Victorian Era Author Classification

Giorgio Montenegro, Nicholas Synovic, Stefan Nelson

Note

Whenever possible we used 6 significant figures to represent results.

Introduction

There are over 3.5 million written books from the 1800s to 2015 that have been digitized and published freely online [1]. However, this only makes up a fraction of the ~130 million books that have been published since the invention of the printing press [2] But not all written works can be classified as books and thus lack book-level metadata, such as authorship. Thus, creating authorship attribution (AA) tooling to assist in the discovery and classification of texts by authors is critical to properly catalog humanity's growing Library of Alexandria.

To assist in this effort, we have created two different AA models using different architectures to evaluate their performance at classifying a subset of documents written by 19th-century authors [3]. These include a Recurrent Neural Network (RNN) and a Convolution Neural Network (CNN). All of our code, including how to download our pre-trained models and dataset is hosted on GitHub [4].

Dataset Description

The Victorian Era Authorship Attribution Data Set [3] is a collection of literary works from various authors from the Victorian era, dating from the early 19th century to the early 20th century. Authors include Charles Dickens, Oscar Wilde, and Bram Stoker, as well as lesser-known authors. The dataset was compiled to provide researchers with a resource to test and develop methods for authorship attribution (AA), the task of determining the author of a given text [5]. The dataset is extracted from the GDELT Historical American Books Archive database [1], a subset of the GDELT Project database [6] which is an open-source project for providing datasets to be used for various research and analysis purposes. We retrieved the dataset from the University of California Irvine Machine Learning Repository [7].

As the dataset was originally intended for benchmarking several different AA techniques [3], we modified the dataset slightly for our purposes. The original dataset contains two comma-separated value (CSV) files. One with labels for training, and one without labels for testing unsupervised methods. Stop words and punctuation were removed by the author and all text was made lowercase. We did not use the original testing data for our work and created a separate training dataset using a random stratified sample of 20% per class. This provided us with a testing dataset that included labels. Additionally, we converted all of the documents to UTF-8 text encoding before analysis. This resulted in a training dataset containing 42,942 labeled documents for training, and 10,736 labeled documents for testing. Stop words were not removed.

Baseline Approach Description

Prior to creating our RNN and CNN approaches, we created a Multinomial Naive Bayes model to classify the documents. To vectorize the data, we tried both the *CountVectorizer* and *TfidfVectorizer* provided by scikit-learn [8]. The training data was split into 85% training data, and 15% validation data.

During training, the best accuracy that we were able to achieve on the validation data was 37% using the *CountVectorizer*, and 20% using the *TfidfVectorizer* . When evaluating our model on the testing data, we achieved the following scores for accuracy, precision, recall, and the f1-score on the model using the *TfidfVectorizer*:

- Accuracy: 19.5417%
- Precision: 12.9272%
- Recall: 19.5417%
- F1-Score: 10.5286%

When evaluating our model on the testing data, we achieved the following scores for accuracy, precision, recall, and the f1-score on the model using the *CountVectorizer*:

- Accuracy: 37.1647%
- Precision: 51.8899%
- Recall: 37.1647%
- F1-Score: 30.8241%

We visualize these scores as well as the model using the *TfidfVectorizer* in Figure 1.

