Author Identification

Eric Tay and Diane Lin

May 18, 2021

1 Introduction

For our final project, we chose the NLP problem of author identification. For this problem, we chose the Reuters_50_50 dataset [1], a subset of Reuters Corpus Volume I (RCV1) [2], an archive of over 800,000 manually categorized newswire stories made available by Reuters, Ltd. for research purposes. This particular subset selects the top 50 authors (with respect to total size of articles) in RCV1, whereby each author has at least one text labeled with the subtopic of class CCAT(corporate/industrial). This attempts to minimize the topic factor in distinguishing amongst texts. The training set consists of 2,500 texts (50 per author) and the test set includes another 2,500 texts (50 per author). This dataset has already been used in other works studying author identification [3] [4] (although the methods used in this project would be distinct from those papers). Due to computational reasons, we further subset the data, sorting the authors alphabetically and choosing to consider the first 10 authors.

To address the problem of author identification, we first introduce a generative probabilistic model in Section 2.2, which is trained on our training data, and generates synthetic train and test data. We have chosen to use the n-gram model (with n = 6) for this model. Based on this generative probabilistic model, we then develop a probabilistic solution for the problem of author identification (Section 2.3). We also develop a neural network solution for the problem in general (Section 3). We then evaluate both solutions in Section 4, by training and tuning the models with the synthetic train data, and reporting train and test accuracy (with misclassification error being our error metric, and testing on the synthetic test data). For the synthetic data, our probabilistic solution achieves a training accuracy of 100% and a testing accuracy of 95.8%, while our neural network solution achieves a training accuracy of 100% and a testing accuracy of 99.0%.

In Section 5, we then train our solutions with the real training data, and report train and test accuracy, this time testing with the real test data. Our probabilistic solution achieves a training accuracy of 100% and a testing accuracy of 79.0%, while our neural network solution achieves a training accuracy of 100% and a testing accuracy of 87.0%.

Finally, we discuss the merits and drawbacks of each approach in Section 6, as well as possible further extensions for each model.

2 Generative Probabilistic Model and Probabilistic Solution

2.1 Data Processing

For each text in the train and test set, we first split them into sentences using `nltk`'s Punkt sentence tokenizer. Then, we split each sentence into tokens using `nltk`'s Treebank word tokenizer and converted each of these tokens into lowercase. Next, we carry out pos-tagging on each of these words using `nltk`'s averaged perceptron tagger. Finally, the tokens and their tags are used in `nltk`'s Wordnet lemmatizer, to lemmatize the tokens. The result is that each text is deconstructed into a list of lemmatized tokens, or a list of sentences, whereby each sentence is a list of lemmatized tokens (We process the text in two ways, as sentence structure is used in the Neural Network solution). The process of converting the tokens into lowercase and subsequent lemmatization are to group similar tokens together, for our n-gram models (both the generative probabilistic model and the probabilistic solution) to have higher effectiveness.

2.2 Generative Probabilistic Model

For our generative probabilistic model, we use the n-gram model (with backoff). We choose n = 6 as this gives the best validation accuracy when nested cross-validation was carried out with the Probabilistic solution on real data in Section 5.1. We also note that this selection of n generates text that is reasonably realistic.

Let there be N unique tokens in the train data, $w_1, \ldots, w_N$. Also let the 50 authors be denoted as $a_1, \ldots, a_{50}$. We then begin by constructing $b_{ij}, c_{ijk}, d_{ijkl}, e_{ijklm}, f_{ijklmn}, g_{ijklmno}$, whereby $g_{ijklmno}$ denotes the number of occurrences of sequence $\{w_j, w_k, w_l, w_m, w_n, w_o\}$ in the 50 training texts by author a_i. b, c, d, e and f are similarly defined for sequences of length 1, 2, 3, 4 and 5 respectively.

We then generate texts for a_i as follows.

1. Take the last 5 tokens in the existing sentence. If the existing sentence has less than 5 tokens, take all the tokens (this handles the generation of the first few tokens).

2. If the sequence of tokens does not exist in the 50 training texts by author a_i, remove the first token in the sequence until it does, or until the sequence of tokens has length 0, whichever comes first.

3. Generate a new token from the appropriate vector (b if the sequence of tokens is length 0, c if length 1, ..., g if length 5). Suppose g is the chosen vector, with the sequence being $\{w_j, w_k, w_l, w_m, w_n\}$. Then we would generate token w_o with probability $\frac{g_{ijklmno}}{\sum_{p=1}^{N} g_{ijklmnp}}$. Define probabilities analogously for sequences of other lengths.

