Lyrical Distinction

Jesse Bartola jbartola@
Phillip Michalowski pmichalowski@
Nischal Tamang ntamang@
Max Berezin mberezin@

1 Problem statement

Music is an important part of our everyday lives. Technological advances in modern society have made it easier than ever for new artists to distribute their songs over the Internet. Unfortunately, the massive volume of songs that are regularly released into the public can make it difficult to discern which song is made by which artist. Moreover, a more important question arises: Are the most popular artists the ones with the most distinct music?

Aside from sound, the words an artist uses in a song are perhaps the most important determining factor in how it will appeal to the public. Lyrics contain semantic relevance, which allows for the possibility of implicit analysis of a songs cultural relevance. Some of the greatest artists of all time are renown for their unique use of lyricism. Our project will aim to determine if:

(a) Given a set of lyrics, who the artist is with a high-degree of certainty.
(b) Given our trained prediction model, what the lyrical uniqueness rank for various artists is.

2 Our Dataset

For the implementation of our described methodology, we chose to initially use a Kaggle sourced CSV file for our dataset Mousehead (2017). This dataset contains a repository of 57,650 songs of a multitude of artists and genres. The distribution of artists and songs can be seen in Figure 1, which uses a Pareto Distribution to visualize how close to 80% of the content within this dataset is described by only around 20% of the artists.

The column values of the dataset are

artist, song, link, text

In this schema, the column *text* contains the important lyrical information pertaining to the current song and artist.

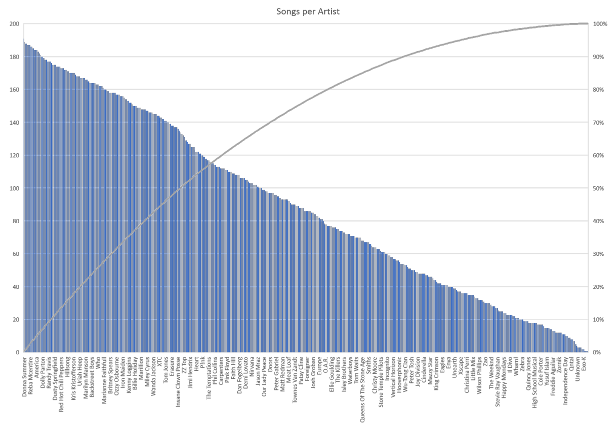

Figure 1: Songs per Artist: Mousehead (2017)

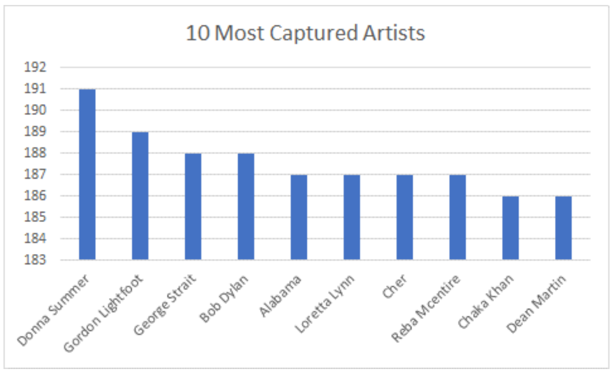

Figure 2: Most Captured Artists: Mousehead (2017)

For our future implementations, we plan on using Mishra (2017) with 362,237 songs for both its expanded amount of lyrical content. The column values of this dataset are :

index, song, year, artist, genre, lyrics

This allows us to utilize the additional song information for enahnced modeling (i.e Genre, Year).

3 Baselines

For our baseline approach, we used a simple averaging model. We first created a word matrix W. Each row in the matrix W contains the pre-trained

	Average	*Std. Dev*
Songs Per Artist	94	49
Word Count [Fig:3]	219.48	108.81

Table 1: Dataset Statistics

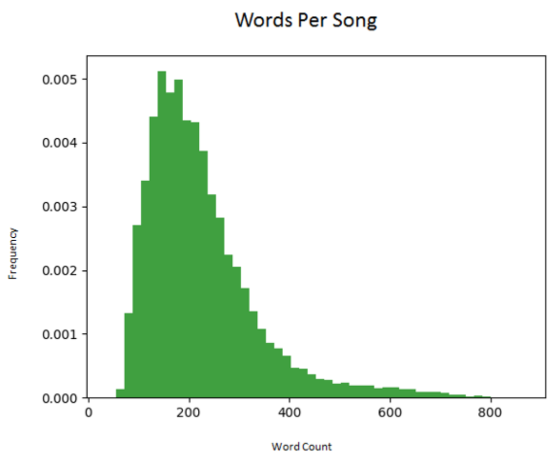

Figure 3

GloVe embedding Pennington et al. (2014) for a corresponding word in a song. A song vector is then computed by calculating the mean of each value in the row of matrix W. This vector thus acts as a unique representation for a particular song. For a particular artist, a songs matrix S is created. Each row in the matrix S contains all of the different song vectors for that artist. We then averaged the values of each row in matrix S to compute the final artist vector.

