Authorship Attribution on Classical Works

Robert Chatterton
Khoury College of Computer Science, Northeastern University
DS4400 Machine Learning and Data Mining I, *Spring 2022*
chatterton.r@northeastern.edu

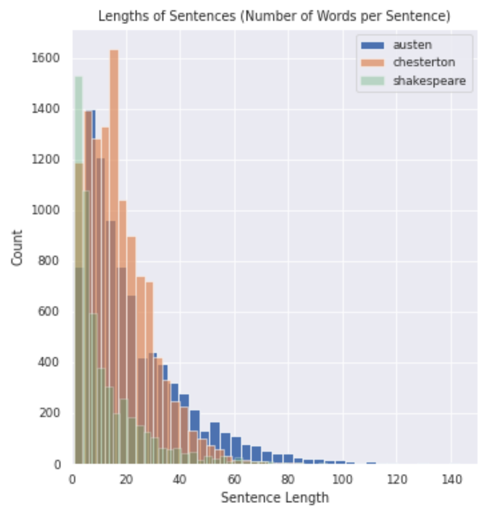
Project Code Google Colab
Project Presentation Link
TA: Noah

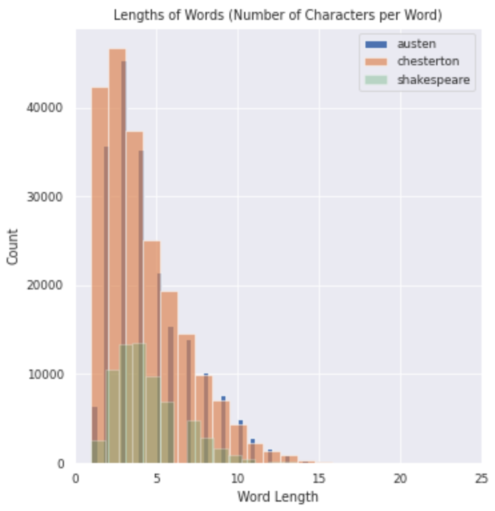
I. Introduction and Problem Description

Authorship attribution, put plainly, is to use factors about an author's writing style to identify whether or not a document was written by them. The forms of an author's writing style can take several forms: whether it is sentence structure, vocabulary, sentiment and tone, or punctuation usage. As far as real-world applications in today's age of social media, authorship attribution is a useful tool for spam/bot detection, forensic linguistics, and other areas of criminology. For my final project, I wanted to use several Machine Learning and Deep Learning algorithms to classify the author of a given sentence based on some of these attributes. The algorithms I modeled for this task were a Support Vector Classification (SVC) model, a Random Forest Ensemble, a Naive Bayes Classifier, and a Convolutional Neural Network (CNN).

II. Data Analysis

The data for my project was all taken from the Project Gutenberg corpus. The works I used were as follows: *Persuasion* and *Sense and Sensibility* by Jane Austen, *The Innocence of Father Brown*, *The Ball and the Cross*, and *The Man Who Was Thursday: A Nightmare* by G. K. Chesterton, and *Julius Caesar*, *Hamlet*, and *Macbeth* by William Shakespeare. While I limited the scope of my project to just these works and authors, the Gutenberg corpus is immense and continually growing, so it could be expanded upon to encompass more. Using the Natural Language Toolkit (NLTK) to first load each work, I then tokenized each work into a list of sentences, cleaned up some of the characters, and added it to a list of sentences, split by author. The final dataset before any real manipulation occurred was this list, made up of a randomly chosen 5,000 sentences from each author. I wanted to ensure that the dataset was both randomly chosen from the available data, and that every author got the same amount of sentences in the dataset.





There were two formats of the data that I had - one format was through the creation of 3-grams, and the other was a TF-IDF vectorized version of each sentence. For context, an N-gram is a sequence of N words taken from the corpus. TF-IDF vectorization is the multiplication of Term Frequency (TF) and Inverse Document Frequency (IDF) for a "document," a sentence in the case of this project. TF is simply the frequency of a word in a document D, while IDF is the measure of the "importance" of a word w, calculated by the function below:

$$\text{IDF}(w, D) = ln(\frac{\text{\# of documents in corpus } D}{\text{\# of documents containing } w})$$

The Naive Bayes model and CNN implemented both use N-grams as input, while the SVC and Random Forest Ensemble use the TF-IDF vectorized sentences as input. The TF-IDF vectorization inputs were first lemmatized in order to lower the total vocabulary of the corpus, using the WordNet Lemmatizer from NLTK. All training and testing data is randomly chosen with a split of 75% training to 25% testing.