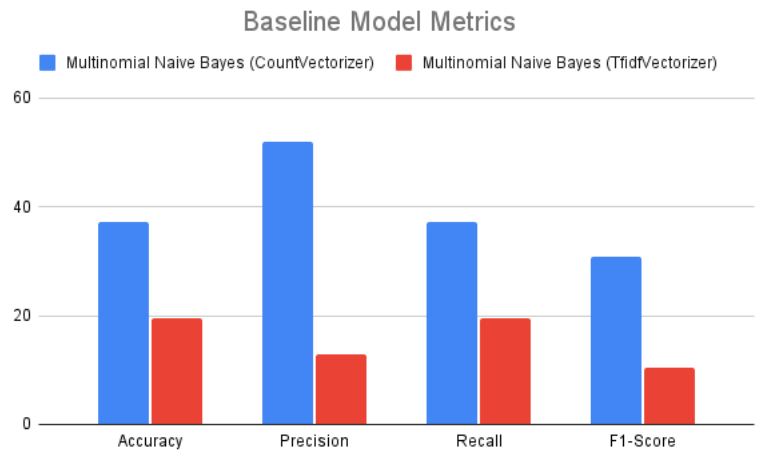

Figure 1: Visualization of baseline model metrics

As the accuracy score achieved for our best baseline model using the *CountVectorizer* was worse than random chance (50%), we decided to pursue two different neural architectures to assist in classifying Victorian Era authors: an RNN and a CNN.

Method Description

RNN

Our first model to perform Victorian Era Author Classification was a Recurrent Neural Network (RNN). We specifically utilized a Long-Short Term Memory (LSTM) RNN for our model.

In a traditional RNN model, the weights used for classification are continually updated based on a calculated gradient, which is the difference between the expected label and returned label. However, where there are many sequences in the training data and training weights become less than one, the gradient can become too small and vanish. This stops the model from learning any more. To resolve this, LSTM based RNNs limit the amount of data the model learns from to only the most significant. This prevents the gradient from vanishing.

Prior to processing the data, we split the data similar to what we did for the Multinomial Naive Bayes baseline model: 85% for training, 15% for validation. Following this, we trained a Keras [9] text *Tokenizer* on the training data split and then tokenized both the training and validation data splits, as well as the testing data, on the *Tokenizer.* We also padded all document sequences to a max length of 1000 to ensure similarity between text inputs.

Our RNN model took on the following architecture:

- An *Embedding* layer to take our tokenized text into dense vectors
- An *LSTM* layer to learn the text representations of each class with a *TanH* activation
- A *Dense* layer to return the classification probabilities given the text using a *Softmax* activation

Figure 2 visualizes our architecture as well as providing additional insights into our input and output dimensions:

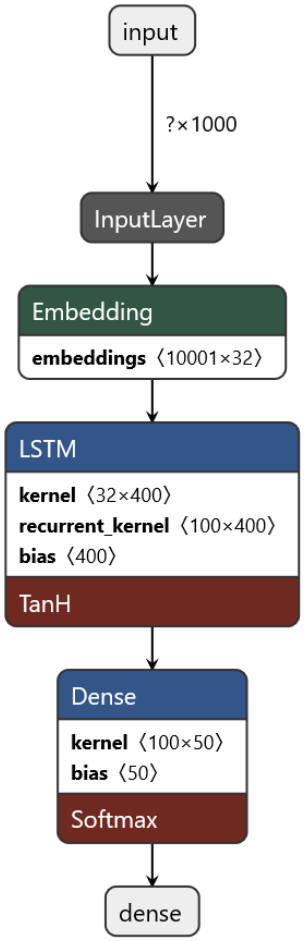

Figure 2: RNN Architecture made with Netron [10]

During training, we utilized the Keras [9] *categorical_crossentropy* function to reduce our loss. We also utilized the A*dam* optimizer [11], and trained our model for 30 epochs with a batch size of 32.

The following charts were generated automatically during training using TensorBoard [12]. The first chart displays the loss after each epoch, and the second chart displays the accuracy after each epoch when evaluating the training dataset.

