Deep Learning
Spooky Author Identification

Mohammad Poul Doust (MLDM) Eduardo Brandao (MLDM)

Contents

1 Introduction

The aim of this report is to explore the problem of Authorship identification from text. For that purpose, we will be using Kaggle competition dataset[1] for Spooky Authors Identification. The dataset contains short passages from horror stories belonging to three different authors. Specifically, Edgar Allan Poe, Mary Shelley, and H.P. Lovecraft. Data processing and deep learning techniques were used to solve this problem.

The main idea behind our approach is that *literary* author identification is fundamentally different from other author identification problems, such as tweets', or posts'. We therefore decided to use an ensemble of deep methods, each individually capturing a certain aspect of literary text. The ensemble strategy led to considerable improvement in the Kaggle ranking compared to the best individual strategy (from about 800^{th} to about 280^{th} place on the ranking.

In the first model, we used standard pre-processing before building the TF-IDF representation of the data. The author classification task is done by a multi-layered perceptron (MLP). Since this a bag-of-words approach, we expect it to classify according to the *vocabulary* (and frequency) specific to each author.

The second model vectorizes text by passing it through a pre-trained historical embedding layer (trained on English corpora up the year 1900), before passing it on to an MLP classifier. We expect it to classify based on the *meaning* of the terms used by the authors.

The third model uses FastText, a model that considers each word as a bag of character n-grams, and with which we hoped to capture author-specific morphological information.

The fourth model is LSTM, with which we intended to capture the use of context to define term meaning for each different author.

Pre-processing, syntax, and semantics

Traditionally, we separate text analysis into syntactic and semantic components. The semantic aspects of a text roughly match the *meaning* of the text, whereas syntax is a matter of form.

Removing punctuation and stop words is a standard pre-processing step in text classification tasks, in spite of their acknowledged semantic role of facilitating "*relational links between text units in text sentences*"[1]. In most such tasks, the advantages of the drastic dimensionality reduction after this pre-processing step are seen as compensating the loss of their semantic weight.

But this is not necessarily the case. In fact, relatively recent research [2] seems to show that removing stop words in a non-dynamical way can actually hinder classification tasks. Interestingly, one of the authors in this classification task agrees with the assessment that punctuation serves a semantic role. In the words of Poe[3]:

> "That punctuation is important all agree; but how few comprehend the extent of its importance! The writer who neglects punctuation, or mix-punctuates, is liable to be misunderstood — this, according to the popular idea, is the sum of the evils arising from heedlessness or ignorance. It does not seem to be known that, even where the sense is perfectly clear, a sentence may be deprived of half its force — its spirit — its point — by improper punctuation. For the want of merely a comma, it often occurs that an axiom appears a paradox, or that a sarcasm is converted into a sermonoid."

Removing stop words dynamically being impractical given the time constraints of this assignment, we relied on open-sourced, static stop word lists. In spite of their convenience and widespread use – scikit-learn's[4] is a notable example – these lists have well-known inconsistencies, even with tokenizers in the same toolset[5]. In addition, a major limitation of these stop words lists is "*their inapplicability to new domains and languages*" [5].

[1] https://www.kaggle.com/c/spooky-author-identification/

In conclusion, removing stop words and punctuation is helpful because it reduces noise and the dimensionality of the problem. But in the case of literary authors' classification, at least, there is reason to believe that in discarding them we may loose precious information.

Word embeddings, syntax, and semantics

Word embeddings are a major tool to capture semantic meaning in text. The major limitations here are the size of the corpus – a small corpus will be unable to capture the relationships between tokens –, and the actual content of the corpus. The first limitation can be solved by using a pre-trained model, such as Glove's [6] or Word2Vec's[7]. In spite of their usefulness, there is a major limitation when seen as tools to capture text semantics: they were trained on *modern* corpora, and whatever meaning they may extract is relative to these. It is well-known that this meaning evolves in time, and that the magnitude of this change is inversely proportional to word frequency (known as "*the law of conformity*"[8].

This suggests yet another problem in removing stop words designed for non-modern text classification: old stop words, being frequent, will tend to remain stop words, due to the law of conformity, but the law of conformity following the arrow of time, new stop words might not have been considered as such in the times of Poe, Shelley, and Lovecraft: take the word "system", in scikit-learn's stop word list, as an example.

Literary authors, syntax, and semantics And this leads us to the literary author identification task: literary authors are called literary because they use language as *tool* – often in unorthodox ways.

- Literary text often involves rendering of characters' discourse, with character-specific syntax and character-specific semantics. Who is the author of the in that case? The original author or the character?
- We hypothesize that literary text tends to use words in context-sensitive ways more than common text.
- Literary authors often use neologisms, or different morphological variants of the same word ("mammy" and "mommy" can very well be in the same sentence.

A coordinated approach For the reasons that we described above, we opted for a coordinated approach:

Difficulty	**Approach**
Removing stopwords and punctuation reduces semantic information but reduces noise	Combine independent models with and without stopword and punctuation preprocessing
Pretrained embeddings capture semantics best, but are often trained on modern text, and semantics evolves in time	Combine models that use pretrained historical embeddings, and others that resort to training on the problem corpus alone.
Literary authors use language in a specific way	Combine models that can capture subword information, and models that take into account parts-of-speech and other syntactic information
There is no single model that can resolve all these different axes at the same time	use an ensemble of methods, while trying to keep them, in some sense, orthogonal.

