# Song Lyric Classification by Artist

Udacity Machine Learning Nanodegree
Capstone Project

Greg Mogavero
July 18, 2017

# I. Definition

## Project Overview

Machine learning is commonly used to perform natural language processing (NLP), a "field of computer science, artificial intelligence and computational linguistics concerned with the interactions between computers and human (natural) languages, and, in particular, concerned with programming computers to fruitfully process large natural language corpora [1]." One particular use case for NLP is analyzing song lyrics to classify them by artist. A successful machine learning algorithm would not only have to take into account the songwriter's lexicon, but also pick up on the unique subtleties of the artist's style in order to differentiate artists who write about similar topics.

Efforts have already been made by the machine learning community to classify songs based on their lyrics by genre and artist. Sadovsky and Chen [2] use Maxent and SVM classifiers to accomplish this task fairly successfully. Using no acoustic information whatsoever, they are able to achieve 70-80% artist classification accuracy. Because they use Bag of Words for feature selection, they remark that they might have been able to achieve higher accuracy if they did some sort of semantic analysis.

For this project, I use a dataset comprised of 57,650 song lyrics scraped from LyricsFreak by Sergey Kuznetsov [3].

## Problem Statement

Given song lyrics (text only) as input and the corresponding artists as labels, I attempt to build a supervised learner that can classify new songs by artist. Input is truncated or padded during preprocessing so that each sample is a fixed size.

I attempt to solve the problem of classifying song lyrics by artist by training a Long Short Term Memory (LSTM) Neural Network. These models, when trained on text data, can "remember" information they have seen in the past. I hypothesize that the ability to learn this contextual information will help the learner distinguish songs by different artists who have similar lexicons, but different styles.

In order to get meaningful results, I choose the five artists from the dataset with the largest repertoire of songs. After preprocessing the input text data using Word2Vec, I train and test a basic LSTM network, analyze its performance, and then try to improve results through hyperparameter tuning and architecture modification. Final results are compared against a benchmark model based on the models Sadovsky and Chen used in their experiment.

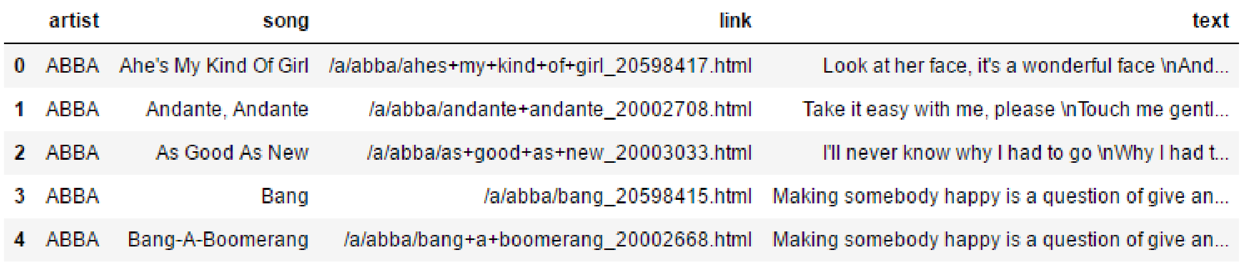
### METRICS

I use accuracy as my main metric for measuring performance. Because I only train the model on the five artists with the most songs, I do not expect there to be significant class imbalance and therefore the accuracy score will be a good measurement of the model's performance. I also analyze the confusion matrix of the prediction results to see which labels the model excels and struggles with.

## II. ANALYSIS

### DATA EXPLORATION

The dataset comes as a .csv file with four columns: artist, song, link, and text. For the purposes of this project, only the text column is necessary for the input and the artist column for the labels. The following figure shows the first five rows of the raw dataset.



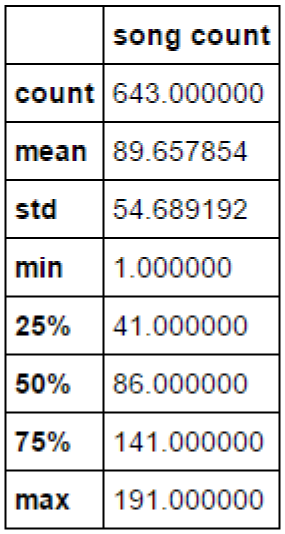
| | artist | song | link | text |
|---|---|---|---|---|
| 0 | ABBA | Ahe's My Kind Of Girl | /a/abba/ahes+my+kind+of+girl_20598417.html | Look at her face, it's a wonderful face \nAnd... |
| 1 | ABBA | Andante, Andante | /a/abba/andante+andante_20002708.html | Take it easy with me, please \nTouch me gentl... |
| 2 | ABBA | As Good As New | /a/abba/as+good+as+new_20003033.html | I'll never know why I had to go \nWhy I had t... |
| 3 | ABBA | Bang | /a/abba/bang_20598415.html | Making somebody happy is a question of give an... |
| 4 | ABBA | Bang-A-Boomerang | /a/abba/bang+a+boomerang_20002668.html | Making somebody happy is a question of give an... |

***Figure 1.*** *First five rows of the raw dataset.*

Right away, within these first five rows, we can see two songs (indices 3 and 4) with very similar lyrics. In fact, they're the same lyrics with minor discrepancies. LyricsFreak's lyrics are all user-submitted, and so duplicates like these (as well as typos and other errors) might slip past any curation they might have. However, because there should be relatively few duplicates, and because the text preprocessing should cause typos to be ignored, this shouldn't cause any problems for our model.