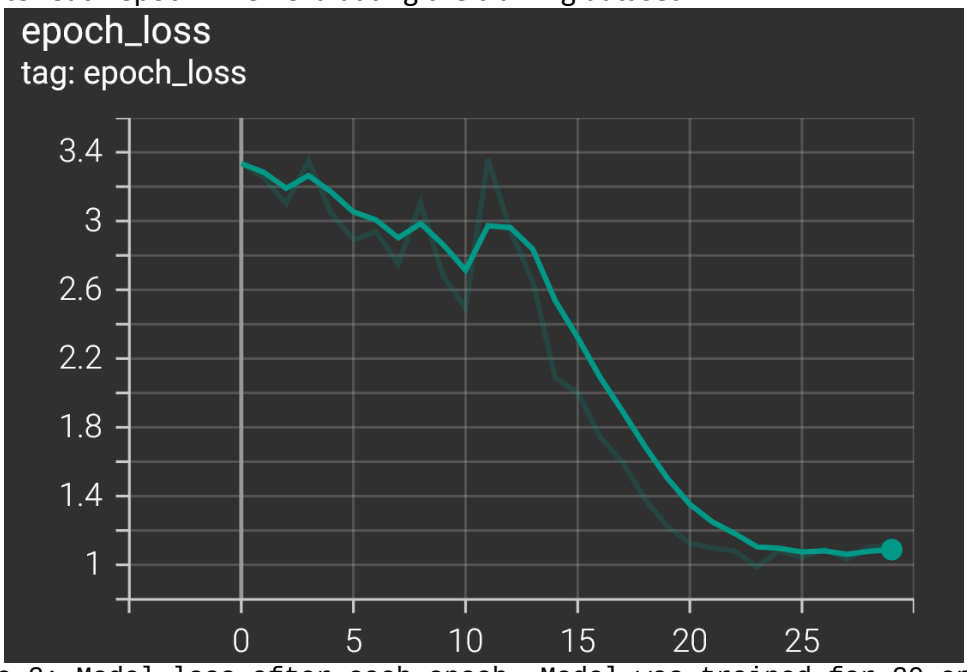

Figure 3: Model loss after each epoch. Model was trained for 30 epochs.

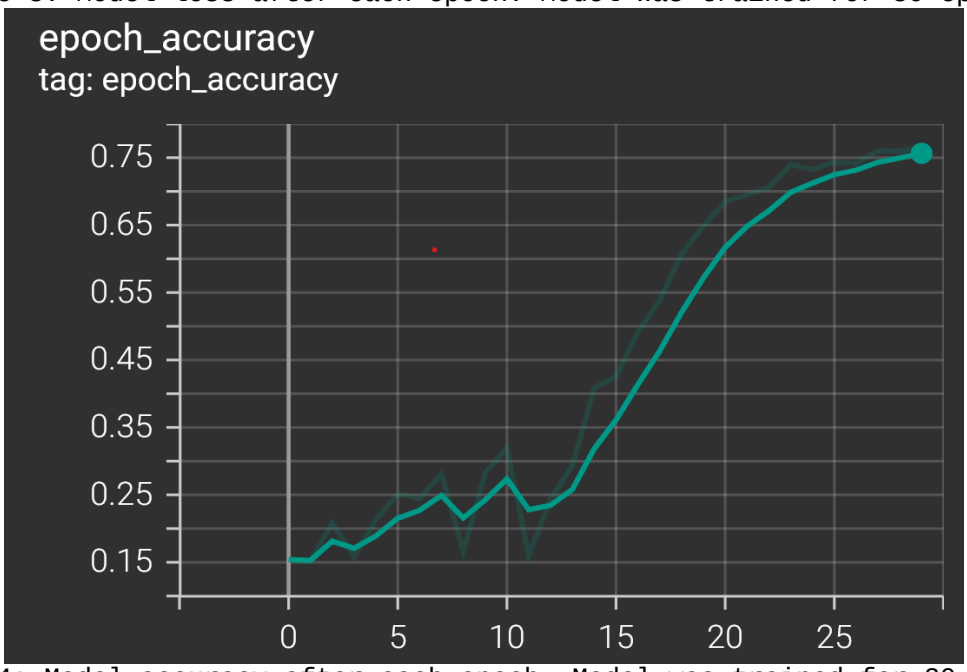

Figure 4: Model accuracy after each epoch. Model was trained for 30 epochs. Accuracy is of the training dataset

We discuss these figures more within our Evaluation section of this paper.

