Authorship Identification with Embeddings and Sentence Variations

Enok Choe
Department of Computer Science
Stanford University
`echoe720@stanford.edu`

Jasper McAvity
Department of Computer Science
Stanford University
`jmcavity@stanford.edu`

Abstract

In this paper, we aimed to solve the problem of predicting the author of a given piece of text. We created datasets mapping different-sized chunks of text to authors, and used them to train several models. These models include a basic ANN that takes in word count vectors, several RNN architectures that take in GloVe embeddings, and a sentence encoder that maps text into a latent space. All models produced impressive results on a variety of datasets, with the ANN performing the best. We conclude by analyzing potential reasons for this surprising result.

1 Introduction

Every author has a unique writing style; their word choice, the way they formulate their sentences, and even overall structure are some of the tangible factors that help distinguish one author's writing from another. With this in mind, we were curious if it would be possible to train a deep learning algorithm that could train on works by different authors and be used to identify the author of a new piece of text. This task is also known as authorship identification.

Authorship identification can be used to solve many interesting problems. For instance, one application of authorship identification could be to confirm that content found online was actually written by the listed author. It could also be helpful in contexts like detecting plagiarism or fake news. This project used several models, such as ANN and RNN architectures, as well as a sentence embedding model, to accomplish this task.

2 Related work

Most existing authorship identification efforts are primarily based on deep neural networks and applications of word embeddings. Most recently, latent Dirichlet allocation, decision trees, and varying neural networks were applied. CNN models applied to this task had about 0.52 average accuracy [1]. In an another work, support vector machine (SVM) classifiers were used, but were only effective for longer documents [2]. Since we wanted to parse datasets into smaller sentence mappings, we decided to avoid SVMs. Another method is unmasking, which repeats the process of determining ten-fold cross-validation experiment results, then eliminating the k most strongly weighted positive and negative features, with SVM with linear kernel [3]. However, for the same reasons as [2], we decided against building upon this model. Other algorithms include Random Forest Classifier, Extra Tree Classifier, Decision Tree Classifier, AdaBoost Classifier, and Gradient Boosting Classifier, all with GloVe embeddings. However, most performed far worse than RNNs in f1 score, and RNNs have more scholarly references [4]. Given this data, we decided to focus on RNNs. A prior use case of C50 training uses RNN classifiers with GRU, LSTM, and BiLSTM models for single sentence and article levels with much higher accuracy for article level [5]. However, we found that there are significant holes in this case such as overfitting, inconsistent distribution of training data to test data and sentence parsing. Our BiLSTM model addresses these issues by building upon this model.

3 Dataset and Features

We used two datasets. The first was taken from a corpus of books from the website Project Gutenberg [6]. The second was taken from a corpus of news articles (Reuters_50_50) from UCI Machine Learning Repository, and called C50 [7]. C50 contains texts written by 50 different authors, which, when parsed into individual sentences, contained 43382 training examples in our training set and a similar number in our test set. Because this number was so large that it required too much computational power for our setup, we decided on a 50/50 split between training and test sets. For Gutenberg, we created a dataset using texts from the 50 most common authors in the dataset. This meant that both datasets had the same number of authors. One thing that was unclear was what size our training examples should be. Therefore, instead of picking one size, we created multiple datasets with different sized training examples. Each new dataset had training examples with a constant number of sentences, where we mapped authors to n-sentence sized chunks of text, where n could be 1, 5, 7, 10, or 100. We also created a dataset that mapped full C50 articles to author. For the rest of this paper, we will refer to individual datasets by their name, followed by their example size. For example, G-5 would refer to the Gutenberg dataset with 5-sentence training examples, while C50-5 would be the same but with the C50 dataset. Because C50 articles were only about 20-30 sentences on average, we did not have a C50-100 dataset. We also did not have a G-1 or G-7 dataset because the Gutenberg corpus was very large, and the amount of time needed to parse these datasets was deemed too high.