4. Repeat steps 1-3 until the desired length is reached.

Given that the average length of texts in both the train and test set were around 600 tokens long, we generated 50 synthetic train texts and 50 synthetic test texts each of length 600 tokens, for each of the 10 authors. An example of the text that was generated from this process is as follows, "leach introduce in the previous congress founder last june amid opposition from the insurance industry and house democrat. analyst have say this year's bill might have a good chance of passage, but that be before leach decide to oppose newt gringrich's reelection a speaker of the a move that could turn the leadership against his bill." We note that the sentence is reasonably realistic, (which supports both the model and choice of n), especially considering how all tokens were converted to lower-case, and attributing the majority of grammatical errors to the previously applied lemmatization.

2.3 Probabilistic Solution

Let our 1000 synthetic train and test texts be $t_1, \ldots, t_{1000}$. Then for any text t_j, we calculated the probability that text t_j was generated from the texts of author a_i, denoted by p_{ij} (Note that this probability was defined up to a proportionality constant common to all p_{ij}s with common j). We therefore classified t_j as being generated from texts of author a_i for $i = \arg\max_{i \in \mathbb{N}, 1 \leq i \leq 50} p_{ij}$.

Let the length of t_j be L_j, and the tokens in t_j be $t_{j1}, \ldots, t_{jL_j}$. We then define $p_{ij} = \prod_{k=1}^{L_j} q(t_{jk}|i, t_j)$, where $q(t_{jk}|i, t_j)$ can be seen as the conditional probability of generating token t_{jk}, and is similarly defined up to a proportionality constant common to all $q(t_{jk}|i, t_j)$s with common j, k.

Given that our generative probabilistic model was an n-gram model with n = 6, we wished to exploit this structure in our probabilistic solution. Therefore, with the 500 synthetic train texts, we similarly constructed $b'_{ij}, c'_{ijk}, d'_{ijkl}, e'_{ijklm}, f'_{ijklmn}$ and $g'_{ijklmno}$ as before. Then, we calculated $q(t_{jk}|i, t_j)$ as follows:

1. Choose a value P and UNK.

2. Take the 5 tokens preceding t_{jk}. If there are less than 5 tokens, take all the tokens. Let the sequence defined by these tokens, along with t_{jk}, be s_{jk}.

3. If s_{jk} does not exist in the 50 synthetic training texts by author a_i, remove the first token in the sequence until it does, or until the sequence of tokens has length 1, whichever comes first.

4. (a) If the sequence is length 1, and the token does not exist in the 50 synthetic training texts by author a_i, $q(t_{jk}|i, t_j) = UNK$.

   (b) If the sequence is length 1, and the token exists in the 50 synthetic training texts by author a_i, let the token be w_q. Then $q(t_{jk}|i, t_j) = \frac{b'_{iq}}{\sum_{h=1}^{N} b'_{ih}}$.

   (c) If the sequence has length greater than 1, first select the appropriate vector (c' if the sequence of tokens is length 2, d' if length 3, ..., g' if length 6). Suppose g' is the chosen vector, with the sequence being $\{w_r, w_u, w_v, w_x, w_y, w_z\}$. Let the length of the sequence at this point be L. Then $q(t_{jk}|i, t_j) = P(L-1)\frac{g'_{iruvxyz}}{\sum_{h=1}^{N} g'_{iruvxyh}}$. Define $q(t_{jk}|i, t_j)$ analogously for sequences of other lengths.

Step 2 is motivated from our generative model being a 6-gram model. Step 4(b) is also similarly modelled after probabilities that the generative probabilistic model uses to generate tokens. On top of this, we introduce two novel ideas in Step 4(a) and 4(c).

For Step 4(a), when the algorithm comes across a token t_{jk} that is unseen in the training data for a_i, we assign the probability of generation $q(t_{jk}|i, t_j) = UNK$, which would be set to a small constant. Therefore, if t_{jk} is unseen in the training data for both a_i and a'_i, $i \neq i'$, then both authors would generate t_{jk} with equal probability. Here we make the assumption that each author would generate an unseen word with equal probability, which is reasonable given that

each author would have a training set of 30,000 tokens. Ideally, we want UNK to be small to represent the belief that if a word has been seen commonly in a_i's training data, but not in a_i's training data, then $q(t_{jk}|i, t_j) > q(t_{jk}|i', t_j)$.

