Andrés García Molina, PhD
March 4, 2021

Using NLP to Predict Song Genres through their Lyrics

I. Introduction

Natural Language Processing (NLP) techniques allow making sense of natural language, one of the most challenging types of data in machine learning given the unbounded, unstructured nature of language. In an era of digital circulation and streaming, music metadata is crucial in the process of accurately tagging and classifying music to aid its discoverability. How might we leverage NLP techniques in order to correctly assign genre labels to songs? This project combines NLP techniques with neural networks in order to predict genre labels on a corpus of song lyrics.

A 2019 report published by Music Business Worldwide asserts that nearly 40,000 tracks are uploaded to Spotify every day. Given such a volume, manual classification is an expensive and slow task that is also error-prone. While in theory those uploading bear the burden of providing meaningful and accurate metadata, automating a process for sanity checks is well worth the investment. This notebook explores some ways to implement a genre-tagging model that is based on neural network training of labeled lyrics datasets.

Stakeholders: Who might be interested in this report?

A wide range of audiences are invested in understanding how to automate genre label tagging, including:

- Artists who upload music to various online platforms and hope to maximize their searchability and findability
- The platforms themselves, who hope to optimize music discovery and recommendation processes
- Audiences who want to find music according to genre labels and also discover new acts that might be of interest

Methods

Several neural network machine learning models are trained and tested on a song lyrics dataset that has undergone preparation based on NLP techniques.

II. Data Wrangling

Perhaps the biggest challenge in this project concerns acquiring good, labeled data that pairs song lyrics with genres. For the task at hand, I analyzed several already-available datasets that have performed similar work, including that of Bajwa et al, Sianipar et al, Kovachev et al, and Ram and Salz. As an initial step, the following datasets were imported and examined:

1. "genre_lyrics_data.csv", by Bajwa et al, available on GitHub, including 6,733 lyrics and 90 genres.

2. "tcc_ceds_music.csv", by Moura et al, available via Mendeley Data, including 28,372 lyrics and 7 genres.

3. "spotify_songs.csv", by Muhammad Nakhaee and available via Kaggle, including 18,454 lyrics and 6 genres.

4. "original_cleaned_lyrics.csv", by Yalamanchili et al, available via GitHub, including 227,449 lyrics and 11 genres. This is a processed version of the no-longer available "380,000+ lyrics from MetroLyrics" dataset (likely removed due to copyright infringement) that served as the starting point for many related projects.

Ultimately, #2, #3 and #4 are selected as the source datasets for this project, given their more restricted range in genres. The first dataset was discarded, given its small number of observations in relation to the vast spread of genres contained within.

The different datasets are joined and only essential features are kept: the lyrics themselves and the corresponding genre label. Some other steps taken include:

1. Dropping missing values: In this case, they refer to no lyrics available or no genre label available. In this particular business case, these values cannot be imputed in a reasonable way.
2. Genre labels are standardized: For example, “rock”, “rock ‘n’ roll”, “rock and roll” all become “Rock.”
3. Duplicates are handled: Only one instance is kept for full duplicates (i.e. both lyrics and genre match) and lyric duplicates with different genres (e.g. a song by The Beatles labeled as both pop and rock).

4. Removing other tokens: These include punctuation, typical lyrics identifiers such as "[Chorus]" or "[Verse]", lyrics comprised of the word "Instrumental" as the only lyrical content, and corrupted/non-ASCII characters.
5. Detecting lyrics languages and only selecting English lyrics.

A decision is made early-on: Our dataset contains over 250,000 lyrics, but the classes are imbalanced. Rock comprises roughly 50% of the genres, and some genres (e.g. folk and reggae) represent less than 1% of the dataset. We thus focus on the top five genres—Rock, Pop, Hip-hop, Metal, and Country—and we use the minority class's size (n_Country = 18,580).

We then tokenize lyrics in order to:

1. Remove stopwords, and
2. Lemmatize, which in turn requires applying part of speech tags.

More details on these preprocessing steps can be found in the Data Wrangling notebook of the project.

III. Exploratory Data Analysis (EDA) and Feature Engineering

EDA was performed in order to get a general sense of the characteristics of the lyrics dataset. Initial exploration revealed that the process of lemmatizing had led to some instances of empty sets, as well as duplicates. These cases were addressed as an initial step.

