Detecting The Author
A study in authorship classification
Julia Stephenson – August 2018

Authorship Attribution is the process of attempting to identity the likely author based on a collection of documents whose authorship is known (source). I plan to investigate the use of machine learning techniques to identify the author of a document. The use of computational methods including machine learning for authorship attribution was discussed in the paper, A Survey of Modern Authorship Attribution Methods, Stamatatos 2009 (here)

The ability to classify documents according to their author has a number of relevant applications for example detecting plagiarism, perhaps identifying the author of unmarked work.

Problem Statement:

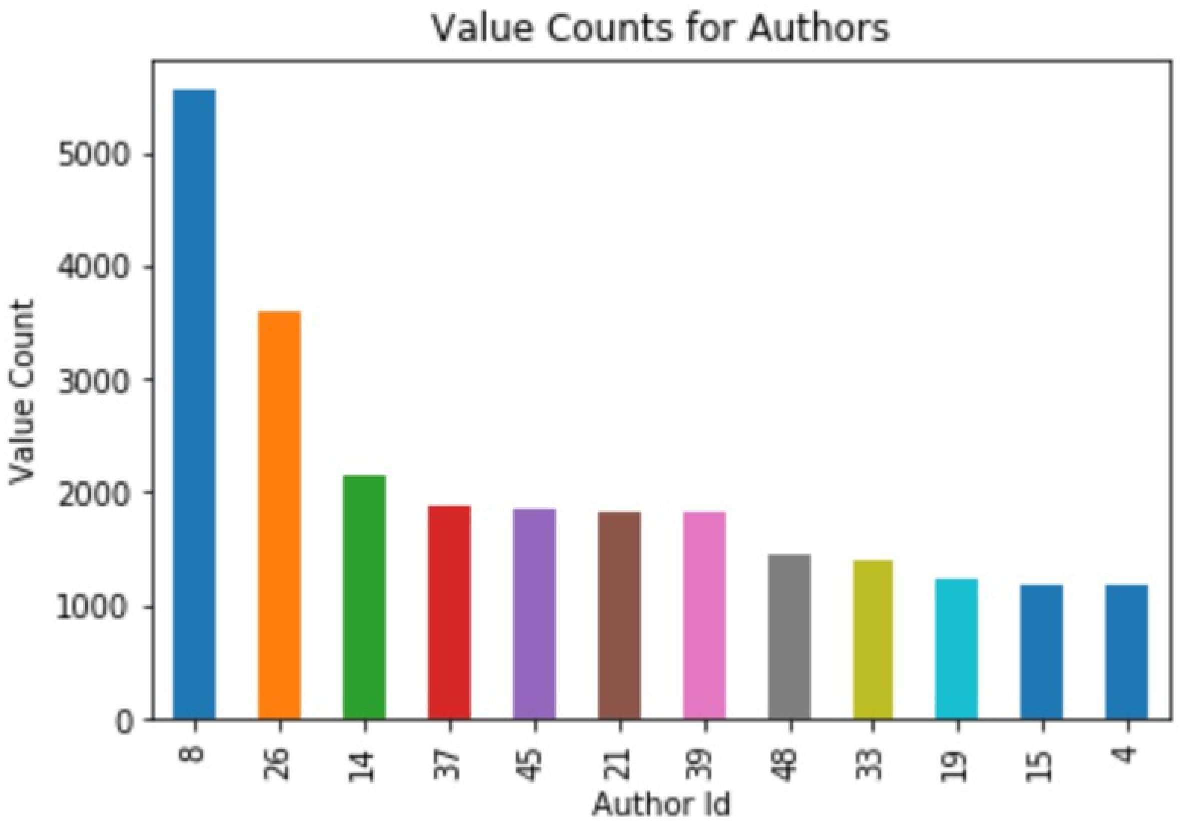
The aim is to build a model that can read in a text and suggest which author it is.
The model will learn from having learned off seeing the authors work previously. This is a supervised multi class classification problem as there are multiple authors and we have a sample of each author's work for the model to learn from. Given a new piece of work the aim is for the classifier to assign it to one of the author's known by the classifier. In this case the assumption of the classifier will be that each piece of work is assigned to one author.

The Data set:

An example record looks like this (I have included the first 1000 characters only)

'towards me and her little fingers round my hand many and many an hour i sit thus but of all those times three times come the on my mind it is morning and made so trim by my aunt s hands me how her pretty hair will curl upon the pillow yet and how long and bright it is and how she likes to have it loosely gathered in that net she wears i not that i am vain of it now you mocking boy she says when i smile but because you used to say you thought it so beautiful and because when i first began to think about you i used to peep in the glass and wonder whether you would like very much to have a lock of it oh what a foolish fellow you were when i gave you one that was on the day when you were painting the flowers i had given you and when i told you how much in love i was ah but i didn t like to tell you says then how i had cried over them because i believed you really liked me when i can run about

The Distribution of the target variable is as follows:



The plot above shows that the distribution of the data is uneven, with author 8 and to some extent author 26 being heavily represented in the dataset.

Problem

This is a multi class classification supervised learning problem.
There are multiple classes for the classifier to identify and there is data with known tags (authorship) available to analyse.

