Deep Learning Project:
Spooky Author Identification Using Deep Learning

Robin Khatri

January 11, 2020

Abstract

In this report, we have presented the usage of deep learning methods on a multi-class classification task of author identification. The dataset used is from an online competition and consists of texts of varying lengths from three different classical authors of the same genre. Our aim in this work was to predict probabilities of a text belonging to each author. We used natural language processing to get high-level features and back-translation as a data augmentation technique. For modelling, we used deep learning methods - both recurrent methods (LSTM with and without convolutional layers) as well as simpler models that were not very deep. We also looked at the impact of dropout regularization and data augmentation on training. We have also presented an ensemble of these methods and we have shown that it is possible to achieve good accuracy with good loss in author attribution task using deep learning. We also deployed our model successfully as an API which can process texts directly without the need of pre-processing.

Keywords: Author Identification, LSTM, Multi-class classification

Contents

List of Tables

List of Figures

1. Introduction

Spooky author identification[1] was an active competition at Kaggle[2] from October 25, 2017 to Dec 16, 2017. The data of the competition consists of short texts from three authors, namely Edgar Allan Poe[3], Howard Philips Lovercraft[4] and Mary Wollstonecraft Shelley[5]. Two of these authors - Edgar Allan Poe and Mary Shelley were contemporaries. The genre of their works and of the texts in the dataset of this competition is Horror fiction.

The following text is a relatively longer sample from the dataset:

"I wept for a long time untill I saw him about to revive, when horror and misery again recurred, and the tide of my sensations rolled back to their former channel: with a terror I could not restrain I sprung up and fled, with winged speed, along the paths of the wood and across the fields untill nearly dead I reached our house and just ordering the servants to seek my father at the spot I indicated, I shut myself up in my own room. Mr. Kirwin charged himself with every care of collecting witnesses and arranging my defence." - Mary Shelley (Mathilda, 1819).

To procced with the task of author identification, we defined our objective to be the same as the objective of official competition. The objective of the competition was to predict authors based on a text sequence. As per the official guidelines of this competition, one has to predict probabilities of a text to belong to each of the three authors. Thus, the problem becomes a multi-class classification and we want to compute probability of a text to be written by an author.

Author identification problem can be traced back to some 19th and 20th works. In 19th century, physicist Thomas Mendenhall analysed the works of Shakespeare, Marlowe and Bacon based on the word frequencies of each author, essentially developing a writing style for each author. His work was one of the first pioneering works of stylometry. "Stylometry is the application of quantitative analysis, primarily to written language, to identify variations in style." [22]. Different authors have different writing styles and these styles can aid in linguistically analysing texts [4] [20]. Further, it is also possible to quantify and statistically analyse these writing styles [3] [21]. Now-a-days, With the advent of natural language processing techniques and the advancements made in deep learning especially in recurrent networks, there has been many research works in the area of text classification and also specifically on the author attribution [13][25][11][10][26]. In particular recurrent networks such as LSTM and GRU are extensively used in the text classification [2][8]. [18] uses TF-IDF and deep learning for author identification. Inspired by this work as well as other stated works, we focused on a good pre-processing and data augmentation

[1]Website of the competition: https://www.kaggle.com/c/spooky-author-identification

[2]Kaggle is an online community of machine learning engineers and data scientists: https://www.kaggle.com

[3]More on Edgar Allan Poe: https://en.wikipedia.org/wiki/Edgar_Allan_Poe

[4]More on H.P. Lovecraft: https://en.wikipedia.org/wiki/H._P._Lovecraft

[5]More on Mary Shelley https://en.wikipedia.org/wiki/Mary_Shelley

techniques along with using recurrent as well as not recurrent methods to fulfill our task. We formulate our task in the following section, thereafter we have discussed the dataset as well as our objective function.

2. Problem Definition

In this section, we have defined our problem as well as our approach to solve it.

Author identification problem can be formulated as follows: Given a training set $\{S_1, S_2, ..., S_N\}$, where each set $S_i; i \in [0, N]$ is a text sequence, and a set of authors $\{A_1, A_2, A_3, ...A_m\}$ where each $A_j; j \in [1, m]$ represents an author, construct a method to compute a set of probabilties $\{p_i^{A_1}, p_i^{A_1}, ..., p_i^{A_m}\}$ for a new text sequence S_t while minimizing some objective function O, where $p_i^{A_j}$ is the probability of S_t to be written by author A_j. For our dataset, we have defined our objective O in the Equation 3.2.

2.1 Approach for Solving the Problem

We approached our problem as a multi-class text classification. We utilized various pre-processing as well as data augmentation techniques as described in the section 4. Based on these, we started with the simplest models keeping in mind the Occam's razor [6] and thereafter moved on to advanced recurrent and ensemble models. To build an ensemble, we used classifiers that do not by themselves capture enough information but when trained to do independent tasks, they provided better classification results.

3. Dataset Description and Experimental Setup

The dataset provided in the competition is divided into two parts - train and test.

Train data	**Test data size**
19579×3	8392×3

Table 1: Data size.

Throughout the report, **EAP**, **HPL** and **MWS** refer to Edgar Allan Poe, H.P. Lovecraft and Mary Shelly respectively.

The training data is labelled and is of the following format:

[6]Occam's razor (Novacula Occami): The species should not be multiplied without necessity. (https://www.britannica.com/topic/Occams-razor)

id	**text**	**author**
id26305	This process, however, afforded me no means of...	EAP
id17569	It never once occurred to me that the fumbling...	HPL
id11008	In his left hand was a gold snuff box, from wh...	EAP
id27763	How lovely is spring As we looked from Windsor...	MWS

Table 2: Data Format. **id**: Identifier for the text string. **text**: texts (variable length). and **author**: Corresponding Author. *e.g.* for **id17569**, the author is **HPL** and the complete text is, *"It never once occurred to me that the fumbling might be a mere mistake."*

One example text for each of the author:
EAP: *"Milder I half smiled in my agony as I thought of such application of such a term."*
HPL: *"It was not allied to the European witch cult, and was virtually unknown beyond its members"*
MWS: *"Whither does this lead?"*

3.1 Performance Evaluation

For training our models, we used categorical cross entropy, also known as multi class log loss. If we have N examples and M classes, and the computed probability of i^{th} example to belong to class j is p_{ij}, the loss can be defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_i^j log(p_i^j), \tag{3.1}$$

where $y_i^j = 1$, if i is in class j, else $y_i^j = 0$.