For Step 4(c), let s_{ijk} be the longest sequence of tokens in t_j ending in t_{jk} such that the sequence exists in the training data for a_i, truncated at the front (all sequences end in t_{jk}) such that the maximum length of s_{ijk} is 6 (this incorporates the structure of our generative probabilistic model). Now if the length of s_{ijk} is greater than the length of $s_{i'jk}$, we assume that $q(t_{jk}|i, t_j) > q(t_{jk}|i', t_j)$, since it is rarer to match a longer sequence of tokens. It is also important to note that Step 3 of the algorithm ensures that at every token t_{jk}, we consider the longest possible sequence s_{ijk}. This allows for the proper functioning of Step 4, and represents our belief that long sequences of tokens encode author-specific writing patterns. To weight these longer sequences more, $q(t_{jk}|i, t_j)$ therefore incorporates a weight of $P(L-1)$, where L is the length of s_{ijk}, and P is a constant that determines how heavily we wish to weight this effect of sequence length by.

3 Neural Network Solution

3.1 Article-level GRU

We base our neural network solution from the "article-level GRU" solution proposed in a paper by Stanford University [5], which similarly tackles the problem of author identification for the Reuters_50_50 dataset. Among the models tested in the paper, the proposed article-level GRU solution achieves the highest accuracy of 69.1% for the 50-class classification problem (as compared to the 10-class we have been considering). It should, however, be noted that the paper reorganizes the training and testing data provided into a 9-1 train-test split (as compared to the 1:1 ratio in the original data).

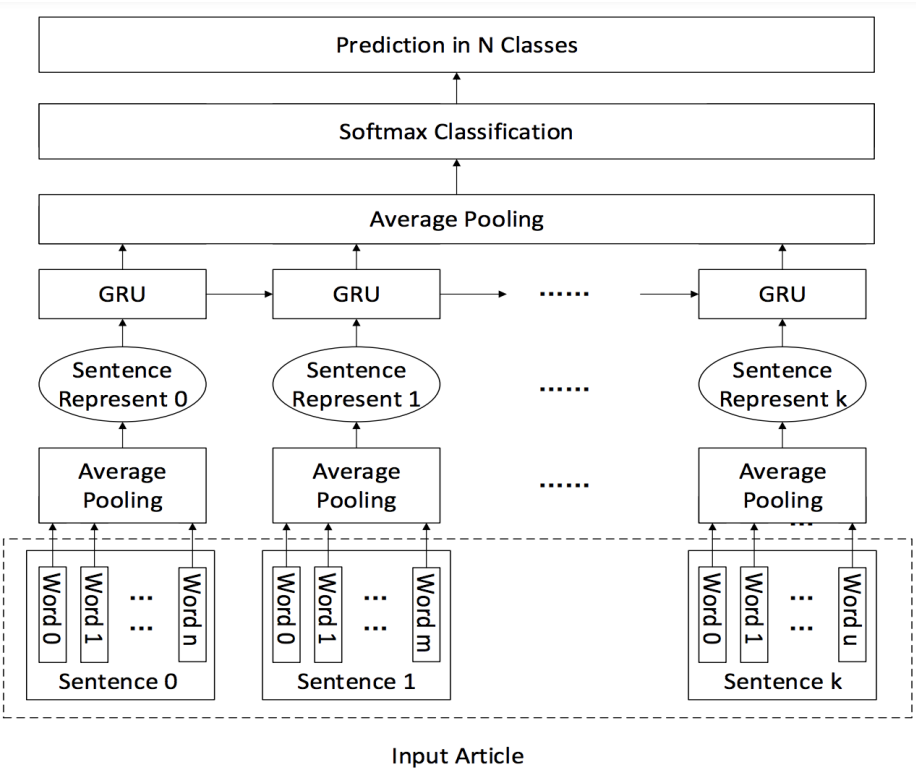


Figure 1: Architecture of article-level GRU, taken from [5].

The paper uses GloVe word vectors of size 50 trained on Wikipedia (2014) and Gigaword 5 as pretrained word embeddings, used to initialize the word embeddings in the neural network model. Rare words that did not have an embedding were eliminated. Next, each sentence was then represented by the average of all its words' vectors, which is described as "Average Pooling" in Figure 1.