During training, the best accuracy that we were able to achieve on the validation data was 76.68%. When evaluating our model on the testing data, we achieved the following scores for these metrics:

- Accuracy: 75.90%
- Precision: 78.7046%
- Recall: 74.7019%
- F1-Score: 76.6511%

CNN

Our second model to perform Victorian Era Author Classification was a Convolutional Neural Network (CNN).

CNN's are more commonly used to handle multidimensional data, such as images. However, it is possible to use CNNs on one dimensional data, such as text. Convolutions are what make it possible for a CNN to learn feature representations of its input data. Convolutions work by sliding over the input data with different sized kernels (e.g. 3x3, 5x5). Once over the data, the dot product is computed at that location. After sliding through the data, the kernels then add together all of the dot products plus a bias term and store it in a feature map. These feature maps are then passed into an activation function to introduce non-linearity. Following the CNN block, max pooling is typically applied to find the most relevant features of the data.

Prior to processing the data, we split the data similar to what we did for the Multinomial Naive Bayes baseline and RNN model : 85% for training, 15% for validation. Following this, we performed the same data preparation as the RNN model.

Our CNN model took on the following architecture:

- An *Embedding* layer to take our tokenized text into dense vectors
- A *Conv1d* layer to learn the text representations of each class with *ReLU* [11: ReLU paper]
- A *GlobalMaxPooling* layer to identify the most relevant data
- A *Dense* layer to up sample the pooled feature maps to 512 dimensions
- A *Dropout* layer to remove any connections that were not activated
- A *Dense* layer to return the classification probabilities given the text

Figure 5 visualizes our architecture as well as providing additional insights into our input and output dimensions:

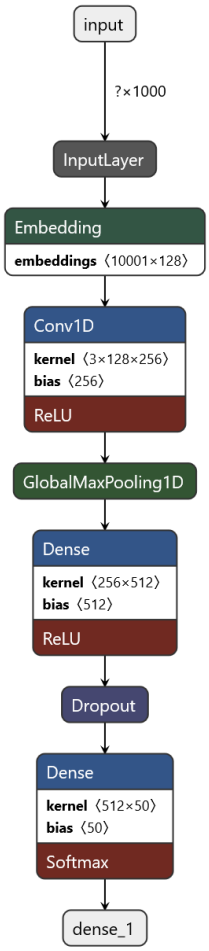

Figure 5: CNN architecture made with Netron [10]

During training, we utilized the Keras [9] *categorical_crossentropy* function to reduce our loss. We also utilized the *RMSProp* optimizer [13] and trained our model for 4 epochs with a batch size of 32.

The following charts were generated automatically during training using TensorBoard [12]. The first chart displays the loss after each epoch, and the second chart displays the accuracy after each epoch when evaluating the training dataset.

