COMS W4995 Deep Learning Project
Application of BERT Model to Genre and Author Identification of Song Lyrics

Jesse Cahill [*1], **Naoto Minakawa** [*2], **Yannis Tzemos** [*3]
[*] Data Science Institute, Columbia University
500 W 120th St, New York, NY 10027
[1] jc4673@cumc.columbia.edu [2] nm3086@columbia.edu [3] it2280@columbia.edu

Abstract

Pre-trained neural Language Models (LM) such as BERT have recently created significant advances in the state of the art in such NLP tasks as question answering, natural language inference and document classification. Classification of song lyrics by genre or their attribution to an artist is a difficult task with many challenges and modern machine learning methods achieve less than ideal results. In addition, modern machine learning methods have not yet been applied at scale to classify songs by artist for a large pool of artists. Our main aim is to identify the genre and artist of a song, given its lyrics, by fine tuning a BERT language model for document classification on these two tasks. We showed that BERT moderately outperformed all other document classification approaches for both genre and artist classification. Finally, in an effort to enhance our Bert implementation, we are combining it with a text summarization technique (BertSum (Liu 2019)) on the lyrics and compare our results.

Introduction

Classifying music into genres is a difficult task in Music Information Retrieval, largely due to the subjectivity in the definition of a genre itself. Some genres are more defined by the way the music itself sounds, such as Jazz and Pop. Others, like Punk, are more about the message and feel of the music and lyrics. And others yet, like Hip-Hop and Country, represent a blend of the two. The inherent ambiguity in the human process of classifying a song into a genre makes this a difficult machine learning task. Learning how to perform it well presents the potential to learn more about how we as humans form genres and use them to classify music.

Related Work

Significant progress has been made in classifying music into genres using audio signal content, usually on midi songs. Basili et al. (Basili, Serafini, and Stellato 2004) used derived features such as melodic intervals and meter/time changes. After the resurgence of neural network popularity, Sigtia et al. (Sigtia and Dixon 2014) achieved 83% accuracy for a 10 genre problem. These are impressive results, but classifying music into genres based on lyrics is a difficult task even for a human, as genres sometimes have overlap in tone and language and are often distinguished more by the sound of the music itself. Several efforts have been made using machine learning relying on word embeddings such as word2vec and GloVe, as well as doc2vec for documents. These models embed words or documents into an arbitrary vector space which on some level captures the meaning of words. Barman et al. (Barman et al. 2019) used such embeddings followed by a number of different machine learning algorithms including traditional techniques such as KNN and random forests as well as CNNs and residual neural networks with GRUs, to some success. Their most successfull method was the simplest, called the "Genre Vector" approach: they transformed all their training data to vectors using doc2vec, and created a mean vector for each genre. They then classified new data by evaluating the cosine distance to each average genre vector. Tsaptsinos (Tsaptsinos 2017) used Hierarchal Attention Networks (HANs), which model the hierarchy of songs using attention. These methods exploit the hierarchal nature of documents, which in this case are songs: a song consists of stanzas (or segments); a stanza consists of lines; a line consists of words. He didn't achieve accuracy comparable to Barman et al. (Barman et al. 2019), but was working with datasets of 20 and 117 genres compared to their 6.

Much less work has been done in the field of authorship attribution for music. This may be because it is difficult to formulate as a classification problem: there is a massive number of recording artists, many with very few songs. Due to these restrictions, any effort to attribute a song to an artist must restrict its dataset to a reasonable number of artists and would do well to choose artists with a large discography to maximize the amount of training data. But an artist can only produce so much music, so there is an inherent difficulty in collecting a large enough dataset. Buzic et al. (Buzic and Doba 2018) used a simple bag of words model with a Naive Baye's classifier solve a 2 class lyric attribution task for songs by Nirvana and Metallica. They achieve an F1 score of 0.94, but on a small scale problem with two bands that have very distinct lyrical styles. Eghbal-zadeh et al. (Eghbal-zadeh, Schedl, and Widmer 2015) used feature extraction followed by Linear Discriminant Analysis on

audio files to achieve 84% accuracy on a dataset of 20 artists. No efforts have been made using modern deep learning architectures, likely because it isn't possible to obtain enough data to support them.