After completing these steps for every artist in the dataset, each artist is represented as a single vector embedding [4].

We created a 80/20 train and test split. We tested the model accuracy by taking a lyric from the test set and creating the artist vector embedding with the steps outlined previously. The k closest vectors measured by cosine similarity to the newly generated artist vector is extracted from the learned embeddings. If the correct artist is contained in the top k vectors, we considered it a correct prediction by the model. We chose k to be 10 since it is

	Cos	*Euclidean*
k = 1	7.78%	6.69%
k = 10	21.31%	18.51%
k = 50	41.99%	40.28%

Table 2: Cos vs. Euclidean Distance Accuracy

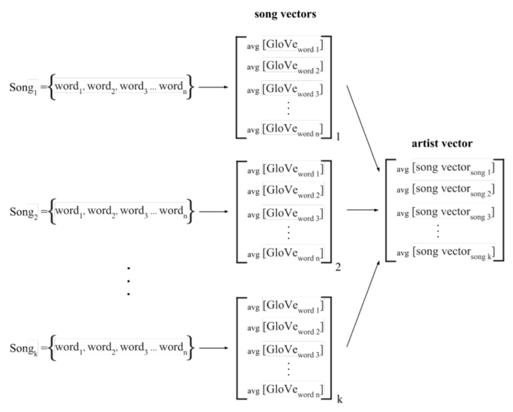

Figure 4: Initial Approach

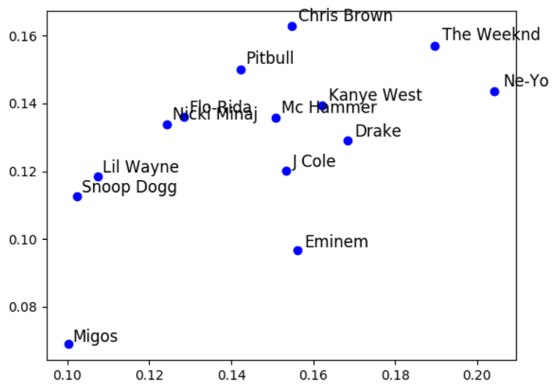

Figure 5: Artist Embeddings Mapped to 2-Dimensions

a fairly low percent of the 643 total artists in the dataset. The model performed at an accuracy of 21.31%.

Regarding the vector similarity testing, we experimented both cosine and euclidean distance. Cosine consistently outperformed euclidean on all k values that we tested. The results are shown in table [2].

Using the learned artist embeddings, we also plotted a small subset of artists on a 2D graph [5].

4 Error analysis

We hypothesize a few reasons to why the baseline model performs so poorly. One is due to there being the same mapping function between the generation of song vectors and the artist vector. Simply averaging rows of matrices loses the meaning between the artists and songs. Our original dataset is also not robust enough. It contains 643 unique artists and around 55,000 songs. The artist with the most songs in the dataset has only about 200 songs. For our final report, we would like to find a bigger dataset with more songs per artist. In addition, we could also find a another dataset and combine it with the dataset we are already using. After annotating failed examples, one thing we noticed was the dataset we used contains song structure labels such as *"chorus"* or *"verse 1"*.

i.e:

[repeat chorus] only wait so long before your flower petals are all gone baby i was trying to burn bridges of stone . [repeat chorus]

We also noticed that all songs the model failed on contained common words in the English language as demonstrated by Zipfs law. In the future, we should remove the song labels and also the most common English words from the dataset as they do not help distinguish between different songs.

Regarding the issue of the naive mapping function between matrices, we plan to use Neural Networks to obtain a better performing model. The following section outlines our procedure for the Neural Networks implementation.

5 Our approach

Repository

5.1 Introduction

Our main approach consisted of building a deep learning model to classify which artist created a song given a string of lyrics. We used the Pytorch package Paszke et al. (2017) and iterated over several implementations of different neural networks.

5.2 Vanilla Neural Network

The first non-trivial implementation we attempted was a simple network with an input layer, three hidden layers of the same dimension and a softmax output layer. To construct the input, we tokenized a string of lyrics into a list of strings, where each string contains exactly one word.

Each word in the input list was mapping to its respective word embedding in the *glove.6B.50d.txt* file obtained from Homework 2. The output layer produced a probability distribution over all 643 artist in our original dataset. Before we could start training, we needed a reasonable method of standardizing the input to the network.