Two sentiment analysis toolkits are applied: Vader and TextBlob. As the images below demonstrates, the analyzer performances are remarkably different. Overall, Vader is quicker to polarize while TextBlob's output approaches a normal distribution:

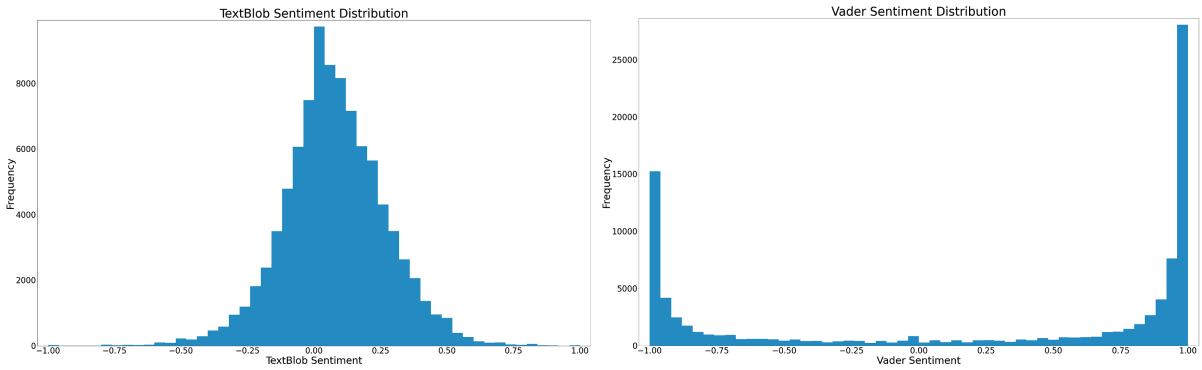

Figure 1: TextBlob (left) and Vader (right) sentiment distributions.

The figure below summarizes these differences per genre:

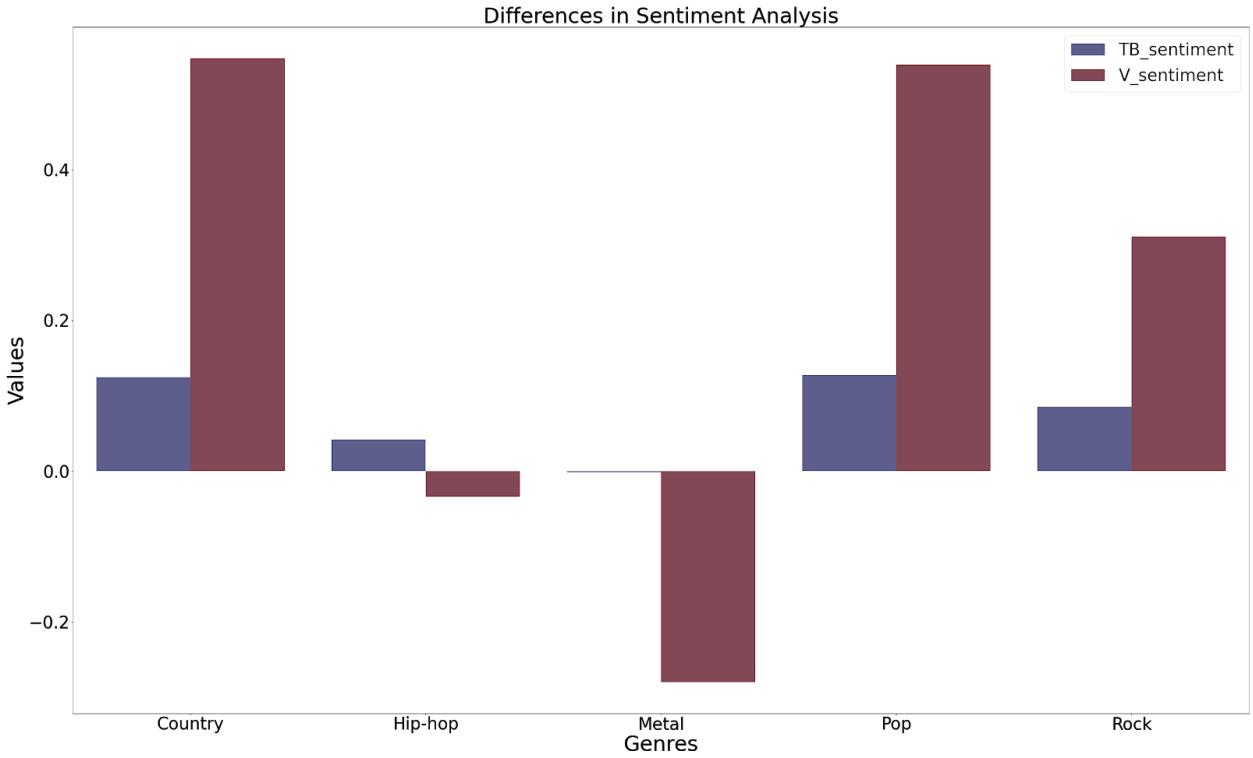

Figure 2: TextBlob and Vader differences in sentiment analysis.

We also explore how verbose each genre is, or put differently, how many unique words are employed in each of the genre labels:

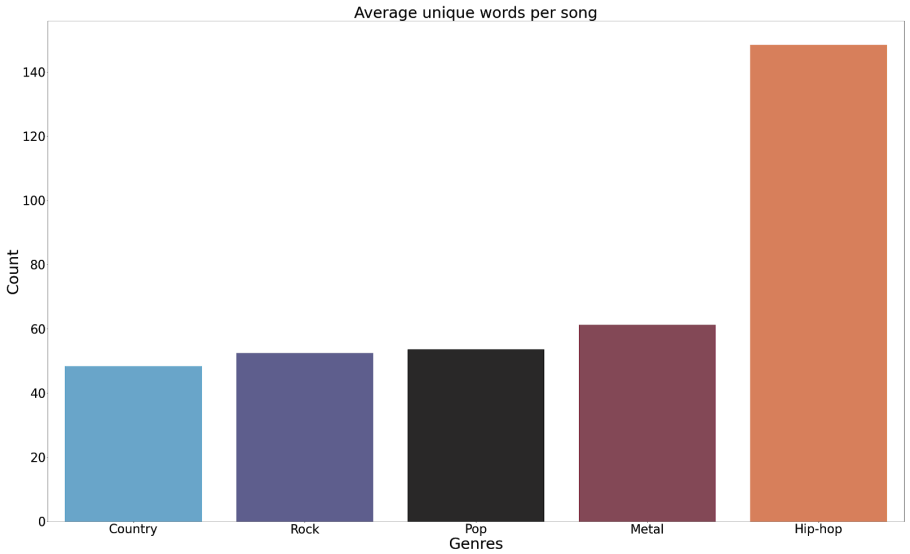

Figure 3: Average unique words per genre. Unsurprisingly, hip-hop is a verbose outlier.

Our EDA exploration also maps the 25 most frequent words per genre. Full details can be seen in the EDA and Feature Engineering notebook of the project.

The following methods are employed in this section:

1. LDA (Latent Dirichlet Allocation)
2. NMF (Non-Negative Matrix Factorization)

Both approaches yield different topic modeling, and below we explore the differences in their outputs. While there is significant overlap in the topics yielded, NMF is favored because the topics presented cover a broader range and are more distinct. In a problem like the one at hand, a more fine-tuned approach might be appropriate if topic modeling were the objective of the project.

For instance, modeling could be genre specific in addition to global. In order to perform LDA modeling, the lyrics are first converted into a document term matrix using a count vectorizer. NMF modeling requires an input in the form of a TFIDF, or term frequency–inverse document frequency matrix, which reflects how important each word is in the entire corpus.

It is important to note that each model yields different topics, and that the way they are presented are through word lists of settable predetermined length that require some subjective interpretation.

LDA Topics

Topic 0: Life, love, longing.
Topic 1: Money and hustling.
Topic 2: Love and sexuality.
Topic 3: Love and fun.
Topic 4: Affection, women.
Topic 5: Death, faith.
Topic 6: Dance.

NMF Topics

Topic 0: Life as a journey.
Topic 1: Money and hustling.
Topic 2: Home and longing.
Topic 3: Love and life.
Topic 4: Love and fun.
Topic 5: Love and sexuality.
Topic 6: Introspection and time

The figure below presents topic distributions for the entire sample.