Problem formulation

The field of NLP has been undergoing rapid development in recent years. The trend towards language models trained on huge corpuses of unlabeled data has led first to ELMo and then BERT, which many are describing as "NLP's ImageNet moment". These models, particularly BERT are trained to promote versatility, and have advanced the state of the art on tasks such as question answering, language inference, and document classification. Moreover, BERT provides publically available pre-trained models that can be re-purposed towards more specific tasks through transfer learning. We need only to add a final layer to the model corresponding to our desired output and fine-tune end to end Devlin et al. (Devlin et al. 2018). This allows us to leverage the massive corpus BERT is trained on for our specific task. We believe that the word context that BERT provides will lead to impressive results in genre attribution. We also believe that transfer learning will allow us to utilize BERTs complex neural architecture despite a relatively small dataset.

Method and Implementation

Our implementation will use the methods outlined by Adhikari et al. (Adhikari et al. 2019) to adapt the pre-trained BERT model, changing the structure of the network to support document classification by introducing a fully-connected layer over the final hidden state and optimizing the entire model end-to-end during fine-tuning. We will implement DocBERT with Hedwig, an open source repo containing PyTorch deep learning models for document classification. Tuning hyperparameters include number of epochs, batch size, learning rate, and maximum sequence length (MSL). Fine-tuning will be done separately for genre and artist attribution. Each song will be tokenized as a document; one document will correspond to one data point with a genre and artist as labels for their specific tasks. For the authorship attribution task we will randomly select 100 artists several times to maintain a reasonable classification problem, reporting the average of the results. For the genre task we will use all available genres.

Dataset

One of the initial challenges of our project was the accumulation of a clean and accurate dataset. We compile our dataset through a concatenation procedure of two datasets mined by Kaggle community as part of their "Kaggle Datasets" initiative. The first one, "380,000+ Lyrics from MetroLyrics", is a collection of approximately 380 thousand songs and includes features like lyrics, genre and year. The second one, "55,000+ SongLyrics", is a collection of a roughly 55 thousand songs and their corresponding links, lyrics. The dataset that we create and use, utilizes only the subset of the following features: song, artist, genre, and lyrics. In the following sections, we describe the exact architecture of our pipeline and the way data are fed from one step to the next.

System Architecture and Design

Following is system architecture diagram in regards to our implementation:

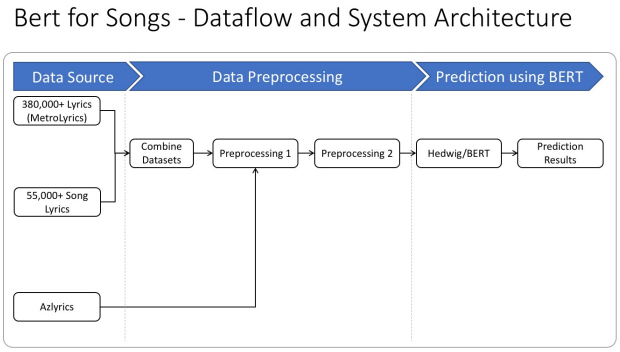

Figure 1: Architecture Diagram

Data Gathering (Pre-processing - Step 1)

As mentioned earlier, one of our initial tasks was to create a clean dataset. Concerning the data gathering step, this was proven to be a difficult procedure as lyrics are governed by copyright and distribution laws. Our problem can be reduced to the fact that the "380,000+ Lyrics from MetroLyrics" dataset is missing lyrics for approximately 26% of its records while the "55,000+ SongLyrics" is missing the genre feature for all of its records. Genius is a media company that aims to freely provide metadata about songs, artists and albums. They collect data on all writers, producers and other contributions to a piece of music, as well as lyrics and artists comments. Genius provides a robust and easy to use API that we used to impute the missing lyrics values for the Kaggle MetroLyrics dataset. As of yet, we were not able to find an equivalent API to manage to update genres for the "55,000+ SongLyrics" dataset, so for now we will choose to use separate datasets for each task.

Aside from the lyrics update process, our initial pipeline implements the following series of procedures before reaching the NLP part of the process. As a first step, we subset the dataset by removing the songs that have lyrics in a language other than English. This was made possible via Python's *langdetect* package which implements a non-deterministic language detection algorithm. Furthermore, we filter out updated lyrics through Genius that we know to be problematic. In order to produce a clean dataset, we remove any duplicates that may occur from the concatenation procedure of the two datasets.