This paper is organized as follows: we begin by briefly describing the data. We proceed by presenting each of the individual approaches, their advantages and shortcomings. We combine all these approaches into an ensemble strategy, which in an attempt to take advantage of their individual merits. We conclude by discussing the results, and propose possible improvements.

2 Data Description

The dataset is formatted as a Comma-separated Values (CSV) and it is split into two sets:

- Training set (With labels) : 19579 rows
- Test set (No labels): 8392 rows

The structure contains three columns: id, text and author. The author columns values are:

- EAP: Edgar Allan Poe
- MWS: Mary Shelley
- HPL: HP Lovecraft

Table 1 shows a sample of the dataset structure. It is worth noticing that the dataset is not perfectly

id	text	author
id16607	Here we barricaded ourselves, and, for the present were secure.	EAP
id22605	To be near him, to be loved by him, to feel him again her own, was the limit of her desires.	MWS
id17569	It never once occurred to me that the fumbling might be a mere mistake.	HPL

Table 1: Dataset Sample

balanced as depicted in Figure 1. There considerably more rows for EAP than the other two authors.

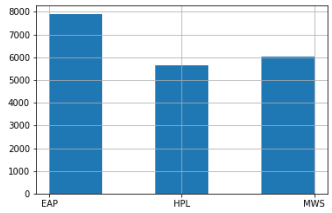

Figure 1: Authors Histogram

3 Method: Simple Dense with Tf-idf features

3.1 Overview

The goal of this method is to check the baseline in text classification (tf-idf) features in authorship attribution setting. Basically, using tf-idf will distinguish authors based on the vocabulary used. no semantic and no syntax included. Some authors more likely to use specific words more than authors.

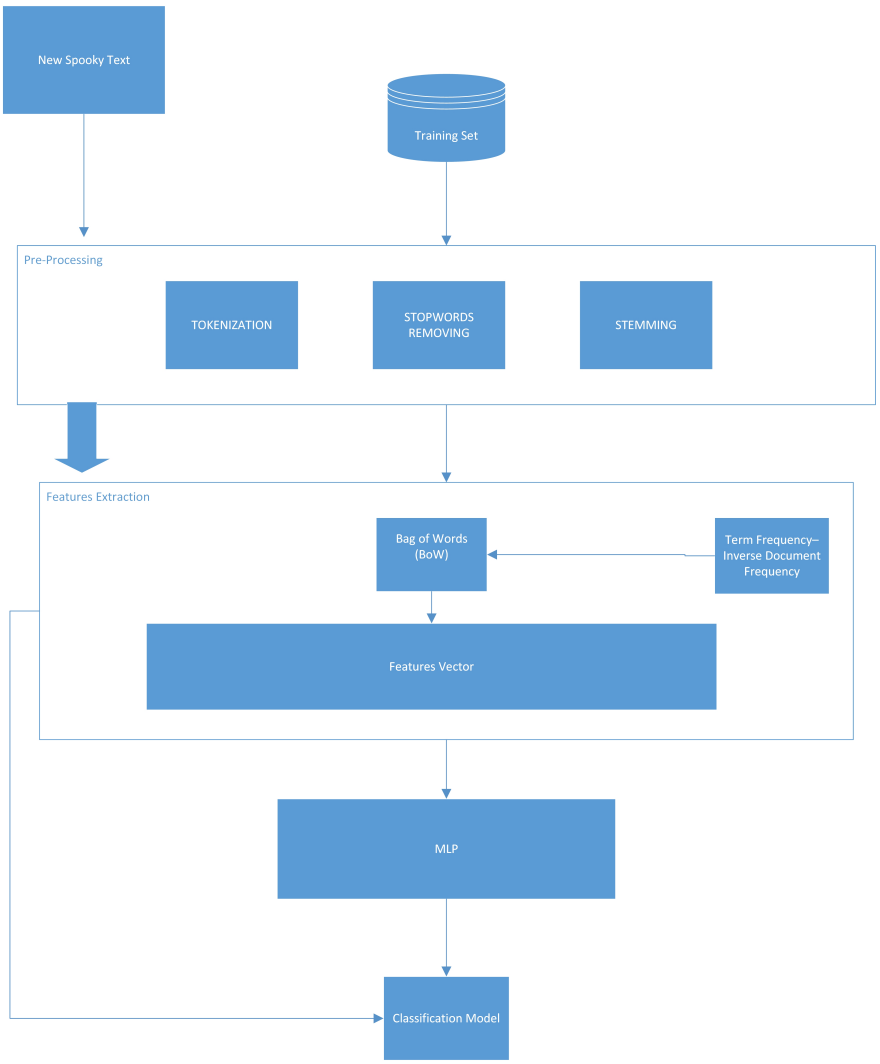

Figure 2: Simple MLP with TF-IDF

3.2 Data Processing

Basic data preprocessing has been applied for this method. Specifically, Tokenization, Stop Words Removal and Stemming.

- Tokenization:
  It is the task of segmenting text sequence into smaller units called tokens. Perhaps at the same time throwing away certain characters. Each token may represent a word, number or a punctuation mark

- Stop Words Removal:
  It is the task of dropping the words that are likely insignificant in terms of meaning and information. Text may contain stop words like 'the', 'is', 'are'. There is no universal list of stop words. We used the custom stop words for Arabic language that was developed in [9].