Proposed Solution:

I propose to develop a classifier which predicts the probability that the work belongs to a given author. I will be exploring a number of machine learning and deep learning models, as well as a variety of approaches to feature engineering and text preprocessing.

Summary of Proposed Approach:

1. Pre Process the text Data to transform the text to numeric form in order to feed it to a machine learning or deep leaning model.

2. Run Baseline Models to get an idea of what results to expect - Naive Bayes Multinomial and logistic regression algorithms will be used to get a baseline result.

3. Feature Engineering – I will explore approaches to extract features from text I am aiming to identify some features that may be distinct to an authors style.

4. Run Machine Learning Models over feature set to see if the results are an improvement on the baseline

5. Tune Hyper parameters and Evaluate models

6. Explore Deep Learning approaches to text classification

7. Use of Glove Vectors to add some context back to the text

8. CNN – Convolutional Neural Networks

9. Evaluate the Best Performing models – Tune Hyperparameters, and Plot learning curves to access model validity

Evaluation Metric

Log Loss

Evaluates the confidence of membership to a given class or in this case the work belonging to a particular author. For example the model calculates a probability that the work belongs to a particular author. The log loss metric will decrease as the model predicts a probability close to the true value.

For clarity the formula is as follows:

(-1/N) (sum from i = 1 to N ai) (sum from j = 1 to M aj) yij, log(pij)

N – is the number of rows in the test set (6275)

M – the number of authors (12 in this case)

pij – the probability that the work belongs to an author.

yij is the true label or author of the work

This data set has imbalanced classes in it (the first author has many more samples of work in the sample so a classifier could be accurate 21% of the time just picking the first author. Log loss gives a more detailed view as it measures the confidence of the algorithm that the work belongs to an author rather than just a prediction that the work belongs to a given author.

The goal of the classifier is to minimise log loss as log loss increases as the predicted probability of a work belonging to a specific author diverges from the actual author.

Initial Data Analysis

Text Pre processing

Text must be converted to numbers in order to be consumed by machine learning models

There are several approaches available:

1. **Bag of Words Model** -for each block of text, number of words and frequency of words is used by the model but the word order is ignored.
2. **Ngrams** – Counting Sequences of Words eg Bigrams (pairs of words found together)
3. T**erm Inverse Document Frequency** – Identifying the words that are most important in a particular document.

Term Inverse Document Frequency as selected to preprocess the text because the chunks of text are large and I was looking for a way to extract the key properties of the chunk of text. The Bag of words model would have lost too much information about the text e.g word order.

Benchmark:

Naive approach: The test set contains 6275 records, the dominant class has 1360 records. So if the classifier just predicted the dominant class all the time it would be accurate 1360 / 6275 or 21% of the time.

Benchmark model

Running Multinominal Naive Bayes having vectorized the text using to *term frequency-inverse document frequency* **the results were: log loss -2.26 which is poor given we want a value near to 0 . I**

think one potential issue with using tfidf for authorship is that it often picks out key characters or themes from the work that are specific to that work and don't generalise to another.

Analysis

Data Exploration

The dataset contains 25097 rows and 12 different authors
The dataset contains the following two features:
Author (an number)
Text (Approximately 5000 words per author)
Please refer to page 2 for an example of the text and the distribution of the target variable (author).

Exploratory Visualisation

(further visualisations will be displayed during the implementation section as these are more relevant to the problem).

Implementation

During the implementation phase a number of data pre-processing methods and feature extraction methods were explored. The implementation and results will be discussed below. The implementation was a process of refinement, examining the results of models and refining the feature set and model architectures as a result. I will present the refinement of the final models and the validation of those results at the end.

To get an initial baseline the text was pre-processed to extract the most important terms only because using a bag of words approach would have generated a very sparse input for the models.

Implementation Steps:

1.Sampling

There are many authors in the dataset whose work is not well represented. For the purpose of this investigation I have selected the records of the top 12 Authors in the dataset. This turns the problem into a 12 class classification problem. Its important to reduce the number of classes in this case if reducing the amount of samples as The consequences of sampling mean that the model will possibly not generalise to a range of authors.

2.Split Into Train and Test Set.

The dataset was then split into a training and test set using stratified sampling based on the author, because the dataset is unbalanced. I plotted the value count of authors in the training and test sets to ensure that they each contained a similar distribution of values for the authors.

Machine Learning Models

Notebook Name: [VictorianAuthorAttribution_MachineLearning.iynb]

Initial Results

Text was pre processed using Term Inverse Document Frequency

Alogrithms:

Multinominal Naive Bayes - Multinomial Naive Bayes classifier is a specific instance of a Naive Bayes classifier which uses a multinomial distribution for each of the features

XGBoost - XGBoost consists of an ensemble of decision trees

Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. (source : Machine Learning Mastery)

Logistic Regression - Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic/sigmoid function

XGBoost achieved a log loss of -2.25 which is poor, ideally the log loss should be close to 0. Looking at the most important features produced by Term Inverse document frequency it seems that it picks out characters or places in the text, i'm not sure that these would be transferable between authors works. I wanted to investigate whether it was possible to extract elements of the authors style from the text.

Examples of features found by TFIDF include
```
amateur night', 'aunt mary', 'bending bough', 'benefit doubt', 'big fat',
'black jack', 'blind girl', 'break day', 'brother francis', 'brother ned',
'chief butler', 'class struggle'
```
These look more reflective of subjects and characters in the works.

Part of Speech feature engineering:

One of the drawbacks of term inverse document frequency was that it didn't extract any information relating to the style of the author. It mostly extracted key words that seemed to be reflecting more characters or the subject of the works – which may not be relevant in identifying other works by the same author.

I performed the feature engineering in one notebook and saved a new dataset rather than creating a pipeline in this instance because the process was fairly long running and I was able to reuse the output in a number of models.

The implementation of POS feature extraction can be found in:

Part of Speech Feature Engineering Notebook: [Victorian Author Attribution NLP Preprocessing for Part of Speech Tags.iymb]

Feature Engineering Notebook:

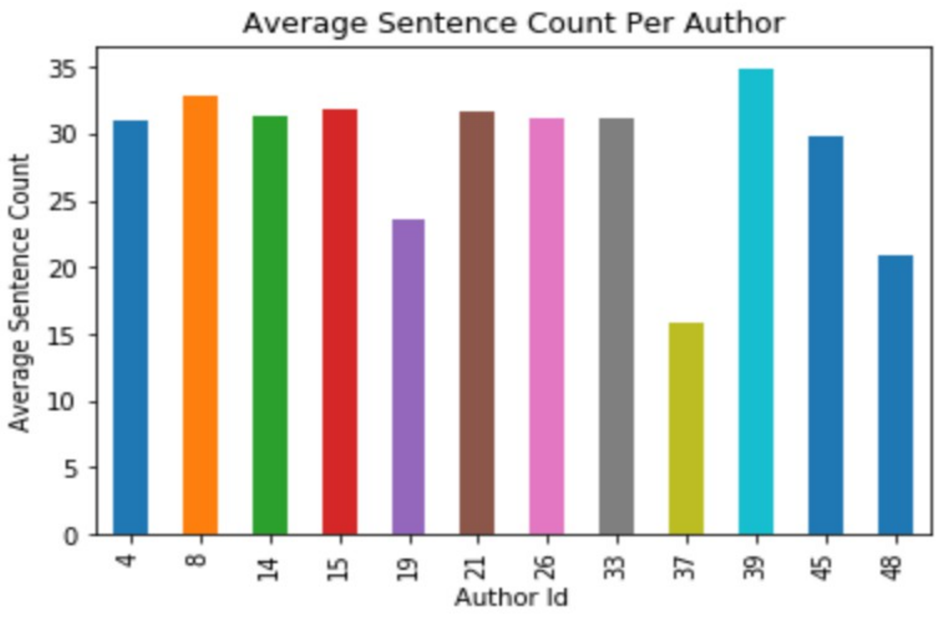
Data Preprocessing:

All records were tagged with parts of speech and the counts extracted into features. The sentence count was also extracted into a column.

The feature set was as follows;

```
'length', - Length of the text
'sentence_counter', - Number of Sentences in the Text
'ADJ', - Number of Adjectives in the text
'ADP', - Number of adpositions
'ADV', - Number of Adverbs
'CCONJ', - Number of Co-ordinating Conjunction (and, or, but)
'INTJ', - Number of Inerjections ( a part of speech that shows the emotion or feeling of the author )
'NOUN', - Number of Nouns
'NUM',  - Number of Numbers
'PART', - Number of Particles ('s, not)
'PRON', - Number of pronouns
'PROPN', - Number of propositions
'PUNCT', - number of Punctuations
'SYM', - Number of Symbols
'VERB', - Number of Verbs
```