4 Methods

The first two methods we used were classifiers that, given a word vector for a piece of text, classified it as one of 50 authors. We used both the C50 and Gutenberg datasets for these methods. Our last method fine-tuned a pre-trained sentence transformer to create embeddings for each training example and used these embeddings to predict authorship. For this method, we only used C50-5. As a baseline, we used a standard artificial neural network with one 100-unit hidden layer that took in average sentence length and average word length. The baseline achieved 14% accuracy on the G-100 training examples, which was significantly better than random chance (2%), but was clearly not practically useful. We will now go into further detail about our other methods.

4.1 Artificial Neural Network with Word Counts

Our first method used the same architecture as our baseline. Our model had one 100-unit hidden layer, using a ReLU activation function. Experiments with more layers gave the model a tendency to over-fit, and best results were found with this architecture. The output layer was a 50-unit softmax, which allowed the model to classify each training example to a particular author. We used the Adam optimizer (with 0.001 learning rate) and binary cross-entropy loss for training. We found most success training for 5 epochs with a batch size of 32.

The inputs to the network were word count vectors created by the scikit-learn CountVectorizer function, which encodes the number of times each word appears in a particular training example. This meant that the word vectors captured surface-level information, and didn't look at co-occurrence between words or other features.

4.2 RNN Architectures with GloVe

Because our ANN model was quite basic, we wanted to try solving the problem using more complicated RNN architectures that could better capture the meaning of text. We tried LSTM, GRU and BiLSTM models with GloVe embeddings as inputs, with more attention on the BiLSTM model after seeing consistent patterns of BiLSTM outperforming the other two. A pre-existing model [11] was fine-tuned to meet our dataset formats.

Each of the input texts were converted into GloVe-vectorized matrices. The GloVe model was trained on a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in the provided dataset. The training data was then trained through a classifier with varying number of LSTM, GRU, BiLSTM layers (e.g. 1, 2, 3) with varying batch sizes (e.g. 64, 128, 256), with hidden layer sizes (e.g. 200, 300, 400), embedding length of 100, and with Adam optimizer. BiLSTM is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. Relatively, BiLSTM has more context available to the algorithm, which in this case would be knowing what precedes and follows a given word. We used Cross-entropy Loss function, and the algorithm was run for 20 epochs.

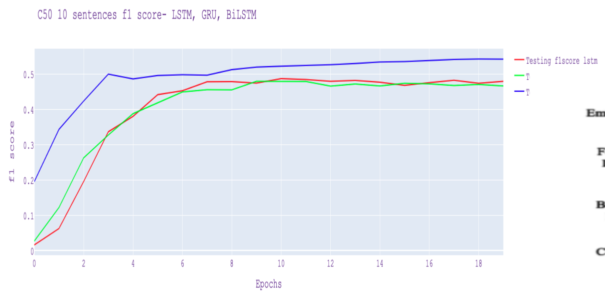

Figure 1: C50-10 f1 score- BiLSTM (Blue) outperforms the others throughout all of the epochs, which was consistent with our results in most trials.

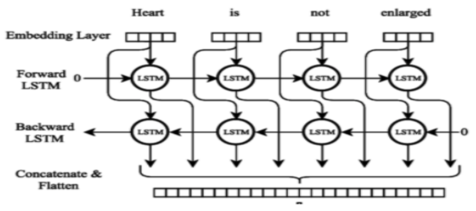

Figure 2: BiLSTM architecture

4.3 Latent Space Encoder with Triplet Loss

One problem with the previous methods was that, as classifiers, they relied on having a fixed number of authors. If a piece of text was not written by an author in the dataset, they would be unable to predict this. Therefore, we trained another algorithm that could handle outside authors. Our model learned a low-dimensional embedding space that could encode text and then find the nearest author in that space (using cosine similarity). This method allowed us to pick a similarity threshold, where if the nearest author was below that threshold, we would classify the example is written by an outside author. We only used the C50 dataset for this task, because it was more well-structured and easier to work with. This method is very similar to one used by many facial recognition algorithms.