- Stemming:
  In linguistic morphology, stemming is the process of reducing words to their word stem, Stemmers in general are language-specific because designing a stemmer for a specific language require a deep knowledge of this language. For this method, we used a stemmer based on Porter Algorithm[10].

3.3 Model

After performing the preprocessing, the feature vector is built using Bag of Words from the TF-IDF representation of the data. Finally, the features vector is fed to our simple Multi-Layer perceptron described in Figure 7. The exact details for the architecture used is listed in Table 2

Total Parameters	Loss	Optimizer
10,043	categorical_crossentropy	Adam

Table 2: Model 1 specifications

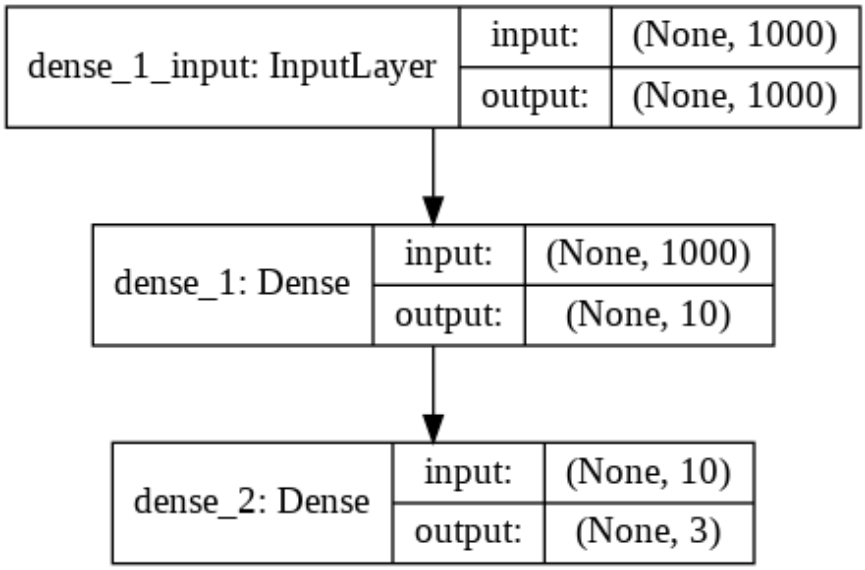

Figure 3: MLP Architecture

4 Historical Embedding with MLP:

4.1 Overview

In this model, the preprocessing stage is not removing the stop words nor the punctuation since they are meaningful to capture the author writing style. Ideally, authors use stop words and punctuation differently. Moreover, make magnify the effect of punctuation, we treated them as separate words (tokens). For the model itself, we used a simple neural network with an Embedding Layer. HistWords embedding. [2] from Stanford. The main motivation for this choice that the authors in the dataset dates back to different time span:

- HP Lovecraft: 1890 to 1937
- Edgar Allan Poe: 1809 to 1849
- Mary Shelley: 1797 to 1851

Knowing that words shift in meaning by time and hence, same word would have different usage/meaning in different time. Moreover, some new words are introduced by time and old words are discarded [8]. That would probably be a problem for other embeddings that was trained on newer corpus.