Exploratory Visualisation

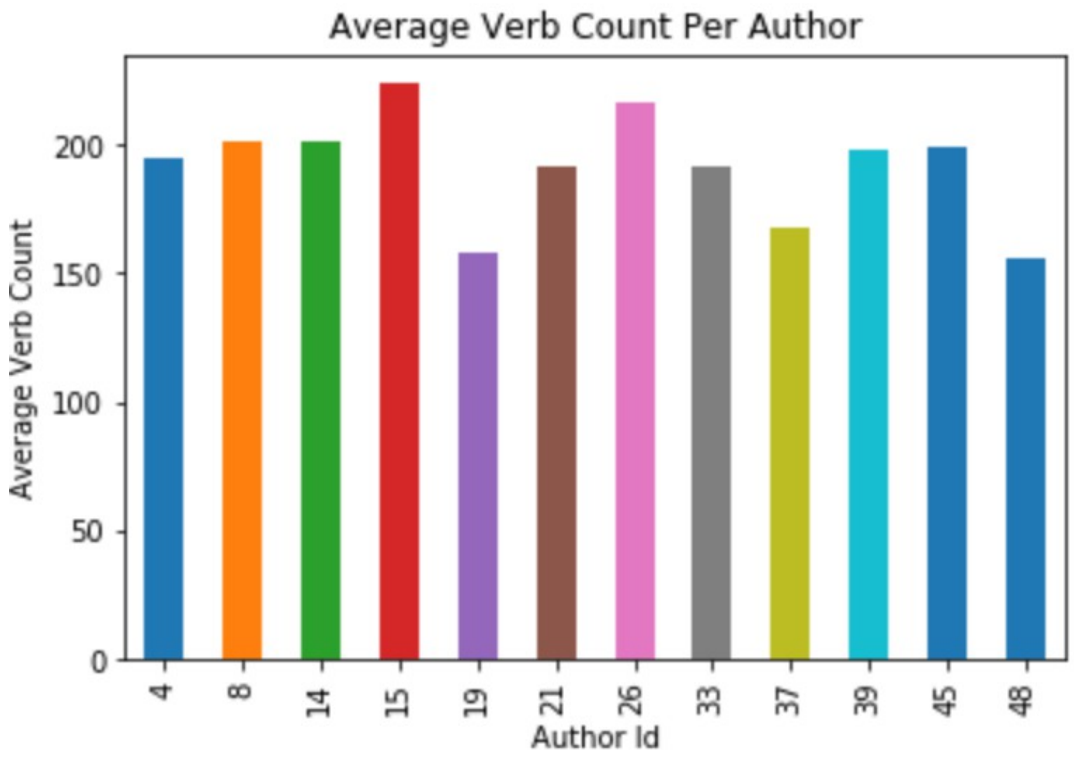


The chart above shows the average sentence count per author. The sentence count does not vary considerably between authors, the majority seem to have between approximately 25-32 sentences. I found that most of the part of speech tag counts did not vary considerably between authors.

Algorithms:

The algorithms used were:
1. XGBoost Classifier
2. Random Forest Classifier
3. Logistic Regression

The Part of Speech features gave slightly better results in that the negative log loss for xgboost improved too -1.3. From plotting the various features it did not appear that the features varied greatly between the authors. Using the feature importances property of Random Forest the 5 most important features were identified as Length of text, adjectives, adpositions , Adverbs, Verbs.



Above is shown the average verb count per author. I have selected this visualisation because verb count was identified as one of the most predictive features by the xgboost algorithm. Why the average verb count seems to vary more between authors than the sentence count most authors seem to have an average verb count in range of between 180 and 200

2. **Deep Learning Approaches to Text Classification**

   The deep learning implementation is found in Notebook : [Victorian Authorship Attribution Deep Learning Approaches.iynb]

One of the key disadvantages of the preprocessing carried out for the machine learning models is that the document is represented as a collection of words either through word counts (bag of words model) or term inverse document frequency which identifies the importances of terms but neither approach preserve the order of words in the document nor their meaning.

Text Pre-Processing

Text was pre processed as follows:

1. converted to lower case
2. Lemmatised (to reduce each word to its root) For example (source Wikipedia), in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk', that one might look up in a dictionary.
3. Stop words were removed
4. Pronouns were removed
5. Punctuation was removed
6. One letter words were removed as they appeared frequently in the vocabulary as a result of removing punctuation.

Steps 3 to 6 were carried out to remove noise from the text. One key aspect of this preprocessing approach is that the order of words is preserved and fed into the models.

The models can only receive numeric input so the top 500 words were inserted into a dictionary and the words replaced by their numeric index. In this way the sequence of words is preserved. The models can only take a fixed width input so sequences were padded to a length of 1000.

I selected a length of 1000 so that I was passing a chunk of text of a reasonable size to the model to see if it could learn patterns.

Word Embeddings

The word vectors obtained by the preprocessing methods described above are very sparse (mostly 0s) and do not include any information about semantic relationships between words that could be useful.
eg.

Another advantage of using word embeddings is the ability to use pre-trained word embeddings essentially a form of transfer learning. There are two main pre-trained word embeddings available, word2vec and Glove. Glove vectors have been used for this problem because they are smaller

Prior to being fed into the models the text was further pre-processed as follows:

The glove vectors were loaded and each of the top 500 words from the text were transformed into a vector using the pre-learned weights from the glove vectors. Any words not found in the glove dataset had a vector with all weights set to 0.

This saves having to learn the weights as part of the training process for the model the pre-trained embedding weights get fed in as the first layer.

Algorithms Used And Reasons for selection

Convolutional Neural Network

Convolutional Neural networks are used in computer vision to extract patterns from local areas of images. They have a similar application in natural language processing.

Input Layer

This Is An Example sentence
This Is An Example sentence
This Is An Example Sentence

Convolutional Filters
Each Filter will pass from the left of the sentence to the right. Each shift will be 1 token.

Convolutional Windows:

A convolution window of size 3 should be able to learn groups of words of 3 or less. This way the network is able to learn ngrams. For this reason, I experimented with a number of different window sizes (results will be reported later).

At each step the weight in the filter is multiplied by the weights in each of the 3 words. The answers are summed up and passed to the next layer.

RelU activation has been used initially so weights are only passed if they sum to a positive number otherwise they are passed as 0.

Down Sampling with a Pooling Layer
For the output of each filter the maximum or average value is computed.