For the genre classification task and our first dataset, we subset the MetroLyrics dataset with imputations from Genius to return a dataset without any missing genres or lyrics. For the lyrics to artist classification task, we will use the

concatenation of the two datasets and keep only the non-empty lyrics subset. The cleaned version of the dataset contains 14,904 unique artists and eleven genres: Pop, Hip-Hop, Rock, Metal, Country, Jazz, Electronic, Folk, RB, Indie, and Other. We will remove all songs classified as "Other" for the genre classification task, as we anticipate it would be difficult to learn a class which is likely an amalgam of many unrelated smaller or difficult to precisely classify genres. The relative frequencies of these genres in the pre-processed data are shown in the table below:

Genre	number of records
Country	15,204
Electronic	9,288
Folk	2,049
Hip-Hop	27,722
Jazz	9,821
Metal	23,523
Pop	37,995
R&B	4,403
Rock	109,082
Indie	4,124
Other	6,914

Basic Natural Language Processing (NLP) (Pre-processing - Step 2)

In the NLP stage of our pipeline's pre-processing, we process further transform the data for compatibility with the Hedwig library implementation. We process and modify the original data to Hedwig specific format, described below.
In the Hedwig library data consists of 2 columns: one is for class and the other for text string. The class column should be formatted as an one-hot vector. More specifically, if we have class yes, neutral and no, the one-hot vector should be 100 for yes and 010 for neutral and 001 for no. For our implementation, in our datasets, this one-hot vector method will be used for representing artist or genre (depending on the task) and the text string would be the song's lyrics.
Regarding the processing of original data, we remove all the song structure words (i.e. [Chorus :], [Verse 1:], Intro, Outro, etc.) from the lyrics of the dataset, as well as any indication of featured artists when the songs have guest vocalists. On top of that, original lyrics texts contain "\n", "\r", "\t", and so on. We removed such character as well. This is to remove all the non-lyrics related word from the original data.
Further lexical steps of pre-processing, such as lower case and tokenization can be achieved by using pre-built functions in Hedwig. However, the purpose of pre-processing at this stage is to clean the data to some extent so that succeeding pre-processing code built into Hedwig functions well on the data.
Regarding Naive Bayes baseline and Doc2Vec approach, we built methodology independent of hedwig. In regards to NLP related processing such as tokenization, stemming, lemmtization, stopwords, we used Python's *nltk* package for those implementation.

Definition of Evaluation Criteria

Our principal evaluation metrics will be precision, recall, and F1 score. We will not evaluate accuracy because our data has unbalanced classes. As this is a multiclass classification problem, we needed to choose a way to create metric averages across classes. The two most common options are a macro average which weights all classes evenly, and an average weighted by the number of records belonging to each class. We chose to use a weighted average as we believe that the model should not be penalized equally for worse results in small classes which are very difficult to predict. Metrics are also computed for a random guessing case for context. We will evaluate our method in two ways:

Against Previous Results in Similar Studies For genre attribution, we will compare against results of two previous works. The first is an implementation of Hierarchical Attention Networks (Tsaptsinos 2017). The second uses Doc2Vec for embedding, followed by the "Genre Vector" approach mentioned in the introduction.
Author attribution is a less explored task for lyrics. We will compare against Buzic et al. (Buzic and Doba 2018), who used Naive Bayes with a bag of words model. Theirs was a binary classification problem and we plan to classify lyrics of 100 artists, so we do not expect to reach similar performance.

Alternative Methods with Our Dataset Hedwig contains several alternative algorithms for document classification which we will measure the effectiveness of our DocBERT approach against:

- Reg-LSTM: Regularized LSTM for document classification (Adhikari et al. 2019) (Liu 2019)
- HAN: Hierarchical Attention Networks (Yang et al. 2016)
- "Genre Vector" (Barman et al. 2019)

We were not able to replicate the results of the "Genre Vector" approach as described by Barman et al. (Barman et al. 2019) and their paper didn't provide sufficient detail to troubleshoot our implementation. Instead we decided to use a Doc2Vec embedding followed by a k-NN classifier.

Results

First and foremost, we conducted genre identification for 10 genres with BERT and other baselines. Second, we conducted artist identification: we started from 2 artists classification as a baseline, then expanded the problem to 25 artists classification. Regarding 25 artist classification, we adopted several methodologies used in 10 genres identification such as HAN and LSTM for baseline comparison. Also, we made sure that the results we obtained outperformed random prediction by implementing a classifier that produced labels randomly with uniform probability.