[2] https://nlp.stanford.edu/projects/histwords/

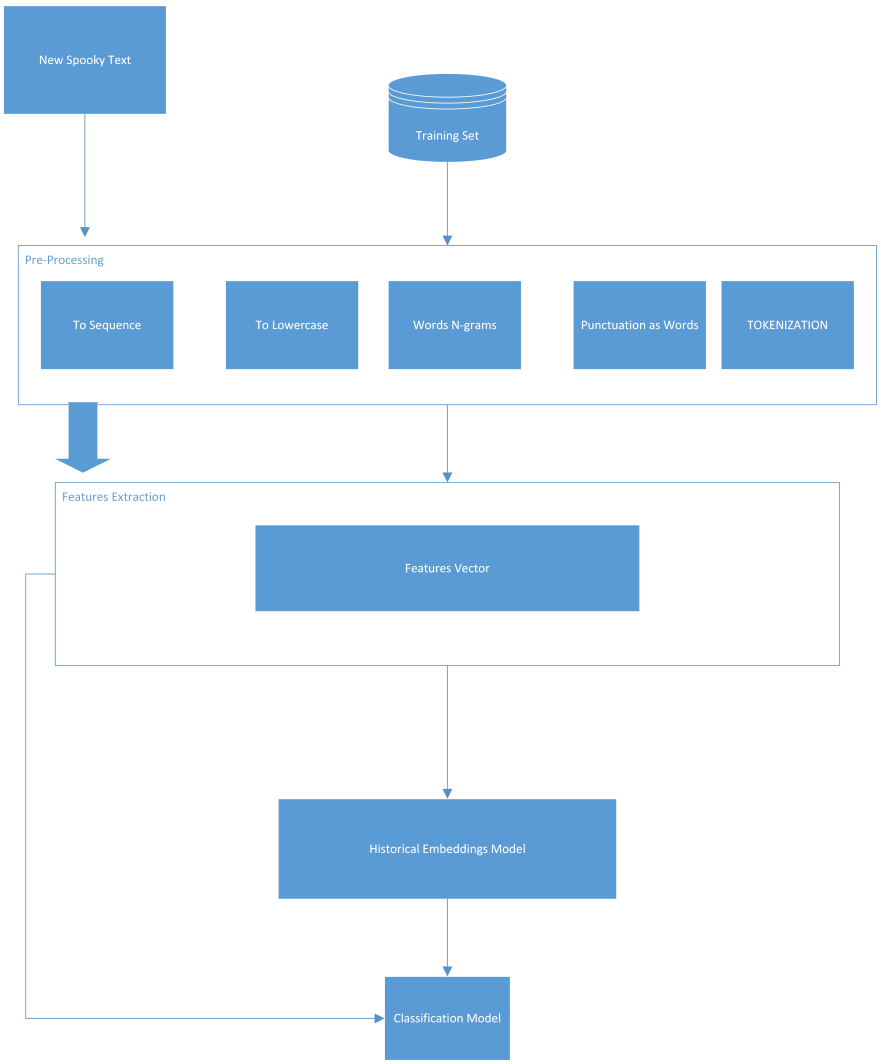

Figure 4: Historical Embedding Model Flow

4.2 Data Processing

- Punctuation To Words:
  As punctuation usage differs from author to author, it was important to magnify their effect. Therefore, a simple preprocessing was done to treat punctuation as a separate token.
- Lowercase:
  Convert all characters into lowercase form.
- To Sequence:
  Convert text into sequence of integers.
- Words N-Grams
- Tokenization

4.3 Model

The model takes as an input the vectorized text passing through an Embedding Layer (300 dimensions). This layer is initialized by the historical embedding weights (using the year 1900). Afterwards, the resulting

output is averaged using Global Average layer and finally classified using a simple Dense layer with three outputs

Total Parameters	Loss	Optimizer
21,021,003	categorical_crossentropy	Adam

Table 3: Historical Embeddings Model Specifications

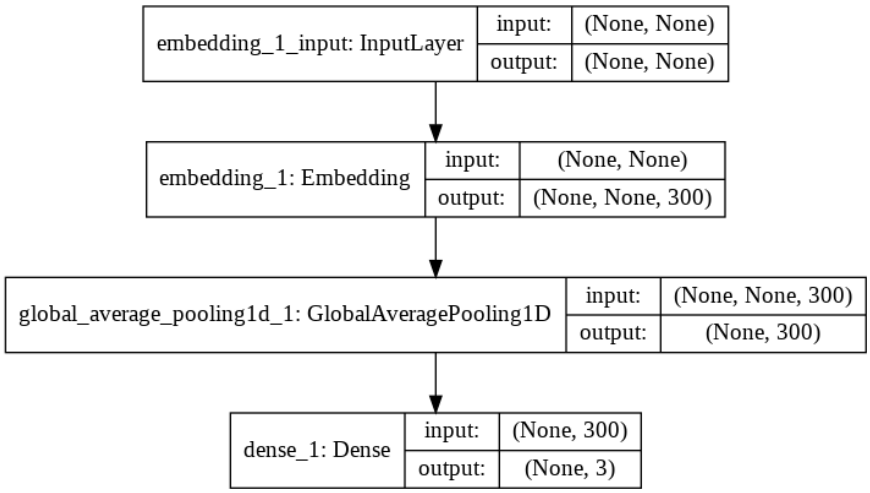

Figure 5: Historical Embedding Model Architecture

5 Functional Model (Embedding with hand-crafted features)

5.1 Overview

The main motivation for this model is based on the idea that to captures the style of each author, we need to employ different levels of indicators. Those indicators some of them could be extracted automatically using simple bag of words (BoW) and by learning an embedding representation.

Moreover, it is important to employ some hand-crafted features that we believe to give insight in identifying each author. For example, some authors tend to use punctuation more often than others, some authors have unique syntax in writing(use nouns more likely).

Similarly, The number of unique words used by author could be a feature for this author. A set of such features have been extracted (Table 4), in addition to the other previously mentioned features.

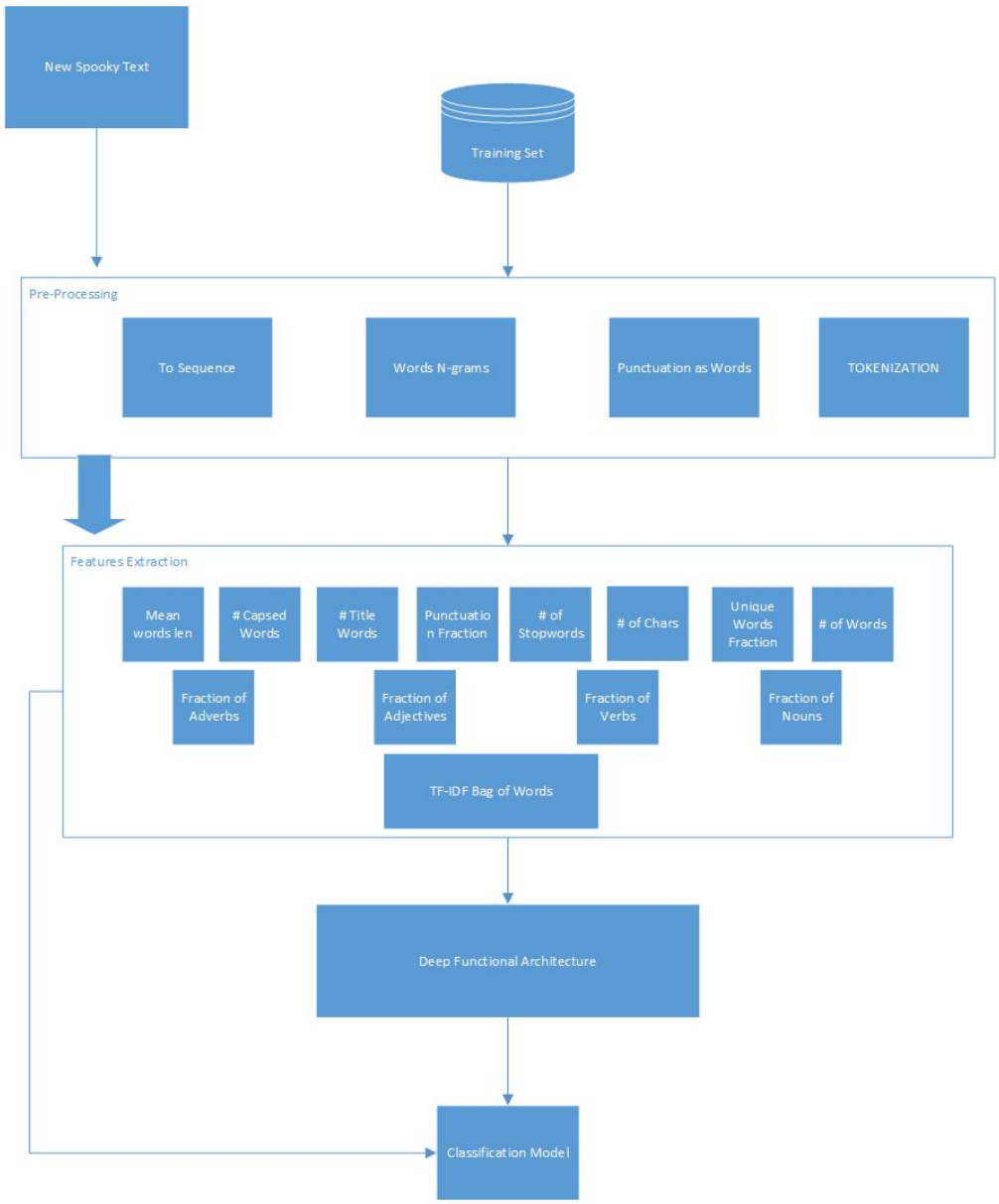

Figure 6: Functional Model Flow

5.2 Data Processing

Similar to the previous method, the same data processing has been applied. However, the letter cases are preserved to be used later as features.

- Punctuation To Words
- To Sequence:
- Words N-Grams
- Tokenization

Feature	Description
num_words	# of words in each text
unique_word_fraction	Fraction of unique words to the number of words in the text
num_chars	# of characters
num_stopwords	# of stop words
punctuations_fraction	Fraction of punctuation over total number of characters
num_words_upper	# of Uppercase words
num_words_title	# of words that beings with an uppercase
mean_word_len	Average length of the words in the text
fraction_noun	Fraction of nouns over total words
fraction_adj	Fraction of adjectives over total words
fraction_verbs	Fraction of verbs over total words
fraction_adverbs	Fraction of adverbs over total words

Table 4: Handcrafted features

5.3 Model

This model ensemble three types of features in one non-sequential model. It has three inputs, each is responsible for handling one type of features. Afterwards, those three branches are merged using concatenation operation and passed to a simple softmax layer for classification. Specifically, first branch (output: 300 dimensional vector for each instance) captures the semantic of words using the Historical Embedding, while the second branch (output 10 dimensional vector) captures the vocabulary using tf-idf features. Finally, the last branch captures information extraction from the hand-crafted features. These branches are extracting different levels of features to make final features vector of size 320 for final classification.

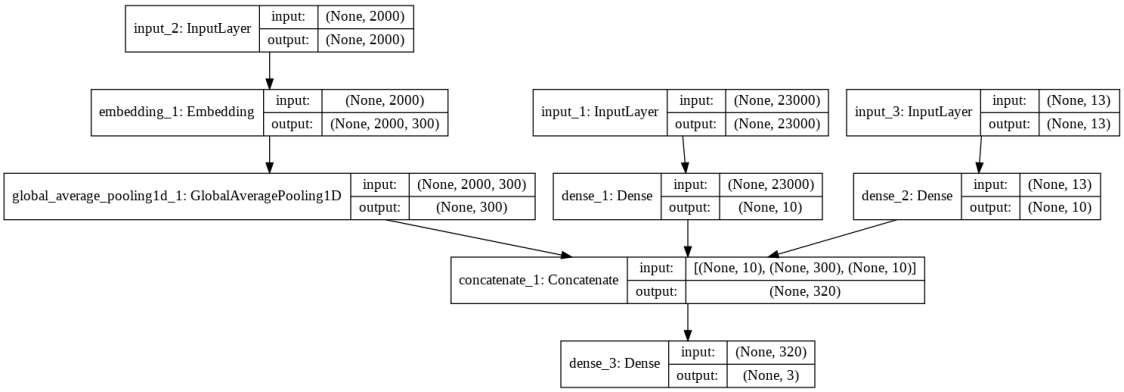

Figure 7: Historical Embedding Model Architecture

Total Parameters	Loss	Optimizer
831,113	categorical_crossentropy	Adam

Table 5: Historical Embeddings Functional Model Specifications

6 Fast Text Embeddings Model:

6.1 Overview

The main motivation for this model is to capture subword information. Models that learn word representations ignore the *morphology* of the words, by by assigning a different vector to each word. Since we expect the authors in this classification task to use specific, *literary*, vocabulary (which might not even exist

elsewhere), we sought a model that would incorporate subword information. We chose Fast Text[11] for its speed, simplicity (motivated by interpretability) and documented efficiency [12], but mainly for its ability to represent words that are not in the training data (provided one of its subword n-grams is).

6.2 Data Processing

We contemplated three different preprocessing steps for this method: with punctuation and stopwords (0.57 loss), without punctuation and stopwords (0.82 loss) and without punctuation but with stopwords (0.62 loss), finally settling on the model that kept the most information in the text: with punctuation and stopwords.

Even though we expected a decrease in performance from using such a model compared to one where a dimensionality reduction technique was used, we intend to use it in an *ensemble*, where we expect it to focus on information that other models will not be able to pick up.

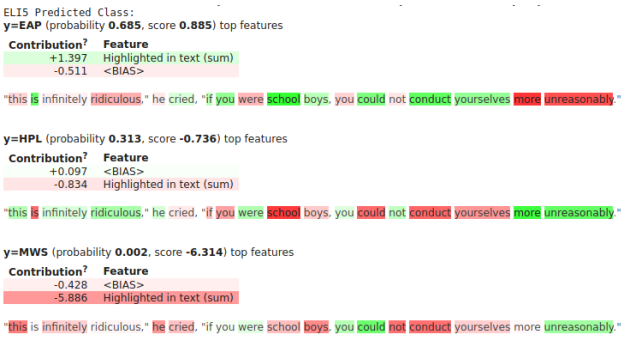

Figure 8: True Class is MWS. We see that Fast Text makes its decision based on many different words. We note that even though the predicted class is EAP, the text is a rendering of character speech – which is arguably more specific to the character than to MWS.

Punctuation Considering punctuation as words has a few challenges with respect to text processing; the symbol for period, for example, can be used to mark the end of a sentence, an abbreviation, or as a formatting symbol (as in a restaurant menu, or a list, and a time). These correspond to different semantic roles, as can be clearly seen in 9, and which we had to account for. The details of this process can be found in the Jupyter notebooks that accompany this report.

Stop words We removed stopwords using the nltk list [13].

6.3 Model

Fast text is a (one layer) deep model [11]. As can be seen in figure 10,

Figure 9: PCA=2 representation of the Fast Text embedding, with the 1000 terms closest to colon (GREEN), semicolon (RED) and period (BLUE), for each author: MWS, EAP, and HPL, from top to bottom . Circle radius is proportional to the term frequency. We see that EAP is the most eclectic when it comes to the use of punctuation in its semantic role (it takes full use of both dimensions in this representation; that HPL has very specific semantic usage for these punctuation marks, denoted by the neat clustering; and that MWS uses commas ans semicolons almost interchangeably. In any case, it is clear that, for each author, there is a different semantic role for punctuation.

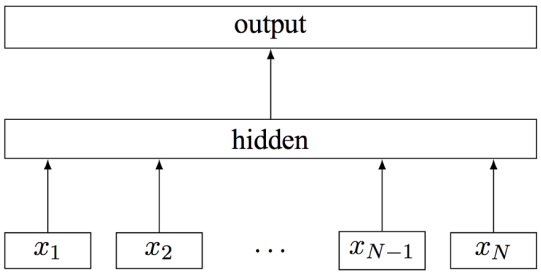

Figure 10: Fast Text Model: model architecture for a sentence with N ngram features, $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

The bag-of-words representation of the text is first fed into a lookup layer, where the word representations are created for word (essentially, the word itself and its character ngrams. The word embeddings are then averaged and fed into a hidden layer. The averaged vector is then fed into a *linear* classifier: where the probability distribution of each class is computed by the softmax function, leading to minimizing the log-likelihood over the classes [12]. We used the automatic parameter tuning provided in the package; the model was trained over 50 epochs, using a learning rate of 0.1 and using 5-grams subword information. The "OneVsAll" loss was used, which corresponds to the sum of binary cross-entropy computed independently for each label.

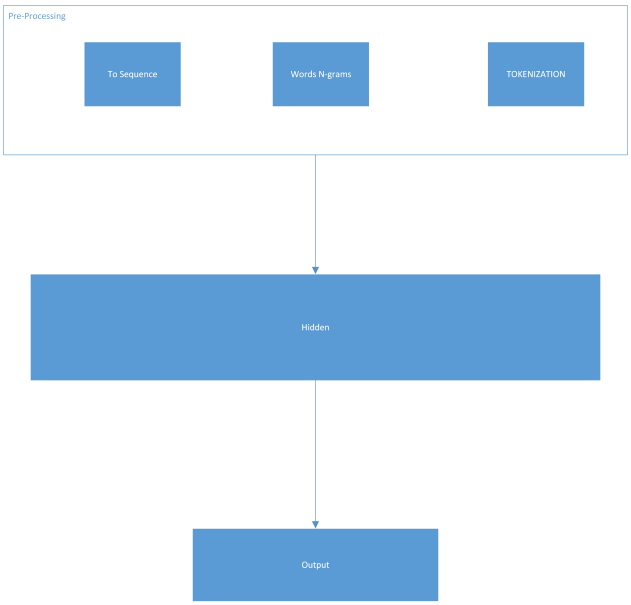

Figure 11: Fast text Architecture

7 LSTM Based Model:

7.1 Overview

We chose a simple LSTM model in an attempt to capture the context-sensitive semantics of terms in literary text [14]. We chose a simple LSTM implementation in an attempt to keep orthogonality with respect to other methods.

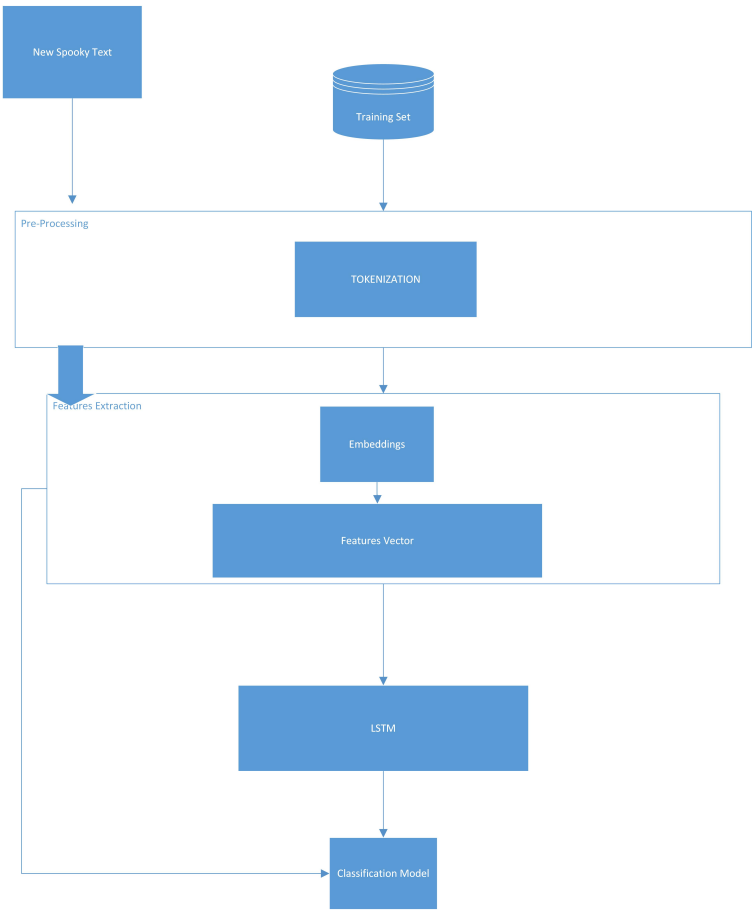

Figure 12: Simple LSTM Flow

7.2 Data Processing

We simply use the nltk tokenizer and pass the tokenized text to a keras embedding layer. This decision was determined by the lackluster scores of the alternatives, as well as by using LIME[15].

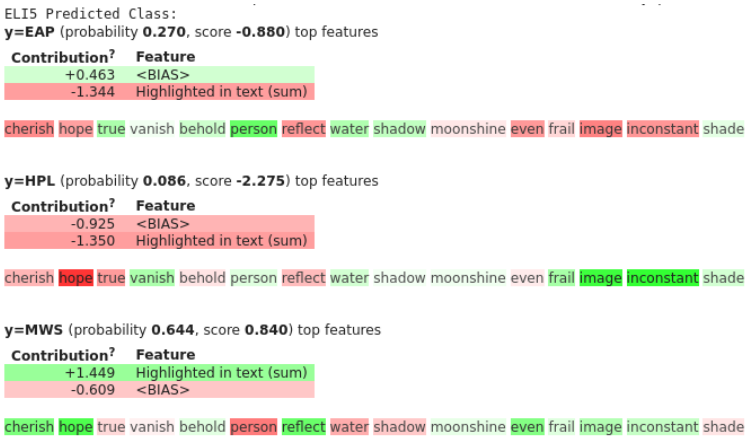

Figure 13: True class is MWS. We see that the decisions are being made based on words such as "person" and "appeared", which don't seem to carry any author specific meaning. We see this across the board, with all pretrained embeddings, for LSTM approaches. We get the best results with the simplest method.

7.3 Model

We trained the model on the training set with categorical cross-entropy loss, and the 'adam' optimizer, over 3 epochs, with a batch size of 16. We tried a variety of architectures, with and without pretrained embeddings (also bidirectional) with results in terms of loss the $0.7 - 1.2$ range. We finally settled on the simplest architecture, which gave gave the best result (0.47 loss).