## Exploratory Visualization

Without modifying the dataset yet, Figure 2 shows us the statistics concerning the number of songs per artist.

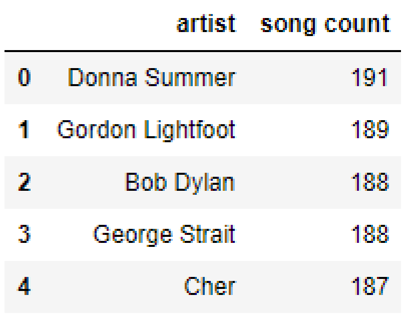


| | song count |
|---|---|
| **count** | 643.000000 |
| **mean** | 89.657854 |
| **std** | 54.689192 |
| **min** | 1.000000 |
| **25%** | 41.000000 |
| **50%** | 86.000000 |
| **75%** | 141.000000 |
| **max** | 191.000000 |

***Figure 2.*** *Statistical description of the number of songs per artist.*

We can see that the number of songs ranges from 1 to 191, with a mean of 90 and a standard deviation of 55. Using the entire dataset would result in a huge class imbalance, and the supervised learner would struggle to learn anything meaningful about the artists with relatively few songs.

Figure 3 shows us what the dataset would look like if we only use the five artists with the highest number of songs.



| | artist | song count |
|---|---|---|
| **0** | Donna Summer | 191 |
| **1** | Gordon Lightfoot | 189 |
| **2** | Bob Dylan | 188 |
| **3** | George Strait | 188 |
| **4** | Cher | 187 |

***Figure 3.*** *Number of songs per artist for only the five artists with the most songs.*

These classes are much more balanced, and because we choose the five artists with the *most* songs, we're effectively maximizing our sample size.

## Algorithms and Techniques

An LSTM is a specialized type of recurrent neural network that excels in remembering long-term dependencies. They tend to perform well on sequential data like text because they can utilize input from the past to help make predictions. Because my input data is solely text, an LSTM is a good choice for this problem.

LSTMs (and recurrent neural networks in general) can be thought of as having a chain-like structure of repeating modules, each containing hidden layers, where the output of one module is fed into the next along with the next portion of input. The figure below illustrates a very basic recurrent neural network.

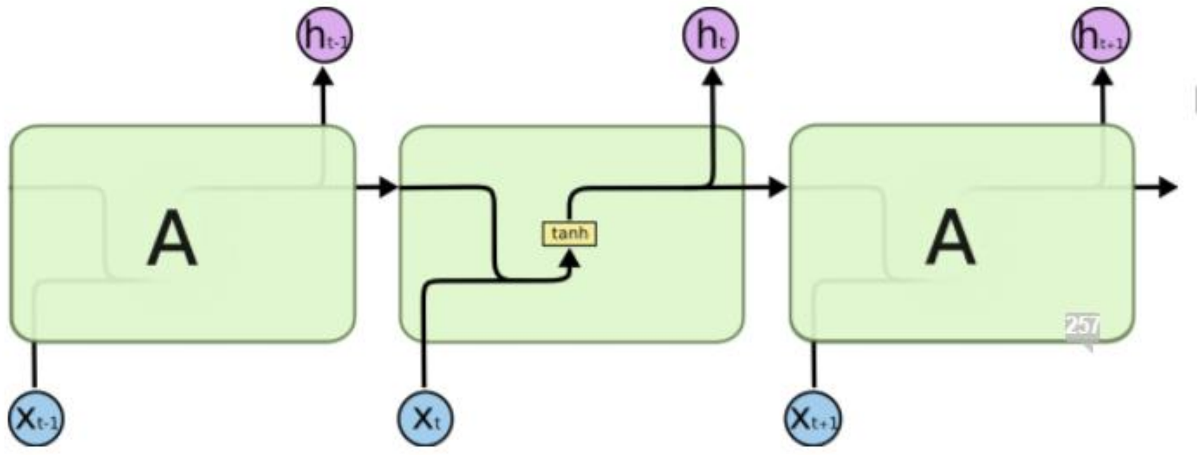


***Figure 4.*** *The repeating module in a standard RNN contains a single layer [4].*

LSTMs have a similar repeating structure, but they contain four very specialized layers, and they keep a cell state. The figure below illustrates this nicely.