The initial architecture I tested was as follows:

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)       (None, 1000, 100)         852500
_________________________________________________________________
conv1d (Conv1D)             (None, 997, 32)           12832
_________________________________________________________________
dropout (Dropout)           (None, 997, 32)           0
_________________________________________________________________
max_pooling1d (MaxPooling   (None, 498, 32)           0
_________________________________________________________________
conv1d (Conv1D)             (None, 495, 16)           2064
_________________________________________________________________
max_pooling1d (MaxPooling   (None, 247, 16)           0
_________________________________________________________________
conv1d_ (Conv1D)            (None, 244, 8)            520
_________________________________________________________________
```

```
max_pooling1d_28 (MaxPooling (None, 122, 8)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 976)             0
_________________________________________________________________
dense_17 (Dense)             (None, 24)             23448
_________________________________________________________________
dense_18 (Dense)             (None, 12)              300
=================================================================
Total params: 891,664
Trainable params: 39,164
Non-trainable params: 852,500
_________________________________________________________________
```

I selected a window size of 4 and 32 feature maps.

My initial results were 65% accuracy and log loss of 0.91 on the validation set after 14 epochs.

I explored a number of combinations of feature map numbers and window sizes which I will discuss in the refinement section.

Summary Of Initial Results Prior to Optimisation

Below is a summary of the results achieved from the approaches discussed above. While each approach discussed so far is too some degree a refinement of the previous approach I performed further optimisation on the best performing model the results will be discussed in the next section.

Algorithm	Feature Set	Result
XGBoostClassifier	TFIDF Vectorized Text	-2.25
XGBoostClassifier	Part of Speech Features	-1.2
Convolutional Neural Network (Convolution Size 4, 32 Feature Maps)	Word to Sequence Vectorized Text using pre-learned word embeddings from glove	-0.91

Refinement

Machine Learning Approaches:

1. Hyper Parameter Turning

There were several aspects to the machine learning models that could be tuned
1. The text preprocessing – text could be preprocessed using term inverse frequency matrix or a count of individual words or a count of bi-grams and tri-grams
2. The part of speech feature set
3. The Hyper parameters of the models.

Due to the large number of combinations of parameters that could be tuned I decided that a grid search was computationally too expensive, so a random search was used instead.

Of the Text Preprocessing Methods the following combinations were listed to be selected by the random search

1. unigrams (single word counts)
2. bi grams
3. tri grams
4. Use TFIDF or not

XGBoost gave the best initial results so XGBoost was tuned as part of the refinement process. There are a considerable number of parameters that could be tuned in XGBoost I focused on the following

1. Max Depth of the decision trees
2. Number of decision trees
3. Regularisation – to prevent overfitting

The regularisation parameter varied the most between random searches.

Part of Speech Features (numeric feature set)

After performing a random search on hyperparamers for xgboost classifier I used the most important features setting on xgboost to see which features the model found to be most important. The plot has already been displayed previously.

The most important features were

1. length of text (I later removed this feature as it could be influenced by the data collection process)
2. Count of Adjectives
3. Count of Adpositions
4. Count of Adverbs
5. Count of Verbs

Running a classifier using only these features produced a negative log loss of -1.64

After checking the the correlation matrix, to see if any of the features were closely related to each other (a high value for correlation suggests that the two features vary together it turned out that pronouns and verbs were highly correlated, so I removed them. I also removed the length of text as a feature as I suggested above it could be influenced by the data collection process.

Correlation Coefficients for the two most correlated features:

```
PRON              VERB     0.734443
sentence_counter  INTJ     0.634832
```

Running xgboost on the following feature set:

'ADV', - Count of Adverbs
 'ADP', - Count of Adpositions
 'ADJ', - Count of Adjectives
'VERB', - Count of Verbs

'NOUN' – Count of Nouns

The log loss did not improve, the result was: -1.68

Deep Learning Approaches:

I focused on tuning the convolutional neural network especially the window size and the number of filters. I experimented with a network of less depth especially as the data set was small however the original depth gave a lower loss so I continued to focus on tuning the window size.

Extensions for tuning

1. Batch Size
2. Convolution Window Size (this would impact the learning of ngrams)
3. Number of Feature Maps

Convolutional Neural Network

Results of Tuning

Algorithm	Feature Set	Result
XGBoostClassifier Max Depth of Trees = 5 Number of Trees = 200 Regularisation = 0.01	Text features processed with Tri Grams Max Features 300 Max_df = 0.5 (50%)	-0.5
XGBoostClassifier Max Depth of Trees = Number of Trees = Regularisation =	Part of Speech Features 5 most helpful features (as selected by XGBoost) only	-1.63
XGBoostClassifier Max Depth of Trees = Number of Trees = Regularisation =	Part of Speech Features 5 most helpful features after the most highly correlated features removed	-1.68
Convolutional Neural Network (Convolution Size 10, 32 Feature Maps)	Word to Sequence Vectorized Text using pre-learned word embeddings from glove	-1.2
Convolutional Neural Network (Convolution Size 5, 32 Feature Maps)	Word to Sequence Vectorized Text using pre-learned word embeddings from glove	-1.15
Convolutional Neural Network (Convolution Size 3, 32 Feature Maps)	Word to Sequence Vectorized Text using pre-learned word embeddings from glove	-1.14
Convolutional Neural Network (Convolution Size 4, 64 Feature Maps)	Word to Sequence Vectorized Text using pre-learned word embeddings from glove	-1.2

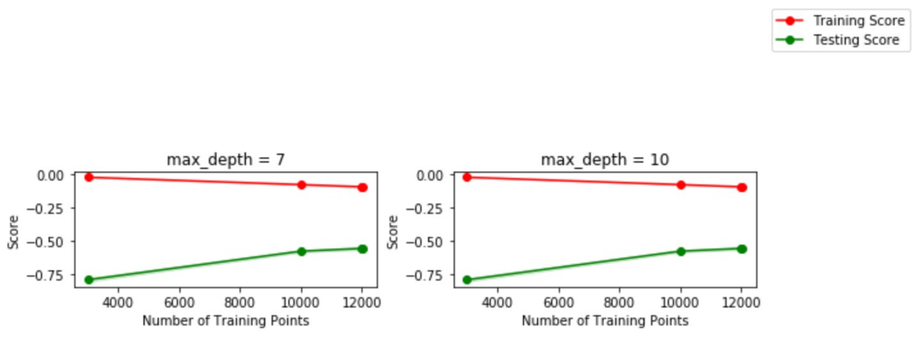
Model Evaluation

The best performing model in terms of minimising log loss was xgboost. The final model achieved 85% accuracy on the test set or a log loss of -0.5. This was a significant improvement on the baseline log loss of -2.25.

XGBoostClassifier Max Depth of Trees = 5 Number of Trees = 200 Regularisation = 0.01	Text features processed with Tri Grams Max Features 300 Max_df = 0.5 (50%)	Predictions on test set : Accuracy 85% Log Loss -0.5
Baseline Multinominal Naive Bayes	TFIDF Vectorized Text	Log Loss -2.26

Training Data:
This is a small subset of the full data set. Plotting the learning curve (below) confirms – The training score is consistently higher than the testing score and it shows no sign of converging which means that more training data is needed. The testing score is starting to fall at around 12000 training points, however the high score on the training data suggests that the model is over fitting on the training data, in order to reduce this more training data is needed or I need to look at ways to simplify the model. I think the first thing I would try to do to reduce this over fitting is reduce the number of decision trees (this will be discussed below). The validation score is starting to flatten out to -0.5