We started off with a pre-trained encoding model and then fine-tuned it. The architecture of the model has two parts: word embedding and pooling. The pre-trained model we started with used a BERT model to generate a word embedding for each word in the example, and then used mean-pooling to combine the embeddings into a 768-dimensional output vector. BERT is a bidirectional language model trained on a large corpus of unlabeled text, and is designed to be fine-tuned.

To fine-tune this pre-tained model to fit our task, we used triplet loss to help update the parameters. Calculating triplet loss uses the following formula:

$$Loss = \sum_{i=1}^{N} \left[\|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha \right]_+$$

For a given training example (called the anchor), calculating the loss requires two other pieces of training data, one with the same label as the anchor (positive) and one with a different label (negative). Triplet loss minimizes the distance between the anchor and positive examples and maximize the distance between the anchor and negative examples.

After fine tuning the pre-trained model by fitting it using triplet loss, we were able to find an embedding for each author by averaging the embeddings of each of their corresponding training examples. We used this set of author embeddings to solve the classification task from the earlier methods. Additionally, we were able to simulate training examples not written by authors in the dataset by treating 40% of the test set as if it were written by outside authors. We were then able to determine a similarity threshold at which the model was best able to predict authorship.

5 Results

5.1 ANN and BiLSTM Results

Performance for the classifier models can be seen in the following table, as well as in figure 4.

Model	Dataset	1 Sentence	5 Sentence	7 Sentence	10 Sentence	Whole text	100 Sentence
ANN	C50	0.425	0.597	0.621	0.644	0.695	N/A
ANN	Gutenberg	N/A	0.734	N/A	0.962	N/A	0.982
BiLSTM	C50	0.433	0.529	0.534	0.543	0.550	N/A
BiLSTM	Gutenberg	N/A	0.600	N/A	0.823	N/A	0.686

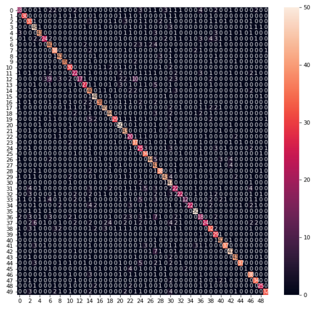

Figure 3: Confusion matrix from BiLSTM (C50-10).

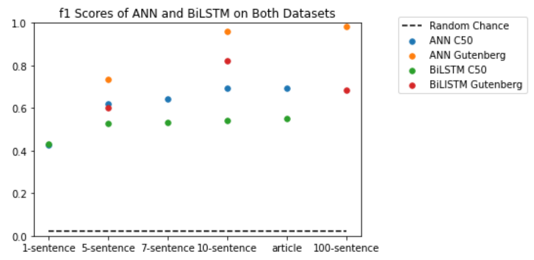

Figure 4: Visual representation of results for ANN and BiLSTM on different-sized training examples.

The two immediate takeaways from Figure 4 are that the ANN with CountVectorizer had higher accuracy than the BiLSTM, and that Gutenberg datasets had higher accuracy than C50 datasets. We will discuss reasons for this in the conclusion. As far as difference between datasets, C50 is composed of news articles, full of quotations that give little information about the author. Gutenberg, in contrast, is made up of stories, and contains many characters, place names, and dialects that may carry a lot of weight in training and make decoding easier. This provides an explanation for the ANN's success for the G-10 and G-100 datasets. In this case, the model has likely overfitted to the dataset. Because of the way the dataset was split into training and test sets, it is almost guaranteed that parts of a given book will be in both sets. Therefore, in a large training example, it is very likely that rare keywords like character names appear, allowing the model to predict authorship and bypass any learning of style all together. For example, if a thousand word piece of text has the words "Oliver Twist" in it, you can be quite certain that the author is Charles Dickens, independent of the other 998 words.