III. Approach and Methodology

For this task, I employed the use of 4 different machine learning models, which ranged in

Support-Vector Classifier

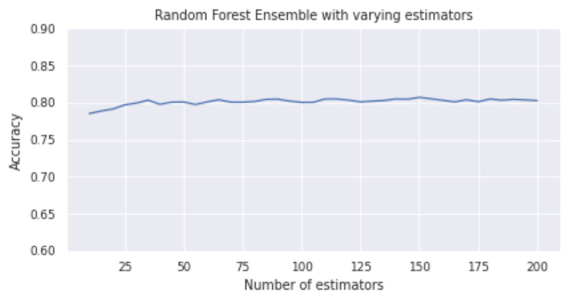
A Support-Vector Classifier (SVC) is a classification model whose whole goal is to (through training) create hyperplane decision boundaries, in order to classify a new data point. In the case of this project, it receives a TF-IDF vector and uses those values as a "point" to be classified.

When training this model, in order to select the best parameters, I used Scikit-Learn's GridSearchCV class to test a number of different parameters, like changing the regularization constant and kernel type. After reviewing the accuracies of the different parameters, the model with the best accuracy had a regularization constant of 1 and a "linear" kernel type. This model had an accuracy of about 0.80.

The final model used in the evaluations for this project was implemented through the Scikit-Learn Python module.

Random Forest Ensemble

A Random Forest ensemble is an ensemble learning model which uses a large number of randomized Decision Trees to make a classification. One of the largest problems with Decision Trees is their tendency to overfit when training, which the random nature of the Random Forest ensemble seeks to correct. Training and classifying with a number of decision trees in parallel (a process known as *bagging*), the results from each individual tree are taken into account and the majority's classification is what gets submitted as the output. This model, like the SVC model, receives a TF-IDF vector as input for the decision tree, each item in the vector a different feature.



When training the Random Forest ensemble for this project, in an attempt to get the best parameters for my model, I tested ensembles with between 10 and 200 estimators. I felt like this would have made more of a impact than it had, the most recent run had the model with 150 estimators have the best accuracy, but running these tests previously would choose a model between 110 and 150 estimators. In the previous graph, it is clear just how little of an effect changing the number of estimators makes.

The final model used in the evaluations for this project was implemented through the Scikit-Learn Python module.

Naive Bayes Classifier

A Naive Bayes classifier is a classifier that makes use of Bayes' theorem to make its classification. Essentially, this classifier trains by learning the prior probabilities for each class $k \in K$ ($P(K = k)$) and learning the conditional probability for each feature X for each class k ($P(K = k|X = x)$). While a true Bayesian network would compute the probability of a given sample for each class k like so:

$$P(K = k|x_1, x_2, ..., x_n) =$$
$$P(x_1|x_2, ..., x_n, K = k) \cdot P(x_2, ..., x_n, K = k)$$

With the usage of the chain rule, a Naive Bayesian network treats each feature X as mutually independent, therefore avoiding the chain rule and turning the calculation into:

$$P(K = k|x_1, x_2, ..., x_n) = P(k) \prod_{i=1}^{n} P(x_i|K = k)$$

This makes the calculation much, much simpler and much less laborious. For training the Naive Bayes model, I used Scikit-Learn's CategoricalNB and fit it to the three gram training set.

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a feed-forward neural network that involves the usage of convolutional and pooling filter layers in an attempt to reduce the total number of features, preserving and exacerbating the ones of high importance (in other words, learn hierarchical feature representations). While usually CNNs are best known for their usages in image analysis, they have usages in the field of natural language processing as well.

For this project, I used a CNN designed specifically for natural language processing. Built from the Keras Python module, it was as straightforward as it could be to understand what each layer and node of the neural network was doing and what was going on in the whole network itself.

Neural Network Visualizations

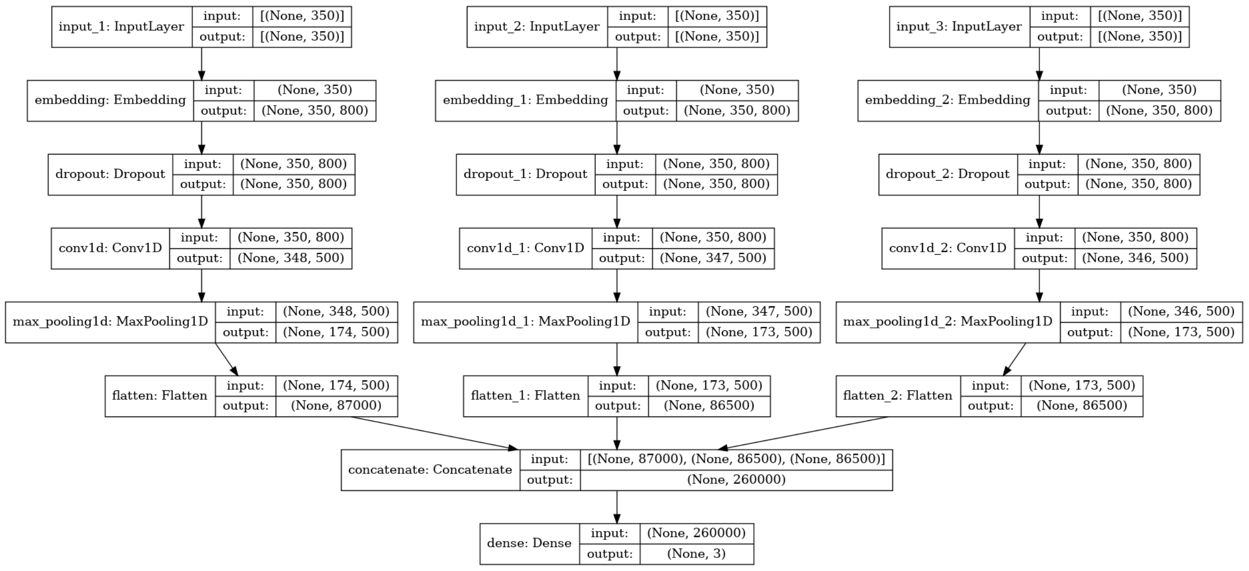

input_1: InputLayer | input: [(None, 350)] | output: [(None, 350)]
input_2: InputLayer | input: [(None, 350)] | output: [(None, 350)]
input_3: InputLayer | input: [(None, 350)] | output: [(None, 350)]
embedding: Embedding | input: (None, 350) | output: (None, 350, 800)
embedding_1: Embedding | input: (None, 350) | output: (None, 350, 800)
embedding_2: Embedding | input: (None, 350) | output: (None, 350, 800)
dropout: Dropout | input: (None, 350, 800) | output: (None, 350, 800)
dropout_1: Dropout | input: (None, 350, 800) | output: (None, 350, 800)
dropout_2: Dropout | input: (None, 350, 800) | output: (None, 350, 800)
conv1d: Conv1D | input: (None, 350, 800) | output: (None, 348, 500)
conv1d_1: Conv1D | input: (None, 350, 800) | output: (None, 347, 500)
conv1d_2: Conv1D | input: (None, 350, 800) | output: (None, 346, 500)
max_pooling1d: MaxPooling1D | input: (None, 348, 500) | output: (None, 174, 500)
max_pooling1d_1: MaxPooling1D | input: (None, 347, 500) | output: (None, 173, 500)
max_pooling1d_2: MaxPooling1D | input: (None, 346, 500) | output: (None, 173, 500)
flatten: Flatten | input: (None, 174, 500) | output: (None, 87000)
flatten_1: Flatten | input: (None, 173, 500) | output: (None, 86500)
flatten_2: Flatten | input: (None, 173, 500) | output: (None, 86500)
concatenate: Concatenate | input: [(None, 87000), (None, 86500), (None, 86500)] | output: (None, 260000)
dense: Dense | input: (None, 260000) | output: (None, 3)