Iearning Curve for XGBoost Classifier – Number of trees 200, learning rate 0.3

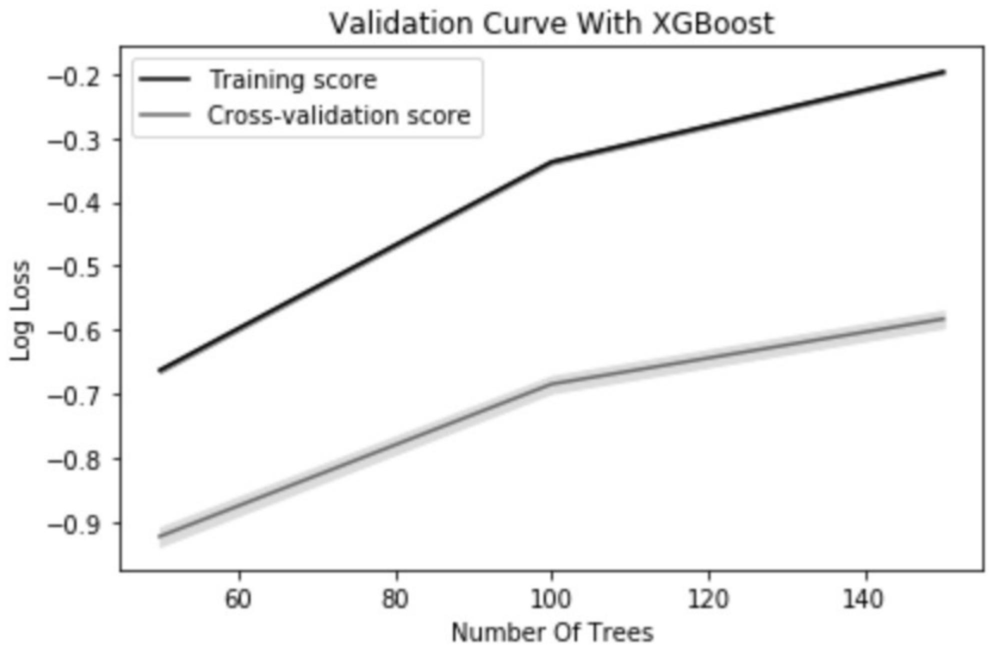


Model Sensitivity – The final model achieved consistent results when trained on differing training data
The log loss varied between (-0.49 and -0.52) meaning that the model is not sensitive to small changes in the training data.

Model Hyperparameters:

The XGBoost Algorithm consists of a number of decision trees. I have focused on tuning the decision tree related parameters. Further tuning could be done to other parameters.

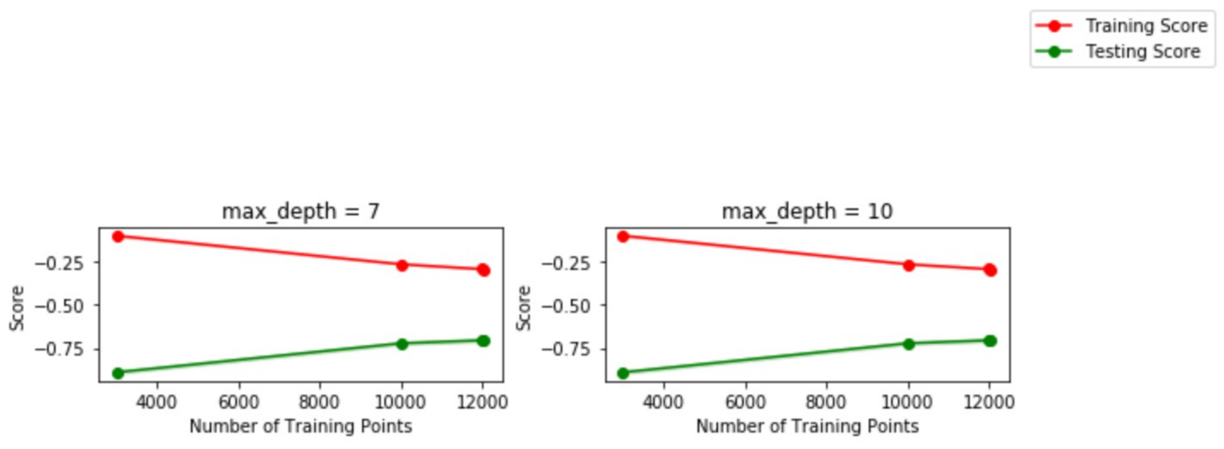
Number of Trees – The number of decision trees in the ensemble. Increasing the number of decision trees can increase over fitting as the model learns the dataset more. Observe the validation scores shown in the illustration below which suggests the score for the training set gets smaller as the number of trees increases (overfitting), the validation score is starting to plateau towards 0.5. I think if I had more time the first thing I would do is try a model with fewer decision trees to see if I can reduce the training score.



Depth of Trees - The maximum depth of a decision tree. Deeper trees capture too many details and can potentially over fit the training data. Shallow trees capture too few details of the data.

Learning Rate = 0.3 – The learning rate is the xgboost default of 0.3

Further tuning of hyper-parameters based on the validation curves such as the one above started to reduce overfitting – The hyper parameters used were
number of trees = 100,
max_depth of trees =5,
learning_rate=0.1



Learning Curve For XGBoost (Number of trees 100, learning rate 0.1)

Based on the two plots above the training score is falling (compared to the previous learning curves above) but still more training samples are needed. The log loss was -0.67.

Justification

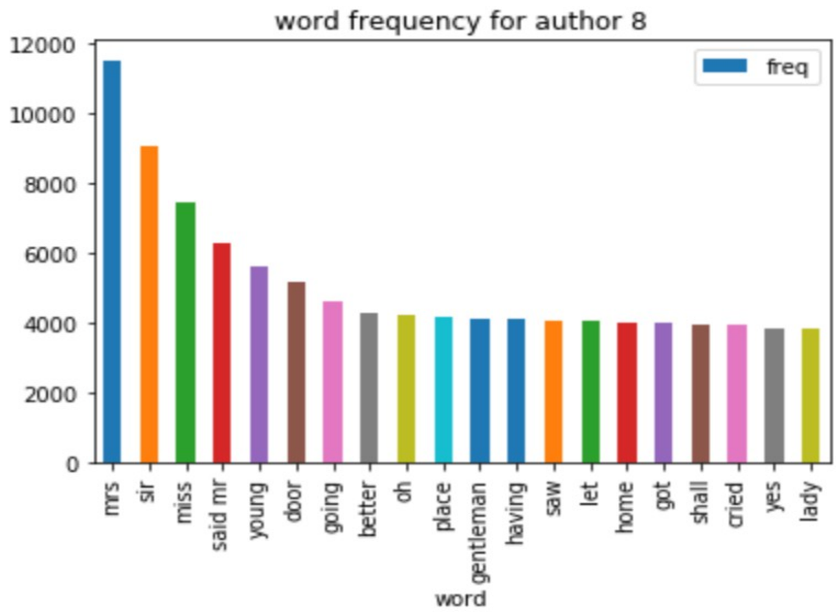
The classifier achieved an 85% accuracy or log loss of -0.5 which exceeded my expectations given the baseline was -2.25. However I think I need to train the model on the entire dataset in order for this model to generalise better.

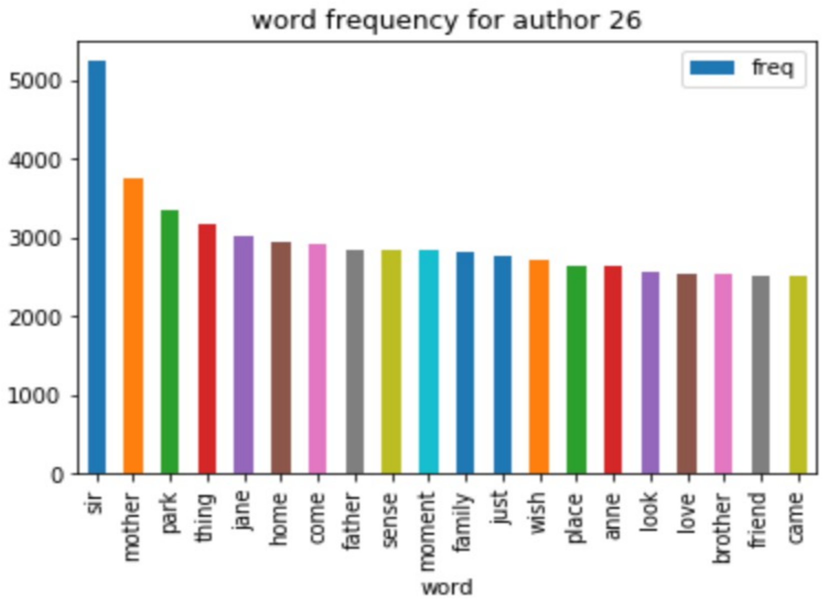
The classifier that was built does solve the problem in that it succeeded in predicting the correct author for 85% of the test set. However further tuning of the model is needed to improve its ability to generalise.

Conclusion

Below I have displayed the frequency of the top 20 words for several of the authors. I have selected this visualisation because I was surprised that the bag of words model proved so effective in being able to predict the authorship, but looking at the plots below the words and frequencies do vary between authorship much more than the part of speech features. The term inverse frequency feature set was quite different again, focusing mostly on characters – which may not be transferable to other works by the same author.

Bag of Words Features





Term Inverse Document Frequency Features:

```
Feature Names Generated from Inverse Term Frequency model
['amateur night', 'aunt mary', 'bending bough', 'benefit doubt', 'big fat',
'black jack', 'blind girl', 'break day', 'brother francis', 'brother ned',
'chief butler', 'class struggle', 'colonel wife', 'drum horse', 'drums
fore', 'edition cr', 'ev ev', 'faith men', 'fat boy', 'father peter',
'father tom', 'ford ford', 'ford river', 'foreign office', 'ft ft', 'general
election', 'good jack', 'hired baby', 'house pride', 'ji game', 'labor
army', 'law development', 'legal member', 'lit lit', 'little britain',
'little foot', 'local color', 'lone chief', 'lord richard', 'loved king',
'madness john', 'majesty king', 'majesty servants', 'man animals', 'man
wooden', 'mary john', 'matter private', 'meeting streams', 'miss abbey',
'miss pinch']
```

Reflection

In summary the process followed was as follows

Text Preprocessing – Text was converted to numeric form in order to be processed by the models. A number of text pre-processing methods were explored including part of speech feature generation, bag of words generation, term inverse document frequency.

A number of machine learning models were applied to the various text feature sets.

XGBoost was the best performing classifier, hyper-parameters relating to the depth and number of decision trees were tuned along with the learning rate.

Hyper-parameters relating to the generation of text features were tuned.

The best performing model was evaluated along with the choices of hyper-parameters and points for improvement were identified.

Convolutional Neural Network approaches to text classification were explored – particular attention was paid to the convolution window size (as this is similar to the ngrams used in the machine learning approaches). The convolutional Neural network architectures did not give as high log loss as xgboost so I focused on tuning xgboost.

I was surprised that an xgboost classifier with a bag of words was able to outperform the convolutional neural network. If I had more time there were a number of other things I wanted to try for example combining the numeric feature set and the text feature set however given how much time it takes to compute the feature set of part of speech counts did not seem to improve the model very much.

The xgboost model I have requires some further tuning as I have discussed above – for example reducing the number of decision trees to reduce overfitting. I could also have spent time tuning the reg alpha parameter

The convolutional neural networks had many more parameters I could have spent more time tuning for example testing different embeddings dimensions, testing different sequence lengths (I used 1000) but could have sent the entire encoded piece of text for example to discover what impact this had on the model. The convolutional neural networks were not predictive on the test set, I did not have time to investigate why as I focused on the best performing model which was the xgboost classifier.

The final model still needs further tuning and needs to be trained on a larger dataset in order to get it to generalise more effectively. However this model did achieve around 85% accuracy and it seems that the log loss is converging at around -0.5 – I would need to add more training samples in order to prove this empirically.

Improvement

There are many more approaches I could have explored. For example trying to combine the most helpful part of speech features with the bag of words model. I would have liked to have spent more time investigating how to improve the results for the convolutional neural networks, but essentially xgboost got the best result so I focused on tuning this model.

References:

Hobson Lane, Cole Howard; Hannes Max Hapke. Natural Language Processing in Action: Understanding, analyzing, and generating text with Python MEAP V08 Manning Publications Co..

Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems . O'Reilly Media.

François Chollet. Deep Learning with Python. Manning Publications.

Deep Learning for Natural Language Processing Jason Brownlee.