10 Genres Identification

We measured each methodology's performance for identifying 10 genres using 227,949 songs. We split 227,949 songs into train, test, and dev datasets with 80%, 10%, 10%

portion, respectively.
We ran the respective models for 2 epochs. This is to keep consistency with the baseline (Tsaptsinos 2017); they run HAN and LSTM for 2 epochs.
We can see all of BERT, HAN, LSTM, and Doc2Vec approaches are much better than random prediction. Compared with HAN, LSTM, Doc2Vec as a baseline (Tsaptsinos 2017). BERT performs better in all criteria; Similar to HAN performance. Regarding Doc2Vec approach, as aforementioned, we decided to use a Doc2Vec embedding followed by a k-NN classifier.

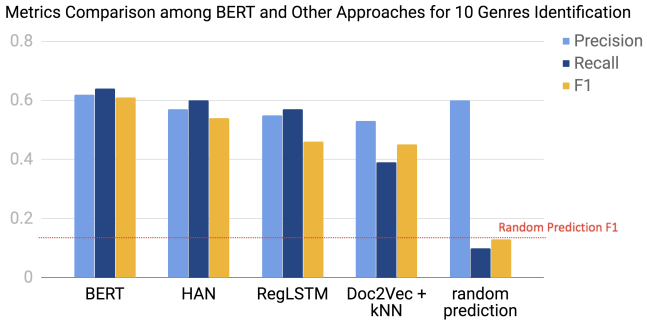

Figure 2: 10 Genres Identification

To interpret what information BERT was able to capture from our 10 genres we inspected the normalized confusion matrix output. The most initially striking conclusion is how heavily rock was favored over all other genres. This was likely caused by the large class in-balance in Rock's favor: nearly 45% of our data is the Rock genre. This is also because we suspect the Rock genre to be one of the most broad in terms of lyrics: subgenres such as soft rock, hard rock, and alternative rock are likely all contained within this large genre and their lyrics are often very stylistically different. We believe this ambiguity in the Rock genre combined with its disproportionate class size have led to it being overclassified. Indie, for example, was never predicted correctly and was predicted as Rock 86% of the time, which suggests that perhaps Indie as a genre could be absorbed into rock in this dataset.

2 Artists Identification

As a baseline for artist classification, we implemented the same approach as Buzic et al. (Buzic and Doba 2018). In this paper, they used Naive Bayes approach combined with bag of words vectors to identify artist given lyrics. Specifically, they obtained 207 songs (127 for Metallica songs and 80 songs for Nirvana songs) from the website Azlyrics.com. After obtaining the songs, they processed the lyrics accordingly. They first removed numeric characters, punctuation, stopwords. Thereafter, they applied stemming. As a total number of dictionary, they had 2,932 terms for 207 songs after preprocessing. After preprocessing, they remove less frequent words which appeared less than 8 times. Finally, they created document-term matrix. They had 69 results for