Our primary accuracy metric was f1 score. As our table indicates, the accuracy of our models tended to increase with training example size, likely because of the increased amount of information in larger training examples. The outlier is G-100 on the BiLSTM. We do not have a great explanation for this, but one possiblity is that with such large pieces of text, it was difficult for the model to extract coherent semantic information. For the BiLSTM model, we tuned several hyperparameters, such as epochs, learning rate, hidden layers, and number of BiLSTM cells. Because we noticed training accuracy increasing until the end of the training process, we tried adding more epochs. However, we observed that this only led to overfitting, and testing accuracy was unaffected. Changing the number of hidden layers similarly had little effect. One explanation for this is that the problem was not complicated enough to be improved by a deeper network. Increasing the batch size, however, led to better results. For C50-7, for instance, changing batch size from 64 to 128 increased the f1 score from 0.515 to 0.535, a significant increase. On the other hand, decreasing batch size decreased the f1 score to 0.50. This is likely because BiLSTMs' learning is dependent on contexts, and more batch sizes means potentially having more contextual clues pertaining to the author.

5.2 Sentence Encoder Results

In solving the classification problem on the C50-5 dataset, the embedding model achieved an accuracy of 0.606, slightly surpassing the performance of both previous models. When we added text not written by authors in the embedding space, we tested the accuracy of the models using thresholds ranging from 0.5 to 1.0, where a cosine similarity between the nearest embedding and a training example of less than the threshold would be classified as written by an outside author. We found that the best threshold was 0.86, which gave an accuracy of 0.548 on this more difficult task. We trained the model using BatchAllTripletLoss for two epochs, with a batch size of 16.

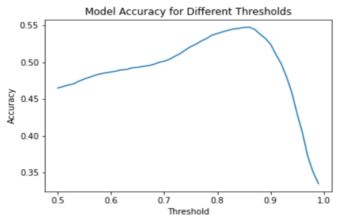

Figure 5: Graph of accuracy from sentence embedding model for different similarity thresholds.

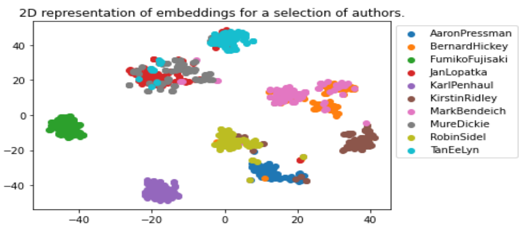

Figure 6: Representations of embeddings for a selection of 9 authors projected onto a 2D space.

Figure 6 plots 2D representations of the embeddings of a selection of 9 authors from C50. It is clear that some authors, like Fujisaki and Penhaul, have embeddings that are tightly clustered and separated from the others. Other authors, like Lopatka, Dickie, and Eelyn, in contrast, have embeddings that are widespread and overlap in this 2D space (though overlap in 2 dimensions by no means implies overlap in 768). When looking at accuracies for individual authors, these factors clearly make a difference. Fujisaki and Penhaul had accuracies of 0.97 and 0.95 respectively, compared to Lopatka, Dickie, and Eelyn with 0.27, 0.35, and 0.19. The extent to which embeddings for an author are clustered is an indication of how easy it was for the model to learn their style of writing, or, in other words, how distinctive their writing is. Another possibility is that these authors wrote about unusual subjects, and the model was able to pick this up. This visualization makes it clear why there is such variation in the accuracies of different authors. Tightly clustered authors are more likely to have a new embedding be close to the average, while spread out authors, especially ones like Eelyn, with some embeddings quite far away from the main cluster, are less likely to have a new embedding be close to the average and be predicted correctly (as well as the possibility of overlap). This reveals a potential problem with our method of fusing embeddings for a given author. One could imagine an extreme case in which an author had two distinct clusters. The averaging approach would yield a value in the middle, leading to low accuracy. An alternate method that kept track of more of the training embeddings could use an algorithm like k-nearest neighbors to potentially do even better.