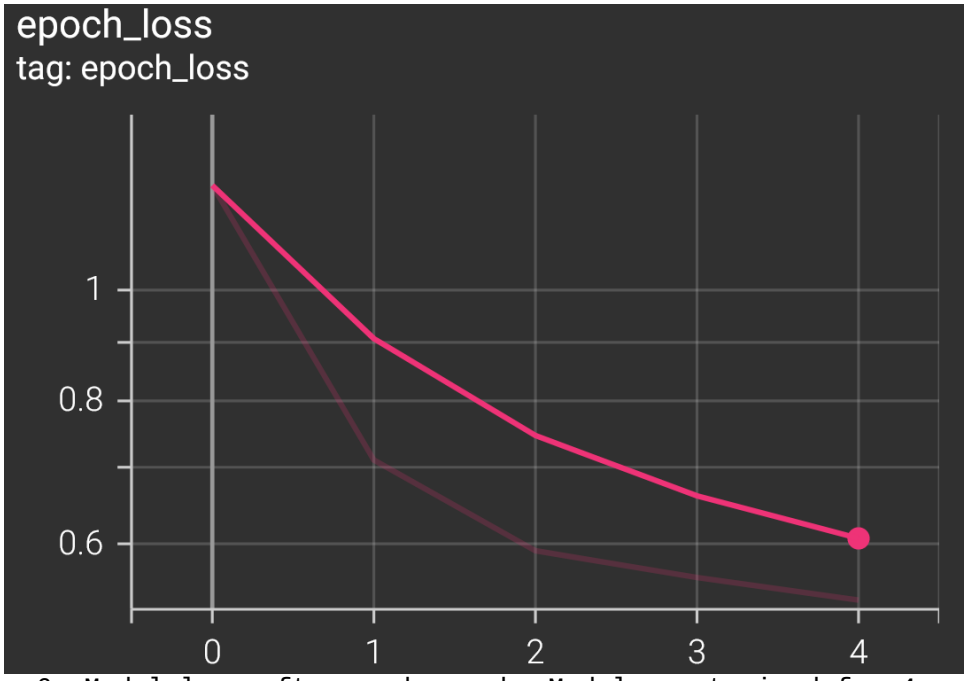

Figure 6: Model loss after each epoch. Model was trained for 4 epochs.

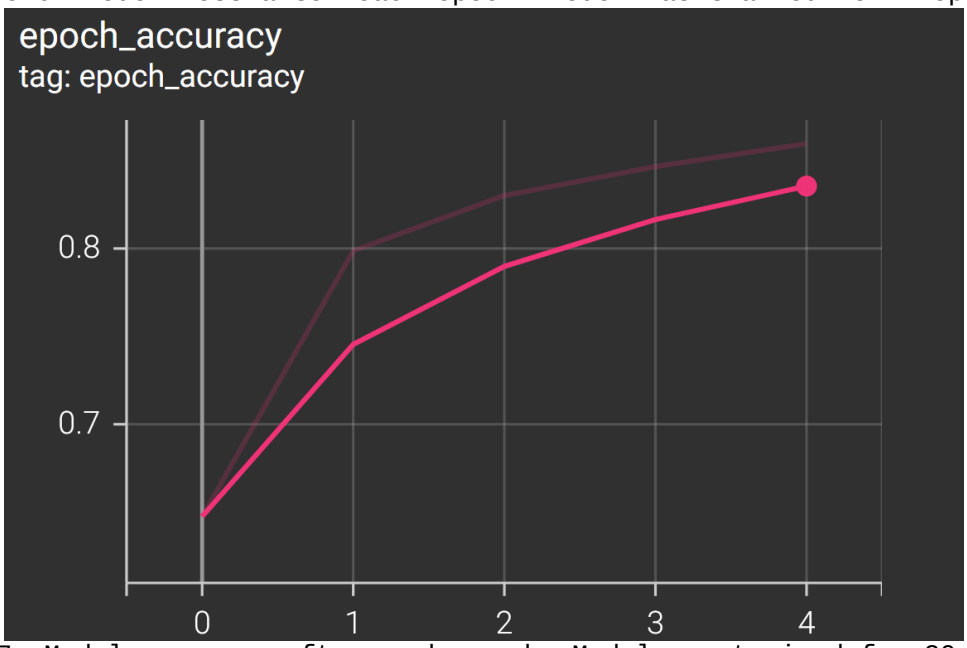

Figure 7: Model accuracy after each epoch. Model was trained for 30 epochs. Accuracy is of the training dataset

We discuss these figures more within our Evaluation section of this paper.

During training, the best accuracy that we were able to achieve on the validation data was 86.6346%. When evaluating our model on the testing data, we achieved the following scores for these metrics:

- Accuracy: 86.75%
- Precision: 89.4798%
- Recall: 85.5626%
- F1-Score: 87.4773%

Evaluation

Comparison of Models

We present two models for the purposes of Victorian Era Author Classification: a RNN and a CNN. However, we found that the CNN model was more accurate and more efficient than its RNN counterpart. Regardless, both the CNN and RNN outperformed the baseline Multinomial Naive Bayes model with respect to accuracy, precision, recall, and f1-score on the testing dataset.

Comparing Figure 4 (RNN accuracy on training dataset) against Figure 7 (CNN accuracy on training dataset) it becomes obvious that, while training, the CNN was able to achieve a higher accuracy. Furthermore, the validation accuracy of the RNN was 76.68%, whereas the validation accuracy of the CNN was 86.6346%; a clear ten percent increase over the RNN. Finally, when evaluating the accuracy of the models on the testing data, the accuracy for the RNN was 75.90%, whereas the CNN's accuracy was 86.75%. With respect to accuracy, the CNN model is more accurate than the RNN.

And when comparing precision, recall, and the f1-score of each model on the testing dataset, the CNN model has larger values across the board. Figure 8 visualizes the differences between these models.

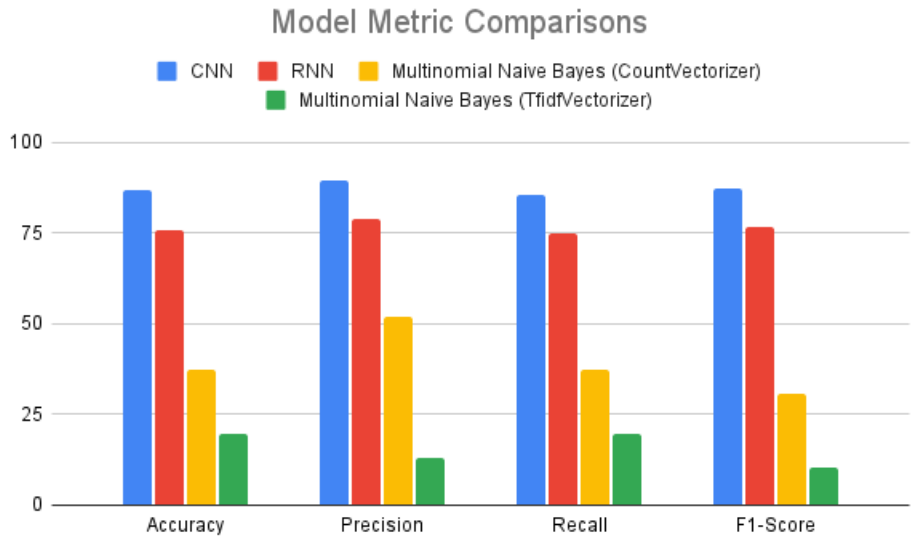

Figure 8: Comparison of CNN, RNN, and baseline Multinomial Naïve Bayes model metrics

With respect to efficiency, the CNN is also more efficient. This was measured by the minimum number of epochs it took for the model to achieve the maximum accuracy on the validation dataset. For the CNN, it took only 4 epochs (see Figure 6), whereas the RNN took 30 epochs (see Figure 3). This is important because the RNN is significantly slower to train per epoch. On one author's laptop, it took 4.5 minutes for the RNN to complete 1 epoch. Whereas the CNN only took 1 minute. While the time it takes for the model to complete a single epoch is less important, the fact that the RNN required 30 epochs to reach a global minimum whereas the CNN only required 4 epochs shows that the CNN was a much more efficient architecture for this task and dataset.

Top 5 Most and Least Accurate Classes per Model

Tables 1 and 2 show the top 5 most and least accurate classes on the testing dataset for the CNN and RNN model respectfully.

Most Accurate Classes		**Least Accurate Classes**	
Anne Manning	0.989865	Charles Dickens	0.627907
James Payn	0.980392	Washington Irving	0.615385
Thomas Hardy	0.960954	William Carleton	0.565789
Horace Greeley	0.944324	John Pendleton Kennedy	0.500000
Isabella Lucy Bird	0.942605	Ralph Emerson	0.459459

Table 1: CNN model's Top 5 Most and Least Accurate Classes by Percentage

Most Accurate Classes		**Least Accurate Classes**	
Anne Manning	97.5225%	George Eliot	39.5062%
James Payn	93.6275%	Robert Louis Stevenson	39.3939%
Mark Twain	91.2801%	Charles Darwin	31.5789%
Horace Greeley	89.6602%	Charles Dickens	27.9070%
Thomas Hardy	865510%	Ralph Emerson	27.0270%

Table 2: RNN model's Top 5 Most and Least Accurate Classes by Percentage

These values can be influenced by the number of training examples that each model was exposed to. For example, Mark Twain has 988,713 words between the testing and training dataset, whereas Charles Dickens only has 368,314 words. Because of discrepancies like these between the classes, it is possible for the models to learn the style or prose of one author better than another. However, if two or more authors share similar writing styles or patterns, then the accuracy can be affected as well as the model is unable to differentiate between the classes.

None the less, the CNN model has better per class accuracy than the RNN model when evaluating the Top 5 Least Accurate Classes. The lowest accuracy of the CNN model is 45.9459%, whereas the RNN model is 27.0270%. While the CNN model performs worse than random chance (50%) on its least accurate class it still is more accurate than the RNN and the baseline Multinomial Naive Bayes model which had an aggregate accuracy of 37.1647%.

Discussion

The study yielded several interesting findings regarding the dataset and the potential for the model to establish connections between authors and their texts. One area that could warrant further exploration is the parameters of the word vectors and embeddings. Potentially utilizing *word2vec* [14] or *fastText* [15] to generate the word vectors prior to analysis might provide better efficiency and accuracy for the RNN and CNN. Given that the task involves linking sections of text to authors' writing styles, the texts must reflect their unique style and all the nuances that come with it. While our model restricted the vocabulary through embedding parameters, it may be worthwhile to investigate whether incorporating all words could enhance the quality of results. Writing style is a deeply personal characteristic that is only discernible through extended passages of text in which authors can demonstrate their prose style. Preserving the texts without significant pre-processing might be a way to capture this aspect accurately.

The task of classifying a work based on a section of text encompasses various dimensions. One perspective that can be adopted is that since all authors operate within a linguistic sphere, their works might be similar and thus suitable for recommendations. To some extent, authors adhere to the language's limits, including grammar and relevant vocabulary, when producing their works. Therefore, similar spheres may generate comparable works, making them viable candidates for suggestions. Moreover, works that belong to similar spheres may form connections more readily when passed through a model. However, in the specific dataset we used, where all the sequences are the same, we could not explore this possibility. Our dataset focuses on authors from the Victorian era, a period rich in famous works. However, for a hypothetical future use case of an author suggestions platform, unless it catered solely to Victorian author enthusiasts, the authors and their works would need to span the entire history of English literature. In such cases, the model may exhibit signs of categorizing texts and authors that exist in the same sphere together, making them suitable candidates for recommendations.

On the other hand, the genre, topic, and plot of a book may vary significantly despite being written during the same period and cultural context. For instance, an author of fantasy fiction may have text sequences that differ significantly from those of someone writing about life in real-world London. It is challenging to predict how such differences may affect the embeddings and the model. In a work of fiction, essential keywords may appear relatively low on the word frequency scale, resulting in their exclusion through parameter regulations. However, if all words are taken into account, as previously suggested, this may shed light on how such factors impact connections between authors with similar writing styles or topics.

Conclusion

Natural Language Processing (NLP) techniques have shown promise in facilitating author recommendations to readers by utilizing a range of factors including writing style, genre, and topic. Through the application of NLP algorithms, the language used in texts can be analyzed and patterns can be identified in the data, leading to accurate and relevant book suggestions that align with readers' preferences. However, there are numerous challenges associated with this task, such as the requirement to incorporate domain-specific knowledge and the difficulty in accurately capturing the subtleties of writing styles. Despite these obstacles, the potential of NLP to assist with the author recommendation process is considerable, especially with advancements in NLP technology and the abundance of digital libraries continually expanding. Consequently, NLP offers an exciting prospect to assist readers in discovering new authors that enrich their reading experiences.

Our models demonstrate strong starting points for systems capable of classifying authors based off sections of text. With an accuracy score of around 75%, our RNN model proved to be a good start in regards to classifying texts according to author. However, with its accuracy rating at over 85%, we found that our CNN architecture is the most efficient and accurate architecture to go with. According to our results, the CNN and RNN achieved similar accuracy with their most accurate author across the dataset. This last only through the first author, after which the RNN begins dropping points each author. The CNN on the other hand, appears to be able to maintain its accuracy above 94% across the top five authors. Showing that on our dataset, a neural network with a convolution setup is better suited than one with recurrent architecture. However, we do suspect that with better word embeddings the RNN model could improve in efficiency and accuracy as well. We also included a Multinomial Naive Bayes for comparison purposes. It performed significantly worse with an accuracy of only around 37%. Next steps forward include generalizing the classifications of the model by incorporating authors outside of

the Victorian Era. Exploration of other neural net architectures could also be warranted. Such as evaluating the performance of a transformer model such as BERT to see how it might compare to the CNN and RNN on this dataset.

Appendix

- Giorgio Montenegro
  - Initial code and project designer
  - Paper writer and reviewer
  - PowerPoint review and creator
- Nicholas Synovic
  - Code implementation
  - Paper writer, reviewer, fact checking, and graphic design
  - PowerPoint reviewer and contributor
- Stefan Nelson
  - Paper reviewer
  - PowerPoint contributor

References

1. "3.5 Million Books 1800-2015: GDELT Processes Internet Archive and HathiTrust Book Archives and Available In Google BigQuery – The GDELT Project." https://blog.gdeltproject.org/3-5-million-books-1800-2015-gdelt-processes-internet-archive-and-hathitrust-book-archives-and-available-in-google-bigquery/ (accessed May 03, 2023).
2. "Books of the world, stand up and be counted! All 129,864,880 of you." http://book-search.blogspot.com/2010/08/books-of-world-stand-up-and-be-counted.html (accessed May 03, 2023).
3. A. Gungor, "Benchmarking Authorship Attribution Techniques Using over a Thousand Books by Fifty Victorian Era Novelists," M.S., Purdue University, United States -- Indiana. Accessed: May 03, 2023. [Online]. Available: https://www.proquest.com/docview/2101510652/abstract/C2F48BEEC24C4B78PQ/1
4. N. Synovic, "Victorian Era Classification." Mar. 29, 2023. Accessed: May 03, 2023. [Online]. Available: https://github.com/NicholasSynovic/nlp-victorianAuthor
5. P. Juola, "Authorship attribution," *Found. Trends Inf. Retr.*, vol. 1, no. 3, pp. 233–334, Dec. 2006, doi: 10.1561/1500000005.
6. "The GDELT Project." https://www.gdeltproject.org/ (accessed May 03, 2023).
7. "UC Irvine Machine Learning Repository." https://archive-beta.ics.uci.edu/ (accessed May 03, 2023).
8. "scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation." https://scikit-learn.org/stable/index.html (accessed May 03, 2023).
9. "Keras: Deep Learning for humans." https://keras.io/ (accessed May 03, 2023).
10. L. Roeder, "Netron, Visualizer for neural network, deep learning, and machine learning models." Dec. 2017. doi: 10.5281/zenodo.5854962.
11. D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." arXiv, Jan. 29, 2017. doi: 10.48550/arXiv.1412.6980.
12. "TensorBoard | TensorFlow." https://www.tensorflow.org/tensorboard (accessed May 03, 2023).

13. K. Team, “Keras documentation: RMSprop.” https://keras.io/api/optimizers/rmsprop/ (accessed May 03, 2023).
14. T. Mikolov, Q. V. Le, and I. Sutskever, “Exploiting Similarities among Languages for Machine Translation.” arXiv, Sep. 16, 2013. Accessed: May 03, 2023. [Online]. Available: http://arxiv.org/abs/1309.4168
15. P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching Word Vectors with Subword Information.” arXiv, Jun. 19, 2017. doi: 10.48550/arXiv.1607.04606.