The sentences are taken as input into a Gated Recurrent Unit (GRU), and all the outputs from the GRU would be averaged again ("Average Pooling" in Figure 1). Given that the authors vary the output dimension of the GRU, we assume that the averaged output goes through a linear layer (not pictured), which outputs a 50-dimensional vector, which then goes through softmax classification for predictions.

3.2 Innovations

The first change we made was to use GloVe word vectors of size 300 trained on Wikipedia (2014) and Gigaword 5 as pretrained word embeddings, instead of those of size 50. We found that this led to improvements in accuracy, especially since we did not further train these embeddings in the network due to computational reasons.

Secondly, we modified the network architecture slightly. We experimented with replacing the GRU with an LSTM as in [5], and similarly found that the GRU did indeed have higher test accuracy. However, we added a convolutional layer before the linear layer. Due to the increased dimensionality of our embeddings and the added convolutional layer, we also increased the output dimension of the GRU to 200, as compared to the optimal 150 found in [5]. We could more rigorously tune this hyperparameter using nested cross-validation, but this was not attempted in this project as it was not the main focus.

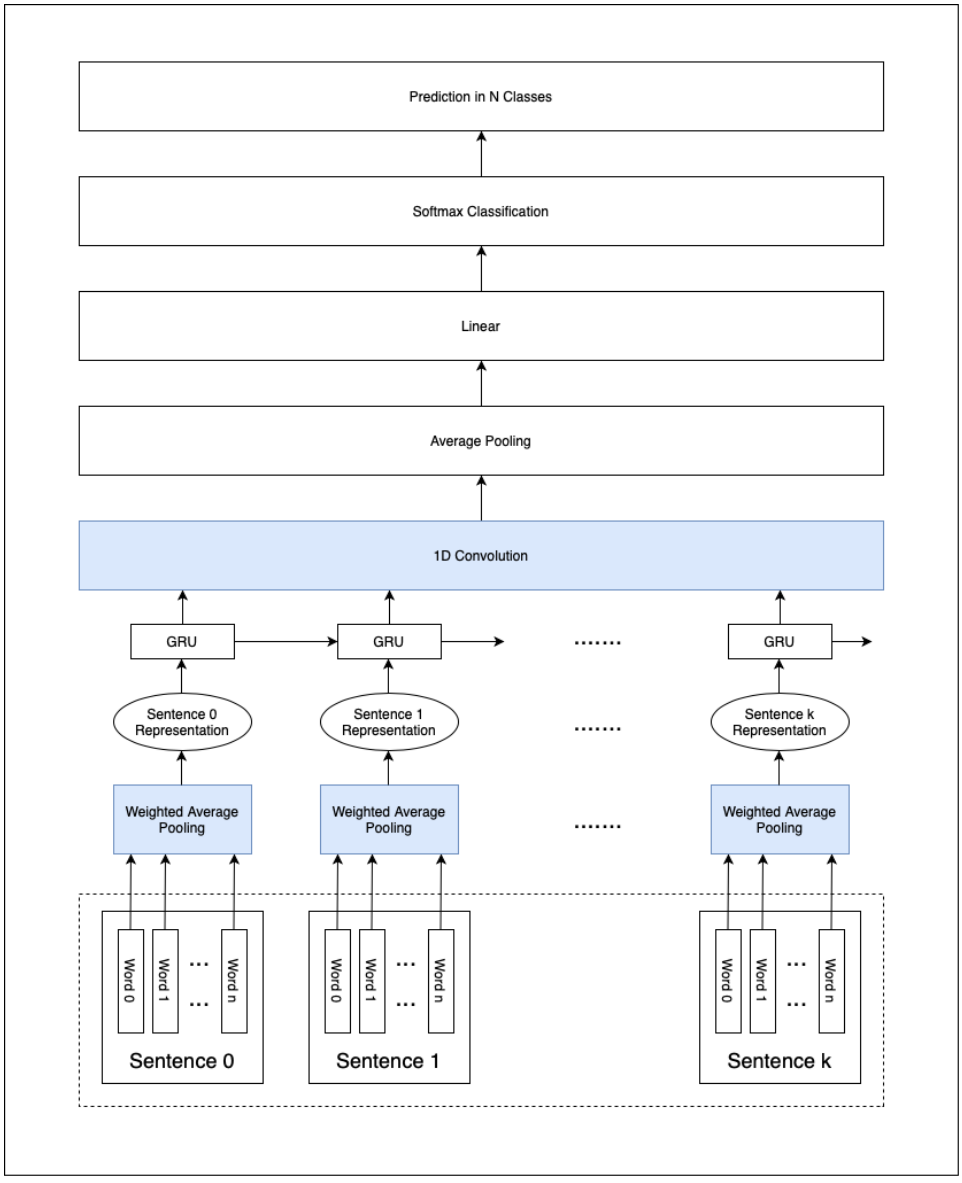
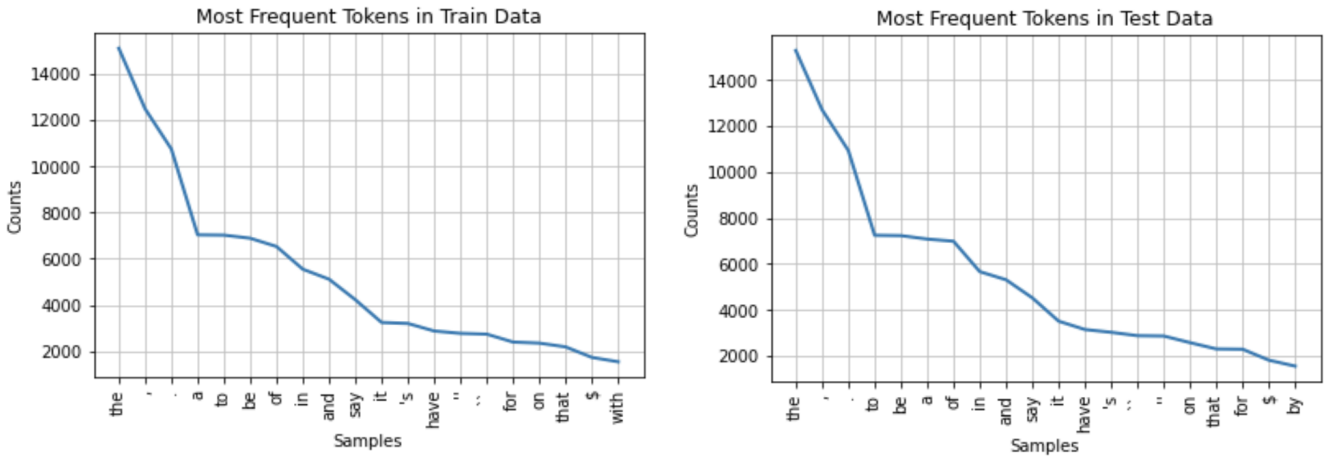