6 Conclusion/Future Work

The ANN model with CountVectorizer tended to outperform the other models across the datsets. Its success compared to more complicated models indicates something about the task of authorship identification. In starting out this project, we hoped that we would be able to train a model that could pick up on the intrinsic style in an author's writing, a task moree suited to models using GloVe or BERT or RNNs. However, our results show that authorship identification in the context of our project can be more easily solved by brute force, by counting as many words as possible until you find keywords that are somewhat unique to a particular author. The ANN therefore performed well on this easier task, while the BiLSTM and sentence embedding models were less successful in a harder task.

As we mentioned, this is a case of overfitting to the dataset, and it is unlikely that our model would perform as well as it did in a practical setting (if one of the C50 authors were to write a new article on a new topic). Therefore, despite the impressive accuracy of the ANN, it would not be useful for some of the topics we discussed, such as fake news detection. We believe that taking more care in training a model that specifically learns style would be more useful, and is something we would be interested in trying out in the future. One idea would be using ideas from CNN architectures such as neural style transfer or facial recognition algorithms that are more traditionally used on images. These could prove more successful in extracting the style of an author's writing instead of the content.

Overall, the more sentences we had mapped to each author, the better the accuracy was. If we had more computational resources and time, we would explore training for more sentence sizes, as the result for the BiLSTM with G-50 decreased in performance compared to G-10. This indicates that there could be an optimal number of sentences, above which performance begins to decrease. Further investigation of different-sized mappings would allow us to answer this question.

7 Contributions

Jasper worked on parsing the Gutenberg dataset and built the ANN architecture. Enok worked on parsing the C50 dataset and developed the RNN architectures. We collaboratively made the sentence encoder and wrote the final paper.

References

[1] Rhodes D. Author Attribution with CNN's. https://cs224d.stanford.edu/reports/RhodesDylan.pdf, 2015.

[2] C. Sanderson and S.Guenter. Short Text Authorship Attribution via Sequence Kernels, Markov Chains and Author Unmasking: An Investigation. In Proceedings of the International Conference on Empirical Methods in Natural Language Processing, pages 482491, 2006.

[3] M. Koppel, J. Schler, and E. Bonchek-Dokow. Measuring Differentiability: Unmasking Pseudonymous Authors. Journal of Machine Learning Research, 8:12611276, 2007.

[4] Zhou, Liuyu Wang, Huafei. News Authorship Identification with Deep Learning. Stanford, CA.

[5] Talati, Arth. Deep-Learning-based-Authorship-Identification, https://github.com/arthtalati/Deep-Learning-based-Authorship-Identification, 2020.

[6] Shibamouli, Lahiri. "Complexity of Word Collocation Networks: A Preliminary Structural Analysis. Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, p.96-105, http://www.aclweb.org/anthology/E14-3011. 2014.

[7] Zhi, Liu. National Engineering Research Center for E-Learning, Hubei Wuhan, China. https://archive.ics.uci.edu/ml/datasets/Reuter_50_50

[8] Qian, C., He, T., Zhang R. "Deep Learning based Authorship Identification." Stanford Department of Electrical Engineering, Stanford, CA, 2017.

[9] Krasser, Martin. face-recognition, https://github.com/krasserm/face-recognition, 2019.

[10] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[11] Bird, Steven, Edward Loper and Ewan Klein, Natural Language Processing with Python. O'Reilly Media Inc., 2009.

[12] Chollet, F., others. Keras. GitHub. Retrieved from https://github.com/fchollet/keras, 2015.

[13] Paszke, Adam and Gross, Sam and Massa, Francisco and Lerer, Adam and Bradbury, James and Chanan, Gregory and Killeen, Trevor and Lin, Zeming and Gimelshein, Natalia and Antiga, Luca and Desmaison, Alban and Kopf, Andreas and Yang, Edward and DeVito, Zachary and Raison, Martin and Tejani, Alykhan and Chilamkurthy, Sasank and Steiner, Benoit and Fang, Lu and Bai, Junjie and Chintala, Soumith, PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32. Curran Associates, Inc. p.8024-8035, 2019.

[14] Reimers, Nils and Gurevych, Iryna. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". Association for Computational Linguistics, 2019.