5.3 Input

We decided to create our input to the network by mapping each word in a song to its respective embedding vector and fixing the number of input words from each song to a constant value. If we made this cutoff too short (say, a maximum of 30 words per song input) it could result in the loss of potentially useful information contained in the rest of the song. If the cutoff value was too large, then much of the training data would be padded with zeros, thus taking up unnecessary memory and adding meaningless values to our inputs.

We approached this situation like any good scientist should, and took a closer look at a histogram of words per song in our dataset [3]. The distribution was right-skewed with the mean around 220 words. After training and testing the network using several different cutoff values, a input length of 300 words seemed to produce the highest test accuracy.

5.4 Training and Testing

Similar to our Baseline model, we used a 80/20 train/test split on the vanilla neural network model. Each training example is a 2-tuple containing a unique integer representing an artist and a fixed-length array of word embeddings corresponding to the lyrics in a song. We standardized the training session to run over 20 epochs, as it seemed like our optimization algorithm had diminishing returns when run for any longer. After each training session, we tested the network on our test data by comparing the element in the output vector with the highest probability to the ground truth value of the artist who created the song.

While training, we adjusted the batch sizes to various powers of 2 (16, 32, 64, 128, and 256) and determined that with other hyperparameters fixed, a batch size of 64 yielded best test accuracy. We experimented using the Adam and SGD optimizers, both of which yielded very similar test results, even when used with different learning rates (ranging from 0.001 to 0.5).

We tested our network numerous times after training it with different combinations of hyperparameters. The accuracy was appalling each and every time. The maximum value we achieved was a test result of 0.38% accuracy. After some investigation, we discovered a few reasons for this:

1. We had 643 different output classes, but less than 20 training examples for many artists.

The asymmetry of songs per artist potentially biased the network into learning stronger weights for some artists than others

2. Our test method was comparing the output class with the highest probability to the true value of the artist. Given the amount of data we had per class, it would have been more reasonable to check if the ground truth artist was contained in the top k output classes (5 or 10) with highest probability

Our poor results in this network led us to try other deep learning implementation that better lend themselves to NLP tasks. Needless to say, vanilla neural network are far from ideal when it comes to tasks involving sequential input.

5.5 LSTM

The second implementation we tried was an LSTM (Long-Short Term Memory) recurrent neural network. LSTM (2017) networks are much better at extracting information from sequential input and time-series data. This time instead of using the pre-trained embeddings from the glove file, we learned all embedding representations from our dataset by creating an embedding layer and learning them through the training process.

Utilizing the same strategies as we did for the vanilla neural network, we tried different values of each hyperparameter (batch size, optimizer, input size, learning rate, hidden state dimensions, etc.) and found that with a batch size of 64, Adam optimizer, input size of 300, learning rate of 0.05, and 2 hidden states of dimension 10 we achieved an average accuracy of 60% over 10 test runs, using k = 30 as a the correctness distance (i.e., the ground truth artist appeared in the 30 highest probabilities in the output distribution, which is 30/643 = 5% of out dataset). Setting k to a lower value dramatically decreased the testing accuracy from 60% to 20% over ten test trials. Compared to the first network, testing with k = 1 resulting in 4.8% accuracy on ten test trials, which was about a 12x improvement.

5.6 Improvements and Next Steps

The LSTM implementation works, and has decent test accuracy for large k. Unfortunately for small k it performs very poorly. It is obvious that the model can be substantially improved by obtaining more training data per artist, including regularization to prevent our model from learning large weights, and learning more about the nuances of neural networks in general. The next steps are to improve our LSTM model by trying some of the methods described above, and then to extract the learned weights from the output layer to get a vector representation of each artist. We will then proceed to project these vectors onto a lower dimensional space in order to visualize the models learned representation of each artists lyrical similarity.

5.7 Timing

Given that our Neural Network was trained on a Macbook Pro (2.2 GHz Intel Core i7), our training time took 40 minutes per epoch, with 20 total epochs. In total, this was 13.3 hours for training. In order to speed up this process, we plan leveraging the enhanced processing power of GPUs. Specifically, the processing power of a NVIDIA GeForce GTX 1080. This will substantially increase our ability to tune our models hyperparameters and to see which configurations of the network learn the best representation for our artists given their songs.

6 Timeline for the rest of the project

1. Parameter Tuning (0.5 weeks) - November 19
2. Visualization Refinement & Network Tuning (2 weeks) - December 3
3. Final report and Presentation Preparation (2 weeks) - Presentation Day

References

LSTM (2017). Long short-term memory - wikipedia.

Mishra (2017). 380000+ lyrics — kaggle.

Mousehead (2017). 55000+ song lyrics — kaggle.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation.