Spooky Author Identification

Chintan Shah
Shah.c@husky.neu.edu

Introduction:

In this year's Halloween playground Kaggle competition, you're challenged to predict the author of excerpts from horror stories by Edgar Allan Poe, Mary Shelley, and HP Lovecraft. Pages from unbound spooky books are scattered across the chamber floor. The task is to figure out how to put them back together according to the authors who wrote them.

DataSet:

Dataset is provided by the Kaggle @ https://www.kaggle.com/c/spooky-author-identification/data
It consists of 2 set of data as below :

1. Train.csv : This set contains 19579 rows & 3 columns which are : id, text & authors related to the text.

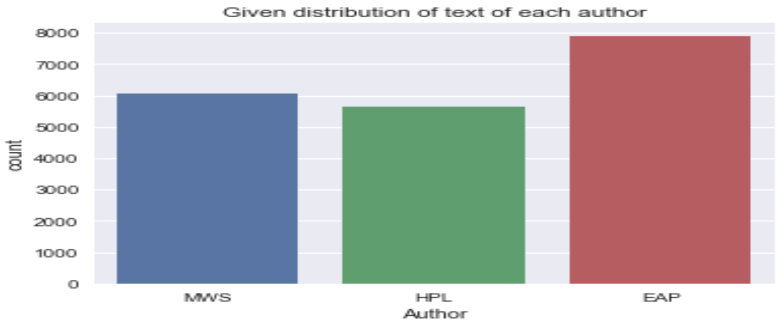
id	text	author
id00001	Idris was well content with this resolve of mine.	MWS
id00002	I was faint, even fainter than the hateful mod...	HPL
id00003	Above all, I burn to know the incidents of you...	EAP
id00004	He might see, perhaps, one or two points with ...	EAP
id00005	All obeyed the Lord Protector of dying England...	MWS

2. Test.csv : This set contains 8392 & 2 columns which are : id, text.
   The task is to find the authors of these text.

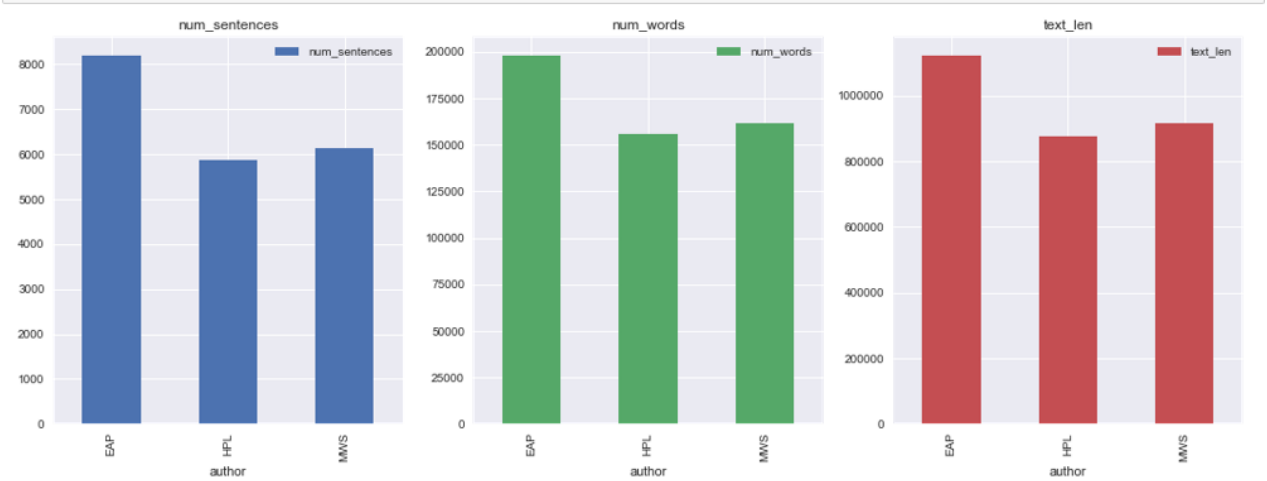
id	Text
id02310	Still, as I urged our leaving Ireland with suc...
id24541	If a fire wanted fanning, it could readily be ...
id00134	And when they had broken down the frail door t...
id27757	While I was thinking how I should possibly man...
id04081	I am not sure to what limit his knowledge may ...

Exploratory data analysis:

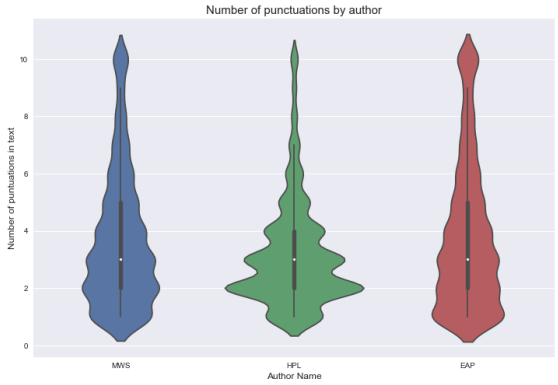
1. Distribution of text on basis of Authors : We have maximum text from EAP



2. Features of text used by each authors :



3. Punctuation Used by the authors :



4. Gram tables of each Author :
   a. EAP :

	1-Gram	Occurence	2-Gram	Occurence	3-Gram	Occurence	4-Gram	Occurence
0	upon	1025	let u	50	madame l espanaye	12	general john b c	12
1	one	671	three four	23	general john b	12	brigadier general john b	7
2	could	453	l etoile	23	john b c	12	ugh ugh ugh ugh	6
3	would	409	one two	23	ha ha ha	11	brevet brigadier general john	6
4	said	356	every thing	22	barrière du roule	10	john b c smith	6

   b. HPL :

	1-Gram	Occurence	2-Gram	Occurence	3-Gram	Occurence	4-Gram	Occurence
0	one	516	old man	59	heh heh heh	9	eric moreland clapham lee	4
1	could	480	could see	31	terrible old man	9	oonai city lute dancing	4
2	thing	433	one night	23	charles le sorcier	8	aira city marble beryl	4
3	old	392	old woman	22	small paned window	8	mad arab abdul alhazred	4
4	would	357	one might	19	great great great	6	necronomicon mad arab abdul	4

   c. MWS:

	1-Gram	Occurence	2-Gram	Occurence	3-Gram	Occurence	4-Gram	Occurence
0	one	489	old man	29	let u go	4	next day next hour	2
1	would	475	lord raymond	28	whole human race	3	nearest town took post	2
2	could	383	fellow creature	23	became every day	3	abode near hyde park	2
3	life	350	one day	21	time lord raymond	3	five year old mother	2
4	yet	318	let u	16	might one day	3	like thousand pack wolf	2

Pre-processing & Model Creation:

To predict the author, I have used the classification Algorithms and build the model to predict the probabilities of each author & have save the result in a csv file. So before creating the model, there is some pre-processing that was needed so as to fit the data into the model.

To start with the pre-processing of text I have used Natural Language Toolkit(NLTK).

1. Natural Language Toolkit :
   NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.
   Using NLTK following things are preformed :
   a. Remove Punctuation.

   b. Removing stopwords : Stop Words are words which do not contain important significance

   c. Stemming & Lemmatization : Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma

   After Process the text using the NLTK we need to convert the text in the form of sparse matrix so that it can be used by the algorithms.

2. CountVectorizer :
   Convert a collection of text documents to a matrix of token counts
   This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix.
   Countvectorizer gives out the sparse matrix which contains the frequency of each word in respective documents/Data

   But on the basis of the count of words, only the words that have appeared for most number of time is given the importance due to which we may miss out some important information that is present with low frequency words. So to balance out this, I have also used TF-IDF

3. Term Frequency–Inverse Document Frequency (TF-IDF) :

   TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

   TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

   IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

   IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

4. Appling the pre-process data on the classifier algorithms & measure the accuracy :
   Below is the table with accuracy of different algorithms :

Classification Algorithms	Accuracy using CountVectorizer	Accuracy using TF-IDF
Logistic Regression	0.789581205	0.801583248
Navie Bayes	**0.822778345**	**0.813329928**
Simple SVM	0.665219612	0.701736466
K nearest neighbors	0.618009091	0.593462717
decision-tree algorithm	0.593462717	0.593462717
Random Forest	0.665730337	0.683350358

As we see Navie Bayes has the best result with both Count Vectorizer& TF-IDF.

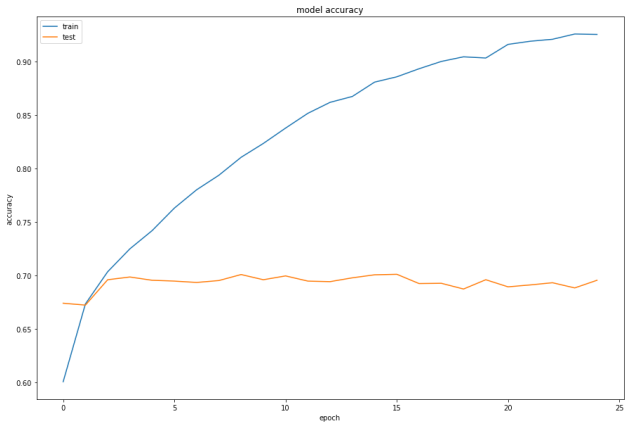
Deep Learning :

In the era of Deep Learning, I have also tried to predict the author using the Neural network concept and have used 2 models: A simple Neural Network & A Simple dense LSTM with GloVe vector representations for words.