Figure 2: Architecture of article-level GRU, adapted from [5]. Main innovations are colored in blue.

The biggest major change we made was to introduce inverse document frequency (IDF) weighting of tokens in the model. As mentioned in Section 3.1, the original model represented each sentence by the average of all its words' vectors. We deemed this potentially inappropriate for the following reasons.

Firstly, the tokens that occurred most frequently in the train and test data were punctuation and stop words (Figure 3). Allowing these "non-informative" components to contribute equally to a sentence as other more "informative" tokens would cause our representations of two unique sentences to be more similar than desired. Therefore, we wanted to weigh unique words more. Secondly, we noted that the train and test data cumulatively had 2, 4 and 3 documents comprising only of 2, 3 and 4 sentences respectively. This meant that it was important for an individual sentence to

carry as much author-specific information as possible, which naturally led to IDF weighting.



(a) (b)

Figure 3: Tokens that occur most frequently in the train and test data.

To implement this, for each token w_i, we define it's weight $W_i = \log(10/A_i)$, where A_i is the number of authors who used token w_i in the train data. Each sentence is then the weighted average of its constituent tokens, with the weights normalized such that the cumulative sum of weights of tokens for each sentence is 1. If the sum of weights for a sentence was 0, then we simply took an average of the tokens in the sentence. This weighting process is also done for the test data, using the W_i calculated from the train data. Tokens that were not seen in the train data and hence had no W_i, were eliminated, although an alternative approach is discussed in Section 6.

4 Application on Synthetic Data

4.1 Probabilistic Solution

Prior to evaluating our algorithm, we first select optimal hyperparameters P and UNK. We do so with a simplified version of nested cross-validation whereby the test set is fixed - The training data is split into 3 (almost) equal folds, 2 for training and 1 for validation, and model accuracy is evaluated for each set of hyperparameters on the validation set. The validation set is then "rotated" twice more, and the total accuracy for each set of hyperparameters would be the sum of the 3 accuracy figures, weighted by the size of the validation set.