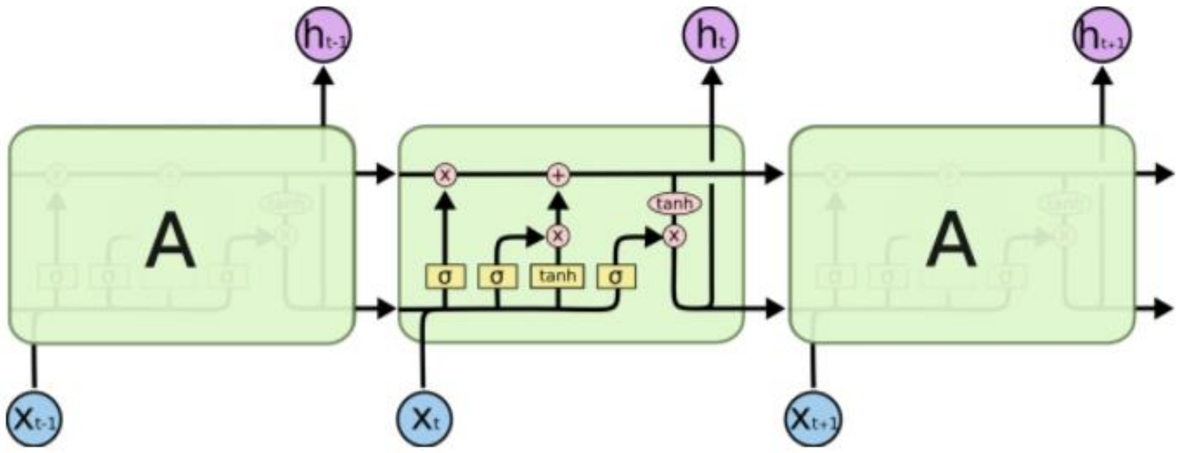
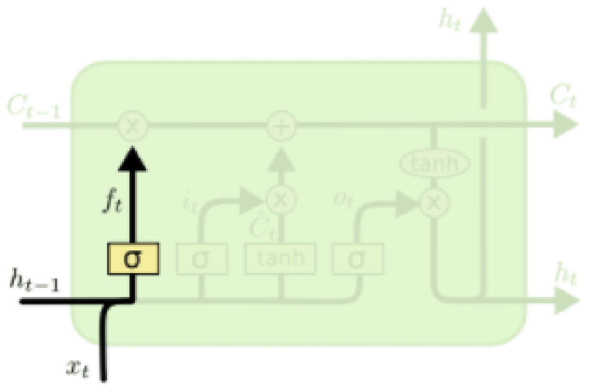


***Figure 5.*** *The repeating module in an LSTM contains four interacting layers [4].*

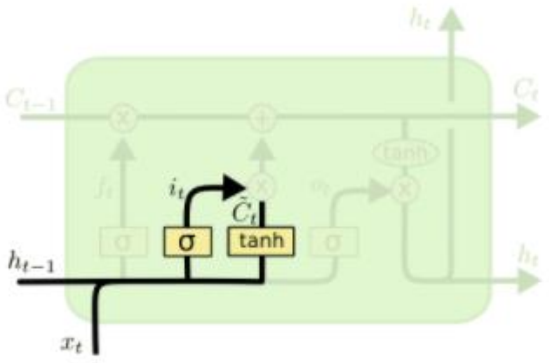
The line running through the top of the figure is the cell state, and LSTMs "have the ability to remove or add information to the cell state, carefully regulated by structures called gates" [4]. The LSTM's first task is to decide what information to throw away from the previous cell state using the "forget gate layer", as the figure below shows.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

***Figure 6.*** *The "forget gate layer" [4].*

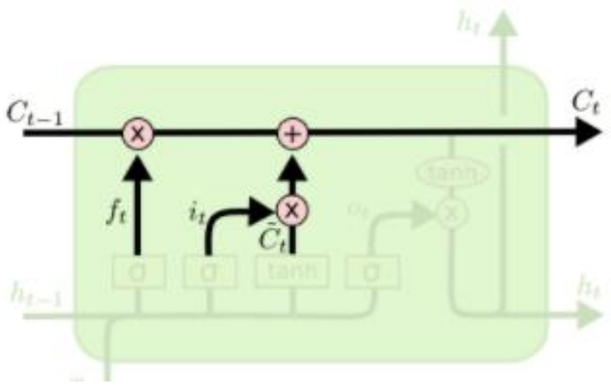
The "forget gate layer" uses the sigmoid function to output 0's or 1's based on sample input and the previous step's output and multiplies this by the cell state, effectively deciding what to keep and what to discard. The next layer is called the "input gate layer" and decides what new information to store in the cell state. The layer first uses the *tanh* function to generate "candidate values" ranging from -1 to 1 from the input, as well as another sigmoid function to decide which of these values to keep.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

***Figure 7.*** *The "input gate layer" generating candidate values [4].*

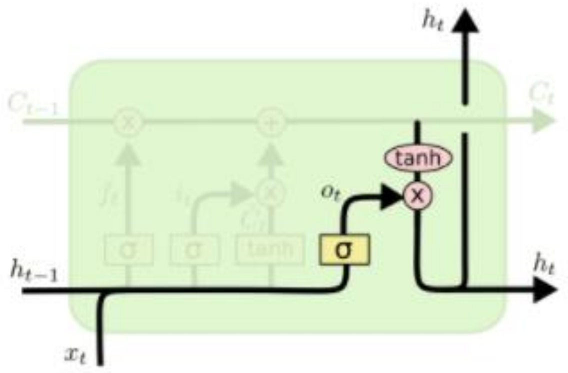
The layer then updates the cell state with these new candidate values using addition.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

***Figure 8.*** *Updating the cell state [4].*

Finally, the LSTM decides what values to output. It runs the cell state through another *tanh* function, constraining the values to between -1 and 1 again, and then selecting which of these values to output by multiplying with the output of another sigmoid function.



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

***Figure 9.*** *Deciding what to output [4].*

One of the downsides to using an LSTM is that they tend to require large datasets. For instance, in his post about sequence classification using LSTMs [5], Jason Brownlee uses the “Large Movie Review Dataset” with roughly 25,000 samples to perform a binary classification. My dataset, after preprocessing, has around 950 samples and 5 classes. This means that my model’s architecture has to be relatively simple, as complex models will quickly overfit to the training data.

## BENCHMARK

I use an SVM classifier trained on bag of words using English stopwords and limited to the 250 most frequent words as my benchmark model, leaving the parameters at default. The benchmark model attains an accuracy score of 42.21% with 5-fold cross-validation. Because a model that uses random guessing to make predictions would achieve an expected accuracy score of 20% with five labels, this means that it is possible to learn to a certain extent how to distinguish artists based on their word usage alone. The figure below shows the confusion matrix for the benchmark's predictions across all folds.

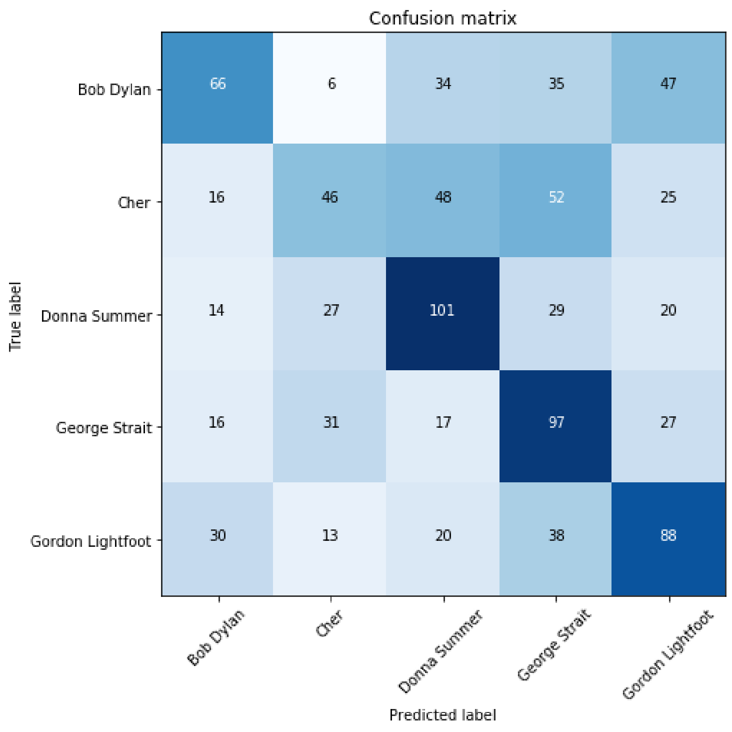


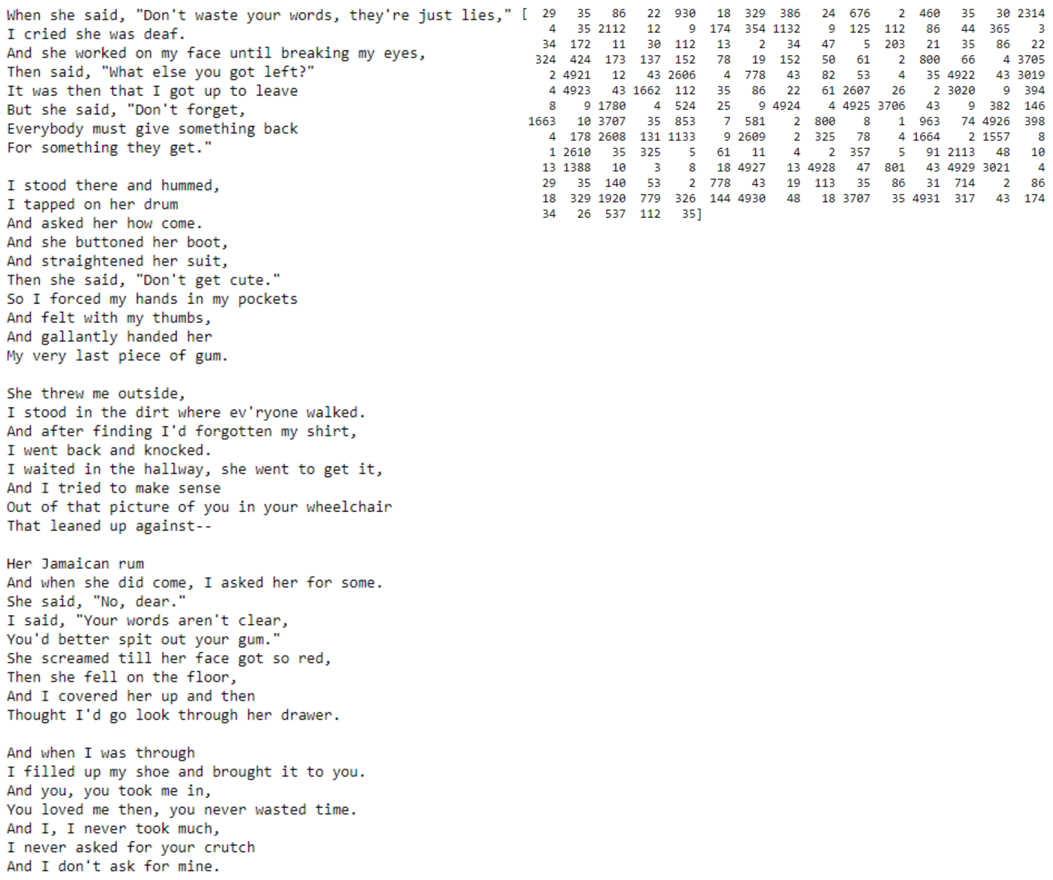
***Figure 10.*** *Confusion matrix for the benchmark's predictions.*

We can see that the model is best at predicting "George Strait" and "Donna Summer" correctly, and is the worst at predicting "Cher". It will be interesting to see how the LSTM model performs with these labels.

## III. METHODOLOGY

### DATA PREPROCESSING

As previously stated, I first trim the dataset down to only contain the five artists with the most songs. After that, I converted each word into a number representing its frequency rank. To do this, I wrote an algorithm to count each word in the reduced dataset, ordered the words by frequency, and then built an index mapping words to frequency rank. Smaller numbers mean higher frequency. The number 0 is special in that since I limit the number of words to index to the 5000 most frequent, then words that appear too infrequently will be assigned a value of 0. Furthermore, each sample gets either truncated or padded with 0's to a fixed length of 200 words. The model should hopefully learn that 0 contains no information. The figure below show a before-and-after view of a sample of the data.



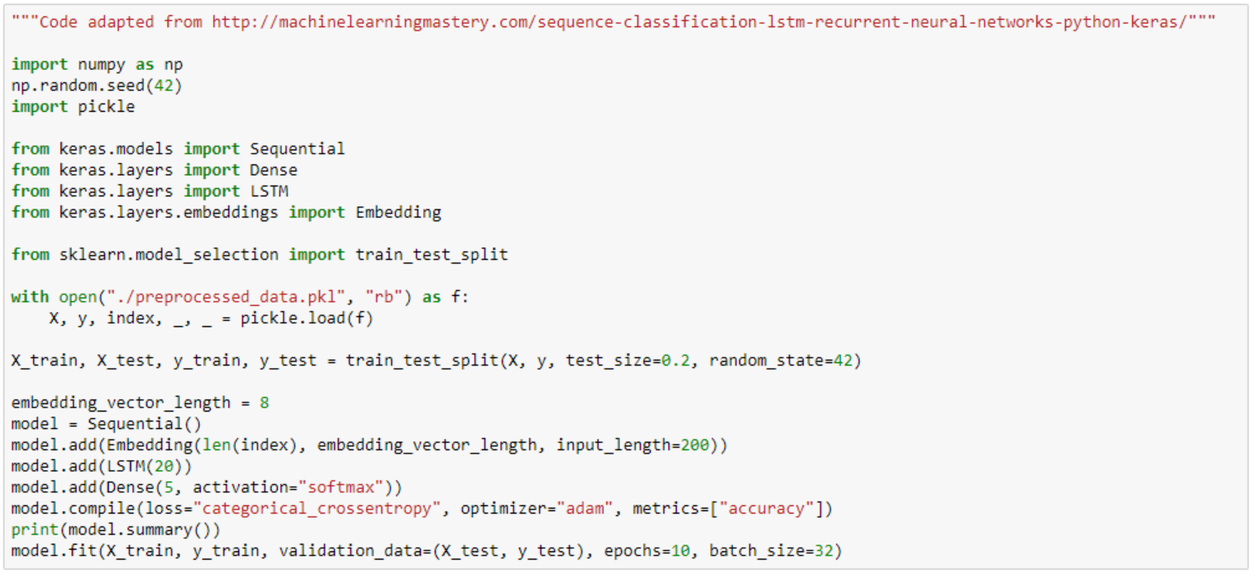
```
When she said, "Don't waste your words, they're just lies,"
I cried she was deaf.
And she worked on my face until breaking my eyes,
Then said, "What else you got left?"
It was then that I got up to leave
But she said, "Don't forget,
Everybody must give something back
For something they get."

I stood there and hummed,
I tapped on her drum
And asked her how come.
And she buttoned her boot,
And straightened her suit,
Then she said, "Don't get cute."
So I forced my hands in my pockets
And felt with my thumbs,
And gallantly handed her
My very last piece of gum.

She threw me outside,
I stood in the dirt where ev'ryone walked.
And after finding I'd forgotten my shirt,
I went back and knocked.
I waited in the hallway, she went to get it,
And I tried to make sense
Out of that picture of you in your wheelchair
That leaned up against--

Her Jamaican rum
And when she did come, I asked her for some.
She said, "No, dear."
I said, "Your words aren't clear,
You'd better spit out your gum."
She screamed till her face got so red,
Then she fell on the floor,
And I covered her up and then
Thought I'd go look through her drawer.

And when I was through
I filled up my shoe and brought it to you.
And you, you took me in,
You loved me then, you never wasted time.
And I, I never took much,
I never asked for your crutch
And I don't ask for mine.
```

```
[  29   35   86   22  930   18  329  386   24  676    2  460   35   30 2314
    4   35 2112   12    9  174  354 1132    9  125  112   86   44  365    3
   34  172   11   30  112   13    2   34   47    5  203   21   35   86   22
  324  424  173  137  152   78   19  152   50   61    2  800   66    4 3705
    2 4921   12   43 2606    4  778   43   82   53    4   35 4922   43 3019
    4 4923   43 1662  112   35   86   22   61 2607   26    2 3020    9  394
    8    9 1780    4  524   25    9 4924    4 4925 3706   43    9  382  146
 1663   10 3707   35  853    7  581    2  800    8    1  963   74 4926  398
    4  178 2608  131 1133    9 2609    2  325   78    4 1664    2 1557    8
    1 2610   35  325    5   61   11    4    2  357    5   91 2113   48   10
   13 1388   10    3    8   18 4927   13 4928   47  801   43 4929 3021    4
   29   35  140   53    2  778   43   19  113   35   86   31  714    2   86
   18  329 1920  779  326  144 4930   48   18 3707   35 4931  317   43  174
   34   26  537  112   35]
```

***Figure 11.*** *Sample song before and after preprocessing.*

## IMPLEMENTATION

I use Jason Brownlee's blog post on sequence classification with LSTM recurrent neural networks [5] as a guide for implementing my own LSTM. Much like in his post, I start with a simple model and then add complexity to see if it improves performance. The figure below shows the code for the first iteration of the model.



```
"""Code adapted from http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/"""

import numpy as np
np.random.seed(42)
import pickle

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding

from sklearn.model_selection import train_test_split

with open("./preprocessed_data.pkl", "rb") as f:
    X, y, index, _, _ = pickle.load(f)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

embedding_vector_length = 8
model = Sequential()
model.add(Embedding(len(index), embedding_vector_length, input_length=200))
model.add(LSTM(20))
model.add(Dense(5, activation="softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
print(model.summary())
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
```

***Figure 12.*** *Code from the first iteration of the model.*

The LSTM model itself is implemented with Keras, and this iteration consists of an embedding, LSTM, and dense layer. I choose to use word embeddings of vector length 8, and my LSTM layer contains 20 nodes. The dense layer uses Softmax activation. The model is trained for 10 epochs with a batch size of 32, and minimizes categorical cross-entropy loss using the default Adam optimizer.

I found that tuning parameters doesn't change validation accuracy that much, but does affect how quickly the model overfits. This version achieves a maximum accuracy score of 33.86% on the validation set. It starts overfitting after the fifth epoch. As I expected, a more complex model like the one found in Brownlee's article quickly overfits to the training data and doesn't achieve good accuracy on the validation set.

## Refinement

The second iteration of my model includes dropout. I add 20% dropout on the input to the LSTM layer (from the embedding layer) and 20% dropout on the LSTM layer's output (to the dense layer). I also increase the number of nodes in the LSTM layer from 20 to 24 to compensate for the reduced capacity of the network during training. Furthermore, following the advice from another blog post by Brownlee regarding dropout [6], I increase the learning rate of the Adam optimizer by an order of magnitude (from 0.001 to 0.01) and add a decay rate of 0.05. The figure below shows the snippet of code where these changes occur.

```
embedding_vector_length = 8
model = Sequential()
model.add(Embedding(len(index), embedding_vector_length, input_length=200))
model.add(LSTM(24, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(5, activation="softmax"))
opt = optimizers.Adam(lr=0.01, decay=0.05)
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
print(model.summary())
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
```

***Figure 13.*** *Changes in the code for the second iteration.*

Dropout is a great technique for improving generalization accuracy, but the figure below from Srivastava et al. [7] shows us that dropout could actually increase classification error if the dataset is not large enough.

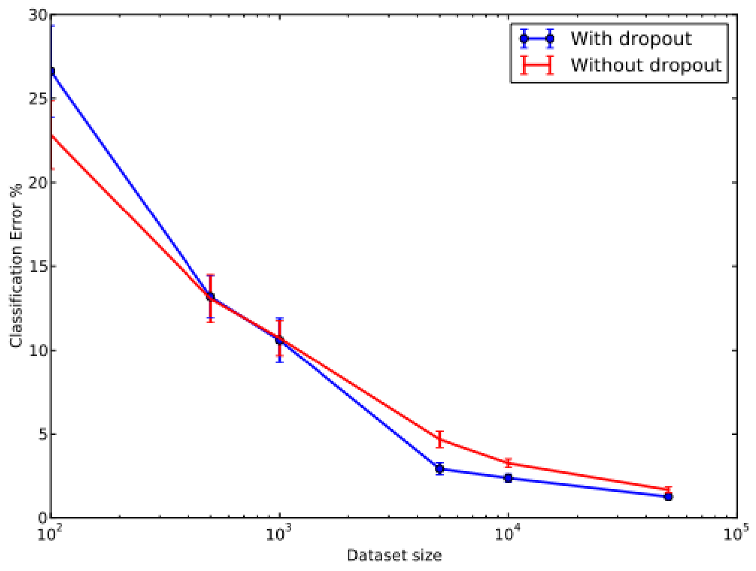


***Figure 14.*** *Effect of dropout.*

This version of the model again achieves a maximum accuracy score of 33.86%, and starts overfitting after the fourth epoch. Based on Figure 6, we most likely don't have a large enough dataset for dropout to help generalization. While the validation accuracy doesn't change much from the first iteration, I would err on the side of caution and not utilize dropout for this problem.

For the third and final iteration of the model, I add a one-dimensional convolutional layer with max pooling before the LSTM layer and remove dropout. The convolutional layer uses a filter size of 8, a kernel size of 3, same padding, and ReLU activation. The max pooling layer uses a pool size of 2. Once again, the figure below shows these changes in the code.

```
embedding_vector_length = 8
model = Sequential()
model.add(Embedding(len(index), embedding_vector_length, input_length=200))
model.add(Conv1D(filters=8, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(20))
model.add(Dense(5, activation="softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
print(model.summary())
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
```

***Figure 15.*** *Changes in the code for the third iteration.*

Brownlee [5] argues, "Convolutional neural networks excel at learning the spatial structure in input data… the CNN may be able to pick out invariant features… [these] learned spatial features may then be learned as sequences by an LSTM layer." The model achieves a maximum accuracy score of 34.92%, and overfits after the fourth epoch. Interestingly, validation loss is the highest out of either past iteration, and even starts increasing after the fifth epoch although validation accuracy doesn't drop. I would not consider this an improvement.

## IV. RESULTS

### MODEL EVALUATION AND VALIDATION

For my final model, I use the architecture from the first, simplest version of the architecture, trained for 5 epochs instead of 10 to avoid overfitting.

Recall that during testing, this model achieved a maximum accuracy score on the validation set of 33.86%. In order to test the model's sensitivity to changes in input, I run a 5-fold cross-validation. The model achieves an average accuracy score of 28.53%, with a standard deviation of 2.76. The drop in accuracy implies that there is sensitivity to the input, and that during the refinement stage, there was some overfitting to the validation set. The table below shows the results from each of the five folds.

| Fold | Training Loss | Training Accuracy | Validation Loss | Validation Accuracy |
|---:|---:|---:|---:|---:|
| 1 | 1.48 | 34.62% | 1.57 | 26.98% |
| 2 | 1.50 | 31.30% | 1.74 | 30.16% |
| 3 | 1.47 | 33.95% | 1.58 | 24.34% |
| 4 | 1.49 | 30.73% | 1.61 | 28.72% |
| 5 | 1.40 | 36.82% | 1.54 | 32.45% |

***Table 1.*** *Results from 5-fold cross-validation of the final model.*

### JUSTIFICATION

Unfortunately, the LSTM model does not outperform the benchmark SVM model, which scores 42.21% in accuracy, versus the final LSTM model's 28.53%. The final results are only slightly better than random guessing, which would have an expected accuracy score of 20%.

These results prove that an LSTM model is *not* a good choice for this type of problem. Neural networks generally need large datasets to perform well, and the dataset for this problem is most likely too small for the neural network to learn anything meaningful. On the other hand, SVMs tend to work well with smaller datasets, which is why we see good performance in the benchmark.

## V. CONCLUSION

### FREE-FORM VISUALIZATION

The figure below shows the confusion matrix of the final model's predictions from the last validation session.

