-to-genre model was not successful, due to our inability to properly embed the inputs and train our model on style and semantics.

In the future, we plan to make our genre-to-genre model better by implementing the style loss, to al-

Genre-to-Genre Case Study

Jazz to Jazz

Input	Output
and this this bitter	and tears what's what's what's what's
church bells <unk> on a sunday morn	dining divine divine divine divine divine

Metal to Metal

Input	Output
i'm grasping harder with every breath	i'm i'd as as lonely lonely lonely lonely
feel my anguish	feel special burn burn

Jazz to Metal

Input	Output
and this this bitter	until until until until until until until
church bells <unk> on a sunday morn	lonely lonely lonely lonely lonely lonely

Metal to Jazz

Input	Output
i'm grasping harder with every breath	time there's there's there's there's there's
feel my anguish	there's there's there's there's

Table 11: Examples of outputs produced by the genre-to-genre model

low our model to have an objective that enforces style. This also requires modifying our classification model to have the same vocab as our generation model and to take in logits of each word. All models could benefit from training on more data points and further fine tuning with more computing resources, and we'd like to see how the models perform on other song datasets or classifying and styling based on artist as opposed genre.

Acknowledgements

We would like to thank Prof. Jacob Andreas, Prof. Jim Glass, and the TAs for their continuous support throughout this semester.

Figure 15: Picture of us during our presentation :)

References

Matei Bejan. 2021. Multi-lingual lyrics for genre classification.

D. Bužić and J. Dobša. 2018. Lyrics classification using naive bayes. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1011–1015.

Ning Dai, Jianze Liang, Xipeng Qiu, and Xuanjing Huang. 2019. Style transformer: Unpaired text style transfer without disentangled latent representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5997–6007, Florence, Italy. Association for Computational Linguistics.

Michael Fell and Caroline Sporleder. 2014. Lyrics-based analysis and classification of music. *COLING*, 2014:620–641.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. 2017. Controllable text generation. *CoRR*, abs/1703.00955.

Vineet John, Lili Mou, Hareesh Bahuleyan, and Olga Vechtomova. 2019. Disentangled representation learning for non-parallel text style transfer. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 424–434, Florence, Italy. Association for Computational Linguistics.

Keith Kahn-Harris. 2006. *Extreme metal: Music and culture on the edge*. Berg.

Heejin Kim and Kyung-Ah Sohn. 2020. How positive are you: Text style transfer using adaptive style embedding. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2115–2125, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Guillaume Lample, Sandeep Subramanian, Eric Smith, Ludovic Denoyer, Marc'Aurelio Ranzato, and Y-Lan Boureau. 2019. Multiple-attribute text rewriting. In *International Conference on Learning Representations*.

Juncen Li, Robin Jia, He He, and Percy Liang. 2018. Delete, retrieve, generate: A simple approach to sentiment and style transfer. *CoRR*, abs/1804.06437.

Bill C Malone. 2003. *Singing cowboys and musical mountaineers: Southern culture and the roots of country music*, volume 34. University of Georgia Press.

Cory McKay, John Ashley Burgoyne, Jason Hockman, Jordan BL Smith, Gabriel Vigliensoni, and Ichiro Fujinaga. 2010. Evaluating the genre classification performance of lyrical features relative to audio, symbolic and cultural features. *ISMIR*, page 213–218.

Chris Rojek. 2011. *Pop music, pop culture*. Polity.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. *CoRR*, abs/1705.09655.

Siddharth Sigtia and Simon Dixon. 2014. Improved music feature learning with deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6959–6963.

Alexandros Tsaptsinos. Music genre classification by lyrics using a hierarchical attention network.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *In*, pages 1096–1103. ACM.

Jingjing Xu, Xu Sun, Qi Zeng, Xuancheng Ren, Xiaodong Zhang, Houfeng Wang, and Wenjie Li. 2018. Unpaired sentiment-to-sentiment translation: A cycled reinforcement learning approach. *CoRR*, abs/1805.05181.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

Teh Chao Ying, Shyamala Doraisamy, and Lili Nurliyana Abdullah. 2012. Genre and mood classification using lyric features. In *2012 International Conference on Information Retrieval Knowledge Management*, pages 260–263.