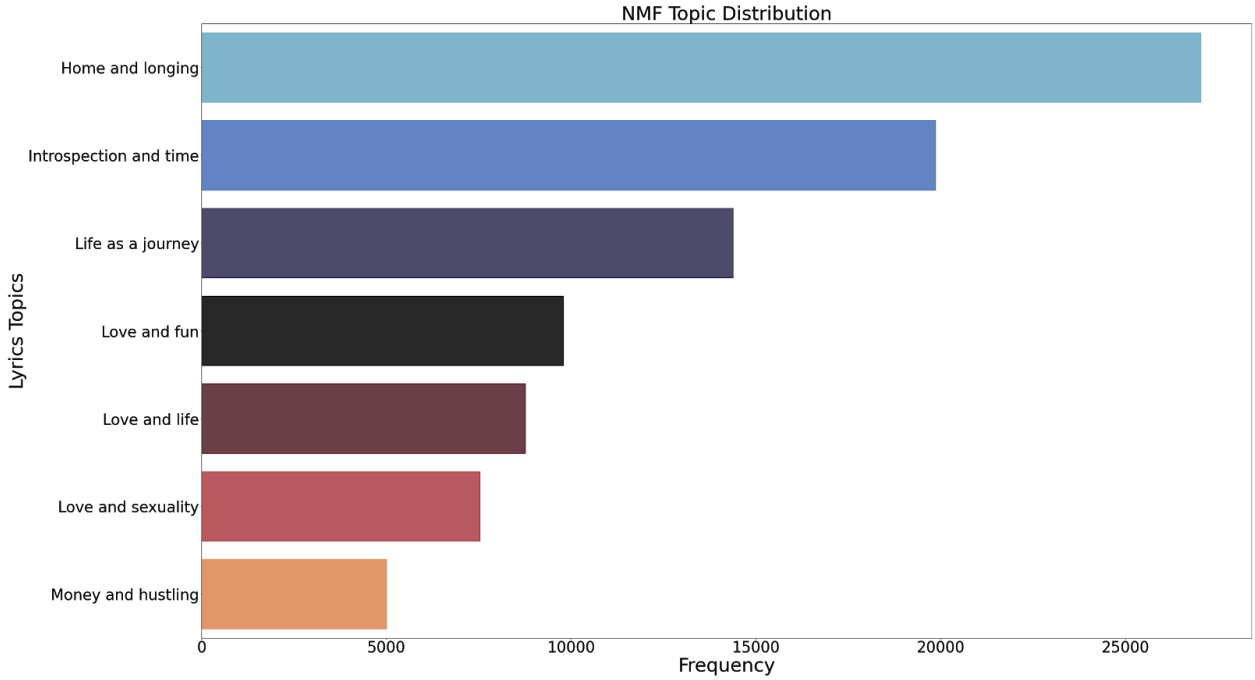

Figure 4: Global topic distributions. "Home and longing" is a clear dominator as a common theme across genres.

Our EDA exploration also maps how topics are distributed across specific genres. Full details can be seen in the [EDA and Feature Engineering notebook of the project.]

IV. Modeling

Prior to training and testing models, we use a label encoder for our genres target variable. We also follow the conventional splitting of our data into training and test sets, and for the purposes of training a neural network we use a "bag of words" (BOW) representation of our lyrics, which is a vectorized count transformation. We also normalize the BOW representation so that all values in our matrix are between 0 and 1; scaled data allows neural networks to achieve better results.

As a baseline model we employ a logistic regression model fit on the normalized BOW training data, achieving 62% accuracy.

We start with a basic keras Sequential model with one 100-neuron layer that achieves 77% training accuracy and 63% testing accuracy after 60 epochs. The results below suggest that validation accuracy stops improving after around 20 epochs, and validation loss stagnates roughly around the same time.

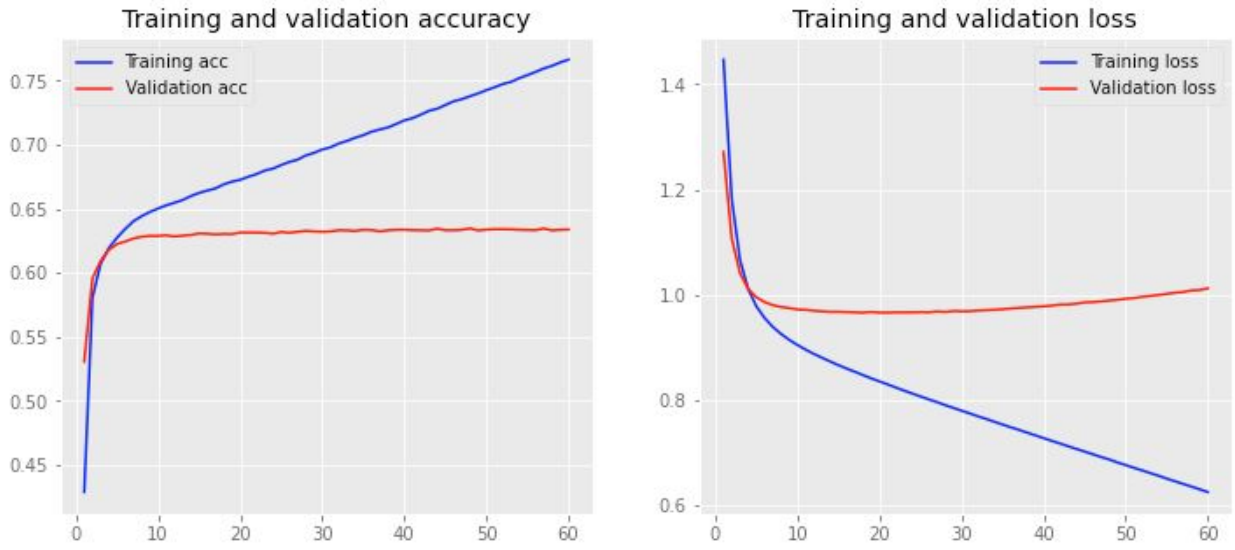

Figure 5: Training and validation accuracy for a neural network with an inner layer of 100 neurons after 60 epochs.

The model is re-run with a smaller epoch number and a smaller batch size. With the modifications made, the training accuracy improves, but the validation accuracy diminishes, suggesting overfitting. One final experiment is made, setting the number of nodes in the hidden layer to 200, achieving a training accuracy of 65% and a testing accuracy of 63%.

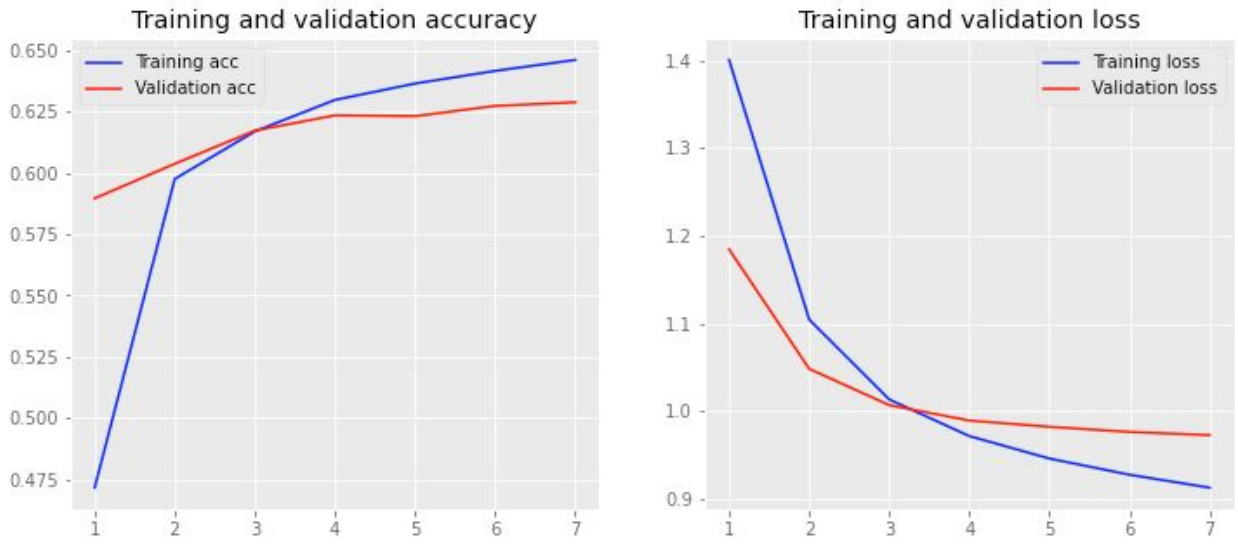

Figure 6: A simple Sequential model with a single 200-node layer after 7 epochs.