In our problem, we have three classes, namely **EAP**, **HPL** and **MWS**, so the loss function is,

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i^{\mathbf{EAP}} log(p_i^{\mathbf{EAP}}) + y_i^{\mathbf{HPL}} log(p_i^{\mathbf{HPL}}) + y_i^{\mathbf{MWS}} log(p_i^{\mathbf{MWS}}), \tag{3.2}$$

This is also the function used by the competition to assess performance on the test set. Two losses are calculated on different subsets of the test set, providing namely the public and private score. When this competition was active, only the public score was given at the time of submissions. For the final ranking, the competition used private scores for two best public score submissions. During our evaluation, we, therefore looked at public scores and for our final evaluation, we looked at the private scores as well as deviation of our final model(s) from the public score. This deviation between public and private scores will be interesting to check the performance of model on different sets of data. We also did hyperparamter tuning as well as cross-validation to get the best generalized models. In the following section, we have presented our pre-processing as well as data augmentation method.

4. Pre-processing and Feature Engineering

Firstly, we created three different columns from the column **author** by one hot encoding. To see the high-level differences between authors, we looked at the lengths of text sequences of training set by individual authors, number of characters and number of unique words used. Other features were number of punctuation marks used, mean text lengths, number of stopwords used, number of uppercase and titlecase words (*e.g.* "THE" is an uppercase word and "The" is a titlecase word). Figure 1 is the correlation matrix of these features with other features including our classes.

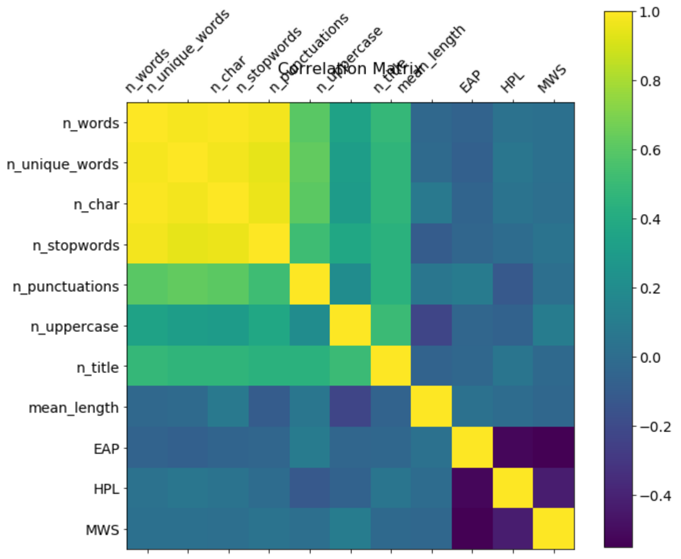


Figure 1: Correlation matrix between features and classes.

Clearly, features n_words (Count of words), n_chars (Count of characters), n_stopwords (Count of stopwords) and n_unique_words (Count of unique words) have a high positive correlation among themselves.

These correlations are numerically represented in the Table 3.

Features	**EAP**	**HPL**	**MWS**
n_words	-0.055617	0.035682	0.024094
n_unique_words	-0.075854	0.061497	0.020287
n_char	-0.052609	0.040393	0.016284
n_stopwords	-0.036248	-0.002282	0.040732
n_punctuations	0.092105	-0.11667	0.016527
n_uppercase	-0.037984	-0.063259	0.102336
n_title	-0.030793	0.051317	-0.017591
mean_length	0.028093	0.000395	-0.030221

Table 3: Correlation between features and author classes.

While the correlations between authors and features are not large, there are some differences with their correlations with individual authors, *e.g.* n_words, n_words, n_char, n_stopwords are negatively correlated with class **EAP** but positively correlated with other authors. Features mean_length and n_title are negatively correlated with **MWS** while positively correlated with **HPL** and mean_length is positively correlated with **HPL**.

The figure 2 represents the most frequent words used over all authors while the figures 3, 4 and 5 show the most frequent words used by each author. Evidently, words like "upon", "time", and "will" are used by all authors quite frequently.

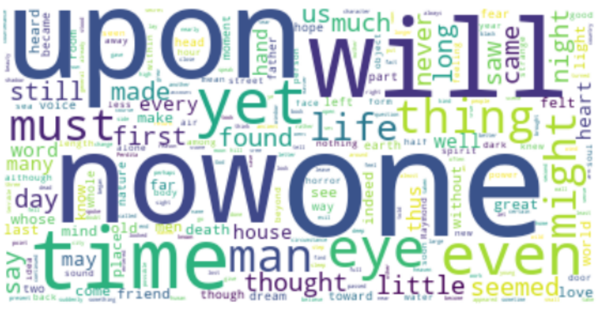


Figure 2: Word cloud for entire training texts.

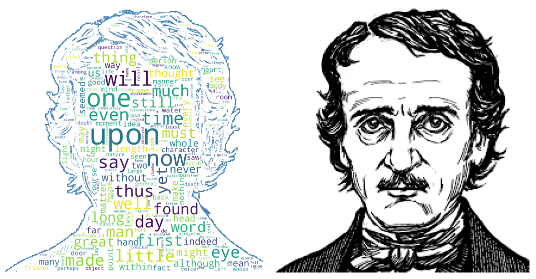


Figure 3: Word cloud for EAP (Edgar Allan Poe).

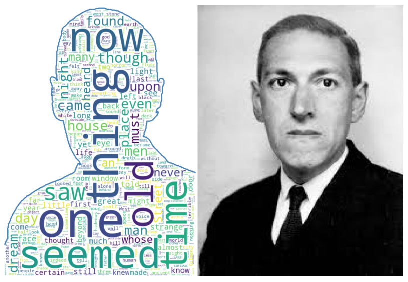


Figure 4: Word cloud for HPL (H.P. Lovecraft).

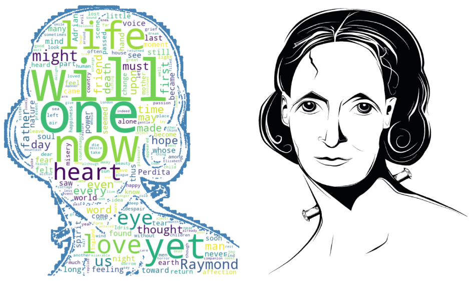


Figure 5: Word cloud for MWS (Mary Shelley).

Since there are obvious, however little, differences between these features and classes and there are many common words used by all authors, it is interesting to look at their usage of n-grams [7] (after removing stopwords (from nltk [12])). The frequency usage of 2-,3- and 4-grams of the authors are presented in the tables 4, 5 and 6.

2 gram	**Count**	**3 gram**	**Count**	**4 gram**	**Count**
let us	50	madame l espanaye	12	general john b c	12
one two	23	john b c	12	brigadier general john b	7
three four	23	general john b	12	ugh ugh ugh ugh	6
l etoile	23	ha ha ha	11	brevet brigadier general john	6
ha ha	22	barrière du roule	10	john b c smith	6

Table 4: 2-, 3- and 4- gram frequencies for Edgar Allan Poe (**EAP** class).