For this section, we consider $P \in \{100, 125, 150, 175, 200\}$ and $UNK \in \{0.0001, 0.001, 0.01, 0.1, 1\}$ (These values were chosen such that the validation accuracy was highest for the middle values). n could be one more hyperparameter we could tune, as done in section 5.1, but this would incur even higher computation cost for synthetic data - For each n, we would need to re-generate the synthetic training data. As such, we choose n = 6 as this gives the best validation accuracy when nested cross-validation was carried out with the Probabilistic solution on real data in Section 5.1. We also note that this selection of n generates text that is reasonably realistic. From our nested cross-validation applied to the synthetic train data, validation accuracy is highest when $(P, UNK) = \{(175, 0.001), (175, 0.01), (175, 0.1), (175, 1)\}$. Over the set of 25 accuracy figures, we also note that $UNK = 0.01$ yielded the highest average accuracy as compared to other selections of UNK. The choice of 0.01 also corresponds with our choice in Section 5.1. We therefore selected $P, UNK = 175, 0.01$.

With the chosen hyperparameters, we then train our algorithm on all the synthetic train data, and evaluate the algorithm on both the synthetic train and test data. Our algorithm achieves a training accuracy of 100% and a testing accuracy of 95.8%.

4.2 Neural Network Solution

To recover sentence structure from the list of synthetic tokens, we use `nltk`'s Treebank word detokenizer to convert the tokens into text, before splitting the text into sentences again using `nltk`'s Punkt sentence tokenizer. Processing is then done as per Section 2.1.

Following which, we process the synthetic text as per [5], additionally calculating IDF weights W_i, and train our algorithm on all the synthetic train data. Then, we evaluate the algorithm on both the synthetic train and test data.

Our algorithm achieves a training accuracy of 100% and a testing accuracy of 99.0%.

5 Application on Real Data

5.1 Probabilistic Solution

We now apply our probabilistic solution on real data, tuning our hyperparameters with the real train set and evaluating the algorithm with optimized hyperparameters on both the real train and test set. For hyperparameter tuning, we carry out nested cross-validation as described in Section 4.1, this time also optimizing for n. We consider $P \in \{75, 100, 125, 150, 175\}, UNK \in \{0.001, 0.005, 0.01, 0.05, 0.1\}$ and n $\in \{4, 5, 6, 7\}$, again choosing a grid of values such that the validation accuracy is highest in the middle of these values.

We find that validation accuracy is maximized for the following set of hyperparameters: $(P, UNK, \text{n}) = \{(150, 0.01, 6), (150, 0.01, 7)\}$. We additionally note that accuracy figures are the same for n = 6, 7. This is most probably because the set of 7-grams are sparse and are extremely unlikely to exist in the validation/test set. Due to computational savings, we therefore choose n = 6. On the whole, n = 6 also performed markedly better than n = 4, 5, which indicates that this selection of n could be more suitable for realistic text generation, justifying our choice of n = 6 in Section 2.2.

With the chosen hyperparameters, we then train our algorithm on all the real train data, and evaluate the algorithm on both the real train and test data. Our algorithm achieves a training accuracy of 100% and a testing accuracy of 79.0%, which is decent for a 10-class classification problem, but significantly lower than that for our synthetic data.

5.2 Neural Network Solution

We began with an implementation of the solution in [5] (with the assumed additional linear layer). This yields a testing accuracy of 69.8% for our 10-class classification problem, which is only slightly higher than the accuracy of 69.1% reported in [5], which dealt with the 50-class classification problem. This irregularity in accuracy is presumably for the following reasons:

- We simply use the GloVe vectors as embeddings, without further tuning the embeddings in the model, for computational reasons.
- The Stanford paper uses a 9-1 train-test split while we only use a 1-1 train-test split. The comparative lack of training data hurts our accuracy.

Then, we modified the algorithm as detailed in Section 3.2, which achieves a training accuracy of 100% and a testing accuracy of 87.0%. This provides support that the modifications we suggested were appropriate for the problem of author identification.

To assess this claim, we evaluate our algorithm on the 50-classification problem in [5]. We first take data from all 50 authors, and then reorganize the train and test data into a 9-1 split (by taking the first 40 documents of each author in the test data and moving it over to the train data instead). Then, we train our algorithm for the 50-class classification problem on the train data, and evaluate it on both the train and test data. Our algorithm achieves a training accuracy of 99.5% and a testing accuracy of 78.0%, which is markedly higher than the 69.1% reported in [5], despite our algorithm not optimizing the word embeddings within the network as done in [5], which would probably further boost the testing accuracy of our model. This further justifies the proposed modifications.