We employ several other neural networks and their associated techniques in order to attempt higher accuracy scores in our validation data. As a general trend, we notice that

it is indeed possible to obtain near-perfect accuracy in training, which in no way correlates to better accuracy in validation.

We begin using word embeddings in order to train a neural network with an embedding layer that maps our BOW representation to dense vectors, thus reducing dimensionality. Details on the required pre-processing steps can be reviewed in the Modeling notebook of the project.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 2955, 50)          5611200
_________________________________________________________________
flatten (Flatten)            (None, 147750)            0
_________________________________________________________________
dense (Dense)                (None, 50)                7387550
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 255
=================================================================
Total params: 12,999,005
Trainable params: 12,999,005
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Structure of the sequential model with embedding.

A verbose summary of the model as it trains alerts us to a general trend: epochs increase training accuracy but validation accuracy peaks at 62.27% around 25 epochs and then decays. Global max pooling is also employed as a technique to reduce dimensionality:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 2955, 50)          5611200
_________________________________________________________________
global_max_pooling1d (Global (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 50)                2550
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 255
=================================================================
Total params: 5,614,005
Trainable params: 5,614,005
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: Global max pooling after embedding.

This is the best performing model, which achieves 64% testing accuracy. The general trend, however, remains constant, where testing accuracy stagnates quickly.

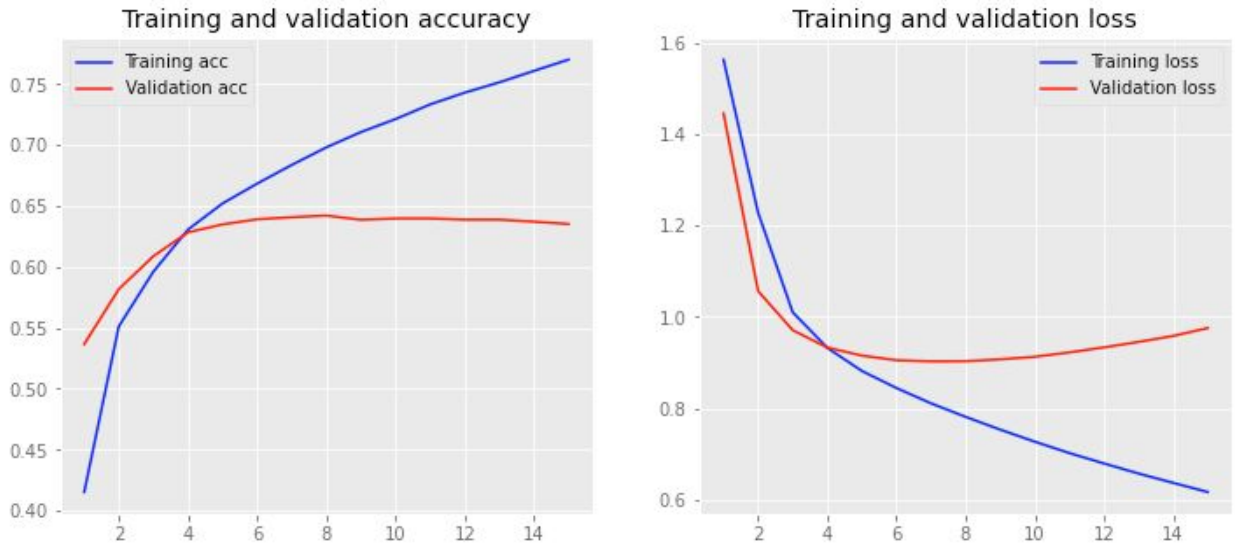

Figure 9: Embedding + global max pooling performance after 15 epochs.

The Modeling notebook of the project details behavior across epochs and also using other approaches. A common approach is to use pre-existing, pre-trained word embeddings rather than creating our own based on our local corpus. The embeddings used by GloVe (Global Vectors for Word Representation)--developed by the Stanford NLP Group--are employed. Specifically, the word embeddings from the Wikipedia 2014 + Gigaword 5 6 billion token model is used (the size of the vocabulary is 400k). The percentage of our vocabulary that is covered by the pretrained model is slightly over 50%. The results obtained stagnate around 48%; with such a low representation of our vocabulary (50%), a pre-trained word embeddings model cannot realistically obtain an accuracy higher than the percentage of representation.

We then test if anything changes once we allow the embedding to be trainable, by setting the hyperparameter "trainable" to True. The trainable embedding and max pooling model appears to stagnate around 30 epochs, around a 62% of accuracy.

Finally, we also apply Convolutional Neural Networks (CNN) as a way to generate and explore hidden patterns in our dataset. More specifically, a convolutional layer is added between embedding and pooling.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 2955, 100)         11222400
_________________________________________________________________
conv1d (Conv1D)              (None, 2951, 128)         64128
_________________________________________________________________
global_max_pooling1d (Global (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 10)                1290
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 55
=================================================================
Total params: 11,287,873
Trainable params: 11,287,873
Non-trainable params: 0
_________________________________________________________________
```

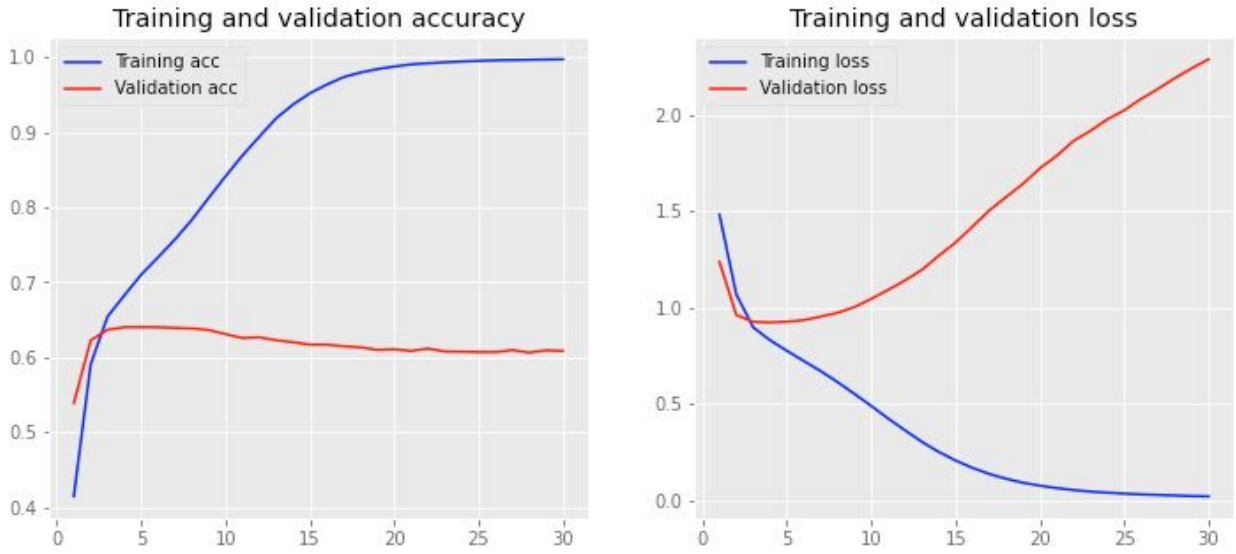

Figure 10: A convolutional neural network and its performance after 30 epochs.

As with other models, convolutional networks quickly start overfitting, and testing accuracy stagnates in a similar range as do all other models. We run two final sanity checks with simpler models. A Random Forest Classifier and a Multinomial Naive Bayes model confirm the general ceiling of this dataset: the random forest's testing accuracy clocks in at 62% while the NB model reaches 61% accuracy.

Conclusions and Recommendations

The results above show that a Random Forest Classifier trained on a Bag of Words representation, as with Logistic Regression, appear to produce similar results as neural networks. If this application were to be deployed, neural networks would not be favored

as an appropriate solution, as the yielded results do not improve significantly on simpler models, but are much more computationally expensive.

The next steps would require further testing with hyperparameter tuning using simple models and one of the less computationally expensive neural networks. Additional neural networks could be experimented with, such as a Long Short-Term Memory (LSTM) neural network, Recurrent Neural Networks (RNN), Gated Recurrent Unit (GRU), Hierarchical Attention Networks, Recurrent Convolutional Neural Networks (RCNN), Random Multimodel Deep Learning (RMDL), and Hierarchical Deep Learning for Text (HDLTex).

Ultimately, the question might lie in the quality of the data, as the acquired datasets come from multiple sources and data quality is certainly an issue. This project ultimately demonstrates that one of the more crucial steps in the data science lifecycle concerns data acquisition and quality. In this case, generating a tidy lyrics dataset would produce high pre-processing costs.