[7]An n-gram is a continuous sequence of n-terms. These terms can either be characters or words. *e.g.*"Here I go" is a 3-gram and "Here I" is a 2-gram [9]

2 gram	**Count**	**3 gram**	**Count**	**4 gram**	**Count**
old man	59	terrible old man	9	aira city marble beryl	4
could see	31	heh heh heh	9	oonai city lutes dancing	4
one night	23	charles le sorcier	8	eric moreland clapham lee	4
old woman	21	small paned windows	8	mad arab abdul alhazred	4
one might	18	great great great	6	necronomicon mad arab abdul	4

Table 5: 2-, 3- and 4- gram frequencies for H.P. Lovercraft (**HPL** class).

2 gram	**Count**	**3 gram**	**Count**	**4 gram**	**Count**
old man	29	let us go	4	next day next hour	2
lord raymond	28	months elapsed since	3	one day may claim	2
fellow creatures	22	time lord raymond	3	five years old mother	2
one day	20	nearly two years	3	day may claim hands	2
let us	16	first rank among	3	like thousand packs wolves	2

Table 6: 2-, 3- and 4- gram frequencies for Mary Shelley (**MWS** class).

So, although there are some most-used words overall that are used by all authors frequently, there are different in their usage of 2-,3- and 4-grams. We also noticed that there were some errors in the formatting of sentences. This was expected as as per the competition, there were some odd sentences and mistakes in parsing of the text [1].

To use n-grams, we used tfidf and count vectorizers from module `sklearn.feature_extraction.text` [14]. tfidf vectorizer [8]converts the text data to a matrix of TF-IDF (term frequency - Inverse document frequency) features, and count vectorizer converts the text data to a matrix of counts of each index token. For both the Count and TF-IDF vectorizers, we transformed the text data to vectors for 1-3 and 3-7 grams and in both cases, we kept the distinction between lower and upper cases since we have seen there is some difference, however small, among the authors in their usage of capital letters (as signified by features n_title and n_upper).

In aspiration of using these features later on, we added these features (including TFIDF and Count vector matrices) to our original train set creating a separate train set containing them.

4.1 Data Augmentation

Our dataset is relatively small with 19579 text sequences and many of these sequences are not very long as evident in the distribution of number of words in the texts of each author

[8]tfidf and count vectorizers of sklearn:https://scikit-learn.org/stable/modules/classes.htmlmodule-sklearn.feature_extraction.text

in the Figure 6.

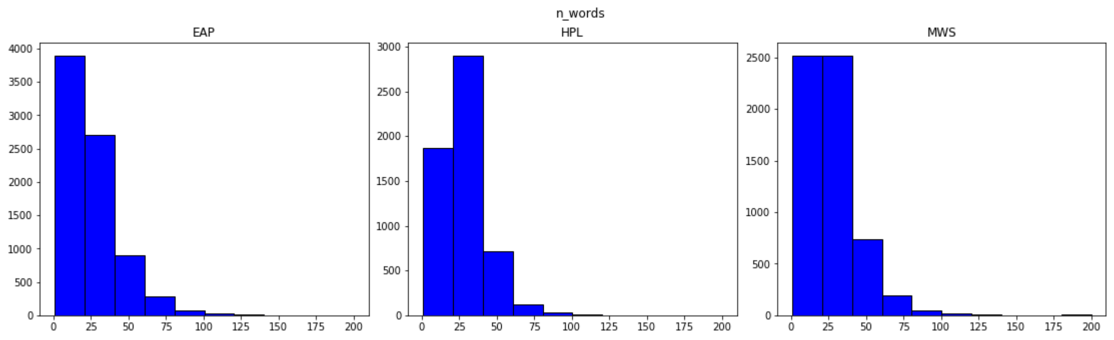


Figure 6: Distribution of number of words per text sequence for each author. From left to right - **EAP, HPL, MWS**.

Most text sequences are within the range of 0-50. To augment text data, therefore slicing through texts is not a good option here. To better do the data augmentation task, we used back-translation [19][7][23]. We translated dataset from English to French and then back to English using an API googletrans [9] that uses Google Translate [10] for processing translation requests. Even though the API requests to the API googletrans are unlimited, due to some compatibility issues, the resource limit ran out after around 500 text translations, therefore, we could only do the translation from English to French for 500 texts. Thereafter, we translated the French texts to English and added to our dataset. We kept the datasets both with back-translation as well as without back-translation to compare the performance. The comparison is made in the section 5. and it has shown to improve the performance on the validation sets. This is understandable since back-translation has not only increased the size of the data but has also provided a regularization due to translations not being perfect [24].

5. Deep Neural Networks (Without Recurrent Layers)

We started from a simple model without any recurrent layer. We used a tokenizer from keras [6]' preprocessing module to tokenize the sequences with parameters num_words=20000 (The maximum number of words to keep) and lower=True (*i.e.* covert the text to lowercase). The input list of sequences were padded with a maximum length of 256.

[9]googletrans: https://pypi.org/project/googletrans/
[10]Google Translate: https://translate.google.com/

Layer (type)	**Output shape**	**Parameters**
embedding_1 (Embedding)	(None, 256, 20)	521180
global_average_pooling1d_1	(None, 20)	0
dense_1 (Dense)	(None, 3)	63
Total Params: 521,243	Trainable params: 521,243	Non-trainable params: 0

Table 7: Model (Without recurrent layers) Architecture.

To train the model, we used early stopping callback option with a patience of 10 epoch to monitor the validation categorical cross-entropy. For training, 10% of data was used for validation.

To compare the performance of our data augmentation, we trained two models for data with and without the back-translation data augmentation. The data was preprocessed to remove stopwords. The results are presented in figures 7 and 8 respectively.