6 Discussion

6.1 Remarks About Neural Network Solution

Here, we provide a discussion about the treatment of tokens in the test data that were not seen in the train data. In our proposed algorithm, we suggested the simple approach of eliminating such tokens. This yielded test accuracy values of 99.0%, 87.0% and 78.0% for the synthetic 10-class problem, real 10-class problem, and real 50-class problem respectively.

However, we also experimented with a more complex alternative, which gave these tokens a weight of $\log(M)$, whereby M is the total number of authors in our train/test data. This makes the assumption that any token unseen in the train data, would be relatively rare and probably used by only one author in the test data. When this change was made, the accuracy values above changed to 99.2%, 89.0% and 76.2%.

A possible reason for this is that in the 9-1 train-test split, most words have been encountered in the train data, and the remaining words existing in the test data are "irrelevant" and do not provide useful author identification

information. However, the converse would be true for the 1-1 train-test split. Therefore, placing more weight on these unseen tokens was more beneficial for the even split. Indeed, when we inspected these "unseen" tokens, many of them for the 9-1 train-test split were pronouns and figures (but still had GloVe embeddings), while those for the 1-1 train-test split were considerably more meaningful. Future extensions can therefore look into better ways of filtering for "meaningful" words which would allow for this weighting. Another possible extension is to treat this weight as a hyperparameter, and use nested cross-validation (keeping the ratio of train : validation similar to that of (train + validation) : test) to optimize for this value.

6.2 Comparison of Probabilistic and Neural Network Solution

Comparing the probabilistic solution and the neural network solution, we note that the neural network solution has significantly higher accuracy for both synthetic and real data, which speaks to the expressiveness of the neural network solution. As expected, since the probabilistic solution was modelled after the generative probabilistic model, its performance was much higher on the synthetic data. It was, however, surprising that the neural network achieved even higher testing accuracy on the synthetic data, which is presumably due to the relatively simple structure of the synthetic data.

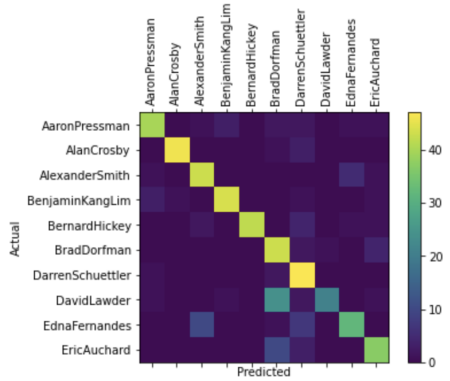


Figure 4: Confusion Matrix for 10 Author Probabilistic Solution on Real Test Data

```
[[40  0  1  3  0  2  2  0  1  1]
 [ 0 46  0  0  0  1  3  0  0  0]
 [ 1  0 43  0  0  0  0  0  5  1]
 [ 3  1  0 44  0  0  1  0  0  1]
 [ 0  0  2  0 42  0  4  0  1  1]
 [ 0  0  0  0  0 43  2  1  0  4]
 [ 1  0  0  0  0  2 47  0  0  0]
 [ 1  0  0  1  0 24  2 21  0  1]
 [ 0  0 10  0  0  1  7  0 32  0]
 [ 0  0  0  0  0 10  3  0  0 37]]
```

Figure 5: Probabilistic Confusion Matrix with Numbers

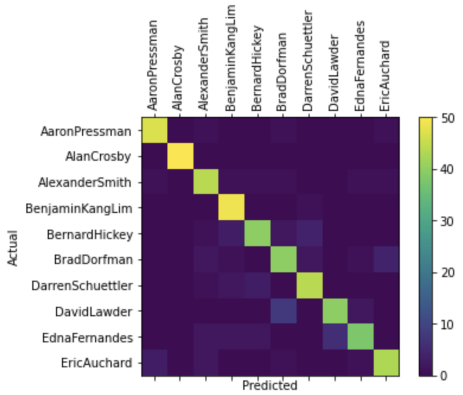


Figure 6: Confusion Matrix for 10 Author Neural Network Solution on Real Test Data

```
[[47  0  1  0  0  1  0  0  0  1]
 [ 0 50  0  0  0  0  0  0  0  0]
 [ 1  0 44  1  1  1  0  0  1  1]
 [ 0  0  0 49  0  0  1  0  0  0]
 [ 0  0  1  3 40  2  4  0  0  0]
 [ 0  0  2  1  0 40  2  0  1  4]
 [ 0  0  1  2  3  0 44  0  0  0]
 [ 0  0  0  0  0  8  0 40  2  0]
 [ 0  0  2  2  2  0  0  6 38  0]
 [ 3  0  2  0  0  1  0  0  1 43]]
```