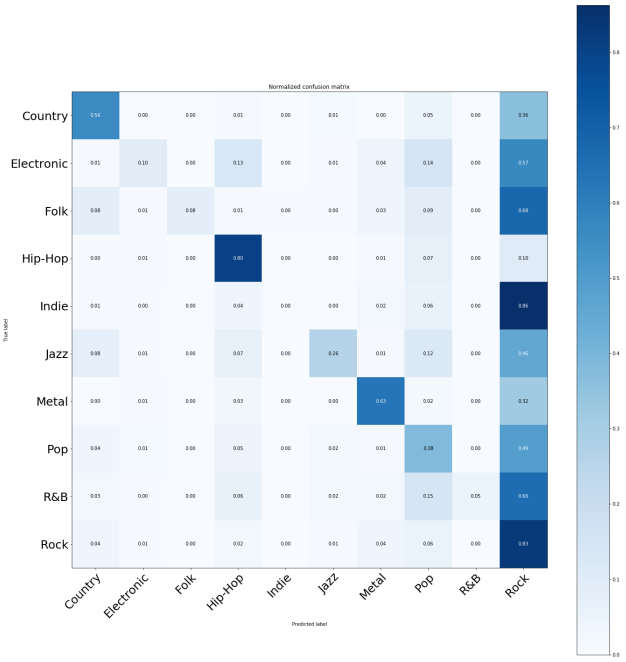

Figure 3: Normalized Confusion Matrix for 10 Genre Classification

testing, which is about 1/3 of 207 songs. They used 2/3 for training datasets. As a result, they get 90% accuracy and F1 scores as performance.
We implemented the same approach. We obtained 127 for Metallica songs and 75 songs via Azlyrics API. Since there were some limitation issues in terms of the number of request call we can make, we could not obtain 80 songs for Nirvana. Thereafter, we followed the paper to conduct same preprocessing; we first applied same preprocessing functions to clean up lyrics. Second, we removed numeric characters, punctuation, and stopwords. Then, we applied tokenization and stemming. We used *nltk* library to apply NLP techniques. Finally, removed less frequently appeared word which appears less than 8 times from doc-term matrix as they mentioned in the paper. As a result, we obtained 80% accuracy and F1 score. However, we could not achieve 90% of accuracy and F1 scores. There appears to be some differences regarding how they obtained data, what kind of libraries they used, so we concluded our result is valid to use as a baseline. The results showed small fluctuation for different runs, we obtained 0.68 precision, 0.85recall, and 0.75 F1 score.
BERT outperforms Naive Bayes results in terms of all metrics; We obtained 0.93 precision, 0.81 recall, and 0.87 F1 score. precision for random prediction is about 0.6 for 10 genre identification. This is because genre data is highly skewed; i.e. There are many data points for Rock ($\tilde{1}$10,000 records), on the other hand, there are few data points for Folk ($\tilde{2}$,000 records).

25 Artists Identification

This is much harder problem compared to the previous 2 artists classification problem. We conducted 25 Artists Classification for 32,703 Songs, running for 20 epochs. All of BERT, HAN, LSTM approaches significantly outperform random prediction. We initially started from running for 2 epochs to maintain consistency with 10 genres classification, however, it did not perform well with 2 epochs. Therefore, we experimented to increase the number of epochs.
BERT showed well-balanced results as well as performed modestly better than HAN and RegLSTM. All of BERT, HAN and RegLSTM outperformed random prediction results.
Regarding interpretation of random prediction, whereas precision for random predicition was about 0.6 for 10 genre identification, precision for random predicition was almost 0 for 25 artists classification. We concluded this is because of relatively equally distributed characteristics of artists data.

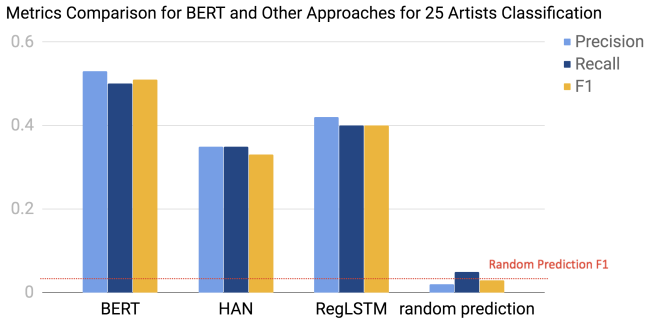

Figure 4: 25 Artists Identification

Lyrics Summarization

In order to further explore our initial problem, we decided to include an additional step between the pipeline and BERT implementation. A summarization layer was introduced to try and reduce the amount of computational time for the system and to explore whether a summarized dataset was sufficient enough to produce the same kind of results or comparable to the the original ones. A summary of a document can either be extractive or abstractive. We used three extractive methods that seek for relevant phrases within the input document and concatenate them to form a summary. We focused on this type of summarization because an abstractive one is by definition a more difficult task since it tries to generate a summary that also keeps the original intent of the document. In addition, we decided to focus on summarization methods that provide a scalable complexity and as a consequence running computational time.

As a first method, we used TextRank (Mihalcea and Tarau 2004), a graph-based ranking algorithm implementation. TextRank operates on words/tokens that perceives as entities and represents as nodes. It utilizes a graph representation of their interconnections as its base, which can be directed or

Method	Example
TextRank	Dear heart I wish you were here to warm this night And dear heart it seems like a year since you ve been out of my sight A single room a table for one its a lonesome town alright But soon Ill kiss you hello at our front door And dear heart I want you to know Ill leave your arms never more A single room a table for one
LexRank	When I cried over you And would always be true When I cried over you I wasted my tears When I cried over you I shouldve known You would never be true I loved and I lost you Now Im so blue
BertSum	sad just to hear you lie all that you say youre not this time it was your fault you forget what you just said dont lay the trip on me i would be a bridge to both