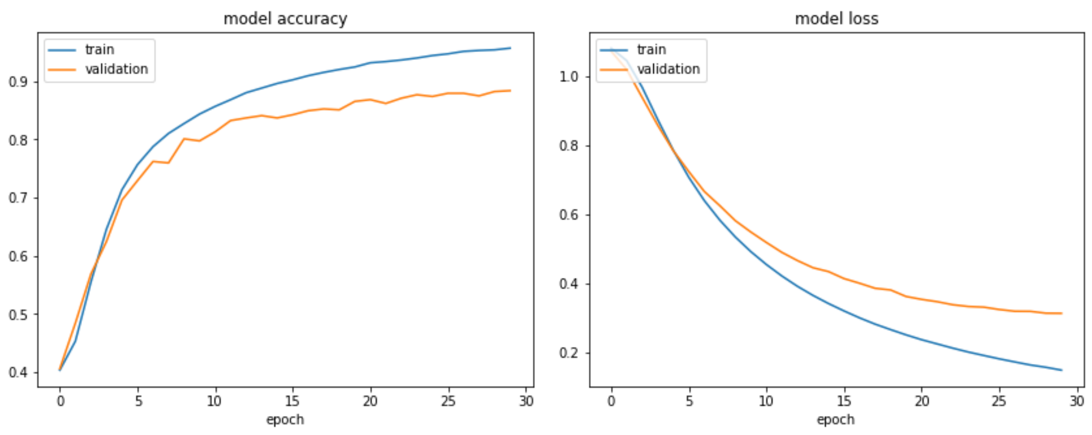


Figure 7: Model loss (Right) and accuracy (Left) - With back translation (Validation loss = 0.3128).

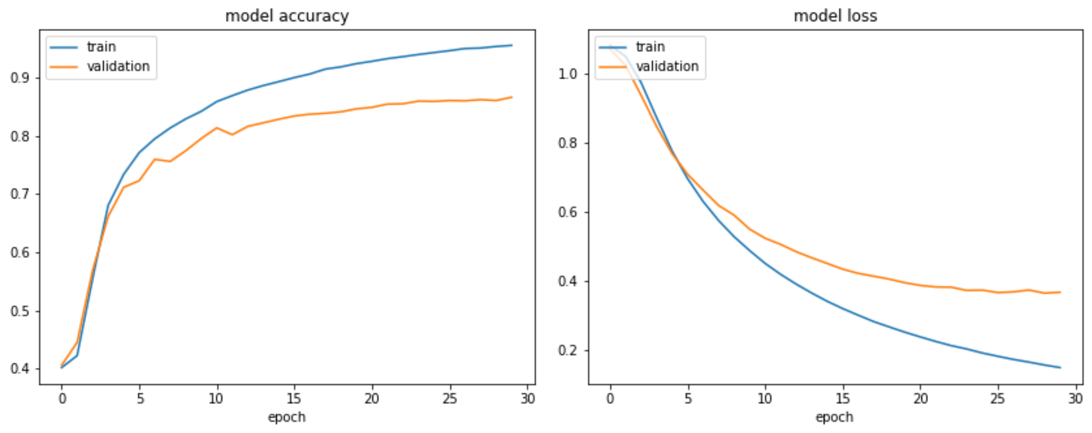


Figure 8: Model loss (Right) and accuracy (Left) - Without back translation (Validation loss = 0.368).

For both these train sets, the model seems to converge after 30 epochs and the loss curve has reached a plateau. For our data augmentation, we used only 500 training samples to translate to French and then back to English. Doing this resulted in an accuracy increase (0.88 vs 0.86) and loss decrease (0.3128 vs 0.368). Also, the public scores on the test set were 0.385 and 0.396 respectively for with and without data augmentation. The private scores on the other hand were much better for the back translation (0.365 as opposed to 0.38). There is clearly an overfitting due to the small data size and when we do data augmentation using back translation, it has led to some regularization as well because the translations are not perfect. Therefore, we decided to use the data with back translation augmentation for all our models.

We also trained a model without removing stopwords. This was interesting to check because during our preprocessing, we have seen that there was some correlation between the author class and the number of stopwords used. The architecture of this model is the same as before and as mentioned, we have included the back translated data as well. To test the convergence of the mdoel more accurately, we decided to increase the number of epochs to 50. The results are presented below in the figure 11

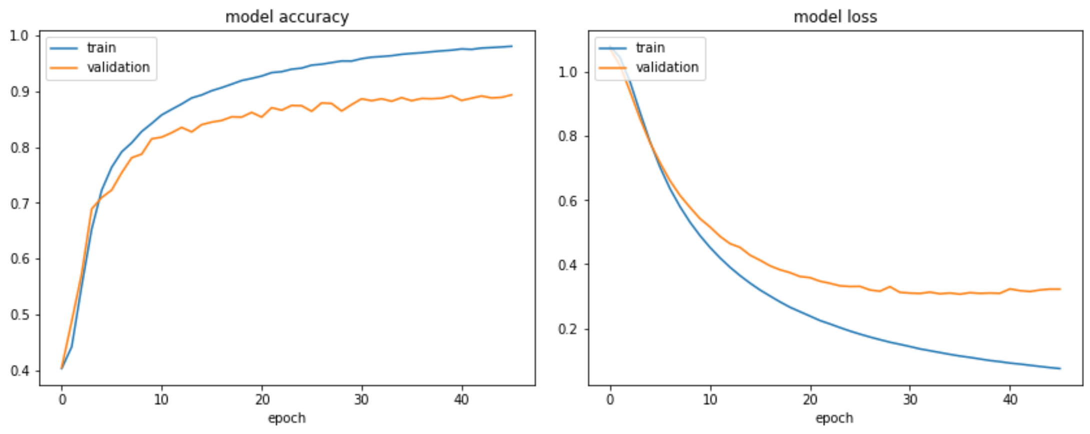


Figure 9: Model loss (Right) and accuracy (Left) - Without removing stopwords.

There is a slightly upward trend in the loss curve after 35 epochs. Therefore, we decided to keep the model after 30 epochs as we do not want to overfit. The validation loss was 0.31 and on kaggle test set, the public score obtained was 0.37 with private score being 0.360. Since, our validation loss is slightly better without removing the stopwords. We decided to keep the stopwords as they seem to provide additional information about an author's literary style.

5.1 Transfer Learning with Pre-trained GloVe Embedding:

To better train the neural networks, it is a common approach to use pre-trained models or a part of models. To test this hypothesis in our dataset, we also trained a model with pre-trained embeddings. In particular, we trained two models with "trainable" hyperpramater = True (*i.e.*to retrain the embedding to suit our data) and False (*i.e.* to use the embedding as it is). We used Global Vectors for word representation (GloVe) embedding with 50 dimensions vectors [15] with callback option to monitor decrease in log loss with patience of 10 epochs. The model was stoppped after 25 epochs of initial 30 epochs as the validation log loss did not improve after 10 epochs. With trainable = False, the performance was worse with validation log loss 0.85 and validation accuracy 0.62 as expected, since the model was not retrained for our data. For trainable = False, The validation log loss was 0.38 which is substantially less than our validation log-loss of 0.312 and validation accuracy of 0.87 from our previous model. On test set as well, the public log loss was 0.41 and private log loss was 0.39.