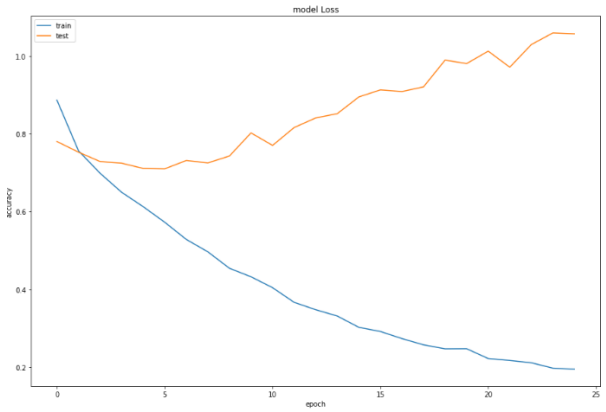
a. Simple 4 Layer Network :
I have used the simple NN layers as below :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 300)               90300
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_2 (Dense)              (None, 300)               90300
_________________________________________________________________
dropout_2 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_3 (Dense)              (None, 300)               90300
_________________________________________________________________
dropout_3 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 903
_________________________________________________________________
activation_1 (Activation)    (None, 3)                 0
=================================================================
Total params: 271,803
Trainable params: 271,803
Non-trainable params: 0
```

Using the simple NN we got the below accuracy & loss :



Model Accuracy



Model loss

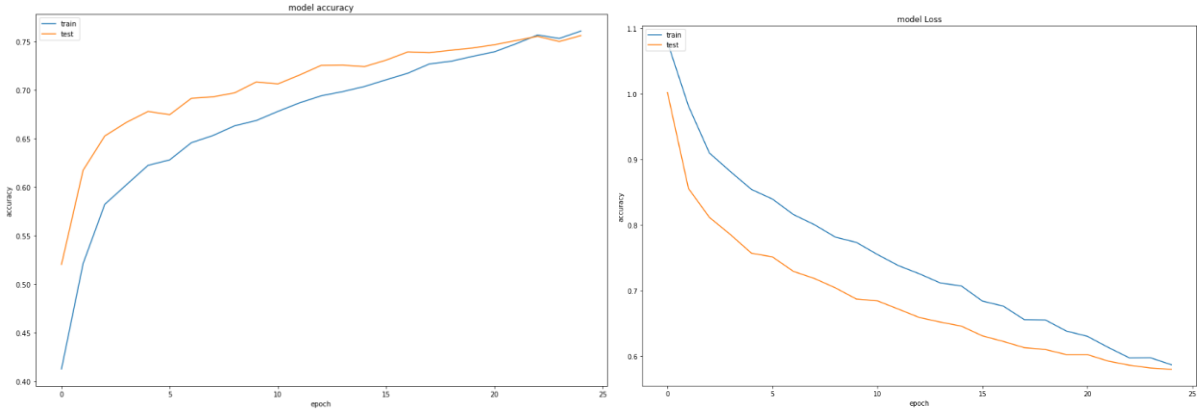
Using the simple NN model we didn't had a great result & so I decided to use the LSTM NN.

b. Long short-term memory (LSTM) :
I have used the embedding layer with LSTM layer & then fully connected layers.
Below is the model summary :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 70, 300)           7783200
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 70, 300)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               160400
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              103424
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_6 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 3075
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 9,099,699
Trainable params: 1,316,499
Non-trainable params: 7,783,200
_________________________________________________________________
```

Using the LSTM model I have got much better result than the previous NN model.

Accuracy & loss v/s Epochs :



Model Accuracy Model Loss

As we see after each epoch the accuracy increases and also the loss decreases

Conclusion:

So with Classifiers we have the best result when we used Naïve Bayes Classifier & also we see that when we used LSTM RNN model with 25epochs the accuracy increases & the loss decreases. In future, if more parameters are tuned in we could fetch much better result with Naïve bayes & LSTM models. Also in future we cn use XGBoost & GridSearch CV to produce better results.

References :

1. https://www.youtube.com/watch?v=Tp3SaRbqI4k – different keras Layers.
2. http://ruder.io/optimizing-gradient-descent/ - Gradient descent optimization algorithms
3. https://www.youtube.com/watch?v=4rG8IsKdC3U – how to implement LSTM
4. https://www.youtube.com/watch?v=WCUNPb-5EYI – A very good example which helps in understanding LSTM (example starts from 18.00)
5. https://www.kaggle.com/abhishek/approaching-almost-any-nlp-problem-on-kaggle - Approaching NLP problem.
6. https://nlp.stanford.edu/projects/glove/ - GloVe Word Vector
7. https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714 - Article helps to understand LSTM.