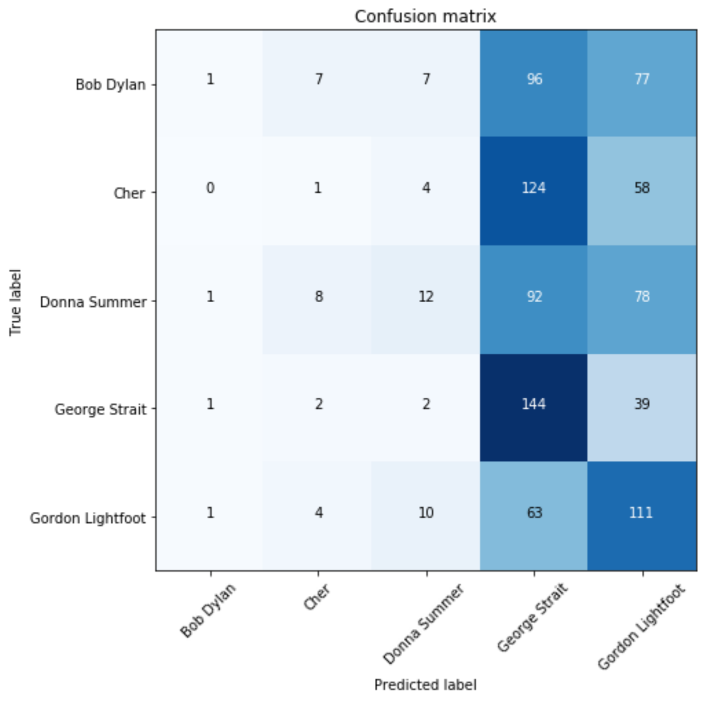


***Figure 16.*** *Confusion matrix for the final model.*

Immediately, there are a few things that stand out. We can see that the model is clearly the best at predicting "George Strait" and "Gordon Lightfoot", but it also often incorrectly predicts these labels, meaning it has high recall but low precision for these two particular classes. It almost never predicts

"Donna Summer", "Cher", or "Bob Dylan". Recall that the benchmark model is also very good at predicting "George Strait", implying that there is some defining characteristic about his songs that both models can learn well. However, the benchmark is the best at predicting "Donna Summer", while the LSTM model fails to predict her most of the time.

## REFLECTION

To summarize the entire project, I decide to use a Long Short Term Memory (LSTM) neural network to classify songs by artist based solely on the song lyrics. I decide to use only the songs from the top 5 artists with the most songs in the entire dataset in order to get meaningful results. As a benchmark, I transform the input using the bag of words technique and then train an SVM classifier, achieving an accuracy score of 42.21%.

Before building the LSTM, I preprocess the dataset by transforming each word into a number representing its frequency rank among all words present in the entire dataset. Each song is then truncated or padded to a fixed length.

I use Keras to implement my LSTM model. For the model's first iteration, I use an Embedding layer, followed by an LSTM layer and finally a Dense layer. For the second iteration, I experiment with adding dropout to both the input and output of the LSTM layer. For the third and final iteration, I remove dropout and add convolutional and max pooling layers before the LSTM layer.

Because each successive iteration does not improve results (and sometimes results in worse results), I use a variation of the first iteration for my final model in favor of simplicity. The final model scores 28.53% in accuracy – significantly worse than the benchmark.

The most interesting aspect of this project was learning how LSTM networks work. Chris Olah's article [4] about the subject is extremely well-written and was invaluable in understanding a potentially confusing topic.

The most difficult part of the project was tuning each version of the model to achieve optimal parameters. This involved doing research about neural networks in general to find out what values work well depending on the type of problem, as well as much experimentation.

Unfortunately, the final model did not meet my expectations for this problem. I knew that I had a relatively small dataset to work with, but I was hopeful that the LSTM would at least outperform the benchmark. Needless to say, an LSTM is not the right choice for this particular problem where the dataset is so small; however, they are proven to work well on sequential data like text when there is enough data available.

## IMPROVEMENT

It's difficult to think of any ways this solution to this particular problem can be improved. The main issue lies in the size of the dataset, and the size is fundamentally limited by how many songs artists write. In other words, there's no way to just "collect more data".

If we try to imagine ourselves trying to classify songs by artist based on song lyrics alone, it would be a very difficult task. So then, how are our brains able to know who wrote a song we've never heard before, provided we're familiar with that particular artist's other material? Oftentimes, we're able to recognize the singer's voice, but that's not always applicable, especially when the song is purely instrumental. Most artists have a distinctive style that can be picked out from the instruments they use, the melodies they construct, and a whole lot of other subtle attributes best left to a music critic to describe. If we could somehow transform this "musical signature" into numbers – something a neural network can understand – then maybe we can get even better results than Sadovsky and Chen did; however, this is a monumentally difficult problem and far outside the scope of this project. Maybe there will be a better type of neural network for this problem in the future, but for now let's stick to SVMs.

## REFERENCES

[1] "Natural language processing," [Online]. Available: https://en.wikipedia.org/wiki/Natural_language_processing.

[2] A. Sadovsky and X. Chen, "Song Genre and Artist Classification via Supervised Learning from Lyrics," 2006. [Online]. Available: https://nlp.stanford.edu/courses/cs224n/2006/fp/sadovsky-x1n9-1-224n_final_report.pdf.

[3] "55000+ Song Lyrics | Kaggle," [Online]. Available: https://www.kaggle.com/mousehead/songlyrics.

[4] C. Olah, "Understanding LSTM Networks," [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[5] J. Brownlee, "Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras," [Online]. Available: http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/.

[6] J. Brownlee, "Dropout Regularization in Deep Learning Models With Keras," [Online]. Available: http://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/.

[7] N. Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* no. 15, pp. 1929-1958, 2014.