Figure 7: Neural Network Confusion Matrix with Numbers

We further analyze both solutions' performance on real data. As we can see from Figure 5 and Figure 7, the neural network model has a better overall accuracy and consistency than the probabilistic model. However, it should be noted that the neural network solution performs worse for authors Bernard Hickey, Brad Dorfman and Darren Shuettler. Considering that the probabilistic solution only considered word to word sequences (up to 6 words), while the neural network solution focused on sentence-level structure, perhaps incorporating word-level structure would further improve our neural network solution.

In terms of data, we see that both solutions perform reasonably well with only half the data, and expect both solutions to generalize better with more data. It should, however, be noted that both proposed solutions do not

perform well on real data when data is extremely scarce, due to our individual implementations. The probabilistic model assigns a low weight to tokens unseen in the training data (author-specific), and the neural network solution disregards tokens unseen in the training data (non-author-specific) entirely. If training data is indeed extremely limited, the probabilistic model can be extended by representing rare words in the training data as an unknown token, and the neural network solution can explore options as detailed in 6.1.

For computational requirements, we note that the probabilistic solution is more computationally inexpensive than the neural network solution, incurring lower computational cost and time, which allowed us to easily tune our hyperparameters with nested cross-validation in a reasonable amount of time. Comparatively, the neural network took much more time and resources, especially for the 9-1 train-test, 50-class classification problem. Having said so, both processes can be parallelized, which would alleviate computational concerns. We also sped up training for the neural network solution by using mini batch gradient descent, with each batch containing one document from each author.

The probabilistic solution is also more interpretable, as we can fully understand the n-gram generative model and the resulting probabilistic decomposition of our solution. On the other hand, the neural network acts more like a black-box, and interpretability is gets progressively more difficult as we add more layers and increase the complexity of the architecture. Of course, our work on feature engineering (representing each sentence as an IDF-weighted average of its constituent tokens) does still provide us with some interpretability.

7 Conclusion

In conclusion, we present an n-gram generative probabilistic model with n = 6, which is trained on our train data. This model is then used to generate synthetic train and test data, which is used to evaluate our proposed probabilistic and neural net solutions. Our probabilistic solution is modeled after this generative probabilistic model while the neural net solution introduces key modifications to the article-level GRU framework proposed by [5]. As expected, the probabilistic model has high testing accuracy on the synthetic data, and reasonably high but significantly lower accuracy on the testing data. Comparatively, the neural net solution does better for both tasks, at the cost of training time and interpretability. In this paper, we also compare our neural net solution with that in [5] and deem our modifications to be appropriate. Further extensions of this work would include determining the suitability of an unknown token to represent rare words for the probabilistic solution, incorporating the word embeddings into the neural net, and extensive hyperparameter tuning for the neural net solution.

SUPPLEMENTAL MATERIALS

Code.ipynb: File that uses the data in `C50`, and implements the methods described in the paper to handle the 10-class classification problem, with a 1-1 train-test split. The code is ordered and labeled as per the assignment pdf.

50_authors.ipynb: Auxiliary file that uses the data in `C50`, and implements the methods described in the paper to handle the 50-class classification problem, with a 9-1 train-test split. This is mainly used for performance comparison with [5].

C50: Folder containing the data retrieved from [1].

References

[1] Z. Liu (2011). Reuter_50_50 Data Set. Retrieved from `https://archive.ics.uci.edu/ml/datasets/Reuter_50_50#`.

[2] D. D. Lewis, Y. Yang, T. G. Rose, F. Li (2004). RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, 5, 361-397.

[3] J. Houvardas, E. Stamatatos (2006). N-gram Feature Selection for Authorship Identification. *Artificial Intelligence: Methodology, Systems, and Applications*, 4183, 77-86.

[4] E. Stamatatos (2007). Author Identification Using Imbalanced and Limited Training Texts. *International Conference on Database and Expert Systems Applications.*

[5] C. Qian, T. He, R. Zhang (2007). Deep Learning based Authorship Identification. Retrieved from `https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2760185.pdf`.