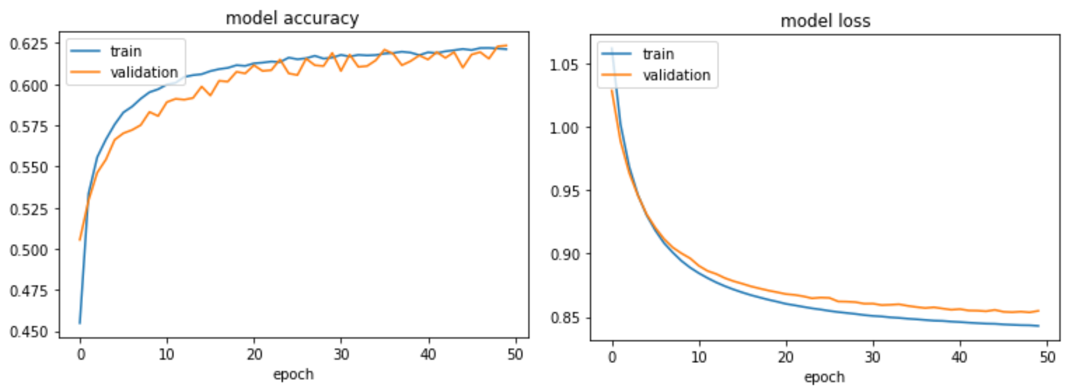


Figure 10: Model loss (Right) and accuracy (Left) - Without retraining GloVe embedding.

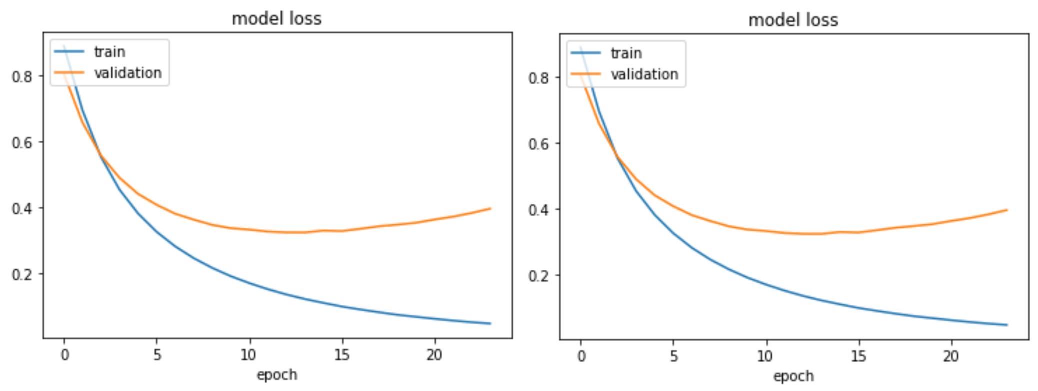


Figure 11: Model loss (Right) and accuracy (Left) - Retraining GloVe embedding.

Since the pre-trained embedding did not provide much advantage in our training, infact the performance was slightly worse even after retraining for our data, in the best model so far, we decided to not include the pre-trained embeddings. We further tried 100d as well as 300d versions of the GloVe embeddings, however, the performance was similar as the 50d.

6. Recurrent Networks

In this section, we have explored the training on the Spooky Author Identification dataset using recurrent as well as convolutional layers in our models. The training dataset included the dataset with back-translation as the performance has been better as seen in figures 7 and 8. We started by including one LSTM layer in our model. The architecture of this model is given below.

Layer (type)	**Output shape**	**Parameters**
embedding_8 (Embedding)	(None, None, 128)	3335552
dropout_13 (Dropout)	(None, None, 128)	0
lstm_3 (LSTM)	(None, None, 128)	131584
global_max_pooling1d_8	(None, 128)	0
dense_17 (Dense)	(None, 3)	387
Total params: 3,467,523	Trainable params: 3,467,523	Non-trainable params: 0

Table 8: Recurrent Model Architecture.

We trained this model with our tokenized training set without removing stopwords (An observation made in the Section 5.) We used callbacks to monitor the validation log-loss with a patience of 3. The model stoppped after 4 epochs as the validation loss didn't decrease. The performance of this model is substantially lower than what we have seen in our previous model. The validation log loss was 0.40 with validation accuracy 0.87. We decided to try the model without adding the dropout layer after embedding layer to see if we can improve. The log-loss on the training set decreased, however the validation log-loss increased which was expected because here we are clearly overfitting and not regularizing by removing the dropout layer. The validation accuracy and log loss curves are given in figure 13 .

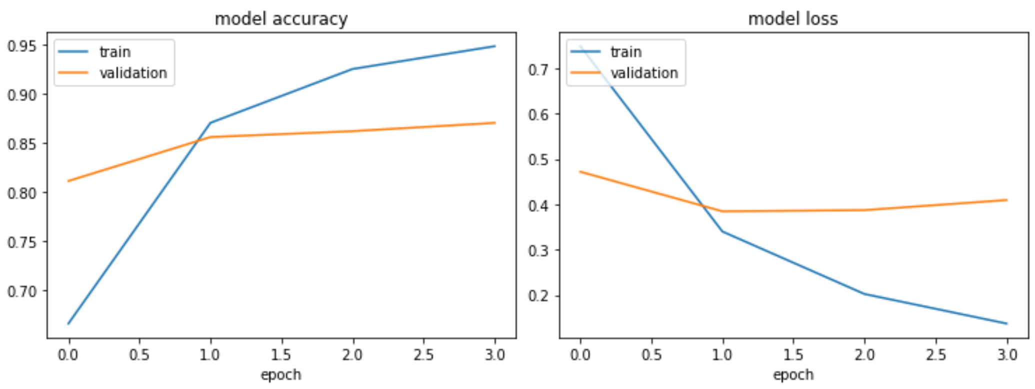


Figure 12: Model loss (Right) and accuracy (Left) - With dropout.

Clearly, the model's validation log-loss (With and without dropout layers) increased after 2 epochs and even stopped after 3 epochs by including a callbacks. Therefore, we kept the model after training of 2 epochs and we observed a validation log loss of 0.38. The Kaggle score on this model was 0.44.

Next, we decided to include bidirectional recurrent layer followed by a convolutional layer in our model. After running several experiments, we also decided to reduce the output shapes of the embedding layer thus decreasing the number of parameters to be trained. For convolutions we chose the kernel size to be 3 (We also experimented with other sizes and found this to be the best). The model's architecture is given below.

Layer (type)	**Output shape**	**Parameters**
embedding_1 (Embedding)	(None, None, 30)	781770
dropout_1 (Dropout)	(None, None, 30)	0
bidirectional_1 (Bidirectional)	(None, None, 256)	122112
conv1d_1 (Conv1D)	(None, None, 64)	49216
global_max_pooling1d_1	(None, 64)	0
dense_1 (Dense)	(None, 15)	975
dense_2 (Dense)	(None, 3)	48
Total params: 954,121	Trainable params: 954,121	Non-trainable params: 0

Table 9: Recurrent Model Architecture with Bidirectional and Convolutional layers (Kernel size=3).