Table 1: Example of Summarizations per Algorithm

un-directed, weighted or unweighted. Using this graph representation, it can then produce a keyword extraction summary or a sentence one. We used the latter to produce a dataset that was then used as input to BERT. In a similar way, LexRank (Erkan and Radev 2011), our second method implements a stochastic graph-based approach in an effort to identify the relative importance of a predefined text unit, which in our case is sentences. Once more, using the graph we get the summary of the document that acts as input to BERT. Finally, we used a BERT implementation for Extractive Summarization (Liu 2019). This more complex method extends BERT to the point that it is able to handle multiple sentences and then uses three summarization layers of a Classifier, a Transformer and a Recurrent Neural Network to produce the summary. This is the final input that we tested on our own BERT implementation.

In Figure 6, we present our BERT model's performance after a summarization step is introduced between it and the pipeline. Upon producing Figure 6, we were able to draw the following conclusions.
First of all, it became clear to us that the summarization step did not help the performance of our implementation. Although, we had an expected reduction in its running time BERT's initial results were better. This may be partly due to the fact that we chose to use the predefined configuration and setup of each one of the summarization methods, even though they offer a wide variety of options. In addition, implementing summarization techniques on top of our datasets may very well interfere with the bidirectional aspect of BERT's approach and this is something that we should take into consideration. Furthermore, summarizing lyrics can be a elaborate task in the sense that there is no actual method to set an acceptable sentence barrier, thus making trial and error our only option. Finally, after feeding our data through the summarization techniques, we had to respect a various number of cut-offs and reduction steps that

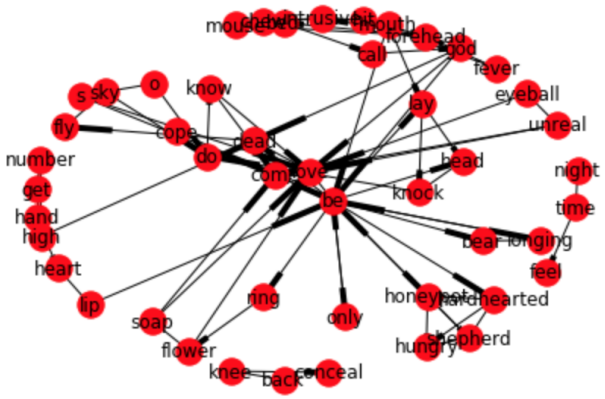

Figure 5: Example of Text Rank Graph
The above figure is a graph based representation of the lyrics of a song from our dataset. It is produced by the *pytextrank* package which is a TextRank algorithm representation. Its nodes are equivalent to words from the lyrics of the song and its edges are their interconnections based on their occurrence in the song. The bold part of the edges indicate the weights of the graph.

they were introducing as part of their implementation, thus reducing our available dataset.

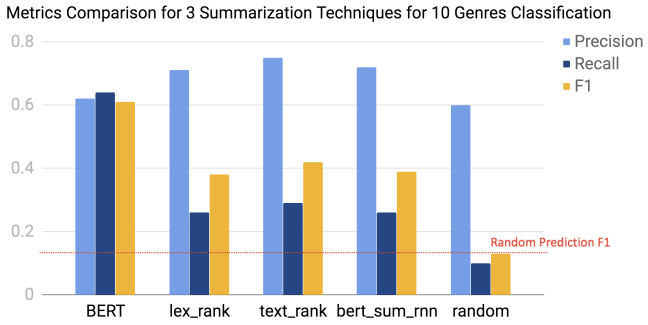

Figure 6: 10 Genres Identification with Lyrics Summarization Step Implemented

Although, a summarization step between our BERT implementation and the pipeline did not fully achieved our initial goal, more extensive testing should be done before clearly rejecting the notion. A glance at Figure 6, clearly shows that our results from the summirization step are performing a lot better than random prediction. A step towards the right direction, would be a trial and error method or a robust search over the desired configuration of the summarization methods. This will be beneficial since with experimentation, the correct summarization method with its own optimal configuration may in turn help our BERT model achieve better results while keeping the reduced running time.