Fig. 1. Architecture of the CNN

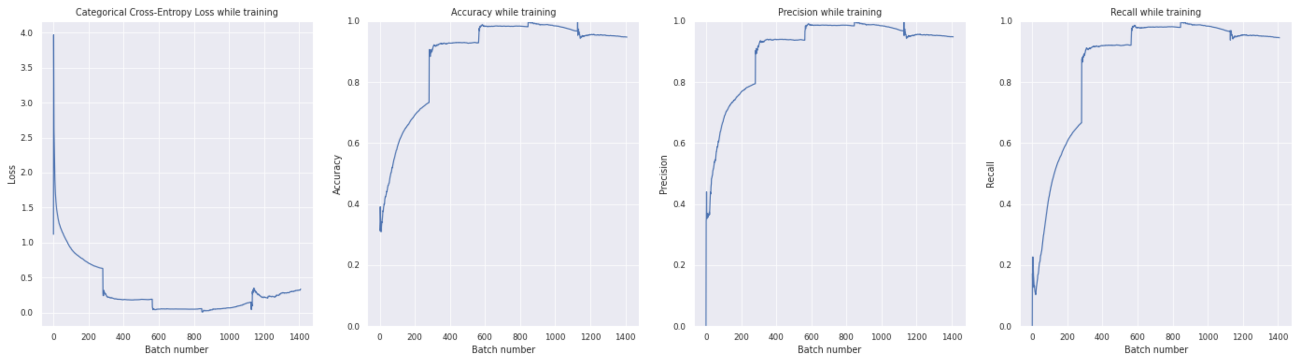
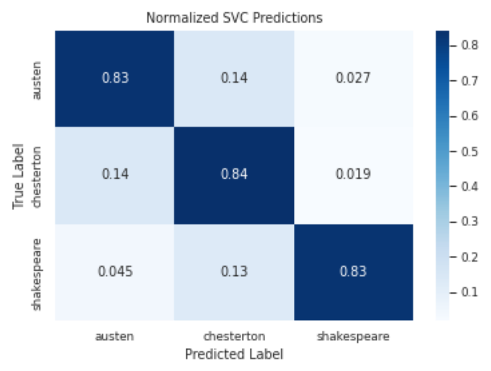


Fig. 2. Training information from each batch. Each spike that can be made out is a new epoch, happening about every 260 epochs.

IV. Performance and Metrics

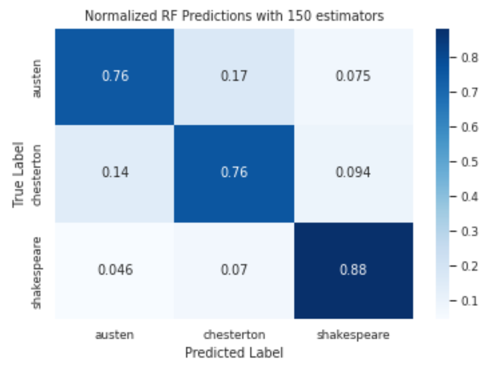
Performance Metrics for SVC Model

Author	*Precision*	*Recall*	*F1-Score*
Austen	0.82	0.83	0.83
Chesterton	0.75	0.84	0.79
Shakespeare	0.95	0.83	0.88
Overall Accuracy			**0.83**



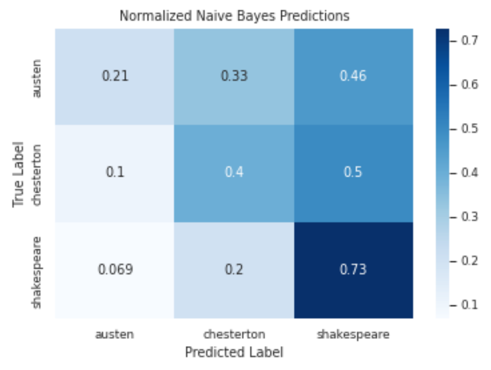
Performance Metrics for Random Forest

Author	*Precision*	*Recall*	*F1-Score*
Austen	0.80	0.76	0.78
Chesterton	0.75	0.76	0.76
Shakespeare	0.84	0.88	0.86
Overall Accuracy			**0.80**



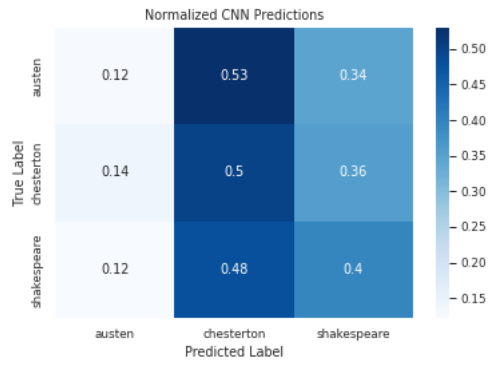
Performance Metrics for Naive Bayes

Author	*Precision*	*Recall*	*F1-Score*
Austen	0.55	0.21	0.30
Chesterton	0.41	0.40	0.41
Shakespeare	0.44	0.73	0.55
Overall Accuracy			**0.45**



Performance Metrics for CNN Model

Author	*Precision*	*Recall*	*F1-Score*
Austen	0.33	0.12	0.18
Chesterton	0.32	0.50	0.39
Shakespeare	0.37	0.40	0.38
Overall Accuracy			**0.34**



All metrics on the previous page were performed using the same set of training data and the same set of testing data. Although the SVC model and Random Forest ensemble used the TF-IDF vectorization version of the data, and the Naive Bayes classifier and CNN model used the 3-grams, it was the same information in both sets.