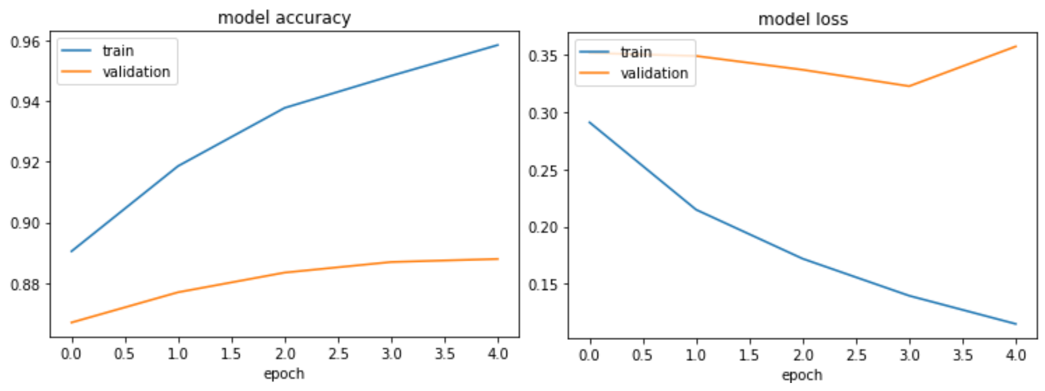


Figure 13: Model loss (Right) and accuracy (Left) - Without the convolutional layer. The Validation loss increased after the 3rd epochs and the training stopped after the callback option and did not complete specified 5 epochs. We kept the model till the 3rd epoch.

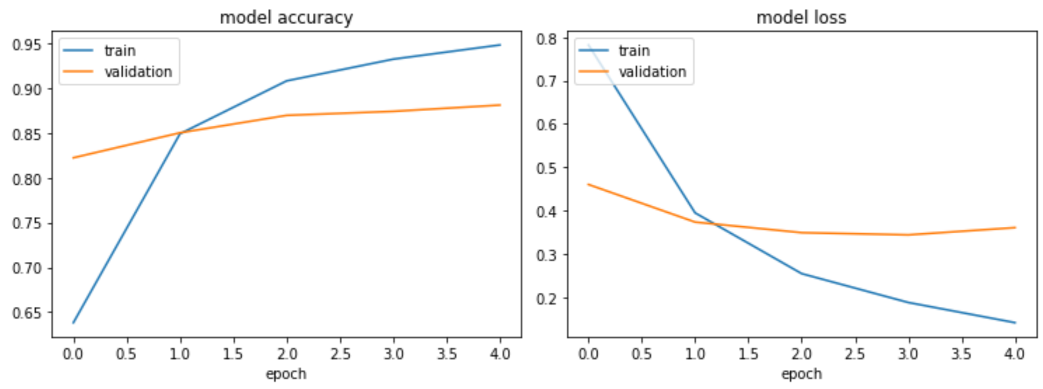


Figure 14: Model loss (Right) and accuracy (Left) - With the convolutional layer. The validation accuracy (0.88) plateau after the 3rd epochs and same for validation loss (0.36). We kept the model till the 3rd epoch.

We tested the model by excluding the convolutional layer from our model. After training for 3 epochs the model had a validation log loss of 0.33 and validation accuracy of 0.89. On the kaggle test set, the score was 0.42 (Public: 0.44, Private: 0.42) which is better than our last model. Therefore, we found this to be our best recurrent model. The loss and accuracy curves are given in the Figure 14.

7. An Ensemble of Deep Neural Networks

To test whether an ensemble can be good on our dataset, we decided to naively take weighted average of predictions from the best models described in the sections 5. and 6.. The individual test log loss (Public) for these models were 0.385 and 0.44 respectively. By averaging (80 for the best model from the section 5. and 20 for the best model from the section 6.), we observed a validation log loss of 0.30 and on Kaggle, the score was 0.380 (public) and 0.350 (private). This motivated us to further utilize the ensemble of neural networks. This also opened a room for utilizing the engineering features and information that we gathered in 4..

We looked to ensemble three independent models that were trained to do independent tasks as mentioned in the Table 11. To select the best generalized model on all subsets of training and validation sets, we used KFold splits (K=3). In training these individual models, we were not looking to select the model that performs best on a validation set but rather, the model that performs best on all splits of train-validation sets. We did not take into account the test set for selecting the best model for each of these settings. For cross-validating and generating output predictions for both training and test set, we created three different functions - one for each of the methods, to better use them. The time complexity of cross-validation is high ($\tilde{1}$ hour on a GPU for the longest running model for one 3 fold validation), however, since the data is small, the run time is not very high as we can expect in a bigger size data. To later use output predictions by these models, we saved two sets of these features - one containing only the model predictions, and one containing the model predictions as well as the features described in the Section 4.

To gain information from the TF-IDF and Count features, we used a model which has architecture as in Table 10.

Layer (type)	**Output Shape**	**Parameters**
dense_1 (Dense)	(None, 10)	8850130
dense_2 (Dense)	(None, 3)	33
Total params: 8,850,163	Trainable params: 8,850,163	Non-trainable params: 0

Table 10: Model for TF-IDF and Count-vectorized matrices.

The validation accuracy and categorical cross-entropy curves for this model (trained on TF-IDF vectors) are given in the Figure 15. The model achieves a validation loss of 0.31.

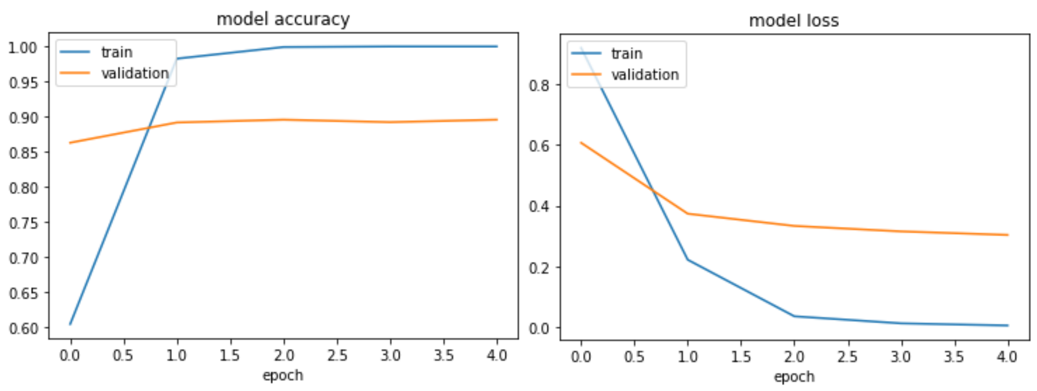
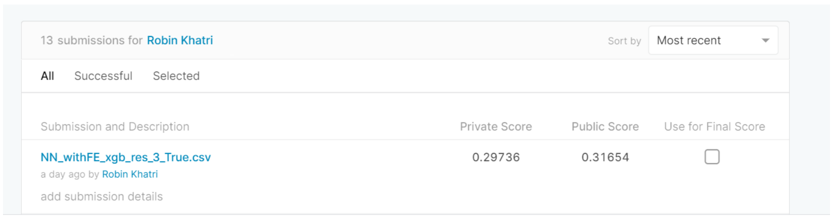


Figure 15: Validation loss (left) and accuracy (Right) for the model trained with TF-IDF. The validation loss as well as accuracy converges after 1 epoch.

Model	**Input**
Recurrent without Bidirectional, convolutional and Max pooling layers (As described in the section 6.)	Padded tokenized sequences
Without recurrent layers (As described in the section 5.)	Padded Tokenized sequences
Model with 1 hidden Dense layer (As described in the Table 10	TF-IDF 1-3 gram
Model with 1 hidden layer (As described in the Table 10)	TF-IDF 3-7 gram
Model with 1 hidden layer (As described in the Table 10)	Count (TF) 1-3 gram
Model with 1 hidden layer (As described in the Table 10)	Count (TF) 3-7 gram

Table 11: Models used in ensemble after cross-validation.


13 submissions for Robin Khatri

Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
NN_withFE_xgb_res_3_True.csv a day ago by Robin Khatri add submission details	0.29736	0.31654	☐

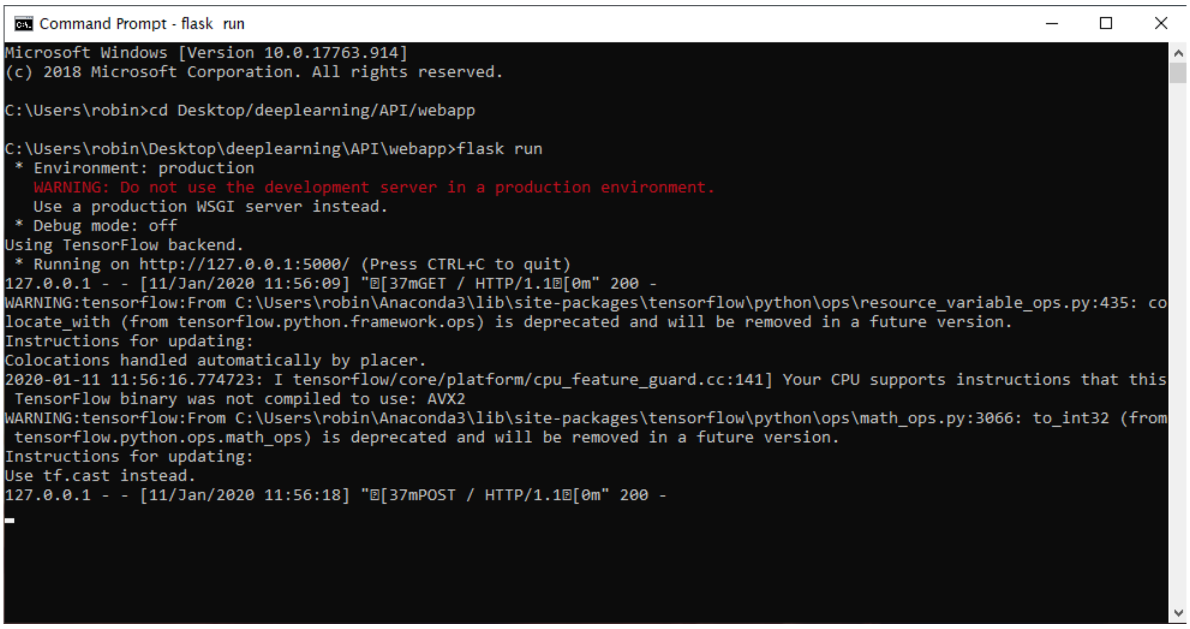
Figure 16: Kaggle submission page for prediction with an ensemble of neural network outputs.

For our final recurrent model to be included in the ensemble, we used a model with an LSTM layer as described in the Section 6. but without the pooling layer since it provided better performance over all validation splits. We also made slight changes in the maximum padded sequence length (150 as opposed to 256 previously. This was done for the same reason as well). To ensemble the models, we used xgboost [5]. For ensemble itself, we used KFold cross-validation (K=3), and we predicted the outcomes on the test set. The validation loss for this ensemble was 0.31 over all 3-Fold splits with the Kaggle score being 0.29 as seen in the figure 16. By including the selected engineered features (n_words, n_counts, n_stopwords, n_punctuation, n_char and n_mean_length) from the section 4. in our ensemble, we were able to further improved the model with a private score of 0.28 which would have placed us comfortably in the top 200 of the Kaggle leaderboard if the competition was still active.

8. Deployment

We deployed the model using flask, Keras and Deep Learning REST API [17]. This API uses our best individual model (Validation log loss 0.31 and Kaggle score of 0.35) as opposed to ensemble which has a much better Kaggle score of 0.29. We built an API for a simpler model for providing better time complexity. The API is optimized to process text fast. This is particularly helpful since to make predictions, the model first has to tokenize as well as pad the text sequences. On our machine, once loaded in the browser (we request the user to follow the command instructions and use a Chrome [11] browser), API was able to process the requests within 1-3 seconds. It can be run using command `flask run in the webapp directory` and can be used as an web application. The interface accepts text of any size and outputs proabbailties for it to be written by each author. The demonstration is present in the figures 18 and 17.