Conclusion

Overall, BERT modestly outperformed related work in terms of precision, recall, and F1 score for both genre identification and author identification. Regarding artists classification, performance got worse as we increase the number of artists to identify. This result is consistent with related work and as expected. As later mentioned in following future work section, we may be able to improve performance by accommodating with suggested work. Our initial thoughts about the application of our model is to use it as an information retrieval system. For example, our initial challenge was to find data that are integrated enough to run our experiments and produce our results. Even through our data retrieval process was extensive, sometimes songs were missing genres, artists information or had mislabeled genres. To solve this problem we may be able to implement our method and fix our data issues. On top of that, BERT is a versatile model that we may even be able to apply as a synthesizing tool for song lyrics.

Future Work

Mainly due to time constraints, we did not have chance to try the following things: First and foremost, we can improve data preprocessing. We could not subset data thoroughly to obtain only English songs. One step towards that direction might be to reevaluate the way we define the non English songs that we remove or to create a web scraper so that we can reduce the missing genre values to potentially zero. We observed some songs have duplicated or multiple counts due to featuring artists or remixes. These phenomenon are often seen for hip hop songs. There may be still irregular characters included in lyrics. Such irregular character are used with no specific pattern. It is difficult to remove all such characters. Although these preprocessing might be out of the scope of the class, we regard it important to improve performance.
Second, we can try running for more epochs. We observed performance was often improved as we run more epochs. It took $\tilde{3}$ hours to run BERT for 1 epoch on Google Cloud. Therefore, we could not run for many epochs.
For similar reason, we did not have opportunity to run BERT large model or optimize hyperparamters for BERT. With all these things combined, we expect BERT's performance would be enhanced.
Finally, BERT with text summarization suggested potential to enhance time efficiency. Whereas normal BERT took $\tilde{3}$ hours to run, it took less time to run BERT with text summarization. We observed some trade-off by summarizing text, which we concluded as losing some of context information from original text. However, it would be worth considering this approach.

Appendix

Project Code Repository

Please refer to the following github link for code repository regarding our project:

https://github.com/yannistze/Deep-Learning-Project
https://github.com/j-cahill/hedwig

References

Adhikari, A.; Ram, A.; Tang, R.; and Lin, J. 2019. Docbert: BERT for document classification. *CoRR* abs/1904.08398. http://arxiv.org/abs/1904.08398.

Barman, M. P.; Dahekar, K.; Anshuman, A.; and Awekar, A. 2019. It's only words and words are all I have. *2019 European Conference on Information Retrieval. ECIR 2019. CoRR* abs/1901.05227. http://arxiv.org/abs/1901.05227.

Basili, R.; Serafini, A.; and Stellato, A. 2004. Classification of musical genre: a machine learning approach. 5th International Conference on Music Information Retrieval. https://www.researchgate.net/publication/220723746_Classification_of_musical_genre_a_machine_learning_approach.

Buzic, D., and Doba, J. 2018. Lyrics classification using naive bayes. 1011–1015. 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). https://ieeexplore.ieee.org/document/8400185.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. NAACL-HLT 2018. CoRR* abs/1810.04805. http://arxiv.org/abs/1810.04805.

Eghbal-zadeh, H.; Schedl, M.; and Widmer, G. 2015. Timbral modeling for music artist recognition using i-vectors. 1286–1290. 2015 23rd European Signal Processing Conference (EUSIPCO). https://ieeexplore.ieee.org/document/7362591.

Erkan, G., and Radev, D. R. 2011. Lexrank: Graph-based lexical centrality as salience in text summarization. *CoRR* abs/1109.2128.

Liu, Y. 2019. Fine-tune BERT for extractive summarization. *CoRR* abs/1903.10318.

Mihalcea, R., and Tarau, P. 2004. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, 404–411. Barcelona, Spain: Association for Computational Linguistics.

Sigtia, S., and Dixon, S. 2014. Improved music feature learning with deep neural networks. 6959–6963. 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). https://ieeexplore.ieee.org/document/6854949.

Tsaptsinos, A. 2017. Lyrics-based music genre classification using a hierarchical attention network. *International Society for Music Information Retrieval Conference. ISMIR 2017. CoRR* abs/1707.04678. http://arxiv.org/abs/1707.04678.

Yang, Z.; Yang, D.; Dyer, C.; He, X.; Smola, A.; and Hovy, E. 2016. Hierarchical attention networks for document classification. 1480–1489. Association for Computational Linguistics. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. https://www.aclweb.org/anthology/N16-1174.