V. Evaluation and Discussion

Comparing all of the models with each other, I would argue that the SVC was best, followed closely by the Random Forest ensemble, then the Naive Bayes, and lastly, the CNN.

I think that it is no coincidence that the models using the TF-IDF vectors outperformed the N-gram models, as the N-grams just do not contain as much information as the vectors do. That being said, I was really impressed with the speed and performance of the SVC and Random Forest. Training 20 different SVCs and Random Forest ensembles in a row took a remarkably short amount of time - about a minute on my computer. I would love to see the decision boundary created by the SVC to interpret it myself somehow, but I know that it is not feasible to plot out a graph with that many dimensions.

I was amazed during Homework 4 about how well the Random Forest and AdaBoost ensembles worked, so going into this project I was expecting the Random Forest to do the best out of all the models. The only reason I think that the SVC outperformed the Random Forest for this project is because the SVC was much more consistent across the board, identifying all 3 authors at basically the same accuracy. The Random Forest was better than the SVC at identifying Shakespeare, but worse at identifying both Austen and Chesterton. Because Shakespeare definitely has a more unique writing style than the other authors in vocabulary and sentence length, I think that it makes some intuitive sense to see that result.

As for the models that did not perform very well, we will first discuss the Naive Bayes implementation. I think that this model struggled a lot because of the fact that it took in the N-grams as opposed to the TF-IDF vectors that the first two models had. Due to time constraints with the end of the semester coming around for all of my classes in addition to this one, I was not able to implement this on my own for the purposes of this project. However, I think of all the models I utilized for this project, a Naive Bayes model would have been the most achievable. Because I was not able to implement it myself, I was left with a sort of ”black box” algorithm, and so I can only make guesses as to what went so poorly with this one. However, looking at the confusion matrix, it seemed to favor Shakespeare over Chesterton or Austen. I think this may have been because Shakespeare is much more likely to use ”strange” words, while the other two authors are much closer in their vocabularies.

Lastly, the CNN. While I had high hopes for this model, I was really struggling in its implementation and ended up following a bit of a tutorial on setting one up for natural language processing purposes, but since it was considerably more complex and complicated than the Naive Bayes model, I was still at a bit of a loss as to how I could improve the CNN or really where to even start.

Be that as it may, I really did have an awesome time working on this project over the last month or so, and I learned a lot about all kinds of things in the natural language processing space and the machine learning space. Authorship attribution is a challenging task, but the skills that I applied here will be with me for my next project. In the future, I can see myself tackling this project again, with a wider spread of works and more knowledge on ways to accomplish my goal.

Appendices

CNN Structure (Keras Summary)

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 350)]        0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 350)]        0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 350)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 350, 800)     20186400    input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 350, 800)     20186400    input_2[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 350, 800)     20186400    input_3[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 350, 800)     0           embedding[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 350, 800)     0           embedding_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 350, 800)     0           embedding_2[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 348, 500)     1200500     dropout[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 347, 500)     1600500     dropout_1[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 346, 500)     2000500     dropout_2[0][0]
__________________________________________________________________________________________________
max_pooling1d (MaxPooling1D)    (None, 174, 500)     0           conv1d[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 173, 500)     0           conv1d_1[0][0]
__________________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 173, 500)     0           conv1d_2[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 87000)        0           max_pooling1d[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 86500)        0           max_pooling1d_1[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 86500)        0           max_pooling1d_2[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 260000)       0           flatten[0][0]
                                                                 flatten_1[0][0]
                                                                 flatten_2[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 3)            780003      concatenate[0][0]
==================================================================================================
Total params: 66,140,703
Trainable params: 66,140,703
Non-trainable params: 0
```

References

[1] Bozkurt, Ilker Baghoglu, O. Uyar, Erkan. (2007). Authorship attribution. Foundations and Trends in Information Retrieval - FTIR. 1. 1 - 5. 10.1109/ISCIS.2007.4456854. Retrieved May 2, 2022.

[2] Project Gutenberg. (n.d.). Retrieved May 2, 2022, from www.gutenberg.org.

[3] Brownlee, J. (2020, September 2). How to develop a multichannel CNN model for text classification. Machine Learning Mastery. Retrieved May 2, 2022, from https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/

[4] Bird, S., Klein, E., Loper, E. (2009). Natural language processing with Python: analyzing text with the natural language toolkit. ”Reilly Media, Inc.”

[5] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.