8 One Final Ensemble Model:

8.1 Overview

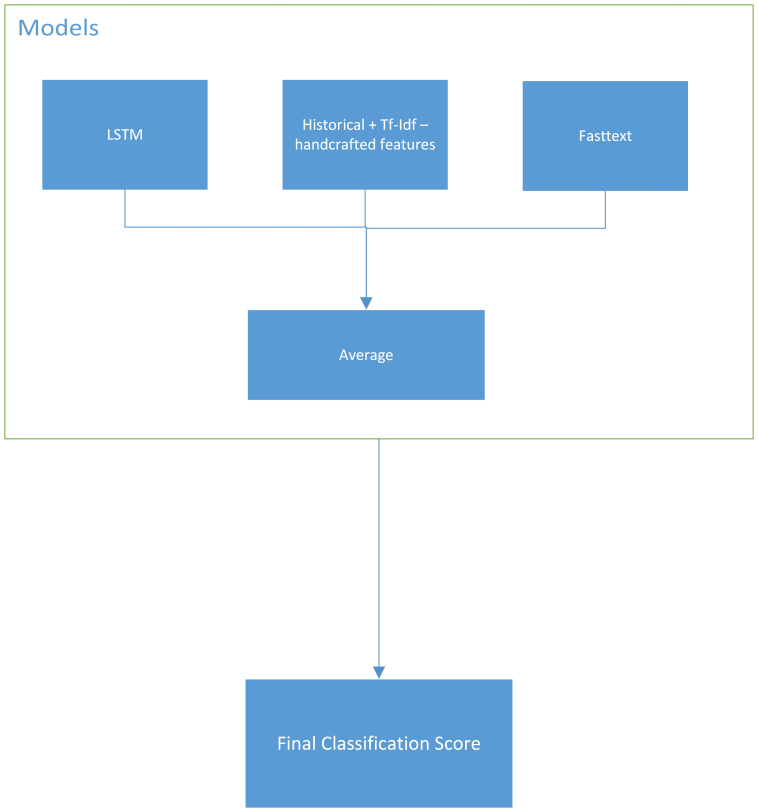

Figure 14: Ensemble Model

8.2 Model

We considered a number of alternatives for the ensemble:

1. normalized average classifier probabilities
2. random convex combinations of classifier probabilities
3. normalized linear combination of classifier probability, weighted by inverse test loss
4. normalized maximally confident (per text) classifier for each class
5. normalized linear combination of classifier probability, weighted by confidence (all corpus)
6. normalized linear combination of classifier probability, weighted by confidence (per text)
7. normalized linear combination of classifier probability, weighted by precision (all corpus)

In the end, we achieved best results for the test loss for the simplest model, which merely predicts the normalized average of the individual classifiers' predictions.

id	EAP	HPL	MWS
id02310	0.003974	0.003179	0.992848
id24541	0.996628	0.002907	0.000465
id00134	0.000395	0.99955	0.002649

Table 6: Submission file format

9 Evaluation and Results

The methods were evaluated by submitting the results (to Kaggle) as comma-separated file (see Table 6) Evaluation calculated using multi-class logarithmic loss:

$$\text{logloss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log\left(p_{ij}\right)$$

N: Test set size
M: of classes (3 classes)

The final submission results for each method depicted in Table 7. The "Ensemble of different methods" achieved the best with log loss of 0.33 which could be justified given the nature of log loss that penalize the more the probability diverges from the actual label. Hence, having an ensemble of methods would increase the confidence of probability and decrease the log loss.

Method	Result (Loss)
Simple Tf-idf Model	0.45
Historical Embedding MLP	0.36
TF-idf + Historical Embedding + handcrafted features	0.35
Fast Text Embedding	0.57
LSTM	0.47
Ensemble of different methods	0.33

Table 7: Results of different approaches

10 Conclusion

In this report, we chose to tackle the problem of author identification using exclusively deep learning tools. We strove to make our models interpretable, by using LIME [15] where possible, or by analysing the relationship between landmarks in the word embedding vector space. This exploration led us to iteratively adapt our approach, which we finally combined into an ensemble, combining different characteristics of literary authors' text. We believe that this approach could be further refined:

The functional model, which is essentially a syntactic model, could be refined by adding part-of-speech, sentiment analysis, and gender, for example, to the list of features – in line with our analysis of the particularities of literary text.

The historical embeddings model seems to be doing an overall good job at detecting meaning, and it is not obvious to us how it could be improved, other than by parameter tuning.

The Fast Text model seems to be doing a good job at detecting subword information. It validates in particular our assessment that, in the case of literary authors, punctuation and stopwords have a semantic

role. It seems to suffer from the small size of the corpus, however. One possibility for improvement would be to use a pretrained historical model in this case as well – but that would remove the orthogonality between our approaches.

We tested the LSTM model in many forms, but the parameter space is vast. We believe that the best avenue for improvement would be to draw inspiration from previous attempts and the literature.

References

[1] Ted Briscoe. The syntax and semantics of punctuation and its use in interpretation. In *Proceedings of the Association for Computational Linguistics Workshop on Punctuation*, pages 1–7, 1996.

[2] Hassan Saif, Miriam Fernández, Yulan He, and Harith Alani. On stopwords, filtering and data sparsity for sentiment analysis of twitter. 2014.

[3] Edgar Allan Poe. Poe, e. (1848). marginalia [part xi]. graham's magazine, (vol. xxxii, no. 2), pp.130-131. *Graham's Magazine*, XXXII(2):130–131, Feb 1848.

[4] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[5] Joel Nothman, Hanmin Qin, and Roman Yurchak. Stop word lists in free open-source software packages. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 7–12, 2018.

[6] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[8] William L Hamilton, Jure Leskovec, and Dan Jurafsky. Diachronic word embeddings reveal statistical laws of semantic change. *arXiv preprint arXiv:1605.09096*, 2016.

[9] Ibrahim Abu El-Khair. Effects of stop words elimination for arabic information retrieval: a comparative study. *International Journal of Computing & Information Sciences*, 4(3):119–133, 2006.

[10] Martin F Porter. An algorithm for suffix stripping. *program*, 14(3):130–137, 1980.

[11] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[12] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[13] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[14] Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

[15] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.