[11]Google Chrome: https://www.google.com/chrome/

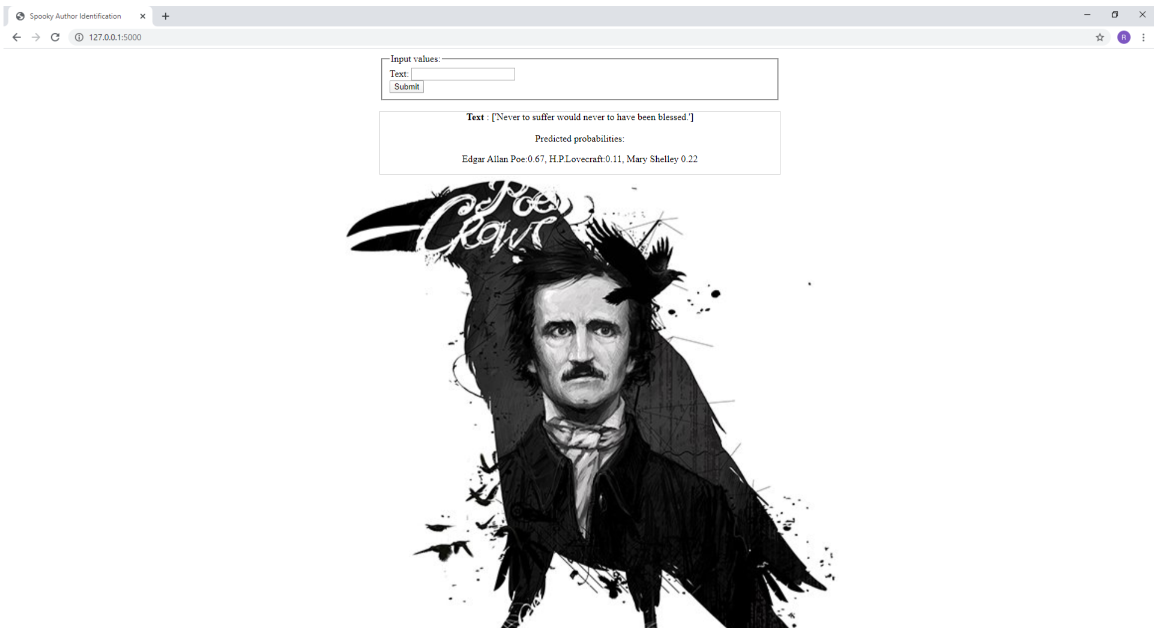


```
Command Prompt - flask  run
Microsoft Windows [Version 10.0.17763.914]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\robin>cd Desktop/deeplearning/API/webapp

C:\Users\robin\Desktop\deeplearning\API\webapp>flask run
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
Using TensorFlow backend.
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [11/Jan/2020 11:56:09] "[37mGET / HTTP/1.1[0m" 200 -
WARNING:tensorflow:From C:\Users\robin\Anaconda3\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:435: co
locate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
2020-01-11 11:56:16.774723: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this
 TensorFlow binary was not compiled to use: AVX2
WARNING:tensorflow:From C:\Users\robin\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
 tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
127.0.0.1 - - [11/Jan/2020 11:56:18] "[37mPOST / HTTP/1.1[0m" 200 -
```

Figure 17: Model Deployment - API server.



Spooky Author Identification

127.0.0.1:5000

Input values:

Text:

Submit

Text : ['Never to suffer would never to have been blessed.']

Predicted probabilities:

Edgar Allan Poe:0.67, H.P.Lovecraft:0.11, Mary Shelley 0.22

Figure 18: Model Deployment - Web application Interface

9. Conclusion and Further Work

We have shown that with good preprocessing and feature engineering techniques such as data augmentation using back-translation and by constructing matrices of n-gram vectors, it is possible to build models even with simple architectures to accomplish a multi-class classification task on a small text dataset. We also see that it is best to start from simpler models to avoid the risk of overfitting as well as to gain idea about baselines. With our pre-processing and data augmentation, we accomplished around 90% validation accuracy and validation log-loss of 0.31 with test validation score of 0.35 (which as per the competition leaderboard) is quite good for a pure deep learning based method. By using advanced models such as recurrent layers in addition to simple models, we have shown that we can further improve the model. Finally, by ensembling the models that were relatively weak individually, we achieved a much higher accuracy as well as a good log-loss on various validation sets as well as on the test set. With a log-loss of 0.29 and accuracy of over 90%, we have succeeed in our task of author identification. Finally, we also deployed the model as an API using flask and keras which is fast and can process a large amount of text.

In our work, we explored one pre-trained embedding - GloVe. In future, we can attempt to improve the model by using other embeddings such as Word2vec to explore transfer learning further. Further, since there was a limitation as explained in the Section in augmenting data to more than 500 texts, we can explore back-translation further. Due to the usage of an external API and limited number of requests, we could only use 500 samples of our dataset to back-translate. In future, we may try to find better ways to do so, this may further improve the training and performance of our methods.

We would leave the reader with the following beautiful quote from the beloved author of Frankenstein -

"Invention, it must be humbly admitted, does not consist in creating out of void, but out of chaos", Frankenstein, 1831, Mary Shelley. [16].

10. References

[1] Spooky author identification.

[2] Douglas Bagnall. Author identification using multi-headed recurrent neural networks. *arXiv preprint arXiv:1506.04891*, 2015.

[3] Sung-Hyuk Cha and Sargur N Srihari. Writer identification: statistical analysis and dichotomizer. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 123–132. Springer, 2000.

[4] Carole E Chaski. Who wrote it? steps toward a science of authorship identification. *National Institute of Justice Journal*, 233(233):15–22, 1997.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[6] François Chollet et al. Keras. `https://keras.io`, 2015.

[7] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*, 2018.

[8] Hankyu Jang, Sunwoo Kim, and Tony Lam. Kaggle competitions: author identification and statoil/c-core iceberg classifier challenge. *Dept. School Inform., Comput., Eng. Indiana Univ., Bloomington, IN, USA, Tech. Rep*, 2017.

[9] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, 2008.

[10] Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 364–371. IEEE, 2017.

[11] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124. ACM, 2017.

[12] Edward Loper and Steven Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[13] Ahmed M Mohsen, Nagwa M El-Makky, and Nagia Ghanem. Author identification using deep learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 898–903. IEEE, 2016.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[16] Maria Popova. 'frankenstein' author mary shelley on creativity, Jun 2018.

[17] Adrian Rosebrock. The keras blog.

[18] Nils Schaetti. Unine at clef 2017: Tf-idf and deep-learning for author profiling. In *CLEF (Working Notes)*, 2017.

[19] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*, 2015.

[20] Lawrence M Solan and Peter M Tiersma. Author identification in american courts. *Applied linguistics*, 25(4):448–465, 2004.

[21] Efstathios Stamatatos, Walter Daelemans, Ben Verhoeven, Martin Potthast, Benno Stein, Patrick Juola, Miguel A Sanchez-Perez, and Alberto Barrón-Cedeño. Overview of the author identification task at pan 2014. In *CLEF 2014 Evaluation Labs and Workshop Working Notes Papers, Sheffield, UK, 2014*, pages 1–21, 2014.

[22] Gen Tsuchiyama. Quantitative research into narrative: Statistical analysis of.

[23] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019.

[24] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.

[25] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[26] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.